

# Apprenez JAVA 1.1 en 21 jours

**LAURA LEMAY ET CHARLES L. PERKINS**

- Toutes les techniques de la programmation objet avec Java
- Conception Web : interactivité et animation grâce aux applets
- Toutes les nouvelles fonctions Java : JDBC, RMI, JavaBeans, etc.



**EN CADEAU !**

Un CD-ROM  
contenant Java  
Development Kit  
pour Mac et PC  
et tous les codes  
source du livre

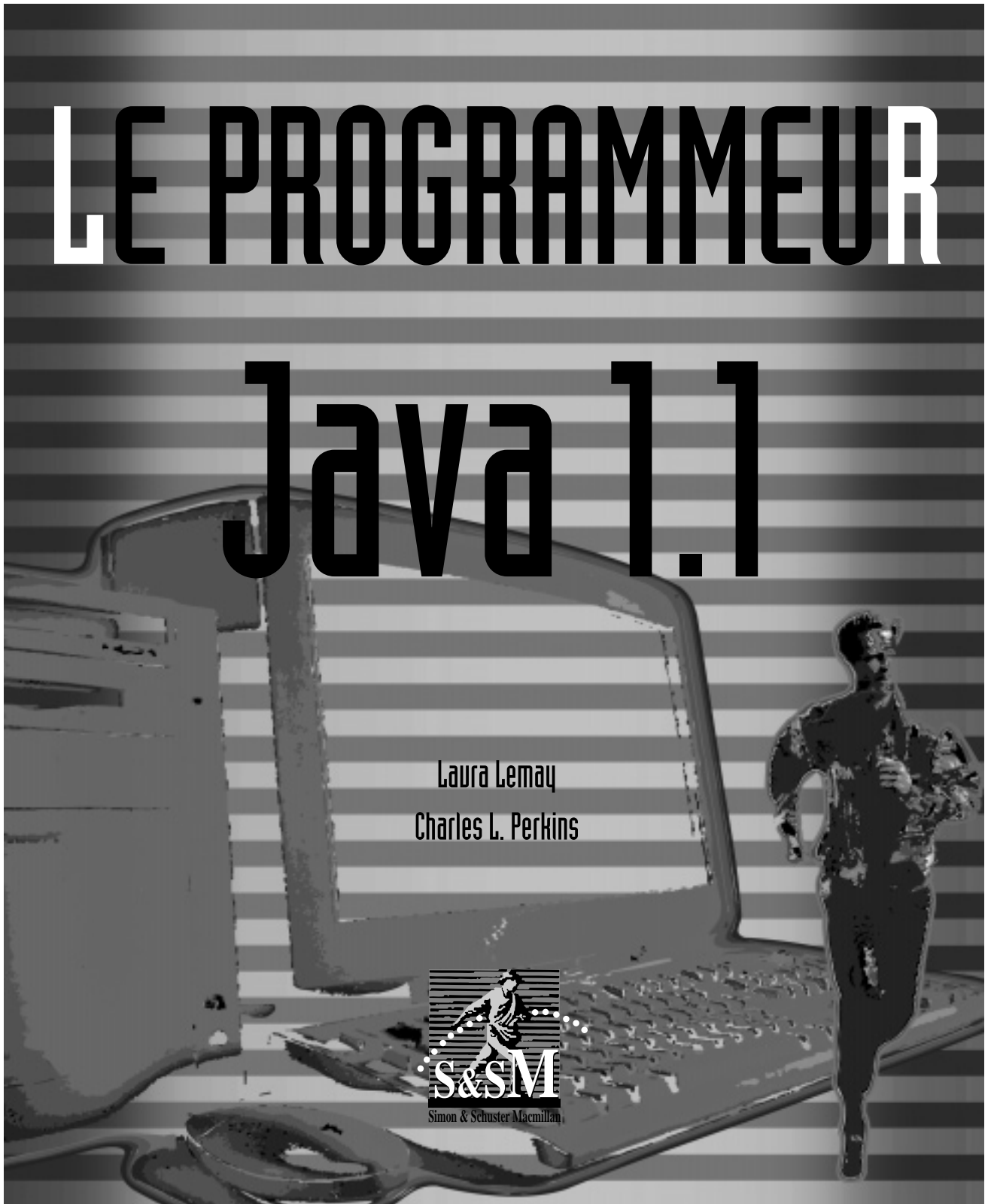


# LE PROGRAMMEUR

# Java 1.1

Laura Lemay

Charles L. Perkins



Simon & Schuster Macmillan (France) a apporté le plus grand soin à la réalisation de ce livre afin de vous fournir une information complète et fiable. Cependant, Simon & Schuster Macmillan (France) n'assume de responsabilités, ni pour son utilisation, ni pour les contrefaçons de brevets ou atteintes aux droits de tierces personnes qui pourraient résulter de cette utilisation.

Les exemples ou les programmes présents dans cet ouvrage sont fournis pour illustrer les descriptions théoriques. Ils ne sont en aucun cas destinés à une utilisation commerciale ou professionnelle.

Le CD-ROM qui accompagne cet ouvrage vous est offert pour vous permettre de mieux mettre en pratique les exemples du livre. Les programmes présents sur ce support peuvent nécessiter des adaptations pour fonctionner sur votre matériel.

Simon & Schuster Macmillan (France) ne pourra en aucun cas être tenu pour responsable des préjudices ou dommages de quelque nature que ce soit pouvant résulter de l'utilisation de ces exemples ou programmes.

Tous les noms de produits ou autres marques cités dans ce livre sont des marques déposées par leurs propriétaires respectifs.

Publié par Simon & Schuster Macmillan  
(France)  
19, rue Michel Le Comte  
75003 PARIS  
Tél. : 01 44 54 51 10  
Mise en page : TyPAO  
**ISBN : 2-7440-0297-6**  
**Copyright © 1997**  
**Simon & Schuster Macmillan (France)**  
Tous droits réservés

Titre original : *Teach yourself Java 1.1 in 21 Days*  
Traduit de l'américain par : Claude Raimond  
**ISBN original : 1-57521-142-4**  
**Copyright © 1997 by Sams.net Publishing**  
Tous droits réservés  
Sams.net est une marque de Macmillan  
Computer Publishing  
201 West 103rd Street  
Indianapolis, Indiana 46290. USA

Toute reproduction, même partielle, par quelque procédé que ce soit, est interdite sans autorisation préalable. Une copie par xérogaphie, photographie, film, support magnétique ou autre, constitue une contrefaçon passible des peines prévues par la loi, du 11 mars 1957 et du 3 juillet 1995, sur la protection des droits d'auteur.

# SOMMAIRE

	Présentation	1
	Introduction	3
1	Introduction à la programmation Java 1.1	13
2	Programmation orientée objet et Java	39
3	Les bases de Java	63
4	Travailler avec les objets	83
5	Tableaux, branchements conditionnels et boucles	105
6	Création des classes et des applications	129
7	Compléments sur les méthodes	147
8	Les bases des applets	165
9	Dessin, polices et couleurs	193
10	Animation, images, threads et sons	221
11	Gestion d'événements simples et de l'interactivité	259
12	Création d'interfaces utilisateurs avec AWT	301
13	Création d'interfaces utilisateurs évoluées avec AWT	353
14	Réseaux, API évoluées et compléments divers	407
15	Modificateurs	461
16	Packages, interfaces et classes internes	483
17	Exceptions	507
18	Multithreading	527
19	Flux Java et E/S	549
20	Méthodes et bibliothèques natives	581
21	Sous le capot	615
A	Résumé du langage Java	667
B	Diagrammes des hiérarchies des classes	675
C	Bibliothèques de classes Java	703
D	Différence entre Java, C et C++	723
E	Contenu du CD-ROM	729
	Index	733



<b>Présentation</b>		
Quoi de neuf ? .....	1	
Public visé .....	1	
Conventions .....	2	
<b>Introduction</b>		
Organisation de l'ouvrage .....	3	
Sites Web pour des informations complémentaires .....	5	
Contenu du CD-ROM .....	6	
Logiciels pour Windows .....	6	
Java .....	6	
Outils HTML .....	6	
Applications graphiques, vidéo et son .....	7	
Utilitaires .....	7	
Logiciel pour Macintosh .....	7	
Java .....	7	
Outils HTML .....	7	
Utilitaires .....	8	
Le shareware .....	8	
Installation du CD-ROM .....	8	
Procédure d'installation pour Windows 95/NT4 ....	8	
Procédure d'installation pour Windows NT 3.51 ...	8	
Procédure d'installation pour Macintosh .....	9	
 <b>PARTIE 1 - LE LANGAGE JAVA</b>		
<b>1 Introduction à la programmation Java 1.1</b>		
Java .....	14	
Java hier, aujourd'hui et demain .....	16	
Pourquoi apprendre Java ? .....	17	
Java ne dépend d'aucune plate-forme .....	18	
Java est orienté objet .....	20	
Java est facile à apprendre .....	20	
Programmer avec Java .....	21	
Acquisition d'un environnement de développement Java .....	21	
Installation du JDK et des fichiers d'exemples ....	22	
Configuration du JDK .....	23	
Création d'une application Java .....	24	
Création d'une applet Java .....	29	
Résolution des problèmes .....	33	
Résumé .....	34	
Questions – réponses .....	34	
<b>2 Programmation orientée objet et Java</b>		
Penser en termes d'objets : l'analogie .....	40	
Objets et classes .....	41	
Attributs et comportement .....	43	
Attributs .....	43	
Comportement .....	43	
Création d'une classe .....	44	
Héritage, interfaces et packages .....	49	
Héritage .....	50	
Création d'une hiérarchie de classes .....	51	
Fonctionnement de l'héritage .....	53	
Héritage simple ou multiple .....	55	
Interfaces et packages .....	55	
Création d'une sous-classe .....	57	
Résumé .....	60	
Questions – réponses .....	61	
<b>3 Les bases de Java</b>		
Instructions et expressions .....	64	
Types de données et de variables .....	65	
Déclaration des variables .....	66	
Noms de variables .....	67	
Types de variables .....	67	
Attribution de valeurs aux variables .....	69	
Commentaires .....	69	
Constantes .....	70	
Constantes nombre .....	70	
Constantes booléennes .....	70	
Constantes caractère .....	70	
Constantes chaîne .....	71	
Opérateurs et expressions .....	72	
Opérateurs arithmétiques .....	72	
Affectations .....	74	
Incrémentatation et décrémentation .....	75	
Comparaisons .....	77	
Opérations logiques .....	77	
Opérateurs sur les bits .....	78	
Hiérarchie des opérateurs .....	78	
Arithmétique sur les chaînes .....	80	
Résumé .....	80	
Questions – réponses .....	82	
<b>4 Travailler avec les objets</b>		
Création de nouveaux objets .....	84	
Opérateur new .....	84	
Opérations réalisées par new .....	86	

<i>Gestion de la mémoire</i> .....	86	Création de variables d'instance et de variables de classe .....	131
Variables d'instance et de classe :		<i>Définition des variables d'instance</i> .....	131
accès, fixation d'une valeur .....	87	<i>Constantes</i> .....	132
<i>Accès aux valeurs</i> .....	87	<i>Variables de classe</i> .....	133
<i>Mise à jour des valeurs</i> .....	87	Création de méthodes .....	133
<i>Variables de classe</i> .....	88	<i>Définition des méthodes</i> .....	133
Appel des méthodes .....	89	<i>Le mot clé this</i> .....	135
<i>Méthodes de classe</i> .....	91	<i>Portée des variables et définition des méthodes</i> ...	136
Références aux objets .....	92	<i>Passage d'arguments à des méthodes</i> .....	137
Conversion de types .....		<i>Méthodes de classe</i> .....	139
de données primaires et d'objets .....	93	Création d'applications Java .....	140
<i>Conversion des types primaires</i> .....	94	<i>Classes auxiliaires</i> .....	141
<i>Conversion des objets</i> .....	94	Applications Java et arguments	
<i>Transformation de types primaires en objets</i> <i>et vice versa</i> .....	95	en ligne de commande .....	141
Compléments divers .....	96	<i>Passage des arguments à des programmes Java</i> ...	141
<i>Comparaison des objets</i> .....	96	<i>Traitement des arguments</i>	
<i>Identification de la classe d'un objet</i> .....	98	<i>dans un programme Java</i> .....	142
Inspection des classes et des méthodes		Résumé .....	144
par la réflexion .....	99	Questions – réponses .....	145
Bibliothèque de classes Java .....	101	<b>7 Compléments sur les méthodes</b>	
Résumé .....	102	Création de méthodes ayant le même nom	
Questions – réponses .....	102	et des arguments différents .....	148
<b>5 Tableaux, branchements conditionnels et boucles</b>		Méthodes constructeurs .....	152
Tableaux .....	106	<i>Constructeurs de base</i> .....	153
<i>Déclaration des variables tableau</i> .....	106	<i>Appel d'un autre constructeur</i> .....	154
<i>Création des objets tableau</i> .....	107	<i>Surcharge des constructeurs</i> .....	154
<i>Accès aux éléments de tableaux</i> .....	108	Rédéfinition de méthodes .....	156
<i>Mise à jour des éléments de tableaux</i> .....	108	<i>Création de méthodes redéfinissant</i> <i>des méthodes existantes</i> .....	156
<i>Tableaux multidimensionnels</i> .....	111	<i>Appel de la méthode d'origine</i> .....	158
Blocs d'instructions .....	111	<i>Redéfinition de constructeurs</i> .....	159
Branchement conditionnel <i>if</i> .....	112	Méthodes de terminaison .....	161
<i>Opérateur conditionnel</i> .....	114	Résumé .....	162
Branchement conditionnel <i>switch</i> .....	115	Questions – réponses .....	162
Boucles <i>for</i> .....	118	<hr/>	
Boucles <i>while</i> et <i>do</i> .....	120	<b>PARTIE 2 - DÉVELOPPEMENT D'APPLETS</b>	
<i>Boucles while</i> .....	120	<hr/>	
<i>Boucles do...while</i> .....	122	<b>8 Les bases des applets</b>	
Sortir des boucles .....	123	Différences entre applets et applications .....	166
<i>Boucles avec étiquettes</i> .....	124	Applets 1.02 et 1.1 .....	168
Résumé .....	125	Création d'applets .....	169
Questions – réponses .....	126	Principales activités des applets .....	170
<b>6 Création des classes et des applications</b>			
Définition des classes .....	130		

<i>Initialisation</i> .....	170	<b>10 Animation, images, threads et sons</b>	
<i>Démarrage</i> .....	171	Création d'animations en Java .....	222
<i>Arrêt</i> .....	171	<i>Peindre et repeindre</i> .....	223
<i>Destruction</i> .....	171	<i>Démarrage et arrêt de l'exécution d'une applet</i> ...	223
<i>Peinture</i> .....	172	<i>Un chaînon manquant : les threads</i> .....	224
Une applet simple .....	172	<i>Écriture d'applets avec des threads</i> .....	225
Inclusion d'une applet dans une page Web .....	174	<i>Mise en œuvre</i> .....	227
<i>La balise &lt;APPLET&gt;</i> .....	174	Réduction du clignotement de l'animation .....	229
<i>Test du résultat</i> .....	175	<i>Le clignotement, et comment l'éviter</i> .....	230
<i>Publication d'applets Java sur le Web</i> .....	175	<i>Redéfinition d'update()</i> .....	230
Compléments sur la balise <APPLET> .....	176	<i>Première solution : pas d'effacement de l'écran</i> ...	231
<i>ALIGN</i> .....	176	Récupération et utilisation d'images .....	233
<i>HSPACE et VSPACE</i> .....	178	<i>Obtention d'images</i> .....	233
<i>CODE et CODEBASE</i> .....	179	<i>Dessin des images</i> .....	235
Archives Java .....	179	<i>Note sur les observateurs d'images</i> .....	237
<i>Autres formats d'archivage</i> .....	180	<i>Modification d'images</i> .....	238
Identification d'applets		Création d'animations à base d'images .....	238
par des signatures numériques .....	182	<i>Un exemple : Neko</i> .....	238
Passation de paramètres à des applets .....	184	<i>Étape 1 : rassemblement des images</i> .....	239
Résumé .....	189	<i>Étape 2 : organisation et chargement</i>	
Questions – réponses .....	190	<i>des images de l'applet</i> .....	239
		<i>Étape 3 : animation des images</i> .....	240
<b>9 Dessin, polices et couleurs</b>		<i>Étape 4 : finition</i> .....	244
La classe <i>Graphics</i> .....	194	Récupération et utilisation des sons .....	247
<i>Système de coordonnées graphiques</i> .....	195	Complément sur le clignotement :	
Dessin et remplissage .....	196	le double buffer .....	250
<i>Lignes</i> .....	196	<i>Création d'applets avec double buffer</i> .....	250
<i>Rectangles</i> .....	197	<i>Note sur l'élimination des contextes Graphics</i> .....	251
<i>Polygones</i> .....	198	<i>Un exemple : le jeu de dames</i> .....	252
<i>Ovales</i> .....	201	Résumé .....	256
<i>Arcs</i> .....	202	Questions – réponses .....	256
<i>Un exemple simple de dessin</i> .....	206		
<i>Copie et effacement</i> .....	207	<b>11 Gestion d'événements simples et de l'interactivité</b>	
Texte et polices de caractères .....	208	Une théorie unifiée des événements .....	260
<i>Création d'objets Font</i> .....	208	<i>En quoi consistent les événements ?</i> .....	260
Dessin de caractères et de chaînes .....	209	<i>Les deux modèles d'événement</i> .....	261
<i>Obtention d'informations sur une police</i>		<i>Quel modèle utiliser ?</i> .....	262
<i>de caractères</i> .....	211	Traitement des clics de souris	
Couleur .....	213	dans le modèle 1.02 .....	263
<i>Objets Color</i> .....	214	<i>Événements souris en bas et souris en haut</i> .....	263
<i>Test et fixation des couleurs courantes</i> .....	215	<i>Un exemple : Spots</i> .....	265
<i>Un exemple simple de couleurs</i> .....	215	<i>Doubles clics</i> .....	268
Couleurs système standard .....	217	Traitement des mouvements de souris	
Résumé .....	218	dans le modèle 1.02 .....	268
Questions – réponses .....	219	<i>Événements glissement de souris</i>	
		<i>et mouvement de souris</i> .....	269

<i>Événements entrée souris et sortie souris</i> .....	269	Focalisation entrée (input focus)	
<i>Un exemple : dessin de lignes</i> .....	270	et exploitation sans souris (1.1 seulement) .....	333
Traitement des événements clavier		Traitement des événements d'interface utilisateur	
dans le modèle d'événements 1.02 .....	274	dans le modèle 1.02 .....	334
<i>Les méthodes keyDown() et keyUp()</i> .....	274	<i>Traitement des événements d'action</i> .....	335
<i>Touches standard</i> .....	275	<i>Traitement des autres événements</i> .....	337
<i>Un exemple : entrée, affichage</i>		Traitement des événements d'interface utilisateur	
<i>et déplacement de caractères</i> .....	276	dans le modèle d'événements 1.1 .....	338
<i>Test des touches de modification</i>		<i>Écouteurs des composants de base de l'interface</i> ...	338
<i>et des boutons de souris multiples</i> .....	279	<i>Enregistrement des écouteurs</i>	
Gestionnaire d'événement générique 1.02 :		<i>de composants d'interface utilisateur</i> .....	339
<i>handleEvent()</i> .....	281	<i>Différences entre les événements 1.02 et 1.1</i> .....	340
Passage au modèle d'événement 1.1 .....	283	Exemple : commutateur de couleur	
<i>Qu'est-ce qui ne va pas dans l'ancien modèle ?</i> ...	283	d'arrière-plan .....	341
<i>Fonctionnement du nouveau modèle</i> .....	284	<i>Ajout du code des événements (1.02)</i> .....	342
Le modèle d'événement 1.1 .....	286	<i>Ajout du code des événements (1.1)</i> .....	344
<i>Étape 1 : détermination des événements</i>		Résumé .....	349
<i>mis en œuvre par l'applet</i> .....	286	Questions – réponses .....	349
<i>Étape 2 : implémentation de l'interface</i> .....	287		
<i>Étape 3 : enregistrement de l'écouteur</i> .....	289	<b>13 Création d'interfaces utilisateurs évoluées avec AWT</b>	
<i>Conversion de code d'événement 1.02 en 1.1</i> .....	291	Imbrication de composants .....	354
<i>Un exemple : conversion de Lines</i> .....	291	<i>Panneaux imbriqués</i> .....	354
Événements souris et clavier		<i>Panneaux imbriqués dans le modèle</i>	
dans le modèle d'événements 1.1 .....	295	<i>d'événements 1.02</i> .....	355
<i>Clics de souris et mouvements de souris</i> .....	295	Autres composants d'interface utilisateur .....	356
<i>Appuis sur les touches</i> .....	296	<i>Zones de texte</i> .....	356
Résumé .....	297	<i>Listes à défilement</i> .....	358
Questions – réponses .....	298	<i>Barres de défilement et glissières</i> .....	361
		<i>Carreaux à défilement (1.1 seulement)</i> .....	365
<b>12 Création d'interfaces utilisateurs avec AWT</b>		<i>Canevas</i> .....	366
<i>Vue d'ensemble d'AWT</i> .....	302	<i> Curseurs</i> .....	367
<i>Composants de base de l'interface utilisateur</i> .....	304	Jouer avec les composants .....	368
<i>Adjonction de composants à une applet</i> .....	305	Un exemple complet :	
<i>Étiquettes</i> .....	305	<i>convertisseur de RGB à HSB</i> .....	370
<i>Boutons</i> .....	307	<i>Conception et création de la mise en page</i>	
<i>Cases à cocher</i> .....	308	<i>de l'applet</i> .....	371
<i>Boutons radio</i> .....	310	<i>Définition des sous-panneaux</i> .....	373
<i>Menus de sélection</i> .....	311	<i>Traitement des événements</i> .....	376
<i>Champs de texte</i> .....	313	<i>Mise à jour du résultat</i> .....	377
Panneaux et mise en page .....	315	<i>Le code source complet</i> .....	380
<i>Gestionnaires de mise en page : vue d'ensemble</i> ..	316	Fenêtres, cadres et boîtes de dialogue .....	383
<i>La classe FlowLayout</i> .....	316	<i>Les classes Window d'AWT</i> .....	383
<i>Mise en page du type grille (GridLayout)</i> .....	318	<i>Cadres</i> .....	384
<i>Mises en page du type BorderLayout</i> .....	319	<i>Boîtes de dialogue</i> .....	388
<i>Mises en page du type cartes (CardLayout)</i> .....	321	<i> Curseurs (1.02 seulement)</i> .....	393
<i>Mises en page du type GridBagLayout</i> .....	321	<i>Événements de fenêtre</i> .....	394
<i>Retraits (insets)</i> .....	332		



Les menus .....	395
<i>Menus et barres de menu</i> .....	396
<i>Articles de menu</i> .....	397
<i>Un exemple : une fenêtre popup avec des menus</i> ...	400
Création d'applications AWT autonomes .....	403
Résumé .....	404
Questions – réponses .....	405

## 14 Réseaux, API évoluées et compléments divers

Divers procédés concernant les applets .....	408
<i>Méthode showStatus()</i> .....	408
<i>Informations sur l'applet</i> .....	409
<i>Création de liens à l'intérieur d'applets</i> .....	409
<i>Communication entre applets</i> .....	412
Les réseaux en Java .....	413
<i>Ouverture de connexions sur le Web</i> .....	414
<i>Sockets</i> .....	418
Trivia : un socket simple client/serveur .....	420
<i>Conception de Trivia</i> .....	420
<i>Implémentation du serveur Trivia</i> .....	421
<i>Implémentation du client Trivia</i> .....	427
<i>Exécution de Trivia</i> .....	428
Impression (Java 1.1) .....	429
Couper, copier et coller (1.1 seulement) .....	431
<i>Création d'objets transférables</i> .....	432
<i>Le Presse-papiers</i> .....	433
Internationalisation (Java 1.1) .....	437
<i>Utilisation de locales</i> .....	437
<i>Formatage international des données</i> .....	439
<i>Pour aller plus loin</i> .....	442
Fonctions avancées de Java 1.1 .....	442
<i>JavaBeans</i> .....	443
<i>Sécurité</i> .....	449
<i>RMI (Remote Method Invocation)</i> .....	450
<i>JDBC (Java Database Connectivity)</i> .....	453
Résumé .....	454
Questions – réponses .....	455

## PARTIE 3 - INFORMATIONS AVANCÉES

### 15 Modificateurs

Les modificateurs .....	462
Contrôle d'accès aux méthodes et aux variables... 464	
<i>Raisons de l'importance du contrôle d'accès</i> .....	464

<i>Les quatre p de la protection</i> .....	465
<i>Protection des méthodes et héritage</i> .....	470
<i>Protection des variables d'instance</i> <i>et méthodes d'accès</i> .....	470
Variables et méthodes de classe .....	473
Finalisation de classes, de méthodes et de variables .....	475
<i>Finalisation de classes</i> .....	476
<i>Finalisation de variables</i> .....	477
<i>Finalisation de méthodes</i> .....	477
Classes et méthodes abstraites .....	478
Résumé .....	480
Questions – réponses .....	480

### 16 Packages, interfaces et classes internes

La programmation en grand et la programmation en petit .....	484
Présentation des packages .....	485
Utilisation des packages .....	486
<i>Noms complets de classes et de packages</i> .....	487
<i>La commande import</i> .....	487
<i>Conflits sur les noms</i> .....	488
<i>Note sur CLASSPATH et l'emplacement</i> <i>des classes</i> .....	488
Création de vos propres packages .....	489
<i>Choix d'un nom de package</i> .....	489
<i>Création de la structure de répertoire</i> .....	490
<i>Utilisation de package pour ajouter</i> <i>une classe à un package</i> .....	490
<i>Packages et protection des classes</i> .....	490
Présentation des interfaces ? .....	493
<i>Le problème de l'héritage unique</i> .....	493
<i>Conception abstraite et implémentation concrète...</i>	495
<i>Interfaces et classes</i> .....	496
Implémentation et utilisation des interfaces .....	496
<i>Mot clé implements</i> .....	496
<i>Implémentation d'interfaces multiples</i> .....	498
<i>Autres utilisations des interfaces</i> .....	498
Création et extension des interfaces .....	499
<i>Nouvelles interfaces</i> .....	500
<i>Méthodes à l'intérieur des interfaces</i> .....	501
<i>Extension des interfaces</i> .....	501
<i>Un exemple : énumération de listes liées</i> .....	502
Classes internes .....	504
Résumé .....	505
Questions – réponses .....	505

<b>17 Exceptions</b>	
Exceptions, la manière traditionnelle, source de confusions .....	508
Exceptions Java .....	509
Gestion des exceptions .....	511
<i>Vérification de cohérence en matière d'exception</i> .....	511
<i>Protection du code et capture des exceptions</i> .....	512
<i>La clause Finally</i> .....	515
Déclaration de méthodes susceptibles de lever des exceptions .....	516
<i>La clause throws</i> .....	516
<i>Quelles exceptions lever ?</i> .....	517
<i>Le passage des exceptions</i> .....	518
<i>throws et l'héritage</i> .....	518
Création et levage de vos propres exceptions .....	519
<i>Levage des exceptions</i> .....	519
<i>Création des exceptions</i> .....	520
<i>Combinaison de throws, try et throw</i> .....	521
Quand utiliser des exceptions et quand s'en abstenir .....	521
<i>Quand utiliser les exceptions</i> .....	521
<i>Exceptions à l'usage des exceptions</i> .....	522
<i>Du mauvais goût en matière d'exceptions</i> .....	522
Résumé .....	523
Questions – réponses .....	524
<b>18 Multithreading</b>	
Les bases des threads .....	528
Problème de parallélisme .....	529
Penser en termes de multithread .....	530
<i>A propos des points</i> .....	532
<i>Protéger une variable de classe</i> .....	534
Création et utilisation des threads .....	535
<i>Interface Runnable</i> .....	536
Savoir quand un thread s'arrête .....	540
Planification des threads .....	541
<i>Planification préemptive et non préemptive</i> .....	541
<i>Test du programmeur</i> .....	542
Résumé .....	546
Questions – réponses .....	546
<b>19 Flux Java et E/S</b>	
<i>Flux d'entrée (input streams) et lecteurs (readers)</i> .....	551
<i>Les classes abstraites InputStream et Reader</i> .....	551
<i>FileInputStream et FileReader</i> .....	557
<i>PipedInputStream et PipedReader</i> .....	566
<i>Sequence InputStream</i> .....	566
<i>StringBufferInputStream et StringReader</i> .....	567
Output Streams et Writers .....	567
<i>Les classes abstraites OutputStream et Writer</i> .....	567
<i>ByteArrayOutputStream et CharArrayWriter</i> .....	569
<i>FileOutputStream et FileWriter</i> .....	571
<i>FilterOutputStream et FilterWriter</i> .....	571
<i>BufferedOutputStream et BufferedWriter</i> .....	572
<i>PipedOutputStream et PipedWriter</i> .....	577
Classes apparentées .....	577
Résumé .....	578
Questions – réponses .....	579
<b>20 Méthodes et bibliothèques natives</b>	
Inconvénients des méthodes natives .....	582
Une efficacité illusoire .....	583
<i>Optimisations intégrées</i> .....	585
<i>Astuces simples d'optimisation</i> .....	586
Ecrire des méthodes native .....	587
<i>Classe exemple</i> .....	587
<i>Génération de fichiers d'en-tête</i> .....	589
<i>Création de SimpleFile.c</i> .....	591
Bibliothèque native .....	594
<i>Tout lier</i> .....	594
<i>Utiliser la bibliothèque</i> .....	594
Interface native Java (JNI) .....	594
<i>Ce que fait JNI</i> .....	595
<i>Pourquoi JNI a été créé</i> .....	596
<i>Comment marche JNI</i> .....	596
<i>Les fonctions JNI en détail</i> .....	602
<i>L'API d'invocation</i> .....	611
Résumé .....	613
Questions – réponses .....	614
<b>21 Sous le capot</b>	
Un avenir prometteur .....	616
Une vision puissante .....	617
La machine virtuelle Java .....	618
<i>Vue d'ensemble</i> .....	619
<i>Les parties fondamentales</i> .....	620
<i>Le pool de constantes</i> .....	624
Les pseudo-codes en détail .....	625
<i>L'interpréteur de pseudo-code</i> .....	625
<i>Compilateurs en temps réel</i> .....	625
<i>Le traducteur java2c</i> .....	626
<i>Les pseudo-codes</i> .....	627

Le format de fichier .class .....	646
Les limitations .....	649
Les descripteurs de méthodes .....	649
Le programme de récupération de mémoire .....	650
<i>Le problème</i> .....	651
<i>La solution</i> .....	651
<i>Le récupérateur de mémoire parallèle de Java</i> .....	654
La question de sécurité .....	654
<i>Pourquoi s'inquiéter ?</i> .....	654
<i>Pourquoi il ne faut pas s'inquiéter</i> .....	654
<i>Le modèle de sécurité de Java</i> .....	655
Résumé .....	663
Questions – réponses .....	663

<i>Classes</i> .....	706
Java.util.zip (Java 1.1) .....	707
<i>Interfaces</i> .....	707
<i>Classes</i> .....	707
java.io .....	707
<i>Interfaces</i> .....	707
<i>Classes</i> .....	708
java.net .....	709
<i>Interfaces</i> .....	709
<i>Classes</i> .....	710
java.awt .....	710
<i>Interfaces</i> .....	710
java.awt.datatransfer (Java 1.1) .....	712
<i>Interfaces</i> .....	712
<i>Classes</i> .....	713
java.awt.event (Java 1.1) .....	713
<i>Interfaces</i> .....	713
<i>Classes</i> .....	713
java.awt.image .....	714
<i>Interfaces</i> .....	714
<i>Classes</i> .....	714
java.awt.peer .....	715
java.applet .....	715
<i>Interfaces</i> .....	715
<i>Classes</i> .....	715
java.beans .....	716
<i>Interfaces</i> .....	716
<i>Classes</i> .....	716
java.rmi .....	717
<i>Interfaces</i> .....	717
<i>Classes</i> .....	717
java.rmi.dgc .....	717
<i>Interfaces</i> .....	717
<i>Classes</i> .....	718
java.rmi.registry .....	718
<i>Interfaces</i> .....	718
<i>Classes</i> .....	718
java.rmi.server .....	718
<i>Interfaces</i> .....	718
<i>Classes</i> .....	719
java.security .....	719
<i>Interfaces</i> .....	719
<i>Classes</i> .....	719
java.security.acl .....	720
<i>Interfaces</i> .....	720
java.security.interfaces .....	721

## PARTIE 4 - ANNEXES

### A Résumé du langage Java

Mots réservés .....	668
Commentaires .....	668
Constantes .....	669
Déclaration de variable .....	669
Affectation de variable .....	670
Opérateurs .....	670
Objets .....	671
Tableaux .....	672
Boucles et branchements conditionnels .....	672
Définitions de classe .....	673
Définitions de méthode et de constructeur .....	673
Packages, interfaces et importation .....	674
Exceptions et mise en garde .....	674

### B Diagrammes des hiérarchies des classes

### C Bibliothèques de classes Java

java.lang .....	704
<i>Interfaces</i> .....	704
<i>Classes</i> .....	704
java.lang.reflect (Java 1.1) .....	705
<i>Interfaces</i> .....	705
<i>Classes</i> .....	705
java.math (Java 1.1) .....	705
<i>Classes</i> .....	705
java.util .....	706
<i>Interfaces</i> .....	706

<i>Interfaces</i> .....	721
java.sql .....	721
<i>Interfaces</i> .....	721
<i>Classes</i> .....	721
java.text .....	722
<i>Interfaces</i> .....	722
<i>Classes</i> .....	722
<b>D</b> <b>Différence entre Java, C et C++</b>	
Pointeurs .....	724
Tableaux .....	724
Chaînes .....	725
Gestion de mémoire .....	725
Types de données .....	725
Opérateurs .....	726
Commande .....	726
Arguments .....	726
Autres différences .....	727
<b>E</b> <b>Contenu du CD-ROM</b>	
Logiciels pour Windows .....	730
<i>Java</i> .....	730
<i>Outils HTML</i> .....	730
<i>Applications graphiques, vidéo et son</i> .....	731
<i>Utilitaires</i> .....	731
Logiciels pour Macintosh .....	731
<i>Java</i> .....	731
<i>Outils HTML</i> .....	731
<i>Logiciels graphiques</i> .....	732
<i>Utilitaires</i> .....	732
A propos du shareware .....	732

# Présentation

## Quoi de neuf ?

L'offre explosive d'outils pour la création d'applications et la diversité des réalisations basées sur Java, visant le Web ou des applications de caractère général, sont une source inépuisable de thèmes de discussion.

La présente édition est une révision complète du livre original JAVA paru dans la collection Le Programmeur. Elle vous propose une description de Java 1.1, des améliorations substantielles de l'ouvrage précédent, entièrement revu et mis à jour, ainsi que des exemples supplémentaires :

- Nouveaux packages, ou packages étendus de Java 1.1.
- Nouveaux thèmes Java tels que les classes internes (inner classes) et la réflexion (reflection).
- Exemples de mise en œuvre des nouvelles possibilités de Java 1.1.
- Chapitre supplémentaire dédié à l'AWT (Abstract Windowing Toolkit), récemment étendu.
- Diagrammes montrant la nouvelle hiérarchie des classes pour Java 1.1, et liste du contenu de la bibliothèque des classes, également pour Java 1.1.
- Etude de JavaBeans, le nouveau composant technologique de Java.
- Nouvelle présentation des événements, étude des changements relatifs aux événements.
- Vue d'ensemble de JDBC et de RMI.
- Traitement étendu des réseaux.
- Nouvelles fonctions de sécurité de Java, y compris le mode de création d'applets signées.
- Etude approfondie du passage de Java 1.02 à Java 1.1.

## Public visé

Ce livre s'adresse aux personnes qui ont acquis les bases de la programmation, aux programmeurs chevronnés, mais également aux débutants. Si vous connaissez les notions de variable, de boucle et de fonction, ce livre est à votre portée. Voici quelques bonnes raisons de le lire :

- Vous maîtrisez le HTML et la programmation CGI (en Perl, AppleScript, Visual basic, etc.) et vous souhaitez en apprendre plus sur la conception de pages Web.

- Vous avez étudié le Basic ou le Pascal à l'école, acquis les notions de base de la programmation et entendu parler de la puissance et de la simplicité de Java.
- Vous programmez en C et en C++ depuis des années et Java vous intrigue.
- Java est particulièrement adapté aux applets basées sur le Web. Convient-il aussi à la création d'applications plus générales ?
- Vous savez programmer, mais pas en orienté objet. Ce n'est pas un handicap, même si la connaissance de ce type de programmation facilite la compréhension des deux premiers chapitres.

Pour les programmeurs débutants, ce livre est sans doute un peu ardu. Toutefois, Java est un langage approprié à l'apprentissage de la programmation. Ainsi donc, en y consacrant du temps, et grâce à l'étude de tous les exemples, vous progresserez et vous commencerez bientôt à créer vos propres applets.

## Conventions

### Info

*Présente une information technique complémentaire.*

### Astuce

*Propose des conseils ou des simplifications.*

### Attention

*Met en garde contre un éventuel problème et propose des solutions.*

### Lexique

*Présente et/ou définit un nouveau terme, composé en italique.*

### Analyse

Expose une analyse du code par l'auteur.

# Introduction

Le World Wide Web est depuis longtemps un moyen de distribuer des informations statiques à un grand nombre de personnes. Mais avec l'arrivée des formes et des images, les pages Web sont devenues interactives. Cette interactivité étant simplement un nouveau mode d'accès aux mêmes informations, les limites du Web devenaient évidentes. Les concepteurs ont alors essayé d'en étendre les possibilités. D'autres innovations, comme le serveur Netscape destiné à créer des animations dynamiques, ont été conçues uniquement pour supporter des documents statiques avec des images et du texte.

Java intervient alors pour les pages Web, avec la possibilité de contenir des applets, de petits programmes créant des animations, des présentations multimédias, des jeux (vidéo) en temps réel, des jeux réseau multi-utilisateurs et une véritable interactivité. En fait, les applets Java réalisent tout ce qu'un petit programme peut faire. De plus, elles sont téléchargées sur le réseau et exécutées dans une page Web par un programme de navigation compatible Java. C'est une réelle avancée, au-delà du concept initial du Web.

L'écriture des applets présente un inconvénient : elles sont écrites en langage Java, qui est un langage de programmation. La création d'une applet Java est donc plus ardue que celle d'une page Web ou d'une forme avec du code HTML.

De nombreux outils et de nombreux programmes sont désormais disponibles pour faciliter la création d'applets Java. Cependant, la seule façon de se plonger dans Java est d'en apprendre le langage et de commencer à en tester le code. Même s'il existe des outils, la réalisation de certaines fonctions nécessite la connaissance du langage.

Ce livre concerne le langage Java dans son intégralité. Vous y apprendrez à créer non seulement des applets pour le Web, mais aussi des applications. Les applications sont des programmes Java plus généraux dont l'exécution ne nécessite pas un logiciel de navigation.

## Organisation de l'ouvrage

Ce livre couvre le langage Java et ses bibliothèques de classes en vingt et un chapitres regroupés en trois parties, chacune d'elles couvrant un aspect différent du développement des applets Java et des applications.

La première partie concerne l'étude du langage de programmation Java :

- Le Chapitre 1 constitue l'introduction : qu'est-ce que Java, quel est son intérêt et comment se le procurer. Vous y créez aussi vos premières applications et applets Java.

- Le Chapitre 2 explore les concepts de la programmation orientée objet tels qu'ils s'appliquent dans Java.
- Le Chapitre 3 débute l'étude plus approfondie du langage, avec les composants de base de Java : les types de données, les variables et les expressions, telles que les expressions arithmétiques et les comparaisons.
- Le Chapitre 4 concerne la manipulation des objets dans Java : comment les créer, accéder à leurs variables et appeler leurs méthodes, et comment les comparer et les copier. Il fournit également un aperçu des bibliothèques de classes de Java.
- Le Chapitre 5 introduit les tableaux, les instructions de contrôle et les boucles.
- Dans le Chapitre 6, vous apprendrez à créer des classes et à les introduire dans une application Java (programme Java pouvant tourner sans navigateur Web), les classes étant les blocs de base d'un programme Java.
- Le Chapitre 7 met en œuvre les connaissances acquises au Chapitre 6. Vous apprendrez à créer et à utiliser les méthodes, et vous étudierez leur redéfinition et leur polymorphisme, ainsi que la création de méthodes particulières appelées constructeurs.

La deuxième partie est consacrée aux applets et aux bibliothèques de classes de Java :

- Le Chapitre 8 est consacré aux bases des applets : il montre en quoi elles sont différentes des applications, comment les créer, et quelles sont les parties les plus importantes de leur cycle de vie. Vous apprendrez aussi à créer des pages HTML contenant des applets Java.
- Le Chapitre 9 traite des classes Java servant à dessiner des formes et des caractères (en noir, blanc, ou toute autre couleur) à l'écran.
- Le Chapitre 10 explique comment animer les formes créées au Chapitre 9. Il traite également de la définition des threads et de leur utilisation, ainsi que de l'addition de sons au moyen de Java.
- Le Chapitre 11 plonge dans l'interactivité, avec la gestion de la souris et du clavier dans les applets Java.
- Le Chapitre 12 est ambitieux. Il explique comment utiliser les outils de fenêtrage abstrait de Java (AWT = Abstract Windowing Toolkit) pour créer une interface utilisateur dans une applet, incluant menus, boutons, boîtes de contrôle et autres éléments.
- Le Chapitre 13 approfondit l'étude de l'AWT.
- Le Chapitre 14 explore les autres bibliothèques de classes importantes servant à la création des applets : fenêtres, dialogues, interconnexion de réseau, etc.

La troisième partie aborde les sujets pointus, utiles pour l'écriture de programmes Java plus complexes.

- Le Chapitre 15 donne des précisions sur les modificateurs du langage Java, relatifs aux classes et aux méthodes dites finales et abstraites, en vue de protéger les informations privées d'une classe vis-à-vis des autres classes.



- Le Chapitre 16 couvre les interfaces et les packages utiles à la déclaration des méthodes (afin d'en faciliter la réutilisation) et au regroupement des classes en catégories.
- Le Chapitre 17 traite les exceptions, telles que les erreurs, les avertissements ou toute autre situation anormale générée par le système ou les programmes.
- Le Chapitre 18 complète l'étude des threads entamée au Chapitre 10 et donne un aperçu du multithreading, qui permet à différentes parties d'un programme Java de se dérouler en parallèle.
- Le Chapitre 19 donne tous les détails sur les flux d'entrées/sorties dans la bibliothèque des entrées/sorties de Java.
- Le Chapitre 20 concerne le code natif : comment introduire du code C dans les programmes Java pour obtenir une fonction supplémentaire ou gagner en performance.
- Le Chapitre 21 donne une vue d'ensemble des détails techniques qui assurent en arrière-plan le fonctionnement de Java : le compilateur de pseudo-code et l'interpréteur, les techniques utilisées par Java pour assurer l'intégrité et la sécurité des programmes, et la façon dont le ramasse-miettes de Java contribue à l'utilisation optimale des ressources.

## Sites Web pour des informations complémentaires

Avant, pendant et après la lecture de ce livre, plusieurs sites Web peuvent intéresser un développeur Java.

Le site Web Java officiel se trouve à l'adresse <http://java.sun.com/>. Vous y trouverez le logiciel de développement de Java et de la documentation sur tous les aspects du langage, y compris une première présentation de Java 1.1. Il possède plusieurs sites miroirs dont il fournit la liste. Ainsi, vous utiliserez le site le plus proche sur Internet.

Il existe également un site pour les ressources développeur, appelé Gamelan, à l'adresse <http://www.gamelan.com/>. Ce site comporte un nombre considérable d'applets et d'applications, des exemples de code, des aides, et beaucoup d'autres informations sur Java et son évolution.

Ce livre possède son propre site Web à l'adresse <http://www.lne.com/web/JavaProf/>. Les informations fournies comprennent des exemples, des détails complémentaires, des corrections de ce livre et bien d'autres choses intéressantes.

Pour toute discussion sur le langage Java et les outils de développement en Java, nous vous suggérons les Newsgroups Usenet relatifs à `comp.lang.java` : `comp.lang.java.programming`, `comp.lang.java.api`, `comp.lang.java.misc`, `comp.lang.java.security`, ainsi que `comp.lang.java.tech`. Ces newsgroups constituent une excellente source de réponses aux questions et d'informations sur les nouveaux développements de Java.

# Contenu du CD-ROM

Un CD-ROM accompagne ce livre. Vous y trouverez les fichiers de tous les exemples étudiés, ainsi qu'un grand nombre d'applications et d'utilitaires.

## Info

*Vous trouverez la liste des logiciels du CD-ROM dans le fichier readme.wri (Windows) ou dans le guide du CD-ROM (Macintosh).*

# Logiciels pour Windows

## Java

- Kit de développement Java (JDK) de Sun pour Windows 95/NT, version 1.1 (incluant aussi les versions Solaris)
- Kit de développement Beans de Sun pour Windows 95/NT (versions Solaris incluses)
- Exemples d'applets Java
- Exemples de scripts JavaScript
- JFactory
- Version d'essai de Jamba pour Windows 95/NT
- Jpad
- Démo Jpad Pro
- Démo Kawa
- Démo Studio J++
- Démo Javelin
- Assistant base de données Jdesigner Pro pour Java

## Outils HTML

- Editeur HTML 32 bits Hot Dog
- Editeur HTML HoTMeTal
- Editeur HTML HTMLed
- Editeur HTML WebEdit Pro
- Démo Spider 1.2
- Démo Web Analyzer

## Applications graphiques, vidéo et son

- Editeur Goldwave pour éditer, enregistrer et jouer des sons
- Utilitaire d'agencement d'images MapThis
- Paint Shop Pro
- Utilitaire de capture d'écran SnagIt
- Navigateur et visualisateur d'images ThumbsPlus
- Bibliothèque d'images de The Rocket Shop

## Utilitaires

- Visualisateur Acrobat d'Adobe
- WinZip pour Windows 95/NT
- Auto-extracteur WinZip

# Logiciel pour Macintosh

## Java

- Kit de développement Java de Sun pour Macintosh, version 1.0.2
- Exemples d'applets
- Exemples de scripts JavaScript

## Outils HTML

- Freeware BBEdit 3.5.1
- Démo BBEdit 4.0
- HTML edit
- Editeur HTML
- HTML Web Weaver
- HTML Markup
- WebMap
- Web Painter
- Graphic Converter
- GIFConverter
- Bibliothèque d'images de The Rocket Shop

### Utilitaires

- Lecteur Acrobat d'Adobe
- SnagIt Pro
- SoundApp
- Sparkle
- ZipIt 1.3.5 pour Macintosh

## Le shareware

Le shareware n'est pas gratuit. Ayez l'obligeance de lire toute la documentation associée aux produits correspondants (généralement dans des fichiers appelés readme.txt ou license.txt), et de vous conformer à toutes les règles énoncées.

## Installation du CD-ROM

### Procédure d'installation pour Windows 95 / NT4

1. Insérez le disque dans le lecteur de CD-ROM.
2. A partir du bureau Windows 95, faites un double-clic sur l'icône Poste de travail.
3. Faites un double-clic sur l'icône représentant votre lecteur de CD-ROM.
4. Pour exécuter le programme d'installation, faites un double-clic sur l'icône intitulée setup.exe.

### Procédure d'installation pour Windows NT 3.51

1. Insérez le CD-ROM dans le lecteur de CD-ROM.
2. Depuis le gestionnaire de programmes ou le gestionnaire de fichiers, choisissez Exécuter dans le menu fichier.
3. Tapez <lecteur>\setup et appuyez sur Entrée, <lecteur> étant la lettre de votre lecteur de CD-ROM. Par exemple, si la lettre de votre lecteur est D:, tapez D:\SETUP et appuyez sur Entrée.
4. Suivez les instructions affichées à l'écran.

## Info

*Les utilisateurs de Windows NT 3.51 peuvent accéder de deux manières au code source, aux listings et aux applets de l'ouvrage. Vous pouvez exécuter le programme d'installation de code source (SOURCE.EXE) situé dans le répertoire racine, ou choisir de décompiler séparément chaque élément. Ces fichiers compactés se trouvent dans le dossier \WINNT351\BOOK\.*

*Les utilisateurs de Windows NT 3.51 ne peuvent accéder au dossier WIN95NT4, ce dossier ayant un nom de fichier long comportant des caractères minuscules et majuscules. Ce lettrage permet aux utilisateurs de Windows 95 et de Windows NT4 d'accéder directement à ces fichiers sur le CD. Tous les autres noms de dossiers ont été traduits pour être conformes aux exigences du système d'exploitation Windows NT 3.51, et sont de ce fait accessibles sans difficulté. (NB : une tentative d'accès au dossier \WINDOWS94NT4 ne provoquera aucun incident ; tout simplement, vous ne pourrez pas lire son contenu.)*

## Procédure d'installation pour Macintosh

1. Insérez le CD-ROM dans le lecteur.
2. Lorsqu'une icône du CD apparaît sur le bureau, ouvrez le disque par un double-clic sur son icône.
3. Faites un double-clic sur l'icône appelée Guide to the CD-ROM, et suivez les instructions qui apparaissent à l'écran.

# Première Partie

# Le langage Java

- 1 Introduction à la programmation Java
- 2 Java et la programmation orientée objet
- 3 Les bases de Java
- 4 Utilisation des objets
- 5 Tableaux, branchements conditionnels et boucles
- 6 Création de classes et d'applications en Java
- 7 Compléments sur les méthodes



1

# Introduction à la programmation Java 1.1



LE PROGRAMMEUR

Bienvenue dans le monde de la programmation Java 1.1. Ce langage vous permettra de développer des programmes exécutables dans les pages Web (*applets*), aussi bien que des programmes autonomes (*applications*).

Ce chapitre présente :

- Java et sa version actuelle
- L'intérêt de Java, ses caractéristiques et ses avantages, en comparaison des autres langages de programmation
- Les débuts avec Java, les logiciels et l'environnement nécessaires, mais aussi la terminologie de base
- Les premiers programmes Java : *applet* et *application*

## Java

A voir l'extraordinaire publicité autour de Java et l'enthousiasme qu'il a provoqué, on pourrait croire que ce langage est appelé, sinon à sauver le monde, du moins à résoudre tous les problèmes de l'Internet. Il n'en est rien. On a beaucoup exagéré ses possibilités. Bien que Java soit effectivement nouveau et intéressant, ce n'est en réalité qu'un autre langage de programmation, qui permet d'écrire des programmes Internet. A cet égard, Java est plus proche de langages de programmation bien connus tels que C++, Visual Basic, ou Pascal, que d'un langage de description de pages tel que le HTML, ou d'un simple langage de scripts tel que JavaScript.

Java est un langage de programmation orienté objet développé par Sun Microsystems, plus connu pour ses stations de travail UNIX. Conçu sur le modèle de C++, Java est simple, concis et portable sur toutes les plates-formes et tous les systèmes d'exploitation. Cette portabilité existe au niveau des sources et des binaires, ce qui veut dire que les programmes Java (applets et applications) peuvent marcher sur toute machine pourvue d'une machine virtuelle Java (ce concept est expliqué plus loin).

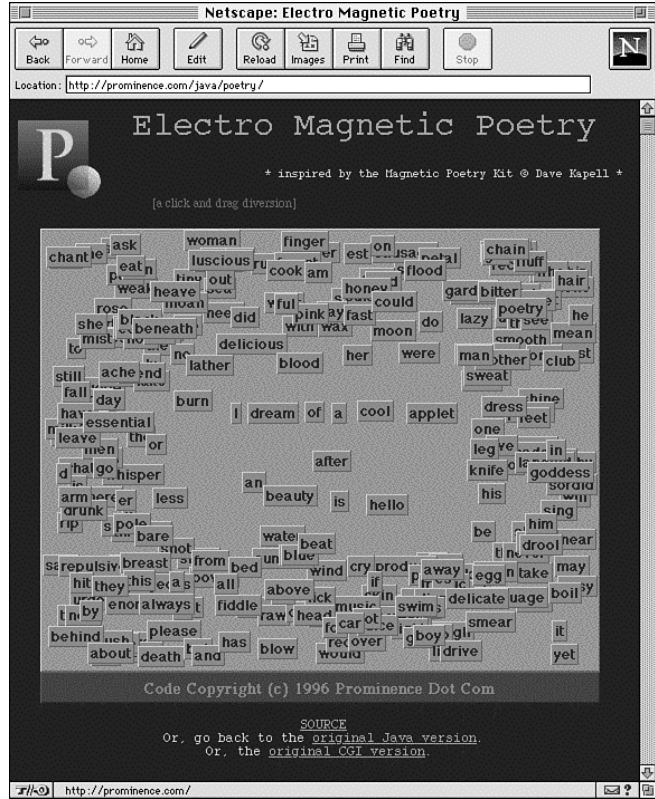
Java est généralement mentionné dans le contexte du World Wide Web, au sein duquel opèrent des navigateurs "compatibles Java", comme Netscape Navigator ou Internet Explorer de Microsoft (voir Figure 1.1). Outre ses caractéristiques Web de base, un navigateur "compatible Java" se distingue par sa capacité à télécharger et exécuter des applets sur le système de l'utilisateur. Les applets apparaissent dans la page Web comme des images. Mais, à l'inverse des images, les applets sont interactives et dynamiques et servent à créer des animations, des figures, des zones de dialogue avec l'utilisateur, des jeux ou tout autre effet interactif, dans les textes et les graphiques des pages Web.

La Figure 1.1 montre une applet exécutée dans Netscape 3.0. (Cette applet, disponible à l'adresse <http://prominence.com/java/poetry/>, génère des "aimants" électroniques que l'on déplace pour écrire des poèmes ou des messages.)



**Figure 1.1**

*Netscape exécutant  
une applet Java.*



## Lexique

*Les applets sont des programmes chargés depuis le World Wide Web par un navigateur Web, et exécutés à l'intérieur d'une page Web HTML. L'exécution d'applets exige l'utilisation d'un navigateur compatible Java, tel que Netscape Navigator ou Internet Explorer de Microsoft.*

Une applet doit être écrite dans le langage Java, compilée avec un compilateur Java et référencée dans le code HTML. L'installation de fichiers HTML et du fichier d'une applet Java sur un site Web est identique à celle d'un fichier image ou HTML ordinaire. Ainsi, lorsqu'un utilisateur utilise un navigateur compatible Java, il visualise la page et son applet. Le navigateur télécharge la page sur le système local et exécute l'applet. L'utilisateur peut ainsi dialoguer avec l'applet (les utilisateurs n'ayant pas de programme de navigation compatible avec Java peuvent voir du texte, une image statique, ou rien du tout). Les applets, les programmes de navigation et le World Wide Web travaillent ensemble. Cet aspect est traité plus loin.

Java permet de créer des applets et bien d'autres choses. C'est un langage de programmation à part entière. De ce fait, il rend les mêmes services et résout les mêmes problèmes que d'autres langages de programmation comme C ou C++.

# Java hier, aujourd'hui et demain

Java a été développé par Sun Microsystems en 1991, dans le cadre d'un projet de développement de logiciel pour des appareils électroniques de grande consommation : télévisions, magnétoscopes, grille-pain, etc. Les objectifs de l'époque étaient la concision, la rapidité, l'efficacité et la portabilité. Java est devenu ainsi un langage idéal pour la distribution d'exécutables *via* le World Wide Web. C'est aussi un langage de programmation à vocation générale, puisque les programmes sont généralement destinés à de multiples plates-formes.

Le langage Java, utilisé dans divers projets de Sun (sous la dénomination Oak), avait peu d'intérêt économique avant d'être associé à HotJava. HotJava a été conçu en 1994 à titre expérimental pour servir de moteur de téléchargement et d'exécution des applets, mais aussi pour être un exemple d'application complexe développée avec Java. Malgré l'attention accordée à HotJava par la communauté du Web, ce n'est que plus tard, lorsque Netscape a incorporé dans son propre navigateur l'aptitude à exécuter des applets, que Java a réellement pris son essor et commencé à provoquer, tant au sein du World Wide Web qu'à l'extérieur, l'enthousiasme qu'il suscite aujourd'hui. Ce succès d'estime est tel que, à l'intérieur du groupe Sun, le groupe Java est devenu une filiale à part entière appelée Javasoft.

Les versions de Java lui-même, ou, comme on l'appelle communément, l'API Java, correspondent à des versions du kit du développeur Java de Sun, ou JDK (Java Developer's Kit). Lors de la rédaction de cet ouvrage, deux versions du JDK retenaient l'attention. Le JDK 1.02, plus ancien, est très largement utilisé et supporté par les navigateurs les plus courants (Netscape 3.0 et Microsoft Internet Explorer 3.0), ainsi que par des outils de développement tels que Café de Symantec et Visual J++ de Microsoft. Java 1.1, publié récemment, comporte de nombreuses fonctions complémentaires, mais, du fait de sa nouveauté, il n'est pas encore aussi largement supporté que la version 1.02. Cet ouvrage décrit aussi bien Java 1.02 que Java 1.1, et signale les différences. Les programmes Java 1.02 marchent sans problème sous Java 1.1, mais la réciproque n'est pas vraie ; les programmes de la version 1.1 ne marchent pas dans un environnement 1.02.

Pour programmer en Java, vous avez besoin d'un environnement de développement Java adapté à votre plate-forme. Le JDK de Sun répond à cette exigence. Il comporte des outils de compilation et de test des applets et des applications Java. En outre, plusieurs excellents outils de développement Java ont été créés, parmi lesquels Java Workshop, le propre atelier de développement de Sun, Café de Symantec, Visual J++ de Microsoft (qui est bien un outil Java, en dépit de son nom) et Roaster de Natural Intelligence. De nouveaux outils de développement apparaissent régulièrement. Si vous codez des programmes pour Java 1.1, assurez-vous que votre environnement de développement supporte le JDK 1.1 ; à la date de l'écriture du livre, le seul environnement de développement pour Java 1.1 est le JDK de Sun. Si vous n'êtes pas particulièrement intéressé par les fonctions spécifiques du 1.1, vous pouvez utiliser au choix un environnement 1.02 ou 1.1.

Pour faire tourner et visualiser des applets Java, il vous faut un navigateur ou un autre outil compatible Java. Comme nous l'avons vu, les versions récentes de Netscape Navigator et d'Internet Explorer (3.0) peuvent l'une et l'autre faire tourner des applets Java. (Notez qu'il existe de grandes différences dans la façon dont Java est supporté par les différents navigateurs.) Si vous n'avez pas de navigateur compatible Java, vous pouvez utiliser l'un des nombreux outils capables de faire marcher des applets. Le JDK en comporte un ; il s'appelle Appletviewer. Comme dans le cas des environnements de développement, méfiez-vous des différences entre Java 1.02 et 1.1 ; à l'heure où nous écrivons, il n'y a pas de navigateur supportant Java 1.1, et la seule manière de tester des applets écrites au moyen du JDK 1.1 est de se servir de l'Appletviewer de Sun. Netscape 3.0 et Internet Explorer ne supportent PAS les applets de version 1.1.

## Info

*Pour l'heure, Netscape Communicator 4 est en version bêta (sortie prévue août 1997). Il supporte certaines fonctions du JDK 1.1, mais pas toutes. Il les supportera peut-être toutes lors de la diffusion de la version finale.*

Quel est l'avenir de Java ? Java 1.1 comporte d'importantes modifications, qui le font passer du stade d'un langage intéressant et amusant pour écrire des applets, à celui d'un langage plus puissant et plus souple, pour la création d'applications de grande envergure. En avançant dans l'étude de cet ouvrage, vous découvrirez la plupart de ces modifications et leur intérêt pour les développements en Java. Les versions futures de Java comporteront des fonctions encore plus puissantes, adaptées à différents types d'applications. Parmi les nouveautés figureront des ensembles de classes Java et d'API pour le multimédia, les applications graphiques et les mises en page sophistiquées, pour des fonctions avancées de sécurité, ainsi que pour une souplesse accrue lors du développement d'interfaces utilisateur graphiques.

En outre, Java s'étend, au-delà du rôle d'un simple langage de programmation, vers des domaines différents. Sun développe des microprocesseurs Java, en vue d'une exécution accélérée des applications Java. Le Network computer de Sun utilise un système d'exploitation Java, conçu pour tourner sur des systèmes petits et limités. La vision de Sun, en ce qui concerne l'avenir, est celle d'un ordinateur entièrement basé sur Java, qu'il s'agisse de sa conception interne ou de ses fonctionnalités externes.

## Pourquoi apprendre Java ?

Les applets sont écrites dans ce langage, d'où l'intérêt de l'apprendre aujourd'hui. Ce n'est pas le seul, Java ayant des avantages significatifs sur les autres langages et environnements de programmation. C'est en fait un langage adapté à toutes les tâches de programmation. Cette section décrit quelques-uns de ces avantages.

## Java ne dépend d'aucune plate-forme

Son indépendance vis-à-vis des plates-formes matérielles est son principal avantage. Il est donc tout indiqué pour les applications qui doivent tourner sur différentes plates-formes. Cette caractéristique est vitale pour tout logiciel destiné au World Wide Web. L'indépendance, dans le cas de Java, se situe à deux niveaux : source et binaire.

### Lexique

*Les programmes indépendants des plates-formes peuvent tourner sur n'importe quel système. Les programmes Java peuvent tourner sur tout système doté d'une machine virtuelle Java.*

Au niveau source, les types de données primaires de Java ont la même taille, quelle que soit la plate-forme de développement. Les bibliothèques de classes standard de Java facilitent l'écriture du code, qui est ensuite porté de plate-forme en plate-forme, sans adaptation.

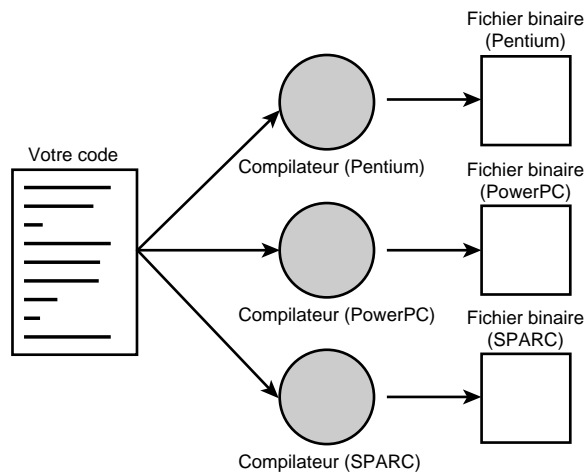
Lorsque vous écrivez un programme en Java, vous n'avez pas à tenir compte des fonctions de votre système d'exploitation pour l'exécution des tâches de base. L'indépendance des plates-formes au niveau source signifie que vous pouvez déplacer des sources d'un système à un autre, puis les compiler et les faire marcher sans problème sur cet autre système.

Dans le cas de Java, l'indépendance des plates-formes ne se limite pas au source. Les fichiers binaires Java compilés sont également indépendants des plates-formes : ils peuvent tourner directement sur toute plate-forme pourvue d'une machine virtuelle Java, de sorte qu'il n'est pas nécessaire de recompiler le source.

Généralement, la compilation d'un programme écrit en langage C (ou dans la majorité des langages) génère du code machine ou des instructions destinées au processeur. Ces instructions sont spécifiques au processeur de l'ordinateur. Ainsi, l'exécutable obtenu après compilation du code sur un système Pentium ne tourne que sur un système Pentium. Pour l'utiliser sur un autre système, il faut recompiler le code source avec un compilateur adapté. La conséquence de ceci est illustrée par la Figure 1.2 : un exécutable par système.

**Figure 1.2**

*Programmes classiques compilés.*

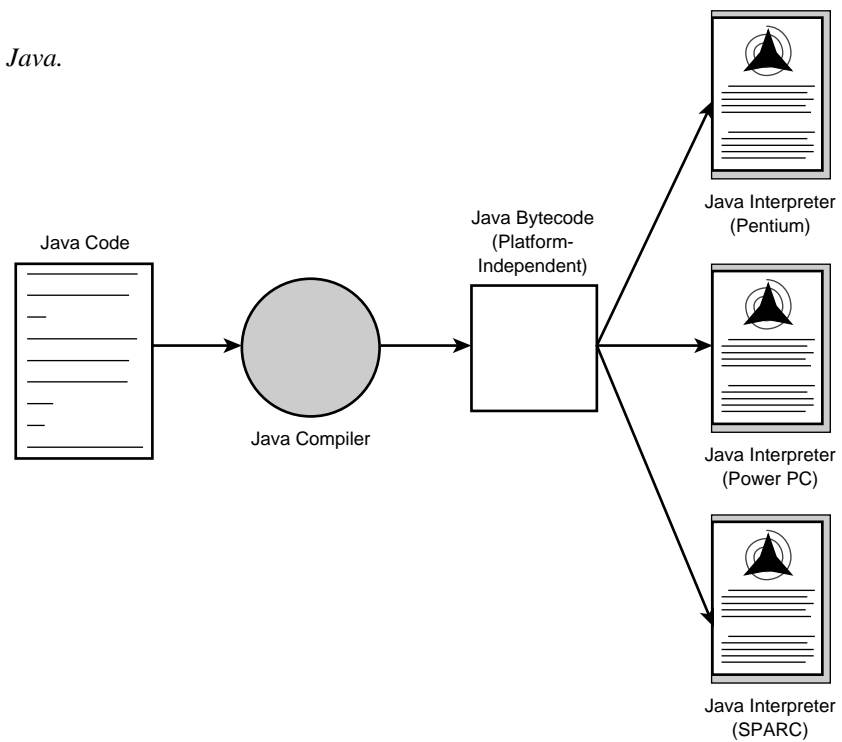


Le mécanisme de compilation est différent pour du code Java. L'environnement de développement Java comprend un compilateur et un interpréteur. Le compilateur ne génère pas du code machine mais du pseudo-code.

Pour exécuter un programme Java, vous utilisez un interpréteur de pseudo-code, qui lit le pseudo-code et exécute le programme Java (voir Figure 1.3). Le pseudo-code comporte des instructions qui ressemblent à du langage machine, mais qui ne sont spécifiques à aucun processeur particulier. Un interpréteur de pseudo-code spécifique à la plate-forme exécute le pseudo-code Java. L'interpréteur de pseudo-code Java est souvent appelé machine virtuelle Java ou interpréteur run-time Java.

**Figure 1.3**

*Programmes Java.*



Où se procure-t-on l'interpréteur de pseudo-code ? Dans le cas des applets, l'interpréteur de pseudo-code est inclus dans tout navigateur compatible Java, de sorte que vous n'avez pas à vous en préoccuper ; les applets Java marchent automatiquement. Dans le cas d'applications Java plus générales, vous devez disposer sur votre système d'un interpréteur pour exécuter les programmes de ces applications. Actuellement, l'interpréteur peut être fourni avec votre environnement de développement. Si vous achetez un programme Java, l'interpréteur est fourni avec.

Dans l'avenir, cependant, l'interpréteur de pseudo-code Java devrait être livré avec tout nouveau système d'exploitation. Ainsi, lors de l'achat d'une machine Windows, Java serait inclus gratuitement.

L'ajout d'un interpréteur de pseudo-code peut passer pour une complication superflue. Cependant, les programmes Java sous forme de pseudo-code s'exécutent sur toutes les plates-formes et tous les systèmes d'exploitation pour lesquels un interpréteur Java est disponible. Cette caractéristique est essentielle pour permettre aux applets de tourner. En effet, le World Wide Web n'est lié à aucune plate-forme. Comme la lecture d'un fichier HTML, l'exécution des applets est donc possible sur les plates-formes possédant un programme de navigation compatible Java.

L'inconvénient du pseudo-code est sa faible vitesse d'exécution. Les programmes spécifiques à un système étant directement liés au matériel pour lequel ils sont compilés, ils tournent plus vite qu'un pseudo-code Java traité par l'interpréteur. Ce n'est pas un problème pour la majorité des programmes Java. Cependant, s'ils doivent avoir une vitesse d'exécution supérieure à celle permise par l'interpréteur, plusieurs solutions existent, notamment l'introduction de code natif dans les programmes Java, ou l'utilisation d'outils spéciaux (appelés compilateurs juste-à-temps) pour convertir le pseudo-code Java en code natif et en accélérer l'exécution. Malheureusement, la portabilité du pseudo-code Java est alors perdue. Ces deux mécanismes sont présentés dans le Chapitre 20.

## Java est orienté objet

Pour certains, la programmation orientée objet (POO) s'assimile à l'organisation des programmes, ce qui est réalisable avec tous les langages. Néanmoins, travailler avec un langage et un environnement de programmation orientés objet permet de tirer le meilleur parti de cette méthodologie, qui permet de créer des programmes souples et modulaires et d'en réutiliser le code.

Les concepts de Java sont hérités de C++ dont il est issu, mais aussi d'autres langages orientés objet. Comme la plupart de ces langages, Java possède également des bibliothèques de classes, qui fournissent les types de données de base, les possibilités d'entrées/sorties du système et d'autres fonctions utilitaires. Ces bibliothèques de base font partie de JDK, qui comprend également des classes pour la gestion de réseau, des protocoles Internet communs et des outils d'interface utilisateur. Ces bibliothèques sont écrites en langage Java, et sont donc portables sur toutes les plates-formes.

Le Chapitre 2 apporte un complément d'informations sur la programmation orientée objet et Java.

## Java est facile à apprendre

Outre sa portabilité et son orientation objet, Java a été conçu pour être concis, simple, facile à écrire, à compiler, à corriger, mais surtout facile à apprendre. Ce langage comporte peu de mots, ce qui augmente sa fiabilité. Ainsi, le risque de tomber sur un problème épineux de programmation est réduit. Java reste souple et puissant, malgré sa taille et sa conception simple.

Java est issu des langages C et C++. C'est pourquoi sa syntaxe et sa structure orientée objet ressemblent à celles de C++. La connaissance de C++ facilite l'apprentissage de Java, car les bases sont déjà acquises.

La complexité des langages C et C++ a été en grande partie exclue de Java. Cette simplification a été obtenue sans réduire vraiment sa puissance. Par exemple, il ne possède ni pointeur, ni arithmétique sur les pointeurs. Ses chaînes et ses tableaux sont de véritables objets. Sa gestion de la mémoire est automatique. Ces caractéristiques perturbent les programmeurs expérimentés, mais favorisent les débutants ou les programmeurs habitués à d'autres langages.

Bien que la conception de Java le rende plus facile à apprendre que d'autres langages de programmation, l'utilisation d'un langage de programmation demeure beaucoup plus compliquée que l'utilisation, par exemple, de HTML. Si vous n'avez aucune expérience des langages de programmation, vous pouvez éprouver des difficultés à comprendre Java. Mais ne vous découragez pas ! L'apprentissage de la programmation est une acquisition utile à l'égard du Web et des ordinateurs en général, et commencer par ce langage présente un très grand intérêt.

## Programmer avec Java

Dans la deuxième moitié de ce chapitre, vous attaquerez les aspects les plus simples de Java en créant deux programmes Java : une application Java indépendante, et une applet, que vous visualiserez au moyen d'un navigateur compatible Java. Bien que ces deux programmes soient extrêmement simples, ils vous donneront une idée de l'aspect d'un programme Java et de la façon dont il est compilé et exécuté.

### Acquisition d'un environnement de développement Java

Pour écrire des programmes Java, vous avez évidemment besoin d'un environnement de développement Java. (Les navigateurs tels que celui de Netscape vous permettent d'exécuter des applets Java, mais ils ne vous permettent pas d'en écrire. Pour cela, il vous faut un outil séparé.) Le JDK de Sun, disponible par téléchargement depuis le site Web de JavaSoft (<http://www.javasoft.com>) et inclus sur le CD joint à cet ouvrage, peut être utilisé dans ce but. Cependant, malgré l'engouement qu'il suscite, ce n'est pas l'outil de développement le plus facile à utiliser. Si vous avez l'habitude d'un environnement de développement comportant une interface utilisateur graphique, un éditeur et un débogueur intégrés, vous jugerez sans doute primitive l'interface du JDK, à base de lignes de commande. Heureusement, le JDK n'est pas le seul outil disponible.

Les environnements de développement intégrés disponibles comprennent Java Workshop de Sun pour Solaris, Windows NT et Windows 95 (des informations sont proposées à <http://www.sun.com/developer-products/java/>) ; Café de Symantec pour Windows 95, Windows NT

et Macintosh (<http://cafe.symantec.com/>) ; Visual J++ de Microsoft pour Windows 95 et Windows NT (<http://www.microsoft.com/visualj/>) ; et Roaster de Natural Intelligence (<http://www.natural.com/pages/products/roaster/index.html>). Tous sont des produits commerciaux, mais vous devriez pouvoir télécharger des versions d'essai ou des versions limitées de ces programmes pour les tester.

**Info**

*Les environnements de développement graphiques sont d'un usage beaucoup plus facile que le JDK standard. Si vos moyens et le temps dont vous disposez le permettent, procurez-vous l'un de ces outils, qui rendra votre activité de développement Java beaucoup plus agréable.*

## Installation du JDK et des fichiers d'exemples

Le JDK 1.1 de Sun pour Solaris et Windows fait partie des programmes du CD-ROM associé au livre. Si vous travaillez sur un Macintosh, le JDK est également disponible sur le CD, mais il est en version 1.02 (il n'existe pas encore de version Mac du JDK 1.1). Le CD-ROM comporte également tous les exemples de code du livre, ce qui vous évitera d'avoir à les saisir. Pour installer le JDK, ou les fichiers d'exemples, ou les deux, appliquez l'une des procédures décrites ci-après :

**Info**

*Si vous ne disposez pas d'un lecteur de CD-ROM, vous pouvez accéder à ces fichiers par l'intermédiaire du World Wide Web. Vous pouvez télécharger le JDK lui-même à partir de <http://www.javasoft.com/products/JDK/1.1/index.html>, et l'installer à partir des instructions des pages ci-après. Si vous utilisez un Mac et si vous souhaitez utiliser les fonctions 1.1 du JDK, consultez périodiquement cette page pour voir si le JDK 1.1 est désormais diffusé.*

*Les fichiers d'exemples de ce livre sont disponibles sur le site Web de l'ouvrage : <http://www.1ne.com/Web/Java1.1/>.*

*Si vous téléchargez le JDK et les fichiers sources, au lieu de les lire depuis le CD-ROM, lisez d'abord la section "Configuration du JDK" pour vous assurer que vos réglages sont corrects.*

Le JDK de Sun tourne sous Windows 95 et Windows NT, mais pas sous Windows 3.x.

Pour installer le JDK ou les fichiers d'exemples sous Windows, exécutez le programme Setup du CD-ROM (lancement automatique par un double-clic sur l'icône du CD). Par défaut, le logiciel est installé sur C:\JKD1.1 ; vous pouvez l'installer à n'importe quel autre emplacement de votre disque dur. Des options permettent d'installer le JDK, les fichiers d'exemples et divers autres fichiers supplémentaires.

Après installation du JDK, repérez le fichier appelé `classes.zip` dans le dossier `JDK\lib`. Il n'est pas nécessaire de décompresser ce fichier.

Le JDK de Sun pour Macintosh tourne sous le système 7 (MacOS) pour 68KB ou Power Mac. Pour installer le JDK ou les fichiers d'exemples sur Macintosh, décompressez le fichier original `bin` ou `hqx` sur votre disque dur, et double-cliquez sur le programme d'installation. Par défaut, le package sera installé dans le dossier Java sur le disque dur ; vous pouvez l'installer n'importe



où ailleurs si vous le souhaitez. Des options permettent d'installer le JDK, les fichiers d'exemples et divers autres fichiers supplémentaires.

Le JDK de Sun pour Solaris tourne sous Solaris 2.3, 2.4 et 2.5, ainsi que sur la version x86 de Solaris.

Le CD-ROM de ce livre place le JDK dans le dossier `JDK/INTESOL/jdk1_1-beta4-solaris-x86.sh` ou dans le dossier `JDK/SPARCSOL/jdk1_1-solaris-sparc.sh`, selon le système dont vous disposez. Vous trouverez des détails relatifs à votre version particulière de Solaris en consultant le site Web JavaSoft à :

<http://www.javasoft.com/products/jdk/1.1/installation-solaris2.html>

Les fichiers d'exemples figurent également sur le CD-ROM dans `source.source.tar`. Créez un dossier où résideront vos fichiers d'exemples (par exemple, un dossier appelé `Javaexemples` dans votre répertoire personnel), copiez le fichier `source.tar` à cet endroit, puis utilisez la commande `tar` pour l'extraire, comme suit :

- `mkdir~/javaexemples`
- `cp/cdrom/source/source.tar`

## Configuration du JDK

Après l'installation du JDK au moyen des programmes de mise en place figurant dans le CD-ROM, il devrait normalement être configuré convenablement. Cependant, la plupart des problèmes rencontrés avec Java étant dus à des erreurs de configuration, il est souhaitable de vérifier qu'elle est correcte. Si vous avez installé le JDK depuis une autre source que le CD-ROM, vous avez sans doute intérêt à lire attentivement cette section.

Le JDK exige deux modifications importantes de votre `autoexec.bat` pour tourner correctement : le répertoire `JDK1.1\bin` doit se trouver sur le chemin d'exécution, et la variable `CLASSPATH` doit avoir été initialisée.

Modifiez votre `autoexec.bat` au moyen de votre éditeur de texte habituel (le Bloc-notes convient parfaitement). Recherchez une ligne du genre :

```
PATH C:\WINDOWS;C:\WINDOWS\COMMAND;C:\DOS; ...
```

Quelque part sur cette ligne vous devez voir une entrée pour le JDK ; si vous installez le JDK depuis le CD-ROM, la ligne devrait avoir l'aspect suivant (les points indiquent qu'il peut y avoir d'autres éléments sur la ligne en question) :

```
PATH C:\WINDOWS; ... C:\JDK1.1\BIN; ...
```

Si vous ne trouvez pas de référence à `JDK\BIN` ou `JAVA\BIN` dans votre `PATH`, ajoutez-la. Ajoutez à la fin de cette ligne le nom complet du chemin conduisant à votre installation JDK, commençant par `C` et finissant par `BIN`; par exemple, `C:\JAVA\BIN` ou `C:\JDK1.1\BIN`.

La seconde addition indispensable au fichier `autoexec.bat` concerne la variable `CLASSPATH`, si elle n'y figure pas déjà. Cherchez une ligne du genre :

```
SET CLASSPATH=C\TEACHY-1\JDK1.1.\lib\classes.zip; ;
```

La variable `CLASSPATH` peut contenir d'autres indications pour Netscape ou Internet Explorer, mais ce qui vous intéresse le plus, c'est une référence au fichier `classes.zip` du JDK, et au répertoire courant (`.`). Si votre fichier `autoexec.bat` n'inclut aucun de ces deux emplacements, ajoutez au fichier une ligne contenant ces deux indications (la ligne ci-avant fera l'affaire).

Après avoir sauvegardé votre fichier `autoexec.bat`, vous devez redémarrer Windows pour rendre effectives les modifications.

Le JDK pour Macintosh ne devrait pas avoir besoin d'être configuré à la suite de l'installation. Pour configurer le JDK dans le cas de Solaris, il suffit d'ajouter le répertoire `java/bin` ou `jdk/bin` à votre chemin d'exécution.

En général, l'insertion d'une ligne telle que la suivante dans vos fichiers `.cshrc`, `.login` ou `.profile` permet d'obtenir le résultat cherché :

```
set path= (-/java/bin/ $path)
```

Cette ligne correspond au cas où le JDK (en tant que dossier `java`) est installé dans votre répertoire personnel ; si vous l'avez installé ailleurs, vous devez substituer le nom de chemin correspondant.

Assurez-vous que vous utilisez la commande `source` avec le nom du fichier adéquat, pour que le changement prenne effet (sinon, faites un `log out` suivi d'un `log in`).

```
source ~/.login
```

## Création d'une application Java

Commençons par une application Java simple : le programme classique Hello World par lequel débutent tous les livres de programmation.

Les applications Java sont différentes des applets. Les applets sont des programmes Java téléchargés par l'intermédiaire du World Wide Web, et exécutés sur la machine de l'utilisateur par un navigateur Web compatible Java.

Les applications Java, en revanche, sont des programmes plus généraux écrits dans le langage Java. Elles n'ont pas besoin d'un navigateur pour tourner ; Java peut être utilisé pour créer toutes sortes d'applications, qui pourraient avoir été créées avec un langage de programmation classique.

Un programme Java peut être une applet ou une application, ou les deux, selon la manière dont le programme est écrit et selon ses fonctions. Au cours de cette première partie, pour apprendre le langage Java, vous écrirez surtout des applications ; puis, dans la seconde partie de l'ouvrage, vous utiliserez ce que vous avez appris pour écrire des applets. Il est préférable de commencer

par ce stade élémentaire, car tout ce que vous apprendrez en créant de simples applications Java vous servira pour créer des applets.

## Création du fichier source

Dans tous les langages de programmation, les fichiers source sont créés à l'aide d'un éditeur de texte. Java n'échappe pas à cette règle ; les fichiers sont sauvegardés en mode ASCII simple, sans caractère de contrôle. Les éditeurs exploitables sous UNIX sont emacs, ped, ou vi. Sous Windows, on peut choisir le Bloc-notes ou l'Editeur DOS (nous vous conseillons d'utiliser le Bloc-notes en shareware). Sur Mac, vous pouvez utiliser SimpleText (fourni avec le Mac) ou le shareware Bbedit. Si vous utilisez un environnement de développement comme Café, vous utiliserez l'éditeur de texte qui lui est associé.

### Info

*Si vous faites votre développement Java sous Windows, assurez-vous que Windows comprend l'extension de fichier .java avant de vous lancer ; autrement, votre éditeur de texte attachera systématiquement une extension .txt à tous vos fichiers. La méthode la plus simple pour obtenir le résultat voulu est de taper le nom de fichier entre guillemets ("HelloWorld.java"). Cependant, il vaut mieux aller à une fenêtre de l'Explorateur Windows, sélectionner Affichage / Options / Types de fichiers, choisir Nouveau type, et inscrire respectivement fichier source Java et .java dans les zones de texte Description du type et Extension associée.*

Après avoir sélectionné l'éditeur, saisissez le programme Java présenté dans le Listing 1.1. N'oubliez ni parenthèse, ni virgule ou guillemet, et utilisez la même casse (majuscules et minuscules) que sur le listing.

### Info

*Le code de ces exemples figure également sur le CD-ROM. Toutefois, si vous tapez vous-même ces premiers exemples, qui sont très courts, vous aurez une idée plus précise de ce qu'est le code Java.*

### Listing 1.1 Première application Java

```
1: class HelloWorld {  
2:     public static void main (String args[]) {  
3:         System.out.println("Hello World!");  
4:     }  
5: }
```

Après avoir saisi le programme, sauvegardez le fichier sur votre disque sous le nom HelloWorld.java. Ce nom est très important. Les fichiers source Java doivent avoir le même nom que la classe qu'ils définissent (y compris les mêmes lettres majuscules ou minuscules), et ils doivent avoir l'extension .java. Ici, la définition de classe a le nom HelloWorld, de sorte que le nom du fichier doit être HelloWorld.java. Si vous donnez un autre nom au fichier (même quelque chose comme helloworld.java ou Helloworld.java), vous ne pourrez pas le compiler. Assurez-vous que le nom est bien HelloWorld.java.

**Attention**

*Les numéros placés en début de ligne ne font pas partie du programme. Ils référencent les lignes et sont repris dans le texte lors de l'analyse d'un programme. Ils ne doivent donc pas être saisis.*

Après avoir saisi le programme, sauvegardez le fichier sur votre disque sous le nom `HelloWorld.java`. Ce nom est très important. Les fichiers source Java doivent avoir le même nom que la classe qu'ils définissent (y compris les mêmes lettres majuscules ou minuscules), et ils doivent avoir l'extension `.java`. Ici, la définition de classe a le nom `HelloWorld`, de sorte que le nom du fichier doit être `HelloWorld.java`. Si vous donnez un autre nom au fichier (même quelque chose comme `helloworld.java` ou `Helloworld.java`), vous ne pourrez pas le compiler. Assurez-vous que le nom est bien `HelloWorld.java`.

Vous pouvez sauvegarder vos fichiers Java n'importe où sur le disque, mais il vaut mieux les conserver tous dans un même dossier. Les fichiers d'exemples de ce chapitre ont été placés dans un dossier appelé `TYJtests` (abréviation de `Teach Yourself Java tests`).

## Compilation et exécution du fichier source

Vous êtes maintenant prêts à compiler le fichier. Si vous utilisez le JDK, conformez-vous aux instructions des pages suivantes. Si vous utilisez un environnement de développement graphique, vous disposerez probablement d'un bouton ou d'une option pour lancer la compilation (vérifiez-le dans la documentation fournie avec le programme).

Le compilateur associé au JDK est piloté par des lignes de commande. Pour utiliser ce compilateur, vous devez d'abord démarrer un shell DOS. Sous Windows 95, ce shell DOS fait partie du menu Programmes (il s'appelle `Commandes MS-DOS`).

Une fois dans le DOS, changez les répertoires pour aller à l'endroit où vous avez sauvegardé votre fichier `HelloWorld.java`. Comme nous avons placé le nôtre dans le dossier `TYJtests`, nous utilisons la commande :

```
CD C:\TYJtests
```

Après avoir commuté sur le répertoire convenable, utilisez la commande `javac` comme suit, en reprenant le nom du fichier que vous avez sauvegardé sous Windows (`javac` veut dire "Java compiler"). Ici encore, le respect de la casse est requis :

```
javac HelloWorld.java
```

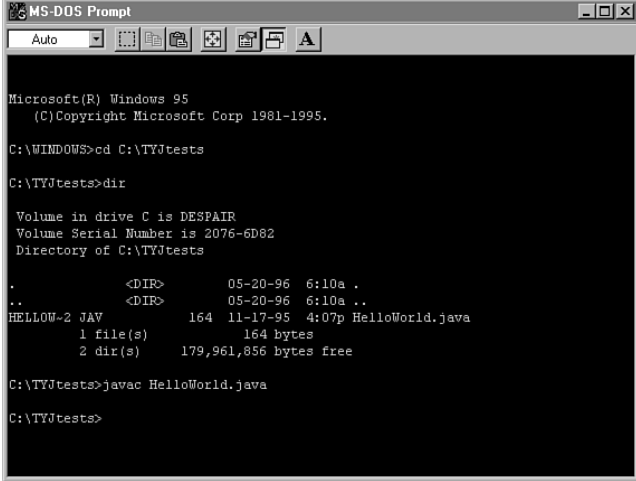
**Info**

*Nous avons insisté sur la nécessité d'utiliser le nom de fichier original, parce qu'une fois à l'intérieur du shell DOS, vous vous apercevrez que les noms de fichiers longs ont été tronqués en noms à 8.3 positions conformes à l'ancien style, et qu'en fait, `HelloWorld.java` apparaît sous la forme `HELLO-1.jav`. Ne vous en inquiétez pas ; ceci est dû simplement à la manière dont Windows 95 gère les noms de fichiers longs. Ignorez le fait que le fichier semble s'appeler `HELLO-1.jav` et tenez-vous en au nom de fichier utilisé lors de la sauvegarde.*

La Figure 1.4 montre ce que nous avons fait à l'intérieur du shell DOS, pour vous permettre de vérifier que vous êtes sur la bonne voie.

**Figure 1.4**

*Compilation Java dans le shell DOS.*



```
MS-DOS Prompt
Auto
Microsoft(R) Windows 95
(C)Copyright Microsoft Corp 1981-1995.
C:\WINDOWS>cd C:\TYJtests
C:\TYJtests>dir
Volume in drive C is DESPAIR
Volume Serial Number is 2076-6D82
Directory of C:\TYJtests

.<DIP>          05-20-96  6:10a  .
..<DIP>         05-20-96  6:10a  ..
HELLOW-2 JAV   164  11-17-95  4:07p  HelloWorld.java
 1 file(s)                164 bytes
 2 dir(s)                 179,961,856 bytes free

C:\TYJtests>javac HelloWorld.java
C:\TYJtests>
```

Si tout se passe bien, vous obtenez un fichier appelé `HelloWorld.class` (du moins, c'est son appellation si vous l'examinez à l'extérieur du shell DOS ; à l'intérieur du DOS, il s'appelle `HELLO-1.c1a`). C'est votre fichier de pseudo-code Java. Si vous obtenez des messages d'erreur, revenez à votre fichier source d'origine et assurez-vous que vous l'avez tapé exactement de la manière présentée sur le Listing 1.1, avec les majuscules et minuscules figurant sur le modèle. Le fichier doit avoir exactement la même casse que le nom de la classe (l'un et l'autre s'écrivent `HelloWorld`).

Lorsque vous disposez d'un fichier de classe, vous pouvez vous en servir au moyen de l'interpréteur de pseudo-code Java. L'interpréteur Java s'appelle simplement `java`, et est lancé depuis le shell DOS de la même manière que `javac`. Faites marcher votre programme Hello World comme suit à partir des lignes de commandes, en respectant bien la casse indiquée (et remarquez que l'argument du programme `java` n'a pas d'extension `.class`) :

```
Java HelloWorld
```

Si votre programme a été tapé et compilé correctement, vous obtenez en retour sur votre écran la phrase `Hello World!`. La Figure 1.5 montre ce que nous avons fait.

## Attention

*Notez bien que le compilateur Java et l'interpréteur Java sont deux choses différentes. Vous utilisez le compilateur Java (`javac`) pour créer des fichiers `.class` à partir de vos fichiers source Java, et vous utilisez l'interpréteur Java (`java`) pour exécuter effectivement vos fichiers `.class`.*

Le JDK pour le Mac est fourni avec une application appelée `Compilateur Java`. Pour compiler votre fichier source Java, faites simplement un glisser-déposer du fichier sur l'icône du compilateur Java. Le programme compile votre fichier Java et, s'il n'y a pas d'erreur, crée un fichier appelé `HelloWorld.class` dans le même dossier que votre fichier source.

**Figure 1.5**

*Exécution d'une application Java dans le shell DOS.*

```

MS-DOS Prompt
Auto
. <DIR> 05-20-96 6:10a .
.. <DIR> 05-20-96 6:10a ..
HELLOW-2 JAV 164 11-17-95 4:07p HelloWorld.java
1 file(s) 164 bytes
2 dir(s) 179,961,856 bytes free

C:\TYJtests>javac HelloWorld.java

C:\TYJtests>dir

Volume in drive C is DESPAIR
Volume Serial Number is 2076-6D82
Directory of C:\TYJtests

. <DIR> 05-20-96 6:10a .
.. <DIR> 05-20-96 6:10a ..
HELLOW-1 CLA 472 05-21-96 6:01p HelloWorld.class
HELLOW-2 JAV 164 11-17-95 4:07p HelloWorld.java
2 file(s) 636 bytes
2 dir(s) 179,945,472 bytes free

C:\TYJtests>java HelloWorld
Hello World!

C:\TYJtests>

```

**Astuce**

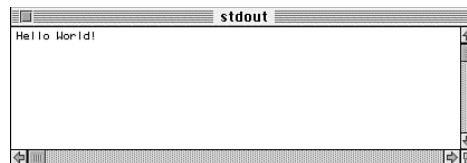
*En plaçant un alias du compilateur Java sur le bureau, vous pouvez plus facilement glisser et déposer des fichiers source Java.*

En cas d'erreur, vérifiez la frappe de votre fichier source d'origine, qui doit être exactement conforme au Listing 1.1, notamment en ce qui concerne la casse. Assurez-vous aussi que la casse du nom du fichier est bien la même que celle de la classe (c'est-à-dire HelloWorld dans l'un et l'autre cas).

Après avoir réussi à générer un fichier HelloWorld.class, faites simplement un double-clic sur ce fichier pour l'exécuter. L'application Java Runner, qui fait partie du JDK, démarre alors, et le programme vous demande les arguments en lignes de commande. Laissez cet écran vide et cliquez sur OK. Une fenêtre appelée stdout apparaît alors avec le message Hello World!. La Figure 1.6 montre cette fenêtre.

**Figure 1.6**

*Exécution d'applications Java sur Mac au moyen de Java Runner.*



C'est tout ! Souvenez-vous que vous vous servez de l'application Compilateur Java pour compiler vos fichiers .java et en faire des fichiers .class, eux-mêmes exécutés en vous servant de Java Runner.

Pour compiler un fichier source Java sous Solaris, vous utilisez le compilateur Java en lignes de commande qui est fourni avec le JDK. Au moyen d'une ligne de commande UNIX cd, passez

sous le répertoire qui contient votre fichier source Java. Le nôtre étant placé dans le répertoire `TYJtests`, il est accessible par la commande :

```
cd~/TYJtests
```

Une fois dans le répertoire adéquat, utilisez la commande `javac` en lui adjoignant le nom du fichier, comme suit :

```
javac HelloWorld.java
```

Si tout se passe normalement, vous obtenez un fichier appelé `HelloWorld.class`, dans le même répertoire que votre fichier source. C'est votre fichier de pseudo-code. En cas d'erreur, vérifiez votre fichier source d'origine et assurez-vous que vous l'avez tapé exactement comme sur le Listing 1.1, en utilisant la même casse. Vérifiez également que la casse pour les lettres du nom de fichier est la même que pour celles du nom de la classe, c'est-à-dire `HelloWorld`.

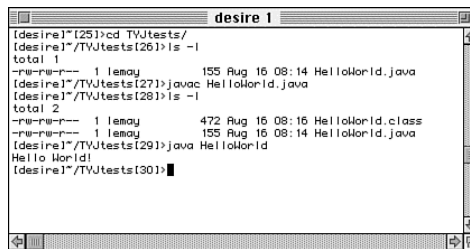
Disposant d'un fichier `.class`, vous pouvez l'exécuter au moyen de l'interpréteur de pseudo-code Java. L'interpréteur Java s'appelle simplement `java`, et vous pouvez le lancer en lignes de commande, tout comme `javac` (notez que l'argument du programme Java ne comporte pas d'extension `.class`) :

```
java HelloWorld
```

Si votre programme a été tapé et compilé correctement, vous devez voir en retour la phrase `Hello World!` s'afficher sur votre écran. La Figure 1.7 montre les commandes que nous avons utilisées jusqu'à ce stade (l'indication `[desire]~[1]` est le prompt utilisé sur notre système).

**Figure 1.7**

*Compilation et exécution  
d'une application Java  
sous Solaris.*



## Info

*Notez bien que le compilateur et l'interpréteur Java sont deux choses différentes. Vous utilisez le compilateur Java (`javac`) pour créer des fichiers `.class` à partir de vos fichiers source Java, et vous utilisez l'interpréteur Java (`java`) pour exécuter effectivement vos fichiers `.class`.*

## Création d'une applet Java

La création d'une applet diffère de celle d'une simple application. En effet, les applets Java tournent et sont affichées dans une page Web, parmi d'autres éléments. Elles suivent, par conséquent, des règles de comportement particulières, rendant leur création plus complexe que celle d'une application.

Par exemple, pour créer une simple applet Hello World, il ne suffit pas de pouvoir imprimer un message sous forme de jeu de caractères, il faut aussi aménager un espace pour le message sur les pages Web, utiliser des fonctions graphiques et définir des polices pour peindre le message sur l'écran.

## Info

*En réalité, on peut faire tourner une simple application Java sous forme d'applet, mais le message Hello World s'affiche alors sur une fenêtre spéciale ou sur un fichier log, selon la manière dont la sortie du navigateur a été paramétrée. Vous verrez ces points plus en détail dans la seconde partie de l'ouvrage.*

## Création du fichier source

Dans l'exemple suivant, vous allez créer l'applet Hello World, la placer dans une page Web et observer le résultat.

Comme dans le cas de l'application Hello World, vous commencez par créer le fichier source au sein d'un simple éditeur de texte. Le Listing 1.2 montre le code de cet exemple.

### Listing 1.2 Applet Hello World

```
1: import java.awt.Graphics;
2:
3: class HelloWorldApplet extends java.applet.Applet {
4:
5:     public void paint(Graphics g) {
6:         g.drawString("Hello World!", 5, 25);
7:     }
8: }
```

Sauvegardez ce fichier de la même manière que vous avez sauvegardé l'application Hello World, en utilisant comme nom de fichier un nom identique à celui de la classe. En l'occurrence, le nom de la classe est `HelloWorldApplet`, vous devez donc sauvegarder le fichier sous le nom `HelloWorldApplet.java`. Nous avons placé le fichier dans un dossier appelé `TYJch01`, mais vous pouvez utiliser tout autre dossier de votre choix.

## Compilation du fichier source

L'étape suivante consiste à compiler le fichier de l'applet Java. Bien qu'il s'agisse d'une applet, vous compilez le fichier exactement de la même manière que celui d'une application, en utilisant l'une des procédures suivantes :

Depuis un shell DOS, faites un `cd` vers le répertoire contenant le fichier source de votre applet, et utilisez la commande `javac` pour compiler ce fichier (attention aux majuscules et aux minuscules) :

```
javac HelloWorldApplet.java
```



Faites glisser le fichier `HelloWorldApplet.java` sur l'icône du compilateur Java.

Dans un système à ligne de commande, faites un `cd` vers le répertoire contenant le fichier source de l'applet, et utilisez la commande `javac` pour le compiler :

```
javac HelloWorldApplet.java
```

## Inclusion de l'applet dans une page Web

Si vous avez tapé le fichier correctement, vous obtenez un fichier appelé `HelloWorldApplet.class`, situé dans le même dossier que le fichier source. C'est le fichier de l'applet Java ; pour faire tourner l'applet dans une page Web, vous devez, à l'intérieur du code HTML de cette page, vous référer à ce fichier class en utilisant la balise `<APPLET>`. Le Listing 1.3 montre un simple fichier HTML susceptible d'être utilisé à cette fin.

### Listing 1.3 Code HTML avec l'applet

```
1: <HTML>
2: <HEAD>
3: <TITLE>Hello to Everyone!</TITLE>
4: </HEAD><BODY>
5: <P>My Java applet says:
6: <APPLET CODE="HelloWorldApplet.class" WIDTH=150 HEIGHT=25>
7: </APPLET>
8: </BODY>
9: </HTML>
```

`<APPLET>` référence une applet dans le fichier HTML. Vous en apprendrez davantage plus loin dans ce chapitre, mais vous pouvez noter deux choses dès à présent :

- L'attribut `CODE` est utilisé pour indiquer le nom de la classe contenant l'applet, ici `HelloWorldApplet.class`.
- Les attributs `WIDTH` et `HEIGHT` indiquent la taille de l'applet sur la page. Le navigateur les utilise pour déterminer l'espace réservé à l'applet sur la page. Dans l'exemple, une boîte de 150 pixels de large sur 25 pixels de haut est créée.

Sauvegardez le fichier HTML dans le même dossier que votre fichier classe, avec un nom descriptif et une extension `.html` (par exemple, vous pourriez utiliser le même nom que pour l'applet : `HelloWorldApplet.html`).

### Info

*Comme nous l'avons déjà vu, votre éditeur de texte peut insister et adjoindre à vos fichiers HTML une extension `.txt`, tant que Windows ne reconnaît pas l'extension `.html`. Choisissez Affichage | Options | Types de fichier à partir d'une fenêtre de l'explorateur de Windows pour régler ce problème.*

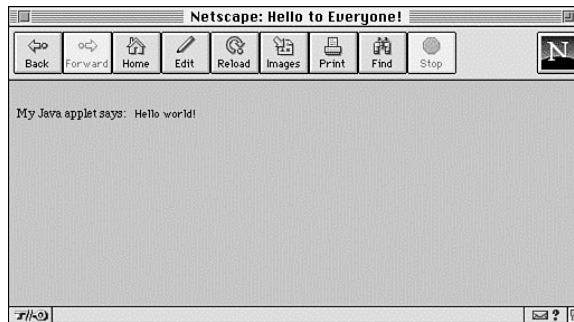
Il reste maintenant à visualiser l'applet. Pour cela, il faut l'un des éléments suivants :

- Un navigateur supportant les applets Java, comme Netscape 2.0. ou Explorer 3.0. Si vous travaillez sur un Macintosh, vous avez besoin de Netscape 3.0 ou d'une version plus récente. Si vous utilisez Windows 95 ou NT, vous avez besoin d'une version 32 bits de Netscape. Et si vous utilisez Internet Explorer, il vous faut la version 3.0 bêta 5 ou une version plus récente (ou la version définitive).
- L'application Appletviewer, incluse dans JDK. Appletviewer n'est pas un programme de navigation Web, et ne permet pas la visualisation de la page Web entière. C'est une solution acceptable, faute de mieux, pour tester et voir à quoi ressemble une applet.
- Un visualisateur d'applet ou un outil d'exécution fourni avec votre environnement de développement.

Avec un programme de navigation compatible Java comme Netscape, choisissez Fichier | Ouvrir fichier pour naviguer vers le fichier HTML contenant l'applet et la visualiser (assurez-vous d'ouvrir le fichier HTML et non le fichier classe). Dans le cas d'Internet Explorer, choisissez Fichier | Ouvrir, puis sélectionnez Parcourir pour trouver le fichier sur votre disque. Pour le moment, aucune installation sur le serveur Web n'est nécessaire : tout se passe sur le système local. Notez que l'applet Java peut mettre un certain temps à démarrer une fois la page chargée ; soyez patient. La Figure 1.8 montre le résultat du fonctionnement de l'applet sous Netscape.

**Figure 1.8**

*Applet tournant sous Netscape.*



Sans programme de navigation compatible Java, ou si vous voulez tester des applets écrites pour Java 1.1, vous pouvez utiliser l'Appletviewer du JDK.

Pour faire tourner l'Appletviewer des versions Windows ou Solaris du JDK, faites un cd vers le répertoire où se trouvent vos fichiers classe et HTML, et utilisez la commande appletviewer avec le nom du fichier HTML que vous venez de créer :

```
appletviewer HelloWorldApplet.html
```

L'appletviewer ne montrera que l'applet elle-même, sans le texte HTML correspondant.

# Résolution des problèmes

Cette section peut vous aider à surmonter d'éventuelles difficultés rencontrées lors de la mise en œuvre des exemples précédents. Voici les problèmes les plus courants, et leurs solutions.

- `Bad command or filename` ou `Command not found`

Ce type d'erreur apparaît lorsque le répertoire contenant le binaire du JDK ne se situe pas sur le chemin affiché pour l'exécution, ou si le chemin d'accès à ce répertoire est erroné. Sous Windows, vérifiez soigneusement votre fichier `autoexec.bat` ; sous UNIX, vérifiez le fichier système contenant les commandes de définition du chemin (`.cshrc`, `.login`, `.profile`, ou un fichier similaire).

- `javac: invalid argument`

Assurez-vous que le nom du fichier utilisé dans la commande `javac` est bien celui du fichier. En particulier, dans le cas du shell DOS, vous devez utiliser le nom de fichier Windows comportant une extension `.java`, et non l'équivalent DOS (`HELLO-1.java`, par exemple).

- `Warning: public class HelloWorldApplet must be defined in a file called HelloWorldApplet.java`

Cette erreur résulte le plus souvent de la non-concordance entre le nom de la classe indiqué par le fichier Java lui-même (le nom suivant le mot `class`) et le nom du fichier source Java. Les deux noms doivent être identiques, en tenant compte des minuscules et majuscules (cette erreur particulière correspond en général à des casses non identiques). Renommez soit le fichier, soit la classe, et l'erreur disparaîtra.

- Erreurs du type `Insufficient memory`

Le JDK n'utilise pas la mémoire d'une manière très efficace. Confronté à ce problème de mémoire, essayez de quitter des programmes volumineux avant de compiler ou de lancer vos programmes Java, mettez en œuvre la mémoire virtuelle, ou installez davantage de RAM.

- Autres types d'erreur de code

Si vous ne réussissez pas à compiler vos fichiers source Java pour d'autres raisons que celles mentionnées ci-avant, vérifiez encore que votre frappe correspond bien à ce qui est indiqué dans les exemples, notamment en ce qui concerne les majuscules et les minuscules. Java est sensible à la casse, ce qui veut dire que les minuscules et les majuscules sont traitées différemment. Si vous ne voyez pas d'où peut venir une erreur, essayez de comparer vos fichiers source aux fichiers du CD-ROM.

## Résumé

Ce chapitre a présenté le langage Java avec ses caractéristiques et ses objectifs. Il ressemble à C ou C++ et permet de développer de nombreux types de programmes. L'une de ses utilisations les plus courantes actuellement est le développement d'applets pour les navigateurs compatibles Java. Les applets sont des programmes Java téléchargés et exécutés au sein d'une page Web. Elles peuvent créer des animations, des jeux, des programmes interactifs et d'autres effets multimédias sur les pages Web.

Les forces de Java sont sa portabilité (source et binaire), sa conception orientée objet et sa simplicité. Ces caractéristiques rendent possible la création des applets. De plus, elles font de Java un langage excellent pour l'écriture de programmes plus généraux, s'exécutant sans navigateur compatible Java. Pour les distinguer des applets, ces programmes sont appelés applications.

A la fin du chapitre, vous avez élaboré une applet et une application. Leurs différences ont ainsi été mises en évidence. La compilation et l'écriture de programmes Java ont été traitées, ainsi que l'insertion d'applets dans une page Web. Vous avez ainsi acquis les bases nécessaires à la conception d'applications ou d'applets plus complexes.

## Questions — réponses

- Q Je connais bien HTML et peu la programmation. Puis-je, malgré cela, écrire des programmes Java ?**
- R** Sans expérience de programmation, programmer en Java va vous sembler nettement plus difficile qu'en HTML. Mais pour apprendre à programmer, Java est un bon langage. Étudiez consciencieusement tous les exemples et les exercices de ce livre pour commencer à utiliser Java.
- Q Quel est le rapport entre JavaScript et Java ?**
- R** Une erreur courante dans le monde du Web consiste à prêter à ces deux entités plus de points communs qu'elles n'en ont en réalité. Java est un langage de programmation, répondant à des besoins multiples ; vous pouvez l'utiliser pour créer des applets. JavaScript est un langage de script inventé par Netscape, qui ressemble à Java ; il permet de faire différentes choses astucieuses dans le cadre des pages Web. Ce sont des langages indépendants, conçus pour des objectifs différents. Si la programmation en JavaScript vous intéresse, procurez-vous un autre ouvrage, consacré à JavaScript.
- Q La distinction entre Java 1.02 et Java 1.1 est une cause de perplexité. Cet ouvrage est censé être consacré à Java 1.1, mais selon vous, la version 1.02 est plus répandue, et très peu de navigateurs supportent 1.1. Quelle version dois-je utiliser ?**

**R** Nous vivons une période de transition entre la version 1.02, plus répandue, et la nouvelle version. La version 1.1 sera beaucoup plus largement supportée à l'avenir, et les raisons d'hésiter entre les deux s'atténueront.

En règle générale, si vous écrivez du code conforme à l'API 1.02, il marchera n'importe où, même avec la version 1.1 du JDK. Mais si vous utilisez des fonctions de la version 1.1, vous limitez la portée de votre code, puisqu'il ne pourra être compilé et exécuté que dans un environnement 1.1. Selon l'usage que vous voulez faire de Java, il vous appartient de décider si vous voulez ou non utiliser les nouveautés 1.1. (Dans le cas des applets, il vaut probablement mieux en rester à 1.02 pour le moment. Dans le cas des applications, vous disposez d'une plus grande latitude.)

Ce problème ne doit pas vous troubler outre mesure au stade actuel de la lecture du livre. Pour la plupart, les fonctions de base de Java sont rigoureusement les mêmes entre les deux versions ; les différences ne prendront de l'importance que lorsque nous plongerons plus profond dans le JDK. Les fonctions propres à 1.1 seront signalées au fur et à mesure de leur apparition.

**Q** **Selon ce premier chapitre, un navigateur compatible Java tel que Netscape télécharge les applets pour exécution sur le système de l'utilisateur. Qu'en est-il de la sécurité ? Qu'est-ce qui empêche quelqu'un d'écrire un applet compromettant la sécurité de mon système ou y causant des dommages ?**

**R** L'équipe Java de Sun s'est penchée sur les problèmes de sécurité liés aux applets dans les programmes de navigation compatibles Java. Plusieurs contrôles existent pour que les applets ne causent aucun dégât :

- Les applets Java ne peuvent ni lire ni écrire sur le disque du système local.
- Les applets Java ne peuvent pas exécuter un programme sur le système local.
- Les applets Java ne peuvent pas se connecter à une machine du Web, à l'exception du serveur à partir duquel elles ont été téléchargées

Notez que certaines de ces restrictions peuvent ne pas jouer pour certains navigateurs, ou être levées. Attendez-vous néanmoins à ce qu'aucune des possibilités en question ne soit autorisée.

En outre, le compilateur et l'interpréteur vérifient le code source et le pseudo-code. Ils détectent ainsi les coups sournois du programmeur, par exemple, dépasser la limite de la mémoire tampon ou d'une pile.

Ces contrôles ne suppriment pas tous les problèmes de sécurité (ce qu'aucun système n'est en mesure de faire !), mais réduisent beaucoup les risques de tomber sur une applet malfaisante. Les questions de sécurité des applets sont abordées au Chapitre 8, et de façon plus détaillée au Chapitre 21.

**Q** **Ce chapitre mentionne Solaris, Windows et Macintosh. Et les autres systèmes d'exploitation ?**

**R** Si vous utilisez une autre version d'UNIX que Solaris, il y a de bonnes chances pour que le JDK ait été porté sur votre système. Voici quelques exemples :

- La version SGI du JDK peut être obtenue à [http://www.sgi.com/Products/cosmo/cosmo\\_instructions.html](http://www.sgi.com/Products/cosmo/cosmo_instructions.html).
- Des informations sur Java pour Linux sont disponibles à <http://www.blackdown.org/java-linux/>.
- IBM a porté le JDK sur OS/2 et AIX. Renseignez-vous de manière plus précise à <http://www.ncc.hurley.ibm.com/javainfo/>.
- OSF porte le JDK sur HP/UX, Unixware, Sony NEWS, et l'UNIX de Digital. Voir <http://www.osf.org/mall/web/javaport.htm>.

(Merci à Elliot Rusty Harold pour ces informations extraites des réponses aux questions fréquemment posées (FAQ) à <http://www.sunsite.unc.edu/javafaq/javafaq/html>.)

**Q** **J'utilise le bloc-notes sur Windows pour préparer mes fichiers Java. Le programme insiste pour ajouter une extension .txt à tous mes fichiers, quelle que soit la façon dont je les nomme (de sorte que j'obtiens toujours des fichiers tels que HelloWorld.java.txt). Sauf à les renommer avant chaque compilation, comment puis-je régler ce problème ?**

**R** La solution consistant à renommer les fichiers représente une contrainte désagréable, notamment si vous avez de nombreux fichiers. La difficulté provient de ce que Windows ne comprend pas l'extension .java (l'extension .html du HTML soulève les mêmes difficultés).

Pour régler cette question, allez dans une fenêtre de l'explorateur Windows, choisissez Affichage | Options | Types de fichiers. Dans la boîte de dialogue qui apparaît alors, choisissez Nouveau type. Entrez Fichiers source Java dans la zone de texte Description du type, et .java dans la zone de texte Extension associée. Puis, cliquez sur OK. Faites de même pour les fichiers HTML si nécessaire, et cliquez de nouveau sur OK. Vous devez pouvoir utiliser désormais le bloc-notes (ou tout autre éditeur de texte) pour créer et sauvegarder des fichiers Java et HTML.

Une méthode plus simple, mais qu'il faut se rappeler à chaque fois, consiste à sauvegarder les fichiers en plaçant des doubles guillemets de part et d'autre du nom de fichier ; par exemple, "HelloWorld.java".

**Q** **Où puis-je apprendre davantage de choses sur Java et trouver des applets et des applications pour jouer avec ?**

**R** Vous pouvez déjà finir de lire ce livre ! Voici d'autres endroits où vous trouverez des informations sur Java et sur les applets Java :

- La page d'accueil Java, à <http://www.java.sun.com/> est la source officielle des informations sur Java, y compris les informations sur le JDK, sur la prochaine diffusion 1.1,

et sur des outils de développement tels que Java Workshop. Vous y trouverez également une documentation bien étoffée.

- Gamelan, à <http://www.gamelan.com/>, est une source d'informations sur les applets et sur Java, organisée par catégories. Pour jouer avec des applets et des applications, c'est ici qu'il convient de vous adresser.
- Pour toute discussion sur Java, consultez le newsgroup `comp.lang.java`, et notamment `comp.lang.java.programmer`, `comp.lang.java.tech`, `comp.lang.java.advocacy`, etc. (Pour accéder à ces newsgroups, vous avez besoin d'un lecteur de news Usenet.)

2

# Programmation orientée objet et Java



LE PR  GRAMMEUR



La programmation orientée objet (POO) est l'une des plus importantes innovations de ces dernières années en matière de programmation, et son ampleur pourrait vous faire redouter de mettre des années pour apprendre ses méthodologies et profiter de ses avantages par rapport aux méthodes traditionnelles. En fait, les méthodes de la POO répondent toutes à une organisation des programmes proche de la réalité.

Ce chapitre traite des concepts de programmation orientée objet de Java et de leur mise en œuvre. Il aborde aussi la structuration des programmes :

- La définition des classes, des objets et de leurs relations
- Les parties essentielles d'une classe ou d'un objet : les comportements et les attributs
- L'héritage des classes et son influence sur la conception des programmes
- Quelques informations sur les packages et les interfaces

## Penser en termes d'objets : l'analogie

Les Lego ont des similitudes avec la POO. Ce sont de petits blocs de construction en plastique, de tailles et de couleurs variables, qui s'encastrent les uns dans les autres. De leur assemblage naissent différents objets plus volumineux. Plusieurs pièces (roues, moteur, charnière) permettent de créer des châteaux, des automobiles, des robots géants, etc. L'assemblage des Lego est déterminé par leur forme. Chaque pièce a un rôle dans la construction d'un objet plus grand. C'est à peu près de cette façon que fonctionne la programmation orientée objet : assembler de petits éléments pour en construire de plus grands.

Un autre exemple : la fabrication d'un ordinateur complet à partir de divers composants : une carte mère, un disque dur, un clavier, etc. Le système final résulte de l'assemblage de toutes les unités.

Chaque composant interne est particulier de par sa complexité, son origine (constructeur) et sa conception. Son fonctionnement spécifique présente peu d'intérêt, seul le résultat compte : un appui sur la touche A provoque l'affichage du caractère A à l'écran. Chaque composant est une unité indépendante et son seul intérêt réside dans son interaction avec les autres composants. Cette carte vidéo s'adapte-t-elle dans un slot de la carte mère ? Cet écran est-il compatible avec la carte vidéo ? Les composants dialoguent-ils correctement entre eux pour permettre aux différentes parties de converser ? Connaître la nature des interactions entre les différents composants et pouvoir les mettre en œuvre facilite l'assemblage complet du système.

La programmation orientée objet fonctionne de la même façon. Ainsi, ce type de programme est constitué de plusieurs composants (objets) autonomes. Ils ont un rôle spécifique et dialoguent de façon préétablie.

# Objets et classes

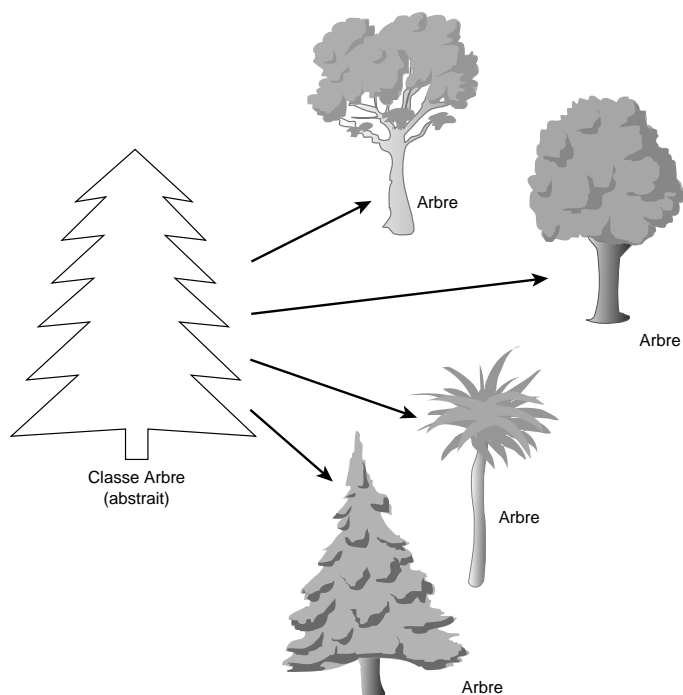
La programmation orientée objet est basée sur ce principe. Dans la réalité, des objets en constituent d'autres plus grands. Ce procédé de combinaisons n'est toutefois qu'un aspect très général de la POO. La programmation orientée objet présente plusieurs autres concepts et caractéristiques, qui facilitent et assouplissent la création et l'utilisation des objets. Les classes en sont les éléments les plus importants.

L'écriture d'un programme dans un langage POO ne définit pas des objets réels, mais des classes d'objets, une classe étant un modèle applicable à de multiples objets ayant des caractéristiques similaires. Une classe décrit toutes les caractéristiques communes à un ensemble d'objets.

Par exemple, une classe *Arbre* peut décrire les caractéristiques de tous les arbres (possèdent des feuilles et des racines, poussent, fabriquent de la chlorophylle). Elle sert alors de modèle abstrait pour le concept d'arbre. Pour la sélectionner, dialoguer avec elle ou l'abattre, il faut avoir une instance de cet arbre. Bien sûr, il est possible de créer de nombreuses instances d'une classe d'arbres déterminée. Chacune d'elles a éventuellement des caractéristiques différentes (court, haut, touffu, perd ses feuilles en automne), même si elles se ressemblent et sont identifiables en tant qu'arbre (voir Figure 2.1).

**Figure 2.1**

*La classe Arbre et plusieurs instances d'Arbre.*



**Lexique**

*Une classe est un modèle commun à un ensemble d'objets de mêmes caractéristiques.*

Une *instance* de classe est l'autre nom d'un objet réel. Si les classes sont une représentation abstraite d'un objet, une instance en est une représentation concrète.

Il n'y a donc aucune différence entre une instance de classe et un objet. Le terme objet est plus général mais ils sont tous deux une représentation concrète d'une classe. Les termes d'instance et d'objet s'utilisent indifféremment en langage de POO. Une instance arbre et un objet arbre représentent la même chose.

**Lexique**

*Une instance est une manifestation concrète d'une classe.*

Voici un exemple proche de la programmation Java : la création d'une classe pour un bouton de l'interface utilisateur. La classe `Button` définit les caractéristiques d'un bouton (son nom, sa taille, son aspect) et son comportement (faut-il un simple ou un double-clic pour l'activer, que fait-il quand il est activé ?). Après la définition de la classe `Button`, la création de ses instances est aisée (donc des objets bouton). Leurs caractéristiques de base sont définies dans la classe, mais leur aspect et leur comportement diffèrent éventuellement. Ce dernier est fonction des besoins. Avec une classe `Button`, la réécriture du code pour chaque bouton n'est pas obligatoire. La classe `Button` est utile pour créer différents types de boutons dans ce programme ou dans un autre.

**Astuce**

*Dans le langage C, la classe s'assimile à la création d'un nouveau type de donnée avec `struct` et `typedef`. Les classes fournissent plus qu'une simple collection de données. Ceci est illustré dans la suite du chapitre.*

L'écriture d'un programme Java nécessite la conception et la construction de plusieurs classes. Ainsi, les programmes s'exécutent, des instances de ces classes se créent et s'effacent au fur et à mesure des besoins. Le travail d'un programmeur Java est de créer le bon ensemble de classes pour que les programmes puissent fonctionner correctement.

En guise d'aide, l'environnement Java offre un ensemble standard de classes (une bibliothèque de classes) couvrant un grand nombre de comportements de base. Ce sont des tâches de programmation classiques (des classes pour des fonctions mathématiques de base, des tableaux, des chaînes de caractères, etc.) mais aussi des comportements graphiques ou de gestion de réseaux. Souvent, les bibliothèques de classes Java réduisent l'apport du programmeur à la création d'une classe unique utilisant les bibliothèques standard. Pour les programmes Java complexes, il faut créer un ensemble complet de classes. Celles-ci dialoguent alors selon des règles définies.

**Lexique**

*Une bibliothèque de classes est une collection de classes susceptibles d'être utilisées de façon répétitive dans différents programmes. Les bibliothèques de classes standard de Java comportent un nombre important de classes relatives aux tâches de base en Java.*

# Attributs et comportement

Chaque classe écrite en Java se compose généralement de deux parties : les attributs et le comportement. Dans cette section, une classe théorique simple appelée `Motorcycle` permet de les étudier. Ensuite, le code Java met en œuvre la représentation d'une moto.

## Attributs

Les attributs sont les caractéristiques individuelles qui différencient un objet d'un autre. Ils en définissent l'apparence, l'état et les autres qualités. Soit une classe théorique appelée `Motorcycle` avec les attributs suivants :

- *couleur* : rouge, vert, gris, brun
- *style* : patrouille, sport, standard
- *marque* : Honda, BMW, Bultaco

Les attributs d'un objet peuvent aussi inclure des informations sur son état, telles que l'état du moteur (en marche ou à l'arrêt), ou la vitesse sélectionnée.

Les attributs sont définis dans les classes par des variables. Les types et les noms de ces variables sont définis dans la classe. Chaque objet d'une classe pouvant avoir des valeurs différentes pour ses variables, on utilise le terme *variable d'instance* pour désigner la variable d'un objet.

## Lexique

*Une variable d'instance définit un attribut d'un objet. Les types et les noms des variables sont définis dans la classe, mais leurs valeurs sont fixées et modifiées dans l'objet.*

Les variables d'instance peuvent être initialisées à la création d'un objet et demeurer constantes pendant toute la vie de l'objet, ou être changées au cours du déroulement du programme. La modification des valeurs des variables entraîne celle des attributs de l'objet.

Aux variables d'instance viennent s'ajouter des variables de classe. Ces dernières s'appliquent à la classe elle-même et à tous ses objets. Les valeurs des variables d'instance sont stockées dans l'instance. Les valeurs des variables de classe sont stockées dans la classe elle-même. Le chapitre suivant étudie les caractéristiques des variables d'instance et de classe.

## Comportement

Le comportement d'une classe, c'est la manière dont opère une instance de cette classe ; par exemple, la façon dont elle réagit à une demande d'une autre classe ou d'un autre objet, ou ce qu'elle fait lorsque son état interne se modifie. Le comportement d'un objet, c'est sa façon de faire quelque chose ou de l'obtenir. A titre d'exemple, voici quelques comportements de la classe `Motorcycle` :

- Démarrer le moteur

- Arrêter le moteur
- Accélérer
- Changer de vitesse
- Caler

La définition du comportement d'un objet passe par la création de méthodes, des ensembles d'instructions Java exécutant certaines tâches. Elles ressemblent aux fonctions des autres langages, mais elles sont définies dans une classe et accessibles seulement dans cette classe. A l'inverse de C++, Java ne possède pas de fonctions définies en dehors des classes.

## Lexique

*Les méthodes sont des fonctions définies dans les classes. Elles agissent sur les objets de ces dernières.*

Les méthodes peuvent opérer sur un seul objet individuel, ou encore permettre à des objets de communiquer entre eux. Une classe ou un objet peut appeler les méthodes d'une autre classe ou d'un objet différent, pour signaler des modifications d'environnement ou demander à cet autre objet de changer d'état.

De même qu'il existe des variables de classe et des variables d'instance, il existe aussi des méthodes d'instance et des méthodes de classe. Les méthodes d'instance (appelées simplement méthodes) agissent sur un objet. Les méthodes de classe agissent sur une classe (ou sur d'autres objets). Ces dernières sont traitées en détail dans les prochains chapitres.

## Création d'une classe

Cette section est une application de la théorie étudiée jusque-là. La création de la classe `Motorcycle` montre comment définir les méthodes et les variables d'instance dans une classe. Vient ensuite l'application Java pour créer un nouvel objet de la classe `Motorcycle` et en afficher les variables d'instance.

## Info

*Nous laissons de côté l'étude détaillée de la syntaxe, qui compliquerait la compréhension de l'exemple, et qui de toute façon deviendra beaucoup plus évidente au cours des chapitres suivants. Pour le moment, l'important est de comprendre les structures de base de la définition de classe.*

Saisissez les lignes suivantes avec votre éditeur, en respectant la casse. Il s'agit d'une définition de classe basique.

```
class Motorcycle {  
}
```

Vous venez de créer une classe, même si elle ne réalise pas grand-chose pour le moment. C'est une classe Java dans sa version la plus simple.

Ajoutez les trois lignes suivantes juste sous la première, afin de créer des variables d'instance pour cette classe :

```
• String make;  
• String color;  
• boolean engineState;
```

Parmi les trois variables d'instance de l'exemple, `make` et `color` contiennent des objets `String` (un `String` est un chaîne de caractères ; `String` s'écrit avec un `S` majuscule et fait partie de la bibliothèque de classes standard mentionnée précédemment). La troisième, `engineState`, est une variable booléenne et indique si le moteur est en marche ou à l'arrêt ; la valeur `false` veut dire que le moteur est arrêté, et `true` veut dire qu'il tourne. Notez que `boolean` s'écrit avec un `b` minuscule. Une variable booléenne ne peut prendre que l'une des deux valeurs `true` ou `false`.

## Info

*Dans le langage Java, `boolean` est un véritable type de données. Il accepte les valeurs `true` ou `false`. A l'inverse de leurs homologues en langage C, les données booléennes en Java ne sont pas des nombres.*

Les comportements (méthodes) de la classe `moto` peuvent être nombreux, mais l'exemple se limite à un seul : une méthode qui démarre le moteur. Ajoutez maintenant les lignes suivantes sous les variables d'instance dans votre définition de classe :

```
• void startEngine() {  
•     if (engineState == true)  
•         System.out.println("The engine is already on.");  
•     else {  
•         engineState = true;  
•         System.out.println("The engine is now on.");  
•     }  
• }
```

La méthode `startEngine` teste le moteur pour savoir s'il tourne déjà (ligne `engineState == true`). Si c'est le cas, elle affiche un message qui l'indique. Si le moteur est à l'arrêt, elle change son état en `true` (mettant le moteur en marche), puis affiche le message de démarrage du moteur. Enfin, la méthode `startEngine()` ne retournant pas de valeur, sa définition commence par le mot `void`. (On peut aussi définir des méthodes qui retournent des valeurs ; ce sera l'un des thèmes du Chapitre 6.)

## Info

*Ici et tout au long de l'ouvrage, nous ajoutons systématiquement des parenthèses vides à la suite du nom d'une méthode à laquelle il est fait référence (par exemple, la méthode `startEngine()` du paragraphe précédent). C'est une convention adoptée par la communauté des programmeurs pour indiquer qu'il s'agit d'une méthode et non d'une variable.*

Sauvegardez le programme dans un fichier appelé `Motorcycle.java` (les fichiers doivent avoir le même nom que la classe qu'ils définissent). Le Listing 2.1 montre l'aspect du programme au stade actuel :

**Listing 2.1. Le fichier `Motorcycle.java`**

```
1 : class Motorcycle {
2 :
3 :     String make;
4 :     String color;
5 :     boolean engineState = false;
6 :
7 :     void startEngine() {
8 :         if (engineState == true)
9 :             System.out.println("The engine is already on.");
10 :        else {
11 :            engineState = true;
12 :            System.out.println("The engine is now on.");
13 :        }
14 :    }
15 : }
```

**Astuce**

*Le compilateur Java ne tient pas compte de l'indentation des différentes parties de la classe. L'utilisation d'une indentation particulière permet cependant de relire plus facilement la définition de classe. Celle de l'exemple (variables et méthodes décalées de la définition de la classe) est conservée dans tout le livre. Les bibliothèques de classes Java utilisent une indentation similaire. Le choix de l'indentation appartient au programmeur.*

Avant de compiler cette classe, ajoutez la méthode `showAtts`, sous la méthode `startEngine` (entre les lignes 14 et 15). Elle affiche les valeurs courantes de toutes les variables d'instance dans une instance de la classe `Motorcycle`. Voici cette méthode :

```
void showAtts() {
    System.out.println("This motorcycle is a"
        + color + " " + make);
    if (engineState == true)
        System.out.println("The engine is on.");
    else System.out.println("The engine is off.");
}
```

La méthode `showAtts` affiche deux lignes à l'écran : `make` et `color` de l'objet `motorcycle` et l'état du moteur.

Vous avez créé ainsi une classe Java comportant trois variables d'instance et deux méthodes définies.

Sauvegardez de nouveau ce fichier et compilez-le par l'une des méthodes suivantes :

*Après cet exemple, les mécanismes de compilation et d'exécution des programmes Java seront considérés comme acquis, et ne seront plus décrits.*

Depuis un shell DOS, faites un CD vers le répertoire contenant votre fichier source Java, et utilisez la commande `javac` pour le compiler :

```
javac Motorcycle.java
```

Glissez et déposez le fichier `Motorcycle.java` sur l'icône du compilateur Java.

Depuis une ligne de commande, faites un CD vers le répertoire contenant votre fichier source Java, et utilisez la commande `javac` pour le compiler :

```
javac Motorcycle.java
```

Lorsque vous exécutez ce petit programme en utilisant l'un des programmes `java` ou `Java Runner`, vous obtenez une erreur. Pourquoi ? Si vous exécutez directement une classe qui vient d'être compilée, Java suppose qu'il s'agit d'une application et recherche une méthode `main()`. Comme vous n'avez pas défini de méthode `main()` à l'intérieur de la classe, l'interpréteur Java produit l'un des deux messages d'erreur ci-après :

- In class Motorcycle: void main(String argv[]) is not defined
- Exception in thread "main": java.lang.UnknownError

Pour utiliser la classe `Motorcycle` (créer et manipuler des objets), il faut écrire un applet ou une application Java qui l'utilise, ou lui ajouter une méthode `main`. La deuxième solution est la plus simple (voir Listing 2.2). Cette méthode doit être ajoutée juste avant la dernière accolade de fermeture `()`, sous les méthodes `startEngine()` et `showAtts()`.

### **Listing 2.2** Méthode `main()` pour `Motorcycle.java`

```

1 : public static void main (String args[]) {
2 :     Motorcycle m = new Motorcycle();
3 :     m.make = "Yamaha RZ350";
4 :     m.color = "yellow";
5 :     System.out.println("Calling showAtts...");
6 :     m.showAtts();
7 :     System.out.println("-----");
8 :     System.out.println("Starting engine...");
9 :     m.startEngine();
10:    System.out.println("-----");
11:    System.out.println("Calling showAtts...");
12:    m.showAtts();
13:    System.out.println("-----");
14:    System.out.println("Starting engine...");
15:    m.startEngine();
16: }
```



Grâce à la méthode `main()`, la classe `Motorcycle` devient une application officielle. La compilation est alors possible sans message d'erreur. Le résultat de l'exécution est le suivant :

```

• Calling showAtts...
• This motorcycle is a yellow Yamaha RZ350
• The engine is off.
• -----
• Starting engine...
• The engine is on.
• -----
• Calling showAtts...
• This motorcycle is a yellow Yamaha RZ350
• The engine is now on.
• -----
• Starting engine...
• The engine is already on.

```

## Analyse

La méthode `main()` comportant beaucoup d'éléments nouveaux, voici, ligne par ligne, l'indication de ce qu'elle réalise (les deux chapitres suivants donnent des détails plus spécifiques).

La première ligne déclare la méthode `main()`. Elle prend toujours cette forme. Chacune de ses parties sera étudiée dans les chapitres suivants.

La ligne 2 crée un nouvel objet `Motorcycle` et stocke sa référence dans la variable `m`. Il est rare d'agir directement sur les classes dans les programmes Java. Au lieu de cela, on crée des objets, puis on appelle des méthodes pour ces objets.

Les lignes 3 et 4 mettent en place les variables d'instance de l'objet `Motorcycle`. La marque est maintenant `Yamaha RZ350` et la couleur `yellow`.

La ligne 5 imprime un message pour annoncer l'appel de la méthode `ShowAtts`, puis la ligne 6 appelle cette méthode, qui a été définie dans l'objet `Motorcycle`. Le nouvel objet `Motorcycle` affiche alors la valeur de ses variables d'instance (`make` et `color` renseignées précédemment) et signale l'arrêt du moteur.

La ligne 7 affiche une ligne de séparation pour une lecture plus facile.

La ligne 9 appelle la méthode `startEngine()` dans l'objet `motorcycle` pour démarrer le moteur. Son état change alors : il tourne.

La ligne 12 affiche à nouveau la valeur des variables d'instance. Cette fois, le message signale que le moteur tourne.

La ligne 15 essaye de démarrer à nouveau le moteur. Puisqu'il tourne déjà, cette tentative provoque l'affichage du message `The engine is already on`.

Le Listing 2.3 montre la version finale de la classe `Motorcycle`, telle qu'elle doit pouvoir être compilée et exécutée sans problème. N'oubliez pas que cet exemple, comme tous les autres exemples du livre, est inclus dans le CD-ROM fourni avec l'ouvrage.

*Listing 2.3. La version finale de Motorcycle Java*

```
1: class Motorcycle {
2:
3:     String make;
4:     String color;
5:     boolean engineState;
6:
7:     void startEngine() {
8:         if (engineState == true)
9:             System.out.println("The engine is already on.");
10:        else {
11:            engineState = true;
12:            System.out.println("The engine is now on.");
13:        }
14:    }
15:
16:    void showAtts() {
17:        System.out.println("This motorcycle is a "
18: + color + " " + make);
19:        if (engineState == true)
20:            System.out.println("The engine is on.");
21:        else System.out.println("The engine is off.");
22:    }
23:
24:    public static void main (String args[]) {
25:        Motorcycle m = new Motorcycle();
26:        m.make = "Yamaha RZ350";
27:        m.color = "yellow";
28:        System.out.println("Calling showAtts...");
29:        m.showAtts();
30:        System.out.println("-----");
31:        System.out.println("Starting engine...");
32:        m.startEngine();
33:        System.out.println("-----");
34:        System.out.println("Calling showAtts...");
35:        m.showAtts();
36:        System.out.println("-----");
37:        System.out.println("Starting engine...");
38:        m.startEngine();
39:    }
40:}
```

## Héritage, interfaces et packages

Vous êtes désormais familiarisé avec les classes, les objets, les méthodes et les variables, et vous savez les réunir dans un programme Java. L'héritage, les interfaces et les packages sont des mécanismes d'organisation des classes et de leurs comportements. Les bibliothèques de classes

Java utilisent tous ces concepts. La création de bonnes bibliothèques de classes pour les programmes passe par l'utilisation de ces mêmes concepts.

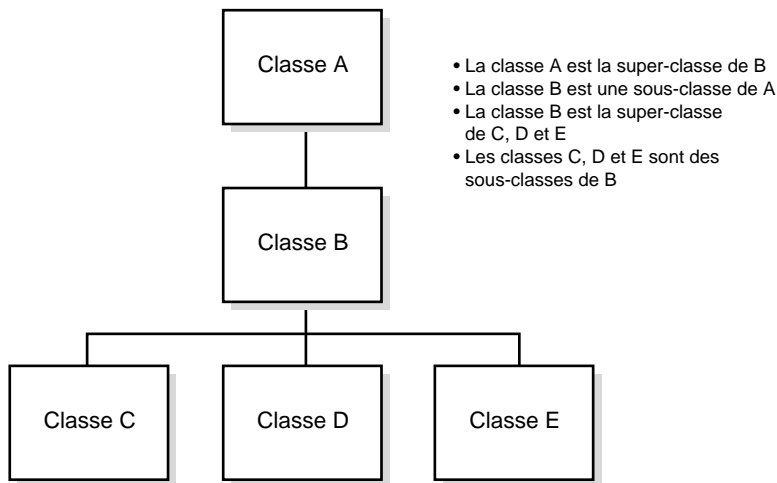
## Héritage

L'héritage est essentiel pour la programmation orientée objet et influence directement la conception et l'écriture des classes Java. L'héritage est un mécanisme puissant qui facilite l'écriture d'une classe. En effet, il suffit de spécifier les différences de cette classe par rapport à une autre ; l'héritage donne un accès automatique à toute l'information contenue dans cette autre classe.

Avec l'héritage, toutes les classes (écrites ou issues d'autres bibliothèques de classes ainsi que les classes utilitaires standard) suivent une hiérarchie stricte (voir Figure 2.2).

**Figure 2.2**

*Une hiérarchie de classes.*



Chaque classe possède une super-classe (la classe située au-dessus dans la hiérarchie) et éventuellement une ou plusieurs sous-classes (les classes situées au-dessous dans la hiérarchie). Les classes héritent de celles qui sont situées au-dessus d'elles dans la hiérarchie.

Les sous-classes héritent de toutes les méthodes et variables de leurs super-classes. Ainsi, si une super-classe définit un comportement utile à une classe, il n'est pas nécessaire de redéfinir cette dernière ou de copier ce code à partir d'une autre classe. En effet, la classe récupère automatiquement le comportement à partir de sa super-classe, et ainsi de suite jusqu'au sommet de la hiérarchie. Une classe devient alors une combinaison de toutes les caractéristiques des classes dont elle hérite.

## Lexique

*L'héritage est un concept de la programmation orientée objet, selon lequel toutes les classes font partie d'une hiérarchie stricte. Chaque classe possède des super-classes (les classes situées au-dessus dans la hiérarchie), et un nombre quelconque de sous-classes (les classes situées au-dessous dans la hiérarchie). Les sous-classes héritent des attributs et des comportements de leurs super-classes.*

La classe `Object` se trouve au sommet de la hiérarchie des classes Java. Toutes les classes héritent de cette super-classe. C'est la classe la plus générale de la hiérarchie. Elle définit le comportement spécifique à tous les objets dans Java. Chaque classe située à un niveau plus bas dans la hiérarchie ajoute des informations supplémentaires. Aussi, plus elles sont basses dans la hiérarchie, plus leur usage est spécifique. Ainsi, la hiérarchie de classe est un ensemble de concepts. Ils sont très abstraits au sommet, mais deviennent concrets quand on descend dans la chaîne des super-classes.

Il est souvent nécessaire de créer une classe qui possède non seulement toutes les informations d'une autre classe, mais également des caractéristiques complémentaires. Par exemple, vous pouvez vouloir créer une version de `Button` possédant une étiquette spécifique. Pour récupérer toutes les informations `Button`, il suffit de définir une classe qui en hérite. La classe récupère alors tous les comportements définis dans `Button` (et dans les super-classes de `Button`). Ensuite, il reste à gérer les différences entre cette classe et `Button`. Ce mécanisme, qui consiste à définir de nouvelles classes en fonction de leurs différences avec leurs super-classes, s'appelle *sous-classement*.

Le *sous-classement* concerne la création d'une classe qui hérite d'autres classes de la hiérarchie. Avec le sous-classement, il suffit de définir les différences entre la classe et ses parents. Les comportements complémentaires sont tous disponibles pour la classe grâce à l'héritage.

## Lexique

*Le sous-classement revient à définir une nouvelle classe héritière d'une classe qui existe déjà.*

Si la classe définit complètement un nouveau comportement et si elle n'est pas une véritable sous-classe, elle peut hériter directement de la classe `Object`, ce qui lui permet de s'intégrer elle aussi à la hiérarchie des classes Java. En fait, la définition d'une classe sans super-classe en première ligne amène Java à supposer qu'elle hérite de la classe `Object`. La classe `Motorcycle` créée dans la section précédente hérite ainsi de la classe `Object`.

## Création d'une hiérarchie de classes

Lorsqu'un très grand nombre de classes sont créées, dans le cas d'un programme très complexe, il est logique de penser qu'elles forment une hiérarchie. Elles héritent ainsi de la hiérarchie de classes et peuvent elles-mêmes former une hiérarchie. Créer une hiérarchie nécessite une organisation préalable du code Java. Cet effort supplémentaire est récompensé par de précieux avantages :

- Le développement des classes dans une hiérarchie permet de placer les informations communes à de nombreuses classes dans des super-classes. L'utilisation répétée de ces informations est ensuite possible. En fait, chaque sous-classe obtient ces informations communes par le mécanisme de l'héritage.
- Le changement (ou l'insertion) d'une classe dans le haut de la hiérarchie modifie automatiquement le comportement de ses sous-classes. En effet, ces dernières obtiennent les nouvelles informations par héritage et non par recopie du code, de sorte qu'il n'est pas nécessaire de les modifier et de les recompiler.

Soit la classe `Motorcycle` de l'exemple précédent et un programme qui met en œuvre toutes les caractéristiques d'une moto. Il s'exécute alors correctement, tout fonctionne. Il s'avère ensuite nécessaire de créer une classe `Car`. `Car` et `Motorcycle` ont beaucoup de points communs : l'une et l'autre comprennent des véhicules mûs par des moteurs, possédant des transmissions, des phares et des compteurs. Une première réaction consisterait à ouvrir la classe `Motorcycle` et à copier dans la nouvelle classe `Car` une bonne partie des informations déjà définies.

La meilleure solution est de placer les informations communes à `Car` et `Motorcycle` dans une hiérarchie de classes plus générale. Le travail de modification des classes `Car` et `Motorcycle` est important. À l'inverse, l'ajout des classes `Bicycle`, `Scooter`, `Truck`, etc., en nécessite peu. En effet, le comportement commun défini dans la super-classe le réduit de façon significative.

Voici une conception de hiérarchie de classes conforme à ce principe. Au sommet se trouve la classe `Object`, racine de toutes les classes Java. La classe la plus générale commune à `Car` et `Motorcycle` se nomme `Vehicle`. Un véhicule (*vehicle*) est un moyen de transport d'un endroit à un autre. Le seul comportement défini dans la classe `Vehicle` est le transport d'un point *a* vers un point *b*.

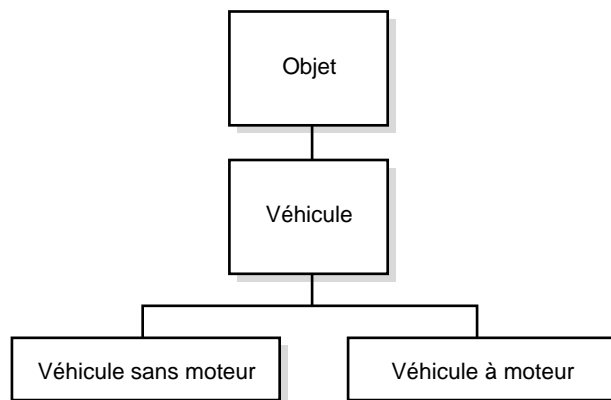
Pourquoi ne pas mettre ensuite sous la classe `Vehicle` les deux classes suivantes : `PersonPoweredVehicle` et `EnginePoweredVehicle`. `EnginePoweredVehicle` est différente de `Vehicle` tout court, à cause du moteur, auquel sont associés des comportements. Ces comportements peuvent être le démarrage et l'arrêt du moteur, la réserve en essence et huile, et aussi la vitesse de rotation du moteur. Les véhicules sans moteur (`PersonPoweredVehicle`) ont eux aussi leur comportement propre. Par exemple, ils transforment l'énergie humaine pour déplacer le véhicule (pédales). La Figure 2.3 illustre tout ceci.

L'ajout d'autres classes apporte une plus grande précision : `Motorcycle`, `Car`, `Truck`, etc. L'isolement de certains comportements dans des classes intermédiaires est également possible. Ainsi, les classes `TwoWheeled` et `FourWheeled` peuvent posséder différents comportements des véhicules à deux ou quatre roues (voir Figure 2.4).

Enfin, à partir de la sous-classe des deux-roues à moteur, on peut définir la classe des motos, `Motorcycle`. On peut définir aussi une classe pour les scooters ou une autre pour les mobylettes. En effet, ce sont aussi des véhicules à moteur à deux roues, mais avec des caractéristiques différentes de celles des motos.

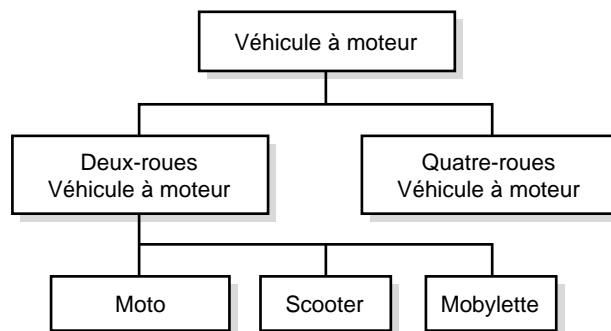
**Figure 2.3**

*Hiérarchie de base  
des véhicules.*



**Figure 2.4**

*Véhicules à deux  
et quatre roues.*



Où placer des caractéristiques comme la marque ou la couleur ? Là où elles s'intègrent le mieux dans la hiérarchie de classes. Par exemple, dans la classe `Vehicule` pour que toutes les sous-classes puissent y accéder. Une caractéristique ou un comportement doit être défini une seule fois dans la hiérarchie. Les sous-classes les utilisent ensuite automatiquement.

## Fonctionnement de l'héritage

Comment fonctionne l'héritage ? Comment des instances de classe accèdent-elles automatiquement aux méthodes et variables des classes situées au-dessus dans la hiérarchie ?

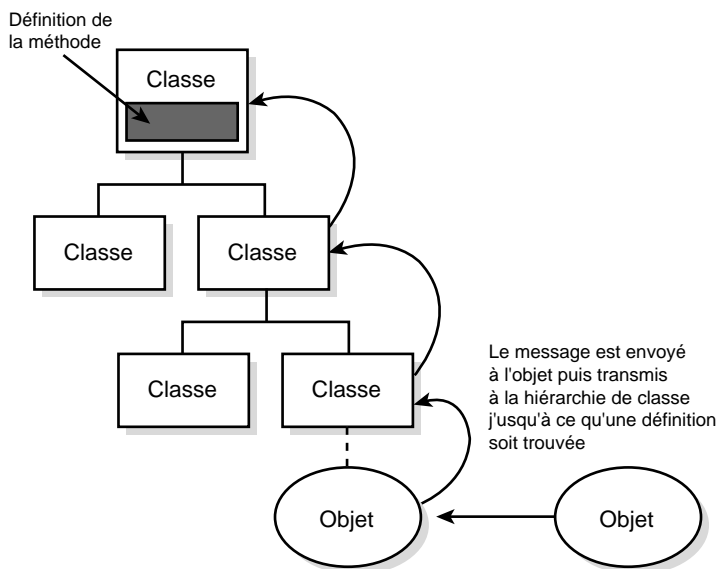
Pour les variables d'instance, lorsque vous créez un nouvel objet dans une classe, un emplacement est attribué à chaque variable définie dans la classe courante et dans toutes ses super-classes. Ainsi, la combinaison de toutes les classes constitue un modèle de l'objet courant. De plus, chaque objet apporte l'information appropriée à sa situation.

Les méthodes opèrent de manière similaire. Ainsi, les nouveaux objets ont accès à tous les noms de méthodes de leur classe et de leurs super-classes. Par contre, les définitions de méthodes sont choisies dynamiquement au moment de l'appel. Si l'appel porte sur un objet particulier, Java commence par chercher la définition de la méthode dans la classe de l'objet. S'il ne la trouve pas, il poursuit sa recherche dans la super-classe et ainsi de suite (voir Figure 2.5).

Les choses se compliquent lorsque deux méthodes définies dans une sous-classe et une super-classe ont la même signature (nom, nombre et types d'arguments). Dans ce cas, Java exécute la première définition trouvée (du bas de la hiérarchie vers le haut). Ce mécanisme permet de définir dans une sous-classe une méthode ayant la signature d'une méthode de l'une de ses super-classes, et ainsi de masquer cette dernière. On dit que la méthode a été redéfinie. La redéfinition est étudiée plus en détail au Chapitre 7.

**Figure 2.5**

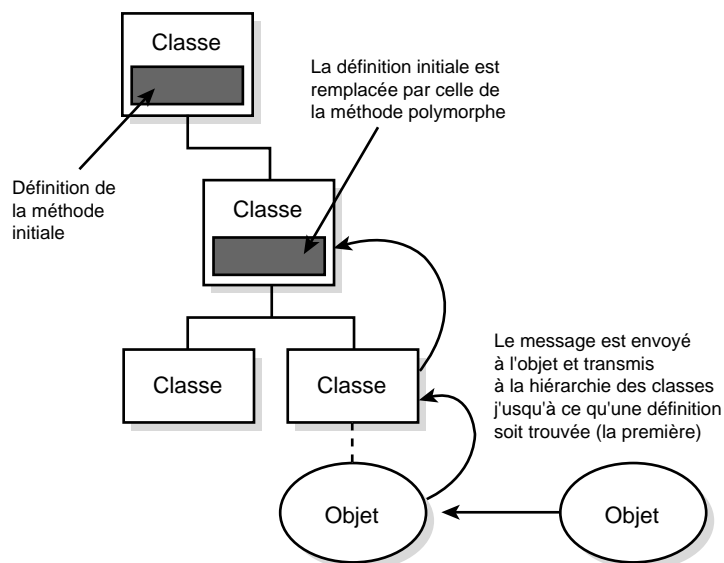
*Détermination de l'endroit où se trouve une méthode.*



## Lexique

*La redéfinition consiste à créer dans une sous-classe une méthode dont la signature (nom, nombre et type d'arguments) est identique à celle d'une super-classe. La nouvelle méthode "masque" alors celle de la super-classe (voir Figure 2.6).*

**Figure 2.6**  
*Redéfinition  
de méthodes.*



## Héritage simple ou multiple

La forme d'héritage étudiée précédemment s'appelle *héritage simple*. Dans l'héritage simple, chaque classe Java a une seule super-classe (même si une super-classe donnée possède éventuellement plusieurs sous-classes).

Dans les autres langages de programmation orientés objet comme C++ ou Smalltalk, les classes peuvent avoir plusieurs super-classes. Elles héritent alors de la combinaison des variables et méthodes de toutes leurs super-classes. Cela s'appelle l'*héritage multiple*. Ce mécanisme puissant permet aux classes d'avoir toutes sortes de comportements. Malheureusement, il complique souvent la définition des classes et le code. Pour ces raisons, Java se limite à l'héritage simple.

## Interfaces et packages

Il reste deux concepts de Java à étudier dans ce chapitre : les packages et les interfaces. Ils permettent la conception et la mise en œuvre de groupes de classes et du comportement de ces classes. Le Chapitre 16 couvre plus en détail les interfaces et les packages, mais il est utile de les aborder dès à présent.

Chacune des classes Java a une super-classe et une seule. Elle hérite des méthodes et des variables de sa super-classe, et de toutes les super-classes situées au-dessus. Avec l'héritage simple, les relations entre les classes et leurs fonctions sont faciles à comprendre et à concevoir. Cependant, cela s'avère souvent restrictif. C'est le cas si l'on veut mettre en œuvre des comportements



similaires sur différentes "branches" de la hiérarchie des classes. Java résout ce problème de comportement partagé grâce au concept d'interface. Une interface permet de rassembler un ensemble de noms de méthodes en un seul endroit, et d'ajouter ces méthodes en bloc à des classes qui en ont besoin. Notez que les interfaces ne contiennent que les noms de méthodes et les indications d'interface (par exemple les arguments), mais pas les définitions des méthodes concernées.

Bien qu'une classe Java possède une super-classe et une seule (héritage simple), elle peut implémenter un nombre quelconque d'interfaces. L'implémentation d'une interface consiste à définir des méthodes dont les noms figurent dans l'interface. Ainsi, deux classes très différentes peuvent répondre aux mêmes appels de méthodes mentionnées dans l'interface implémentée dans les deux classes. Ces classes peuvent faire des choses très différentes en réponse aux appels de ces méthodes.

## Lexique

*Une interface est une collection de noms de méthodes, sans définitions, qui peut être ajoutée à des classes pour leur adjoindre des comportements non représentés par les méthodes que la classe a définies elle-même, ou dont elle a hérité de ses super-classes.*

Ce chapitre ne va pas plus loin dans l'étude des interfaces, qui sera reprise ultérieurement.

Abordons maintenant le concept des packages.

Les *packages* regroupent des classes et des interfaces ayant un rapport entre elles, dans une seule collection ou bibliothèque. Grâce à eux, les groupes de classes sont disponibles uniquement s'ils sont nécessaires, ce qui permet notamment d'éviter des conflits entre noms de classes de différents groupes.

La troisième partie de ce livre étudie les packages en détail. Le présent chapitre se limite aux points suivants :

- Les bibliothèques de classes du JDK sont dans le package `java`. Les classes de ce package sont compatibles avec toutes les installations Java. La compatibilité n'est assurée que pour ces seules classes. Le package `java` contient d'autres packages dont les classes définissent le langage, les entrées/sorties, la gestion de réseaux de base, les fonctions de fenêtrage, et bien entendu les applets. Quand elles sont issues d'autres packages (de Sun ou de Netscape), les classes s'utilisent uniquement avec des installations spécifiques.
- Par défaut, les classes accèdent seulement à `java.lang` (le package du langage de base fourni avec le package Java). Pour utiliser celles d'un autre package, il faut les référencer explicitement avec le nom du package ou les importer dans le fichier source.
- Pour référencer une classe d'un package, il faut spécifier les noms des packages dans lesquels elle se situe à différents niveaux, et son nom, et séparer ces différents noms par un point (.). Par exemple, la classe `color`, contenue dans le package `awt` (*awt* signifie *Abstract Windowing Toolkit*), lui-même inclus dans le package `java`, est référencée ainsi : `java.awt.color`.

# Création d'une sous-classe

Pour terminer ce chapitre, vous allez créer une sous-classe d'une autre classe et redéfinir des méthodes. Cet exemple illustre en outre le fonctionnement des packages.

Un exemple typique de création d'une sous-classe est celui de la création d'une applet. Toutes les applets sont des sous-classes de la classe `Applet` (incluse dans le package `java.applet`). A la création d'une sous-classe d'`Applet`, les comportements des outils de fenêtrage et les classes de mise en page sont récupérés automatiquement. Ainsi, l'applet est dessinée au bon endroit dans la page et dialogue avec le système, pour répondre aux touches de fonction ou à la souris par exemple.

Dans cet exemple, l'applet est similaire à celle du chapitre précédent (Hello World), mais la chaîne Hello est écrite dans une couleur différente et une police plus grande. Pour commencer, il faut construire la définition de la classe. Depuis votre éditeur de texte, saisissez la définition de classe suivante :

```
public class HelloAgainApplet extends java.applet.Applet {  
}
```

La classe `HelloAgainApplet` est maintenant créée. Dans l'instruction, `extends java.applet.Applet` définit `HelloAgainApplet` comme une sous-classe de la classe `Applet` contenue dans le package `java.applet`. Pour accéder à cette classe, il faut la référencer explicitement, c'est-à-dire indiquer son nom et celui de son package.

Le mot clé `public` rend la classe disponible pour tout le système Java dès son chargement. `Public` est généralement utilisé pour rendre une classe visible par toutes celles du programme Java. Pour les applets, le mot clé `public` est obligatoire (les classes `public` sont traitées dans la troisième partie de ce livre).

Une définition de classe vide (sans variables ni méthodes) n'a aucune utilité. Pour qu'une sous-classe serve à quelque chose, il faut lui ajouter des méthodes et des variables, ou redéfinir des méthodes de sa super-classe. Ajoutez des informations à cette classe, entre les deux accolades, pour la différencier de sa super-classe.

Dans un premier temps, ajoutez-lui une variable d'instance contenant un objet `Font` :

```
Font f = new Font ("TimesRoman",Font.BOLD,36);
```

La variable d'instance `f` contient maintenant une nouvelle instance de la classe `Font` incluse dans le package `java.awt`. Cet objet `Font` est la police de caractères Times Roman, gras, 36 points. Dans l'applet Hello World précédente, la police par défaut `Font` utilisée était : 12 points, Times Roman. L'utilisation d'un objet `Font` permet de changer la police du texte écrit par l'applet.

La variable d'instance créée pour contenir l'objet "Font" rend celui-ci accessible à toutes les méthodes de la classe. Créez maintenant une méthode qui l'utilise.

Une applet emploie plusieurs méthodes standard déjà définies dans ses super-classes, et qui sont le plus souvent redéfinies dans la nouvelle classe applet. Cela concerne les méthodes d'initialisation et de lancement de l'applet, mais aussi celles qui dirigent des opérations comme le mouvement et les clics de la souris, ou la libération des ressources en fin d'exécution. Une de ces méthodes standard est `paint()`, qui affiche l'applet sur l'écran. La définition par défaut de `paint()` ne fait rien, c'est une méthode vide. En redéfinissant `paint()`, on indique simplement à l'applet ce qu'elle doit afficher. Voici une définition de `paint()` :

```
• public void paint (Graphics g) {  
•     g.setFont(f);  
•     g.setColor(Color.red);  
•     g.drawString("Hello again !", 5, 25);  
• }
```

En ce qui concerne la méthode `paint()`, deux notions sont à retenir. En premier lieu, tout comme l'applet, cette méthode est déclarée `public`. Cependant, la raison est différente : la méthode qu'elle redéfinit est elle-même `public`. Si une méthode d'une super-classe est définie comme `public`, une méthode qui la redéfinit doit elle aussi être `public`, sinon une erreur se produit à la compilation de la classe.

En second lieu, la méthode `paint()` reçoit un seul argument : une instance de la classe `Graphics`. Cette classe fournit des comportements indépendants de la plate-forme pour les polices, les couleurs et les opérations de base de dessin de lignes et de formes. La deuxième partie de ce livre étudie la classe `Graphics` au travers d'applets plus importantes.

Dans l'exemple, la méthode `paint()` vous a permis de faire trois choses :

- Indiquer à l'objet `graphics` la police à utiliser par défaut, celle contenue dans la variable `f`.
- Indiquer à l'objet `graphics` la couleur par défaut, une instance (`red`) de la classe `Color` :
- Afficher la chaîne "Hello Again !" à l'écran aux positions `x` et `y` suivantes : 5 et 25. Cette chaîne apparaît avec la nouvelle police et la nouvelle couleur.

Cela suffit pour une applet aussi simple. Elle ressemble maintenant à ceci :

```
• public class HelloAgainApplet extends java.applet.Applet {  
•     Font f = new Font ("TimesRoman",Font.BOLD,36);  
•     public void paint (Graphics g) {  
•         g.setFont(f);  
•         g.setColor(Color.red);  
•         g.drawString("Hello again !", 5, 50);  
•     }  
• }
```

Cet exemple présente une erreur. Pour en prendre connaissance, sauvegardez le fichier sous le nom de la classe `HelloAgainApplet.java` et compilez-le. Le compilateur Java envoie le message d'erreur suivant :

```
HelloAgainApplet.java :7 :Class Graphics not found in type declaration.
```

Ce message apparaît parce que les classes référencées dans l'exemple font partie d'un package qui n'est pas disponible par défaut. Le seul package accessible automatiquement dans votre programme Java est `java.lang`. Ainsi, la référence à la classe `Applet` dans la première ligne de la définition de classe indique le nom de son package complet (`java.applet.Applet`). Dans les lignes suivantes, toutes les autres classes sont référencées comme si elles étaient déjà disponibles. Le compilateur s'en aperçoit et vous dit que vous n'avez pas accès à ces autres classes.

Il existe deux méthodes pour résoudre ce problème : référencer toutes les classes externes avec le nom du package complet, ou importer la classe ou le package approprié en tête du fichier classe. Le choix importe peu, mais la seconde solution diminue le volume de code à écrire si vous vous référez souvent à des classes d'un même package.

Dans l'exemple, importez les classes dont vous avez besoin. Elles sont au nombre de trois : `Graphics`, `Font` et `Color`, et appartiennent au package `java.awt`. Pour les importer, placez les codes suivants en tête du programme, avant la définition de la classe :

```
• import java.awt.Graphics;
• import java.awt.Font;
• import java.awt.Color;
```

## Astuce

L'astérisque (\*) remplace le nom de la classe pour l'importation d'un package complet de classes (public). Par exemple, pour importer toutes les classes du package `awt`, utilisez l'instruction suivante :

```
import java.awt.*;
```

Les classes sont maintenant correctement importées dans le programme. La compilation de `HelloAgainApplet` doit alors s'effectuer correctement. Le Listing 2.4 vous montre sa version finale.

### Listing 2.4. La version finale de `HelloAgainApplet.java`

```
• 1:import java.awt.Graphics;
• 2:import java.awt.Font;
• 3:import java.awt.Color;
• 4:
• 5:public class HelloAgainApplet extends java.applet.Applet {
• 6:
• 7:    Font f = new Font("TimesRoman",Font.BOLD,36);
• 8:
• 9:    public void paint(Graphics g) {
• 10:        g.setFont(f);
• 11:        g.setColor(Color.red);
• 12:        g.drawString("Hello again!", 5, 40);
• 13:    }
• 14:}
```

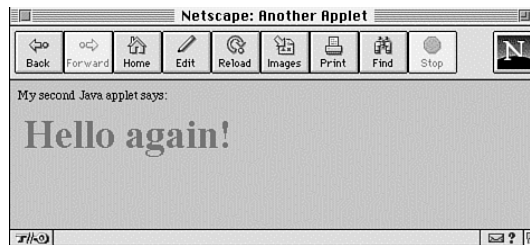
Pour tester le fichier classe obtenu, créez un fichier HTML avec l'étiquette <APPLET> comme dans le chapitre précédent :

```
<HTML>
<HEAD>
<TITLE>Another Applet</TITLE>
</HEAD>
<BODY>
<P>My second Java applet says:
<BR><APPLET CODE="HelloAgainApplet.class" WIDTH=200 HEIGHT=50>
</APPLET>
</BODY>
</HTML>
```

Dans cet exemple, le fichier classe Java et le fichier HTML se trouvent dans le même répertoire. Sauvegardez le fichier dans `HelloAgainApplet.html`. Lancez ensuite le navigateur compatible Java ou Appletviewer de Java. La Figure 2.7 affiche le résultat de l'exécution (la chaîne "Hello Again!" est rouge).

**Figure 2.7**

*L'applet HelloAgain.*



## Résumé

Une partie des informations de ce chapitre peut sembler trop théorique aux débutants en programmation orientée objet. Il n'y a pas à s'en inquiéter : ils comprendront mieux en avançant dans le livre et en créant des classes et des applications.

Une des difficultés essentielles de la programmation orientée objet réside dans le nom des concepts. La POO possède un jargon très étendu. Voici un résumé des termes et des concepts abordés dans ce chapitre :

*Classe* : modèle d'objet contenant des variables et des méthodes. Celles-ci définissent le comportement et les attributs. Les classes peuvent hériter de variables et de méthodes issues d'autres classes.

*Méthode de classe* : méthode définie dans une classe, opérant sur la classe elle-même et pouvant être appelée par l'intermédiaire de la classe ou l'une de ses instances.

*Variable de classe* : variable "appartenant" à la classe et à toutes les instances prises dans leur ensemble, dont la valeur est stockée dans la classe.

*Instance* : la même chose qu'un objet ; tout objet est une instance d'une classe.

*Méthode d'instance* : méthode définie dans une classe, qui opère sur une instance de cette classe. Les méthodes d'instance sont habituellement appelées simplement méthodes.

*Variable d'instance* : variable appartenant à une instance individuelle, dont la valeur est stockée dans l'instance.

*Interface* : une collection de spécifications abstraites de comportements, qui peuvent être implémentées par les classes individuelles.

*Objet* : instance concrète d'une classe. Tous les objets qui sont des instances d'une même classe accèdent aux mêmes méthodes, mais leurs variables d'instance ont en général des valeurs différentes.

*Package* : collection de classes et d'interfaces. Dans le cas des classes incluses dans d'autres packages que `java.lang`, le nom du package complet doit être spécifié pour les référencer, ou le package doit être importé.

*Sous-classe* : une classe située plus bas, dans la hiérarchie de l'héritage, que sa classe parente appelée super-classe. La création d'une classe est souvent appelée sous-classement.

*Super-classe* : une classe située au-dessus de ses sous-classes dans la hiérarchie de l'héritage.

## Questions — réponses

**Q Les méthodes sont des fonctions définies dans les classes. Si elles ressemblent à des fonctions et se comportent comme elles, pourquoi les appelle-t-on méthodes ?**

**R** Certains langages de programmation orientés objet les nomment fonctions (C++ les appelle fonctions membres). D'autres langages orientés objet font la différence entre les fonctions internes à une classe ou un objet et les fonctions externes. Dans ce cas, avoir des termes différents est important pour comprendre le fonctionnement de chacune. Java a choisi le terme de méthode car la différence entre fonctions internes et externes n'est significative que pour d'autres langages, et parce que le terme méthode est maintenant couramment utilisé dans la technologie orientée objet.

**Q Je comprends ce que représentent les variables et méthodes d'instance mais pas la notion de méthodes et de variables de classe.**

**R** Tout ce qui est fait dans des programmes Java se réalise avec des objets. Malgré tout, il est utile de stocker certains comportements et attributs dans la classe et non dans l'objet. Par exemple, pour créer une nouvelle instance dans une classe, il faut une méthode définie et disponible dans la classe elle-même, faute de quoi on ne peut pas créer cette instance.

Les variables de classe, elles, sont souvent utilisées lorsque la valeur d'un attribut doit être partagée par toutes les instances de la classe.

En général, vous utilisez des variables et des méthodes d'instance. Vous trouverez des informations détaillées sur les variables et les méthodes de classe dans les chapitre suivants de cette première partie.

3

# Les bases de Java



LE PR  GRAMMEUR



Vous avez vu jusqu'ici les aspects les plus généraux de la programmation Java, ce qu'est l'exécutable d'un programme Java, et comment créer des classes simples. Le Chapitre 3 traite plus en détail du langage de programmation Java.

Dans ce chapitre, vous ne définirez ni classes ni objets, et vous n'aurez pas à vous soucier de la façon dont ces éléments communiquent au sein d'un programme Java. Vous vous contenterez d'examiner les instructions Java élémentaires. Elles sont utilisées pour réaliser les tâches de base dans une définition de méthode telle que `main()`.

Ce chapitre aborde les thèmes suivants :

- Les instructions et expressions de Java
- Les types de données et de variables
- Les commentaires
- Les constantes
- Les opérateurs arithmétiques
- Les comparaisons
- Les opérateurs logiques

## Info

*Java ressemble à C++ et, par extension, au langage C. Une grande partie de sa syntaxe est connue des programmeurs expérimentés en langage C ou C++. Les notes techniques (comme celle-ci) les intéresseront plus particulièrement, puisqu'elles les informent des différences que C et C++ ou les autres langages traditionnels présentent avec Java.*

# Instructions et expressions

Une instruction s'écrit très simplement avec Java. Elle représente une opération, comme le montrent les exemples suivants :

```
● int i = 1;  
● import java.awt.Font;  
● System.out.println("This motorcycle is a"  
  + color + " " + make);  
● m.engineState = true;
```

Les instructions retournent parfois des valeurs. C'est le cas pour l'addition de deux nombres ou pour un test d'égalité entre deux valeurs. Ces instructions, appelées expressions, sont étudiées plus loin dans ce chapitre.

Les espaces placés dans les instructions Java sont ignorés, comme dans le langage C. Une instruction peut figurer sur une seule ligne ou sur plusieurs, ce qui est sans incidence sur la façon dont elle est lue par le compilateur Java. Ce qui importe, c'est que toutes les instructions Java se terminent par un point-virgule. L'absence de celui-ci provoque une erreur à la compilation.

Java possède également des instructions composées, ou blocs, utilisées comme une instruction simple. Elles commencent et finissent par des accolades (`{}`). Le Chapitre 5 apporte des précisions sur la notion de blocs.

## Types de données et de variables

Les variables sont des emplacements de mémoire dans lesquels on peut stocker des valeurs. Toute variable a un nom, un type et une valeur. Avant d'utiliser une variable, il faut la déclarer. On peut ensuite lui attribuer une valeur (comme vous allez le voir plus loin, on peut lui attribuer une valeur en même temps qu'on la déclare).

Java possède trois sortes de variables : les variables d'instance, les variables de classe et les variables locales.

Les variables d'instance déterminent les attributs ou l'état d'un objet donné. Les variables de classe ont le même type de fonction, mais leurs valeurs s'appliquent à tous les objets de la classe (et à la classe elle-même) et non à un seul objet.

Les variables locales sont déclarées et utilisées dans les définitions de méthodes. Elles servent de compteur d'index dans une boucle, de variable temporaire ou encore, elles contiennent des valeurs utiles à la définition de la méthode. Elles sont aussi utilisées dans les blocs, étudiés au Chapitre 5. La définition de variable et sa valeur cessent d'exister après l'exécution de la méthode (ou du bloc). Elles sont utiles pour stocker les informations destinées à une seule méthode. On utilise des variables locales pour stocker des données utilisées par une seule méthode (ou un seul bloc), et les variables d'instance pour les données utilisées par plusieurs méthodes dans un objet.

Les variables locales et les variables d'instance sont déclarées de façon similaire, mais l'accès et l'attribution de valeurs aux variables d'instance ou de classe diffèrent de ceux des variables locales. Ce chapitre traite plus particulièrement de l'utilisation des variables dans les définitions de méthodes, c'est-à-dire des variables locales. Le chapitre suivant montre comment s'y prendre avec les variables de classe et les variables d'instance.

### Info

*Contrairement à d'autres langages, Java ne possède pas de variable globale. Les variables d'instance et de classe les remplacent. Elles servent à communiquer des informations "globales" dans et entre les différents objets. Comme Java est un langage orienté objet, il faut penser en termes d'objets et de dialogues entre objets, et non en termes de programmes.*

## Déclaration des variables

Une variable doit être déclarée avant d'être utilisée dans un programme. Sa déclaration se compose d'un type et d'un nom de variable :

```
• int myAge;  
• String myName;  
• boolean isTired;
```

Les définitions de variables se placent n'importe où dans la définition de la méthode (là où il peut y avoir une instruction Java habituelle), mais elles sont souvent déclarées en tête :

```
• public static void main (String args[]) {  
•     int count;  
•     String title;  
•     boolean isAsleep;  
•     ...  
• }
```

Toutes les variables du même type peuvent être déclarées ensemble :

```
• int x, y, z;  
• String firstName, LastName;
```

L'initialisation d'une variable est possible dans la déclaration :

```
• int myAge, mySize, numShoes = 28;  
• String myName = "Laura";  
• boolean isTired = true;  
• int a = 4, b = 5, c = 6;
```

Si une ligne contient plusieurs variables mais une valeur d'initialisation unique (voir première ligne de l'exemple), seule la dernière variable est initialisée. Pour initialiser plusieurs variables sur une seule ligne, il suffit de les séparer avec une virgule, comme dans la dernière ligne de l'exemple ci-avant.

Avant d'utiliser une variable locale, il faut lui attribuer une valeur (la compilation du programme Java échoue si une variable locale est vide). Pour cette raison, il est préférable d'initialiser les variables locales dans leurs déclarations. Cette contrainte n'existe pas pour les variables d'instance ou de classe, qui ont une valeur par défaut en fonction de leur type :

<code>null</code>	pour les instances de classe,
<code>0</code>	pour les variables numériques,
<code>'\0'</code>	pour les caractères, et
<code>false</code>	pour les booléens.

## Noms de variables

Les noms de variables du langage Java commencent par une lettre, le signe de soulignement (`_`) ou le signe dollar (`$`). Ils ne peuvent pas commencer par un chiffre. Les caractères suivants sont des chiffres ou des lettres. Les symboles comme `%`, `*`, `@` ou `/` doivent être évités car ils sont souvent réservés aux opérateurs de Java.

Java utilise également le jeu de caractères Unicode. Unicode regroupe tous les caractères ASCII standard et plusieurs milliers d'autres caractères. La plupart des alphabets internationaux sont ainsi représentés. Les noms de variables peuvent alors contenir des caractères accentués ou tout autre glyphe, à condition que le code du caractère Unicode soit supérieur à `00C0`.



*La codification Unicode est regroupée en deux volumes contenant des milliers de caractères. Si Unicode n'est pas réellement nécessaire, il est préférable de se limiter aux chiffres et aux lettres pour nommer les variables.*

Enfin, Java différencie les majuscules et les minuscules. Aussi, la variable `X` est-elle différente de la variable `x`. Un autre exemple : `rose` n'est ni une `Rose` ni une `ROSE`.

Par convention, un nom de variable Java doit être représentatif de cette variable. Il se compose souvent de plusieurs mots combinés. Le premier est écrit en minuscules et les mots suivants commencent par une majuscule :

```
● bouton theButton;  
● long reallyBigNumber;  
● boolean currentWeatherStateOfPlanetXShortVersion;
```

## Types de variables

Une déclaration de variable se compose d'un nom mais aussi d'un type, qui détermine les valeurs possibles de cette variable. Un type de variable :

- un des huit types de données primaires,
- un nom de classe,
- un tableau.

La déclaration et l'utilisation des variables tableaux sont traitées au Chapitre 5. Le présent chapitre se limite aux types de données primaires et au type de classe.

## Types de données primaires

Les huit types de données primaires regroupent les types entiers usuels, les nombres à virgule flottante, les caractères et les valeurs booléennes. Ils sont appelés primaires car ils sont construits dans le système et ne sont pas des objets, ce qui rend leur utilisation plus efficace. Ils sont indépendants de la machine, et par suite, leur taille et leurs caractéristiques ne varient pas selon les programmes.

Il existe quatre types d'entiers dans Java, correspondant chacun à un intervalle de valeurs (voir Tableau 3.1). Ils sont tous signés, et acceptent donc les nombres négatifs ou positifs. Le choix du type dépend des valeurs maximales et minimales que peut prendre la variable. Si une valeur stockée devient trop grande pour le type de la variable, elle est tronquée, sans autre indication.

**Tableau 3.1** Les types d'entiers

Type	Taille	Intervalle de valeurs
byte	8 bits	-128 à 127
short	16 bits	-32 768 à 32 767
int	32 bits	-2 147 483 648 à 2 147 483 647
long	64 bits	-9 223 372 036 854 775 808 à 9 223 372 036 854 775 807

Les nombres à virgule flottante sont utilisés pour les nombres comportant une partie décimale. Dans Java, ils respectent les règles IEEE 754 (un standard international pour l'arithmétique et la définition des nombres à virgule flottante). Java contient deux types de nombres à virgule flottante : `float` (32 bits, simple précision) et `double` (64 bits, double précision).

Le type `char` définit les caractères. Comme Java utilise la codification Unicode, le type `char` est non signé, avec une précision de 16 bits.

Enfin, le type `boolean` accepte deux valeurs : `true` ou `false`. Pour Java, contrairement aux langages analogues à C, `boolean` n'est pas un nombre. Il ne peut donc être traité comme tel. Tous les tests d'une variable booléenne doivent rechercher les valeurs `true` ou `false`.

Notez que tous les types de données primaires sont indiqués en minuscules. Ce point est important, car il existe aussi des classes de mêmes noms, mais commençant par une majuscule, et dont le comportement est différent. Par exemple, le type primaire `boolean` est différent de la classe `Boolean`. Vous en apprendrez davantage sur ces classes particulières et sur leur utilisation au Chapitre 4.

## Types de classes

Les variables de Java peuvent aussi être déclarées pour recevoir un objet d'une classe particulière :

```

● String LastName;
● Font basicFont;
● OvalShape myOval;

```

Ces variables reçoivent uniquement les objets d'une classe donnée ou de l'une de ses sous-classes. Cette seconde variante est utile lorsqu'une variable doit pouvoir contenir des instances de diverses classes ayant une même super-classe. Par exemple, dans le cas d'un ensemble de classes de fruits (Pomme, Poire, Fraise, etc.), toutes héritières de la classe générale `Fruit`, si vous déclarez une variable de type `Fruit`, cette variable peut contenir des instances de n'importe

laquelle des sous-classes de `Fruit`. Si l'on déclare une variable de type `Object`, cette variable peut contenir n'importe quel objet.

## Info

*Java ne possédant pas d'instruction `typedef` (comme en C ou C++), la déclaration d'un nouveau type passe par celle d'une nouvelle classe. Il est ensuite possible de déclarer une variable du type de cette classe.*

## Attribution de valeurs aux variables

Lorsqu'une variable a été déclarée, on peut lui attribuer une valeur au moyen de l'opérateur d'affectation (`=`), de la façon suivante :

```
• size = 14;  
• tooMuchCaffeine = true;
```

## Commentaires

Java possède trois sortes de commentaires, deux pour les commentaires ordinaires du code source, et une troisième pour le système particulier de documentation `javadoc`.

Les symboles `/*` et `*/` entourent des commentaires multilignes. Le texte compris entre ces deux bornes est ignoré :

```
• /* I don't know how I wrote this next part; I was working  
• really late one night and it just sort of appeared. I  
• suspect the code elves did it for me. It might be wise  
• not to try and change it.  
• */
```

Les commentaires ne s'imbriquent pas ; autrement dit, vous ne pouvez pas placer un commentaire à l'intérieur d'un commentaire.

Le double slash (`//`) précède les commentaires dans une ligne. Tout le texte situé entre le double slash et la fin de ligne est ignoré :

```
int vices =7; // Ceci est un commentaire
```

Enfin, les commentaires spéciaux destinés au système `javadoc` commencent avec `/**` et se terminent par `*/`. `Javadoc` génère de la documentation API à partir du code. Ce livre ne traite pas `javadoc`. Pour obtenir des informations sur cet outil, consultez la documentation fournie avec le JDK ou la page d'accueil Java de Sun (<http://java.sun.com>).

# Constantes

Les constantes stockent des valeurs simples utiles aux programmes, suivant le principe : ce que vous tapez, c'est aussi ce que vous obtenez.

*Constante* est un terme de programmation. La frappe de `4` dans un programme Java entraîne la mémorisation de la valeur 4 dans un entier. Si vous tapez `'a'` vous obtenez un caractère ayant la valeur `a`.

Il existe plusieurs types de constantes dans Java : nombres, caractères, chaînes et valeurs booléennes.

## Constantes nombre

Les entiers sont représentés par plusieurs types de constantes. Par exemple, le nombre 4 est une constante entière décimale de type `int` (même si cette valeur peut être attribuée à une variable de type `byte` ou `short`, vu sa taille). Une constante entière décimale trop grande pour un type `int` est automatiquement de type `long`. Ajouter `L` ou `l` à un nombre force ce type `long` (`4L` est un entier long de valeur 4). Les entiers négatifs sont précédés d'un signe moins (`-`) : par exemple, `-45`.

Les entiers s'expriment en octal ou en hexadécimal. Si un entier commence par `0`, le nombre est en octal : `0777` ou `0004`, par exemple. Si l'entier commence par `0x` (ou `0X`), c'est un nombre hexadécimal (`0xFF`, `0XAF45`). Les nombres hexadécimaux sont constitués de chiffres (0 à 9) et de lettres majuscules ou minuscules (`a` [ou `A`] à `f` [ou `F`]).

Les constantes à virgule flottante sont généralement constituées de deux parties : la part entière et la part décimale (par exemple `5.677777`). Elles deviennent alors des nombres à virgule flottante de type `double` sans tenir compte de la précision. Ajouter la lettre `f` (ou `F`) à un nombre force le type `float` : `2.56F`.

Pour avoir des exposants avec les constantes à virgule flottante, il faut ajouter les lettres `e` ou `E` suivies de l'exposant (qui peut être négatif) à la fin du nombre : `10e45` ou `.36E-2`.

## Constantes booléennes

Les constantes booléennes sont représentées par deux mots clés : `true` ou `false`. Vous pouvez utiliser ces mots clés partout où vous voulez effectuer un test, ou bien encore, comme les deux seules valeurs possibles pour les variables booléennes.

## Constantes caractère

Les constantes caractère se composent d'une lettre entourée de guillemets simples : `'a'`, `'#'`, `'3'`, etc. Ces caractères sont stockés sous forme de codes Unicode 16 bits. Le Tableau 3.2

présente les codes spéciaux. Ce sont des caractères non imprimables ou Unicode. Dans les séquences, la lettre *d* représente un chiffre (0 à 9) ou une lettre (a à f ou A à F).

**Tableau 3.2 Codes des caractères spéciaux**

<i>Séquence</i>	<i>Signification</i>
<code>\n</code>	Retour à la ligne
<code>\t</code>	Tabulation
<code>\b</code>	Retour arrière
<code>\r</code>	Retour chariot
<code>\f</code>	Saut de page
<code>\\</code>	Slash inversé
<code>\'</code>	Guillemet simple
<code>\"</code>	Guillemet double
<code>\ddd</code>	Octal
<code>\xdd</code>	Hexadécimal
<code>\udddd</code>	Caractère Unicode

## Info

*Les programmeurs en C et C++ auront certainement remarqué l'absence des caractères de contrôle `\a` (alarme) et `\v` (tabulation verticale).*

## Constantes chaîne

Une chaîne est une combinaison de caractères. En langage Java, les chaînes sont des objets de la classe `String`, et non de simples tableaux de caractères, comme dans C ou C++. De plus, elles empruntent certaines de leurs caractéristiques aux tableaux (la possibilité de tester leur longueur, d'ajouter ou d'effacer certains caractères). Comme ce sont des objets à part entière pour Java, elles possèdent des méthodes permettant de les combiner, de les tester ou de les modifier très facilement.

Les chaînes sont représentées par une série de caractères entourée de guillemets doubles :

- `"Hi, I'm a string literal. "`
- `" " //une chaîne vide`

Les chaînes peuvent contenir des caractères de contrôle (retour à la ligne, tabulation ou caractères Unicode) :

- `"A string with a \t tab in it"`
- `"Nested strings are \"strings inside of\" other strings"`
- `"This string brought to you by Java\u2122"`

Dans le dernier exemple, la séquence Unicode `\u2122` produit le symbole <sup>TM</sup>.



## Info

Pouvoir représenter un caractère par son code Unicode ne signifie pas qu'il peut être affiché. L'ordinateur ou le système d'exploitation utilisé ne supporte peut-être pas Unicode, ou votre police de caractères peut ne pas contenir son image. La reconnaissance des séquences Unicode par Java offre un moyen de coder des caractères spéciaux pour des systèmes compatibles Unicode. Java 1.02 était plus limité à cet égard que Java 1.1 ; en Java 1.1, vous pouvez afficher n'importe quel caractère Unicode, dès lors qu'il existe une image pour ce caractère. Pour plus d'informations sur Unicode, consultez la page d'accueil Unicode à <http://unicode.org>.

L'utilisation d'une constante chaîne dans un programme Java entraîne la création automatique d'un objet de la classe `String` avec la valeur fournie. C'est la particularité des chaînes. Les autres constantes ne se comportent pas ainsi : les types primaires ne sont pas de véritables objets, et la création d'un nouvel objet, autre qu'un objet `String`, nécessite la création explicite d'une nouvelle instance de classe.

## Opérateurs et expressions

Dans Java, une expression est la plus simple forme d'une instruction accomplissant quelque chose. Toutes les expressions, lorsqu'elles sont évaluées, retournent une valeur (les autres instructions ne le font pas nécessairement).

Les opérations arithmétiques et les tests d'égalité sont des expressions. Une expression retourne une valeur, qui est attribuée à une variable ou testée dans d'autres instructions.

La plupart des expressions Java utilisent des opérateurs. Les opérateurs sont des symboles spéciaux, pour des opérations arithmétiques, différentes formes d'affectation, des opérations d'incrémement ou de décrémement, ainsi que pour des opérations logiques.

## Lexique

*Les expressions sont des instructions qui retournent une valeur.*

*Les opérateurs sont des symboles spéciaux couramment utilisés dans les expressions.*

### Opérateurs arithmétiques

Java possède cinq opérateurs arithmétiques de base (voir Tableau 3.3).

**Tableau 3.3 Opérateurs arithmétiques**

Opérateur	Signification	Exemple
+	Addition	3 + 4
-	Soustraction	5 - 7
*	Multiplication	5 * 5
/	Division	14 / 7
%	Modulo	20 % 7

Tous ces opérateurs utilisent deux opérandes et sont placés entre ces derniers. L'opérateur de soustraction (-) peut aussi servir à inverser le signe d'un opérande.

Le résultat de la division de deux entiers est un entier : comme les entiers n'ont pas de partie décimale, le reste de la division est ignoré. Par exemple, ce résultat de l'expression  $31 / 9$  est 3.

L'opérateur modulo (%) donne le reste de la division de deux opérandes. Par exemple,  $31 \% 9$  égale 4 (3 fois 9 égale 27, plus 4 pour obtenir 31).

Pour les entiers, le résultat d'une opération est de type `int` ou `long`, indépendamment du type des opérandes (les types `short` ou `byte` sont automatiquement convertis en `int`). Les grands résultats sont de type `long` et les autres de type `int`. Les opérations arithmétiques sur un entier et un nombre à virgule flottante ont pour résultat un nombre à virgule flottante. Les spécifications du langage Java décrivent les mécanismes de manipulation et de conversion des données numériques d'un type vers un autre. Le Listing 3.1 donne un exemple d'arithmétique simple en Java.

### Listing 3.1 Exemple d'opérations arithmétiques simples

```
1: class ArithmeticTest {
2:     public static void main (String[] args) {
3:         short x = 6;
4:         int y = 4;
5:         float a = 12.5f;
6:         float b = 7f;
7:
8:         System.out.println("x is " + x + ", y is " + y);
9:         System.out.println("x + y = " + (x + y));
10:        System.out.println("x - y = " + (x - y));
11:        System.out.println("x / y = " + (x / y));
12:        System.out.println("x % y = " + (x % y));
13:
14:        System.out.println("a is " + a + ", b is " + b);
15:        System.out.println("a / b = " + (a / b));
16:    }
17:
18: }
```

Résultat :

```
x is 6, y is 4
x + y = 10
x - y = 2
x / y = 1
x % y = 2
a is 12.5, b is 7
a / b = 1.78571
```

Dans l'exemple (remarquez la méthode `main()`), les lignes 3 à 6 définissent quatre variables. Les deux premières, `x` et `y`, sont des entiers (type `int`). Les suivantes, `a` et `b`, sont des nombres à virgule flottante (type `float`). Un rappel : le type par défaut des constantes à virgule flottante (comme 12.5) est `double`. Pour obtenir des nombres de type `float`, il faut les faire suivre de la lettre `f` (lignes 5 et 6).

La suite du programme effectue des opérations mathématiques simples avec des entiers et des nombres à virgule flottante. Il affiche ensuite les résultats.

Notez aussi la présence de la méthode `System.out.println()`, utilisée dans les chapitres précédents sans explication. Cette méthode envoie simplement un message vers la sortie standard de votre système (l'écran), une fenêtre particulière ou un fichier log. La destination est fonction du système et de l'environnement de développement utilisés. La méthode reçoit un seul argument (une chaîne), le signe `+` permettant la concaténation de valeurs dans une chaîne.

## Info

*A propos d'opérations mathématiques, signalons deux particularités de Java. La première est la classe `Math`, qui fournit de nombreuses opérations mathématiques courantes comme la racine carrée, la valeur absolue, le cosinus, etc., que vous verrez de plus près au chapitre suivant, "Travailler avec les objets". La seconde particularité est propre à Java 1.1 : les classes `BigDecimal` et `BigInteger`. Ces classes fournissent des mécanismes pour stocker et traiter des nombres extrêmement grands, appelés "bignums", qui peuvent l'être trop pour les types primaires Java standard. Si ce sujet vous intéresse, consultez la documentation API du package `java.math`.*

## Affectations

L'affectation des variables est une forme d'expression qui retourne une valeur. Aussi est-il possible de *chaîner* des variables :

```
x = y = z = 0;
```

Dans cet exemple, les trois variables prennent la valeur `0`.

La partie droite de l'expression d'affectation est toujours évaluée avant l'opération d'affectation. Une expression comme `x = x + 2` est donc traitée ainsi : 2 est ajouté à la valeur de `x`, puis le résultat est affecté à `x`. Ce type d'opération est tellement courant que Java en possède une version abrégée empruntée à C et C++. Le Tableau 3.4 liste ces opérateurs d'affectation combinée.

**Tableau 3.4 Opérateurs d'affectation**

<i>Expression</i>	<i>Signification</i>
<code>x += y</code>	<code>x = x + y</code>
<code>x -= y</code>	<code>x = x - y</code>
<code>x *= y</code>	<code>x = x * y</code>
<code>x /= y</code>	<code>x = x / y</code>

Les opérations d'affectation abrégée et les combinaisons qu'elles sont censées remplacer ne sont pas strictement équivalentes, notamment quand  $x$  et  $y$  sont elles-mêmes des expressions complexes et dans la mesure où le code exploite certains effets de bord de ces expressions complexes. Dans la plupart des cas, cependant, les deux formes sont équivalentes, d'un point de vue fonctionnel. Pour obtenir des informations complémentaires sur l'évaluation des expressions complexes, consultez les spécifications du langage Java.

## Incrémentation et décrémentation

Comme en C ou C++, les opérateurs ++ et -- permettent d'incrémenter ou de décrémenter de 1 la valeur d'une variable. Par exemple,  $x++$  ajoute 1 à la valeur de  $x$  comme le ferait l'expression  $x = x + 1$ . De la même façon,  $x--$  diminue de 1 la valeur de  $x$ . (Contrairement à C et C++, Java permet à  $x$  d'être du type virgule flottante).

Ces deux opérateurs se placent devant ou derrière la valeur à incrémenter ou à décrémenter. Dans le cas d'une expression simple, leur position importe peu. À l'inverse, pour des affectations complexes avec attribution du résultat de l'expression qui incrémente ou décrémente, ce choix fait toute la différence.

Soit les deux expressions suivantes :

- $y = x++;$
- $y = ++x;$

Elles donnent des résultats différents. Quand les opérateurs sont placés après la valeur ( $x++$  ou  $x--$ ),  $y$  prend la valeur de  $x$  avant incrémentation. Avec la notation en préfixe, la valeur de  $x$  est attribuée à  $y$  après incrémentation. Le Listing 3.2 illustre le fonctionnement de ces opérateurs.

**Listing 3.2** Test des opérateurs d'incrément en mode postfixe ou préfixe

```
1: class PrePostFixTest {
2:
3: public static void main (String args[]) {
4:     int x = 0;
5:     int y = 0;
6:
7:     System.out.println("x and y are " + x + " and " + y);
8:     x++;
9:     System.out.println("x++ results in " + x);
10:    ++x;
11:    System.out.println("++x results in " + x);
12:    System.out.println("Resetting x back to 0.");
13:    x = 0;
14:    System.out.println("-----");
15:    y = x++;
16:    System.out.println("y = x++ (postfix) results in:");
17:    System.out.println("x is " + x);
18:    System.out.println("y is " + y);
```

```

19: System.out.println("-----");
20:
21: y = ++x;
22: System.out.println("y = ++x (prefix) results in:");
23: System.out.println("x is " + x);
24: System.out.println("y is " + y);
25: System.out.println("-----");
26:
27: }
28:
29: }

```

Résultat :

```

• x and y are 0 and 0
• x++ results in 1
• ++x results in 2
• Resetting x back to 0.
• -----
• y = x++ (postfix) results in:
• x is 1
• y is 0
• -----
• y = ++x (prefix) results in:
• x is 2
• y is 2
• -----

```

Dans la première partie de l'exemple, la valeur de  $x$  est simplement augmentée avec les notations en postfixe et en préfixe des opérateurs d'incrément. Dans les deux cas, la valeur de  $x$  augmente de 1. Dans une forme simple comme celle-ci, les deux notations donnent le même résultat.

Dans la deuxième partie de l'exemple, l'expression  $y = x++$  utilise la notation en postfixe de l'opérateur d'incrément. La valeur de  $x$  est alors incrémentée *après* son affectation à  $y$ .  $y$  prend donc la valeur initiale de  $x$  (0), qui est ensuite augmentée de 1.

Dans la troisième partie, l'expression  $y = ++x$  utilise la notation en préfixe. La valeur de  $x$  est alors incrémentée avant d'être attribuée à  $y$ . La valeur 1 de  $x$  passe donc à 2, puis elle est attribuée à  $y$ .  $x$  et  $y$  finissent avec la même valeur : 2.

## Info

*D'un point de vue technique, cette description est incorrecte. En réalité, Java évalue toujours entièrement toutes les expressions à droite d'une affectation avant d'attribuer cette valeur à une variable. Affecter  $x$  à  $y$  avant d'incrémenter  $x$  ne représente donc pas la réalité. En fait, Java mémorise la valeur de  $x$ , évalue  $x$  (incrément) puis attribue l'ancienne valeur de  $x$  à  $y$ . Pour la majorité des cas simples, cette distinction a peu d'importance, mais pour les expressions plus complexes, elle affecte le comportement de l'expression complète. Pour obtenir des informations complémentaires sur l'évaluation des expressions dans Java, consultez les spécifications du langage Java.*

## Comparaisons

Java possède plusieurs expressions permettant d'effectuer des comparaisons (voir Tableau 3.5). Chacune d'elles retourne une valeur booléenne (`true` ou `false`).

**Tableau 3.5 Opérateurs de comparaison**

Opérateur	Signification	Exemple
<code>==</code>	Egal	<code>x == 3</code>
<code>!=</code>	Différent	<code>x != 3</code>
<code>&lt;</code>	Inférieur à	<code>x &lt; 3</code>
<code>&gt;</code>	Supérieur à	<code>x &gt; 3</code>
<code>&lt;=</code>	Inférieur ou égal à	<code>x &lt;= 3</code>
<code>&gt;=</code>	Supérieur ou égal à	<code>x &gt;= 3</code>

## Opérations logiques

Les expressions dont le résultat est une valeur booléenne (par exemple, les opérations de comparaison) peuvent être combinées à l'aide d'opérateurs logiques, ET, OU, XOU (ou exclusif) et NON.

Le ET logique est représenté par les opérateurs `&` ou `&&`. L'expression est vraie si les deux opérandes sont également vrais. Si l'un des deux est faux, alors la combinaison est fautive. La différence entre les deux opérateurs réside dans l'évaluation de l'expression. Avec `&`, les deux côtés de l'expression sont évalués indépendamment des résultats obtenus. Avec le `&&`, si la partie gauche de l'expression est fautive, la combinaison est évaluée fautive et la partie droite ignorée.

Le OU est symbolisé par `|` ou `||`. Les expressions OU sont vraies si l'un des deux opérandes est vrai. S'ils sont tous les deux faux, l'expression est évaluée fautive. Comme pour `&` et `&&`, l'opérateur `|` évalue les deux côtés de l'expression indépendamment des résultats. Avec `||`, si la partie gauche de l'expression est vraie, l'expression complète est évaluée vraie et la partie droite ignorée.

Le OU EXCLUSIF `^` s'ajoute à ces deux opérations logiques. Il retourne la valeur `true` (vrai) si les deux opérandes sont différents (l'un vrai et l'autre faux ou *vice versa*). Dans le cas contraire, il retourne la valeur `false` (faux).

En général, les opérateurs utilisés pour de vraies combinaisons logiques sont `&&` et `||`. Les opérateurs `&`, `|` et `^` sont plutôt utilisés pour des opérations logiques au niveau bit.

Le NON logique est représenté par l'opérateur `!`. Il ne concerne qu'un argument de l'expression. La valeur d'une expression NON est la négation de l'expression. Si `x` est vrai, `!x` est faux.

## Opérateurs sur les bits

Pour terminer, voici un court résumé des opérateurs Java au niveau des bits. Ils sont pour la plupart issus des langages C et C++. Ils permettent de réaliser des opérations directement sur les bits des nombres entiers. Ce livre ne traite pas ces opérations. C'est un sujet pointu très bien expliqué dans les manuels de C ou C++. Le Tableau 3.6 présente ces opérateurs.

*Tableau 3.6 Opérateurs sur les bits*

<i>Opérateur</i>	<i>Signification</i>
&	ET bit à bit
	OU bit à bit
^	OU exclusif bit à bit
<<	Décalage à gauche bit à bit
>>	Décalage à droite bit à bit
>>>	Décalage à droite bit à bit avec remplissage par des 0
-	Négation bit à bit
<<=	Décalage à gauche et affectation ( $x = x \ll y$ )
>>=	Décalage à droite et affectation ( $x = x \gg y$ )
>>>=	Décalage à droite, remplissage par des 0 et affectation ( $x = x \ggg y$ )
$x \&= y$	ET affectation ( $x = x \& y$ )
$x \  = \ y$	OU affectation ( $x = x \   \ y$ )
$x \ ^= \ y$	NON affectation ( $x = x \ ^ \ y$ )

## Hierarchie des opérateurs

La hiérarchie des opérateurs indique l'ordre d'évaluation des expressions. Dans certains cas, elle détermine la valeur de l'expression complète. Soit l'expression suivante :

$$y = 6 + 4 / 2$$

La valeur de  $y$  est 5 ou 8, selon la partie de l'expression qui est évaluée en premier ( $6 + 4$  ou  $4 / 2$ ). La hiérarchie des opérateurs impose un ordre de traitement des opérations. Le résultat d'une expression est ainsi prévisible. En général, l'incrémement et la décrémement sont traités avant les opérations arithmétiques, les opérations arithmétiques sont évaluées avant les comparaisons, et les comparaisons avant les expressions logiques. Les affectations sont traitées en dernier.

Le Tableau 3.7 indique la hiérarchie propre aux différents opérateurs de Java. Les opérateurs de la première ligne du tableau sont évalués en premier, ceux de la deuxième en second, etc. Les opérateurs présentés dans une même ligne ont une " priorité " égale et sont évalués de gauche à droite (position dans l'expression). Ce tableau montre que la division est traitée avant l'addition : le résultat de  $y = 6 + 4 / 2$  est donc 8.

**Tableau 3.7** *Ordre de traitement des opérateurs*

<b>Opérateur</b>	<b>Commentaire</b>
<code>. [] ()</code>	Les parenthèses <code>()</code> groupent les expressions. Le point <code>.</code> permet d'accéder aux méthodes et variables dans les classes et les objets (voir le chapitre suivant). <code>[]</code> est utilisé pour les tableaux (cet opérateur sera étudié dans un autre chapitre de la première partie de l'ouvrage).
<code>++ -- ! ~ instanceof</code>	<code>instanceof</code> retourne vrai ( <code>true</code> ) si l'objet appartient à la classe désignée, et faux ( <code>false</code> ) s'il appartient à l'une de ses super-classes (voir chapitre suivant).
<code>new (type) expression</code>	L'opérateur <code>new</code> crée de nouvelles instances de classes. Dans ce cas, les <code>()</code> attribuent un autre <code>type</code> à une valeur (voir le chapitre suivant).
<code>* / %</code>	Multiplication, division, modulo.
<code>+ -</code>	Addition, soustraction.
<code>&lt;&lt; &gt;&gt; &gt;&gt;&gt;</code>	Décalages à gauche et à droite bit à bit.
<code>&lt; &gt; &gt;= &lt;=</code>	Comparaison.
<code>== !=</code>	Egalités.
<code>&amp;</code>	ET.
<code>^</code>	OU exclusif.
<code> </code>	OU.
<code>&amp;&amp;</code>	ET logique.
<code>  </code>	OU logique.
<code>? :</code>	Raccourcis pour <code>if...then...else</code> (voir Chapitre 5).
<code>= += -= *= /= %= ^=</code>	Affectations diverses.
<code>&amp;=  = &lt;&lt;= &gt;&gt;= &gt;&gt;&gt;=</code>	Affectations diverses (suite).

Pour changer l'ordre de traitement, il suffit de mettre entre parenthèses les expressions à évaluer en premier. De plus, les parenthèses s'imbriquent, celles qui se trouvent le plus à l'intérieur étant traitées en premier. L'expression suivante a pour résultat 5, car l'expression `6 + 4` est évaluée d'abord :

```
y = (6 + 4) / 2
```

Les parenthèses sont souvent utiles pour clarifier l'ordre de traitement dans une expression. En fait, elles facilitent la lecture et la compréhension du code. L'ajout de parenthèses n'ayant aucun effet nocif, il n'y a pas à se restreindre, surtout si cet ajout facilite la compréhension de l'évaluation des expressions.



# Arithmétique sur les chaînes

Une particularité de Java est l'utilisation de l'opérateur d'addition (+) pour créer et concaténer des chaînes. Dans la plupart des exemples précédents apparaissaient des lignes comme celle-ci :

```
System.out.println(name + " is a " + color + " beetle");
```

Cette ligne affiche une chaîne de caractères dans laquelle la valeur des variables (ici `name` et `color`) est insérée aux endroits indiqués dans l'instruction.

L'opérateur + employé avec des chaînes ou d'autres objets construit une chaîne unique. C'est le résultat de la concaténation de tous les opérandes. Si l'un des opérandes n'est pas une chaîne, il est automatiquement converti en chaîne. Ceci facilite la création des lignes en sortie de ce type.

## Info

*Un objet ou un type peut être converti en chaîne grâce à la méthode `toString()`. Tout objet possède une représentation chaîne par défaut (le nom de la classe suivi d'accolades), mais les classes redéfinissent généralement `toString()` pour offrir une meilleure représentation en sortie.*

La concaténation des chaînes facilite la construction de lignes comme dans l'exemple précédent. Pour créer une chaîne, il vous suffit d'en réunir les différents composants (descriptions et variables) et d'envoyer le résultat vers la sortie standard, l'écran, un applet, etc.

L'opérateur +=, étudié précédemment, est aussi utilisé dans une chaîne. L'instruction :

```
myName += " Jr. ";
```

est équivalente à :

```
myName = myName + " Jr. ";
```

exactement comme pour les nombres. Dans ce cas, la valeur de `myName` est modifiée (John Smith devient John Smith Jr.).

## Résumé

Un programme Java est constitué essentiellement de classes et d'objets, eux-mêmes constitués de méthodes et de variables. Les méthodes sont construites à l'aide d'instructions et d'expressions étudiées dans ce chapitre. Ce sont les bases de la création de classes et de méthodes. Leur association permet d'obtenir un programme Java complet.

Nous avons donc étudié : les variables, la façon de les déclarer et de leur attribuer une valeur, les constantes pour créer facilement des nombres, les caractères et les chaînes, les opérations arithmétiques, les tests, etc. Cette syntaxe de base permet d'aborder le chapitre suivant, c'est-à-dire d'apprendre à travailler avec les objets et à construire des programmes Java utiles et simples.

Pour terminer, le Tableau 3.8 présente tous les opérateurs vus dans ce chapitre. C'est une référence.

**Tableau 3.8 Résumé des opérateurs**

<i>Opérateur</i>	<i>Signification</i>
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo
<	Inférieur à
>	Supérieur à
<=	Inférieur ou égal à
>=	Supérieur ou égal à
!=	Différent de
&&	ET logique
	OU logique
!	NON logique
&	ET bit à bit
	OU bit à bit
^	OU exclusif bit à bit
<<	Décalage à gauche bit à bit
>>	Décalage à droite bit à bit
>>>	Décalage à droite bit à bit avec remplissage par des 0
~	Négation bit à bit
=	Affectation
++	Incrémententation
--	Décrémententation
+=	Addition et affectation
-=	Soustraction et affectation
*=	Multiplication et affectation
/=	Division et affectation
%=	Modulo et affectation
&=	ET et affectation
=	OU et affectation
<<=	Décalage à gauche et affectation
>>=	Décalage à droite et affectation
>>>=	Décalage à droite, remplissage par des 0 et affectation

## Questions — réponses

**Q N'y a-t-il aucun moyen de définir les constantes ?**

**R** Java interdit la création de constantes locales. Il permet uniquement la création de variables de classe et d'objets constants (voir chapitre suivant).

**Q Que se passe-t-il si la valeur attribuée à une variable de type entier n'appartient pas à l'intervalle des valeurs autorisées pour le type déclaré ?**

**R** La variable n'est pas convertie dans le type adéquat. Il se produit un dépassement de capacité (`overflow`). Un nombre trop grand pour une variable donnée est remplacé par le plus petit nombre négatif du type ; on compte ensuite à nouveau jusqu'à zéro.

Comme cela donne des résultats fantaisistes (et faux), assurez-vous d'avoir déclaré le bon type d'entier. Si un nombre présente un risque de dépassement de capacité, choisissez le type supérieur.

**Q Comment trouver le type d'une variable donnée ?**

**R** Ce n'est pas possible avec les types de base (`int`, `float`, `boolean`). Si vous avez besoin d'un type déterminé, il faut convertir la valeur en question dans le type désiré (voir chapitre suivant).

Avec des types de classe, vous pouvez utiliser l'opérateur `instanceof` (voir chapitre suivant).

**Q Pourquoi y a-t-il tant d'opérateurs raccourcis pour les opérations arithmétiques et les affectations ? Cela complique la lecture du code.**

**R** La syntaxe de Java est basée sur la syntaxe de C++ et donc sur celle de C. Un des objectifs du langage C est de réaliser des choses évoluées avec le minimum de frappe. C'est pour cette raison que les opérateurs raccourcis sont couramment utilisés.

Rien ne vous oblige à les utiliser dans vos programmes. Si vous pensez gagner en lisibilité du code, choisissez la forme longue.

**Q Vous avez traité les opérations mathématiques simples dans la section consacrée aux opérateurs. Puis-je supposer que Java a d'autres moyens pour faire des opérations mathématiques plus complexes ?**

**R** En effet, une classe spéciale du package `java.lang`, appelée `java.lang.Math`, comporte un certain nombre de méthodes pour le calcul exponentiel, la trigonométrie et d'autres opérations mathématiques de base. En fait, comme on peut appeler ces méthodes en utilisant la classe `Math` elle-même, ce sont des exemples types de méthodes de classe. Vous en apprendrez plus sur ce sujet au chapitre suivant.

4

# Travailler avec les objets



Tout d'abord, une évidence : Java étant un langage orienté objet, de nombreux objets seront traités dans ce chapitre. Il faut les créer, les mettre à jour, les déplacer, modifier leurs variables, appeler leurs méthodes, les combiner entre eux et, bien sûr, développer des classes.

Ce chapitre couvre l'étude de l'objet Java dans son milieu naturel, au travers des sujets suivants :

- Création d'instances de classes
- Test et modification des variables d'instance et de classe dans une nouvelle instance
- Appel de méthodes dans un objet
- Conversion d'objets et d'autres types de données d'une classe vers une autre
- Compléments divers sur le travail avec des objets
- Vue d'ensemble des bibliothèques de classes Java

## Création de nouveaux objets

L'écriture d'un programme Java passe par la définition de classes, qui sont des modèles pour les objets (voir Chapitre 2). Généralement, l'utilisation des classes se résume à la création et à la manipulation des objets. Cette section explique comment créer un nouvel objet dans une classe donnée.

L'utilisation d'une constante chaîne (une série de caractères entre guillemets doubles) crée un nouvel objet de la classe `String` avec la valeur de cette chaîne (voir chapitre précédent).

Cette caractéristique de la classe `String` est exceptionnelle : une constante suffit à la création d'un de ses objets, bien que ce soit une classe. Ce n'est pas le cas pour les autres classes. Pour créer leurs objets, il faut impérativement utiliser l'opérateur `new`.

### Info

*A l'inverse des constantes chaîne, les constantes des nombres et des caractères ne créent pas d'objet. Leurs types de données primaires définissent des nombres et des caractères, mais l'efficacité veut qu'ils ne soient pas des objets réels. Pour les traiter comme des objets, il faut les entourer d'objets-écrans. C'est l'un des thèmes de ce chapitre, dans la section "Conversion d'objets et de types de données primaires".*

## Opérateur new

Les objets sont créés au moyen de l'opérateur `new`. Pour cela, `new` doit être suivi du nom de la classe dans laquelle l'objet est créé, et de parenthèses. Les exemples suivants créent de nouvelles instances des classes `String`, `Random` et `Motorcycle`, et stockent ces nouvelles instances dans des variables de types adéquats :

- `String str = new String();`
- `Random r = new Random();`
- `Motorcycle m2 = new Motorcycle();`

Les parenthèses sont importantes, ne les oubliez pas. Quand elles sont vides, comme dans l'exemple ci-avant, un objet basique est créé. Si elles contiennent des arguments, ceux-ci déterminent les valeurs initiales des variables d'instance, ou toute autre qualité initiale de l'objet :

- `Date dt = new Date(90, 4, 1, 4, 30);`
- `Point pt = new Point(0,0);`

Le nombre et le type des arguments de `new` à l'intérieur des parenthèses sont définis par la classe elle-même, au moyen d'une méthode spéciale appelée constructeur (nous reviendrons sur les constructeurs au cours du présent chapitre). Si vous essayez de créer une nouvelle instance d'une classe avec un mauvais nombre ou un mauvais type d'arguments (ou si vous omettez les arguments alors qu'ils sont nécessaires), vous obtiendrez une erreur à la compilation. Voici un exemple de création de différents types d'objets utilisant des nombres et des types d'arguments différents. La classe `Date`, qui fait partie du package `java.util`, crée des objets qui représentent la date courante. Le Listing 4.1 décrit trois façons différentes de créer un objet `Date` avec l'opérateur `new`.

#### *Listing 4.1 Programme Date de Laura*

```
1: import java.util.Date;
2:
3: class CreateDates {
4:
5:     public static void main (String args[]) {
6:         Date d1, d2, d3;
7:
8:         d1 = new Date();
9:         System.out.println("Date 1: " + d1);
10:
11:         d2 = new Date(71, 7, 1, 7, 30);
12:         System.out.println("Date 2: " + d2);
13:
14:         d3 = new Date("April 3 1993 3:24 PM");
15:         System.out.println("Date 3: " + d3);
16:     }
17: }
```

*Résultat :*

- `Date 1: Sun Nov 26 19:10:56 PST 1995`
- `Date 2: Sun Aug 01 07:30:00 PDT 1971`
- `Date 3: Sat Apr 03 15:24:00 PST 1993`

Dans cet exemple, trois objets `Date` différents sont créés grâce aux différents arguments transmis à la classe indiquée après `new`. Pour le premier objet (ligne 8), l'opérateur `new` ne prend pas d'argument. Le résultat (voir la première ligne de sortie standard) est un objet `Date` avec la date du jour.

Le second objet `Date` contient cinq arguments entiers. Ils représentent une date : année, mois, jour, heures et minutes. Le résultat est un objet `Date` avec : `Sun Aug 01 07:30:00 PDT 1971`.

## Attention

*Java numérote les mois en commençant par 0. Ainsi, bien que vous vous attendiez à ce que le 7ème mois soit juillet, en Java il s'agit en fait du mois d'août.*

La troisième version de `Date` reçoit un unique argument : une date sous forme de chaîne de caractères. La chaîne est analysée et l'objet `Date` est créé (voir troisième ligne du résultat). Une chaîne date prend de nombreuses formes différentes, décrites dans la documentation API fournie avec le package `java.util`.

## Opérations réalisées par `new`

La programmation de l'opérateur `new` déclenche une série d'actions. D'abord, un nouvel objet d'une classe donnée est créé, et de la mémoire lui est allouée. Puis, une méthode particulière, définie dans la classe concernée, est appelée. Cette méthode est un constructeur. Les constructeurs sont des méthodes spéciales, définies dans les classes, qui servent à créer et à initialiser de nouvelles instances de ces classes.

Les constructeurs initialisent l'objet et ses variables et créent tous les autres objets dont cet objet peut lui-même avoir besoin ; ils effectuent en fait toutes les autres opérations dont le nouvel objet a besoin pour être initialisé.

Différentes définitions de constructeurs dans une classe peuvent avoir différents nombres ou types d'arguments. Ainsi, le constructeur appelé correspond aux arguments passés à `new`. Cela explique l'exemple précédent : à chacune des versions de `new` correspond un résultat différent.

Pour obtenir différents comportements d'une classe, il suffit de définir autant de constructeurs que l'on souhaite de comportements différents pour cette classe (voir Chapitre 7).

## Gestion de la mémoire

Dans Java, la gestion de la mémoire est dynamique et automatique. Java alloue automatiquement la mémoire nécessaire à chaque nouvel objet créé ; aucune action explicite n'est nécessaire.

L'opération inverse est elle aussi automatique. Lorsque vous n'avez plus besoin d'un objet, vous attribuez de nouvelles valeurs à toutes les variables qui pourraient le contenir, et vous l'éliminez de tous les tableaux où il figurait ; l'objet devient ainsi inutilisable. Java possède un programme récupérateur de mémoire, parfois appelé aussi ramasse-miettes. Ce programme recherche les objets inutilisés et libère la mémoire qu'ils occupent. Il n'est donc pas nécessaire de libérer explicitement la mémoire, il suffit de vous assurer que vous ne maintenez pas quelque part un objet dont vous voulez vous débarrasser. Le Chapitre 21, traite du rôle et du fonctionnement du programme récupérateur de mémoire.

Ce programme est un outil spécial de l'environnement Java recherchant les objets inutilisés. Lorsqu'il en trouve, il les élimine et libère la mémoire qu'ils utilisaient.

## Variables d'instance et de classe : accès, fixation d'une valeur

L'objet obtenu à ce stade peut contenir des variables d'instance ou de classe. Travailler avec ces dernières est aisé. Elles se comportent comme les variables locales du chapitre précédent. Seul le mode d'accès est légèrement différent.

### Accès aux valeurs

Pour obtenir la valeur d'une variable d'instance, on se sert d'une expression conforme à la *notation point*.

Avec la *notation point*, la référence à une variable d'instance ou de classe est composée de deux parties : l'objet, à gauche du point, et la variable, à droite.

La notation point sert à référencer, au moyen d'une expression, des variables d'instance et des méthodes d'instance figurant dans un objet.

Soit un objet attribué à la variable `myObject`. S'il possède une variable appelée `var`, vous vous référez à la valeur de cette variable de la manière suivante :

```
myObject.var;
```

Cette forme d'accès aux variables est une expression (elle retourne une valeur) et les deux parties séparées par le point peuvent également être des expressions. Il est donc possible d'imbriquer des accès aux variables d'instance. Si `var` contient elle-même un objet ayant sa propre variable d'instance appelée `state`, la valeur de cette dernière s'obtient ainsi :

```
myObject.var.state;
```

Les expressions à points sont évaluées de gauche à droite. Dans l'exemple, la variable `var` de `myObject` est traitée en premier. Comme elle pointe sur un autre objet comportant la variable `state`, celle-ci est ensuite prise en compte. Le résultat final est la valeur de la variable `state`, après évaluation de l'expression complète.

### Mise à jour des valeurs

Pour attribuer une valeur à une variable, il suffit d'ajouter l'opérateur d'affectation à sa droite :

```
myObject.var.state = true;
```



L'exemple de programme présenté dans le Listing 4.2 teste et modifie les variables d'instance d'un objet `Point`. `Point` appartient au package `java.awt` et référence les coordonnées d'un point à l'aide des valeurs `x` et `y`.

#### Listing 4.2. La classe `TestPoint`

```
1: import java.awt.Point;
2:
3: class TestPoint {
4:     public static void main(String args[]) {
5:         Point thePoint = new Point(10,10);
6:
7:         System.out.println("X is " + thePoint.x);
8:         System.out.println("Y is " + thePoint.y);
9:
10:        System.out.println("Setting X to 5.");
11:        thePoint.x = 5;
12:        System.out.println("Setting Y to 15.");
13:        thePoint.y = 15;
14:
15:        System.out.println("X is " + thePoint.x);
16:        System.out.println("Y is " + thePoint.y);
17:
18:    }
19: }
```

Résultat :

```
X is 10
Y is 10
Setting X to 5.
Setting Y to 15.
X is 5
Y is 15
```

Dans cet exemple, l'objet `Point` est créé avec la valeur `10` pour `x` et `y` (ligne 5). Les lignes 7 et 8 les affichent à l'aide de la notation `point`. Les lignes 10 à 13 remplacent les valeurs de ces variables par `5` et `15`. Enfin, les lignes 15 et 16 affichent à nouveau les valeurs de `x` et `y` pour preuve de la mise à jour.

## Variables de classe

Les variables de classe sont définies et stockées dans la classe elle-même. Leurs valeurs s'appliquent à la classe et à tous ses objets.

Le fonctionnement des variables d'instance diffère de celui des variables de classe. Pour les premières, chaque nouvelle instance de classe récupère une copie des variables d'instance définies par cette classe. Chaque instance met ainsi à jour la valeur de ces variables. Il n'y a pas d'inter-

férence avec une autre instance. Pour les secondes, il existe une seule copie de la variable. Chaque instance de classe y accède, mais il n'y a qu'une valeur. Quand elle est modifiée, elle l'est pour toutes les instances de la classe.

Les variables de classe sont définies avec le mot clé `static` placé devant la variable. Ce sujet est traité en détail dans le Chapitre 6. Soit cet extrait d'une définition de classe :

```
• class FamilyMember {  
•     static String surname = "Johnson";  
•     String name;  
•     int age;  
•     ...  
• }
```

Les instances de la classe `FamilyMember` ont chacune leurs propres valeurs pour `name` et `age`. A l'inverse, la variable de classe `surname` a une seule valeur pour tous les membres de la famille. Si la valeur de `surname` est modifiée, toutes les instances de `FamilyMember` le sont aussi.

L'accès aux variables de classe et d'instance utilise la même notation. Pour accéder ou modifier la valeur d'une variable de classe, l'objet ou le nom de la classe doit précéder le point. Les deux lignes affichées par le code suivant sont identiques :

```
• FamilyMember dad = new FamilyMember();  
• System.out.println("Family's surname is: " + dad.surname);  
• System.out.println("Family's surname is: " + FamilyMember.surname);
```

Avec la possibilité de modifier la valeur d'une variable de classe à l'aide d'une instance, il est facile de se perdre entre les variables de classe et l'origine de leurs valeurs. C'est pourquoi il vaut mieux référencer une variable de classe à l'aide du nom de la classe. Le code est alors compréhensible et les résultats fantaisistes plus faciles à corriger.

## Appel des méthodes

L'appel d'une méthode est similaire à la référence aux variables d'instance d'un objet : les appels de méthodes dans des objets utilisent aussi la notation point. L'objet est placé à gauche du point, la méthode et ses arguments à droite :

```
myObject.methodOne (arg1, arg2, arg3);
```

Toutes les méthodes doivent être suivies de parenthèses, même si elles n'ont pas d'arguments :

```
myObject.methodNoArgs();
```

Si la méthode appelée est un objet qui contient aussi des méthodes, ces dernières s'imbriquent, comme pour les variables. L'exemple suivant appelle la méthode `getName`, définie dans l'objet retourné par la méthode `getClass`, elle-même définie dans `myObject`.

```
myObject.getClass().getName();
```

Les appels imbriqués de méthodes peuvent se combiner avec des références à des variables d'instance également imbriquées. Dans l'exemple suivant, vous appelez la méthode `methodTwo()`, définie dans l'objet stocké par la variable d'instance `var`, qui fait elle-même partie de l'objet `myObject`.

```
myObject.var.methodTwo(arg1, arg2);
```

La méthode `System.out.println()` est un bon exemple de variables et de méthodes imbriquées. La classe `System` (du package `java.lang`) décrit les comportements propres au système. `System.out` est une variable de classe. Elle contient une instance de la classe `PrintStream` qui pointe sur la sortie standard du système. Les instances de `PrintStream` ont une méthode `println()` qui envoie une chaîne dans le flux de sortie.

Le Listing 4.3 présente un exemple d'appels de méthodes définies dans la classe `String`. Celle-ci comprend des méthodes pour les tests et les modifications de chaînes.

#### Listing 4.3 Exemples d'utilisation des méthodes de `String`

```
1: class TestString {
2:
3:     public static void main (String args[]) {
4:         String str = "Now is the winter of our discontent";
5:
6:         System.out.println("The string is : " + str);
7:         System.out.println("Length of this string : "
8:             + str.length());
9:         System.out.println("The character at position 5 : "
10:            + str.charAt(5));
11:        System.out.println("The substring from 11 to 18 : "
12:            + str.substring(11, 18));
13:        System.out.println("The index of the character d : "
14:            + str.indexOf('d'));
15:        System.out.print("The index of the beginning of the ");
16:        System.out.println("substring \"winter\":"
17:            + str.indexOf("winter"));
18:        System.out.println("The string in upper case : "
19:            + str.toUpperCase());
20:    }
21: }
```

Résultat :

```
• The string is: Now is the winter of our discontent
• Length of this string: 35
• The character at position 5: s
• The substring from positions 11 to 18: winter
• The index of the character d: 25
• The index of the beginning of the substring "winter": 11
• The string in upper case: NOW IS THE WINTER OF OUR DISCONTENT
```

Une nouvelle instance de `String` est créée en ligne 4 à l'aide d'une constante chaîne (c'est plus facile que de passer par `new` et de rentrer les caractères individuellement). La suite du programme appelle simplement différentes méthodes de `String` pour effectuer des opérations sur la chaîne :

- La ligne 6 affiche la valeur de la chaîne créée en ligne 4 : "Now is the winter of our discontent".
- La ligne 7 appelle la méthode `length()` dans le nouvel objet `String`. Cette chaîne a une longueur de 35 caractères.
- La ligne 9 appelle la méthode `charAt()`. Elle retourne le caractère situé à la position donnée dans la chaîne. Comme la première position est 0, le caractère en position 5 est `s`.
- La ligne 11 appelle la méthode `substring()`. Elle reçoit deux entiers représentant un intervalle et retourne la sous-chaîne des caractères situés dans cet intervalle. La méthode `substring()` peut aussi être appelée avec un seul argument. Elle retourne alors la sous-chaîne qui commence à cette position et se termine en fin de chaîne.
- La ligne 13 appelle la méthode `indexOf()`. Elle retourne la position de la première occurrence du caractère donné (ici `'d'`).
- La ligne 15 montre un autre usage de la méthode `indexOf()`. Elle reçoit un argument chaîne et retourne l'index du début de cette chaîne.
- La ligne 19 utilise la méthode `toUpperCase()`. Elle retourne une copie de la chaîne en majuscules.

## Méthodes de classe

Comme les variables de classe, les méthodes de classe s'appliquent à la classe tout entière et non à ses instances. Elles sont souvent utilisées pour des méthodes à vocation générale. Ces dernières n'opèrent pas directement sur une instance de la classe, même s'il y a adéquation d'un point de vue conceptuel. Par exemple, la classe `String` contient une méthode de classe appelée `valueOf()` qui reçoit l'un des nombreux types d'argument (entiers, booléens, autres objets, etc.). `valueOf()` retourne ensuite une instance de `String()`, contenant la valeur chaîne de l'argument transmis. Cette méthode n'opère pas directement sur une instance de `String` existante. Cependant, récupérer une chaîne à partir d'un autre objet ou type de données est bien une opération de style chaîne. Elle a donc toute sa place dans la classe `String`.

Les méthodes de classe servent aussi à grouper au même endroit, dans une classe, des méthodes à caractère général. Par exemple, la classe `Math` définie dans le package `java.lang` contient de nombreuses méthodes de classe. Ce sont des opérations mathématiques. Il n'existe pas d'instance de la classe `Math`, mais on peut utiliser ses méthodes avec des arguments numériques ou booléens. Par exemple, la méthode de classe `Math.max()` prend deux arguments et retourne le

plus grand des deux. Il n'est pas nécessaire de créer une nouvelle instance de `Math` ; il suffit d'appeler la méthode là où on en a besoin, comme suit :

```
in biggerOne = Math.max(x,y);
```

L'appel des méthodes de classe utilise la notation point, comme l'appel des méthodes d'instance. La classe ou l'une de ses instances est à gauche du point. Cette syntaxe est identique à celle employée pour les variables de classe. Cependant, l'emploi du nom de la classe facilite la lecture du code, pour les raisons déjà évoquées avec les variables de classe. Les deux dernières lignes de l'exemple suivant produisent le même résultat (la chaîne "5") :

```
• String s, s2;  
• s = "foo";  
• s2 = s.valueOf(5);  
• s2 = String.valueOf(5);
```

## Références aux objets

Même si elles sont occultées, les références aux objets manipulés sont importantes. Lorsque des objets sont attribués à des variables ou passés en arguments à une méthode, ce sont leurs références qui sont transmises. En effet, ni les objets eux-mêmes, ni leur copie ne sont passés.

Le Listing 4.4 est un exemple simple illustrant le fonctionnement des références.

### *Listing 4.4. Exemple de références*

```
• 1: import java.awt.Point;  
• 2:  
• 3: class ReferencesTest {  
• 4:     public static void main (String args[] ) {  
• 5:         Point pt1, pt2;  
• 6:         pt1 = new Point(100, 100);  
• 7:         pt2 = pt1;  
• 8:  
• 9:         pt1.x = 200;  
• 10:        pt1.y = 200;  
• 11:        System.out.println("Point1: " + pt1.x + ", " + pt1.y);  
• 12:        System.out.println("Point2: " + pt2.x + ", " + pt2.y);  
• 13:    }  
• 14: }
```

*Résultat :*

```
• Point1: 200, 200  
• Point2: 200, 200
```

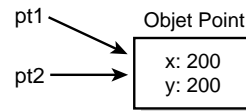
Dans la première partie de ce programme, on déclare deux variables de type `Point` (ligne 5), on crée un nouvel objet `Point` affecté à `pt1` (ligne 6), et, enfin, on affecte à `pt2` la valeur de `pt1` (ligne 7).

La question est : après la mise à jour des variables d'instance `x` et `y` de `pt1` par les lignes 9 et 10, que devient `pt2` ?

Comme vous pouvez le voir, les variables d'instance `x` et `y` de `pt2` ont également été modifiées, bien qu'on ne l'ait pas fait explicitement. L'affectation de la valeur de `pt1` à `pt2` a véritablement créé à partir de `pt2` une référence à l'objet pointé par `pt1`, comme on peut le voir sur la Figure 4.1. Quand l'objet référencé par `pt2` est modifié, celui sur lequel pointe `pt1` l'est également, parce qu'il s'agit de deux références au même objet.

**Figure 4.1**

*Références à des objets.*



## Astuce

*Pour que `pt1` et `pt2` pointent sur des objets distincts, il faut coder `new Point()` sur chacune des deux lignes pour créer deux objets distincts.*

L'utilisation des références prend toute son importance lorsque des arguments sont transmis aux méthodes, comme vous le verrez plus loin dans ce même chapitre.

## Info

*Java n'autorise ni les pointeurs explicites, ni les opérations arithmétiques sur les pointeurs, utilisés dans les langages de type C. Vous disposez uniquement de références, mais celles-ci comportent la plupart des possibilités des pointeurs, sans entraîner la confusion et les bogues insidieux résultant de l'utilisation des pointeurs explicites.*

# Conversion de types de données primaires et d'objets

Parfois, certaines valeurs sont stockées dans un type qui ne convient pas pour ce que vous voulez en faire. Cela peut être l'instance d'une mauvaise classe, un nombre à virgule flottante (`float`) au lieu d'un entier (`int`), ou un entier au lieu d'une chaîne. Le mécanisme consistant à transformer la valeur d'un type vers un autre est une conversion.

La conversion est un mécanisme qui, à partir d'un objet ou d'une valeur d'un certain type, crée un autre objet ou une autre valeur de type différent. Le résultat de la conversion est un nouvel objet ou une nouvelle valeur. La conversion n'affecte pas l'objet ou la valeur d'origine.

Le concept de conversion est simple, mais les règles spécifiant quels types peuvent être convertis dans d'autres types sont compliquées en Java par l'existence de types primaires d'une part (`int`, `float`, `boolean`), et de types d'objets d'autre part (`string`, `Point`, `Window`, etc.). Trois formes de conversions sont abordées dans cette section :

- Conversion entre types primaires : un entier (`int`) vers un nombre à virgule flottante (`float`) ou un `float` vers un `double`.
- Conversion entre types d'objet : une instance de classe vers celle d'une autre classe.
- Conversion de types primaires en objets et vice versa.

## Conversion des types primaires

La conversion des types primaires consiste à convertir la valeur d'un type primaire dans un autre. Un exemple : l'affectation d'un nombre de type `x` à une variable de type `y`. Cette opération est fréquente sur des types numériques. Les valeurs booléennes ne peuvent pas être converties en d'autres types primaires.

Les conversions vers un type plus "grand" que celui de la valeur à transformer peuvent ne pas être explicites. Vous pouvez le plus souvent traiter automatiquement (conversion implicite) un type `byte` ou `character` comme un `long`, un `int` ou un `float`, et n'importe lequel des types précédents comme un `double`. Dans la plupart des cas, le type le plus grand apporte une précision supérieure et aucune information n'est perdue lors de la conversion. Toutefois, la conversion d'entiers à valeurs virgule flottante fait exception à cette règle générale ; la conversion d'un `int` ou d'un `long` en un `float`, ou d'un `long` en un `double`, peut entraîner une perte de précision.

Une conversion d'une valeur dans un type plus "petit" que le sien doit être explicite, la conversion pouvant entraîner une perte de précision. La conversion explicite utilise le format suivant :

```
(typename) value
```

Dans cette instruction, `typename` est le nom du type dans lequel `value` (une expression) doit être convertie (en `short`, `int`, `float`, `boolean`, par exemple). L'expression suivante divise `x` par `y` et convertit le résultat en type `int` :

```
(int) (x / y);
```

La conversion est prioritaire sur les opérations arithmétiques, d'où l'emploi des parenthèses dans notre exemple, pour que la conversion porte bien sur le résultat de la division. Autrement, la valeur de `x` serait d'abord convertie, puis divisée par `y`, ce qui pourrait donner un résultat différent.

## Conversion des objets

Les instances d'une classe peuvent aussi être converties en instances d'autres classes, à une condition : les classes des objets à convertir et convertis doivent être liées par le mécanisme d'héritage. Autrement dit, la conversion d'un objet n'est possible que vers une instance d'une sous-classe ou d'une super-classe de sa propre classe.

D'une manière analogue à la conversion de valeurs vers un type plus "grand", la conversion de certains objets n'est pas obligatoirement explicite. En effet, les instances de sous-classes contiennent généralement toutes les informations des instances des super-classes. Il est donc possible d'utiliser une instance d'une sous-classe là où l'on s'attend à utiliser une instance d'une de ses super-classes. Par exemple, si une méthode reçoit deux arguments, le premier de type `Object` et le second de type `Number`, lui passer une instance de ces classes particulières n'est pas obligatoire. En effet, pour l'argument de type `Object`, une sous-classe quelconque d'`Object` (donc tout objet) suffit. De même, pour l'argument de type `Number`, il suffit de passer une instance d'une sous-classe de `Number` (`Integer`, `Boolean`, `Float`, etc.). Il n'est pas nécessaire de faire d'abord une conversion explicite.

La conversion vers le bas dans la hiérarchie des classes est automatique, mais la conversion vers le haut ne l'est pas. La conversion d'un objet dans une instance d'une de ses super-classes entraîne une perte des informations fournies par la sous-classe et doit, de ce fait, être explicite. La conversion d'un objet vers une autre classe s'effectue par la même opération que celle utilisée pour les types de données primaires :

```
(classname) object
```

Dans cette instruction, `classname` est le nom de la classe dans laquelle `object` (une référence à un objet) doit être converti. La conversion crée une référence, du type `classname`, à l'ancien objet. L'ancien objet continue d'exister sans subir de changement.

Voici un exemple de conversion d'une instance de la classe `GreenApple` vers la classe `Apple` (où `GreenApple` est une sous-classe de `Apple`, contenant des informations supplémentaires pour définir la pomme comme verte) :

```
• GreenApple a;  
• Apple a2;  
• a = new GreenApple();  
• a2 = (Apple) a;
```

La conversion des objets peut s'effectuer non seulement vers des classes, mais aussi vers des interfaces. La conversion n'est possible que si la classe de l'objet, ou l'une de ses super-classes, implémente effectivement l'interface. Ce type de conversion autorise l'appel d'une des méthodes de l'interface, même si cette dernière n'est pas effectivement implémentée par la classe de l'objet. Les interfaces sont traitées plus en détail dans la troisième partie de ce livre.

## Transformation de types primaires en objets et vice versa

Vous avez vu comment convertir un type de données primaire vers un autre, et comment effectuer des conversions entre classes. Peut-il y avoir des conversions d'un type primaire vers une classe ou dans l'autre sens ?

La réponse est non. Les types de données primaires et les objets sont deux choses très différentes en Java. Par conséquent, il n'est pas possible de convertir l'une en l'autre. Cependant, le package



`java.lang` contient des classes spéciales qui correspondent à chaque type de données primaire : `Integer` pour les entiers, `Float` pour les nombres à virgule flottante, `Boolean` pour les booléens, etc. Notez que les noms de classes commencent par une lettre majuscule, et les types de données primaires par une minuscule. Java traite ces noms de manière très différente, aussi ne les confondez pas, car vos variables et vos méthodes ne se comporteraient pas comme prévue.

Grâce aux méthodes de classe définies dans ces classes, on peut créer un équivalent objet pour tous les types de base, à l'aide de l'opérateur `new`. La ligne de code suivante crée une instance de la classe `Integer` avec la valeur 35 :

```
Integer intObject = new Integer(35);
```

Disposant d'objets réels, vous pouvez traiter ces valeurs comme des objets. Ensuite, s'il s'avère nécessaire de revenir aux valeurs de type primaire, vous disposez de méthodes pour ce faire. Par exemple, `intValue()` extrait une valeur de type `int` d'un objet `Integer` :

```
int theInt = intObject.intValue() ; // retourne 35
```

Consultez la documentation *Java API* relative à ces classes particulières pour plus de précisions sur la conversion des types de données primaires en objets et inversement.

## Info

*Java 1.0 fournit les classes spéciales suivantes correspondant aux types primaires : `Boolean`, `Character`, `Double`, `Float`, `Integer` et `Long`. Java 1.1 comporte en outre les classes `Byte` et `short`, ainsi qu'une classe enveloppe spéciale, `Void`. Ces dernières classes sont principalement utilisées pour la réflexion des objets.*

## Compléments divers

Cette section regroupe diverses informations complémentaires sur la manipulation des objets. En particulier :

- Comparaison d'objets.
- Identification de la classe d'un objet.
- Test d'un objet pour déterminer s'il s'agit d'une instance d'une classe donnée.

### Comparaison des objets

Le chapitre précédent fournit une liste d'opérateurs destinés à la comparaison de valeurs : égal, différent, inférieur, etc. La plupart d'entre eux ne marchent qu'avec des types de données primaires, et pas avec des objets. Si leurs opérandes ne sont pas des nombres, le compilateur Java produit des erreurs.

Les opérateurs `==` (égal) et `!=` (différent) font exception à cette règle : utilisés avec des objets, ils vérifient les deux opérandes font référence au même objet en mémoire.

## Info

Pour obtenir un résultat significatif lors de la comparaison d'objets d'une classe, il faut créer des méthodes spéciales dans la classe et les appeler.

*Dans Java, le polymorphisme n'existe pas pour les opérateurs, ce qui veut dire qu'il n'est pas possible de redéfinir le comportement des opérateurs en utilisant des méthodes dans les classes. Les opérateurs prévus dans Java sont définis uniquement pour les nombres.*

La classe `String` illustre bien ce qu'est la comparaison d'objets. Elle permet d'avoir deux chaînes (deux objets indépendants) en mémoire avec les mêmes valeurs, c'est-à-dire les mêmes caractères placés dans le même ordre. Vis-à-vis de l'opérateur `==`, ces deux objets `String` ne sont pas égaux, car, même si leur contenu est identique, ils ne représentent pas le même objet.

Cependant, la classe `String` possède une méthode appelée `equals()`. Cette méthode teste chaque caractère de deux chaînes et retourne la valeur `true` s'il y a égalité parfaite (voir Listing 4.5).

### Listing 4.5 Test d'égalité des chaînes

```
1: class EqualsTest {
2: public static void main(String args[]) {
3:     String str1, str2;
4:     str1 = "les chemises de l'archiduchesse sont sèches.";
5:     str2 = str1;
6:
7:     System.out.println("String1: " + str1);
8:     System.out.println("String2: " + str2);
9:     System.out.println("Same object? " + (str1 == str2));
10:
11:    str2 = new String(str1);
12:
13:    System.out.println("String1: " + str1);
14:    System.out.println("String2: " + str2);
15:    System.out.println("Same object? " + (str1 == str2));
16:    System.out.println("Same value? " + str1.equals(str2));
17: }
18: }
```

#### Résultat :

```
String1: les chemises de l'archiduchesse sont sèches.
String2: les chemises de l'archiduchesse sont sèches.
Same object? true
String1: les chemises de l'archiduchesse sont sèches.
String2: les chemises de l'archiduchesse sont sèches.
Same object? false
Same value? true
```

La première partie du programme (lignes 3 à 5) déclare deux variables (`str1` et `str2`) et affecte la constante `Les chemises de l'archiduchesse sont sèches` à `str1` puis la valeur de `str1` à `str2`. Comme nous l'avons vu précédemment lors de l'étude des références d'objets, `str1` et `str2` pointent maintenant sur le même objet et le test de la ligne 9 le confirme.

Dans la seconde partie, `new` crée un nouvel objet `String` ayant la même valeur que `str1`, et affecte `str2` à ce nouvel objet de type chaîne. Il y a maintenant deux objets `String` différents (leurs références sont différentes) avec la même valeur. La ligne 15 les teste à l'aide de l'opérateur `==` pour savoir s'il s'agit du même objet. Comme prévu, la réponse est non. De même, le test d'égalité sur leurs valeurs, effectué avec la méthode `equals()` en ligne 16, retourne `true` : elles ont la même valeur.

## Info

*Pourquoi ne pas utiliser une autre constante au lieu de `new` lors de la modification de `str2` ? Les constantes chaîne (`String`) sont optimisées dans Java. Ainsi, la création d'une chaîne via une constante, puis l'utilisation d'une autre constante avec les mêmes caractères, déclenchent l'optimisation. Java redonne le premier objet `String`. Les deux chaînes représentent alors le même objet. Pour créer deux objets différents, il faut employer une technique différente.*

## Identification de la classe d'un objet

Voici comment identifier la classe d'un objet attribué à la variable `obj` :

```
String name = obj.getClass().getName() ;
```

`getClass()` est une méthode de la classe `Object`. Elle est donc disponible pour tous les objets. De l'appel de cette méthode résulte un objet de la classe `Class`. Il contient une méthode appelée `getName()` qui retourne le nom de la classe sous forme de chaîne.

L'opérateur `instanceof` permet aussi de réaliser des tests. Il a deux opérands : un objet, à gauche, et le nom d'une classe, à droite. L'expression retourne `true` si l'objet appartient à la classe désignée ou à l'une de ses sous-classes, et `false` dans le cas contraire :

```
• "foo" instanceof String // true
• Point pt = new Point(10,10);
• pt instanceof String // false
```

L'opérateur `instanceof` s'utilise aussi avec les interfaces. Si un objet implémente une interface, `instanceof`, avec le nom de l'interface à droite, retourne la valeur `true`. Les interfaces sont traitées dans la troisième partie de ce livre.

# Inspection des classes et des méthodes par la réflexion

L'une des améliorations apportées au langage Java par la diffusion 1.1 est l'introduction de la réflexion (en anglais, reflection), aussi appelée introspection. La réflexion permet à une classe Java, telle qu'un programme que vous écrivez, d'obtenir des précisions sur n'importe quelle autre classe.

Grâce à la réflexion, un programme Java peut charger une classe dont il ne sait rien, en découvrir les variables, les méthodes et les constructeurs, et travailler avec ces éléments.

Un exemple permettra de voir d'emblée ce dont il s'agit. Le Listing 4.6 montre une brève application appelée `seeMethods`.

**Listing 4.6** Texte intégral de `SeeMethods.java`

```
1: import java.lang.reflect.*;
2: import java.util.Random;
3:
4: class SeeMethods {
5:     public static void main(String[] argumentss) {
6:         Random rd = new Random();
7:         Class className = rd.getClass();
8:         Method[] methods = className.getMethods();
9:         for (int i = 0; i < methods.length; i++) {
10:             System.out.println("Method: " + methods[i]);
11:         }
12:     }
13: }
```

Ce programme utilise le groupe de classes `java.lang.reflect.*`, qui fournit des informations sur les attributs, les méthodes et les constructeurs de n'importe quelle classe.

L'application `SeeMethods` crée un objet `Random` à la ligne 6, puis utilise la réflexion pour afficher toutes les méthodes *publiques* faisant partie de la classe. Le Listing 4.7 montre la sortie de cette application.

**Listing 4.7.** La sortie de l'application `SeeMethods`

```
1: Method: public final native java.lang.Class java.lang.Object.getClass()
2: Method: public native int java.lang.Object.hashCode()
3: Method: public boolean java.lang.Object.equals(java.lang.Object)
4: Method: public java.lang.String java.lang.Object.toString()
5: Method: public final native void java.lang.Object.notify()
6: Method: public final native void java.lang.Object.notifyAll()
```

```
● 7: Method: public final native void java.lang.Object.wait(long)
● 8: Method: public final void java.lang.Object.wait(long,int)
● 9: Method: public final void java.lang.Object.wait()
● 10: Method: public synchronized void java.util.Random.setSeed(long)
● 11: Method: public void java.util.Random.nextBytes(byte[])
● 12: Method: public int java.util.Random.nextInt()
● 13: Method: public long java.util.Random.nextLong()
● 14: Method: public float java.util.Random.nextFloat()
● 15: Method: public double java.util.Random.nextDouble()
● 16: Method: public synchronized double java.util.Random.nextGaussian()
```

En utilisant la réflexion, l'application `SeeMethods` peut connaître chacune des méthodes de la classe `Random`, ainsi que toutes les méthodes qu'elle a hérité de ses super-classes. Chaque ligne du listing donne les informations suivantes sur une méthode :

- Méthode `public` ou non
- Type d'objet ou de variable retourné par la méthode
- Méthode de la classe ou d'une de ses super-classes
- Nom de la méthode
- Types des objets ou des variables utilisés comme arguments lors de l'appel de la méthode

L'application `SeeMethods` pourrait être utilisée avec n'importe quelle classe d'objets. Il suffit de changer la ligne 6 de `SeeMethods.java` pour créer un objet différent et examiner ses mécanismes internes.

La réflexion est surtout utilisée par des outils servant à parcourir des classes, ou par des débogueurs, pour obtenir plus d'informations sur les classes ou les objets examinés ou en cours de mise au point. On s'en sert aussi avec `JavaBeans`, pour s'informer sur un autre objet et découvrir ce qu'il peut faire (en vue de lui demander de faire quelque chose), ce qui facilite la construction d'applications d'une bonne envergure. Le Chapitre 14 donne plus de précisions sur `JavaBeans`.

Le package `java.lang.reflect` comporte les classes suivantes :

- `Field`, pour obtenir et gérer l'information sur les variables de classe et d'instance.
- `Method`, pour les méthodes de classe et les méthodes d'instance.
- `Constructor`, pour les méthodes spéciales servant à la création de nouvelles instances des classes.
- `Array`, pour les tableaux.
- `Modifier`, pour décoder les informations de modification relative aux classes, aux variables et aux méthodes (les modificateurs sont traités au Chapitre 15).

Un certain nombre de nouvelles méthodes seront en outre disponibles au sein d'une classe appelée `Class`, pour faire le lien entre les diverses classes de réflexion.

La réflexion est une fonction avancée que vous n'utiliserez peut-être pas tout de suite dans vos programmes. Elle présente un grand intérêt pour la sérialisation des objets, `JavaBeans`, ou d'autres types de programmes sophistiqués.

# Bibliothèque de classes Java

La bibliothèque de classes Java comprend l'ensemble des classes dont la fourniture est assurée pour tout environnement Java de caractère commercial (par exemple, tout environnement de développement Java ou tout navigateur comme celui de Netscape). Ces classes se trouvent dans le package Java. Toutes celles qui ont été étudiées jusqu'à présent en font partie, comme d'ailleurs un grand nombre d'autres classes présentées plus loin dans cet ouvrage, et d'autres encore que vous ne connaîtrez peut-être jamais.

Le JDK est livré avec une documentation sur la totalité de la bibliothèque de classes Java. Cette documentation décrit les variables d'instance de chaque classe, les méthodes, les constructeurs, les interfaces, etc. Vous pouvez vous la procurer (elle s'appelle Java Application Programmer's Interface, ou API) *via* le Web à <http://java.sun.com:80/products/JDK/CurrentRelease/api/packages.html>. Un court résumé de l'API Java se trouve en Annexe C. Parcourir la bibliothèque de classes Java avec ses méthodes et ses variables d'instance est un bon moyen de savoir ce que Java peut ou ne peut pas faire, et de voir de quelle manière ces éléments peuvent servir de point de départ de vos propres développements.

Les packages suivants font partie de la bibliothèque de classes Java :

- `java.lang` : ses classes concernent le langage lui-même. `Object`, `String` et `System` en font partie. Ce package contient aussi des classes spéciales destinées à la représentation des types de données primaires (`Integer`, `Character`, `Float`, etc.). La première partie du présent ouvrage donne au moins un aperçu de la plupart des classes de ce package.
- `java.util` : ces classes sont des utilitaires, comme `Date`, ou de simples classes de collections, comme `Vector` et `Hashtable`.
- `java.io` : ces classes écrivent ou lisent dans les flux d'entrées/sorties (comme l'entrée ou la sortie standard), ou encore, gèrent les fichiers. Le Chapitre 19, est consacré aux classes de ce package.
- `java.net` : ce sont des classes de gestion de réseau, comme `Socket` et `URL` (une classe permettant de référencer des documents sur le World wide Web). Le Chapitre 14 donne quelques indications supplémentaires sur les réseaux.
- `java.awt` (c'est l'Abstract Windowing Toolkit, ou *jeu d'outils de fenêtrage abstrait*) : ces classes permettent d'implémenter une interface utilisateur graphique. `Window`, `Menu`, `Button`, `Font`, `CheckBox`, etc., en font partie. Ce package contient aussi des mécanismes de gestion des événements système, et des classes de traitement d'images (le package `java.awt.Image`). AWT est traité dans la seconde partie de l'ouvrage.
- `java.applet` : ces classes servent à implémenter des applets Java.

Outre les classes de Java, l'environnement de développement peut en contenir d'autres, fournissant d'autres utilitaires ou des fonctions différentes. Elles sont certainement très utiles, mais n'appartiennent pas à la bibliothèque Java standard. D'autres personnes essayant de faire marcher

vos programmes Java risquent de ne pas en disposer, à moins que vous n'incluez explicitement ces classes dans vos programmes. Ceci est particulièrement important dans le cas des applets, destinées à tourner sur de nombreuses plates-formes à l'aide d'un programme de navigation compatible Java. Seules les classes contenues dans le package Java sont toujours disponibles sur tous les programmes de navigation et dans tous les environnements Java.

## Résumé

Des objets, toujours des objets. Ce chapitre a traité de leur manipulation : création, recherche et mise à jour des valeurs de leurs variables, appel de leurs méthodes. Les mécanismes de copie, de comparaison et de conversion d'objets ont eux aussi été étudiés. Enfin, vous avez découvert la bibliothèque de classes de Java (qui constitue une matière première importante pour programmer).

Vous disposez désormais des bases pour travailler avec les composants les plus simples du langage. Il vous reste à étudier les tableaux, les instructions de contrôle et les boucles. C'est le sujet du prochain chapitre. Au Chapitre 6, vous apprendrez ensuite à définir et à utiliser des classes au sein des applications Java, et dans la seconde partie de l'ouvrage, vous déboucherez directement sur les applets. La quasi-totalité de la programmation avec Java se résume au concept d'objet.

## Questions — réponses

- Q J'ai du mal à faire la différence entre les objets et les types de données primaires comme `int` et `boolean`.**
- R** Les types de données primaires (`byte`, `short`, `int`, `long`, `float`, `double` et `char`) sont les plus petites entités du langage. Ce ne sont pas des objets, même s'ils peuvent, de diverses manières, être traités comme tel. Par exemple, il est possible de les affecter à des variables et de les passer aux méthodes comme arguments, ou de les obtenir en retour. Malgré tout, la plupart des opérations portant sur les objets ne marchent pas avec les types de données primaires.
- Les objets sont des instances d'une classe et, par conséquent, des types de données bien plus complexes que de simples nombres ou caractères. Ils contiennent souvent des nombres et des caractères comme variables d'instance ou de classe.
- Q Pas de pointeur en Java ? Sans pointeurs, comment pouvez-vous travailler avec des listes liées, dans lesquelles un pointeur d'une liste correspond à une autre liste et permet de les parcourir ?**

- R** Java n'est pas totalement dépourvu de pointeurs ; il n'a pas de pointeur explicite. Les références aux objets sont, en fait, des pointeurs. Ainsi, pour créer une entité du type liste liée, vous créez une classe appelée `Node`, qui aura une variable d'instance également du type `Node`. Pour lier ensemble les objets `Node`, il vous suffit alors d'affecter un objet `Node` à la variable d'instance de l'objet qui le précède immédiatement dans la liste. Comme les références aux objets sont des pointeurs, les listes liées agencées de cette manière se comporteront comme vous pouvez le souhaiter.
- Q** **Dans la section traitant de l'appel des méthodes, une méthode était appelée avec un nombre d'arguments différent à chaque fois, et cela donnait des résultats variés. Comment est-ce possible ?**
- R** Ce concept s'appelle le polymorphisme. Pour un même nom de fonction, il existe divers comportements. Ils dépendent des arguments transmis au moment de l'appel. Les arguments varient en nombre et/ou en type. Les méthodes sont créées dans les classes avec des signatures distinctes et des définitions différentes. Lorsque la méthode est appelée, Java recherche la définition correspondant au nombre et aux types des arguments transmis.
- Ce sujet est traité au Chapitre 6.
- Q** **Java étant basé sur C++, pourquoi le polymorphisme ne s'applique-t-il pas aux opérateurs ?**
- R** Java est effectivement basé sur le langage C++, mais il a également été conçu pour être simple. C'est pourquoi de nombreuses caractéristiques de C++ ont disparu. Redéfinir les opérateurs complique leur manipulation et la lecture du code. Les concepteurs de Java ont donc choisi d'abandonner cette caractéristique de C++.