Fundamentals of the Java Programming Language, Java SE 6

Electronic Presentation

SL-110-SE6-FR Rev. E.1

D61796FR10
Edition 1.0
D61796FR10_EP

ORACLE

Sommaire du cours

Préface

À propos de ce cours

Objectifs du cours

À la fin de ce cours, vous serez en mesure d'effectuer les opérations suivantes :

- Bien connaître la technologie Java™, le langage de programmation Java et le cycle de vie du produit
- Utiliser les diverses constructions du langage de programmation Java pour créer plusieurs applications Java
- Utiliser les méthodes et les constructions de boucles et de conditions pour imposer un flux de programme
- Implémenter les concepts intermédiaires de programmation Java et orientésobjet (OO) dans les programmes Java

Plan du cours

Présentation de la programmation Java

- Fonctionnement de la technologie Java™
- Analyse d'un problème et conception d'une solution
- Développement et test d'un programme Java

Notions élémentaires de la programmation Java

- Déclaration, initialisation et utilisation des variables
- Création et utilisation des objets

Contrôle du flux des programmes

- Utilisation des opérateurs et des constructions à boucles
- Utilisation des constructions à boucles
- Développement et utilisation des méthodes

Description des concepts intermédiaires Java et OO

- Implémentation de l'encapsulation et des constructeurs
- Création et utilisation des tableaux
- Implémentation de l'héritage

Sujets non traités

- Programmation Java avancée : cours SL-275 :
  Le langage de programmation Java™
- Analyse et conception OO avancée : cours OO-226 :
  Analyse et conception d'applications orientées objet pour la technologie Java™ (UML)
- Programmation des applets ou conception de pages Web

Êtes-vous suffisamment préparé ?

Pour vérifier que vous êtes prêt à suivre ce cours, pouvez-vous répondre par l'affirmative aux questions suivantes ?

- Savez-vous créer et modifier des fichiers texte dans un éditeur de texte ?
- Savez-vous utiliser un navigateur Web (WWW) ?
- Savez-vous résoudre des problèmes logiques ?

Introductions

- Nom
- Société
- Poste, fonction et responsabilités
- Votre expérience concernant les sujets présentés dans ce cours
- Motif de votre inscription à ce cours
- Ce que vous attendez de ce cours

Icônes

Démonstration

Débat

Note

Avertissement - Électricité

Avertissement - Chaleur

Icônes

Étude de cas

Auto-évaluation

Conventions typographiques

- La police `Courier` est utilisée pour les noms des commandes, des fichiers et des répertoires, le code et les constructions de programmation, ainsi que les résultats affichés à l'écran.
- La police **`Courier gras`** est utilisée pour les caractères et les nombres saisis et pour chaque ligne de code référencée dans une description textuelle.
- La police *`Courier italique`* est utilisée pour les variables et les paramètres substituables de ligne de commande qui sont remplacés par un nom ou une valeur réel(le).

Conventions typographiques

- La police *`Courier italique gras`* est utilisée pour représenter les variables dont les valeurs doivent être saisies par le participant dans le cadre d'une activité.
- La police *Palatino italique* est utilisée pour les titres du manuel, les nouveaux termes ou les mots qui doivent attirer l'attention.

Autres conventions

Les exemples de langage de programmation Java utilisent les conventions supplémentaires suivantes :

- La police `Courier` est utilisée pour les noms de classe, les méthodes et les mots clés.
- Les noms de méthode ne sont pas suivis de parenthèses, à moins qu'une liste de paramètres formels ou réels n'apparaisse.
- Les sauts de ligne sont utilisés en cas de séparations, de conjonctions ou d'espace dans le code.
- Si une commande du système d'exploitation Solaris™ (Solaris OS) diffère de celle utilisée dans la plate-forme Microsoft Windows, les deux commandes sont présentées.

Module 1

Fonctionnement de la technologie Java™

Objectifs

- Décrire les principaux concepts du langage de programmation Java
- Énumérer les trois groupes de produits de la technologie Java
- Résumer chacune des sept phases du cycle de vie du produit

Pertinence

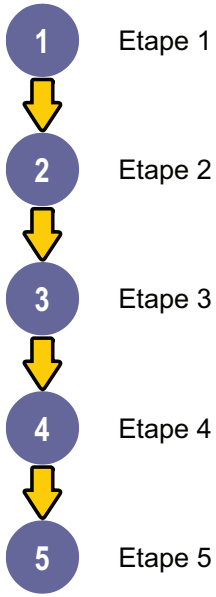
- Quelle est la définition des termes suivants ?
  - Sécurisé
  - Orienté objet
  - Indépendant
  - Dépendant
  - Distribué
- Quelles sont les différentes phases impliquées dans la construction d'un objet, par exemple une maison ou un meuble ?

Principaux concepts du langage de programmation Java

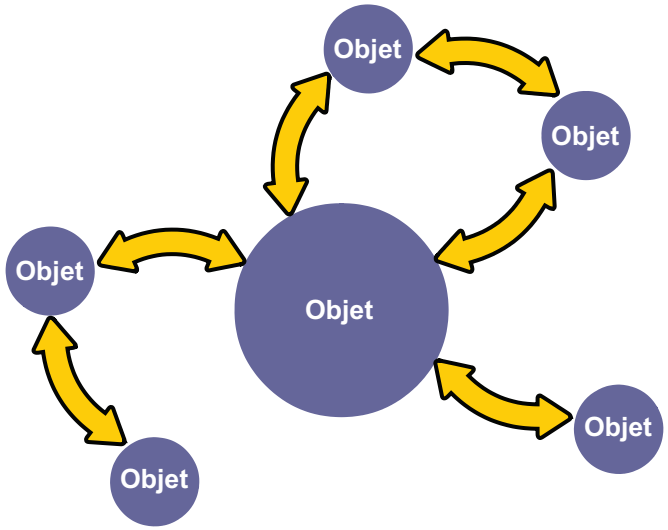
- Orienté objet
- Distribué
- Simple
- À multithread
- Sécurisé
- Indépendant de la plate-forme

Programmation procédurale

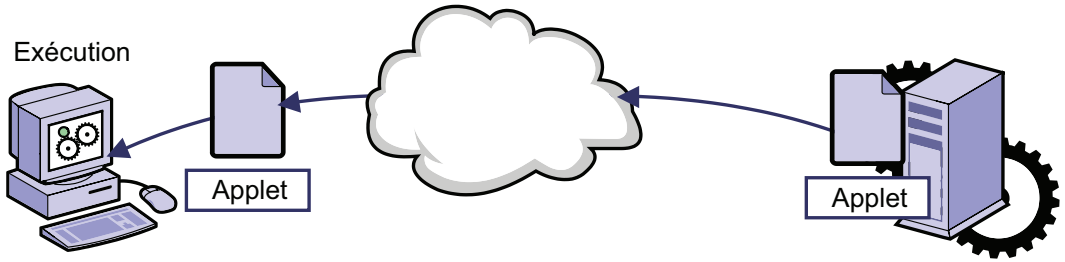
L'objectif de la programmation procédurale est séquentiel.



Orienté objet



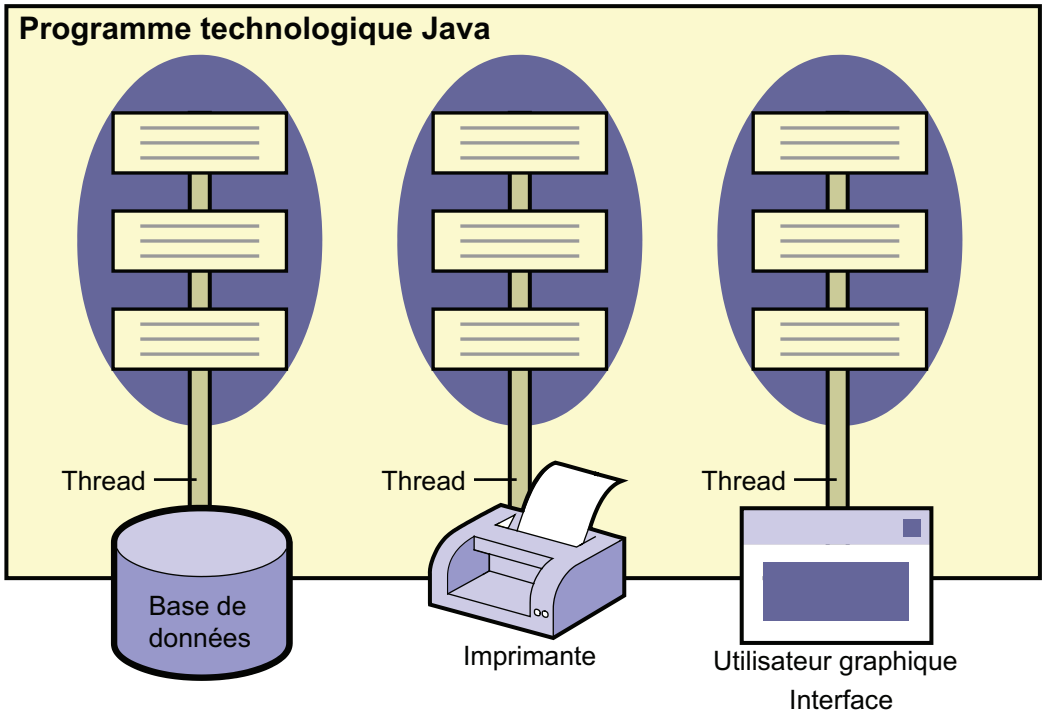
Distribué



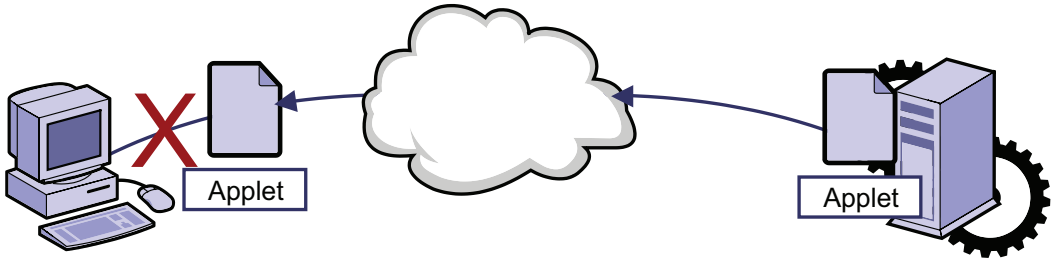
Simple

- Des références sont utilisées à la place des pointeurs mémoire.
- Un type de données `boolean` peut avoir comme valeur `true` ou `false`.
- La gestion de la mémoire est automatique.

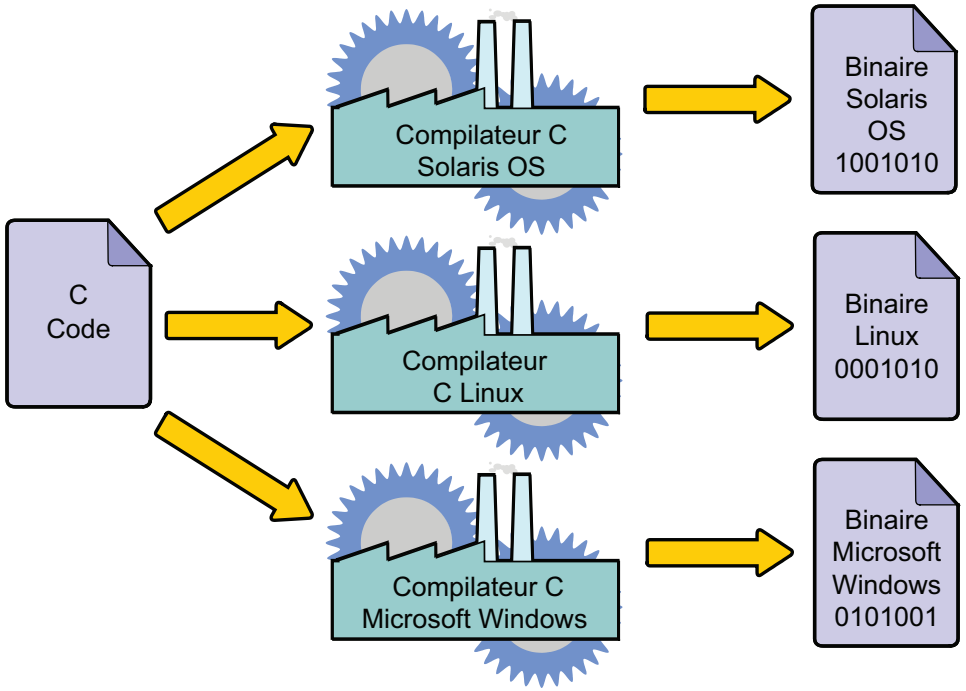
À multithread



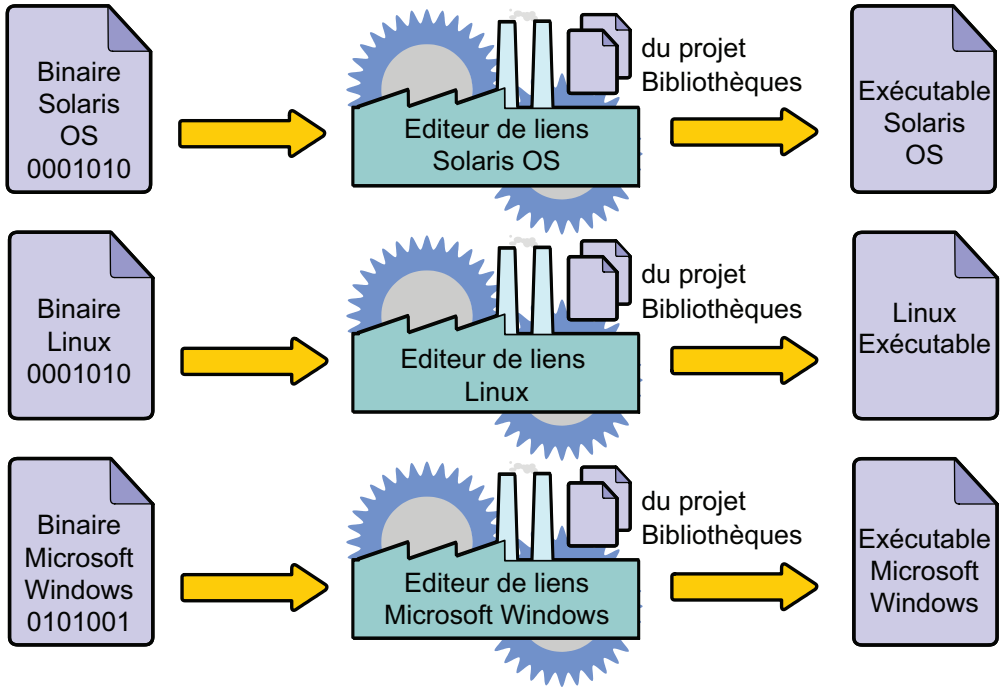
Sécurisé



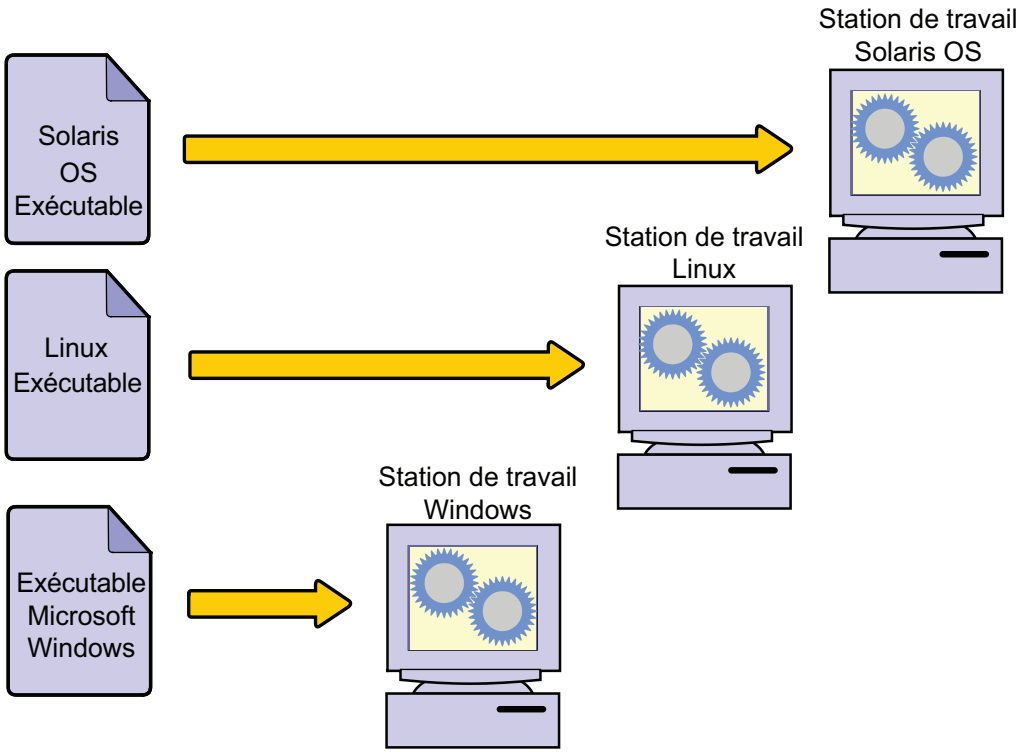
Programmes dépendants de la plate-forme



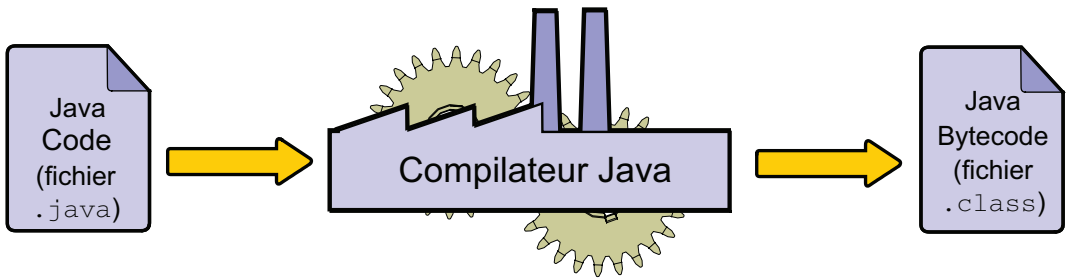
Programmes dépendants de la plate-forme



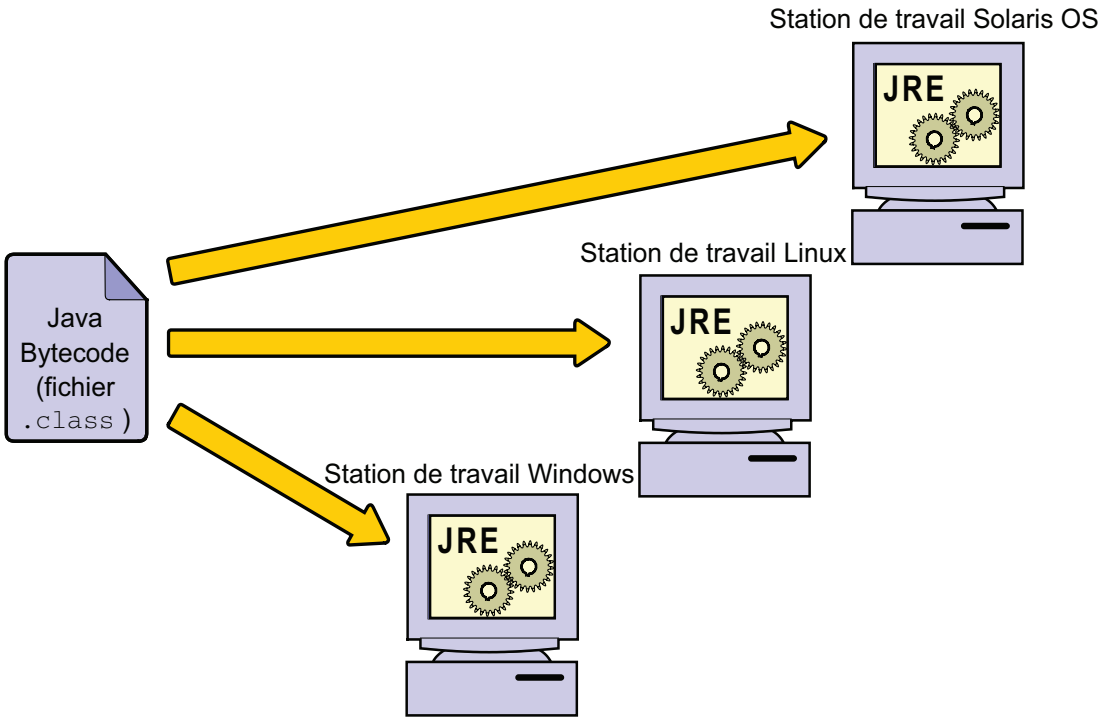
Programmes dépendants de la plate-forme



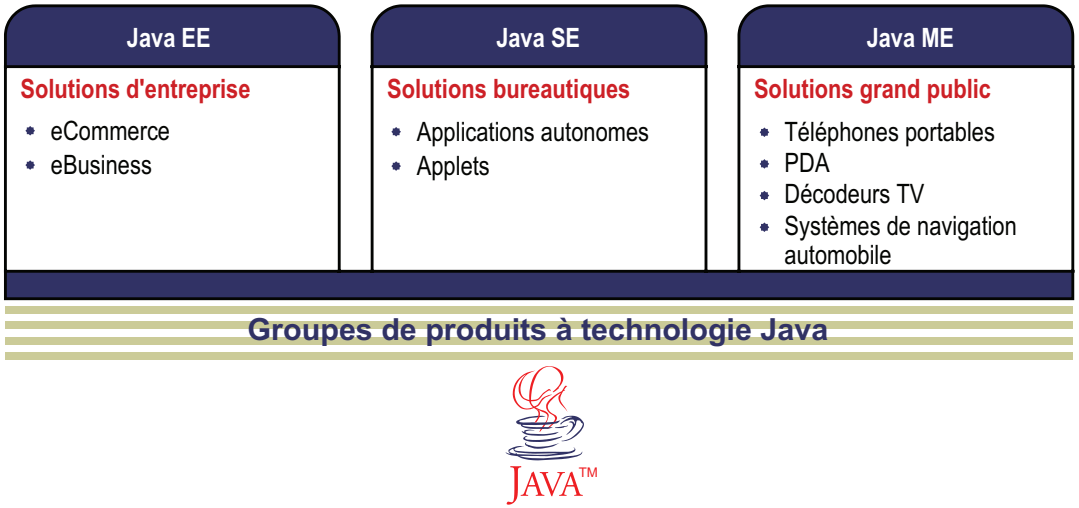
Programmes indépendants de la plate-forme



Programmes indépendants de la plate-forme



Identification des groupes de produits de technologie Java



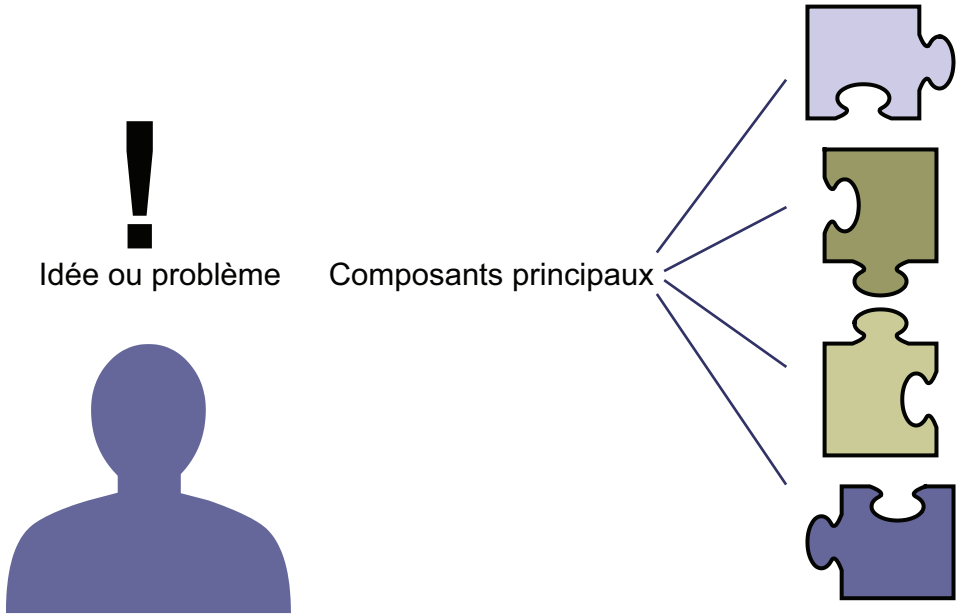
Utilisation des composants de Java Platform, Standard Edition SDK

- Environnement d'exécution Java (JRE) :
  - Une machine virtuelle Java (JVM™) pour la plate-forme choisie
  - Des bibliothèques de classes Java pour la plate-forme choisie
- Un compilateur Java
- La documentation de la bibliothèque de classes Java (API) (téléchargement distinct)
- Des utilitaires supplémentaires, par exemple pour créer des fichiers d'archive Java (JAR) et déboguer les programmes de technologie Java
- Des exemples de programmes de technologie Java

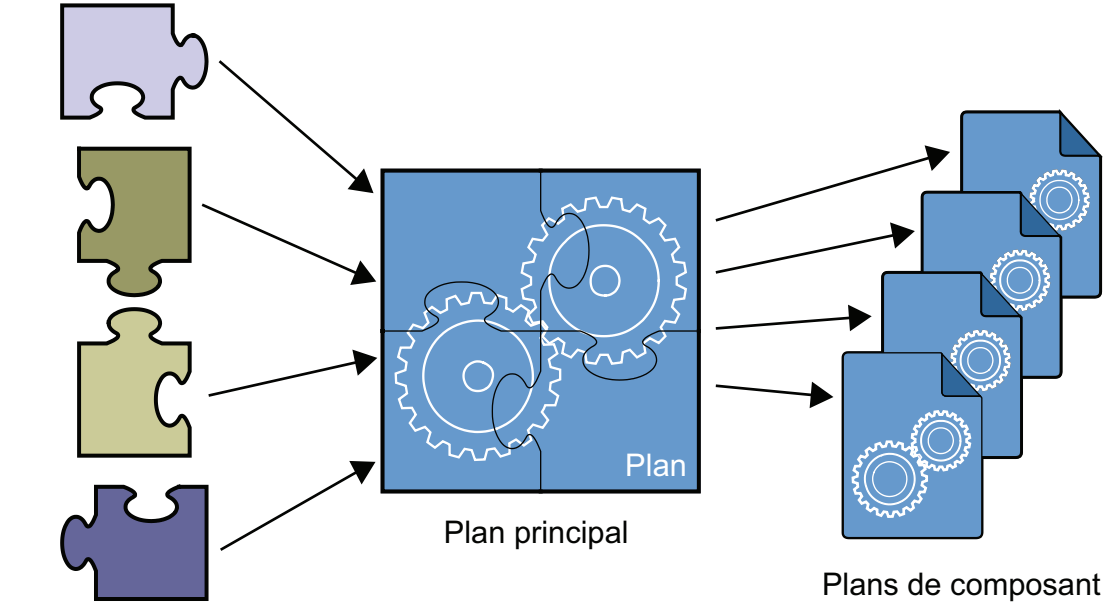
Phases du cycle de vie du produit

1. Analyse
2. Conception
3. Développement
4. Test
5. Implémentation
6. Maintenance
7. Fin de vie (EOL, End-of-life)

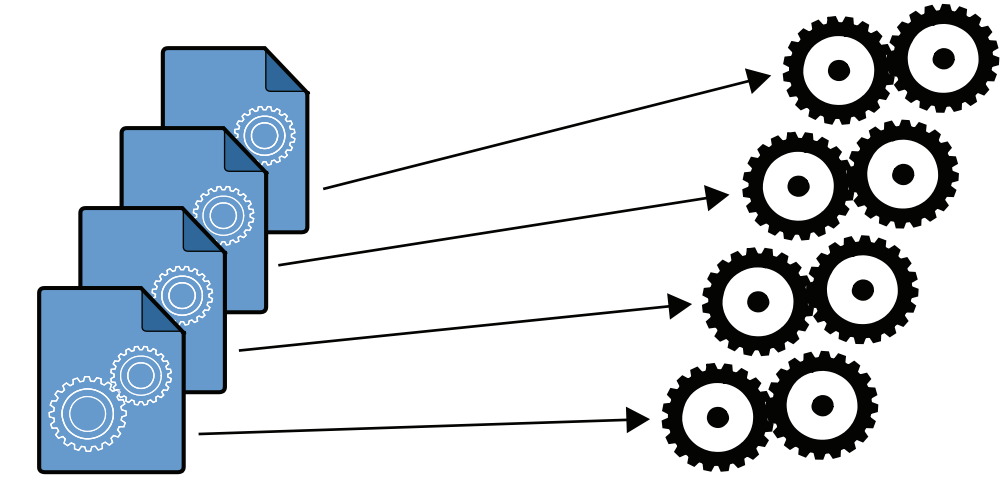
Phase de l'analyse



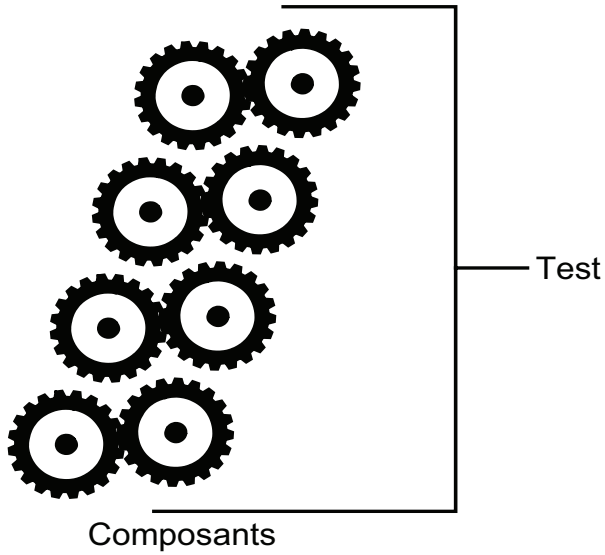
Phase de conception



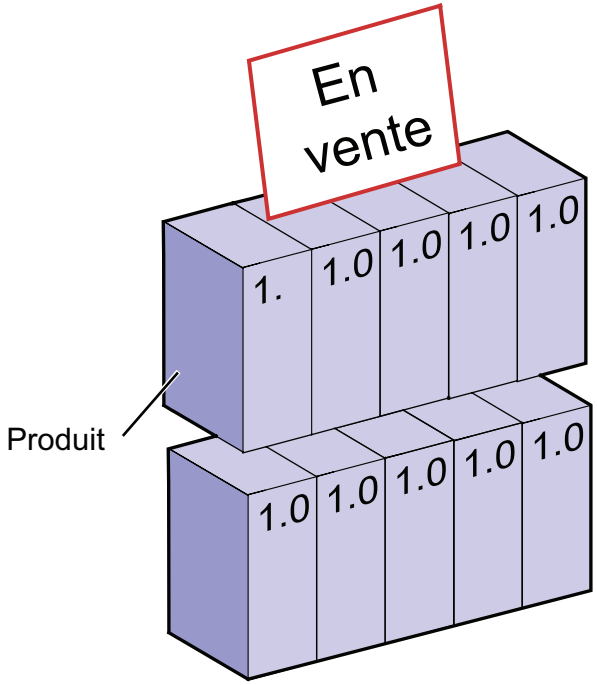
Phase de développement



Phase de test

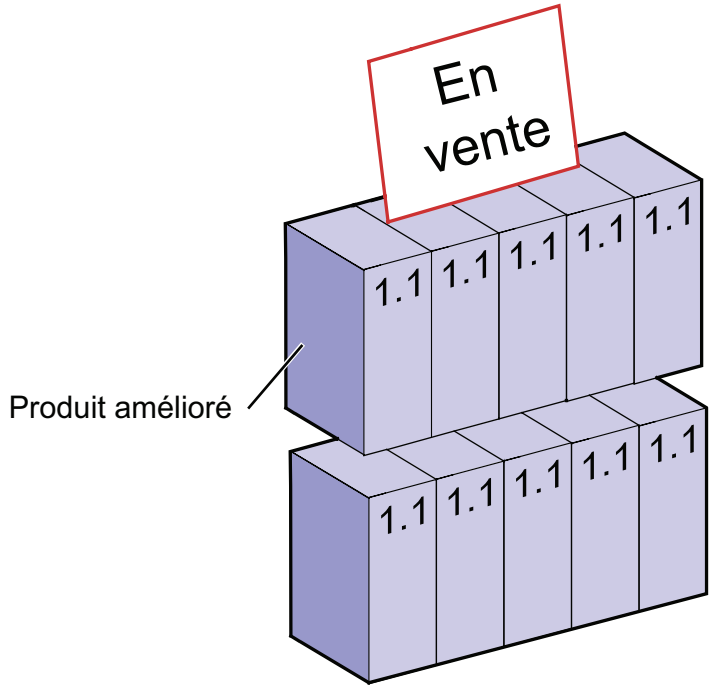


Phase d'implémentation

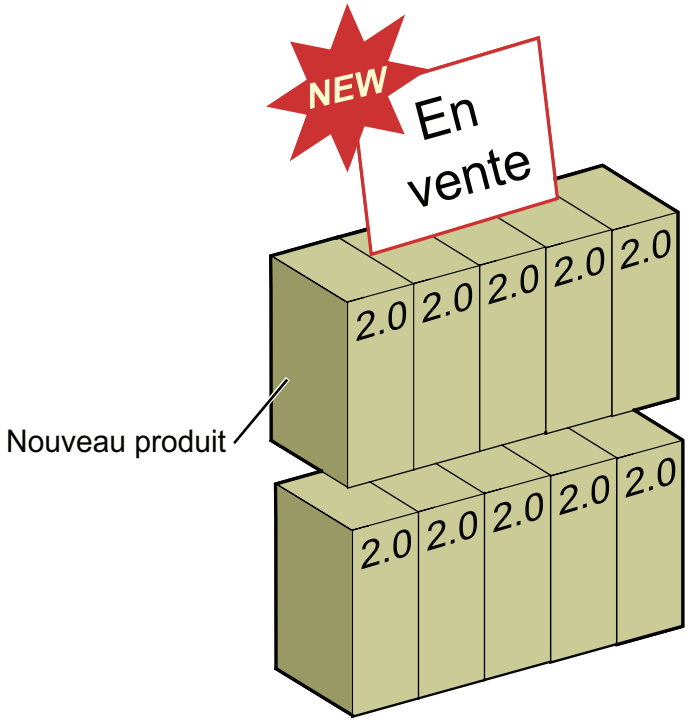


L'implémentation se rapporte à l'expédition d'un produit de telle sorte que les clients puissent l'acquérir.

Phase de maintenance



Phase de fin de vie



Module 2

Analyse d'un problème et conception d'une solution

Objectifs

- Analyser un problème à l'aide de l'Analyse orientée objet (AOO)
- Concevoir des classes à partir desquelles des objets seront créés

Pertinence

- Comment identifiez-vous les composants nécessaires pour une construction, par exemple pour construire une maison ou un meuble ?
- Qu'est-ce que la taxonomie ?
- Comment les organismes sont-ils reliés dans une taxonomie ?
- Quelle est la différence entre attributs et valeurs ?

Analyse d'un problème à l'aide d'OOA

La société DirectClothing, Inc. vend des chemises depuis son catalogue. Le chiffre d'affaires augmente de 30 pour-cent par an et un nouveau système de saisie des commandes est nécessaire.

- DirectClothing publie un catalogue de vêtements tous les six mois et l'envoie à ses abonnés. Chaque chemise du catalogue possède un identifiant d'article (ID), une ou plusieurs couleurs (associées chacune à un code), une ou plusieurs tailles, une description et un prix.
- DirectClothing accepte les paiements par chèques et par l'ensemble des cartes bancaires.
- Les clients peuvent appeler DirectClothing et passer commande directement auprès d'un conseiller du service clientèle (CSR) ou envoyer un formulaire de commande par courrier ou télécopie.

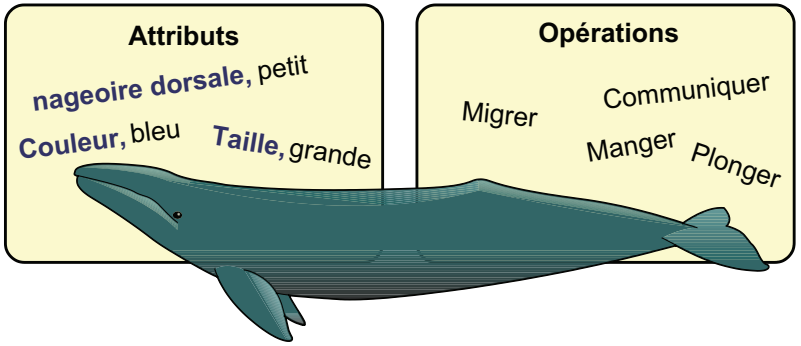
Identification du domaine problématique

- Un domaine problématique est l'étendue du problème à résoudre.
- Par exemple, « Créer un système qui permette au personnel chargé de la saisie des commandes d'entrer et d'accepter le paiement pour une commande. »

Identification des objets

- Un objet peut être physique ou abstrait.
- Les objets ont des *attributs* (caractéristiques), par exemple une taille, un nom, une forme, etc.
- Les objets ont des *opérations (*les actions qu'ils peuvent faire), par exemple définir une valeur, afficher un écran ou augmenter la vitesse.

Identification des objets



Autres critères de reconnaissance des objets

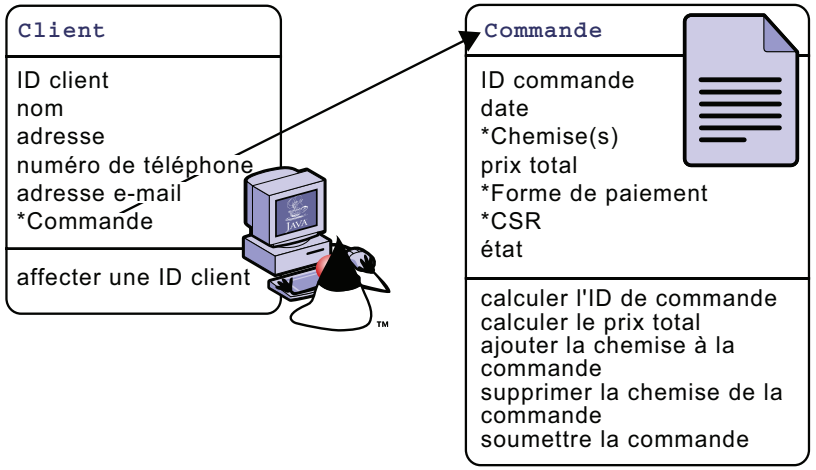
- Pertinence par rapport au domaine problématique :
  - L'objet existe-t-il dans les limites du domaine problématique ?
  - L'objet est-il indispensable à la solution ?
  - L'objet est-il nécessaire dans le cadre d'une interaction entre l'utilisateur et le système ?
- Existence indépendante

Interactions entre les objets dans l'étude de cas DirectClothing

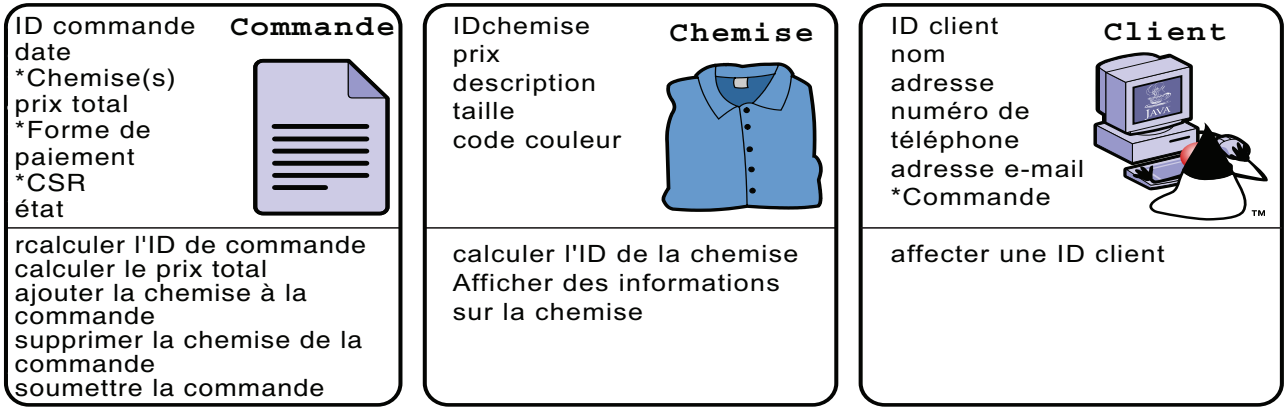
Identification des attributs et des opérations d'un objet

- Les attributs sont des données, telles que :
  - ID
  - Objet Commande
- Les opérations sont des actions, telles que :
  - Supprimer un élément
  - Modifier un ID

Objet avec un autre objet en tant qu'attribut



Attributs et opérations potentiels pour les objets de l'étude de cas DirectClothing



Solution de l'étude de cas

`Commande`	`Chemise`
réf. de commande date *Chemise(s) prix total *Formulaire de paiement *Conseiller du service clientèle état	réf. chemise prix description taille code couleur

Commande	Chemise
calculer réf. de commande calculer le prix total ajouter une chemise à la commande retirer une chemise de la commande envoyer la commande	calculer réf. de chemise afficher les informations sur la chemise

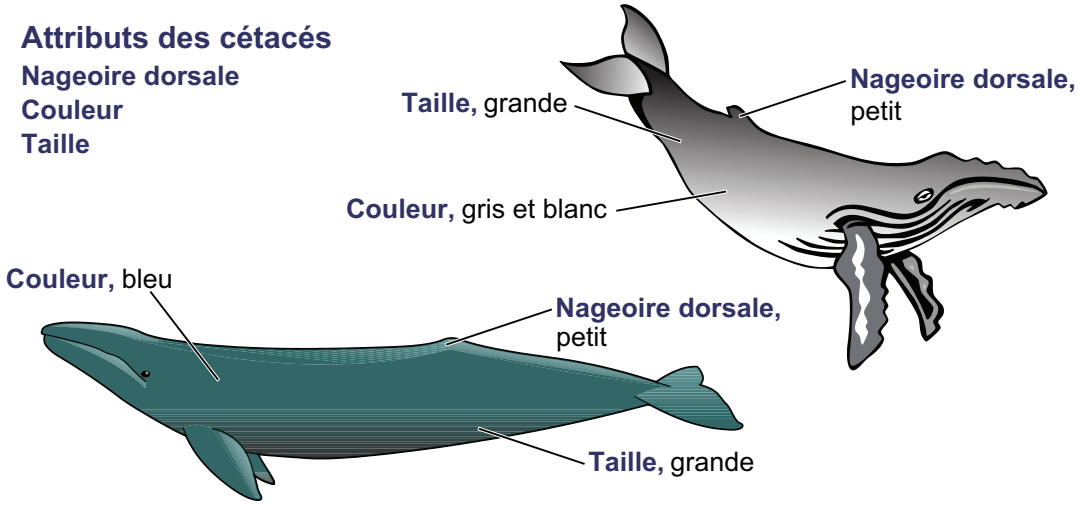
Solution de l'étude de cas

`Client`	`Formulaire de paiement`
réf. client nom adresse numéro de téléphone adresse électronique *Commande	vérifier le numéro numéro de carte bancaire date d'expiration
attribuer une référence client	vérifier le numéro de carte bancaire vérifier le paiement par chèque

Solution de l'étude de cas

Catalogue	Conseiller du service clientèle
*Chemise(s)	nom extension
ajouter une chemise retirer une chemise	

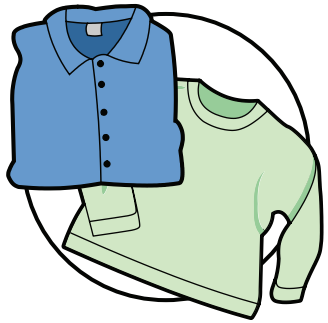
Conception de classes



Classe et objets résultants

Shirt
shirtID price description size colorCode R=Red, B=Blue, G=Green
calculateShirtID() displayShirtInformation()

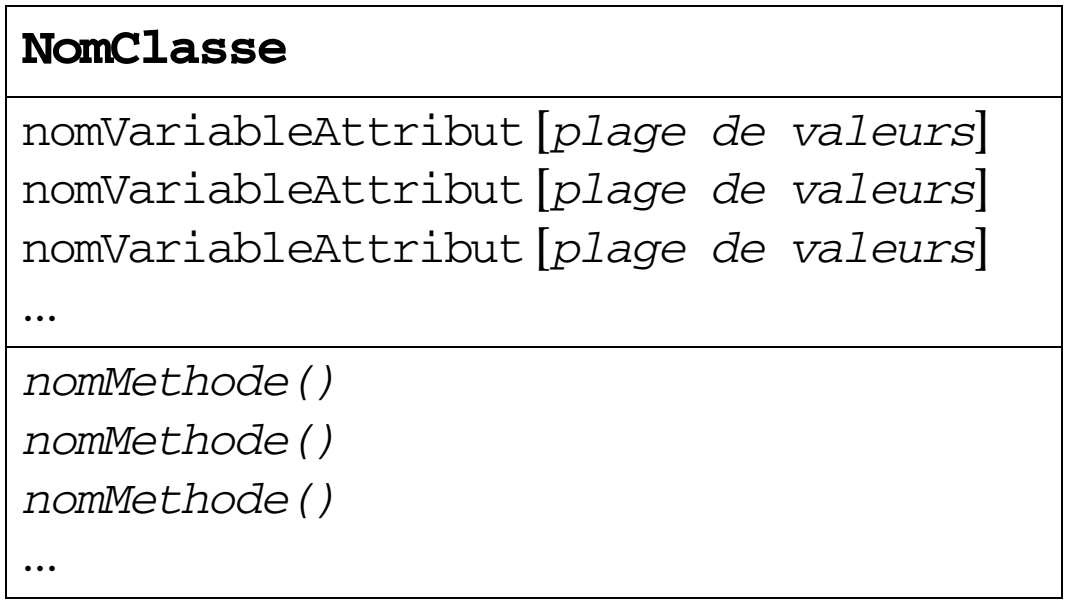
Classe `Shirt`



Objets `Shirt`

Modélisation des classes

- Syntaxe



- Exemple

Chemise
IDchemise prix description taille codeCouleur R=Rouge, B=Bleu, G=Vert
calculerIDChemise() afficherInformations()

Module 3

Développement et test d'un programme de technologie Java

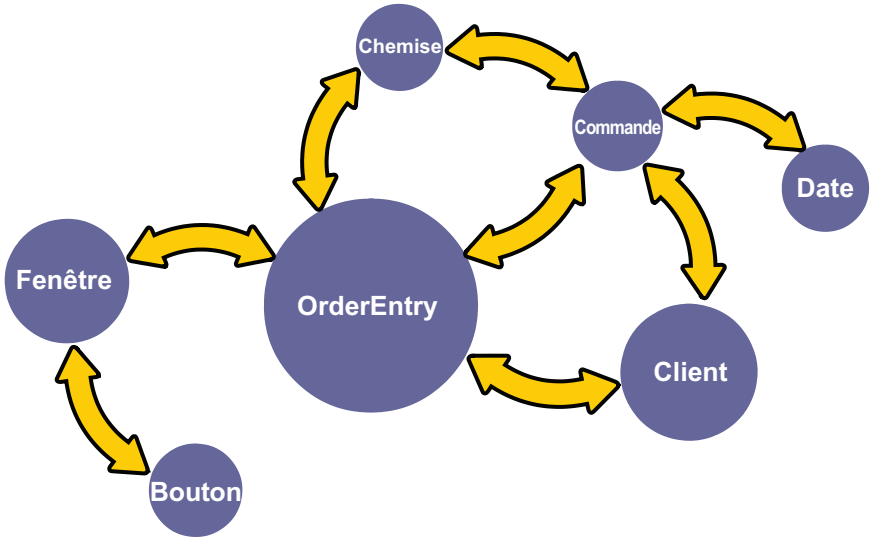
Objectifs

- Identifier les quatre composants d'une classe dans le langage de programmation Java
- Utiliser la méthode `main` dans une classe test pour exécuter un programme de technologie Java à partir de la ligne de commande
- Compiler et exécuter un programme de technologie Java

Pertinence

- Comment pouvez-vous tester un élément que vous avez construit, par exemple une maison, un meuble ou un programme ?

Identification des composants d'une classe



Structuration des classes

- La déclaration de la classe
- Les déclarations et l'initialisation des variables d'attribut (facultatif)
- Les méthodes (facultatif)
- Les commentaires (facultatif)

Structuration des classes

```
1   public class Shirt {
2
3   public int shirtID = 0; // ID par défaut de la chemise
4     public String description = "-description required-"; // par défaut
5   // Les codes de couleur sont R=Red, B=Blue, G=Green, U=Unset
6     public char colorCode = 'U';
7     public double price = 0.0; // Prix par défaut de toutes les chemises
8     public int quantityInStock = 0; // Quantité par défaut de toutes les
9                                     // chemises
10
11  // Cette méthode affiche les valeurs d'un article
12    public void displayInformation() {
13      System.out.println("ID chemise : " + shirtID);
14      System.out.println("Description de la chemise :" + description);
15      System.out.println("Code couleur : " + colorCode);
16      System.out.println("Prix de la chemise : " + price);
17      System.out.println("Quantité en stock : " + quantityInStock);
18
19  } // fin de la méthode d'affichage
20  } // fin de la classe
```

Déclaration d'une classe

- Syntaxe :

  [*modifiers*] `class` *`class_identifier`*
- Exemple :

  `public class Shirt`

Déclaration des variables et affectations

```
public int shirtID = 0;
public String description = "-description required-";
public char colorCode = 'U';
public double price = 0.0;
public int quantityInStock = 0;
```

Commentaires

- À une ligne :

```
public int shirtID = 0; // ID par défaut de la chemise
public double price = 0.0; // Prix par défaut de toutes les chemises

// Les codes de couleur sont R=Red, B=Blue, G=Green
```

- Traditionnel :

```
/**************************************************
* Section de déclaration des variables d'attribut   *
**************************************************/
```

Les méthodes

- Syntaxe :

```
[modificateurs] type_retour identificateur_méthode ([arguments]) {
    bloc_code_méthode
}
```

- Exemple :

```
public void displayInformation() {

    System.out.println("ID chemise : " + shirtID);
    System.out.println("Description de la chemise :" + description);
    System.out.println("Code couleur : " + colorCode);
    System.out.println("Prix de la chemise : " + price);
    System.out.println("Quantité en stock : " + quantityInStock);

  } // fin de la méthode d'affichage
```

Création et utilisation d'une classe test

Exemple :

```
1   class ShirtTest {
2
3     public static void main (String args[]) {
4
5       Shirt myShirt;
6       myShirt = new Shirt();
7
8       myShirt.displayInformation();
9
10
11    }
12  }
13
```

Méthode `main`

- Une méthode spéciale reconnue par la JVM comme le point de départ de tout programme Java exécuté à partir d'une ligne de command

- Syntaxe :

```
public static void main (String [] args)
```

Compilation d'un programme

1. Accédez au répertoire contenant les fichiers de code source.
2. Entrez la commande suivante pour chaque fichier `.java` à compiler.
   - Syntaxe :

javac *nom du fichier*

   - Exemple :

```
javac Shirt.java
```

Exécution (Test) d'un programme

1. Accédez au répertoire contenant les fichiers de classe.
2. Entrez ce qui suit pour le fichier de classe contenant la méthode `main` :
   - Syntaxe

java *nom de la classe*

   - Exemple

java ShirtTest

   - Output:

```
Shirt ID: 0
Shirt description:-description required-
Color Code: U
Shirt price: 0.0
Quantity in stock: 0
```

Conseils de débogage

- Les messages d'erreur précisent le numéro de la ligne dans laquelle chaque erreur se produit. Cette ligne n'est pas toujours la source réelle de l'erreur.
- Vérifiez que chaque ligne se termine par un point-virgule lorsque nécessaire, et rien d'autre.
- Assurez-vous d'avoir un nombre pair d'accolades.
- Assurez-vous d'avoir utilisé une indentation cohérente dans votre programme, telle qu'illustrée dans les exemples de ce cours.

Module 4

Déclaration, initialisation et utilisation des variables

Objectifs

- Identifier les utilisations des variables et définir leur syntaxe
- Énumérer les huit types de primitives du langage de programmation Java
- Déclarer, initialiser et utiliser des variables et des constantes selon les directives et les normes de codage du langage Java
- Modifier les valeurs des variables à l'aide d'opérateurs
- Utiliser la promotion et la conversion du type

Pertinence

- Une variable désigne un élément qui peut changer. Les variables peuvent contenir une valeur ou un ensemble de valeurs. Où avez-vous déjà vu des variables ?
- Quels types de données pensez-vous que les variables peuvent stocker ?

Identification de l'utilisation des variables et syntaxe

Exemple :

```
1   public class Shirt {
2
3     public int shirtID = 0; // ID par défaut de la chemise
4
5     public String description = "-description required-"; // par défaut
6
7     // Les codes de couleur sont R=Red, B=Blue, G=Green, U=Unset
8     public char colorCode = 'U';
9
10    public double price = 0.0; // Prix par défaut de toutes les chemises
11
12    public int quantityInStock = 0; // Quantité par défaut de toutes les
13                                    // chemises
14
15  // Cette méthode affiche les valeurs d'un article
16    public void displayInformation() {
17
18      System.out.println("ID chemise : " + shirtID);
```

Identification de l'utilisation des variables et syntaxe

Exemple (suite)

```
19      System.out.println("Description de la chemise :" + description);
20      System.out.println("Code couleur : " + colorCode);
21      System.out.println("Prix de la chemise : " + price);
22      System.out.println("Quantité en stock : " + quantityInStock);
23
24    } // fin de la méthode d'affichage
25
26  } // fin de la classe
```

Utilisation des variables

- La conservation de données uniques pour l'instance d'un objet
- L'attribution de la valeur d'une variable à une autre.
- La représentation des valeurs dans une expression mathématique
- L'impression de valeurs à l'écran
- La conservation des références à d'autres objets

Déclaration et initialisation des variables

- Syntaxe (variables d'attribut ou d'instance) :

[*modificateurs*] *type identificateur [= valeur*];

- Syntaxe (variables locales) :

type identificateur;

- Syntaxe (variables locales)

type identificateur [= valeur];

- Exemples :

```
public int shirtID = 0;
public String description = "-description required-";
public char colorCode = 'U';
public double price = 0.0;
public int quantityInStock = 0;
```

Description des types de données primitives

- Types de nombre entier (`byte`, `short`, `int` et `long`)
- Types à virgule flottante (`float` et `double`)
- Type de texte (`char`)
- Type logique (`boolean`)

Types de primitives de nombre entier

Type	Longueur	Plage	Exemples de valeurs littérales autorisées
`byte`	8 bits	-2^7 à 2^7 -1 (-128 à 127 ou 256 valeurs possibles)	2 -114
`short`	16 bits	-2^{15} à 2^{15} -1 (-32 768 à 32 767 ou 65 535 valeurs possibles)	2 -32699
`int`	32 bits	-2^{31} à 2^{31} -1 (-2 147 483 648 à 2 147 483 647 ou 4 294 967 296 valeurs possibles)	2 147334778

Types de primitives de nombre entier

Type	Longueur	Plage	Exemples de valeurs littérales autorisées
`long`	64 bits	-2^{63} à $2^{63}-1$ (-9 223 372 036 854 775 808 à 9 223 372 036 854 775 807 ou 18 446 744 073 709 551 616 valeurs possibles)	2 -2036854775808L 1L

Types de primitives de nombre entier

```
public int shirtID = 0; // ID par défaut de la chemise
public int quantityInStock = 0; // Quantité par défaut de toutes les chemises
```

Types de primitives à virgule flottante

Type	Longueur	Exemples de valeurs littérales autorisées
`float`	32 bits	99F -327456,99.01F 4.2E6F (notation scientifique de $4,2 * 10^6$)
`double`	64 bits	-1111 2.1E12 99970132745699.999

```
public double price = 0.0; // Prix par défaut de toutes les chemises
```

Type de primitive textuelle

- Le seul type de données est `char`
- Utilisé pour un seul caractère (16 bits)
- Exemple :

```
public char colorCode = 'U';
```

Type de primitive logique

- Le seul type de données est `boolean`
- Peut stocker uniquement `true` ou `false`
- Conserve le résultat d'une expression qui évalue uniquement `true` ou `false`.

Appellation d'une variable

Règles :

- Les identificateurs de variable doivent commencer par une lettre majuscule ou minuscule, un caractère de soulignement (_) ou le signe dollar ($).
- Les identificateurs de variable ne peuvent pas contenir de signes de ponctuation, d'espaces ni de tirets.
- Les mots-clés Java ne peuvent pas être utilisés.

Appellation d'une variable

Directives :

- Commencez chaque variable par une lettre minuscule ; les mots suivants doivent commencer par une majuscule, par exemple `maVariable`.
- Choisissez des noms faciles à retenir et qui indiquent à l'observateur occasionnel l'objectif de la variable.

Attribution d'une valeur à une variable

- Exemple :

```
double price = 12.99;
```

- Exemple (`boolean`) :

```
boolean isOpen = false;
```

Déclaration et initialisation de plusieurs variables sur une même ligne de code

- Syntaxe :

type identificateur = valeur [, *identificateur = valeur*];

- Exemple :

```
double price = 0.0, wholesalePrice = 0.0;
```

Autres manières de déclarer des variables et de leur attribuer des valeurs

- Attribution de valeurs littérales :

```
int ID = 0;
float pi = 3.14F;
char myChar = 'G';
boolean isOpen = false;
```

- Attribution de la valeur d'une variable à une autre variable :

```
int ID = 0;
int saleID = ID;
```

Autres manières de déclarer des variables et de leur attribuer des valeurs

- Attribution du résultat d'une expression à des variables de type nombre entier, à virgule flottante ou booléen

```
float numberOrdered = 908.5F;
float casePrice = 19.99F;
float price = (casePrice * numberOrdered);

int hour = 12;
boolean isOpen = (hour > 8);
```

- Attribution à une variable de la valeur renvoyée par un appel de méthode

Constantes

- Variable (peut changer) :

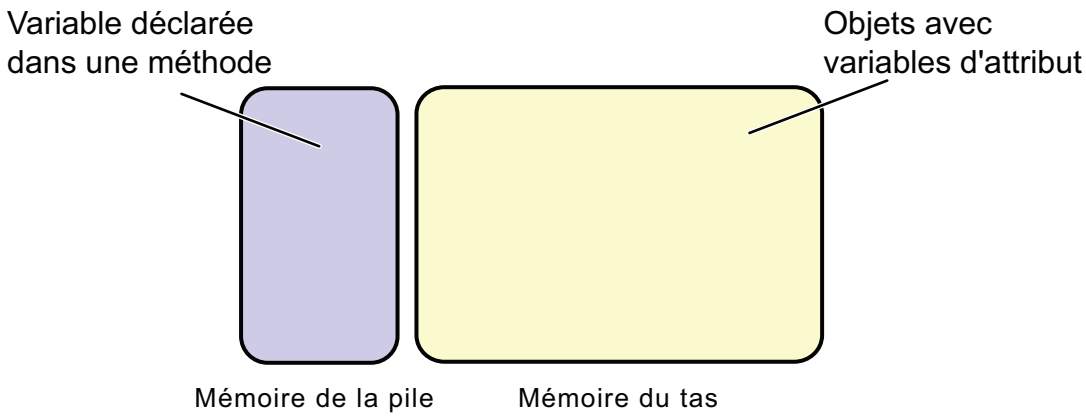
```
double tauxTVA = 6.25;
```

- Constante (ne peut pas changer) :

```
final double TAUX_TVA = 6.25;
```

- Directive – Les constantes doivent être en majuscules et leurs mots séparés par un caractère de soulignement (_).

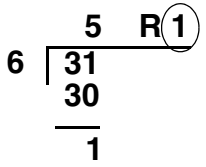
Stockage des primitives et des constantes en mémoire



Opérateurs mathématiques standard

Objectif	Opérateur	Exemple	Commentaires
Addition	+	`sum = num1 + num2;` Si `num1` est 10 et `num2` est 2, `diff` est 12.	
Soustraction	–	`diff = num1 – num2;` Si `num1` est 10 et `num2` est 2, `diff` est 8.	
Multiplication	*	`prod = num1 * num2;` Si `num1` est 10 et `num2` est 2, `prod` est 20.	
Division	/	`quot = num1 / num2;` Si `num1` est 31 et `num2` est 6, `quot` est 5	La division renvoie une valeur entière (sans reste).

Opérateurs mathématiques standard

Objectif	Opérateur	Exemple	Commentaires
Reste	`%`	`mod = num1 % num2;` Si `num1` est 31 et `num2` est 6, `mod` est 1	Le reste correspond à l'élément restant lorsque le premier nombre est divisé par le second nombre.  Le reste donne toujours une réponse du même signe que le premier opérande.

Opérateurs d'incrémentation et de décrémentation (++ et --)

La syntaxe longue :

```
age = age + 1;
```

Opérateurs d'incrémentation et de décrémentation (++ et --)

La syntaxe courte :

Opérateur	Objectif	Exemple	Notes

`++`	Pré-incrémentation (`++`*`variable`*)	`int i = 6;`
		`int j = ++i;`
		`i est 7, j est 7`	

	Post-incrémentation (*`variable`*`++`)	`int i = 6;`
		`int j = i++;`
		`i est 7, j est 6`
			La valeur de `i` est attribuée à `j` avant l'incrémentation de `i`. De ce fait, `j` reçoit la valeur `6`.

Opérateurs d'incrémentation et de décrémentation (++ et --)

Opérateur	Objectif	Exemple	Notes

`--`	Pré-décrémentation (`--`*`variable`*)	`int i = 6;`
		`int j = --i;`
		`i est 5, j est 5`	

	Post-décrémentation (*`variable`*`--`)	`int i = 6;`
		`int j = i--;`
		`i est 5, j est 6`
			La valeur de `i` est attribuée à `j` avant la décrémentation de `i`. De ce fait, `j` reçoit la valeur 6.

Opérateurs d'incrémentation et de décrémentation (++ et --)

Exemples :

```
int count=15;
int a, b, c, d;
a = count++;
b = count;
c = ++count;
d = count;
System.out.println(a + ", " + b + ", " + c + ", " + d);
```

Priorité des opérateurs

Règles des priorités :

1. Opérateurs placés entre parenthèses
2. Opérateurs d'incrémentation et de décrémentation
3. Opérateurs de multiplication et de division, évalués de gauche à droite
4. Opérateurs d'addition et de soustraction, évalués de gauche à droite

Exemple d'application des règles de priorité (la réponseest-elle 34 ou 9 ?) :

```
c = 25 - 5 * 4 / 2 - 10 + 4;
```

Utilisation des parenthèses

Exemples :

```
c = (((25 - 5) * 4) / (2 - 10)) + 4;
c = ((20 * 4) / (2 - 10)) + 4;
c = (80 / (2 - 10)) + 4;
c = (80 / -8) + 4;
c = -10 + 4;
c = -6;
```

Utilisation de la promotion et de la conversion de types

- Exemple de problème potentiel :

```
int num1 = 53; // 32 bits de mémoire pour détenir la valeur
int num2 = 47; // 32 bits de mémoire pour détenir la valeur
byte num3; // 8 bits de mémoire réservée
num3 = (num1 + num2); // entraîne une erreur de compilation
```

- Exemple de solution potentielle :

```
int num1 = 53;
int num2 = 47;
long num3;
num3 = (num1 + num2);
```

Promotion

- Promotions automatiques :
  - Si vous attribuez un type plus petit à un type plus grand
  - Si vous attribuez un type nombre entier à un type à virgule flottante
- Exemples de promotions automatiques :

```
long big = 6;
```

Conversion de types

- Syntaxe :

identificateur = (*type_cible*) *valeur*

- Exemple de problème potentiel :

```
int num1 = 53; // 32 bits de mémoire pour détenir la valeur
int num2 = 47; // 32 bits de mémoire pour détenir la valeur
byte num3; // 8 bits de mémoire réservée
num3 = (num1 + num2); // entraîne une erreur de compilation
```

- Exemple de solution potentielle :

```
int num1 = 53; // 32 bits de mémoire pour détenir la valeur
int num2 = 47; // 32 bits de mémoire pour détenir la valeur
byte num3; // 8 bits de mémoire réservée
num3 = (byte)(num1 + num2); // pas de perte de données
```

Conversion de types

Exemples :

```
int myInt;
long myLong = 99L;
myInt = (int) (myLong); // Pas de perte de données, seulement des zéros.
                        // Un nombre beaucoup plus grand entraînerait une
                        // perte de données.

int myInt;
long myLong = 123987654321L;
myInt = (int) (myLong); // Le nombre est « tronqué »
```

Hypothèses du compilateur pour les types nombre entier et à virgule flottante

- Exemple de problème potentiel :

```
short a, b, c;
a = 1 ;
b = 2 ;
c = a + b ; // erreur de compilation
```

- Exemple de solutions potentielles :
  - Si vous déclarez `c` en type `int` dans la déclaration d'origine :

```
int c;
```

  - Si vous convertissez le type du résultat (`a+b`) dans la ligne de l'attribution :

```
c = (short)(a+b);
```

Types de données à virgule flottante et attribution

- Exemple de problème potentiel :

```
float float1 = 27.9;// erreur de compilation
```

- Exemple de solutions potentielles :
  - Le `F` indique au compilateur que 27.9 est une valeur `float` :

```
float float1 = 27.9F;
```

  - La valeur `27.9` est convertie en type `float` :

```
float float1 = (float) 27.9;
```

Exemple

```
1   public class Person {
2
3     public int ageYears = 32;
4
5     public void calculateAge() {
6
7       int ageDays = ageYears * 365;
8       long ageSeconds = ageYears * 365 * 24L * 60 * 60;
9
10      System.out.println("Vous êtes âgé de " + ageDays + " jours.");
11      System.out.println("Vous êtes âgé de " + ageSeconds + " secondes.");
12
13    } // fin de la méthode calculateAge
14  } // fin de la classe
```

Module 5

Création et utilisation d'objets

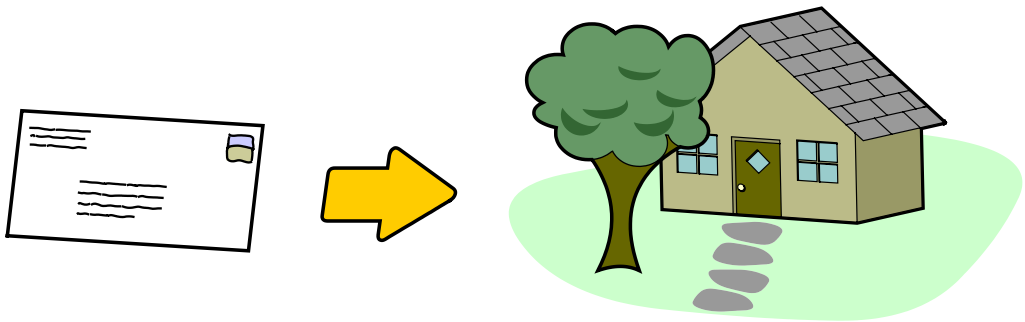
Objectifs

- Déclarer, instancier et initialiser des variables de référence d'objet
- Comparer le stockage des variables de référence d'objet et des variables primitives
- Utiliser la classe `String`, incluse dans le kit de développement logiciel Java (SDK)
- Utiliser les spécifications de la bibliothèque de classes Java SE™ (Java Platform Standard Edition) pour découvrir les autres classes de cette API (application programming interface)

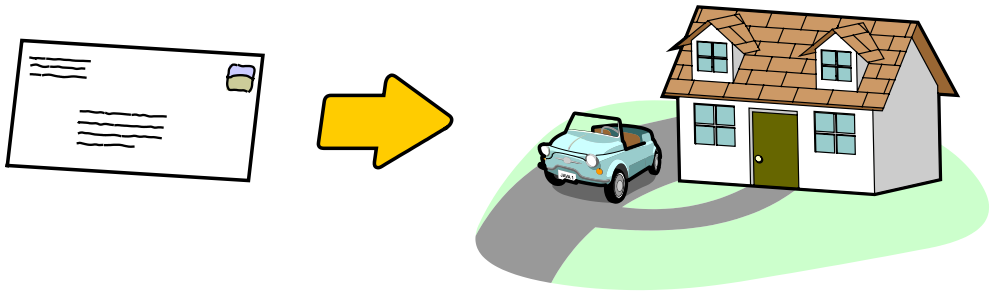
Pertinence

- Que signifie créer une instance du plan d'une maison ?
- Comment pouvez-vous faire référence aux différentes maisons d'une même rue ?
- Lorsqu'un entrepreneur construit une maison, en fabrique-t-il chaque composant, y compris les fenêtres, les portes et les placards ?

Déclaration de références d'objet, instanciation d'objets et initialisation des références d'objet



3222 Jones St.



777 Boulder Ln.

Déclaration de références d'objet, instanciation d'objets et initialisation des références d'objet

Exemple :

```
1   class ShirtTest {
2
3   public static void main (String args[]) {
4
5       Shirt myShirt = new Shirt();
6
7       myShirt.displayInformation();
8
9     }
10  }
```

Déclaration des variables de référence d'objet

- Syntaxe :

  NomClasse identificateur;

- Exemple :

```
Shirt myShirt;
```

Instanciation d'un objet

Syntaxe :

```
new NomClasse()
```

Initialisation des variables de référence d'objet

- L'opérateur d'attribution
- Exemple :

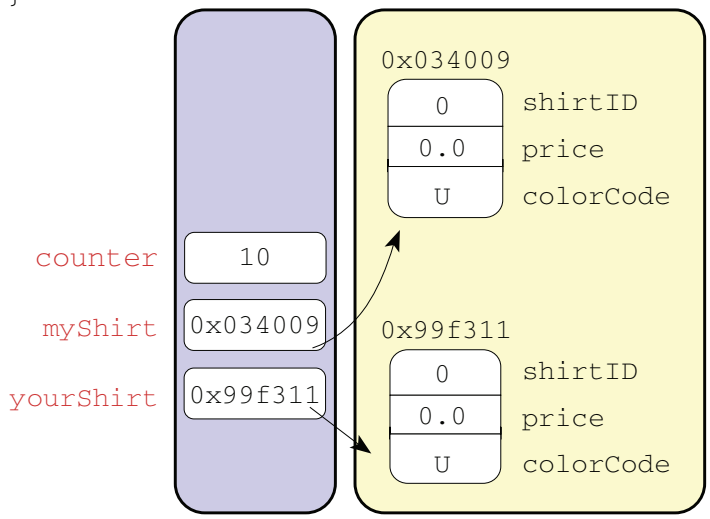
```
myShirt = new Shirt();
```

Utilisation d'une variable de référence d'objet pour manipuler des données

```
1   public class ShirtTestTwo {
2
3     public static void main (String args[]) {
4
5       Shirt myShirt = new Shirt();
6       Shirt yourShirt = new Shirt();
7
8       myShirt.displayInformation();
9       yourShirt.displayInformation();
10
11      myShirt.colorCode='R';
12      yourShirt.colorCode='G';
13
14      myShirt.displayInformation();
15      yourShirt.displayInformation();
16
17    }
18  }
```

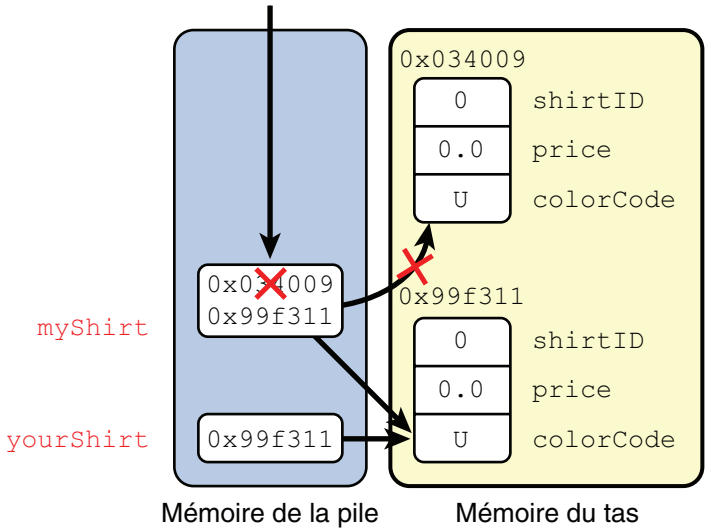
Stockage des variables de référence d'objet en mémoire

```
public static void main (String args[]) {

  int counter;
  counter = 10;
  Shirt myShirt = new Shirt ( );
}
```



Affectation d'une référence d'objet d'une variable à une autre

```
1  Shirt  myShirt - new Shirt( );
2  Shirt  yourShirt = new Shirt( );
3  myShirt = yourShirt;
```



Utilisation de la classe `String`

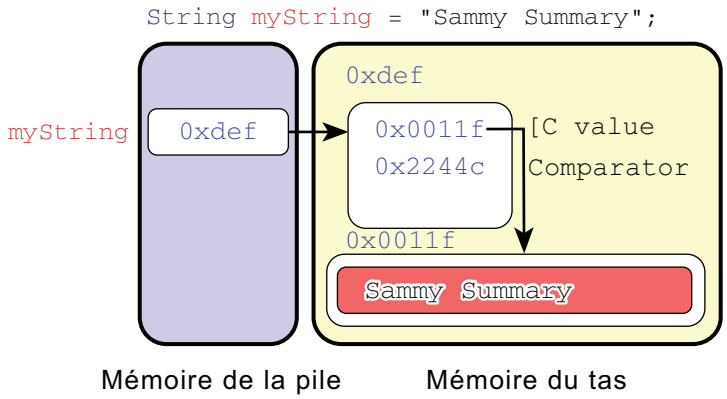
- Création d'un objet `String` avec le mot-clé `new` :

```
String myName = new String("Fred Smith");
```

- Création d'un objet `String` sans le mot-clé `new` :

```
String myName = "Fred Smith";
```

Stockage des objets `String` en mémoire



Utilisation de variables de référence pour les objets `String`

Exemple :

```
1   public class PersonTwo {
2
3     public String name = "Jonathan";
4     public String job = "Goûteur de glaces à la crème";
5
6     public void display(){
7       System.out.println("Mon nom est " + name + ", je suis " + job);
8     }
9   } // fin de la classe
```

Examen des bibliothèques de classes Java

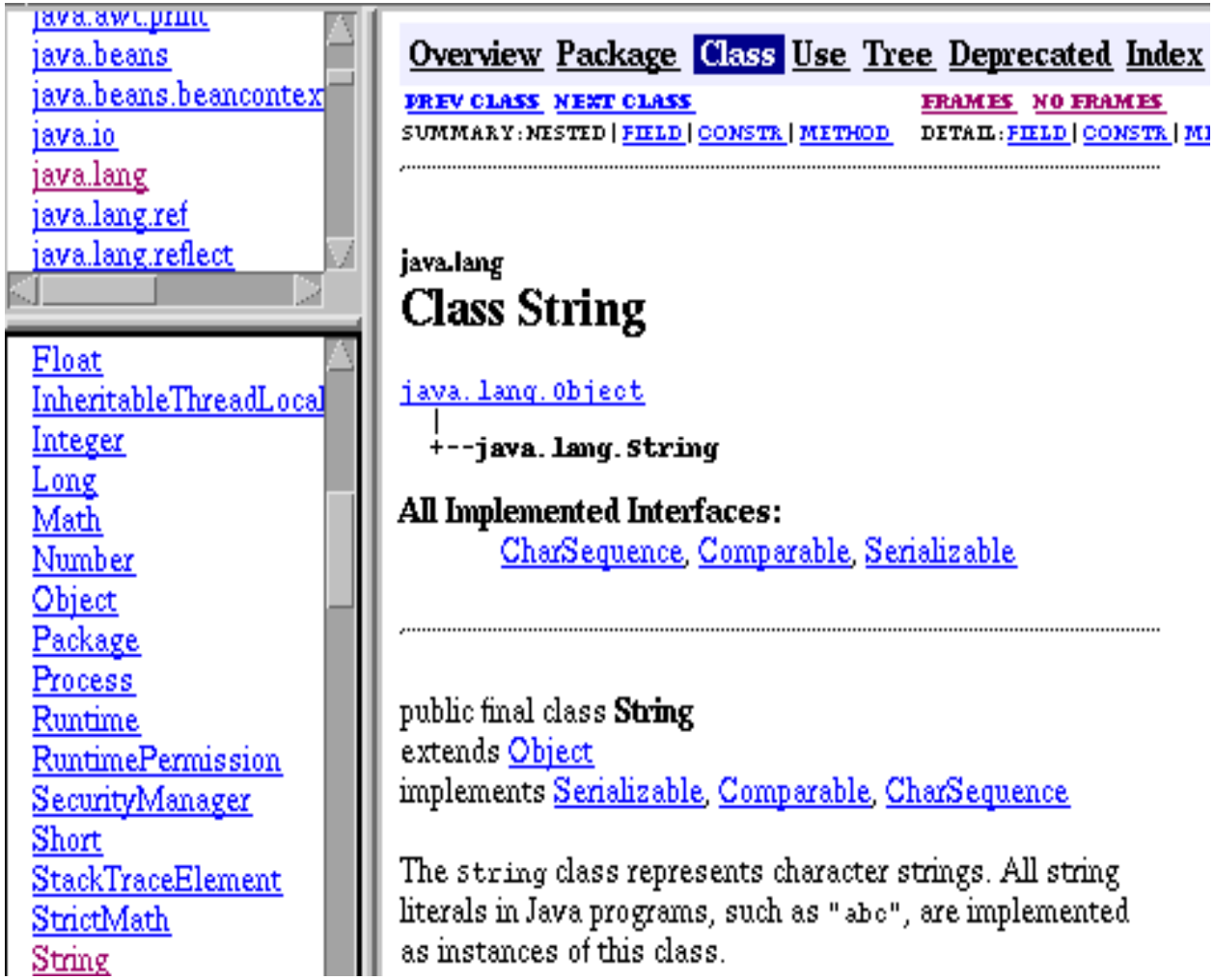
- URL (Universal Resource Locator) où consulter la spécification Java SE :

```
http://java.sun.com/reference/api/
```

- Exemple :

```
http://java.sun.com/javase/6/docs/api/
```

Examen des bibliothèques de classes Java



Utilisation des spécifications des bibliothèques de classes Java pour s'initier à une méthode

- Laméthode `println`

```
System.out.println(données_à_imprimer_à_l_écran);
```

- Exemple :

```
System.out.print("Carpe diem");
System.out.println("Seize the day");
```

imprime :

```
Carpe diem Seize the day
```

Module 6

Utilisation des opérateurs et des constructions conditionnelles

Objectifs

- Identifier les opérateurs relationnels et conditionnels
- Créer des constructions `if` et `if/else`
- Utiliser les constructions `switch`

Pertinence

- Lorsque vous devez prendre une décision parmi plusieurs options, comment faites-vous votre choix final ?
- Par exemple, que vous vient-il à l'esprit lorsque vous devez acheter un article ?

Utilisation des opérateurs relationnels et conditionnels

Exemple de la classe Elevator

```
1   public class Elevator {
2
3     public boolean doorOpen=false; // Les portes sont fermées par défaut.
4     public int currentFloor = 1; // Tous les ascenseurs partent du premier
5                                  // étage.
6     public final int TOP_FLOOR = 10;
7     public final int MIN_FLOORS = 1;
8
9     public void openDoor() {
10      System.out.println("Ouverture de la porte.");
11      doorOpen = true;
12      System.out.println("La porte est ouverte.");
13    }
14
15    public void closeDoor() {
16      System.out.println("Fermeture de la porte.");
17      doorOpen = false;
18      System.out.println("La porte est fermée.");
19    }
```

Exemple de la classe Elevator

```
20
21    public void goUp() {
22      System.out.println(“Monter d'un étage.”);
23      currentFloor++;
24      System.out.println(“Étage : “ + currentFloor);
25    }
26
27    public void goDown() {
28      System.out.println(“Descendre d'un étage.”);
29      currentFloor--;
30      System.out.println(“Étage : “ + currentFloor);
31    }
32
33
34  }
```

Le fichier `ElevatorTest.java`

```
1   public class ElevatorTest {
2     public static void main (String args[]) {
3
4       Elevator myElevator = new Elevator();
5
6        myElevator.openDoor();
7        myElevator.closeDoor();
8        myElevator.goDown();
9        myElevator.goUp();
10       myElevator.goUp();
11       myElevator.goUp();
12       myElevator.openDoor();
13       myElevator.closeDoor();
14       myElevator.goDown();
15       myElevator.openDoor();
16       myElevator.goDown();
17       myElevator.openDoor();
18    }
19  }
```

Opérateurs relationnels

Condition	Opérateur	Exemple
est égal à	`==`	`int i=1;` `(i == 1)`
n'est pas égal à	`!=`	`int i=2;` `(i != 1)`
Est inférieur à	`<`	`int i=0;` `(i < 1)`
Est inférieur ou égal à	`<=`	`int i=1;` `(i <= 1)`
Est supérieur à	`>`	`int i=2;` `(i > 1)`
Est supérieur ou égal à	`>=`	`int i=1;` `(i >= 1)`

Test d'égalité entre des chaînes

Exemple :

```
1   public class Employees {
2
3     public String name1 = "Fred Smith";
4     public String name2 = "Joseph Smith";
5
6     public void areNamesEqual() {
7
8       if (name1.equals(name2)) {
9         System.out.println("Même nom.");
10      }
11      else {
12        System.out.println("Nom différent.");
13      }
14    }
15  }
16
```

Opérateurs conditionnels courants

Opération	Opérateur	Exemple

Si une condition ET une autre condition	`&&`	`int i = 2;`
		`int j = 8;`
		`((i < 1) && (j > 6))`

Si une condition OU une autre condition	`||`	`int i = 2;`
		`int j = 8;`
		`((i < 1) || (j > 10))`

PAS	`!`	`int i = 2;` `(!(i < 3))`

Construction `if`

- Syntaxe :

```
if (expression_booléenne) {
   bloc_de_code;
}   // fin de la construction if
// le programme continue ici
```

- Exemple de résultat potentiel :

```
Ouverture de la porte.
La porte est ouverte.
Fermeture de la porte.
La porte est fermée.
Descendre d'un étage.
Étage : 0 <--- Erreur de logique
Monter d'un étage.
Étage : 1
Monter d'un étage.
Étage : 2
...
```

Construction `if`

Exemple de solution potentielle :

```
1
2   public class IfElevator {
3
4     public boolean doorOpen=false; // Les portes sont fermées par défaut.
5     public int currentFloor = 1; // Tous les ascenseurs partent du premier
6                                  // étage.
7     public final int TOP_FLOOR = 10;
8     public final int MIN_FLOORS = 1;
9
10    public void openDoor() {
11      System.out.println("Ouverture de la porte.");
12      doorOpen = true;
13      System.out.println("La porte est ouverte.");
14    }
15    public void closeDoor() {
16      System.out.println("Fermeture de la porte.");
17      doorOpen = false;
```

Construction `if`

```
18  System.out.println("La porte est fermée.");
19    }
20    public void goUp() {
21      System.out.println("Monter d'un étage.");
22      currentFloor++;
23      System.out.println("Étage : " + currentFloor);
24    }
25    public void goDown() {
26
27      if (currentFloor == MIN_FLOORS) {
28        System.out.println("Impossible de descendre");
29      }
30      if (currentFloor > MIN_FLOORS) {
31        System.out.println("Descendre d'un étage.");
32        currentFloor--;
33        System.out.println("Étage : " + currentFloor);
34      }
35    }
36  }
```

Construction `if`

Exemple de résultat potentiel :

```
Ouverture de la porte.
La porte est ouverte.
Fermeture de la porte.
La porte est fermée.
Impossible de descendre <--- la logique de l'ascenseur évite le problème
Monter d'un étage.
Étage : 2
Monter d'un étage.
Étage : 3
...
```

Instructions `if` imbriquées

Exemple :

```
1
2   public class NestedIfElevator {
3
4     public boolean doorOpen=false; // Les portes sont fermées par défaut.
5     public int currentFloor = 1; // Tous les ascenseurs partent du premier
6                                  // étage.
7     public final int TOP_FLOOR = 10;
8     public final int MIN_FLOORS = 1;
9
10    public void openDoor() {
11      System.out.println("Ouverture de la porte.");
12      doorOpen = true;
13      System.out.println("La porte est ouverte.");
14    }
15
16    public void closeDoor() {
17      System.out.println("Fermeture de la porte.");
18      doorOpen = false;
```

Instructions `if` imbriquées

```
19  System.out.println("La porte est fermée.");
20    }
21
22    public void goUp() {
23      System.out.println("Monter d'un étage.");
24      currentFloor++;
25      System.out.println("Étage : " + currentFloor);
26    }
27
28    public void goDown() {
29
30      if (currentFloor == MIN_FLOORS) {
31        System.out.println("Impossible de descendre");
32      }
33
34      if (currentFloor > MIN_FLOORS) {
35
36        if (!doorOpen) {
37
```

Instructions `if` imbriquées

```
38          System.out.println("Descendre d'un étage.");
39          currentFloor--;
40          System.out.println("Étage : " + currentFloor);
41        }
42      }
43    }
44
45
46  }
47
```

Construction `if/else`

Syntaxe :

```
if (expression_booléenne) {

     bloc_de_code;

} // fin de la construction if

else {

     bloc_de_code;

} // fin de la construction else

// le programme continue ici
```

Construction `if/else`

Exemple :

```
1   public class IfElseElevator {
2
3     public boolean doorOpen=false; // Les portes sont fermées par défaut.
4     public int currentFloor = 1; // Tous les ascenseurs partent du premier
5                                  // étage.
6     public final int TOP_FLOOR = 10;
7     public final int MIN_FLOORS = 1;
8
9     public void openDoor() {
10      System.out.println("Ouverture de la porte.");
11      doorOpen = true;
12      System.out.println("La porte est ouverte.");
13    }
14    public void closeDoor() {
15      System.out.println("Fermeture de la porte.");
16      doorOpen = false;
17      System.out.println("La porte est fermée.");
18    }
```

Construction `if/else`

```
19
20    public void goUp() {
21      System.out.println("Monter d'un étage.");
22      currentFloor++;
23      System.out.println("Étage : " + currentFloor);
24    }
25
26    public void goDown() {
27
28      if (currentFloor == MIN_FLOORS) {
29        System.out.println("Impossible de descendre");
30      }
31      else {
32        System.out.println("Descendre d'un étage.");
33        currentFloor--;
34        System.out.println("Étage : " + currentFloor);}
35      }
36    }
37  }
```

Construction `if/else`

Exemple de résultat potentiel :

```
Ouverture de la porte.
La porte est ouverte.
Fermeture de la porte.
La porte est fermée.
Impossible de descendre <--- la logique de l'ascenseur évite le problème
Monter d'un étage.
Étage : 2
Monter d'un étage.
Étage : 3
...
```

Constructions `if/else` chaînées

Syntaxe :

```
if (expression_booléenne) {

     bloc_de_code;

} // fin de la construction if

else if (expression_booléenne){

     bloc_de_code;

} // fin de la construction else if

else {

     bloc_de_code;
}
// le programme continue ici
```

Constructions `if/else` chaînées

Exemple :

```
1
2   public class IfElseDate {
3
4     public int month = 10;
5
6     public void calculateNumDays() {
7
8       if (month == 1 || month == 3 || month == 5 || month == 7 ||
9     month == 8 || month == 10 || month == 12) {
10
11        System.out.println("Il y a 31 jours dans ce mois.");
12      }
13
14      else if (month == 2) {
15        System.out.println("Il y a 28 jours dans ce mois.");
16      }
17
```

Constructions `if/else` chaînées

```
18  else if (month == 4 || month == 6 || month == 9 || month == 11) {
19        System.out.println("Il y a 30 jours dans ce mois.");
20      }
21
22      else {
23        System.out.println("Mois non valide.");
24      }
25    }
26  }
27
```

Utilisation de la construction `switch`

Syntaxe :

```
switch (variable) {
    case valeur_littérale:
        bloc_de_code;
        [break;]
    case autre_valeur_littérale:
         bloc_de_code;
        [break;]
    [default:]
         bloc_de_code;
}
```

Utilisation de la construction `switch`

Exemple :

```
1
2   public class SwitchDate {
3
4     public int month = 10;
5
6     public void calculateNumDays() {
7
8       switch(month) {
9       case 1:
10      case 3:
11      case 5:
12      case 7:
13      case 8:
14      case 10:
15      case 12:
16        System.out.println("Il y a 31 jours dans ce mois.");
17        break;
```

Utilisation de la construction `switch`

```
18  case 2:
19        System.out.println("Il y a 28 jours dans ce mois.");
20        break;
21      case 4:
22      case 6:
23      case 9:
24      case 11:
25        System.out.println("Il y a 30 jours dans ce mois.");
26        break;
27      default:
28        System.out.println("Mois non valide.");
29        break;
30      }
31    }
32  }
33
```

À quel moment utiliser la construction `switch` ?

- Les tests d'égalité
- Des tests par rapport à une valeur *unique*, par exemple `customerStatus`
- Des tests par rapport à une valeur de type `int`, `short`, `byte` ou `char`

Module 7

Utilisation de constructions en boucle

Objectifs

- Créer des boucles `while`
- Développer des boucles `for`
- Créer des boucles `do/while`

Pertinence

Dans quelles situations souhaiteriez-vous continuer à effectuer une action donnée tant qu'une certaine condition est remplie ?

Création de boucles `while`

Syntaxe :

```
while (expression_booléenne) {

   bloc_de_code;

}   // fin de la construction while

// le programme continue ici
```

Création de boucles `while`

Exemple :

```
1
2   public class WhileElevator {
3
4     public boolean doorOpen=false;
5     public int currentFloor = 1;
6
7     public final int TOP_FLOOR = 5;
8     public final int BOTTOM_FLOOR = 1;
9
10    public void openDoor() {
11      System.out.println("Ouverture de la porte.");
12      doorOpen = true;
13      System.out.println("La porte est ouverte.");
14    }
15
16    public void closeDoor() {
17      System.out.println("Fermeture de la porte.");
```

Création de boucles `while`

```
18  doorOpen = false;
19      System.out.println("La porte est fermée.");
20    }
21
22    public void goUp() {
23      System.out.println("Monter d'un étage.");
24      currentFloor++;
25      System.out.println("Étage : " + currentFloor);
26    }
27
28    public void goDown() {
29      System.out.println("Descendre d'un étage.");
30      currentFloor--;
31      System.out.println("Étage : " + currentFloor);
32    }
33
34    public void setFloor() {
35
```

Création de boucles `while`

```
36  // Normalement, vous devriez transmettre l'étage désiré (desiredFloor)
37      // sous forme d'argument à la méthode setFloor. Toutefois, comme
38      // vous n'avez pas encore appris à effectuer cette opération,
39      // desiredFloor est défini sur un nombre spécifique (5) ci-dessous.
40
41      int desiredFloor = 5;
42
43      while (currentFloor != desiredFloor){
44      if (currentFloor < desiredFloor) {
45        goUp();
46        }
47      else {
48        goDown();
49        }
50      }
51
52  }
53
```

Boucles `while` imbriquées

Exemple de solution potentielle :

```
1   public class WhileRectangle {
2     public int height = 3;
3     public int width = 10;
4     public void displayRectangle() {
5       int colCount = 0;
6       int rowCount = 0;
7       while (rowCount < height) {
8         colCount=0;
9         while (colCount < width) {
10          System.out.print("@");
11          colCount++;
12  }
13        System.out.println();
14        rowCount++;
15      }
16    }
17  }
```

Développement d'une boucle `for`

Syntaxe :

```
for (initialize[,initialize]; expression_booléenne;
     update[,update]) {

     bloc_de_code;
}
```

Développement d'une boucle `for`

Exemple :

```
1
2   public class ForElevator {
3
4     public boolean doorOpen=false;
5     public int currentFloor = 1;
6
7     public final int TOP_FLOOR = 5;
8     public final int BOTTOM_FLOOR = 1;
9
10    public void openDoor() {
11      System.out.println("Ouverture de la porte.");
12      doorOpen = true;
13      System.out.println("La porte est ouverte.");
14    }
15
16    public void closeDoor() {
17      System.out.println("Fermeture de la porte.");
```

Développement d'une boucle `for`

```
18  doorOpen = false;
19      System.out.println("La porte est fermée.");
20    }
21
22    public void goUp() {
23      System.out.println("Monter d'un étage.");
24      currentFloor++;
25      System.out.println("Étage : " + currentFloor);
26    }
27
28    public void goDown() {
29      System.out.println("Descendre d'un étage.");
30      currentFloor--;
31      System.out.println("Étage : " + currentFloor);
32    }
33
34    public void setFloor() {
35
```

Développement d'une boucle `for`

```
36  // Normalement, vous devriez transmettre l'étage désiré (desiredFloor)
37      // sous forme d'argument à la méthode setFloor. Toutefois, comme
38      // vous n'avez pas encore appris à effectuer cette opération,
39      // desiredFloor est défini sur un nombre spécifique (5) ci-dessous.
40      int desiredFloor = 5;
41
42      if (currentFloor > desiredFloor) {
43        for (int down = currentFloor; down != desiredFloor; --down) {
44          goDown();
45        }
46      }
47      else {
48        for (int up = currentFloor; up != desiredFloor; ++up) {
49          goUp();
50        }
51      }
52    }
53  }
54
```

Boucles `for` imbriquées

Exemple :

```
1
2   public class ForRectangle {
3
4     public int height = 3;
5     public int width = 10;
6
7     public void displayRectangle() {
8
9       for (int rowCount = 0; rowCount < height; rowCount++) {
10        for (int colCount = 0; colCount < width; colCount++) {
11          System.out.print("@");
12        }
13        System.out.println();
14      }
15    }
16  }
17
```

Codage d'une boucle `do/while`

Syntaxe :

```
do {

     bloc_de_code;
}
while (expression_booléenne);// Le point-virgule est obligatoire.
```

Codage d'une boucle `do/while`

Exemple :

```
1
2   public class DoWhileElevator {
3
4     public boolean doorOpen=false;
5     public int currentFloor = 1;
6
7     public final int TOP_FLOOR = 5;
8     public final int BOTTOM_FLOOR = 1;
9
10    public void openDoor() {
11      System.out.println("Ouverture de la porte.");
12      doorOpen = true;
13      System.out.println("La porte est ouverte.");
14    }
15
16    public void closeDoor() {
17      System.out.println("Fermeture de la porte.");
```

Codage d'une boucle `do/while`

```
18  doorOpen = false;
19      System.out.println("La porte est fermée.");
20    }
21
22    public void goUp() {
23      System.out.println("Monter d'un étage.");
24      currentFloor++;
25      System.out.println("Étage : " + currentFloor);
26    }
27
28    public void goDown() {
29      System.out.println("Descendre d'un étage.");
30      currentFloor--;
31      System.out.println("Étage : " + currentFloor);
32    }
33
34    public void setFloor() {
35
```

Codage d'une boucle `do/while`

```
36  // Normalement, vous devriez transmettre l'étage désiré (desiredFloor)
37      // sous forme d'argument à la méthode setFloor. Toutefois, comme
38      // vous n'avez pas encore appris à effectuer cette opération,
39      // desiredFloor est défini sur un nombre spécifique (5) ci-dessous.
40
41      int desiredFloor = 5;
42
43      do {
44        if (currentFloor < desiredFloor) {
45          goUp();
46        }
47      else if (currentFloor > desiredFloor) {
48          goDown();
49        }
50      }
51  while (currentFloor != desiredFloor);
52    }
53
54  }
```

Boucles `do/while` imbriquées

Exemple :

```
1
2   public class DoWhileRectangle {
3
4     public int height = 3;
5     public int width = 10;
6
7     public void displayRectangle() {
8
9       int rowCount = 0;
10      int colCount = 0;
11
12      do {
13        colCount = 0;
14
15        do {
16          System.out.print("@");
17          colCount++;
```

Boucles `do/while` imbriquées

```
18  }
19        while (colCount < width);
20
21        System.out.println();
22        rowCount++;
23      }
24      while (rowCount < height);
25    }
26  }
27
```

Comparaison des constructions en boucle

- Utilisez la boucle `while` pour effectuer une itération à travers les instructions indéfiniment et pour exécuter les instructions zéro ou plusieurs fois.
- Utilisez la boucle `do/while` pour effectuer une itération à travers les instructions indéfiniment et pour exécuter les instructions *une* ou plusieurs fois.
- Utilisez la boucle `for` pour parcourir les instructions un nombre de fois prédéfini.

Module 8

Développement et utilisation de méthodes

Présentation

- Objectifs :
  - Décrire les avantages des méthodes et définir des méthodes de travail et d'appel
  - Déclarer et invoquer une méthode
  - Comparer les méthodes d'objet et statiques
  - Utiliser des méthodes surchargées

Pertinence

Comment structurez-vous ou implémentez-vous les opérations exécutées sur un objet ?

Création et invocation de méthodes

Syntaxe :

```
[modificateurs] type_retour identificateur_méthode ([arguments]) {
   bloc_code_méthode
}
```

Forme de base d'une méthode

Exemple :

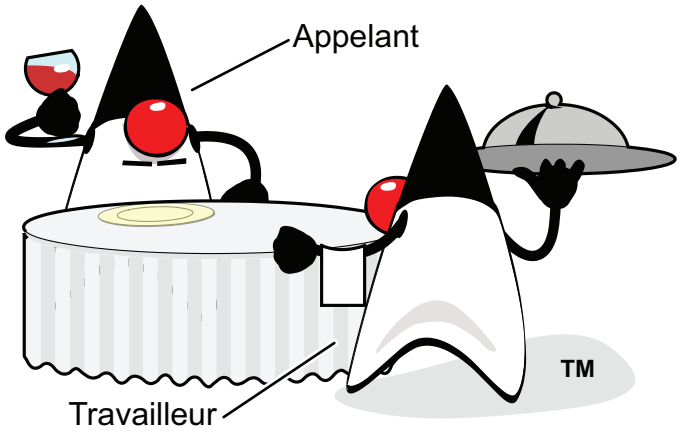
```
public void displayInformation() {
    System.out.println("ID chemise : " + shirtID);
    System.out.println("Description de la chemise :" + description);
    System.out.println("Code couleur : " + colorCode);
    System.out.println("Prix de la chemise : " + price);
    System.out.println("Quantité en stock : " + quantityInStock);
} // fin de la méthode d'affichage
```

Invocation d'une méthode d'une classe différente

Exemple :

```
1
2   public class ShirtTest {
3
4     public static void main (String args[]) {
5
6     Shirt myShirt;
7     myShirt = new Shirt();
8
9     myShirt.displayInformation();
10
11
12     }
13  }
14
```

Méthodes d'appel et de travail



Invocation d'une méthode dans la même classe

Exemple :

```
1
2   public class Elevator {
3
4     public boolean doorOpen=false;
5     public int currentFloor = 1;
6
7     public final int TOP_FLOOR = 5;
8     public final int BOTTOM_FLOOR = 1;
9
10    public void openDoor() {
11      System.out.println("Ouverture de la porte.");
12      doorOpen = true;
13      System.out.println("La porte est ouverte.");
14    }
15
16    public void closeDoor() {
17      System.out.println("Fermeture de la porte.");
```

Invocation d'une méthode dans la même classe

```
18  doorOpen = false;
19      System.out.println("La porte est fermée.");
20    }
21
22    public void goUp() {
23      System.out.println("Monter d'un étage.");
24      currentFloor++;
25      System.out.println("Étage : " + currentFloor);
26    }
27
28    public void goDown() {
29      System.out.println("Descendre d'un étage.");
30      currentFloor--;
31      System.out.println("Étage : " + currentFloor);
32    }
33
34    public void setFloor(int desiredFloor) {
35      while (currentFloor != desiredFloor){
36      if (currentFloor < desiredFloor) {
```

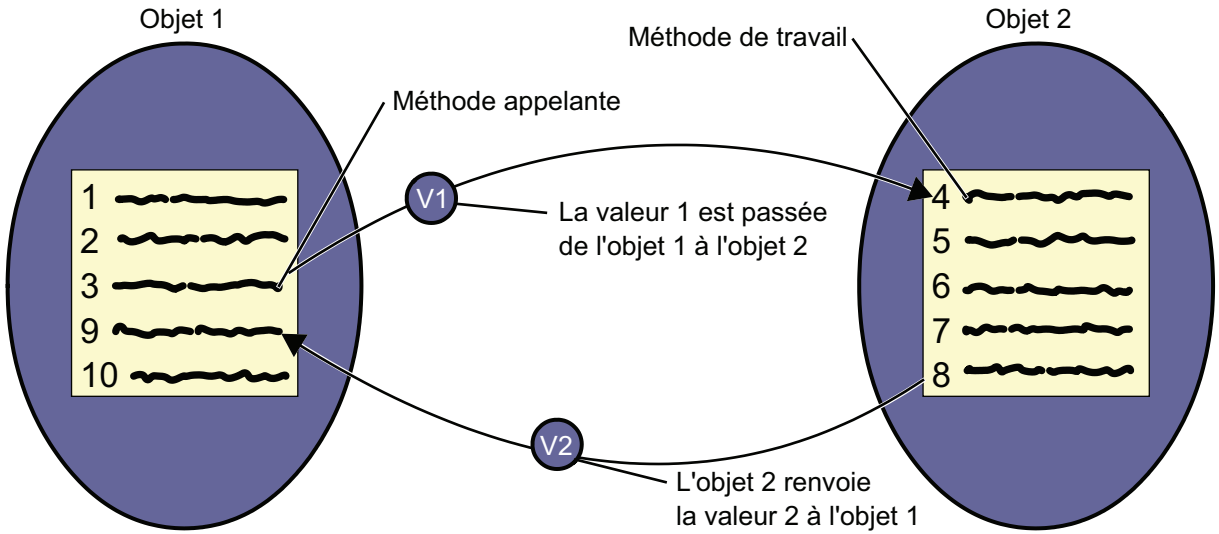
Invocation d'une méthode dans la même classe

```
37          goUp();
38          }
39        else {
40          goDown();
41          }
42      }
43      }
44
45      public int getFloor() {
46        return currentFloor;
47      }
48
49      public boolean checkDoorStatus() {
50        return doorOpen;
51      }
52  }
53
```

Directives relatives à l'invocation des méthodes

- Le nombre d'appels de méthode qu'une méthode d'appel peut invoquer n'est pas limité.
- La méthode d'appel et la méthode de travail peuvent appartenir à la même classe ou à des classes différentes.
- La manière dont vous invoquez la méthode de travail diffère selon si elle se situe dans la même classe ou dans une autre classe que la méthode d'appel.
- Vous pouvez invoquer des méthodes dans n'importe quel ordre. Les méthodes ne doivent pas obligatoirement s'exécuter dans leur ordre de déclaration dans la classe (la classe contenant les méthodes de travail).

Transmission d'arguments et de valeurs de retour



Déclaration de méthodes avec arguments

- Exemple :

```
public void setFloor(int desiredFloor) {
    while (currentFloor != desiredFloor){
      if (currentFloor < desiredFloor) {
       goUp();
      }
      else {
       goDown();
      }
   }
}
```

- Exemple :

```
public void multiply(int numberOne, int numberTwo)
```

Méthode `main`

- Exemple :

```
public static void main (String args[])
```

- Exemple (invocation) :

```
java ShirtTest 12.99 R
```

Invocation de méthodes avec arguments

Exemple :

```
1   public class ElevatorTest {
2
3      public static void main (String args[]) {
4
5       Elevator myElevator = new Elevator();
6
7        myElevator.openDoor();
8        myElevator.closeDoor();
9        myElevator.goUp();
10       myElevator.goUp();
11       myElevator.goUp();
12       myElevator.openDoor();
13       myElevator.closeDoor();
14       myElevator.goDown();
15       myElevator.openDoor();
16       myElevator.closeDoor();
```

Invocation de méthodes avec arguments

```
17        myElevator.goDown();
18
19        myElevator.setFloor(myElevator.TOP_FLOOR);
20
21        myElevator.openDoor();
22      }
23  }
24
```

Déclaration de méthodes avec valeurs de retour

Déclaration :

```
public int sum(int numberOne, int numberTwo)
```

Renvoi d'une valeur

- Exemple :

```
public int sum(int numberOne, int numberTwo) {
     int result= numberOne + numberTwo;

     return result;
}
```

- Exemple :

```
public int getFloor() {
    return currentFloor;
  }
```

Réception des valeurs de retour

Exemple :

```
1
2   public class ElevatorTestTwo {
3
4      public static void main (String args[]) {
5
6       Elevator myElevator = new Elevator();
7
8        myElevator.openDoor();
9        myElevator.closeDoor();
10       myElevator.goUp();
11       myElevator.goUp();
12       myElevator.goUp();
13       myElevator.openDoor();
14       myElevator.closeDoor();
15       myElevator.goDown();
16       myElevator.openDoor();
17       myElevator.closeDoor();
```

Réception des valeurs de retour

```
18  myElevator.goDown();
19
20        int curFloor = myElevator.getFloor();
21        System.out.println("Étage actuel : " + curFloor);
22
23        myElevator.setFloor(curFloor+1);
24
25        myElevator.openDoor();
26     }
27  }
28
```

Avantages de l'utilisation des méthodes

- Grâce aux méthodes, les programmes sont plus lisibles et leur maintenance plus facile.
- Les méthodes accélèrent le développement et la maintenance.
- Les méthodes sont la pièce maîtresse des logiciels réutilisables.
- Les méthodes permettent à des objets distincts de communiquer entre eux et de répartir le travail effectué par le programme.

Création de méthodes et de variables `static`

- Comparaison des méthodes et des variables d'instance et `static`
- Déclaration de méthodes `static` :

  ```
  static Properties getProperties()
  ```

- Invocation de méthodes `static` :

  ```
  Classname.method();
  ```

Création de méthodes et de variables `static`

- Exemple :

```
public static char convertShirtSize(int numericalSize) {

    if (numericalSize < 10) {
      return 'S';
    }

    else if (numericalSize < 14) {
      return 'M';
    }

    else if (numericalSize < 18) {
      return 'L';
    }

    else {
      return 'X';
   }
}
```

Création de méthodes et de variables `static`

- Exemple :

```
char size = Shirt.convertShirtSize(16);
```

Création de méthodes et de variables `static`

- Déclaration de variables `static` :

```
static double tauxTVA = 8.25;
```

- Accès aux variables `static` :

```
Classname.variable;
```

- Exemple :

```
double myPI;
myPI = Math.PI;
```

Méthodes et variables statiques de l'API Java

Exemples :

- Classe `Math`
- Classe `System`

Méthodes et variables statiques de l'API Java

À quel moment doit-on déclarer une méthode ou une variable `static` ?

- Il n'est pas important d'exécuter l'opération sur un objet individuel ou d'associer la variable avec un type d'objet spécifique.
- Il est important d'accéder à la variable ou à la méthode avant l'instanciation d'un objet.
- La méthode ou la variable n'appartient pas de façon logique à un objet, mais éventuellement à une classe d'utilitaire, par exemple à la classe `Math`, incluse dans l'API Java.

Utilisation de la surcharge de méthodes

Exemple de méthodes surchargées :

```
1
2   public class Calculator {
3
4       public int sum(int numberOne, int numberTwo) {
5
6         System.out.println("Méthode 1");
7
8         return numberOne + numberTwo;
9       }
10
11    public float sum(float numberOne, float numberTwo) {
12
13        System.out.println("Méthode 2");
14
15        return numberOne + numberTwo;
16    }
```

Utilisation de la surcharge de méthodes

```
17
18    public float sum(int numberOne, float numberTwo) {
19
20      System.out.println("Méthode 3");
21
22      return numberOne + numberTwo;
23    }
24  }
25
```

Utilisation de la surcharge de méthodes

Exemple d'invocation de méthode :

```
1   public class CalculatorTest {
2
3     public static void main(String [] args) {
4
5       Calculator myCalculator = new Calculator();
6
7       int totalOne = myCalculator.sum(2,3);
8       System.out.println(totalOne);
9
10      float totalTwo = myCalculator.sum(15.99F, 12.85F);
11      System.out.println(totalTwo);
12
13      float totalThree = myCalculator.sum(2, 12.85F);
14      System.out.println(totalThree);
15    }
16  }
```

Surcharge de méthodes et API Java

Méthode	Utilisation
`void println()`	Termine la ligne active en écrivant la chaîne du séparateur de ligne
`void println(boolean x)`	Imprime une `boolean value`, puis termine la ligne
`void println(char x)`	Imprime un caractère, puis termine la ligne
`void println(char[] x)`	Imprime un tableau de caractères, puis termine la ligne
`void println(double x)`	Imprime une valeur `double`, puis termine la ligne
`void println(float x)`	Imprime une valeur `float`, puis termine la ligne
`void println(int x)`	Imprime une valeur `int`, puis termine la ligne
`void println(long x)`	Imprime une valeur `long`, puis termine la ligne
`void println(Object x)`	Imprime un objet, puis termine la ligne
`void println(String x)`	Imprime une chaîne, puis termine la ligne

Utilisations de la surcharge de méthodes

Exemples :

```
public int sum(int numberOne, int numberTwo)
public int sum(int numberOne, int numberTwo, int numberThree)
public int sum(int numberOne, int numberTwo, int numberThree, int numberFour)
```

Utilisations de la surcharge de méthodes

Exemple :

```
1
2   public class ShirtTwo {
3
4     public int shirtID = 0; // ID par défaut de la chemise
5     public String description = "-description required-"; // par défaut
6
7     // Les codes de couleur sont R=Red, B=Blue, G=Green, U=Unset
8     public char colorCode = 'U';
9     public double price = 0.0; // Prix par défaut de tous les articles
10    public int quantityInStock = 0; // Quantité par défaut de tous les
11                                    // articles
12
13    public void setShirtInfo(int ID, String desc, double cost){
14      shirtID = ID;
15      description = desc;
16      price = cost;
17    }
18
```

Utilisations de la surcharge de méthodes

```
19  public void setShirtInfo(int ID, String desc, double cost, char color){
20      shirtID = ID;
21      description = desc;
22      price = cost;
23      colorCode = color;
24    }
25
26    public void setShirtInfo(int ID,  String desc, double cost,
27     char color, int quantity){
28      shirtID = ID;
29      description = desc;
30      price = cost;
31      colorCode = color;
32      quantityInStock = quantity;
33    }
34
35    // Cette méthode affiche les valeurs d'un article
36    public void display(){
37
```

Utilisations de la surcharge de méthodes

```
38      System.out.println("Item ID: " + shirtID);
39      System.out.println("Item description:" + description);
40      System.out.println("Code couleur : " + colorCode);
41      System.out.println("Item price: " + price);
42      System.out.println("Quantité en stock : " + quantityInStock);
43
44    } // fin de la méthode d'affichage
45  } // fin de la classe
46
```

Utilisations de la surcharge de méthodes

Exemple :

```
1   class ShirtTwoTest {
2
3     public static void main (String args[]) {
4       ShirtTwo shirtOne = new ShirtTwo();
5       ShirtTwo shirtTwo = new ShirtTwo();
6       ShirtTwo shirtThree = new ShirtTwo();
7
8       shirtOne.setShirtInfo(100, "Button Down", 12.99);
9       shirtTwo.setShirtInfo(101, "Long Sleeve Oxford", 27.99, 'G');
10      shirtThree.setShirtInfo(102, "Shirt Sleeve T-Shirt", 9.99, 'B', 50);
11
12      shirtOne.display();
13      shirtTwo.display();
14      shirtThree.display();
15    }
16  }
17
```

Module 9

Implémentation de l'encapsulation et des constructeurs

Objectifs

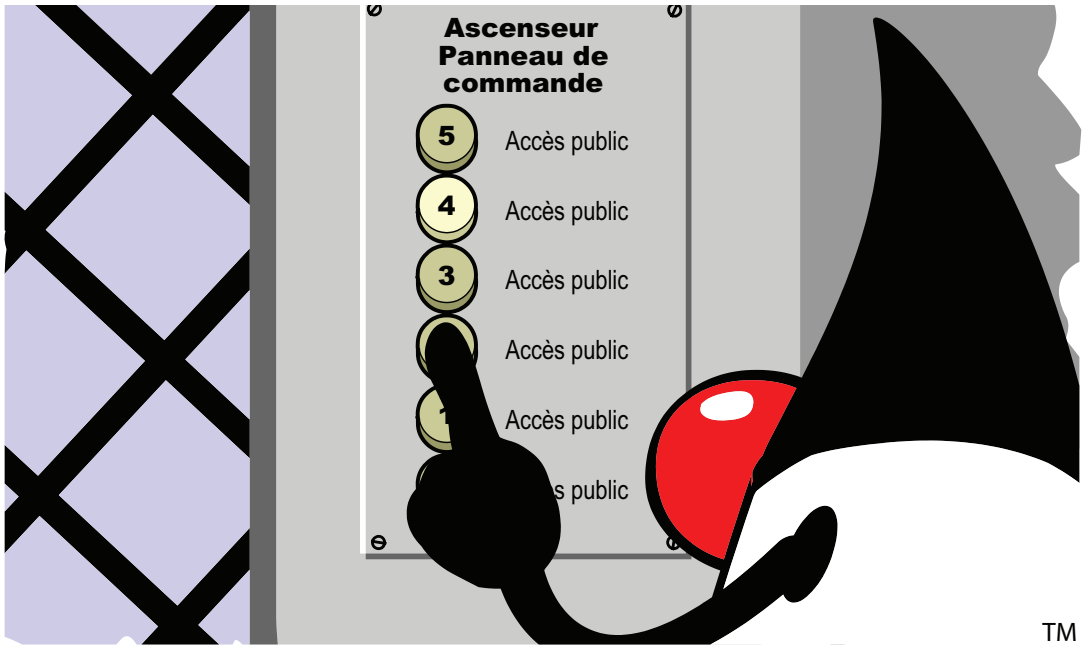
- Utiliser l'encapsulation pour protéger les données
- Créer des constructeurs pour initialiser des objets

Pertinence

- Pour fonctionner, les ascenseurs d'autrefois, ou monte-charges, exigeaient que l'utilisateur manipule des poulies, des cordes et des roues. Les ascenseurs modernes cachent leurs mécanismes et exigent seulement que l'utilisateur appuie sur quelques boutons. Quels sont les avantages des ascenseurs modernes par rapport aux anciens modèles ?
- La plupart des ascenseurs, par exemple l'ascenseur de service d'une usine, imposent l'utilisation de clés pour fonctionner. D'autres imposent l'utilisation d'une clé pour atteindre un étage particulier, par exemple le dernier étage d'un hôtel. Pourquoi ces clés sont-elles importantes ?
- Que vous suggèrent les termes *privé* et *public* ?

Utilisation de l'encapsulation

Modificateur `public`




```
public int currentFloor=1;

public void setFloor(int desiredFloor) {
      ...
}
```

Modificateur `public`

Exemple :

```
1
2   public class PublicElevator {
3
4       public boolean doorOpen=false;
5       public int currentFloor = 1;
6       public int weight =0;
7
8       public final int CAPACITY=1000;
9       public final int TOP_FLOOR = 5;
10      public final int BOTTOM_FLOOR = 1;
11  }
12
```

Modificateur `public`

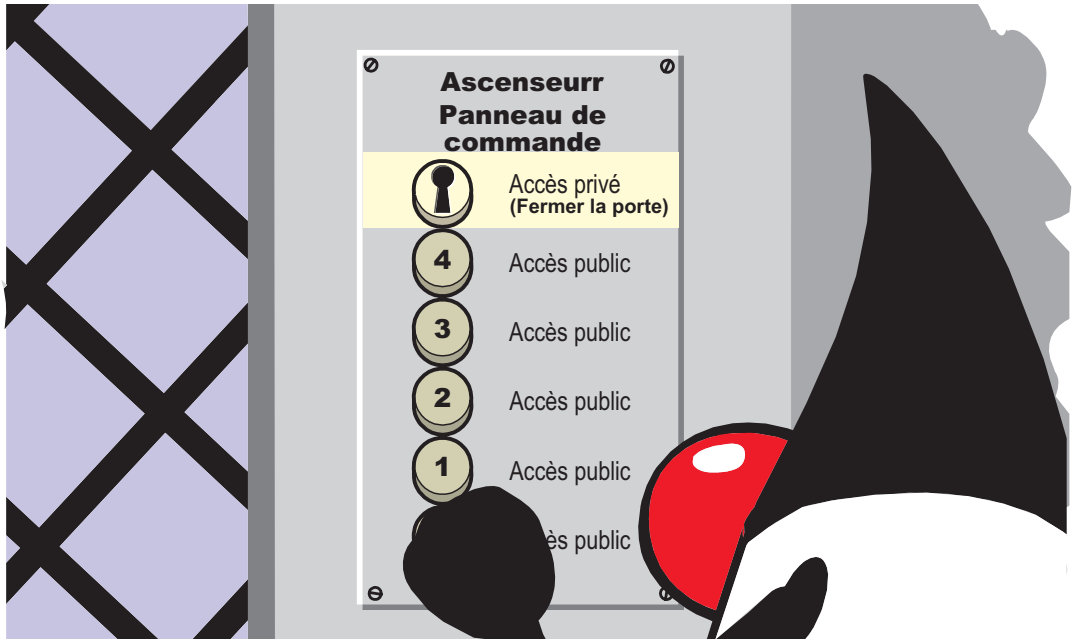
Exemple :

```
1
2   public class PublicElevatorTest {
3
4     public static void main (String args[]) {
5
6       PublicElevator pubElevator = new PublicElevator();
7
8       pubElevator.doorOpen = true;  //Les passagers entrent
9       pubElevator.doorOpen = false; //Les portes se ferment
10      //descente au niveau 0 (au-dessous du rez-de-chaussée)
11      pubElevator.currentFloor--;
12      pubElevator.currentFloor++;
13
14      //accès au niveau 7 (il n'y a que 5 étages dans le bâtiment)
15      pubElevator.currentFloor = 7;
16      pubElevator.doorOpen = true;  //Les passagers entrent/sortent
17      pubElevator.doorOpen = false;
```

Modificateur `public`

```
18  pubElevator.currentFloor = 1; //accès au premier étage
19      pubElevator.doorOpen = true;  //Les passagers entrent/sortent
20      pubElevator.currentFloor++;  //l'ascenseur se déplace avec la porte
21                                   //ouverte
22      pubElevator.doorOpen = false;
23      pubElevator.currentFloor--;
24      pubElevator.currentFloor--;
25    }
26  }
27
```

Modificateur `private`




```
private int currentFloor=1;
private void calculateCapacity() {
    ...
}
```

Modificateur `private`

Exemple :

```
1
2   public class PrivateElevator1 {
3
4      private boolean doorOpen=false;
5      private int currentFloor = 1;
6      private int weight =0;
7
8      private final int CAPACITY=1000;
9      private final int TOP_FLOOR = 5;
10     private final int BOTTOM_FLOOR = 1;
11  }
12
```

Modificateur `private`

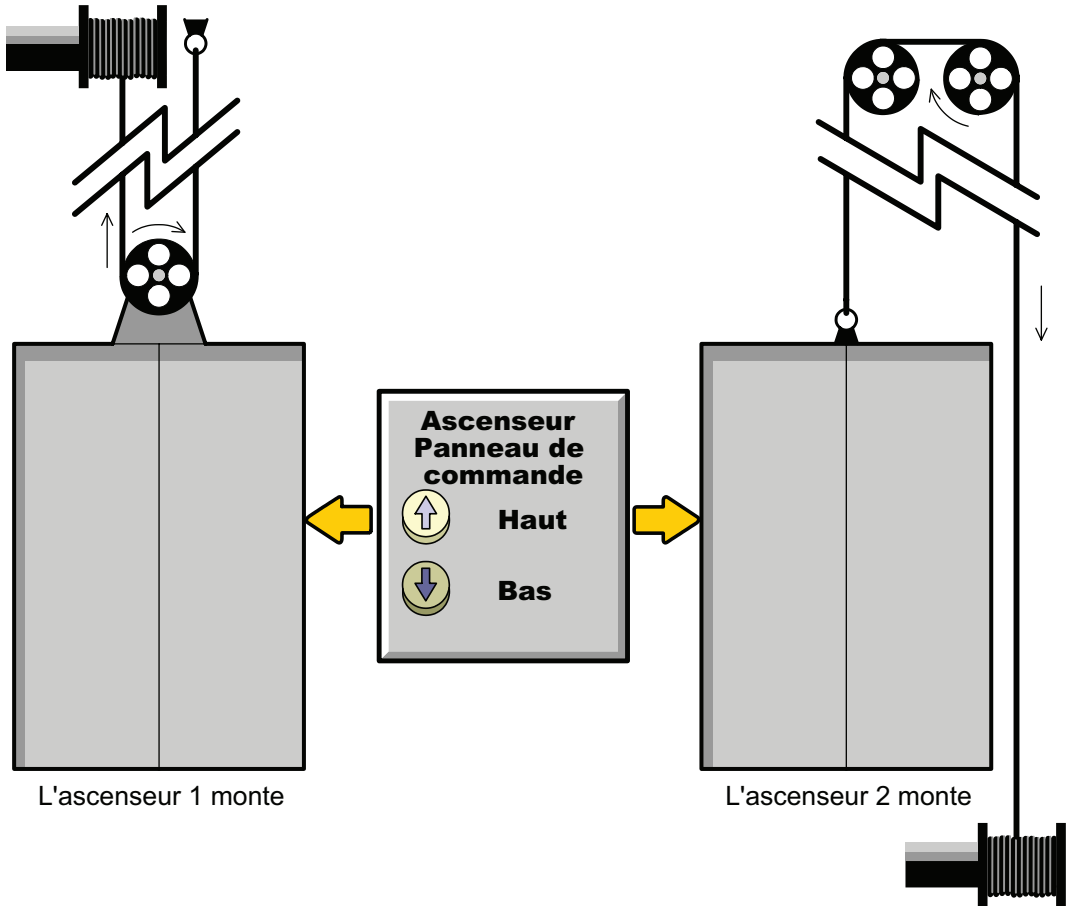
Exemple :

```
1
2   public class PrivateElevator1Test {
3
4     public static void main (String args[]) {
5
6       PrivateElevator1 privElevator = new PrivateElevator1();
7
8       /*****************************************************
9       * Les lignes de code suivantes ne seront pas compilées    *
10      * car elles tentent d'accéder à des variables privées.    *
11      *****************************************************/
12
13      privElevator.doorOpen = true;  //les passagers entrent
14      privElevator.doorOpen = false; //les portes se ferment
15      //descente au niveau 0 (au-dessous du rez-de-chaussée)
16      privElevator.currentFloor--;
```

Modificateur `private`

```
17      privElevator.currentFloor++;
18
19      //accès au niveau 7 (il n'y a que 5 étages dans le bâtiment)
20      privElevator.currentFloor = 7;
21      privElevator.doorOpen = true;  //les passagers entrent/sortent
22      privElevator.doorOpen = false;
23      privElevator.currentFloor = 1; //accès au premier étage
24      privElevator.doorOpen = true;  //les passagers entrent/sortent
25      privElevator.currentFloor++;   //l'ascenseur se déplace avec la
26                                     //porte ouverte
27      privElevator.doorOpen = false;
28      privElevator.currentFloor--;
29      privElevator.currentFloor--;
30    }
31  }
32
```

Interface et implémentation



Interface et implémentation

Exemple :

```
1
2   public class PrivateShirt1 {
3
4     private int shirtID = 0; // ID par défaut de la chemise
5     private String description = "-description required-"; // par défaut
6
7     // Les codes de couleur sont R=Red, B=Blue, G=Green, U=Unset
8     private char colorCode = 'U';
9     private double price = 0.0; // Prix par défaut de tous les articles
10    private int quantityInStock = 0; // Quantité par défaut de tous les
11                                     // articles
12
13    public char getColorCode() {
14      return colorCode;
15    }
16
17    public void setColorCode(char newCode) {
18      colorCode = newCode;
```

Interface et implémentation

```
19  }
20
21    // D'autres méthodes get et set pour l'ID de la chemise, la
22    // description, le prix et la quantité en stock devraient suivre
23
24  } // fin de la classe
25
```

Interface et implémentation

Exemple :

```
1
2   public class PrivateShirt1Test {
3
4     public static void main (String args[]) {
5
6     PrivateShirt1 privShirt = new PrivateShirt1();
7     char colorCode;
8
9     // Définir un code de couleur valide
10    privShirt.setColorCode('R');
11    colorCode = privShirt.getColorCode();
12
13    // La classe PrivateShirtTest1 peut définir un code de couleur valide.
14    System.out.println("Code couleur : " + colorCode);
15
16    // Définit un code de couleur non valide
17    privShirt.setColorCode('Z');
```

Interface et implémentation

```
18  colorCode = privShirt.getColorCode();
19
20    // La classe PrivateShirtTest1 peut définir un code de couleur non valide.
21    System.out.println("Code couleur : " + colorCode);
22    }
23  }
24
```

Interface et implémentation

Exemple :

```
1
2   public class PrivateShirt2 {
3
4     private int shirtID = 0; // ID par défaut de la chemise
5     private String description = "-description required-"; // par défaut
6
7     // Les codes de couleur sont R=Red, B=Blue, G=Green, U=Unset
8     private char colorCode = 'U';
9     private double price = 0.0; // Prix par défaut de tous les articles
10    private int quantityInStock = 0; // Quantité par défaut de tous les
11                                     // articles
12
13    public char getColorCode() {
14      return colorCode;
15    }
16
17    public void setColorCode(char newCode) {
18
```

Interface et implémentation

```
19  switch (newCode) {
20      case 'R':
21      case 'G':
22      case 'B':
23        colorCode = newCode;
24        break;
25      default:
26        System.out.println("Code couleur non valide. Utilisez R, G ou B");
27      }
28    }
29
30    // D'autres méthodes get et set pour l'ID de la chemise, la
31    // description, le prix et la quantité en stock devraient suivre
32
33  } // fin de la classe
34
```

Interface et implémentation

Exemple :

```
1
2   public class PrivateShirt2Test {
3
4     public static void main (String args[]) {
5       PrivateShirt2 privShirt = new PrivateShirt2();
6       char colorCode;
7
8       // Définit un code de couleur valide
9       privShirt.setColorCode('R');
10      colorCode = privShirt.getColorCode();
11
12      // La classe PrivateShirtTest2 peut définir un code de couleur valide.
13      System.out.println("Code couleur : " + colorCode);
14
15      // Définit un code de couleur non valide
16      privShirt.setColorCode('Z');
17      colorCode = privShirt.getColorCode();
```

Interface et implémentation

```
18
19      // La classe PrivateShirtTest2 ne peut pas définir un code de
20      // couleur non valide.
21      // Le code de couleur reste R
22      System.out.println("Code couleur : " + colorCode);
23    }
24  }
25
```

Ascenseur encapsulé

Exemple :

```
1
2   public class PrivateElevator2 {
3
4     private boolean doorOpen=false;
5     private int currentFloor = 1;
6     private int weight =0;
7
8     private final int CAPACITY = 1000;
9     private final int TOP_FLOOR = 5;
10    private final int BOTTOM_FLOOR = 1;
11
12    public void openDoor() {
13      doorOpen = true;
14    }
15
16    public void closeDoor() {
17      calculateCapacity();
```

Ascenseur encapsulé

```
18
19      if (weight <= CAPACITY) {
20        doorOpen = false;
21      }
22      else {
23        System.out.println("La capacité de l'ascenseur est dépassée.");
24        System.out.println("Les portes resteront ouvertes jusqu'à la
25         sortie d'une personne.");
26      }
27    }
28
29    // Dans la pratique, des capteurs de poids permettraient à l'ascenseur
30    // de vérifier le poids réel de l'ascenseur, mais pour plus de
31    // simplicité, nous avons choisi un nombre aléatoire représentant le
32    // poids de l'ascenseur.
33
34     private void calculateCapacity() {
35      weight = (int) (Math.random() * 1500);
36      System.out.println("Le poids est " + weight);
37    }
```

Ascenseur encapsulé

```
38
39    public void goUp() {
40      if (!doorOpen) {
41        if (currentFloor < TOP_FLOOR) {
42        currentFloor++;
43        System.out.println(currentFloor);
44        }
45      else {
46          System.out.println("Déjà au dernier étage.");
47          }
48      }
49      else {
50        System.out.println("Les portes sont encore ouvertes.");
51        }
52      }
53
54    public void goDown() {
55      if (!doorOpen) {
56      if (currentFloor > BOTTOM_FLOOR) {
```

Ascenseur encapsulé

```
57          currentFloor--;
58          System.out.println(currentFloor);
59        }
60      else {
61        System.out.println("Déjà au premier étage.");
62        }
63      }
64      else {
65        System.out.println("Les portes sont encore ouvertes.");
66        }
67    }
68
69    public void setFloor(int desiredFloor) {
70      if ((desiredFloor >= BOTTOM_FLOOR) && (desiredFloor<=TOP_FLOOR)) {
71
72        while (currentFloor != desiredFloor){
73    if (currentFloor < desiredFloor) {
74      goUp();
75    }
```

Ascenseur encapsulé

```
76
77    else {
78    goDown();
79    }
80        }
81      }
82      else {
83        System.out.println("Étage non valide");
84      }
85    }
86
87    public int getFloor() {
88      return currentFloor;
89    }
90
91    public boolean getDoorStatus() {
92      return doorOpen;
93    }
94  }
```

Ascenseur encapsulé

Exemple :

```
1
2   public class PrivateElevator2Test {
3
4      public static void main (String args[]) {
5
6        PrivateElevator2 privElevator = new PrivateElevator2();
7
8        privElevator.openDoor();
9        privElevator.closeDoor();
10       privElevator.goDown();
11       privElevator.goUp();
12       privElevator.goUp();
13       privElevator.openDoor();
14       privElevator.closeDoor();
15       privElevator.goDown();
16       privElevator.openDoor();
17       privElevator.goDown();
```

Ascenseur encapsulé

```
18  privElevator.closeDoor();
19       privElevator.goDown();
20       privElevator.goDown();
21
22       int curFloor = privElevator.getFloor();
23
24       if (curFloor != 5 && ! privElevator.getDoorStatus()) {
25           privElevator.setFloor(5);
26       }
27
28       privElevator.setFloor(10);
29       privElevator.openDoor();
30     }
31  }
32
```

Sortie de test

```
Le poids est 453.
Déjà à l'étage le plus bas.
2
3
Le poids est 899.
2
Les portes sont encore ouvertes.
Le poids est 974.
1
Déjà à l'étage le plus bas.
2
3
4
5
```

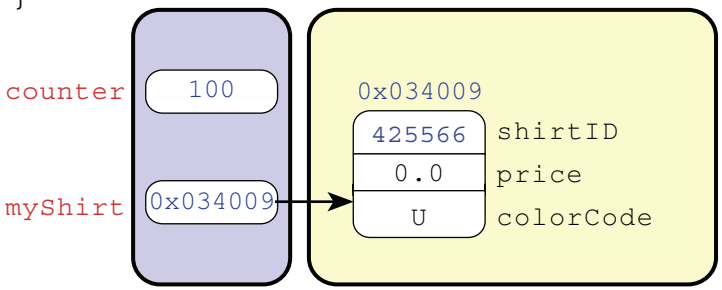
Description de l'étendue des variables

Exemple :

```
1   public class Person2 {
2
3     // début de l'étendue de la variable int age
4     private int age = 34;
5
6     public void displayName() {
7       // début de l'étendue de la variable String name
8       String name = "Peter Simmons";
9       System.out.println("Mon nom est " + name + " et j'ai " + age ans);
10
11    }   // fin de l'étendue de la variable String name
12
13    public String getName () {
14
15      return name; // entraîne une erreur
16    }
17  }   // fin de l'étendue de la variable int age
```

Présence en mémoire des variables d'instance et des variables locales

```
public static void main (String args[]) {

  int counter = 100;
  Shirt myShirt = new Shirt ( );
  myShirt.shirtID = 425566 ;
}
```



Création de constructeurs

Syntaxe :

```
[modificateurs] class NomClasse {

     [modificateurs] NomConstructeur([arguments]) {
       bloc_de_code
     }
}
```

Création de constructeurs

Exemple :

```
1
2   public class ConstructorShirt1 {
3
4     private int shirtID = 0; // ID par défaut de la chemise
5     private String description = "-description required-"; // par défaut
6
7     // Les codes de couleur sont R=Red, B=Blue, G=Green, U=Unset
8     private char colorCode = 'U';
9     private double price = 0.0; // Prix par défaut de tous les articles
10    private int quantityInStock = 0; // Quantité par défaut de tous les
11                                     // articles
12
13    public ConstructorShirt1(char startingCode) {
14
15      switch (startingCode) {
16      case 'R':
17      case 'G':
18      case 'B':
```

Création de constructeurs

```
19  colorCode = startingCode;
20        break;
21      default:
22        System.out.println("Code couleur non valide. Utilisez R, G ou B");
23      }
24    }
25
26    public char getColorCode() {
27      return colorCode;
28    }
29  } // fin de la classe
30
```

Création de constructeurs

Exemple :

```
1
2   public class ConstructorShirt1Test {
3
4     public static void main (String args[]) {
5
6       ConstructorShirt1 constShirt = new ConstructorShirt1('R');
7       char colorCode;
8
9       colorCode = constShirt.getColorCode();
10
11      System.out.println("Code couleur : " + colorCode);
12    }
13  }
14
```

Constructeur par défaut

- Exemple :

```
ConstructorShirt1 constShirt = new ConstructorShirt1();
```

- Exemple :

```
1
2   public class DefaultShirt {
3
4     private int shirtID = 0; // ID par défaut de la chemise
5     private String description = "-description required-"; // par défaut
6
7     // Les codes de couleur sont R=Red, B=Blue, G=Green, U=Unset
8     private char colorCode = 'U';
9     private double price = 0.0; // Prix par défaut de tous les articles
10    private int quantityInStock = 0; // Quantité par défaut de tous les
11                                     // articles
12
13    public DefaultShirt() {
14      colorCode = 'R';
15    }
```

Constructeur par défaut

```
16
17    public char getColorCode() {
18      return colorCode;
19    }
20  } // fin de la classe
21
```

Surcharge des constructeurs

Exemple :

```
1
2   public class ConstructorShirt2 {
3
4     private int shirtID = 0; // ID par défaut de la chemise
5     private String description = "-description required-"; // par défaut
6
7     // Les codes de couleur sont R=Red, B=Blue, G=Green, U=Unset
8     private char colorCode = 'U';
9     private double price = 0.0; // Prix par défaut de tous les articles
10    private int quantityInStock = 0; // Quantité par défaut de tous les
11                                     // articles
12
13    public ConstructorShirt2() {
14      colorCode = 'R';
15    }
16
17    public ConstructorShirt2 (char startingCode) {
18
```

Surcharge des constructeurs

```
19  switch (startingCode) {
20      case 'R':
21      case 'G':
22      case 'B':
23        colorCode = startingCode;
24        break;
25      default:
26        System.out.println("Code couleur non valide. Utilisez R, G ou B");
27      }
28    }
29    public ConstructorShirt2 (char startingCode, int startingQuantity) {
30
31      switch (startingCode) {
32      case 'R':
33        colorCode = startingCode;
34        break;
35      case 'G':
36        colorCode = startingCode;
37        break;
```

Surcharge des constructeurs

```
38  case 'B':
39        colorCode = startingCode;
40        break;
41      default:
42        System.out.println("Code couleur non valide. Utilisez R, G ou B");
43      }
44
45      if (startingQuantity > 0 && startingQuantity < 2000) {
46        quantityInStock = startingQuantity;
47      }
48
49      else {
50        System.out.println("Quantité non valide. Doit être > 0 ou < 2000");
51      }
52    }
53
54    public char getColorCode() {
55      return colorCode;
56      }
```

Surcharge des constructeurs

```
57  public int getQuantityInStock() {
58        return quantityInStock;
59    }
60
61  } // fin de la classe
62
```

Surcharge des constructeurs

Exemple :

```
1
2   public class ConstructorShirt2Test {
3
4     public static void main (String args[]) {
5
6      ConstructorShirt2 constShirtFirst = new ConstructorShirt2();
7      ConstructorShirt2 constShirtSecond = new ConstructorShirt2('G');
8      ConstructorShirt2 constShirtThird = new ConstructorShirt2('B', 1000);
9
10      char colorCode;
11      int quantity;
12
13      colorCode = constShirtFirst.getColorCode();
14      System.out.println("Code couleur de l'objet 1 : " + colorCode);
15
16      colorCode = constShirtSecond.getColorCode();
```

Surcharge des constructeurs

```
17  System.out.println("Code couleur de l'objet 2 : " + colorCode);
18
19      colorCode = constShirtThird.getColorCode();
20      quantity = constShirtThird.getQuantityInStock();
21      System.out.println("Code couleur de l'objet 3 : " + colorCode);
22      System.out.println("Quantité en stock de l'objet 3 : " + quantity);
23    }
24  }
25
```

Module 10

Création et utilisation de tableaux

Objectifs

- Coder des tableaux unidimensionnels
- Définir les valeurs des tableaux à l'aide de l'attribut `length` et d'une boucle
- Transmettre des arguments à la méthode `main` en vue de leur utilisation dans un programme
- Créer des tableaux bidimensionnels

Pertinence

- Un tableau est une organisation ordonnée d'éléments, par exemple une liste triée. Dans notre vie quotidienne, à quelles occasions utilisons-nous des tableaux ?
- Si un tableau unidimensionnel est une liste d'éléments, qu'est-ce qu'un tableau bidimensionnel ?
- Comment peut-on accéder aux éléments d'un tableau ?

Création de tableaux unidimensionnels

Exemple :

```
int ageOne = 27;
int ageTwo = 12;
int ageThree = 82;
int ageFour = 70;
int ageFive = 54;
int ageSix = 6;
int ageSeven = 1;
int ageEight = 30;
int ageNine = 34;
int ageTen = 42;
```

Création de tableaux unidimensionnels

Tableau de `int`

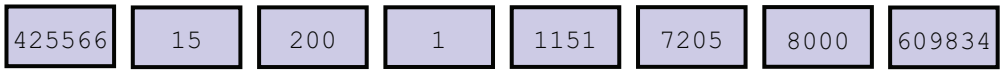


Tableau de `chemises`

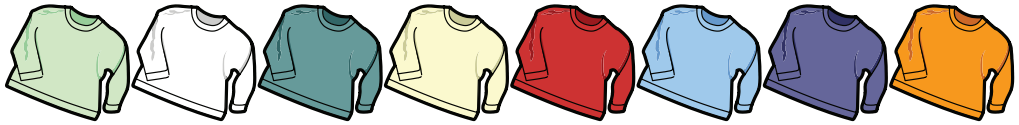


Tableau de `chaînes`

Déclaration d'un tableau unidimensionnel

- Syntaxe :

  type [] identificateur_tableau;

- Exemples :

```
char [] status;
int [] ages;

Shirt [] shirts;
String [] names;
```

Instanciation d'un tableau unidimensionnel

- Syntaxe :

  identificateur_tableau = new type [longueur];

- Exemples :

```
status = new char [20];
ages = new int [5];

names = new String [7];
shirts = new Shirt [3];
```

Initialisation d'un tableau unidimensionnel

- Syntaxe :

  identificateur_tableau[index] = valeur;

- Exemples :

```
ages[0] = 19;
ages[1] = 42;
ages[2] = 92;
ages[3] = 33;
ages[4] = 46;

shirts[0] = new Shirt();
shirts[1] = new Shirt('G');
shirts[2] = new Shirt('G', 1000);
```

Déclaration, instanciation et initialisation de tableaux unidimensionnels

- Syntaxe :

type [] *identificateur_tableau* = {*liste de valeurs ou d'expressions séparées par des virgules*};

- Exemples :

```
int [] ages = {19, 42, 92, 33, 46};

Shirt [] shirts = { new Shirt(), new Shirt('G'), new Shirt('G',1000) };
```

- Erreur :

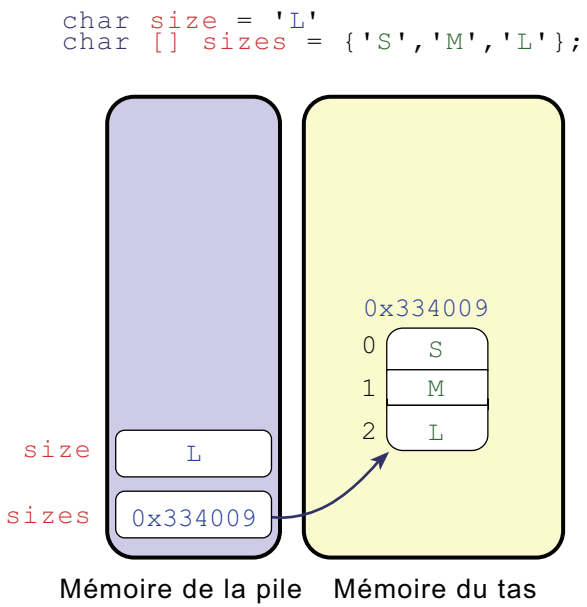
```
int [] ages;
ages = {19, 42, 92, 33, 46};
```

Accès à une valeur dans un tableau

Exemples :

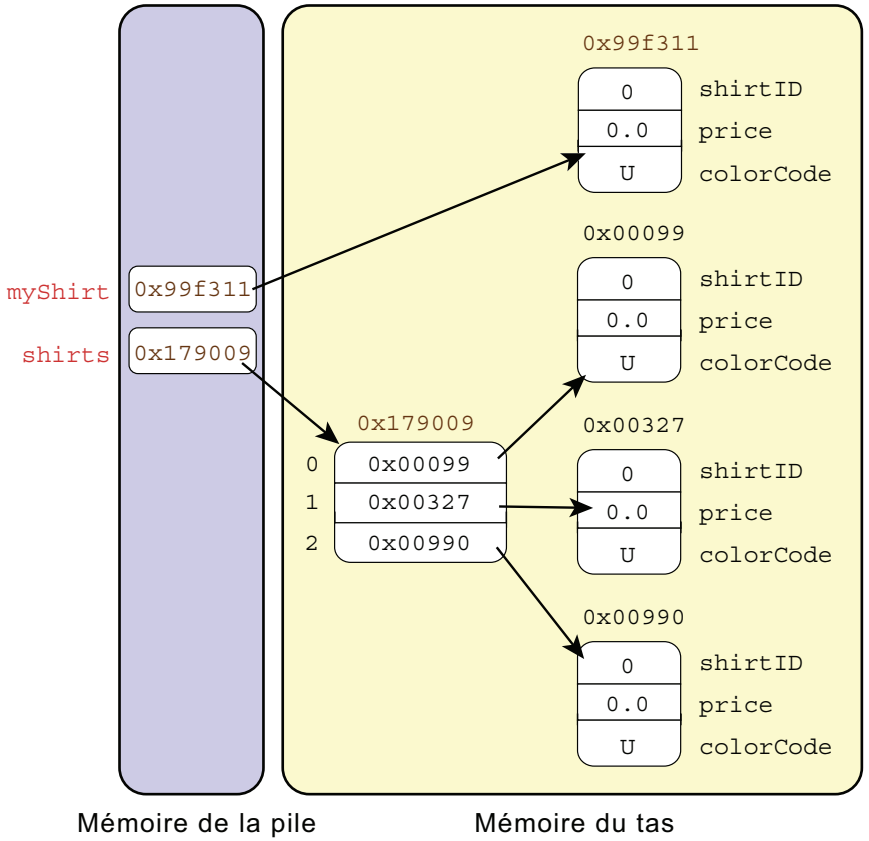
```
status[0] = '3';
names[1] = "Fred Smith";
ages[1] = 19;
prices[2] = 9.99F;

char s = status[0];
String name = names [1];
int age = ages[1];
double price = prices[2];
```

Stockage de variables primitives et de tableaux de primitives en mémoire



Stockage de tableaux de références en mémoire

```
1 Shirt myShirt = new Shirt();
2 Shirt [] shirts = {new Shirt(),
                     new Shirt(),
                     new Shirt()};
```



Définition des valeurs d'un tableau à l'aide de l'attribut `length` et d'une boucle

Exemple :

```
int [] myArray;
myArray = new int[100];

for (int count = 0; count < myArray.length; count++) {
     myArray[count] = count;
}
```

Boucle For optimisée

- La boucle for optimisée permet de créer des boucles plus compactes et plus faciles à lire
- Cette forme de boucle for est spécialement conçue pour effectuer des itérations dans des tableaux.
- Exemple :

```
class ExampleFor {
     public static void main(String [] args) {
          int[] numbers = {1,3,5,7,9,11,13,15,17,19};
          int sum=0;
          for (int item : numbers) {
           sum = sum + item;
          }
          System.out.println("La somme est : " + sum);

     }
}
```

Utilisation du tableau `args` dans la méthode `main`

- Exemple :

```
public static void main (String args[]);
```

- Exemple :

```
1   public class ArgsTest {
2
3     public static void main (String args[]) {
4
5       System.out.println("args[0] est " + args[0]);
6       System.out.println("args[1] est " + args[1]);
7     }
8   }
9
```

Conversion d'arguments `String` en d'autres types

Exemple :

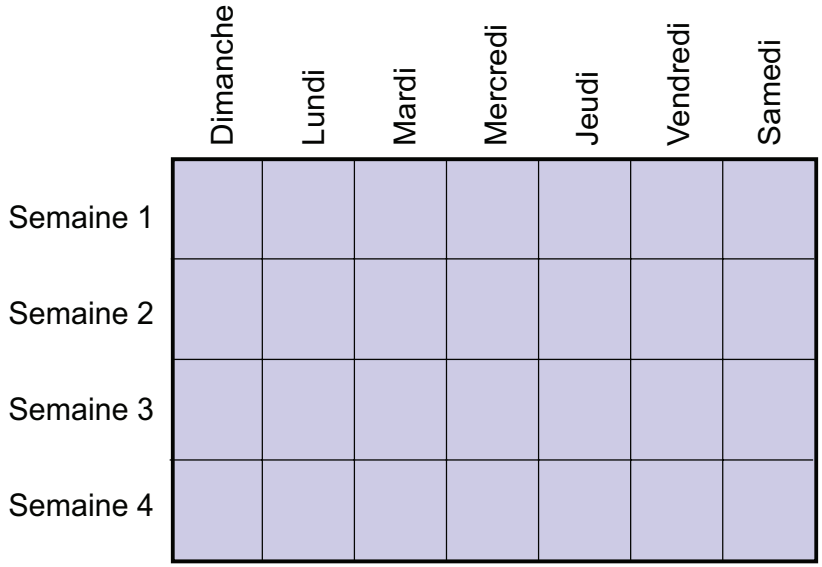
```
int ID = Integer.parseInt(args[0]);
```

Fonctionnalité varargs

- Vous pouvez créer une méthode capable d'accepter un nombre variable d'arguments.
- Une méthode peut avoir au moins un paramètre vararg
- Vararg doit être le dernier paramètre accepté par la méthode. Il est indiqué par le type d'objet, un ensemble d'ellipses (...), et le nom de la variable. Par exemple :

```
class VarMessage{
   public static void showMessage(String... names) {
     for (String list: names)
       System.out.println(list);
   }
   public static void main (String args[]){
     showMessage (args)
}
```

Description des tableaux bidimensionnels



Déclaration d'un tableau bidimensionnel

- Syntaxe :

  type [][] identificateur_tableau;

- Exemple :

```
int [][] yearlySales;
```

Instanciation d'un tableau bidimensionnel

- Syntaxe :

```
identificateur_tableau= new type [nombre_de_tableaux] [longueur];
```

- Exemple :

```
// Instanciation d'un tableau bidimensionnel : 5 tableaux de 4 éléments chacun
yearlySales = new int[5][4];
```

	1er Trimestre	2ème Trimestre	3ème Trimestre	4ème Trimestre
Année 1				
Année 2				
Année 3				
Année 4				
Année 5				

Initialisation d'un tableau bidimensionnel

Exemple :

```
yearlySales[0][0] = 1000;
yearlySales[0][1] = 1500;
yearlySales[0][2] = 1800;
yearlySales[1][0] = 1000;
yearlySales[2][0] = 1400;
yearlySales[3][3] = 2000;
```

	1er Trimestre	**2ème Trimestre**	**3ème Trimestre**	**4ème Trimestre**
Année 1	1000	1500	1800	
Année 2	1000			
Année 3	1400			
Année 4				2000
Année 5				

Module 11

Implémentation de l'héritage

Objectifs

- Définir et tester votre utilisation de l'héritage
- Expliquer le concept d'abstraction
- Identifier explicitement les bibliothèques de classes utilisées dans votre code

Pertinence

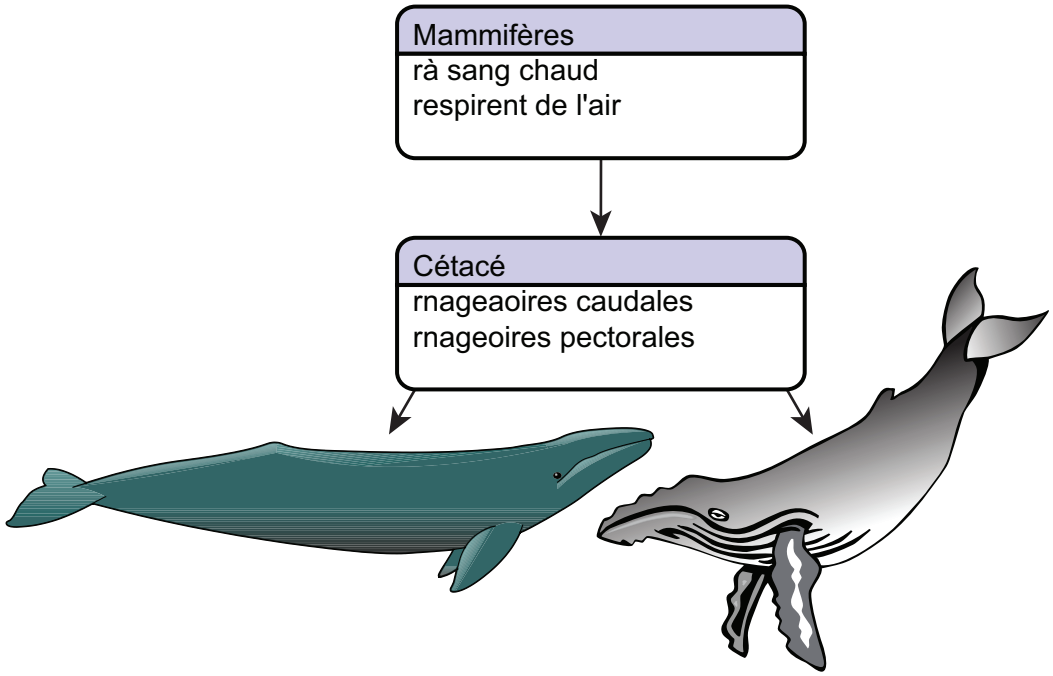
- L'héritage consiste à transmettre un élément d'un organisme à un autre. Pouvez-vous citer quelques caractéristiques physiques dont vous avez hérité ?
- De qui avez-vous hérité vos caractéristiques ?
- De quelle hiérarchie de classe provenez-vous ?
- Avez-vous hérité de caractéristiques provenant de plusieurs classes ?
- Que signifie le terme « abstrait » ?
- Que signifie pour vous classe abstraite ?

de l'héritage

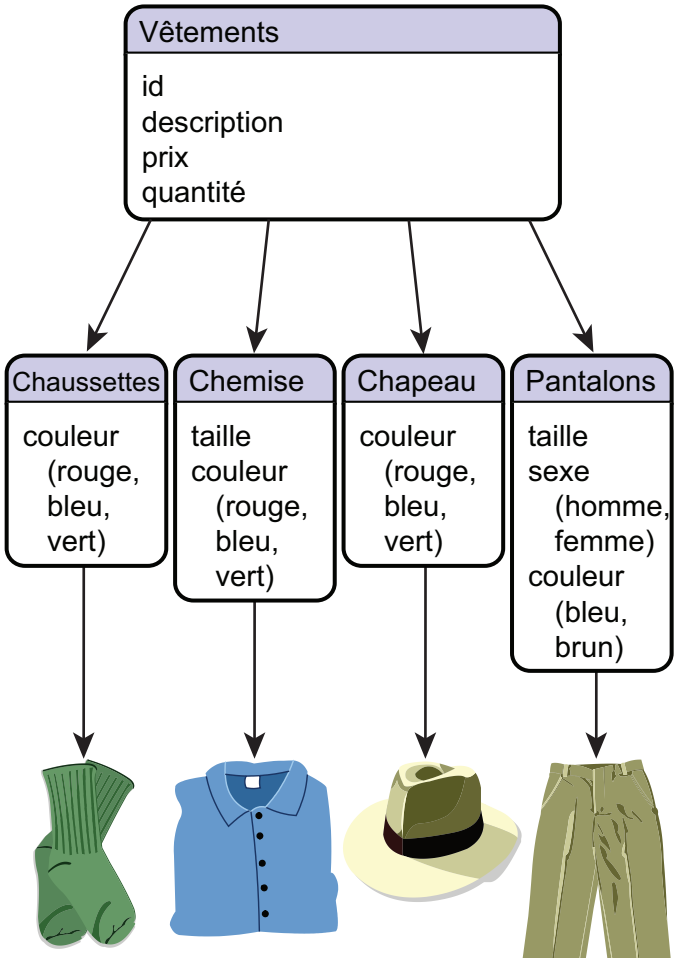
Hat	**Sock**
ID prix description codeCouleur R=Rouge, B=Bleu, G=Vert quantityInStock	ID prix description codeCouleur R=Rouge, B=Bleu, G=Vert quantityInStock
calculateID() afficherInformations()	calculateID() afficherInformations()

Pant	**Chemise**
ID prix taille gender M=Male, F=Female description colorCode B=Blue, T=Tan quantityInStock	ID prix description codeCouleur R=Rouge, B=Bleu, G=Vert quantityInStock
calculateID() afficherInformations()	calculateID() afficherInformations()

Superclasses et sous-classes



Test des relations entre superclasses et sous-classes

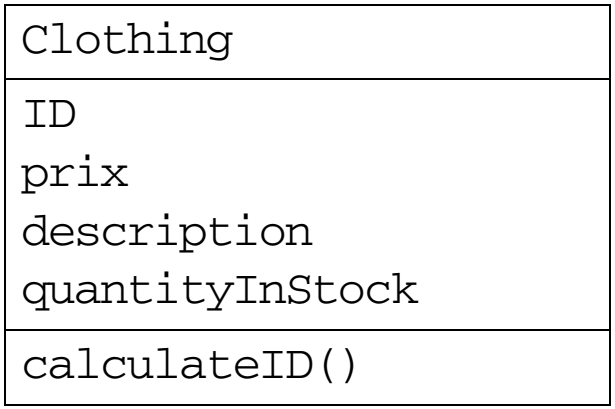


Modélisation des superclasses et des sous-classes

Hat:Clothing	Socks:Clothing
codeCouleur R=Rouge, B=Bleu, G=Vert	codeCouleur R=Rouge, B=Bleu, G=Vert
afficherInformations()	afficherInformations()

Pants:Clothing	Shirt:Clothing
taille gender M=Male, F=Female colorCode B=Blue, T=Tan	taille codeCouleur R=Rouge, B=Bleu, G=Vert
afficherInformations()	afficherInformations()

Modélisation des superclasses et des sous-classes



Déclaration d'une superclasse

Exemple :

```
1
2   public class Clothing {
3
4     private int ID = 0; // ID par défaut de tous les vêtements
5     private String description = "-description required-"; // par défaut
6
7     private double price = 0.0; // Prix par défaut de tous les vêtements
8     private int quantityInStock = 0; // Quantité par défaut de tous les
9                                      // vêtements
10
11    private static int UNIQUE_ID=0; //Membre statique incrémenté dans le
12                                    //constructeur pour générer un
13                                    //identifiant unique
14
15    public Clothing() {
16      ID = UNIQUE_ID++;
17    }
18
19  public int getID() {
```

Déclaration d'une superclasse

```
20  return ID;
21  }
22
23    public void setDescription(String d) {
24      description = d;
25    }
26
27    public String getDescription() {
28      return description;
29    }
30
31    public void setPrice(double p) {
32      price = p;
33    }
34
35    public double getPrice() {
36      return price;
37    }
38
```

Déclaration d'une superclasse

```
39  public void setQuantityInStock(int q) {
40  quantityInStock = q;
41    }
42
43    public int getQuantityInStock() {
44      return quantityInStock;
45    }
46
47  } // fin de la classe
48
```

Déclaration d'une sous-classe

Syntaxe :

```
[modificateur_classe] class identificateur_classe extends
identificateur_superclasse
```

Déclaration d'une sous-classe

Exemple :

```
1   public class Shirt extends Clothing {
2
3     // Les codes de couleur sont R=Red, B=Blue, G=Green, U=Unset
4     public char colorCode = 'U';
5
6     // Cette méthode affiche les valeurs d'un article
7     public void displayInformation() {
8
9       System.out.println("ID chemise : " + getID());
10      System.out.println("Description de la chemise :" + getDescription());
11      System.out.println("Code couleur : " + colorCode);
12      System.out.println("Prix de la chemise : " + getPrice());
13      System.out.println("Quantité en stock : " + getQuantityInStock());
14
15    } // fin de la méthode d'affichage
16  } // fin de la classe
17
```

Abstraction

- Qu'est-ce que l'abstraction ?
- Abstraction dans l'étude de cas DirectClothing, Inc.

Classes de l'API Java

- Classes disponibles implicitement : Le package `java.lang`
- Importation et qualification des classes :
  - Le package `java.awt`
  - Le package `java.applet`
  - Le package `java.net`
  - Le package `java.io`
  - Le package `java.util`

Instruction `import`

- Syntaxe :

```
import nom_package.nom_classe;
import nom_package.*;
```

- Exemple :

```
import java.awt.*;
public class MyPushButton1 extends Button {
    // instructions de la classe

}
```

Spécification du nom complet

- Syntaxe :

nom_package.nom_classe

- Exemple :

```
public class MyPushButton2 extends java.awt.Button {
    // instructions de la classe
}
```