Cours JAVA : Le bases du langage Java.

Version 3.02

Julien Sopena[1]

[1]julien.sopena@lip6.fr
Équipe REGAL - INRIA Rocquencourt
LIP6 - Université Pierre et Marie Curie

Licence professionnelle DANT - 2013/2014

Grandes lignes du cours

- **Java en quelques mots**
- **Comparatif Java et C++**
- **Programmation orientée objets.**
  - Conception par traitements.
  - Conception par objets.
  - Le concept d'encapsulation.
- **Les classes Java**
- **Les attributs.**
- **Les méthodes.**
  - Le mot clé this.
  - Les accesseurs.
  - Les méthodes de classe.
  - La surcharge.
- **Les constructeurs.**
  - Le mécanisme d'instanciation.
  - Constructeur par défaut.
  - Plusieurs constructeurs.
- **Exécutable Java.**
  - Coder un executable.
  - Compilation.
  - Structuration des sources.
- **Des classes utiles.**
  - La classe String
  - Les tableaux.
  - Les enveloppes.

Outline

Java c'est quoi ?

- Un langage : Orienté objet fortement typé avec classes
- Un environnement d'exécution (JRE) : Une machine virtuelle et un ensemble de bibliothèques
- Un environnement de développement (JDK) : Une machine virtuelle et un ensemble d'outils
- Une mascotte : Duke

Java c'est qui ?

La plate-forme et le langage Java sont issus d'un projet de *Sun Microsystems* datant de 1990.
Généralement, on attribut sa paternité a trois de ses ingénieurs :

- James Gosling
- Patrick Naughton
- Mike Sheridan

Figure : 1990 Barbecue chez James Gosling

Java pourquoi ?

Java est devenu aujourd'hui l'un des langages de programmation les plus utilisés.

Il est incontournable dans plusieurs domaines :

- **Systèmes dynamiques :** Chargement dynamique de classes
- **Internet :** Les *Applets* java
- **Systèmes communicants :** *RMI*, *Corba*, *EJB*, etc.

Java pour qui ?

Pour tous : Le 13 novembre 2006, Sun annonce le passage de Java, c'est-à-dire le JDK (JRE et outils de développement) sous **licence GPL**.

Pour vous : Cette UE sur Java servira de base à l'ensemble des UE techniques du deuxième semestre.

L'environnement actuel Java 2 Standard Edition

J2SE

L'outil de base : le JDK (Java Development Kit) de SUN :

- `http://java.sun.com`.
- gratuit.
- Dernière version : 1.6.
- comprend de nombreux outils :
  - le compilateur.
  - le compilateur à la volé "JIT".
  - le débogueur.
  - le générateur de documentation.

Des environnements de développements gratuits

- NetBeans : `http://www.netbeans.org/`
- Eclipse : `http://www.eclipse.org/`

Java évolue tout le temps

Java n'est pas un langage normalisé et il continu d'évoluer. Cette évolution se fait en ajoutant de **nouvelle API**, mais aussi en **modifiant la machine virtuelle**.

L'ensemble de ces modifications est géré par le **JCP (Java Community Process ; http ://www.jcp.org/)** dans lequel Sun continue de tenir une place prépondérante.

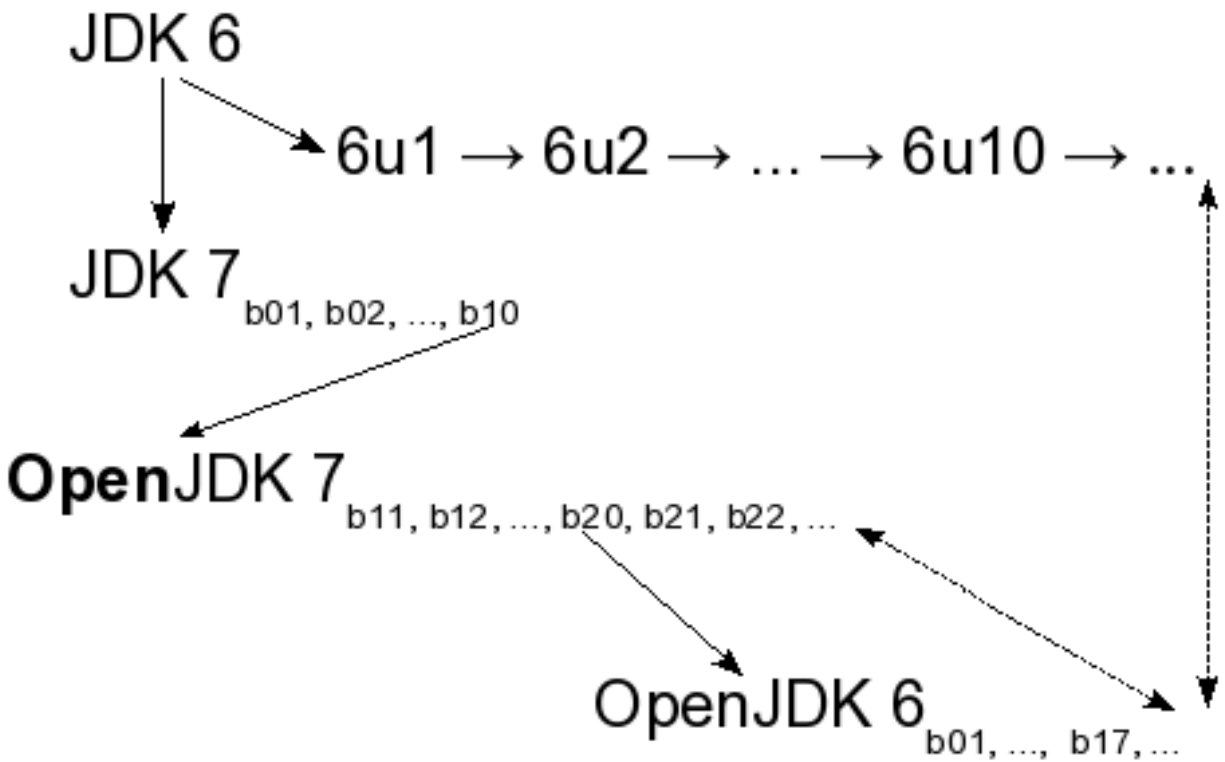
Il peut alors être nécessaire d'identifier une version précise du compilateur et/ou de la machine virtuelle : Ça n'est pas simple.

La numérotation des versions :

$$1.0 \longrightarrow 1.1 \longrightarrow \underbrace{1.2 \longrightarrow 1.3 \longrightarrow 1.4 \longrightarrow 5.0}_{\text{Toutes ces versions : } \textbf{Java 2}} \longrightarrow 6.0$$

Tout se complique

Attention, avec l'arrivée de la GPL tout se complique :



Les versions de Java

JDK 1.0 (1996 - 211 classes et interfaces)
Version initiale.

JDK 1.1 (1997 - 477 classes et interfaces)
Ajoute : classes internes, JavaBeans, JDBC, Java Remote Invocation (RMI).

J2SE 1.2 (1998 - 1 524 classes et interfaces) – Playground
Ajoute : réflection, SWING, compilateur JIT (Just in Time), Java IDL pour Corba.

J2SE 1.3 (2000 - 1 840 classes et interfaces) – Kestrel
Ajoute : HotSpot JVM, service de nomage (JNDI) et JavaSound.

Les versions de Java (suite)

J2SE 1.4 (2002 - 2 723 classes et interfaces) – Merlin
Ajoute : mot-clé assert, expressions rationnelles, chaînage d–exception, parser XML et du moteur XSLT (JAXP), extensions de sécurité JCE (Java Cryptography Extension) et Java Web Start.

J2SE 5.0 (2004 - 3 270 classes et interfaces) – Tiger
Ajoute : syntaxe à la foreach, enumerations (*enum*), classe Integer, autoboxing/unboxing

Java SE 6 (2006 - 3 777 classes et interfaces) – Mustang
Ajoute : covariance (redéfinition avec modification du type de retour), @overhiding.

Java SE 7 – Nom de code Dolphin
Ajoutera : des closures (en cours de spécifications).
Ce sera la première Version 100% open source.

Outline

Java *versus* C++

- Filiation historique :
  - **1983** (AT&T Bell) : C++
  - **1991** (Sun Microsystems) : Java
- Java est très proche du langage C++ (et donc du langage C).
- Toutefois Java est plus simple que le langage C++, car les points "critiques" du langage C++ (ceux qui sont à l'origine des principales erreurs) ont été supprimés.
- Cela comprend :
  - Les pointeurs
  - La surcharge d'opérateurs
  - L'héritage multiple

Java *versus* C++ : concepts (2)

De plus,

- ✍ Tout est dynamique : les instances d'une classe sont instanciées dynamiquement.
- ✍ La libération de mémoire est transparente pour l'utilisateur. Il n'est pas nécessaire de spécifier de mécanisme de destruction. La libération de l'espace mémoire est prise en charge un gestionnaire appelé garbage collector ⇒ chargé de détecter les objets à détruire.

Notes

- gain de fiabilité (pas de désallocation erronée).
- a un coût (perte en rapidité par rapport au C++).

Java *versus* C++ : production du binaire

Une fois achevée la production du logiciel, un choix doit être fait entre fournir le source ou le binaire pour la machine du client.

Généralement, une entreprise souhaite protéger le code source et distribuer le code binaire.

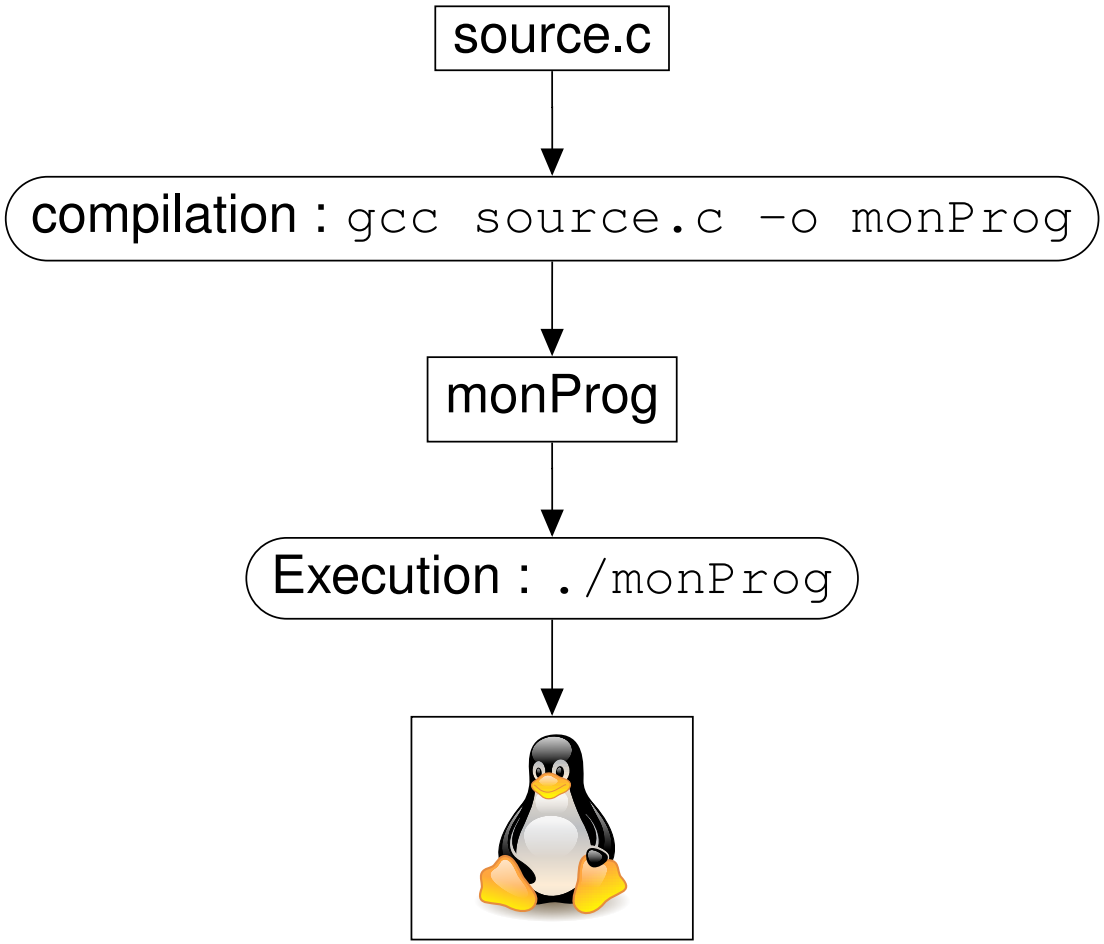
Le code binaire doit donc être portable sur des architectures différentes (processeur, système d'exploitation, etc.).

Java *versus* C++ : compilateur C++

À l'instar du compilateur C, le compilateur C++ produit du code natif, *i.e.*, qu'il produit un exécutable propre à l'environnement de travail ou le code source est compilé.

On doit donc créer les exécutables pour chaque type d'architecture potentielle des clients.

Java *versus* C++ : chaîne de production du C



Java *versus* C++ : compilateur Java

En Java, le code source n'est pas traduit directement dans le langage de l'ordinateur.

Il est d'abord traduit dans un langage appelé "bytecode", langage d'une machine virtuelle (JVM – Java Virtual Machine) définie par Sun.

Portabilité

Le *bytecode* généré par le compilateur ne dépend pas de l'architecture de la machine où a été compilé le code source, c'est-à-dire que les bytecodes produits sur une machine pourront s'exécuter (au travers d'une machine virtuelle) sur des architectures différentes.

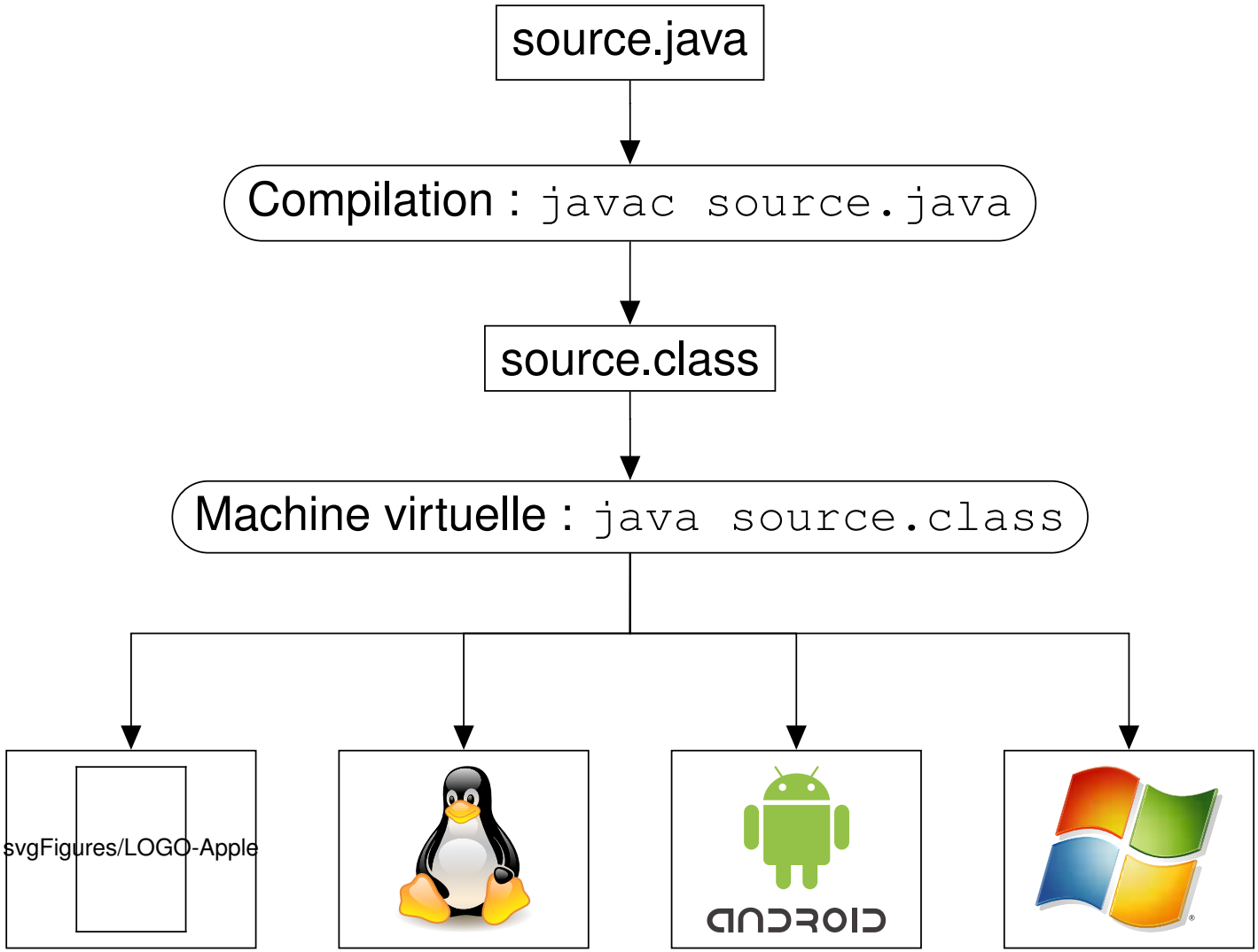
Exécution du bytecode

Le bytecode doit être exécuté par une Machine Virtuelle Java.

Cette JVM n'existe pas. Elle est simulée par un programme qui :

1. lit les instructions (en bytecode) du programme .class
2. fait une passe de vérification (type opérande, taille de pile, flot données, variable bien initialisé,...) pour s'assurer qu'il n'y a aucune action dangereuse.
3. fait plusieurs passes d'optimisation du code
4. les traduit dans le langage natif du processeur de l'ordinateur
5. lance leur exécution

Java *versus* C++ : chaîne de production du Java



Coût de la JVM sur les performances.

Les vérifications effectuées sur le bytecode et la compilation du bytecode vers le langage natif du processeur, ralentissent l'exécution des classes Java.

Mais les techniques de compilation à la volée "Just In Time (JIT)" ou "Hotspot" réduisent ce problème : elles permettent de ne traduire qu'une seule fois en code natif les instructions qui sont (souvent pour Hotspot) exécutées.

Java *versus* C++ : en résumé...

Le langage Java est :

- « C-like » : Syntaxe familière aux programmeurs de C
- Orienté objet : Tout est objet, sauf les types primitifs (entiers, flottants, booléens, ...)
- Robuste : Typage fort, pas de pointeurs, etc.
- Code intermédiaire : Le compilateur ne produit que du bytecode indépendant de l'architecture de la machine où a été compilé le code source

Note

Java perd (un peu) en efficacité par rapport à C++// mais gagne (beaucoup) en portabilité.

Outline

Problématique de la programmation

Le schéma simplifié d'un système informatique peut se résumer par la formule :

$$\boxed{\text{Système informatique} = \text{Structures de données} + \text{Traitements}}$$

Problématique de la programmation

Le schéma simplifié d'un système informatique peut se résumer par la formule :

$$\boxed{\text{Système informatique} = \text{Structures de données} + \text{Traitements}}$$

Le cycle de vie d'un système peut être décomposé en deux grandes phases :

- Une phase de **production** qui consiste à réaliser le logiciel.
- Une phase de **maintenance** qui consiste à corriger et à faire évoluer le logiciel.

Problématique de la programmation

Le schéma simplifié d'un système informatique peut se résumer par la formule :

$$\boxed{\text{Système informatique} = \text{Structures de données} + \text{Traitements}}$$

Le cycle de vie d'un système peut être décomposé en deux grandes phases :

- Une phase de **production** qui consiste à réaliser le logiciel.
- Une phase de **maintenance** qui consiste à corriger et à faire évoluer le logiciel.

Lors de la production du système (au sens industriel du terme), le concepteur a deux grandes options :

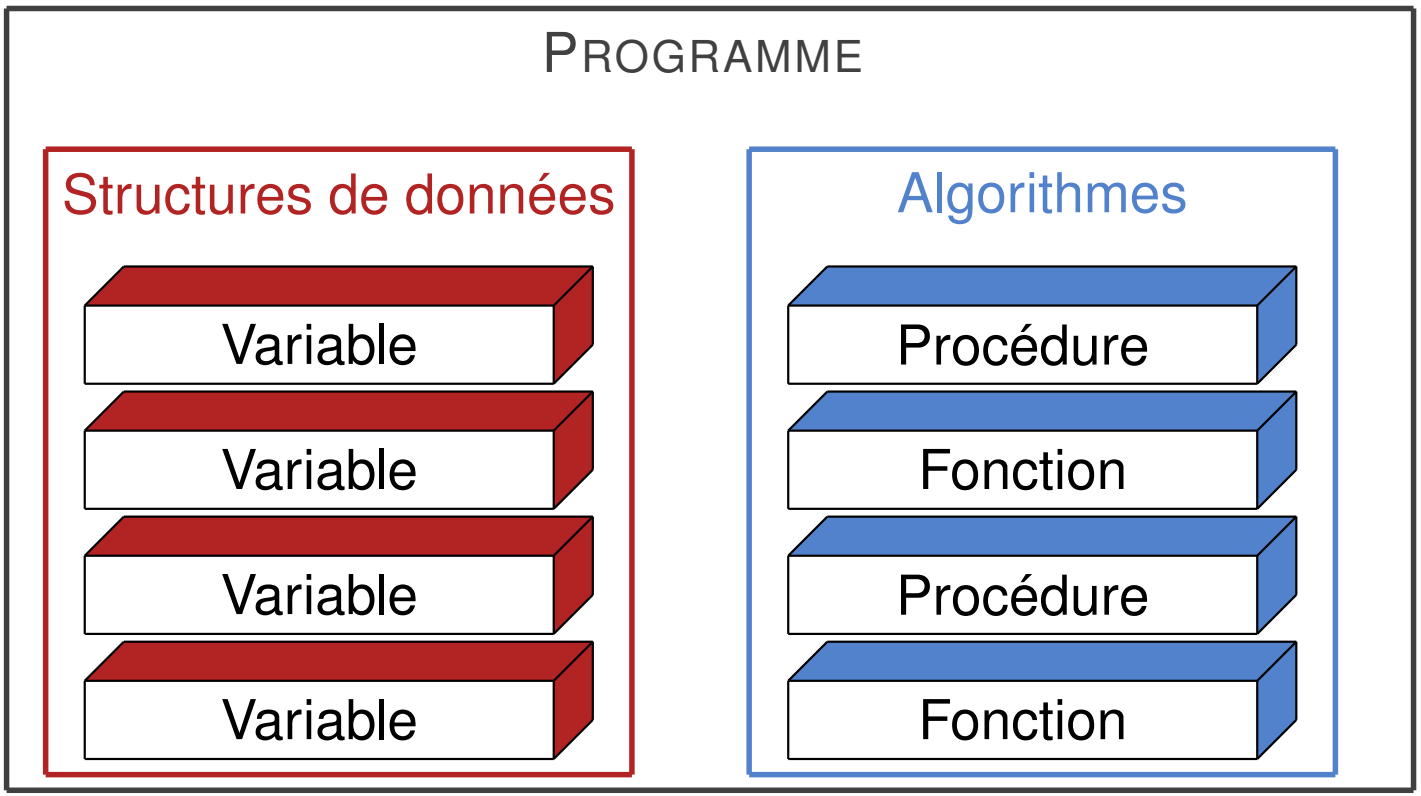
- ☞ soit orienter sa conception sur **les traitements**.
- ☞ soit orienter sa conception sur **les données**.

Outline

Conception par traitements.

Conception par traitements : principe

Principe : On sépare les données des moyens de traitement de ces données.



Conception par traitements : +/-

- Les premiers concepteurs de système informatique ont adopté cette approche : systèmes d'exp., gestionnaires de fenêtres, logiciels de gestion, logiciels de bureautique, logiciels de calcul scientifique, etc.
- De nombreux systèmes informatiques sont encore développés selon cette approche.

☞ Systèmes *ad-hoc*, *i.e.*, adaptés au problème de départ, mais dont la maintenance est difficile.

☞ Les traitements sont généralement beaucoup moins stables que les données : changement de spécification, ajout de nouvelles fonctionnalités, etc.

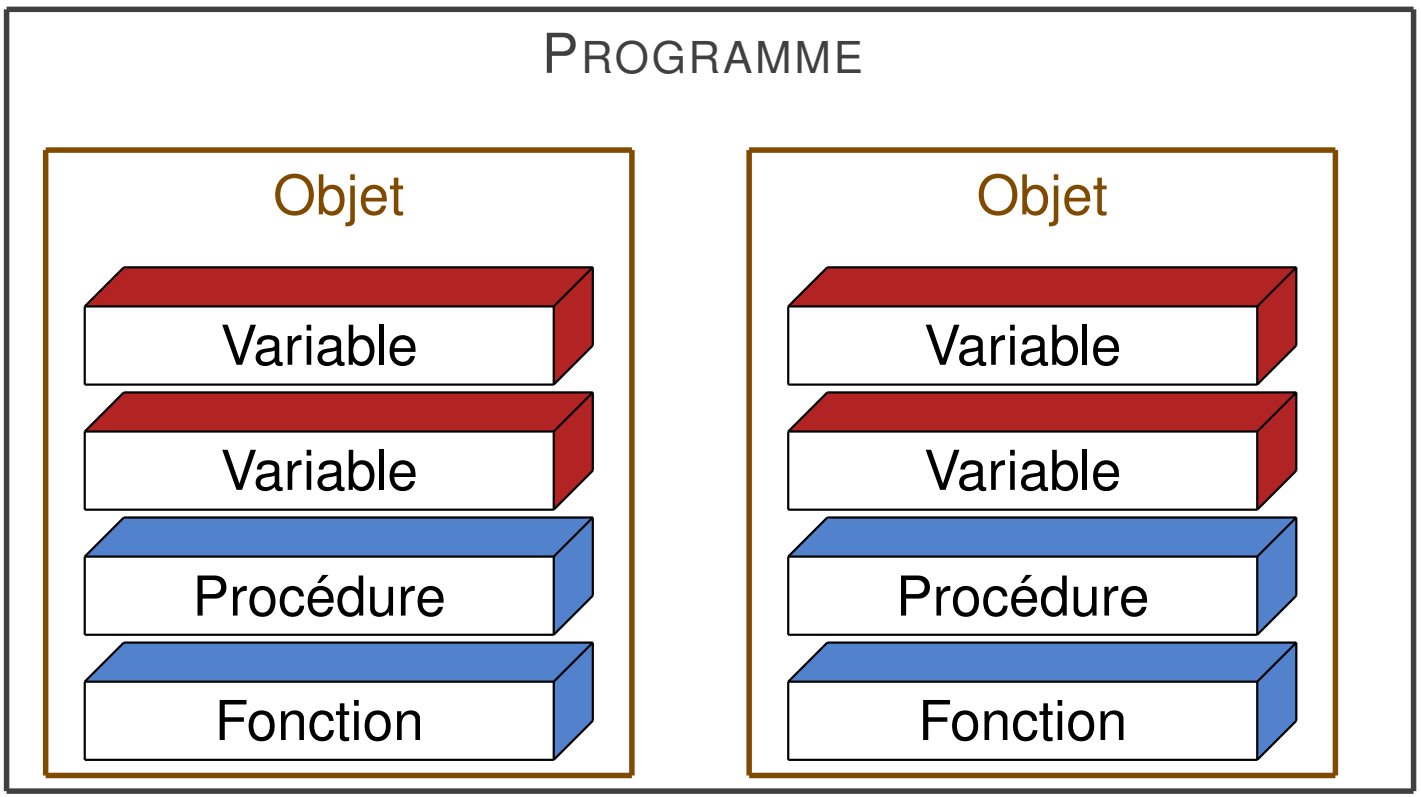
☞ Les structures de données sous-jacentes sont choisies en relation étroite avec les traitements à effectuer.

Outline

Conception par objets.

Conception par objets : principe

Principe : afin d'établir de façon stable et robuste l'architecture d'un système, il semble raisonnable de s'organiser autour des données manipulées.



Conception par objets : points clés

- La construction d'un système va s'axer principalement sur la détermination des données dans un premier temps et la réalisation les traitements (de haut-niveau) agissant sur ces données dans un second temps.
- Cette approche permet de bâtir des systèmes plus simples à maintenir et à faire évoluer.
- On regroupe dans une même entité informatique, appelé objet, les structures de données et les moyens de traitement de ces données.

Le concept d'objet : définition

Définition

*Un **objet** est une entité autonome, qui regroupe un ensemble de propriétés (données) cohérentes et de traitements associés.*

À retenir

Ne commencez pas par vous demander ce que fait l'application mais ce qu'elle manipule.

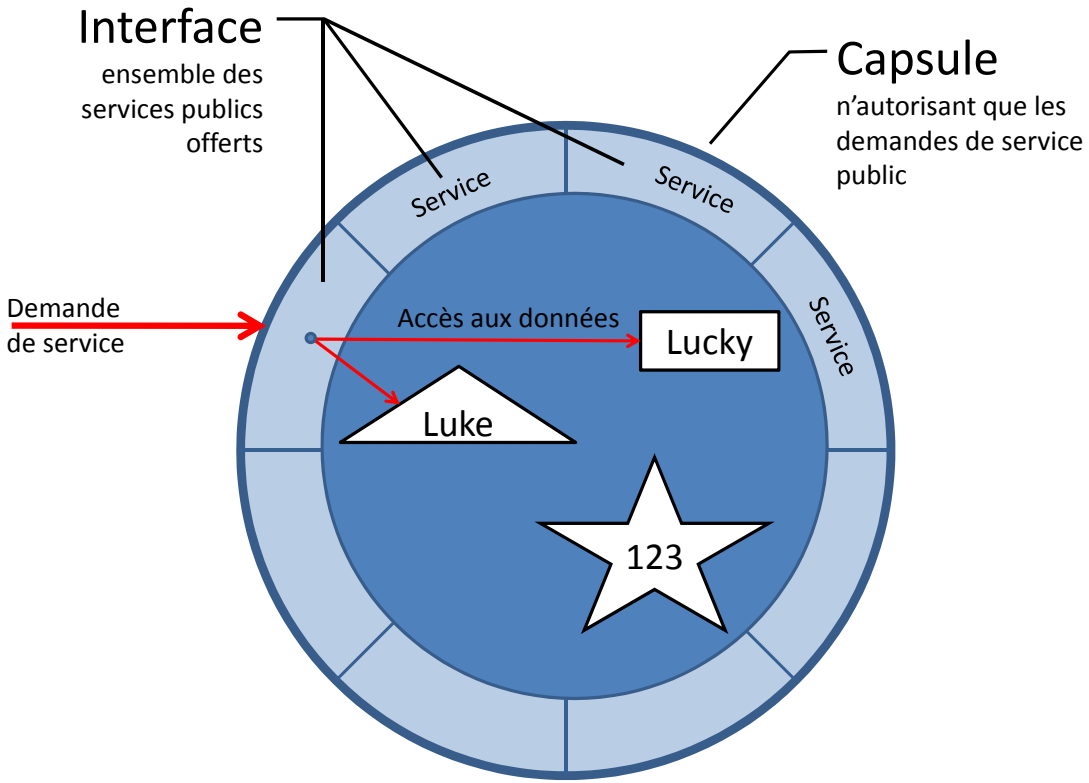
Le concept d'objet : vocabulaire

- Les structures de données définies dans l'objet sont appelés ses attributs (propriétés).
- Les procédures et fonctions définies dans l'objet sont appelés ses méthodes (opérations).
- Les attributs et méthodes d'un objet sont appelés ses membres.
- L'ensemble des valeurs des attributs d'un objet à un instant donné est appelé état interne.

Outline

Le concept d'encapsulation.

Protection de l'information : encapsulation.



Règle

Les données d'un objet (**son état**) peuvent être lues ou modifiées uniquement par les services proposés par l'objet lui-même (**ses méthodes**)

Encapsulation : définition

Définition

Le terme **encapsulation** *désigne le principe consistant à cacher l'information contenue dans un objet et de ne proposer que des méthodes de modification/accès à ces propriétés (attributs).*

Encapsulation : définition

Définition

Le terme **encapsulation** *désigne le principe consistant à cacher l'information contenue dans un objet et de ne proposer que des méthodes de modification/accès à ces propriétés (attributs).*

- L'objet est vu de l'extérieur comme une **boîte noire** ayant certaines propriétés et ayant un comportement spécifié.
- La manière dont le comportement a été implémenté est cachée aux utilisateurs de l'objet.

Intérêt

Protéger la structure interne de l'objet contre toute manipulation non contrôlée, produisant une incohérence.

Encapsulation : en pratique

L'encapsulation nécessite la spécification de **parties publics et privées** de l'objet.

éléments publics : correspond à la partie visible de l'objet depuis l'extérieur. c'est un ensemble de méthodes utilisables par d'autres objets (environnement).

éléments privées : correspond à la partie non visible de l'objet. Il est constitué des éléments de l'objet visibles uniquement de l'intérieur de l'objet et de la définition des méthodes.

Outline

Le concept d'objet : notion de classe

Pour être véritablement intéressante, la notion d'objet doit permettre un certain **degré d'abstraction** $\Rightarrow$ notion de classe.

Le concept d'objet : notion de classe

Pour être véritablement intéressante, la notion d'objet doit permettre un certain **degré d'abstraction** $\Rightarrow$ notion de classe.

Définition

*On appelle **classe** la structure d'un objet, i.e., la déclaration de l'ensemble des membres qui composeront un objet.*

Le concept d'objet : notion de classe

Pour être véritablement intéressante, la notion d'objet doit permettre un certain **degré d'abstraction** $\Rightarrow$ notion de classe.

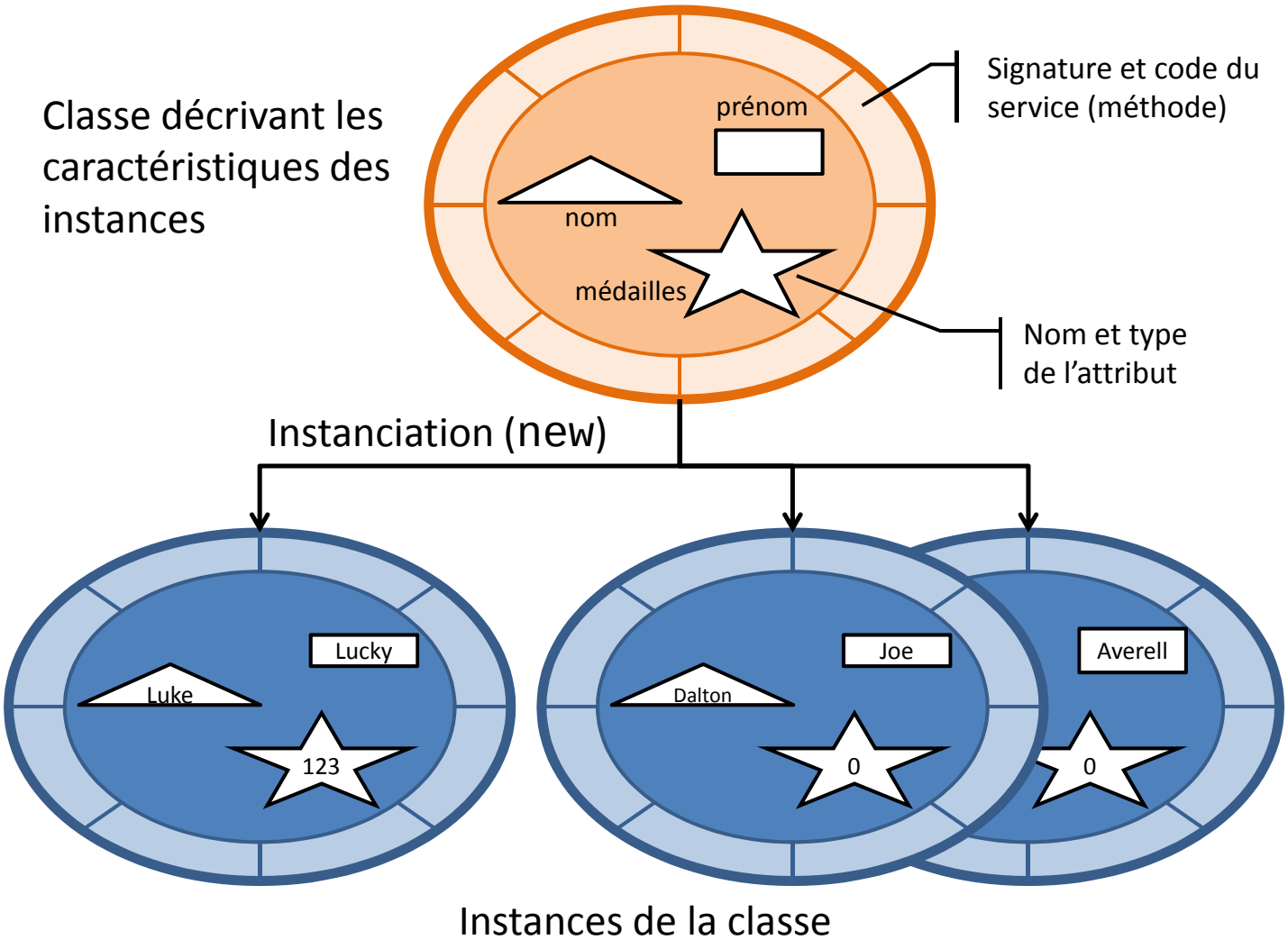
Définition

*On appelle **classe** la structure d'un objet, i.e., la déclaration de l'ensemble des membres qui composeront un objet.*

Définition

*La classe peut être vue comme un moule pour la création des objets, qu'on appelle alors des **instances de la classe**.*

Classe = Modèle d'objets



Différences entre classe et objet.

Il est important de saisir les différences entre les notions de classe et instance de la classe :

classe = attributs + méthodes + mécanismes d'instanciation + mécanismes de destruction

Différences entre classe et objet.

Il est important de saisir les différences entre les notions de classe et instance de la classe :

classe = attributs + méthodes + mécanismes d'instanciation + mécanismes de destruction

instance de la classe = valeurs des attributs + accès aux méthodes

Différences entre classe et objet.

Il est important de saisir les différences entre les notions de classe et instance de la classe :

classe = attributs + méthodes + mécanismes d'instanciation + mécanismes de destruction

instance de la classe = valeurs des attributs + accès aux méthodes

L'instanciation est le mécanisme qui permet de créer des instances dont les traits sont décrits par la classe.

La destruction est le mécanisme qui permet de détruire une instance de la classe.

L'ensemble des instances d'une classe constitue l'extension de la classe .

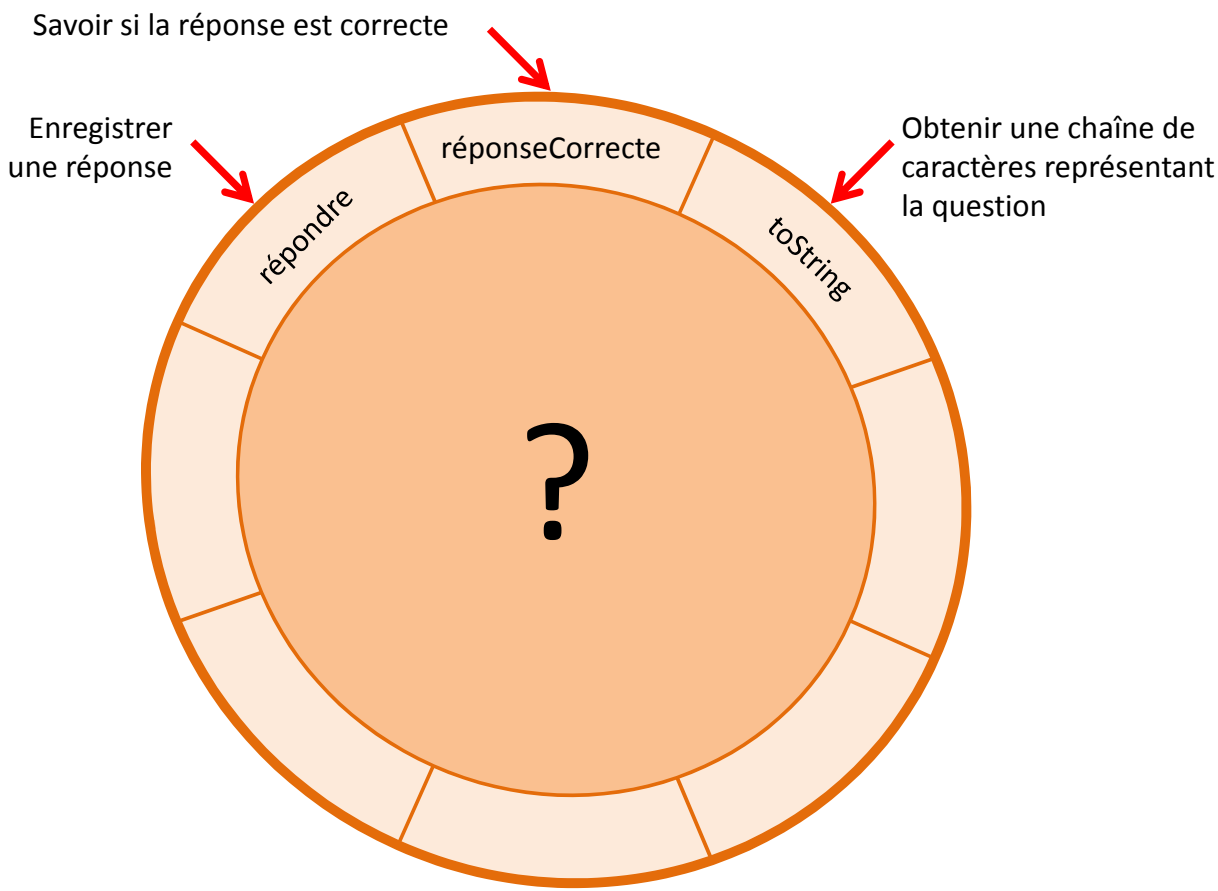
Exemple : Un QCM.

On veut produire une application permettant à des étudiants de répondre à des QCM. Comme un QCM est formé d'une suite de questions, on s'intéresse à définir la classe `Question`. Elle doit permettre de

1. Poser une question (l'affichage ou, mieux, la production d'une chaîne de caractères représentant la question sera employé à cet effet)
2. Enregistrer la réponse fournie par un étudiant
3. Déterminer si la réponse fournie est la bonne

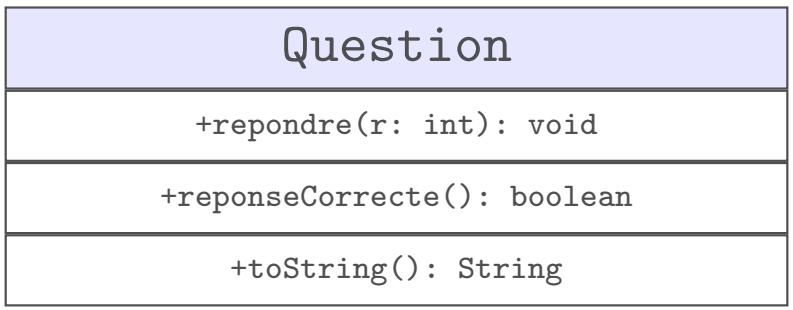
Conception de la classe `Question`

Les services offerts par la classe :



Aucune hypothèse sur la structure interne

Diagramme UML correspondant



Du point de vue de l'utilisateur de la classe, seuls les services publics sont pertinents

Classe : déclaration.

En Java, pour déclarer une classe on utilise le mot-clé **class** suivi du nom de la classe.

```
public class Point {
    ...
}
```

Règles

1. La **première lettre** du nom d'une classe doit toujours être une lettre majuscule (ex : `Chat`).
2. Mélange de minuscule, majuscule avec la première lettre de chaque mot en majuscule (ex : `ChatGris`).
3. Une classe se trouve dans un **fichier portant son nom** suivi l'extention `.java` (ex : `ChatGris.java`)

Classe : visibilité d'un membre

En Java, l'encapsulation est assurée par un ensemble de modificateurs d'accès permettant de préciser la visibilité des membres de la classe :

Définition

Un membre dont la déclaration est précédée par le mot clé ***public*** *est visible depuis toutes instances de toutes classes.*

Définition

Un membre dont la déclaration est précédée par le mot clé ***private*** *n'est visible que dans les instances de la classe.*

Remarque(s)

Il existe d'autres types de visibilité pour un membre que nous n'évoquerons pas dans l'immédiat.

Définition Java du squelette de la classe Question

Le mot clé **public** permet d'indiquer les services qui sont accessibles à l'utilisateur.

```
public class Question {
   ...
   public void repondre(int reponse) {
      ...
   }

   public boolean reponseCorrecte() {
      ...
   }

   public String toString() {
      ...
   }
...
}
```

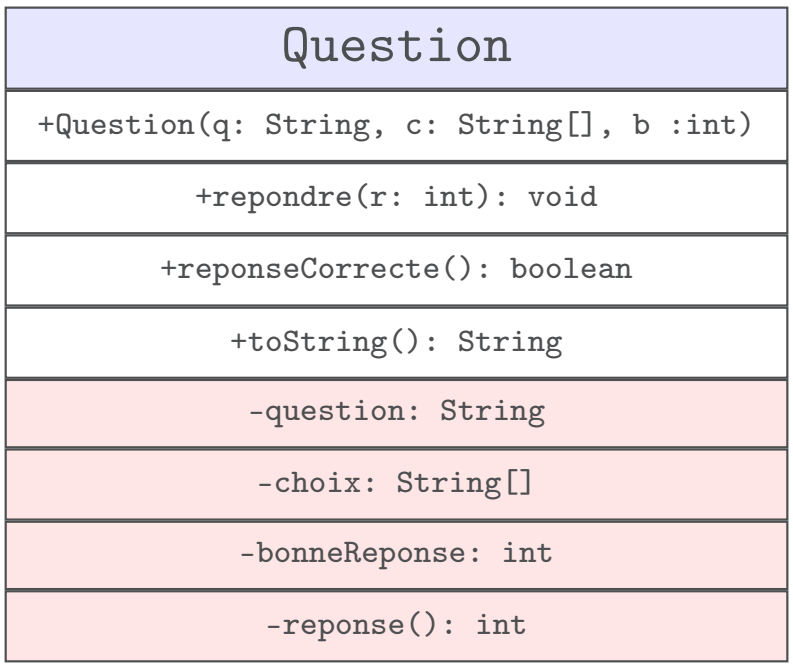
Définition Java du squelette de la classe Question

Le mot clé **private** assure l'encapsulation

```
public class Question {
  // intitulé de la question
  private String question;
  // tableau des choix possibles
  private String[] choix;
  // numéro de la réponse de l'étudiant
  private int reponse;
  // numéro de la bonne réponse
  private int bonneReponse;

  public void repondre(int reponse) {
    ...
  }
  ...
}
```

Diagramme UML complet



Accès aux membres d'une classe.

Définition

L'ensemble des méthodes d'un objet accessibles de l'extérieur (depuis un autre objet) est appelé ***interface****. Elle caractérise le comportement de l'objet.*

Définition

*L'accès à un membre d'une classe se fait au moyen de l'****opérateur « . »****.*

Définition

L'invocation d'une méthode d'interface est appelé ***appel de méthode****. Il peut être vu comme un envoi de message entre objet.*

Accès aux membres.

```
public class Point {
  public double x,y ;
}

public class Rectangle {
  public double longeur,largeur ;
  public Point coin ;
  public Point calculerCentre();
}

public class TestFigure {
  public static void main(String[] args) {
    Rectangle rect ;
    Point coinDuRect = rect.coin ;
    double xDuCoinDuRect = coinDuRect.x ;
    Point centreDuRect = rect.calculerCentre() ;
    double yDuCentreDuRect = centreDuRect.y ;
  }
}
```

Accès récursifs aux membres.

```
public class Point {
  public double x,y ;
}

public class Rectangle {
  public double longeur,largeur ;
  public Point coin ;
  public Point calculerCentre();
}

public class TestFigure {
  public static void main(String[] args) {
    Rectangle rect ;
    double xDuCoinDuRect = rect.coin.x ;
    double yDuCentreDuRect = rect.calculerCentre().y ;
  }
}
```

Classe : membres d'instance et de classes

Quelque soit le niveau de visibilité, on distingue deux types de membres :

Définition

*Un **membre de classe** est un membre **commun à toutes** les instances de la classe et existe dès que la classe est définie en dehors et indépendamment de toute instanciation. Les membres de classe sont déclarés à l'aide du mot-clé **static**.*

Définition

*Un membre qui n'est pas de classe, i.e. dont la déclaration n'est pas précédée du mot-clé static est dit **membre d'instance**. Chaque instance d'une classe possède **son propre exemplaire** d'un attribut d'instance de la classe.*

Variable de classe

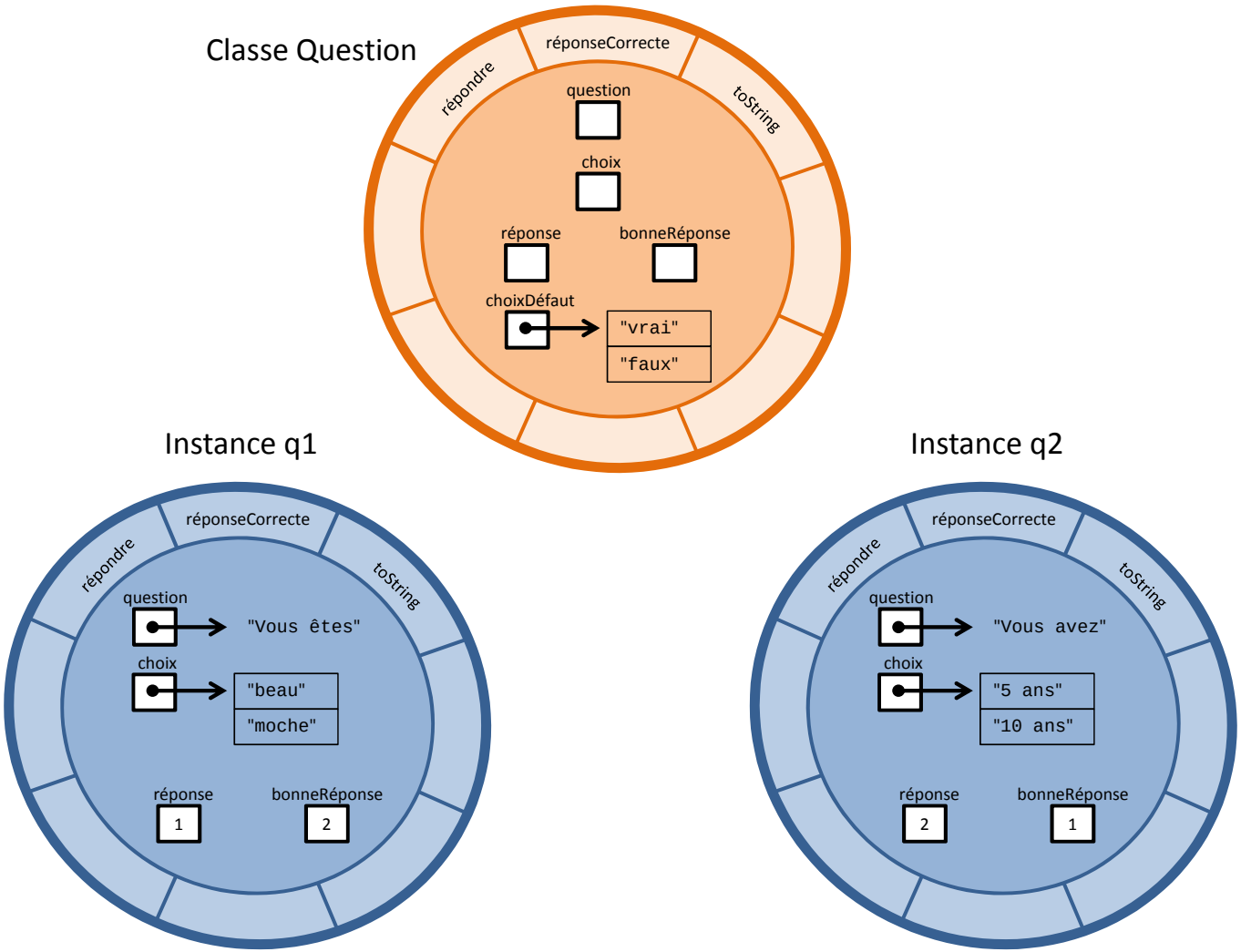
On veut introduire des choix par défaut

```
public class Question {
   ...
   private String[] choixDéfaut = {"vrai", "faux"};
   ...
}
```

Problème : chaque instance dispose de son propre tableau `choixDéfaut` alors qu'il devrait être commun à tous les objets

```
public class Question {
   ...
   private static String[] choixDéfaut
                           = {"vrai", "faux"};
   ...
}
```

Variable de classe



Classe : l'accès aux membres de classe

Les membres de classes d'une classe donnée étant communs à toutes les instances de la classe, l'accès à un membre de classe se fait en appliquant l'opérateur « . » sur le nom de la classe.

```
System.out.println(Question.choixDefaut[1]);
```

L'accès aux membres de classe peut aussi se faire avec une instance de la classe suivie de l'opérateur « . ». Mais ceci est peu lisible et à n'utiliser que pour le polymorphisme (voir cours suivant).

```
Question q1 = new Question();
System.out.println(q1.choixDefaut[1]);
```

Classe java type

Une classe java types contient trois grands types de membres :

```
public class Point {

}
```

Classe java type

Une classe java types contient trois grands types de membres :

```
public class Point {
  // (1) Attributs
  private double x,y;
```

Classe java type

Une classe java types contient trois grands types de membres :

```
public class Point {
  // (1) Attributs
  private double x,y;
  // (2) Constructeurs
  public Point(double x, double y) {...}
```

Classe java type

Une classe java types contient trois grands types de membres :

```
public class Point {
  // (1) Attributs
  private double x,y;
  // (2) Constructeurs
  public Point(double x, double y) {...}
  // (3) Méthodes
  public double distanceAvec(Point p2) {...}
}
```

Classe java type

Une classe java types contient trois grands types de membres :

```
public class Point {
  // (1) Attributs
  private double x,y;
  // (2) Constructeurs
  public Point(double x, double y) {...}
  // (3) Méthodes
  public double distanceAvec(Point p2) {...}
}
```

Remarque(s)

Les constructeurs sont des méthodes particulières, ils seront donc introduits après celles-ci dans ce cours. Cependant, vous devez **toujours déclarer les constructeurs après les attributs et avant les autres méthodes** (voir ci-dessus).

Outline

Déclaration d'un attributs

En Java, toutes les variables doivent être déclarées avant d'être utilisées (les attributs comme les variable locales des méthodes).

La déclaration des attributs se fait de préférence en début de classe et leurs noms commencent par des minuscules.

On indique au compilateur :

1. Un ensemble de modificateurs (*facultatif*).
2. Le type de la variable.
3. Le nom de la variable.

```
private double z;
public String str;
```

Les types en Java

Le type d'une variable en Java peut être :

- **Types dits primitifs : int**, **double**, **boolean**, etc.
- **nom d'une classe :** par exemple, les chaînes de caractères sont des instances de la classe **String**.

Remarque(s)

Nous laissons de côté pour l'instant le cas des tableaux sur lesquels nous reviendrons plus tard.

Les types primitifs

Type	Taille (en bits)	Exemple
byte	8	1
short	16	345
int	32	-2
long	64	2L
float	32	3.14f, 2.5e+5
double	64	0.2d, 1.567e-5
boolean	1	true ou false
char	16	'a'

Attention

Un attribut de type primitif n'est pas un objet !

Affecter une valeur à un attribut

L'affectation d'une valeur à une variable est effectuée par l'instruction

```
variable = expression;
```

L'affectation se fait en deux temps :

1. l'expression est calculée ;
2. la valeur calculée est affectée à la variable.

Exemple

```
x = 36.0d;
y = x + 1;
str = "Linus";
```

Initialisation d'un attribut

En Java, une variable doit être initialisée (recevoir une valeur) avant d'être utilisée dans une expression.

S'ils ne sont pas initialisés expressément par le programmeur, les attributs seront automatiquement initialisés par le compilateur. Ils reçoivent alors une valeur par défaut de leur type : 0 pour les **int**, $0.0d$ pour les **double**, **false** pour les **boolean**, **null** pour les `String`, etc.

Remarque(s)

On peut initialiser un attribut lors de sa déclaration.

```
private int x = -1;
public String str = "Licence␣MIAGE";
```

Les constantes

On peut bloquer la modification de la valeur d'un attribut (en dehors de l'instanciation) à l'aide du mot-clé **final**.

```
final private int x;
```

Remarque(s)

On utilise souvent un attribut de classe déclaré **final** pour définir une constante :

```
final static public double PI = 3.14d;
```

Les types en Java : déclarer les attributs

```java
public class Logement {
  // les attributs
  final public double surface;
  public double prix;
  public String proprietaire;
  private boolean vendu;

}
```

Outline

Les méthodes en Java : syntaxe et définition

La syntaxe pour définir le corps d'une méthode est identique à celle utilisée en `C` pour définir une fonction. Notamment, les noms et la syntaxe

- des instructions (conditionnelles, itératives, etc.) et,
- des opérateurs (arithmétiques, de comparaison, logiques, etc.)

sont les mêmes qu'en `C`.

Outline

Le mot clé this.

Définition du mot clé this.

Définition

Dans le corps d'une méthode, le mot clé ***this*** *désigne l'instance sur laquelle est invoquée la méthode. Ce mot clé est utilisé dans 3 circonstances :*

1. *pour accéder aux attributs de l'objets*
2. *pour comparer la référence de l'objet invoquant la méthode à une autre référence*
3. *pour passer la reference de l'objets invoquant la méthode en paramètre d'une autre méthode*

Utilisation du this : accès à un attribut.

Dans le corps d'une méthode, **this** permet d'accéder aux attributs de l'objet lorsque ceux-ci sont masqués par un paramètre ou par une variable locale.

```
public class Question {
int reponse ;
  ...
  public void repondre(int reponse) {
    this.reponse = reponse;
  }
  public boolean reponseCorrecte() {
    return reponse == bonneReponse;
  }
  ...
}
```

Utilisation du this : comparer la référence.

Dans le corps d'une méthode, **this** permet de comparer la référence de l'objet sur lequel est invoqué la méthode à une autre référence.

```
public class Animal {
  Estomac estomac ;
  ...
  public void manger(Animal victime) {
    // On ne peut pas se manger complètement soi-même
    if ( this != victime ) {
      estomac.addNouriture(victime);
    }
  }
}
```

Utilisation du this : référence en paramètre

Dans le corps d'une méthode, **this** permet de passer la référence de l'objet sur lequel est invoqué à une autre méthode.

```
public class Entreprise {
  public int calculerSalaire(Individu x) {...}
}

public class Individu {
  int compte ;
  Entreprise entreprise ;
  ...
  public void demissionner () {
    compte += entreprise.calculerSalaire( this );
    entreprise = null;
  }
}
```

Outline

Les accesseurs.

Respect de l'encapsulation

Règle

Les attributs doivent :

- **toujours être déclarés privés** :

```
private int longeur ;
```

- être accédé en lecture par un **accesseurs** en lecture :

```
public int getLongeur() ;
```

- être accédé en écriture par un **accesseurs** en écriture :

```
public void setLongeur(int l) ;
```

Pourquoi cacher les attributs ?

```
public class Point {
  public int x,y;
}

public class Figure {
  public Point c;
  public void translater(int x,int y) {
    this.c.x += x;
    this.c.y += y;
  }
}
```

Pourquoi cacher les attributs ?

```
public class Point {
  public int[] tuple;
}

public class Figure {
  public Point c;
  public void translater(int x,int y) {
    this.c.x += x;
    this.c.y += y;
  }
}
```

Pourquoi cacher les attributs ?

```
public class Point {
  public int[] tuple;
}

public class Figure {
  public Point c;
  public void translater(int x,int y) {
    this.c.x += x;
    this.c.y += y;
  }
}
```

Pourquoi cacher les attributs ?

```
public class Point {
  private int x,y;
  public int getX() {...}
  public void setX(int x) {...}
  public int getY() {...}
  public void setY(int y) {...}
}

public class Figure {
  public Point c;
  public void translater(int x,int y) {
    this.c.setX(this.c.getX()+x);
    this.c.setY(this.c.getY()+y);
  }
}
```

Pourquoi cacher les attributs ?

```
public class Point {
  private int[] tuple;
  public int getX() {...}
  public void setX(int x) {...}
  public int getY() {...}
  public void setY(int y) {...}
}

public class Figure {
  public Point c;
  public void translater(int x,int y) {
    this.c.setX(this.c.getX()+x);
    this.c.setY(this.c.getY()+y);
  }
}
```

Rappel

Une classe implémente un concept associant

- des données manipulées exclusivement par les instances de la classe
  - les **attributs** : mode d'accès **private** pour garantir la protection de l'information
- les traitements que l'on souhaite effectuer
  - les **méthodes publiques** : mode d'accès **public** visibles depuis l'extérieur de la classe $\Longrightarrow$ *vision client*
  - les **méthodes privées** : mode d'accès **private** visibles uniquement depuis l'intérieur de la classe (autres méthodes) $\Longrightarrow$ *vision fournisseur*

Accesseurs : bonne et mauvaise pratique

- Mauvaise pratique La création des méthodes publiques ou privées doit se faire en fonction des besoins et non à-priori. On ne définit pas des accesseurs ou (pire) des accesseurs publics systématiquement
- Bonne pratique Si on a besoin d'un accesseur en lecture et/ou en écriture, alors on le/les crée :
  - si le besoin est à l'intérieur de la classe (fournisseur) $\Longrightarrow$ accesseur privé
  - si le besoin est à l'extérieur de la classe (client) $\Longrightarrow$ accesseur public mais attention, danger ! Il faut assurer l'indépendance de la représentation

Outline

Les méthodes de classe.

Les méthodes static en java.

Définition

Comme pour les attributs, une ***méthode de classe*** *est une méthode dont la déclaration est précédée du modificateur* ***static****. Elle ne s'exécute pas relativement à une instance de classe et ne peut donc pas utiliser "directement" :*

- *des attributs instances*
- *des methodes instances*
- *le mot cle* ***this***

Les **méthodes de classe** peuvent être utilsées pour :

1. Manipuler les attributs de classe (lecture ou affectation) ;
2. Définir une méthode générique sur la classe.

Méthode de classe (1) : Utilisation des attributs de classe.

Une méthode de classe peut être utilisée pour manipuler des attributs de classe.

```
public class Individu{
  static int ageRetraite ;
  static void modifierAgeRetraite(int ageRetraite) {
    Individu.ageRetraite = ageRetraite ;
  }
}
```

Attention

Ici, on ne peut pas utiliser le mot clé **this** pour accéder à l'attribut de classe `ageRetraitre`.

Méthode de classe (2) : Les méthodes générique.

Une méthode de classe n'utilise pas toujours des attributs de classe. Elle peut être introduite pour effectuer des actions génériques sur des ensembles d'instances de la classe.

```
public class Point {
  public float distance (Point p) {...}
  static float distanceEntre (Point p1, Point p2) {...}
}

public class TestPoint {
  public void main (String[] args) {
    Point a,b ;
    ...
    dist = a.distance(b);
    dist = Point.distanceEntre(a,b);
  }
}
```

Classes n'ayant que des méthodes static.

Remarque(s)

Une classe peut ne contenir que des membres **static**. Dans ce cas elle ne sert pas a creer des objets ! ! !

Exemple 1 - La classe `Math` est une **classe boîte à outils** ne possédant que des constantes (`E` et `PI`) et méthodes de classes.

```
int rayon = 30 ;
double surface =  Math.PI * Math.pow(rayon,2) ;
```

Exemple 2 - La classe `System` qui représente la VM ne possédant que des attributs et méthodes de classes : il n'y a pas de raison d'en avoir plusieurs instances.

```
// Permet d'afficher des messages dans la console
System.out.println("Hello, World !");
// Permet d'arrêter brutalement un programme.
System.exit(0);
```

Outline

La surcharge.

La signature d'une méthode Java

Définition

*On appelle **signature** d'une méthode, l'ensemble composé de :*

- *son nom*
- *des types de ses arguments*

Elle permet d'identifier de manière unique une méthode au sein d'une classe.

ATTENTION

En java, ne font pas partie de la signature d'une méthode :

1. le nom de ses arguments (comme en C)
2. son type de retour de la méthode (`void` ou non)

La signature d'une méthode Java

Définition

*On appelle **surcharge** (en anglais « overloading ») d'une méthode, la possibilité de définir des comportements différents pour la même méthode selon les arguments passés en paramètres. Dans ce cas, deux méthodes d'une même classe peuvent porter le même "nom" mais n'ont pas la même signature.*

```
public class Rectangle {
  public void redimensionner (int facteur) {
    largeur = largeur * facteur;
    longueur = longueur * facteur;
  }
  public void redimensionner (float a, float b) {
    largeur = largeur + a;
    longueur = longueur +b;
  }
}
```

La signature d'une méthode Java

```
public class Rectangle {
  public void redimensionner (int facteur) {...}
  public void redimensionner (float a, float b) {...}
}

pulic class TestRectangle {
  public static void main (String[] args) {
    Rectangle r1 ;
    ...
    r1.redimensionner(2);
    r1.redimensionner(2, 3.1);
  }
}
```

La signature d'une méthode Java

ATTENTION

Il ne faut pas confondre la **surcharge** (en anglais « *overloading* ») et la **redéfinition** (en anglais « *overriding* ») qui sera étudiée dans le cours sur le polymorphisme et qui correspond à la possibilité de spécifier le comportement d'une méthode à l'exécution selon le type d'objets l'invoquant.

Outline

Outline

Le mécanisme d'instanciation.

Instanciation de classe.

Les instances d'une classe sont créées (construits) par une méthode particulière de la classe appelée **constructeur**.

Constructeur en Java : définition (suite)

Définition

*Un **constructeur** est une méthode qui n'a pas de type de retour, qui porte le même nom que la classe. C'est cette méthode qui sera automatiquement appelée à l'instanciation de la classe, i.e.,au moment de la création d'un objet. Les constructeurs permettent, entre autres, d'initialiser les attributs de la classe.*

```
public class Point {
  private int x;
  private int y;
  public Point (int x, int y) {
    this.x = x ;
    this.y = y;
  }
}
```

Constructeur en Java : invocation

Définition

Pour créer une instance d'une classe, on utilise l'opérateur ***new*** *avec l'un des constructeurs de la classe :* `p = new Point(2,3);`
Cette invocation :

1. *alloue l'espace mémoire nécessaire pour stocker les propriétés*
2. *crée une référence sur cet espace mémoire*
3. *exécute le code du constructeur pour initialiser les données*
4. *retourne la référence ainsi créée.*

Remarque(s)

Tout est dynamique :
les instances d'une classe sont instanciées dynamiquement.

Outline

Constructeur par défaut.

Constructeur par défaut : définition.

Règle

Toute classe posséde au moins un constructeur.

Définition

En abscence de déclaration explicite d'un constructeur, un ***constructeur par défaut*** *sera automatiquement ajouté par le compilateur Java.*

Pour classe Toto, ce constructeur par défaut sera :

```
[public] Toto() {}
```

Constructeur par défaut : ce qui est fait.

Remarque(s)

La notion de constructeur par défaut est en partie lié à l'héritage ; elle sera donc complétée plus tard lorsque nous introduirons cette notion.

Dans un premier temps, on retiendra qu'un constructeur par défaut :

1. initialise les attributs numérique de types primitif à 0 ;
2. initialise les attributs de (**boolean**) à **false** ;
3. initialise les attributs de types Objet à **null**.

Constructeur par défaut : exemple.

```
public class Point {
  private int x;
  private int y;
  public afficher () {
    System.out.println("x="+x+" et y="+y);
  }
}

public class TestPoint () {
  public static void main(String[] args) {
    Point p = new Point();
    p.afficher();
  }
}
```

```
java TestPoint
  x=0 et y=0
```

Constructeur par défaut : la limite.

Attention

Le constructeur par défaut n'est ajouté par Java si et seulement s'il n'y a pas de constructeur explicite.
Une classe ne possède donc pas toujours un construceur sans paramètre.
Si l'on en veut un en plus d'un constructeur à paramètre, il faut le déclarer explicitement.

Constructeur par défaut : la limite.

```
public class Point {
  private int x;
  private int y;
  public Point (int x, int y) {
    this.x = x ;
    this.y = y;
  }
}

public class TestPoint () {
  public static void main(String[] args) {
     Point p = new Point() ;
  }
}
```

Outline

Plusieurs constructeurs.

Surcharge de constructeur.

Si une classe a plusieurs constructeurs, il s'agit de **surcharge** :

- même nom (celui de la classe)
- types des paramètres différents.

Exemple de constructeurs.

```
public class Question {
  ...
  public Question(String question, String[] choix,
                  int bonneReponse) {
    this.question = question;
    this.choix = choix;
    this.bonneReponse = bonneReponse;
    this.reponse = 0;
  }
  public Question(String question, String[] choix) {
    this.question = question;
    this.choix = choix;
    this.bonneReponse = 0;
    this.reponse = 0;
  }
  ...
}
```

Exemple de constructeurs imbriqués.

On peut utiliser un constructeur pour définir un autre constructeur, au moyen du mot-clé **this**, mais cette invocation d'un autre constructeur doit être la première instruction.

```
public class Question {
  ...
  public Question(String question, String[] choix,
                                    int bonneReponse) {
    this.question = question;
    this.choix = choix;
    this.bonneReponse = bonneReponse;
    this.reponse = 0;
  }
  public Question(String question, String[] choix) {
    this(question,choix,0);
  }
  ...
}
```

Exemple d'instanciation

```
public class AppliQuestion {
  public static void main(String[] args) {
    String[] rep = { "Blanc", "Noir" };
    Question q =
          new Question("Le␣cheval␣blanc␣est", rep, 1);
    System.out.println(q.toString());
    ...
}
```

- q est une référence vers une instance de `Question`
- La valeur particulière **null** peut lui être affectée

Classe : définition.

Définir une classe en Java, c'est définir ses membres à savoir :
(1) ses attributs, (2) le(s) constructeur(s), (3) et ses méthodes.

- Il n'est pas nécessaire de spécifier de mécanisme de destruction,
- transparente pour l'utilisateur,
- prise en charge par un gestionnaire appelé **garbage collector** chargé de détecter les instances à détruire.

Conséquences

- gain de fiabilité (pas de désallocation erronée).
- a un coût (perte en rapidité).

Outline

Outline

Coder un executable.

La méthode `main` de Java : concept

Il ne suffit pas de définir les attributs, constructeurs et méthodes des différentes classes, il faut pouvoir exécuter un programme, les concepteurs de Java ont choisi pour cela de particulariser une méthode : **la méthode `main`**.

La méthode main est une méthode de classe publique, qui contient le « programme principal » à exécuter et qui a pour signature :

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

Attention

La méthode *main* ne peut pas retourner d'entier comme en C.

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

Un premier programme Java

```java
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello world !!!");
    }
}
```

```
Hello world !!!
```

Un deuxième programme Java

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        int age;
        Scanner sc = new Scanner(System.in);
        System.out.print("Saisissez votre âge : ");
        age = sc.nextInt();
        if (age > 20)
            System.out.println("vous êtes agé");
        else if (age > 0)
            System.out.println("vous êtes jeune");
        else
            System.out.println("vous êtes en devenir");
    }
}
```

Un deuxième programme Java

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        int age;
        Scanner sc = new Scanner(System.in);
        System.out.print("Saisissez votre âge : ");
        age = sc.nextInt();
        if (age > 20)
            System.out.println("vous êtes agé");
        else if (age > 0)
            System.out.println("vous êtes jeune");
        else
            System.out.println("vous êtes en devenir");
    }
}
```

```
Saisissez votre âge : 16
vous êtes jeune
```

Un troisième programme Java

```java
public class Main {
  public static void f(int[] tab) {
    for (int i = 0; i < tab.length; ++i)
      for (int j = 0; j < tab.length - i - 1; ++j)
        if (tab[j] > tab[j + 1]) {
          int tmp = tab[j];
          tab[j] = tab[j + 1];
          tab[j + 1] = tmp;
        }
  }

  public static void main(String[] args) {
    int[] t = { 2, 7, 1, 5 };
    Main.f(t);
    for (int i : t) System.out.print(i + " ");
  }
}
```

```
1 2 5 7
```

Outline

Compilation.

Compilation et exécution (1)

Étape 1 : on crée un fichier texte dont le nom est le nom de la classe auquel on rajoute l'extension `.java`.

On écrit la définition de la classe `Logement` dans un fichier texte nommé `HelloWord.java`.

Important

Un fichier ne peut contenir qu'une seule classe publique.

Compilation et exécution (2)

Étape 2 : on compile le programme Java à l'aide de la commande javac.

HelloWorld.java

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

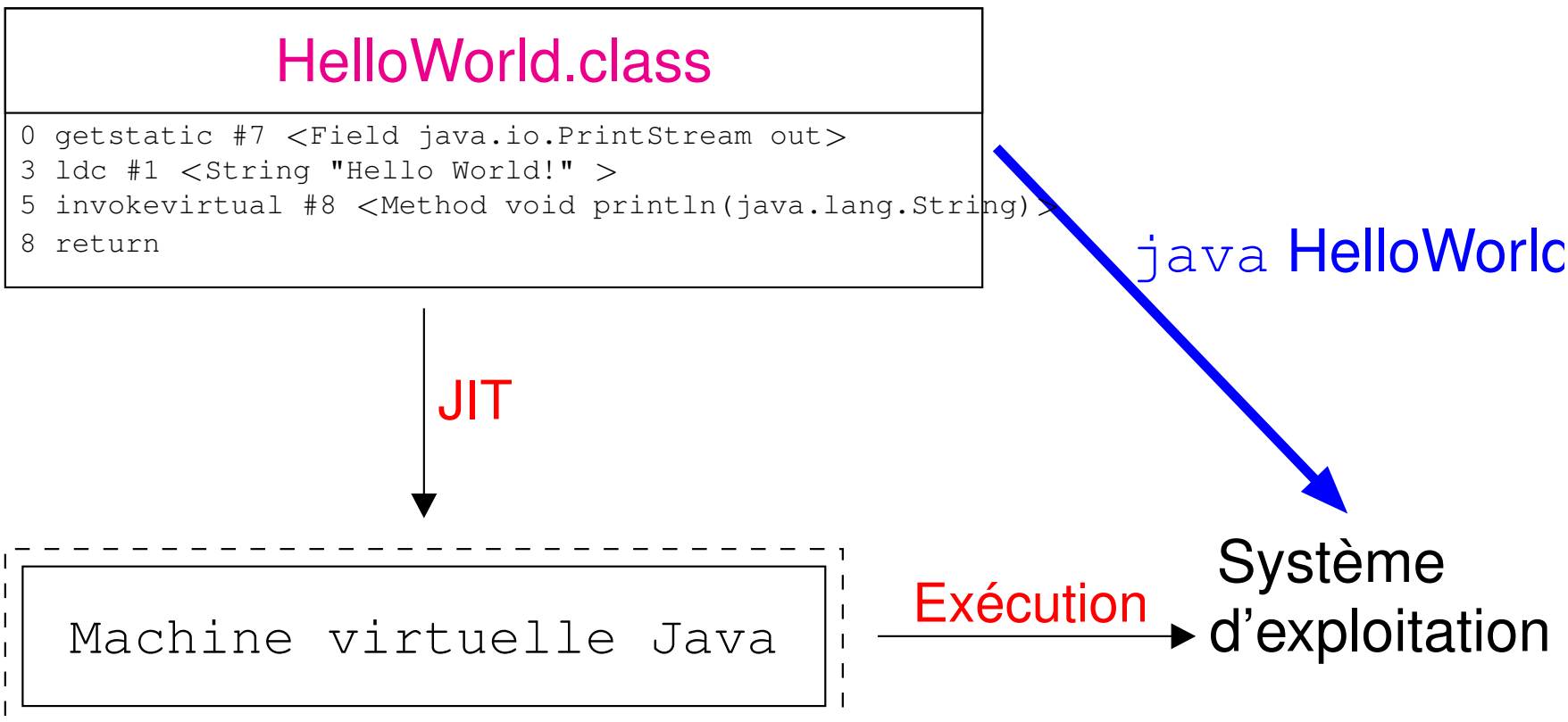
javac HelloWorld.java

HelloWorld.class

```
0 getstatic #7 <Field java.io.PrintStream out>
3 ldc #1 <String "Hello World!" >
5 invokevirtual #8 <Method void println(java.lang.String)>
8 return
```

Compilation et exécution (3)

Étape 3 : le JIT de la machine virtuelle compile à la volée le bytecode produit (c'est-à-dire on exécute la méthode `main`) à l'aide la commande `java`.



Outline

Structuration des sources.

Compilation et interprétation (4)

Important **Bien positionner les variables d'environnement**

`PATH` : doit contenir le répertoire du compilateur et de la machine virtuelle.

`CLASSPATH` : indique le chemin ou se trouve les classes (par défaut, le répertoire courant) sinon vous aurez le message d'erreur

```
Exception in thread "main"
java.lang.NoClassDefFoundError: HelloWorld
```

Outline

Outline

La classe `String`

Les chaînes de caractères : le type String (1)

Les chaînes de caractères sont des instances de la classe `String`. L'opérateur de concaténation des chaînes de caractères est l'opérateur +.

Attention

Pour comparer deux chaînes de caractères, on utilise la méthode `equals` (ou `equalsIgnoreCase`) de la classe `String`.

```
String str1 = ...;
String str2 = ...;

if (str1.equals(str2)) {...} else {...}
```

Les chaînes de caractères : le type String (2)

La classe `String` offre de nombreuses autres possibilités :

- `length()` renvoie la longueur de la chaîne de caractères.
- `toUpperCase` et `toLowerCase` permettent, respectivement, de mettre la chaîne de caractères en lettres majuscules et minuscules.
- `int indexOf (char ch)` renvoie l'indice de la première occurence du caractère `ch` dans la chaîne de caractères.
- `String substring(int beginIndex,int endIndex)` qui retourne la sous-chaîne constitué des caractères d'indice `beginIndex` à `endIndex - 1`.
- etc ...

Outline

Les tableaux.

Les tableaux

Important

Un tableau est un objet !

Deux étapes :

1. Déclaration : déterminer le type de ses éléments.
2. Dimensionnement : déterminer la taille du tableau (c'est-à-dire le nombre d'éléments).

Les tableaux unidimensionnels

La déclaration d'un tableau précise simplement le type des éléments du tableau :

```
int [] tableau;
```

La dimension du tableau est précisé lors de son instanciation

```
// crée un tableau pouvant contenir 50 entiers
tableau = new int [50];
```

- La taille d'un tableau ne peut plus être modifiée par la suite.
- Dimension du tableau : `tableau.length`

Les tableaux unidimensionnels

- On accède à l'élément d'indice `i` du tableau en écrivant `tableau[i]`.
- Les indices commencent à 0.
- `Java` vérifie automatiquement l'indice lors de l'accès.

Remarque(s)

On peut aussi donner explicitement la liste des éléments :d'un tableau au moment de son instanciation :

```
int [] tableau = {1,2,3};
String [] mots = ["Licence","MIAGE","POO"};
```

Remarque(s)

Si l'on déclare un tableau dont les éléments sont des références de type `logement` :

```
Logement[] ville;
```

Alors, la ligne de commande

```
ville = new Logement[100];
```

instancie un tableau de 100 références initialisées à **null**. Si l'on veut qu'une case contienne une référence vers une instance de la classe `Logement`, on doit instancier une instance de la classe `Logement` et écrire la référence de l'instance dans la case :

```
ville[50] = new Logement(67.0d,5e+5);
```

Les tableaux unidimensionnels

L'argument `args` de la méthode `main` est un tableau dont les éléments sont des références vers des chaînes de caractères. Il permet de passer à la méthode main des paramètres en ligne de commande

Si l'on écrit

```
java Programme a 3 5.7 true
```

alors `args` est un tableau de à 1 dimension de taille 4 contenant les références vers les chaînes de caractères "a", "3", "5.7" et "true". Les références sont mises dans le même ordre que sur la ligne de commande. Par exemple, `args[1]` est une référence vers la chaîne de caractères "3".

Outline

Les enveloppes.

Les classes enveloppes des types primitifs

À chaque type primitif est associé une classe qu'on appelle **classe enveloppe** de ce type primitif.
La classe enveloppe du type **int** est la classe `Integer`.

```
Integer entier = new Integer(56);
```

De plus, chaque classe enveloppe d'un type primitif possède une méthode pour convertir une instance de la classe en une valeur du type primitif associé.

```
int i = entier.intValue();
```

Les classes enveloppes des types primitifs

Pour les autres types primitifs, les noms des classes enveloppes sont :

Type primitif	classe enveloppe
short	Short
int	Integer
long	Long
byte	Byte
float	Float
double	Double
boolean	Boolean
char	Character

Les classes enveloppes des types primitifs

Pour chaque classe enveloppe, il existe plusieurs constructeurs qui permettent de faire des conversions explicites depuis les différents types primitifs ou depuis une chaîne de caractère.

```
Integer i = new Inetger(33);
Integer i = new Integer("6");
```

Ces classes enveloppe permettent aussi de convertir, grâce à des méthodes de classe, une chaîne de caractères en types primitifs (**int**, **float**, **boolean**, ...)

```
int i = Integer.parseInt("6");
double d = Double.parseDouble("6.89");
boolean b = Boolean.parseBoolean("false");
```

Les classes enveloppes : Autoboxing/Unboxing

La version 5.0 de Java introduit un mécanisme d'**autoboxing** (mise en boîte) qui automatise le passage des types primitifs vers les classes enveloppe. L'opération inverse s'appelle **unboxing**.

Avant la version 5.0, pour gérer une liste d'**Integer** on devait écrire :

```
liste.add(new Integer(89));
int i = liste.get(n).intValue();
```

Maintenant avec l'**autoboxing**, le code est beaucoup moins lourd :

```
liste.add(89);
int i = liste.get(n);
```