Java

Présentation de Java

(partie 1) version 2.1 (2022)

Présentation du langage

En quelques puces...

- Naissance de **Java en 1991** chez ***Sun*** pour un projet de *code embarqué (exemple TV, machine à laver, etc)*. Acheté par ***Oracle Corporation*** par la suite.
- Fondés sur une syntaxe proche du *C++* mais simplifié.
- Gratuité pour un usage personnel (sinon licence commerciale).
- Java est un langage orienté *Objet* et strictement basé sur des *Classes*.
- Nombreuses fonctionnalités Réseau et Internet (*applets*)
- Fonctionne en mode interprété grâce à la notion de machine virtuelle (grande portabilité)
- Version actuelle : **Java SE 19 - 19 septembre 2022** (depuis 2014 -> nouvelle version tous les 6 mois)

En quelques puces…

- Red Hat, IBM, Amazon, Alibaba, SAP, Azul Systems, Fujitsu contribuent à Java
- Java est un environnement de programmation puissant et polyvalent.
- L'un des plus répandus dans le monde.
- Très utilisé dans les logiciels d'entreprise.
- Il est le troisième langage le plus utilisé (en mai 2022) derrière *Python* et le C.
- Java peut être utilisé dans des systèmes embarqués, pour développer des applications mobiles.
- Java défini des règles regroupées dans la spécification *JLS* (Java Language Specification) qui indique l'implémentation du langage pour ça soit conforme.

Les versions

Version	Release date	End of Free Public Updates[1][5][6][7]	Extended Support Until
Java SE 8 (LTS)	March 2014	**January 2019 for Oracle (commercial)** December 2030 for Oracle (non-commercial) December 2030 for Azul December 2030 for IBM Semeru At least May 2026 for Eclipse Adoptium At least May 2026 for Amazon Corretto	December 2030[9]
Java SE 9	September 2017	March 2018 for OpenJDK	N/A
Java SE 10	March 2018	September 2018 for OpenJDK	N/A
Java SE 11 (LTS)	September 2018	September 2026 for Azul September 2026 for IBM Semeru At least October 2024 for Eclipse Adoptium At least September 2027 for Amazon Corretto At least October 2024 for Microsoft[10][11]	September 2026 September 2026 for Azul[8]
Java SE 12	March 2019	September 2019 for OpenJDK	N/A
Java SE 13	September 2019	March 2020 for OpenJDK	N/A
Java SE 14	March 2020	September 2020 for OpenJDK	N/A
Java SE 15	September 2020	March 2021 for OpenJDK March 2023 for Azul[8]	N/A
Java SE 16	March 2021	September 2021 for OpenJDK	N/A
Java SE 17 (LTS)	September 2021	September 2029 for Azul At least September 2027 for Microsoft At least TBA for Eclipse Adoptium	September 2029 or later September 2029 for Azul
Java SE 18	March 2022	September 2022 for OpenJDK	N/A
Java SE 19	September 2022	March 2023 for OpenJDK	N/A
Java SE 20	March 2023	September 2023 for OpenJDK	N/A
Java SE 21 (LTS)	September 2023	TBA	September 2031[9]

Legend: Old version | Older version, still maintained | **Latest version** | Future release

LTS (Long Term Support) : une version Long-term support ou LTS désigne une version spécifique d'un logiciel dont le support est assuré pour une période de temps plus longue que la normale.

JDK (Java Development Kit) : désigne un ensemble de bibliothèques logicielles de base du langage de programmation Java, ainsi que les outils avec lesquels le code Java peut être compilé, transformé en bytecode destiné à la machine virtuelle Java

OpenJDK : constitue l'implémentation de référence officielle et libre de Java SE, tel que défini par le Java Community Process et ce, depuis sa version 7. Il est le résultat de l'effort de l'entreprise *Sun Microsystems* à vouloir rendre *Java SE* open source

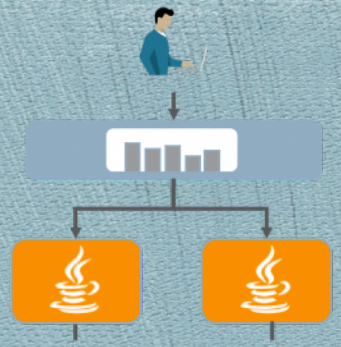
Depuis Java 9 des nouveautés sont introduites: cloud et microservices.

Microservices *: L'architecture de microservices désigne un style d'architecture utilisé dans le développement d'applications. Elle permet de décomposer une application volumineuse en composants indépendants, chaque élément ayant ses propres responsabilités. L'architecture Microservices rencontre un essor fulgurant depuis quelques années. Des géants comme* ***Amazon, Uber, Ebay, Groupon ou encore Netflix****, ont remanié leurs applications et leurs systèmes d'information pour reposer sur cette architecture.*

Les applications qui en résultent sont d'une robustesse et d'une ***scalabilité*** *sans précédent. La complexité de l'application s'en trouve divisée en petits problèmes, facilement abordables. La résilience de l'application s'en trouve ainsi décuplée.*

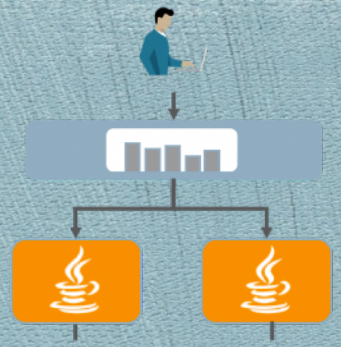
Scalabilité : *faculté d'un produit informatique à s'adapter aux fluctuations de la demande en conservant ses différentes fonctionnalités.*

Vocabulaire



- **JVM** (Java Virtual Machine) est une machine virtuelle qui va *construire* les éléments du logiciel. Quand Java est apparu en 1991 ce fonctionnement était novateur mais de nos jours *.NET* utilise par exemple le même principe de fonctionnement. Nous reviendrons plus tard sur la JVM.
- le **JDK** (Java Development Kit) constitue l'ensemble des outils dont le développeur à besoin. Donc, quand vous téléchargez un JDK celui-ci inclut un JRE compatible avec le version et ce **JRE** comprend lui-même une **JVM**. Il est possible de changer de JRE.
- **JRE** est l'environnement d'exécution Java. C'est donc lui qui va exécuter le code Java.

Vocabulaire



- **Bytecode** (ou Octocode) . Le bytecode est un code intermédiaire (pas du langage ou pas du code machine) qui est exécuté par la JVM. La JVM est une sorte de processeur logiciel. Le format sur un octet permet de traiter très vite les instructions. Chaque octet est transformé par la JVM en codes opératoires pour le processeurs physique.
- Est-ce que Java est un **compilateur** ? Non car le compilateur produit des instructions machine. Or Java (Javac) produit du Bytecode

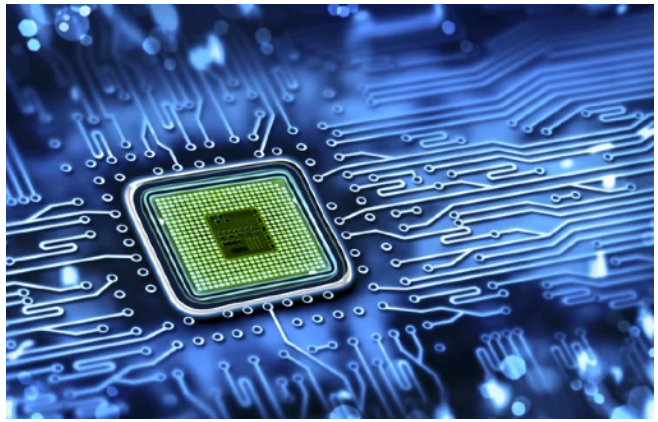
Les environnements

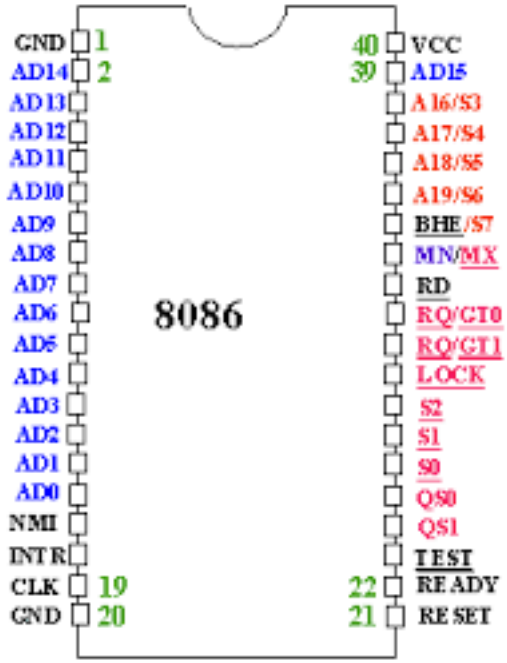
- Il existe plusieurs environnements Java et spécifications :
  - **Java ME** (Java Mobile Edition) -> appareils embarqués
  - **Java SE** (Java Standard Edition) - > env avec lequel on va travailler !
  - **Java EE** (Java Enterprise) -> confié à Eclipse dans le cadre d'un projet nommé Jakarta.
    Alors que Java SE constitue le framework de référence pour Java — avec des bibliothèques standards répondant à la plupart des besoins —, Java EE
    En programmation informatique, un ***framework*** (**cadriciel**[4]) est un ensemble cohérent de composants logiciels structurels qui sert à créer les fondations ainsi que les grandes lignes de tout ou partie d'un logiciel, c'est-à-dire une architecture.

Un peu de théorie

Du binaire à l'objet

60 ans d'évolution

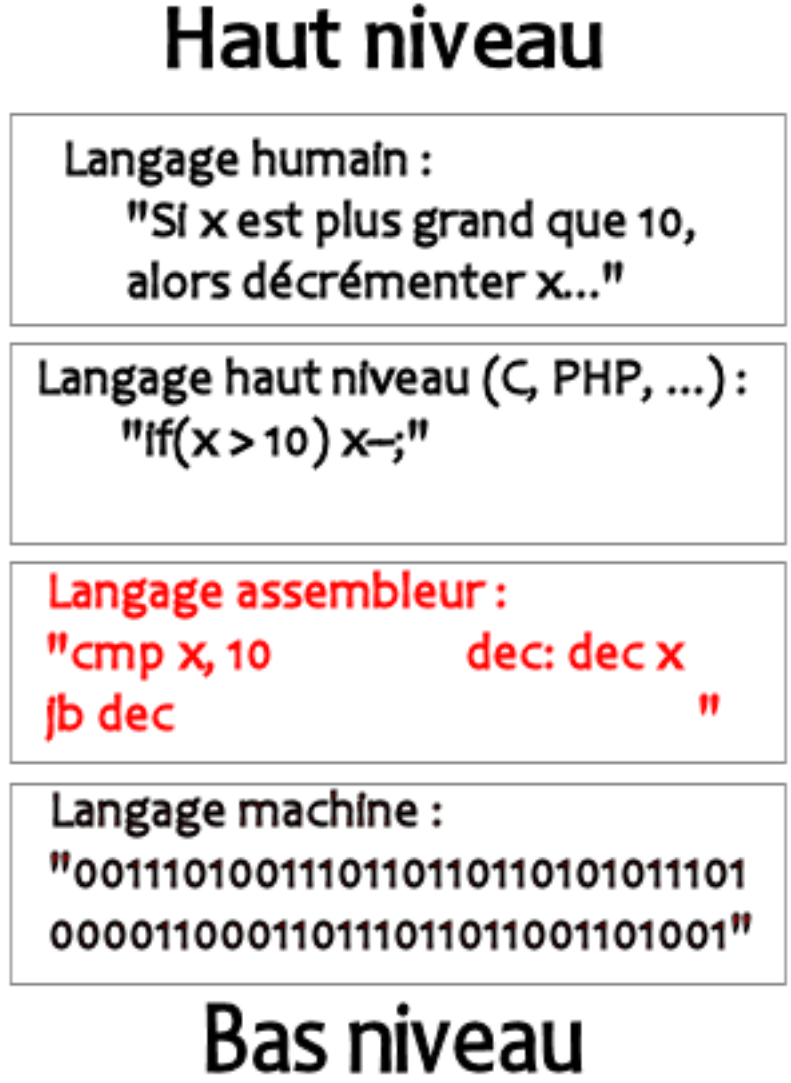





```
// donnees
// debut = @1
B code
dat 0x12
dat 0x20
dat 0x3
dat 0x11
dat 0x0
code:
// R0: adresse à lire, initialisée à 1
// R1: valeur courante
// R2: sommme
MOV R0,#1
MOV R2,#0
boucle:
LDR R1,[R0]
ADD R2, R2, R1
ADD R0, R0, #1
CMP R1, #0
BNE boucle
OUT R2,4
HALT
```

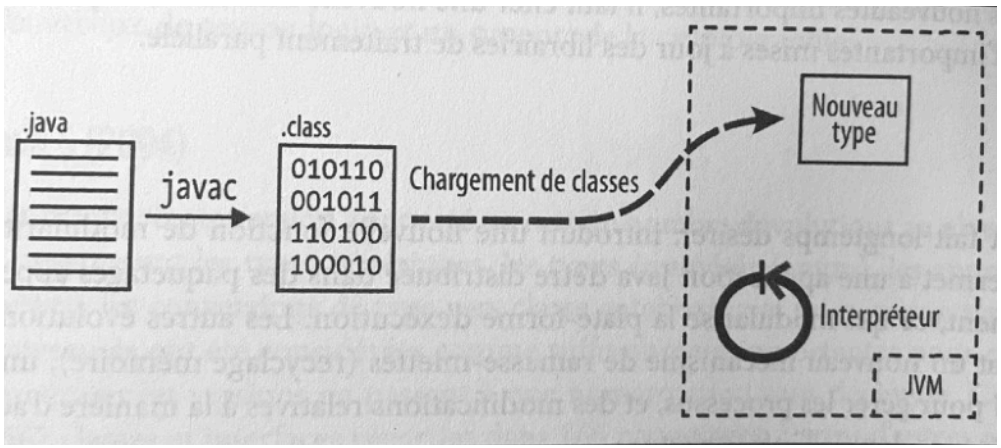
Du binaire à l'objet
60 ans d'évolution

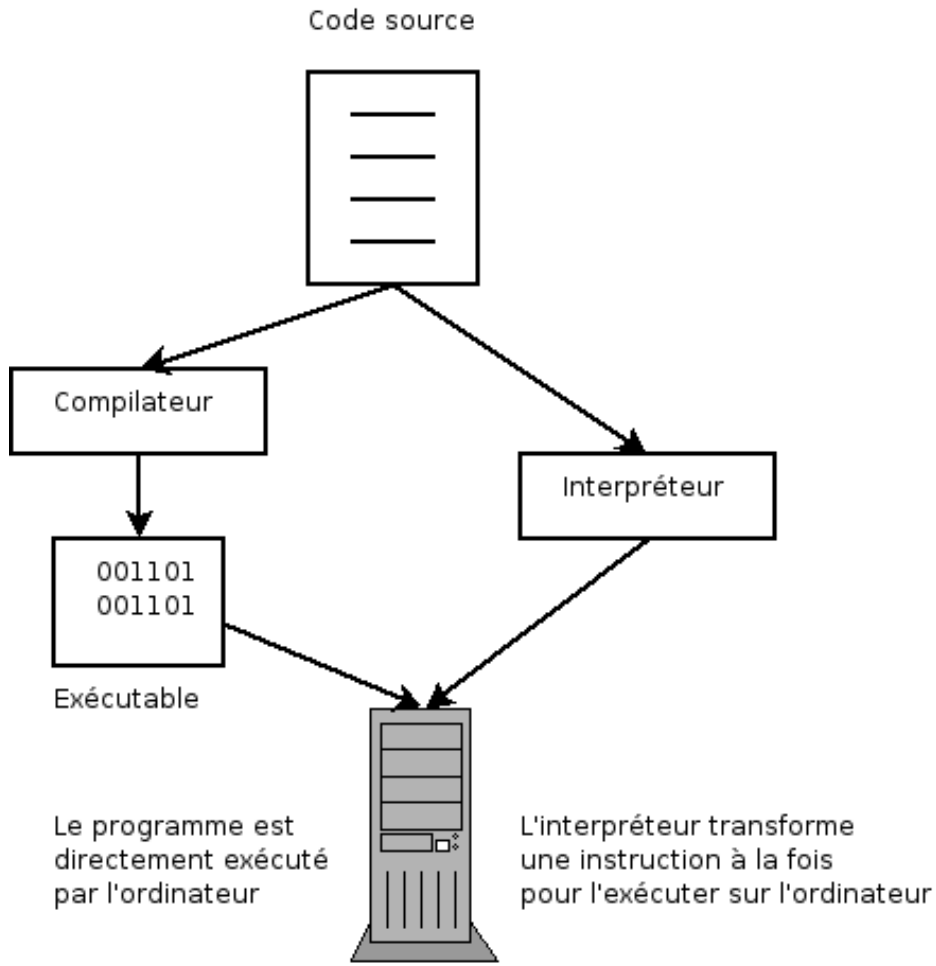
- L'un des premiers langage est le langage assembleur encore très proche du processeur et très obscur pour un simple mortel. **Il y a un langage assembleur par processeur**.
  MOVE 02 R1
  MOVE 10 R2
  ADD R1 R2
- L'idée est de se rapprocher le plus d'un langage humain naturel.



Du binaire à l'objet
60 ans d'évolution

- Le micro-processeur ne manipule que des instructions de base et des données codées en binaire
- L'homme ayant du mal avec les 0 et les 1.
  Il a inventé des langages de programmation.
  Le premier fut l'assembleur.
- Ce langage même minimaliste doit être traduit en binaire (car il va correspondre à des impulsions électriques). Ce traducteur est un compilateur ou un interpréteur



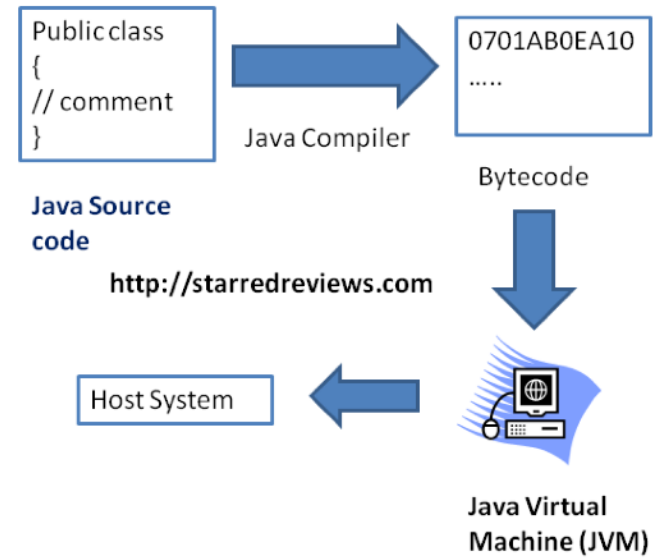


Du binaire à l'objet
60 ans d'évolution

- **L'assembleur** à été abandonné car il dépendait du processeur. Ce qui veut dire qu'un programme ne pouvait fonctionner que pour un processeur précis. En plus de son côté austère et difficile à apprendre.
- **Unix** avait été programmé par *Dennis Ritchie* et *Brian Kerninghan* en assembleur. Mais comme les machines changeaient ils devaient souvent tout reprogrammer pour que Unix fonctionne avec une autre machine (autre processeur)
- Alors ils ont inventé le **langage C** au début des année 1970. Ce fut l'un des premiers langage évolué. Les langages A et B n'ayant pas été concluants.
- Ils ont alors reprogrammé **Unix en C** et quand il y avait un changement d'ordinateur (processeur) ils n'avaient jusqu'à reprogrammer un **compilateur** pour ce processeur mais pas le code d'Unix.

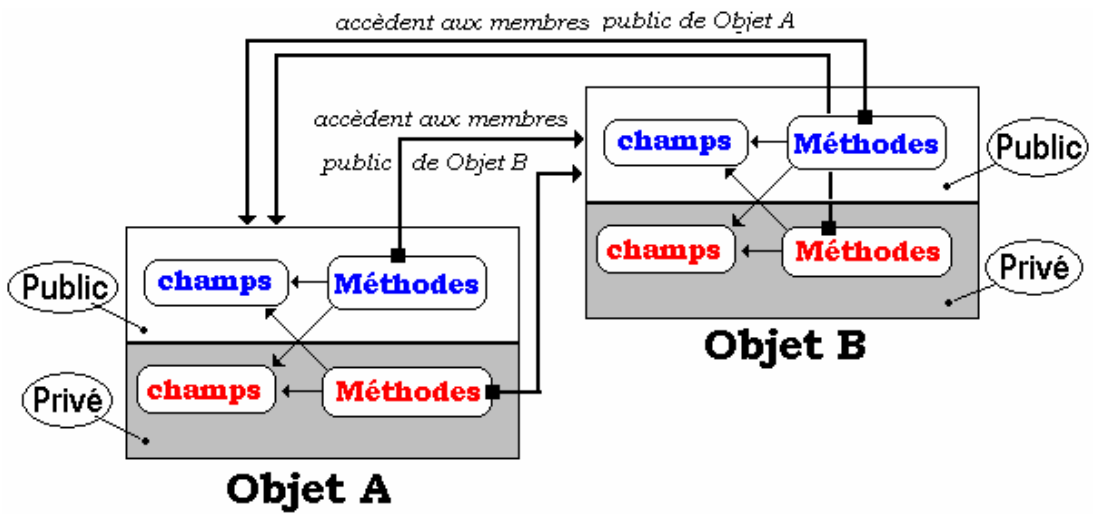
Bytecode

- Le **bytecode** est un pseudo langage «universel».
- Pseudo car il ne peut être directement exécuté par le processeur. Il a besoin pour cela d'une **machine virtuelle**.
- Le bytecode est identique pour tous les S.E c'est la ***machine virtuelle*** qui sera différente.
- Le fichier source porte l'extension *.java*
- Le fichier bytecode porte l'extension *.class*



Du binaire à l'objet
60 ans d'évolution

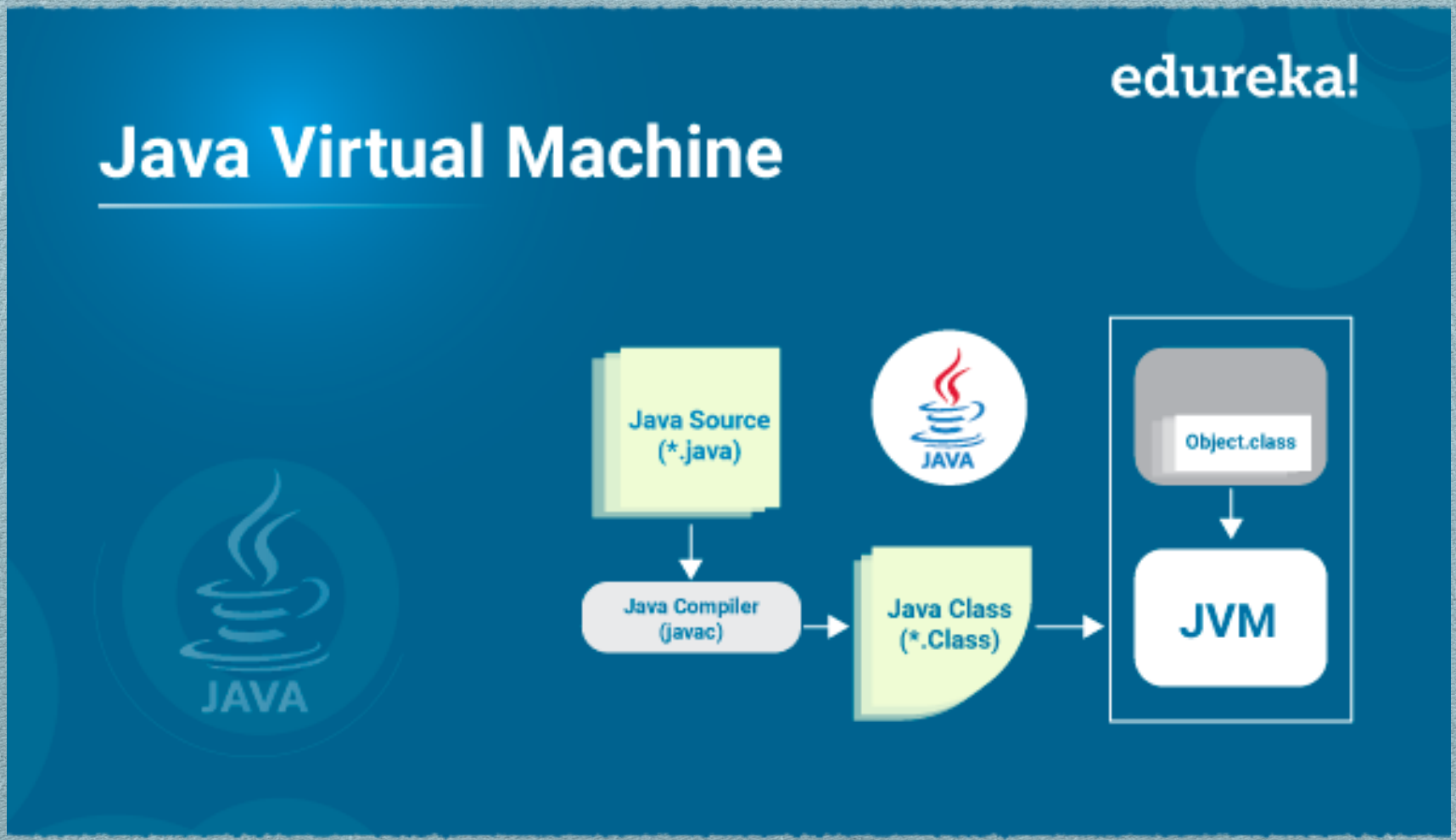
- La programmation objet regroupe les données et les traitements dans un entité nommée : **objet**
- Un objet est une boite noire munie d'une interface (méthodes publiques)
- Dans un programme objet les objets communiquent entre eux par messages.



Qu'est-ce que le langage Java ?

Java

- lignes de code source facile à comprendre pour des humains (développeurs)
- Orienté objets et strictement basé sur des classes
- La syntaxe s'inspire de ces prédécesseurs : C et C++
- Il est parfois un peu long -> verbeux ou bavard
- Grammaire stricte
- Prévu pour des applications professionnels (robuste et stable)



Qu'est-ce que la machine virtuelle JVM ?

JVM

- **JVM** (Java Virtual Machine)
- Il existe des JVM pour les OS (Linux, Mac, Windows) mais aussi téléviseur, lecteur DVD Bluray, Machine à laver, grands système de calcul scientifique.
- La commande **java** *nomDuProgramme* provoque le lancement de la JVM

JVM

- La JVM n'est pas un interpréteur (comme par exemple Python) var le code qui lui est fourni n'est pas le code source mais le Bytecode.
- Les fichiers de type bytecode ont l'extension .class
- Le terme bytecode ou octocode vient que toutes les instructions sont codées sur un seul octet à l'intention de la machine virtuelle.
- Lorsque la JVM voit qu'une opération est consommatrice de temps elle le transforme en code assembleur(code machine)

Premier exemple

Environnement

- Il existe trois environnements d'exécution pour framework Java :
  - Applications
  - Applets (page HTML - Client)
  - Servlet (page HTML - Serveur)
- On s'intéresse nous aux Applications

«Hello World»

Pour écrire un programme Java il faut :

☑ Un éditeur de texte (Coda, SublimeText, Notepad++...)

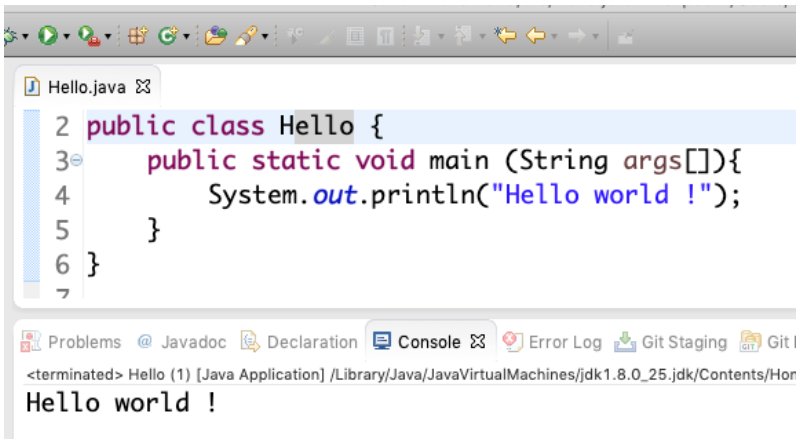
☑ Le compilateur pour générer le bytecode

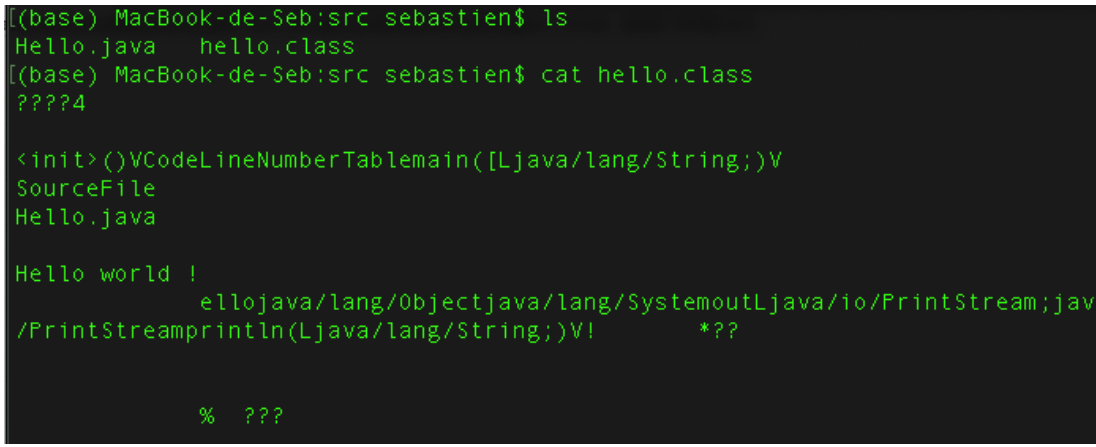
☑ La machine virtuelle

Rassurez vous il existe des environnements de développement intégré (IDE) qui gère tout ça comme ***Eclipse*** que nous utiliserons.

«Hello World»

1. Je tape mon code (ici dans Eclipse)





2. Je l'enregistre avec l'extension .java (ici Hello.java)

Attention : votre programme doit s'appeler comme votre classe (casse comprise)

3. Transformation en bytecode
(javac) puis exécution (java)

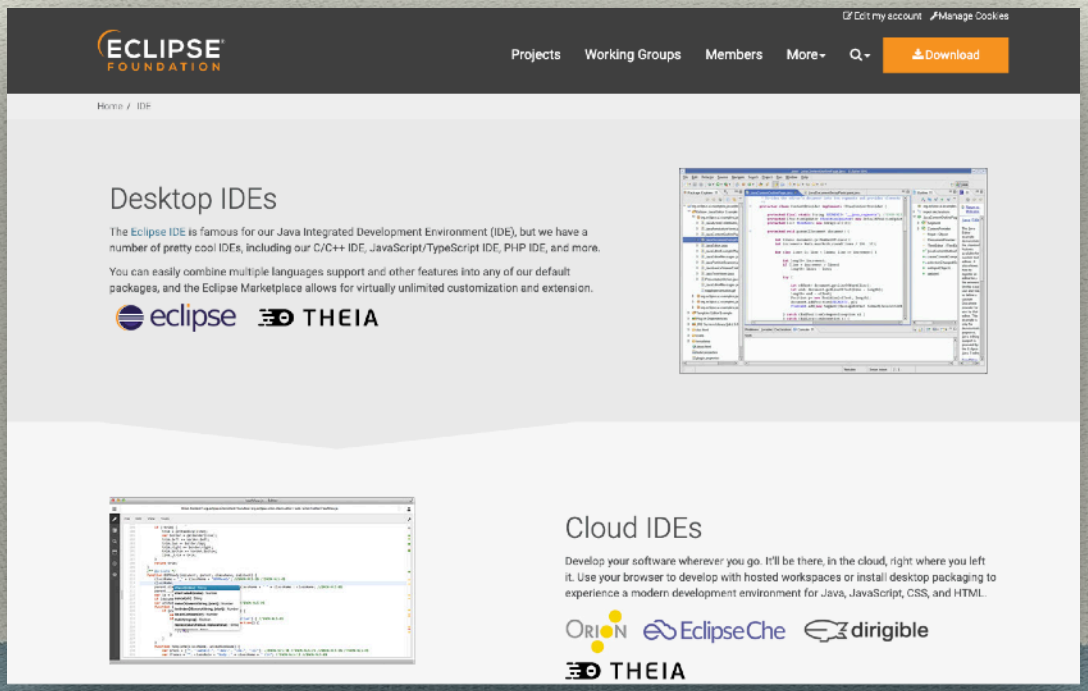
Quand on demande à la machine virtuel d'exécuter hello.class
elle cherche une fonction publique de nom *main*.

Java
Premier petit programme

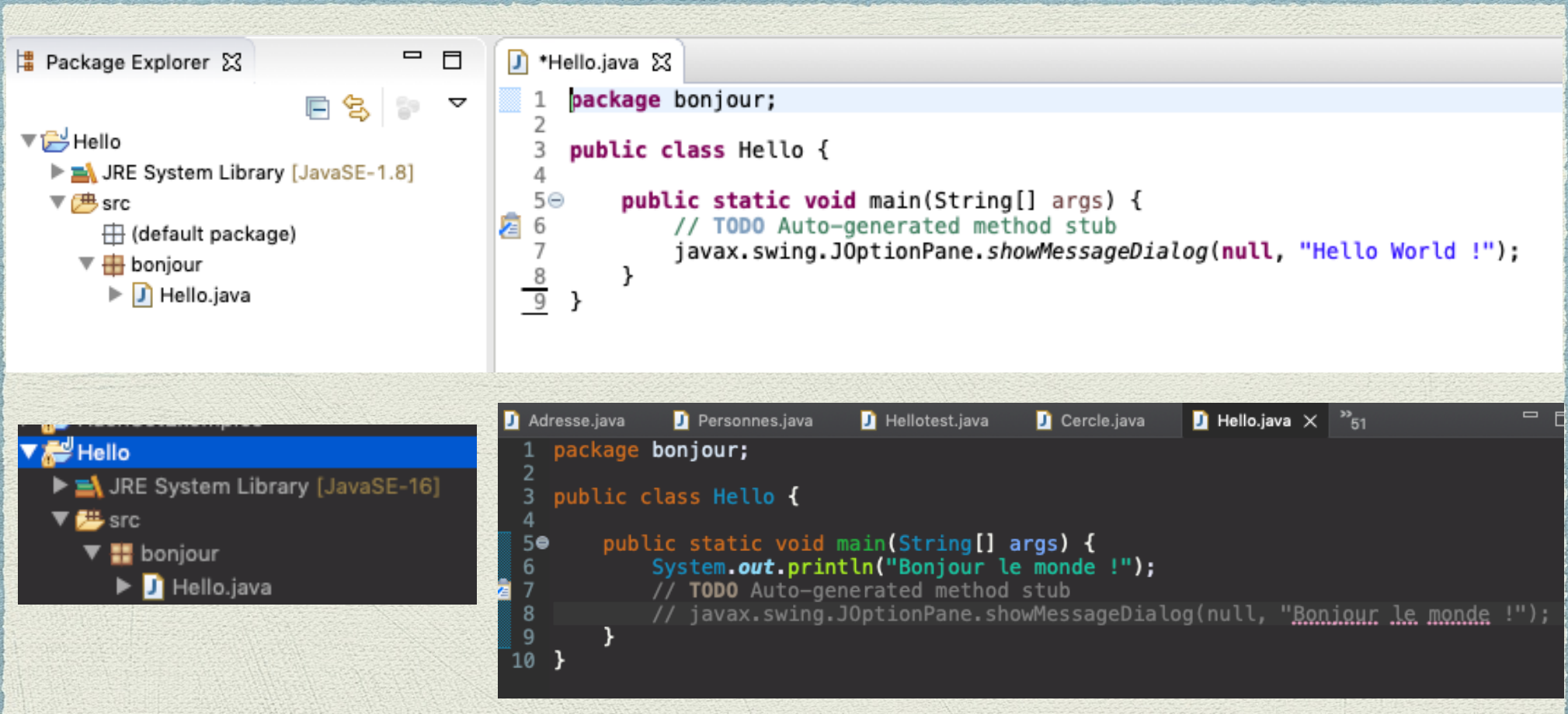
Installation de l'environnement

- **Installer le JDK** : nous avons vu qu'il fallait un JDK pour que Java fonctionne et nous utiliserons OpenJDK https://adoptopenjdk.net/

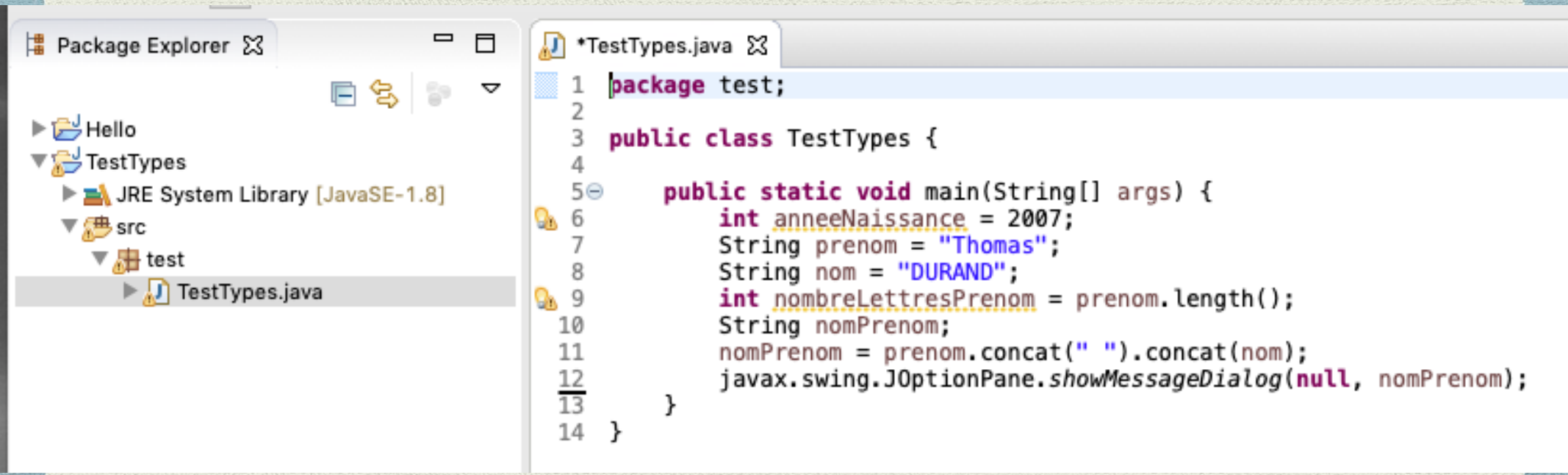
- Installer **Eclipse : https://www.eclipse.org/**



Première fenêtre



Les variables



Les types primitifs

Type	Signification	Taille (en octets)	Plage de valeurs acceptées
char	Caractère Unicode	2	'\u0000' ? '\uffff' (0 à 65535)
byte	Entier très court	1	-128 ? +127
short	Entier court	2	-32 768 ? +32 767
int	Entier	4	-2^{31}?$-2{,}147\times10^{9}$? $+2^{31}-1$?$2{,}147\times10^{9}$
long	Entier long	8	-2^{63}?$-9{,}223\times10^{18}$? $+2^{63}-1$?$9{,}223\times10^{18}$
float	Nombre réel simple	4	$\pm2^{-149}$?1.4×10^{-45} ? $\pm2^{128}-2^{104}$?3.4×10^{38}
double	Nombre réel double	8	$\pm2^{-1074}$?$4{,}9\times10^{-324}$? $\pm2^{1024}-2^{971}$?$1{,}8\times10^{308}$
boolean	Valeur logique (booléen)	1	true (vrai), ou false (faux)

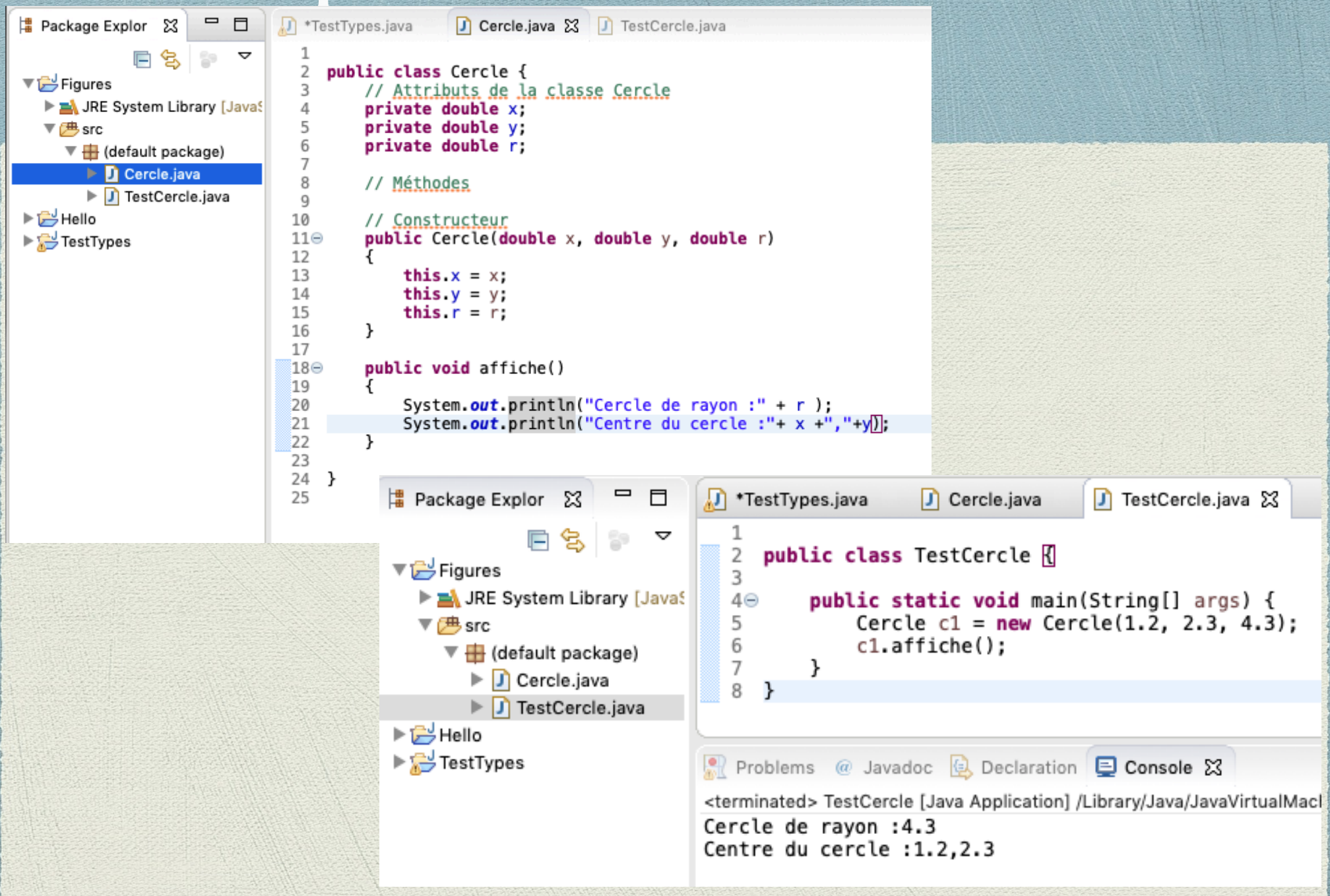
Les attributs

- Les champs ou attributs de la classe doivent être déclaré.
- Un attribut peut être un type primitif ou une classe.
- Si le type de l'attribut est une classe alors son contenu est une référence.
- Pour encapsuler le champ on a recours au modificateur d'accès (private, protected,, public…)
- Un champ est écrit en low Camel.

Les méthodes

- La déclaration d'une méthode est suivie de son implémentation écrite dans le bloc.
- Un bloc en java est défini par des {}
- Une méthode est précédé de son type de retour .
- Si la méthode ne retourne rien on met : void

Exemple


Package Explor
Figures
JRE System Library [Java
src
(default package)
Cercle.java
TestCercle.java
Hello
TestTypes
*TestTypes.java
Cercle.java
TestCercle.java
1 public class Cercle {
2 // Attributs de la classe Cercle
3 private double x;
4 private double y;
5 private double r;
6
7 // Méthodes
8
9 // Constructeur
10 public Cercle(double x, double y, double r)
11 {
12 this.x = x;
13 this.y = y;
14 this.r = r;
15 }
16
17 public void affiche()
18 {
19 System.out.println("Cercle de rayon :" + r);
20 System.out.println("Centre du cercle :"+ x +","+y);
21 }
22
23 }
24
25
Package Explor
Figures
JRE System Library [Java
src
(default package)
Cercle.java
TestCercle.java
Hello
TestTypes
*TestTypes.java
Cercle.java
TestCercle.java
1
2 public class TestCercle {
3
4 public static void main(String[] args) {
5 Cercle c1 = new Cercle(1.2, 2.3, 4.3);
6 c1.affiche();
7 }
8 }
Problems
Javadoc
Declaration
Console
<terminated> TestCercle [Java Application] /Library/Java/JavaVirtualMacl
Cercle de rayon :4.3
Centre du cercle :1.2,2.3

Pratique

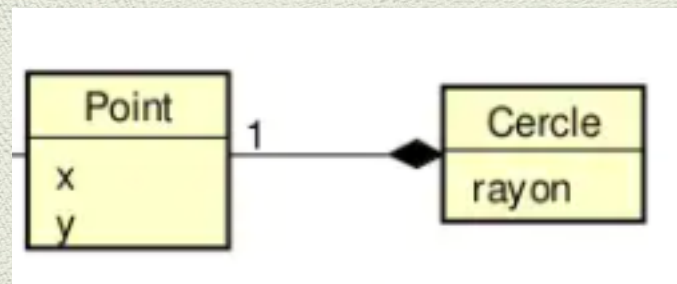
TAF - Figure 1

- Ajouter une méthode déplace() à la classe Cercle qui permet de modifier les coordonnées du centre du cercle.
- Ajouter une méthode surface() à la classe qui retourne la surface du cercle
- Modifier la class TestCercle afin de tester ces nouvelles méthodes.

TAF - Figure 2

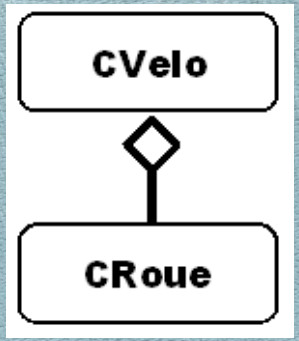
- Le centre du cercle est au final un concept
- Créer cette classe (Centre) -> on veut que ça soit une classe interne. (une classe interne est une classe définit à l'intérieur d'une autre.)
- Cette classe Centre a sa propre méthode affiche()
- On fera donc appelle à cette méthode dans la méthode affiche de la classe Cercle

Composition (UML)



On a un lien d'appartenance
Si on détruit le cercle on détruit le point.
Le point est contenu dans le cercle et est donc une classe interne

L'agrégation



- Une agrégation est un cas particulier d'association non symétrique exprimant une relation de contenance.
- Elle signifie «est de composé de» sans pour autant les lier physiquement. (Elle n'est donc pas interne)
- En programmation cela peut se traduire par des pointeurs ou des références.
- Un objet de la classe Personne peut par exemple avoir un objet de la classe Adresse.
- Cette relation s'utilise simplement en déclarant un champ de type classe.

L'agrégation

- Si je définie une classe Adresse et si je définie une classe Personne.
- Je peux dire qu'une classe Personne possède une Adresse.
- Pour cela je vais déclarer dans les attributs :
  - private Adresse adresse; // par exemple
- Attention cette fois la classe n'est plus inclus dans l'autre classe c'est juste une référence.

TAF

- Réaliser cette agrégation :
  - Adresse (rue, ville, code postal)
    méthodes : getRue(), getCodePostal(), getVille()
  - Personne (nom, prénom, Adresse)
    méthodes : getNom(), getPrenom(), setAdresse(), getAdresse()
  - Créer une personne et associé un objet de type adresse.
  - Changer l'adresse pour cette même personne.

Rappel : Accès

Modificateurs d'accès pour les membres et les classes internes

Modificateur	Signification pour un membre ou une classe interne
public	Accès possible partout où sa classe est accessible (donc partout pour une classe déclarée *public*, depuis le paquetage de la classe pour une classe ayant l'accès de paquetage)
néant (droit de paquetage)	Accès possible depuis toutes les classes du même paquetage (quel que soit le droit d'accès de la classe qui est au minimum celui de paquetage)
protected	Accès possible depuis toutes les classes du même paquetage ou depuis les classes dérivées
private	Accès restreint à la classe où est faite la déclaration (du membre ou de la classe interne)

Transmission des paramètres

- **Certains langages permettent de choisir entre ces deux modes de transfert : par valeur ou par référence. Java emploie systématiquement le premier mode.**

Mais lorsqu'on manipule une variable de type objet, son nom représente en fait sa référence de sorte que la méthode reçoit bien une copie ; mais il s'agit d'une copie de la référence.

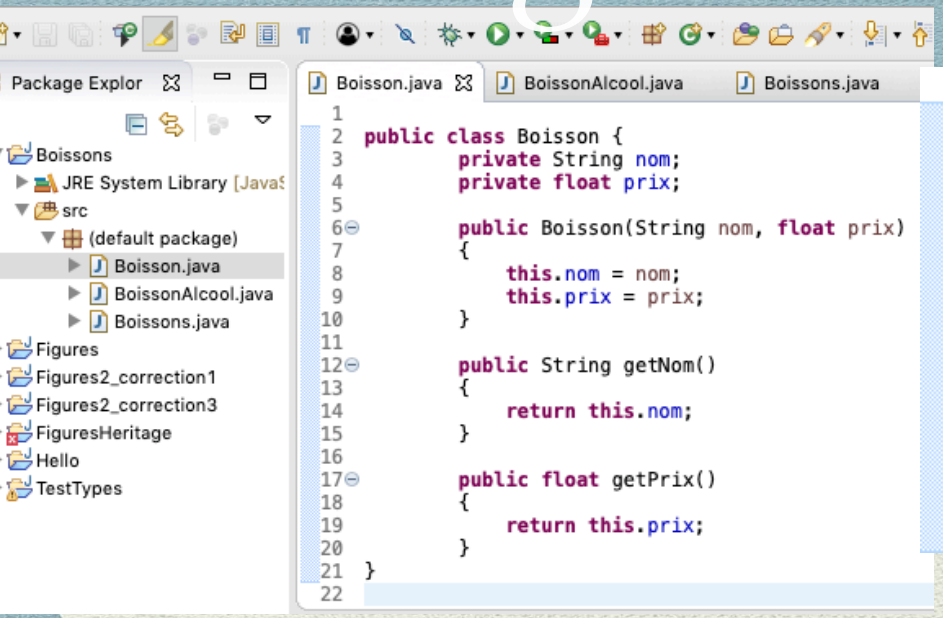
La méthode peut donc modifier l'objet concerné qui, quant à lui, n'a pas été recopié. En définitive, tout se passe comme si on avait affaire à une transmission par valeur pour les types primitifs et à une transmission par référence pour les objets.

- **Idem pour les valeurs de retour.**

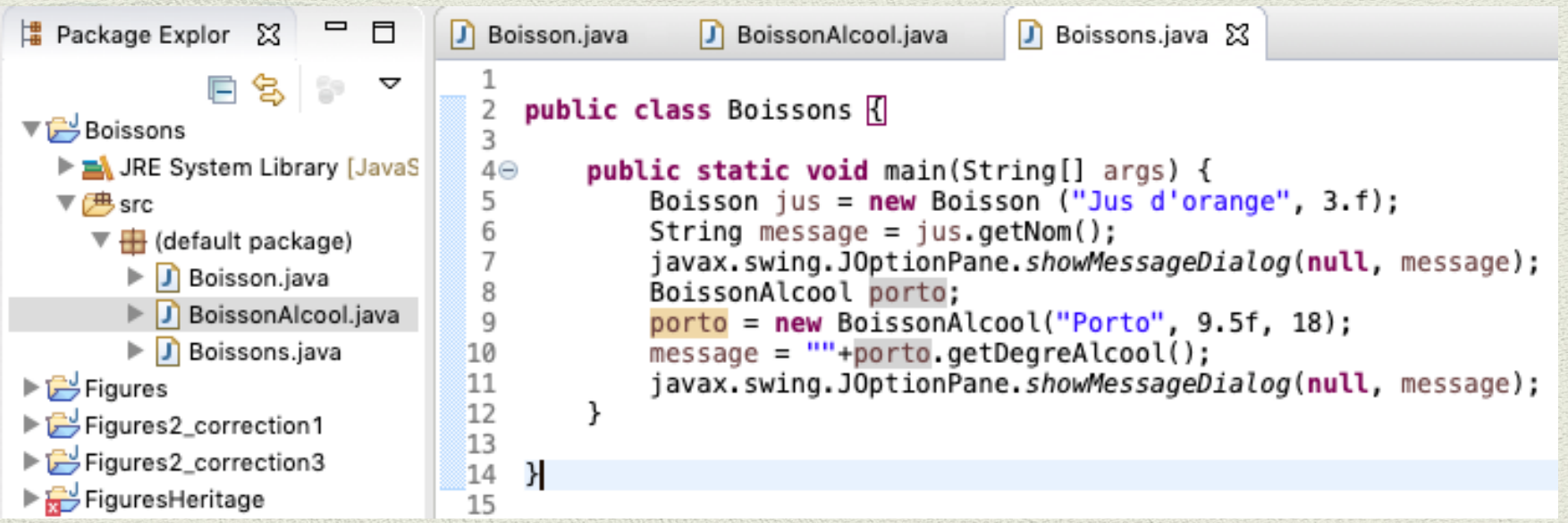
Héritage

- L'idée est aussi de réutiliser du code mais il faut pouvoir dire « est-un »
- Pour qu'une classe hérite d'une autres classe on met le mot **extends**
- La super classe ne se trouve pas forcément dans le même package.
- La sous classe peut définir des méthodes et des attributs qui viennent s'ajouter.

Héritage




```java
public class BoissonAlcool extends Boisson {
    private int degreAlcool;

    public BoissonAlcool(String nom, float prix, int degreAlcool)
    {
        super (nom, prix);
        this.degreAlcool = degreAlcool;
    }

    public int getDegreAlcool()
    {
        return this.degreAlcool;
    }
}
```



Classe abstraite

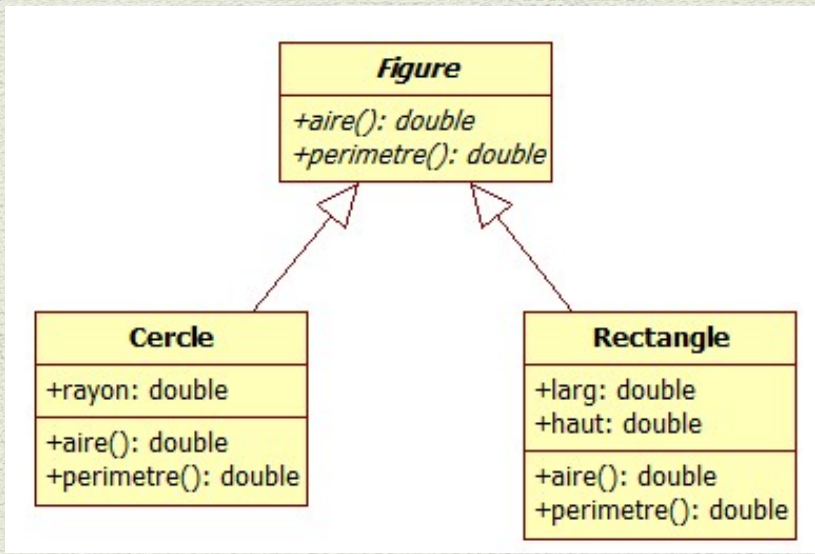
- La super super super classe ;-) représente souvent un concept abstrait.
- Exemple : une figure, une oeuvre
- On ne peut créer un objet à partir de ces classes : elles représentent un modèle.
- Exemple : une figure à une surface mais pour calculer cette surface il me faut connaitre LA figure. (est-ce un cercle ? carré ?)
- Si une classe à une seule méthode abstraite elle devient abstraite.

Classe abstraite

- On ne peut créer une instance d'une sous classe abstrait que si elle a redéfinit toutes les méthodes abstraites de la super-classe.
- Dès qu'une classe ne prévoit pas le corps d'une méthode elle doit être déclarée abstraite
- Mais on peut déclarer une classe abstrait même toutes les méthodes ont été écrites

TAF - Figure Heritage

```
1
2  public abstract class Figure {
3      public abstract double aire();
4      public abstract double perimetre();
5  }
6
```



Terminer l'implémentation de ce schéma UML
Tester avec une classe Figures contenant le main()

Polymorphisme

- Le polymorphisme est la capacité d'une classe à prendre plusieurs formes.
- De même que les conversions numériques on peut « caster » une classe. Cela est autorisé à la compilation si il y a un lien d'héritage
- Exemple : (Boisson)whisky; // whisky est normalement un objet de la classe BoissonAlcoolisee

Polymorphisme

- Quel est l'intérêt de faire une conversion de référence alors que dans la classe mère il y a moins de méthodes à disposition ?
- Il y a deux raisons principales :
  - Si on veux faire une méthode payer il sera plus simple de l'écrire : void payer(Boisson b) car on veut payer une boisson et peut importe la nature de la boisson.
  - Créer un ensemble de références capable de stocker des objets d'une hiérarchie de classe donnée.
    Par exemple : un casier à bouteille

Polymorphisme

```
public class Boissons {

    public static void main(String[] args) {
        Boisson jus = new Boisson ("Jus d'orange", 3.f);
        BoissonAlcool porto = new BoissonAlcool("Porto", 9.5f, 18);
        Boisson uneBoisson;
        uneBoisson = (Boisson)porto;
        javax.swing.JOptionPane.showMessageDialog(null, uneBoisson.getNom());
        javax.swing.JOptionPane.showMessageDialog(null, uneBoisson.get);
        uneBoisson = jus;
        javax.swing.JOptionPane.showMessageDialog(null, uneBoisson.
    }

}
```

getClass() : Class<?> - Object
getNom() : String - Boisson
getPrix() : float - Boisson

On voit bien que les méthodes proposées sont celles de Boisson et non plus Boisson Alcool

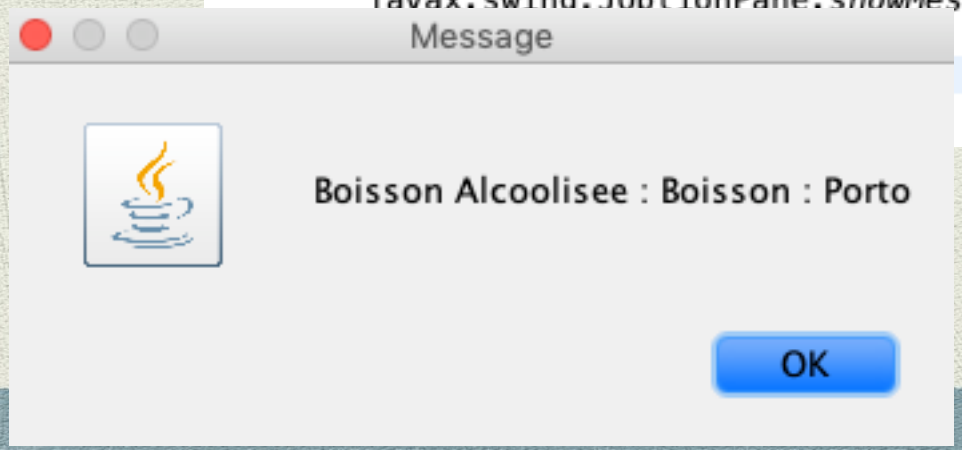
Polymorphisme

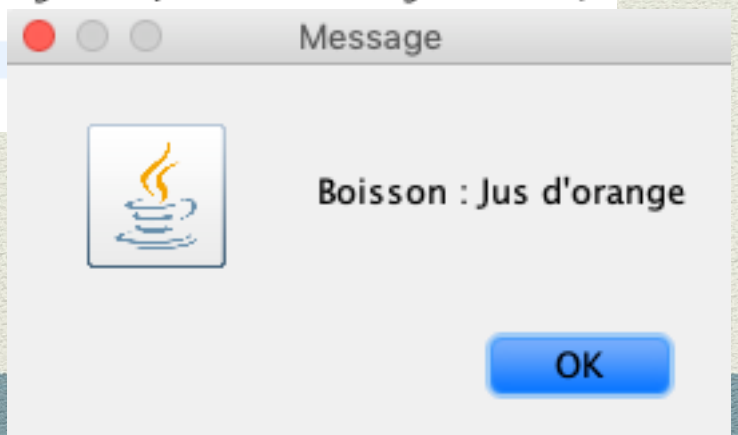
```java
public class Boisson {
        private String nom;
        private float prix;

        public Boisson(String nom, float prix)
        {
            this.nom = nom;
            this.prix = prix;
        }

        public String getNom()
        {
            return "Boisson :"+this.nom;
        }

        public float getPrix()
        {
            return this.prix;
        }
}
```

```java
public class BoissonAlcool extends Boisson {
    private int degreAlcool;

    public BoissonAlcool(String nom, float prix, int degreAlcool)
    {
        super (nom, prix);
        this.degreAlcool = degreAlcool;
    }

    public String getNom()
    {
        return "Boisson Alcoolisee : "+super.getNom();
    }

    public int getDegreAlcool()
    {
        return this.degreAlcool;
    }
}
```

La méthode getNom() appelée est celle définie dans la classe de l'objet et non celle de la classe de référence !

```java
public class Boissons {

    public static void main(String[] args) {
        Boisson jus = new Boisson ("Jus d'orange", 3.f);
        BoissonAlcool porto = new BoissonAlcool("Porto", 9.5f, 18);
        Boisson uneBoisson;
        uneBoisson = (Boisson)porto;
        javax.swing.JOptionPane.showMessageDialog(null, uneBoisson.getNom());
        uneBoisson = jus;
        javax.swing.JOptionPane.showMessageDialog(null, uneBoisson.getNom());
```





TAF - Polymorphisme

- Vous allez taper les codes précédents (Boisson, BoissonAlcool et la classe pour tester Boissons)
- Faire des tests afin de mieux comprendre ce qui se passe. La notion de **polymorphisme** et le comportement est au début surprenant.

Collection

- Création d'une collection avec *ArrayList*:

  ArrayList<String> list = new ArrayList<String>();

- Pour ajouter :

  list.add(« o1 »);

  list.add(« 22 »);

  list.add(« o3 »);

- Pour parcourir :for(String s : list) System.out.println(s);

TAF - Collection

- Créer une collection de différences figures (Cercles,Rectangles)
- Parcourir cette collection pour calculer l'aire totale des figures
- On aura toujours recours à l'héritage avec Figure et on pourra justement utiliser le polymorphisme.

Redéfinition

- Une méthode non private d'une classe est redéfinie (override en anglais) quand elle est définie dans une sous-classe avec la même signature (cad même identificateur, même type de retour, même nombre de paramètres et même type pour chaque paramètre)
- C'était le cas pour les figures géométriques

```
@Override
public double aire() {
    // TODO Auto-generated method stub
    return (double)Math.PI * this.rayon * this.rayon;
}

@Override
public double perimetre() {
    // TODO Auto-generated method stub
    return (double)2 * Math.PI * this.rayon;
}
```

Surcharge

- La surcharge (overload) utilise le même identificateur avec des paramètres différents ou un type de retour différent.

Redéfinition et surdéfinition (surcharge) des méthodes (1):

```
class A
{ .....
        public void f (int n) { ..... }
        public void f (float x) { ..... }
}
class B extends A
{ .....
        public void f (int n) { ..... } // redéfinition du code de f(int)
de A
        public void f (double y) { ..... } // surdéfinition de la
signature de f  par rapport à A et B
}
```

Surcharge

Redéfinition et surdéfinition des méthodes (3): Contraintes portant sur la redéfinition. Lorsqu'on surdéfinit une méthode, on n'est pas obligé de respecter le type de la valeur de retour. Cet exemple est légal :

```
class A { ..... public int f(int n) { .....}
}
class B extends A { ..... public float f(float x) { ..... }
}
```

En revanche, en cas de redéfinition, Java impose non seulement l'identité des signatures, mais aussi celle du type de la valeur de retour.

Surcharge :: constructeur

La surcharge d'une méthode ou d'un constructeur permet de définir plusieurs fois une même méthode/constructeur avec des arguments différents.
Le compilateur choisit la méthode qui doit être appelée en fonction du nombre et du type des arguments .

```
public Rectangle(double larg, double haut) {
    this.larg = larg;
    this.haut = haut;
}

public Rectangle()
{
    this.larg = 0;
    this.haut = 0;
}
```

```
public class Figures {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Cercle c1 = new Cercle(3.2);
        System.out.println(c1);
        Rectangle r1 = new Rectangle(4,2);
        System.out.println(r1);
        System.out.println(r1.aire());
        Rectangle r2 = new Rectangle();
        System.out.println(r2.aire());
    }

}
```

Surcharge :: méthode

```java
@Override
public String toString() {
    return "Rectangle [larg=" + larg + ", haut=" + haut + "]";
}

public String toString(int type) {
    if (type == 0) {
        return "Rectangle [larg=" + larg + ", haut=" + haut + "
    }
    else if (type == 1)
    {
        return larg + "," + haut;
    }
    else return ("");

}
```

```java
public class Figures {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Cercle c1 = new Cercle(3.2);
        System.out.println(c1);
        Rectangle r1 = new Rectangle(4,2);
        System.out.println(r1);
        System.out.println(r1.aire());
        Rectangle r2 = new Rectangle();
        System.out.println(r2.aire());
        System.out.println(r1.toString(1));
    }

}
```

TAF

- Vous aller surcharger la méthode déplace pour qu'elle prenne deux paramètres de type entier cette fois et non plus double.
- A titre d'expérience ajouter un affichage différent dans les deux méthodes.
- Vérifier que quand vous passez des entiers vous avez bien l'une et si vous passer des réels vous avez bien l'autre.

Interface

- L'héritage multiple n'est pas possible en java
- Mais si vous voulez spécifier une liste de méthodes que vous pouvez utiliser sur un objet : il y a les listes.
- Si vous voulez obliger l'implémentation de certaines méthodes dans vos classes : il y a les listes.
- Attention on n'a plus forcément là notion « est un » !
- Une interface Java est une entité distincte d'une classe qui contient une liste de méthodes abstraites.

Interface

```java
public interface Payant {
    float getPrix();
    float prixTTC();
}
```

```java
public class Boisson implements Payant {
        private String nom;
        private float prix;

        @Override
        public float prixTTC() {
            // TODO Auto-generated method stub
            return (float) ((float)prix*1.055);
        }
```

```java
public class BoissonAlcool extends Boisson implements Payant {
    private int degreAlcool;

    public float prixTTC() {
        // Taux de 20 %
        return (float) ((float)this.getPrix()*1.20);
    }
```

Les classes anonymes

Les classes anonymes (1) : Java permet de définir ponctuellement une classe, sans lui donner de nom.

```
class A {
        public A(){.....} //constructeur
        //autres méthodes
}
A a ;
a = new A() {
        // champs et méthodes qu'on introduit dans la classe anonyme derivée de A
} ;
```

Les classes anonymes

Les classes anonymes (2): une classe anonyme ne peut pas introduire de nouvelles méthodes (uniquement redéfinition)

```
class A { public void affiche() { System.out.println ("Je suis un A") ; } }
public class TestAnonyme {
        public static void main (String[] args) {
                A a = new A() {
                                public void affiche () {
                                System.out.println ("Je suis un anonyme
                                dérivé de A") ; }
                        } ;
                a.affiche() ;
        }
} //Affichage du résultat
Je suis un anonyme dérivé de A
```

Les classes anonymes

Les classes anonymes (3): une classe anonyme implémentant une interface

```
interface Affichable { public void affiche() ; }
public class TestAnonyme {
        public static void main (String[] args) {
                Affichable a = new Affichable() {
                        public void affiche () {
                                System.out.println ("Je suis un anonyme
                                implémentant Affichable") ;
                        }
                } ;
                a.affiche() ;
        }
}
// Affichage du résultat : Je suis un anonyme implémentant Affichable
```

TAF

- En reprenant la classe Cercle faites une classe anonyme qui redéfinit la méthode affiche().
- Celle ci peut par exemple afficher « Nouvelle méthode ».
- donc si c2 est ma classe anonyme. c2.affiche() doit donner : Nouvelle méthode

Egalité de deux objets

- Java peut comparer les types primitifs avec le signe ==
- Mais lorsqu'il s'agit d'objet il compare les adresses et pour lui deux objets sont identiques si c'est les mêmes ! (même adresse)
- Il peut être intéressant de dire par exemple que deux points qui ont les mêmes coordonnées sont les mêmes…
- Dans ce cas il faut redéfinir **equals**()

Egalité de deux objets

Dans la classe Point

```
@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }
    if (obj == null) {
        return false;
    }
    if (!(obj instanceof Point)) {
        return false;
    }
    Point other = (Point) obj;
    return Double.doubleToLongBits(x) == Double.doubleToLongBits(other.x)
            && Double.doubleToLongBits(y) == Double.doubleToLongBits(other.y);
}
```

Egalité de deux objets

Dans la classe Point

```
public class Figures {

    public static void main(String[] args) {
        Point p1 = new Point(3,2);
        Point p2 = new Point(3,2);
        if ( p1 == p2)
        {
            System.out.println("Les deux points sont confondus !");
        }

        if ( p1.equals(p2))
        {
            System.out.println("Test avec equals : Les deux points sont confondus !"
        }
```

Egalité de deux objets

- Le hashCode est un entier
- la méthode hashCode est utilisé pour le stockage des objets dans les tables de hash (HashSet, HashMap)
- La règle veut que si obj1=obj2 alors hashCode(obj1)=hashCode(obj2)
- La règle est donc de définir le hashCode en même temps de la méthode Equals()

HastSet

Les collections sont des objets qui permettent de gérer des ensembles d'objets. Ces ensembles de données peuvent être définis avec plusieurs caractéristiques : la possibilité de gérer des doublons, de gérer un ordre de tri, etc. ...

Une collection est un regroupement d'objets qui sont désignés sous le nom d'éléments.

- Une collection de type Set ne permet pas l'ajout de doublons ni l'accès direct à un élément de la collection. Les fonctionnalités de base de ce type de collection sont définies dans l'interface java.util.Set.

Méthode	Rôle
boolean add(E e)	Ajouter l'élément fourni en paramètre à la collection si celle-ci ne le contient pas déjà et renvoyer un booléen qui précise si la collection a été modifiée (l'implémentation de cette opération est optionnelle)
boolean addAll(Collection<? extends E> c)	Ajouter tous les éléments de la collection fournie en paramètre à la collection si celle-ci ne les contient pas déjà et renvoyer un booléen qui précise si la collection a été modifiée (l'implémentation de cette opération est optionnelle)
void clear()	Retirer tous les éléments de la collection (l'implémentation de cette opération est optionnelle)
boolean contains(Object o)	Renvoyer un booléen qui précise si la collection contient l'élément fourni en paramètre
boolean containsAll(Collection<? > c)	Renvoyer un booléen qui précise si tous les éléments de la collection fournie en paramètre sont contenus dans la collection

TAF

- Vous allez créer deux objets cercles.
- Deux cercles sont égaux si il ont le même centre et le même rayon.
- Placer les cercles dans une collection de type HashSet et vous utiliserez la méthode contains() afin de savoir si un cercle est présent dans cette collection.