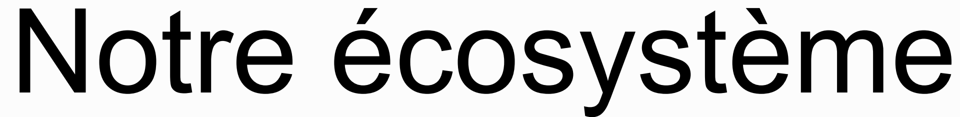
# Cours
# Programmation Orientée Objets

EILCO - ING1

fppt.com

- Comprendre les concepts de la POO
- Maîtriser et s'approprier le langage Java
  - les concepts de la POO s'appliquent à tous les langages
- S'initier aux bonnes pratiques de la POO
  - représentation UML
  - Design patterns

fpt.com




- Java 8  ou une version ultérieure
  https://www.oracle.com/fr/java/technologies/javase/javase-jdk8-downloads.html
- environnement de développement (IDE)
  je recommande IntellijIDEA
  disponible sur https://www.jetbrains.com/idea/
- Eclipse
  Véritable machine à gaz
  https://www.eclipse.org
- Netbeans
  Plus souple que Eclipse
  https://www.netbeans.org
- La ligne de commande

fppt.com

- SmallTalk (années 80)
- Eiffel (année 1986) disponible sur https://www.eiffel.org
- C++ extension du langage C
- Python (très proche de Java)
- Java
- C# (années 2000)

fppt.com

# JAVA c'est quoi ?

- Une technologie développée par SUN Microsystems (acheté par Oracle en 2009) en 1995
  - un langage de programmation
  - une plate-forme, environnement logiciel dans lequel les programmes s'exécutent
- Présente dans de très nombreux domaines d'applications : des serveurs aux téléphones portables

fppt.com

# Quelques caractéristiques de Java

- S'inspire mais se débarrasse des complexités du C (pointeurs, allocation mémoire,..)
- Orienté objet !
- Gestion automatique de la mémoire
- Typage statique fort (à la différence de Python, les variables doivent être déclarées avec leur type)
- Compilation vers le byte-code qui s'exécute dans une machine virtuelle (*cf* infra) → base de la portabilité Java
- Bibliothèque de classes et de packages très riche (graphismes, encryption, …)
- Polymorphisme et introspection

fppt.com

# Situation de Java

- Outil mature pour des environnements hétérogènes
- Estimations (Oracle)
  - 800 millions de PC
  - 2,1 milliards de téléphones portables
  - 3,5 milliards de cartes puce
  - décodeurs, imprimantes, automobiles, bornes de paiement,...
- 2006 passage de JAVA sous licence GPL
- Passage en open source en 2007 avant gratuit **mais pas** open source. Certains fragments de code sont non libres
  → projet IcedTea
- Orientation vers les clients riches

fppt.com

# Idées sur Java

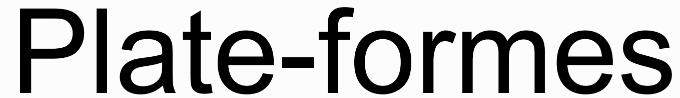
- Java n'est pas sûr
  - Pour les applications : aucune différence entre Java et un autre langage de programmation
  - Pour les applets :
    - Sécurité renforcée
    - Les instructions sont vérifiées par la JVM (machine virtuelle JAVA sur laquelle sont exécutées les instructions Java)
    - Les instructions non autorisées bloquent le programme (exemple écriture locale sur le disque)

fppt.com

# Idées sur Java

- Java est facile à apprendre :
  - Non !
  - Syntaxe simple, proche du C et C++
  - Mais orienté-objet
  - Hiérarchie des classes difficile à comprendre et maîtriser

fppt.com

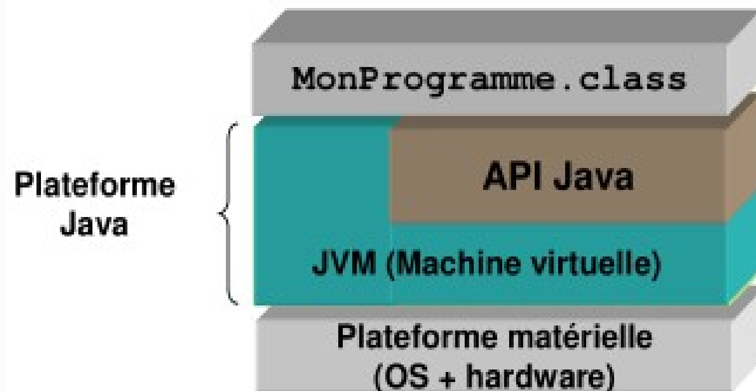



- Plate-forme
  - environnement matériel et/ou logiciel dans lequel s'exécute dans notre case un programme java
  - la plupart des plate-formes sont la combinaison d'un OS et du matériel sous-jacent
    - MS windows + Intel
    - Linux + Intel
    - MacOS + PowerPC/Intel
    - IOS + Apple A5

fppt.com

# Plate-forme java

- L'originalité de la plate-forme java est qu'elle est **entièrement logicielle** et s'exécute au-dessus des plate-formes matérielles

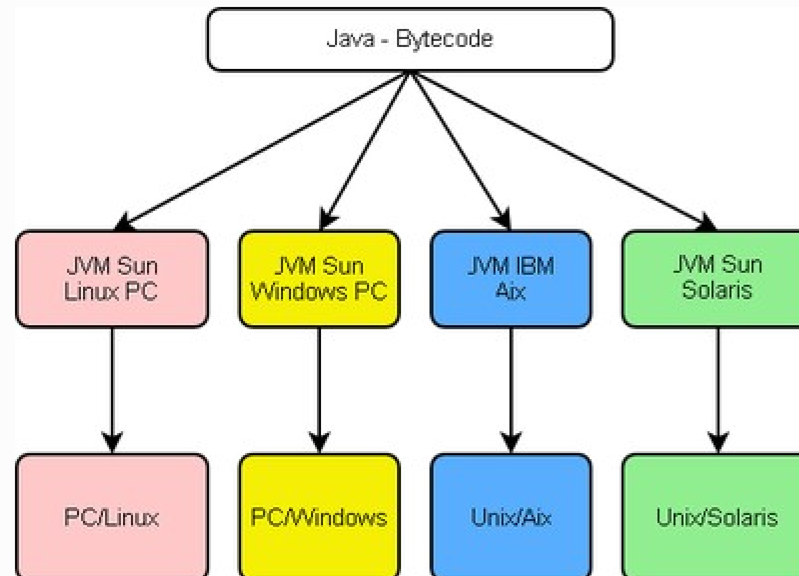



API (Application Programming Interface) : bibliothèques JAVA disponibles pour le programmeur (réseau, graphisme, ...)

fppt.com

# La machine virtuelle java

- le byte-code assure la portabilité des programmes java
  - c'est un langage pour une machine virtuelle ;
  - à l'exécution un interpréteur simule cette machine virtuelle appelée JVM (Java Virtual Machine).



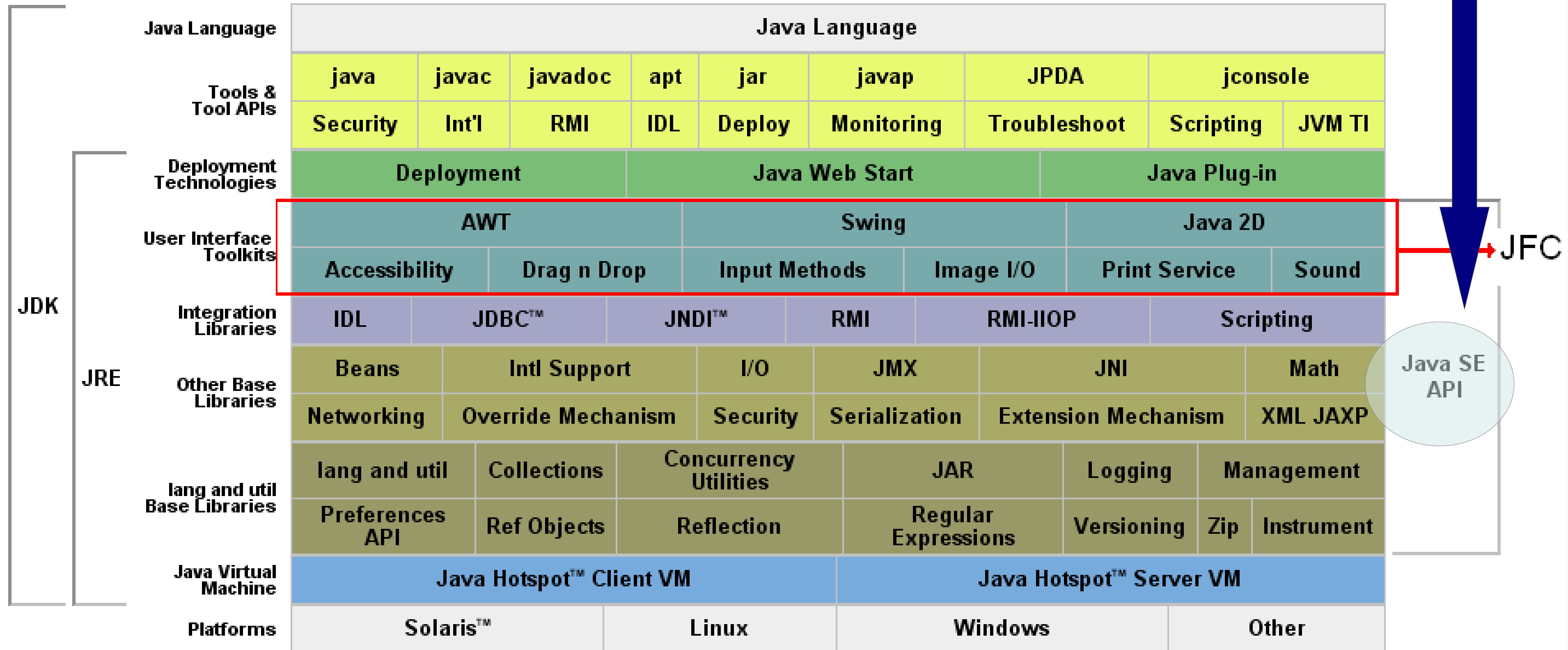

fppt.com

# API java

- très vaste collection de composants logiciels
- organisée en bibliothèques appelées **packages (paquetages en VF)**
- énormément de fonctions/méthodes/services sont disponibles
- programmer en java nécessite de connaître au moins une partie de l'API ou au moins connaître son existence ou savoir retrouver les services nécessaires
  - notion d'API: contrat définissant une classe, les attributs accessibles, le mode d'appel des méthodes etleur retour, ainsi que les erreurs qu'il est possible d'intercepter

fppt.com

# Vue partielle de l'API java




fppt.com

# Vue des API Java

Java™ Platform
Standard Ed. 7

All Classes

**Packages**

java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event
java.awt.font
java.awt.geom

**All Classes**

AbstractAction
AbstractAnnotationValueVisitor6
AbstractAnnotationValueVisitor7
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractCollection
AbstractColorChooserPanel
AbstractDocument
*AbstractDocument.AttributeContex*
*AbstractDocument.Content*
AbstractDocument.ElementEdit
AbstractElementVisitor6
AbstractElementVisitor7
AbstractExecutorService
AbstractInterruptibleChannel
AbstractLayoutCache
AbstractLayoutCache.NodeDimens
AbstractList
AbstractListModel
AbstractMap
AbstractMap.SimpleEntry
AbstractMap.SimpleImmutableEntr
AbstractMarshallerImpl
AbstractMethodError
AbstractOwnableSynchronizer
AbstractPreferences
AbstractProcessor
AbstractQueue
AbstractQueuedLongSynchronizer
AbstractQueuedSynchronizer
AbstractRegionPainter
AbstractRegionPainter.PaintContex
AbstractRegionPainter.PaintContex
AbstractScriptEngine

Overview Package Class Use Tree Deprecated Index Help

*Java™ Platform Standard Ed. 7*

Prev Next Frames No Frames

## Java™ Platform, Standard Edition 7 API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

**Packages**

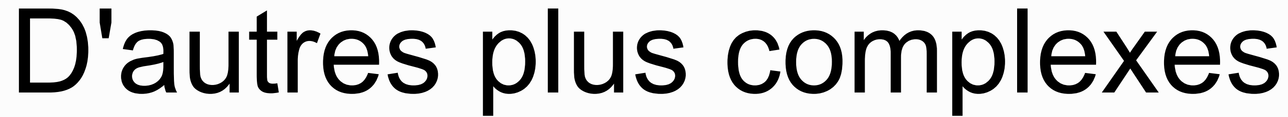
| Package | Description |
|---|---|
| java.applet | Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context. |
| java.awt | Contains all of the classes for creating user interfaces and for painting graphics and images. |
| java.awt.color | Provides classes for color spaces. |
| java.awt.datatransfer | Provides interfaces and classes for transferring data between and within applications. |
| java.awt.dnd | Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI. |
| java.awt.event | Provides interfaces and classes for dealing with different types of events fired by AWT components. |
| java.awt.font | Provides classes and interface relating to fonts. |
| java.awt.geom | Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry. |
| java.awt.im | Provides classes and interfaces for the input method framework. |
| java.awt.im.spi | Provides interfaces that enable the development of input methods that can be used with any Java runtime environment. |
| java.awt.image | Provides classes for creating and modifying images. |
| java.awt.image.renderable | Provides classes and interfaces for producing rendering-independent images. |
| java.awt.print | Provides classes and interfaces for a general printing API. |
| java.beans | Contains classes related to developing *beans* -- components based on the JavaBeans™ architecture. |
| java.beans.beancontext | Provides classes and interfaces relating to bean context. |
| java.io | Provides for system input and output through data streams, serialization and the file system. |
| java.lang | Provides classes that are fundamental to the design of the Java programming language. |
| java.lang.annotation | Provides library support for the Java programming language annotation facility. |
| java.lang.instrument | Provides services that allow Java programming language agents to instrument programs running on the JVM. |

fppt.com

- Principaux paquetages :
  - java.applet : création d'applets.
  - java.awt : création d'interfaces graphiques portables avec AWT.
  - java.bean : création de Java Bean.
  - java.io : accès et gestion des flux en entrées/sorties.
  - java.lang : classes et interfaces fondamentales.
  - java.math : opérations mathématiques.
  - java.net : accès aux réseaux, gestion des communications à travers le réseau
  - ...

fppt.com

# D'autres plus complexes



- javax.crypto : cryptographie.
- javax.imageio :
- javax.jws : Java Web Services.
- javax.lang.model :
- javax.management : API JMX.
- javax.naming : API JNDI (accès aux annuaires).

fppt.com

# Différentes éditions de java

- JSE : Java Standard Edition
  - version pour les ordinateurs de bureau
  - version sur laquelle nous travaillons
- JEE : Java Enterprise Edition
  - version pour les applications « d'entreprises »
  - permet la création d'applications distribuées et de serveurs Glassfish(tm)
  - accès SGBD
  - ...
-

fppt.com

# Java Standard Edition

- 2 principaux composants :
  - JRE (Java Runtime Environment)
    - c'est l'environnement nécessaire à l'exécution d'applets et d'applications créées à l'aide de Java
    - c'est l'implémentation de la machine virtuelle java
  - JDK (Java Development Kit)
    - sur-ensemble du JRE
    - contient le compilateur java
    - javadoc générateur de documentations
    - jdb, le débogueur
    - ...

fppt.com

# Différentes éditions de java

- J2ME : Java 2 Micro Edition
  - version dédiée aux dispositifs mobiles
  - partie réseau et sécurité bien développées
    - JAVA card
    - JAVA TV
    - PDA (Android en partie)
    - Téléphones mobiles
    - …
  - JavaFX
    - pas une version à proprement parlé
    - contenus enrichis et multimédia pour les applications internet
    - glisser/déposer, etc...

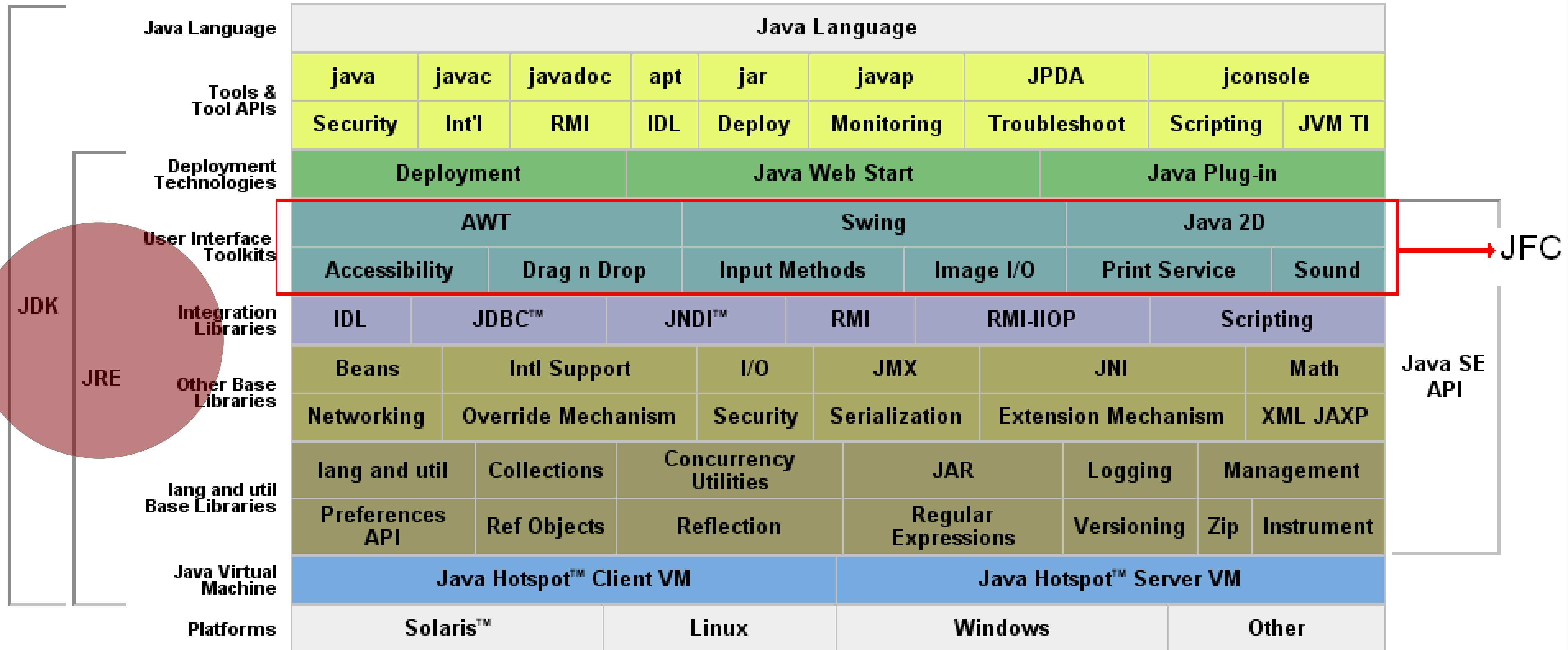
fppt.com

# JSE : situation actuelle

- Début 2021
  - Java SE8 version 1.8 sortie en 2014 (support jusqu'en 2025) version que nous utilisons avec JavaFX (l'année prochaine) Ajout des lambda expressions
  - Java SE7 1.7.0
    - depuis début 2011
    - API comme JAXP (parsing XML)
    - switch avec des String
    - transparence des frames...
    - support jusqu'en 2022
- la version Java SE14 sortie en mars 2020

fppt.com

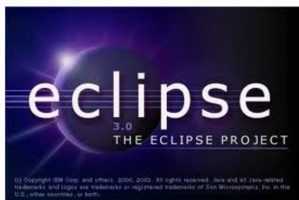
# JSE = JRE + JDK

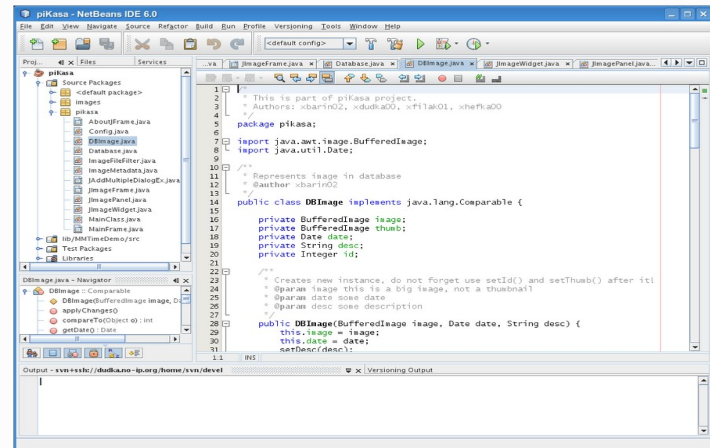



fppt.com

# IDE pour java

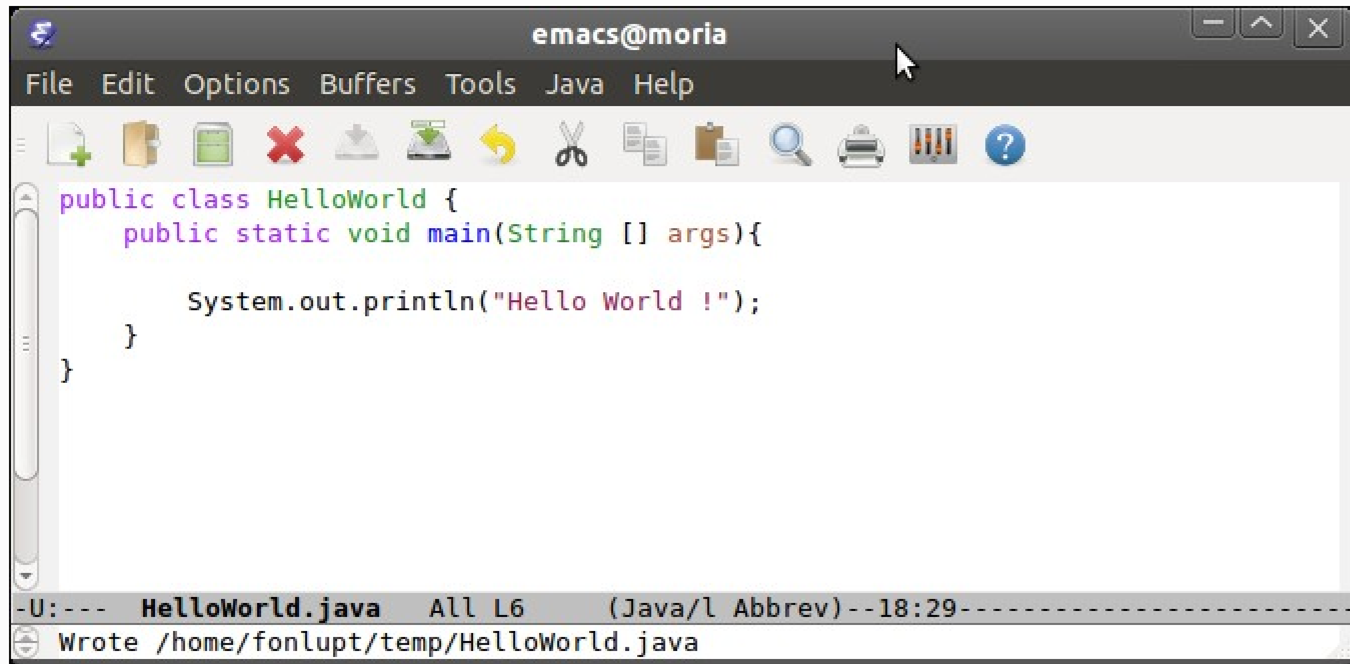
- Très, très nombreux
  - poids lourds
    - Eclipse (libre)
    - Netbeans (oracle, licence open source GPL v2)
    - Intellij IDEA

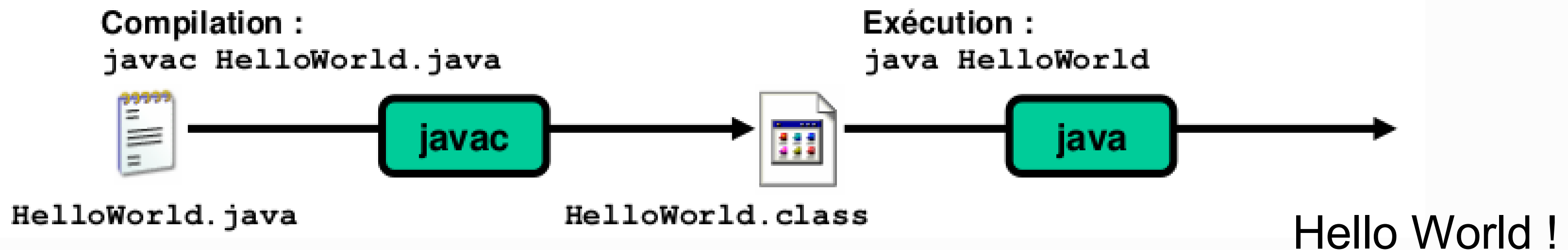



fppt.com

# Mon premier programme




Hello World !

fppt.com

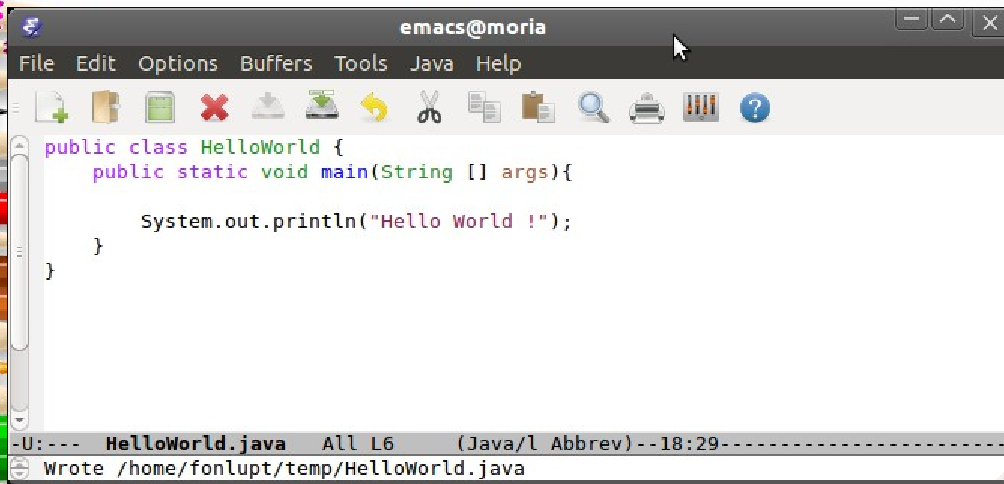
# Remarques

- Tout code java doit être défini à l'intérieur d'une classe :
  public class HelloWorld
- La description de la classe est effectuée à l'intérieur d'un bloc { } ;
- Le code de la classe doit être enregistré dans un fichier de même nom que la classe (casse comprise) HelloWorld.java ;
- Le point d'entrée comme en C est la méthode main (nous reverrons le concept de méthodes ultérieurement)

fppt.com

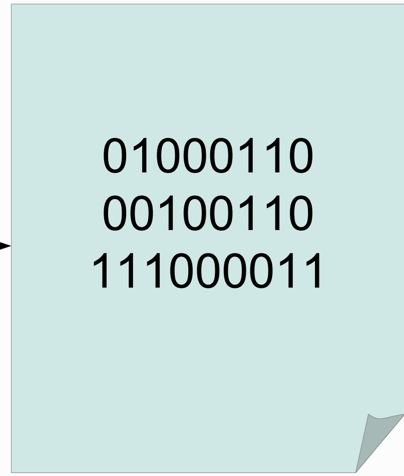
# Un langage compilé/interprété

- Compilation d'un programme java : production de byte-code




HelloWorld.java

javac




HelloWorld.class

**byte-code**

  - indépendant de la plate-forme d'exécution (matériel + OS)

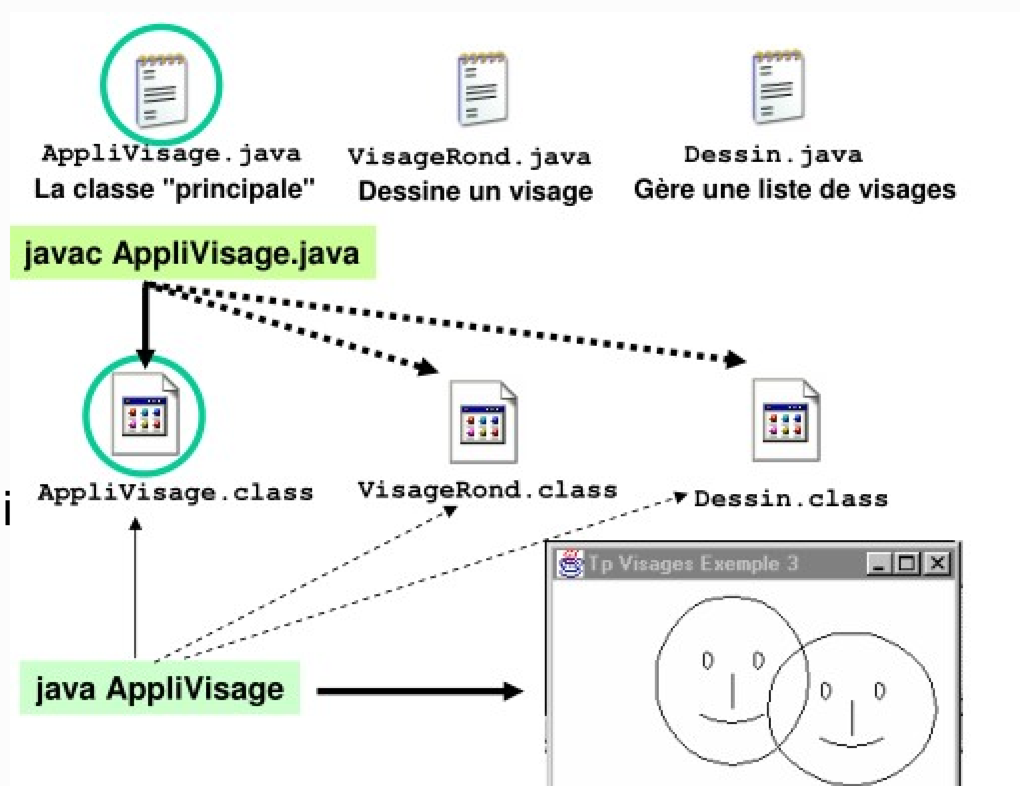
fppt.com

# Application indépendante

- Application est définie par un ensemble de classes dont une jouera le rôle de classe principale

- La compilation de la classe principale entraîne la compilation de toutes les classes utilisées sauf celles qui dont fournies et font partie de la hiérarchie java

- pour exécuter l'application, on indique à l'interpréteur java le nom de la classe principale
- java charge les classes nécessaires au fur et à mesure de l'exécution




fppt.com

# Application indépendante

- L'application doit posséder une classe principale
  - classe possédant une méthode (proche de fonction du C) appelée (on parle de signature)
    public static void main(String[ ] args)

tableau de caractères équivalent à argc ET argv du C

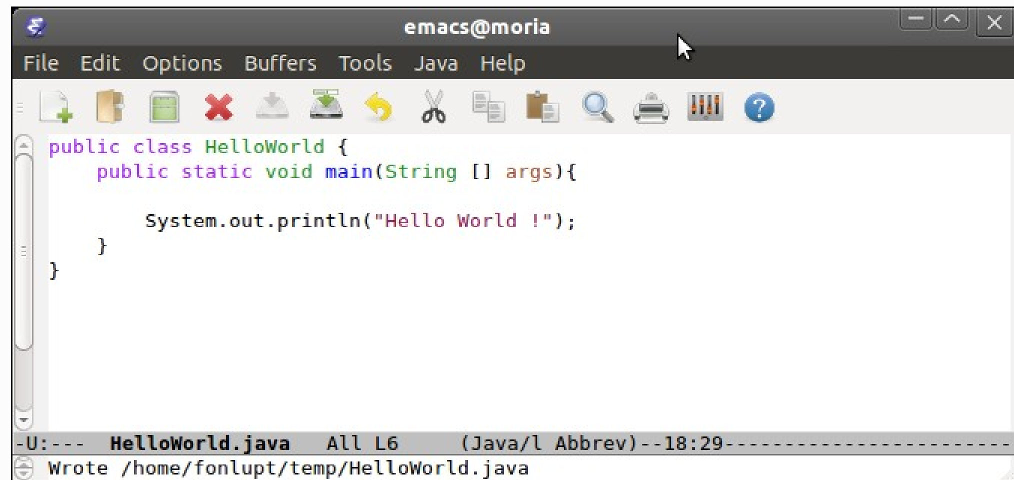
- Cette méthode sert de point d'entrée pour l'exécution
  - l'exécution de l'application démarre par l'exécution de cette méthode
    ex : java AppliVisage1

exécute le code défini dans
la méthode main contenue dans le fichier
AppliVisage1.class

fppt.com

# Retour sur notre premier programme

- System : ceci correspond à l'appel d'une classe « System ». C'est une classe utilitaire qui permet surtout d'utiliser l'entrée et la sortie standard
- out : objet de la classe System qui gère la sortie standard
- print : méthode qui écrit
- exemples Cours1.java et Cours2.java
- méthode : println




fppt.com

# Identificateurs

- Permet de nommer les classes, les variables, les méthodes, …
- Un identificateur Java
  - Est de longueur quelconque ;
  - Commence par une lettre Unicode (https://www.unicode.org);
  - Peut ensuite contenir des lettres ou des chiffres ou le caractères souligné ;
  - Ne doit pas être un mot réservé au langage (mot clef), par exemple if, **switch, class, true,** ...

fppt.com

- Les noms de classe commencent par une majuscule
  - Visage, Objet
- Les mots contenus dans un identificateur commencent par une minuscule:
  - uneClasse, uneMethode, uneVariable
- Les constantes sont en majuscules
  - UNE_CONSTANTE

fppt.com

# Mots-clés Java

- abstract else instanceof static try boolean false assert enum interface strictfp volatile byte true break extends native super while char case final new switch double catch finally package synchronized float
- class for private this int
- const (*) goto (*) protected throw long
- continue if public throws short
- default implements return transient void null
- do import

fppt.com

# Commentaires

- sur une ligne

```
// comme en C++
int i ; // commentaire sur une ligne
```

- sur plusieurs lignes

```
/* comme en C commentaires sur plusieurs
lignes */
```

- commentaires pour l'outil javadoc

```
/**
* pour l'utilisation de javadoc
*/
```

fppt.com

# Commentaires

- Les conseils/consignes utilisés pour les autres langages de programmation sont bien évidemment toujours d'actualité
- commenter le plus possible et judicieusement
- chaque déclaration (variable, méthode ou classe)

fppt.com

- 2 grands groupes de types de données :
  - types primitifs
  - objets (instances de classes)
- Java manipule différemment les valeurs des types primitifs et les objets : les variables contiennent :
  - des valeurs de types primitifs
  - ou des références aux objets

fppt.com

# Types primitifs

- Valeurs logiques
  - boolean (true/false)
  - opérateurs booléens : ! && ||
  - comparaisons ==,  !=,  <, <=, >, >=
- Nombres entiers
  - byte (1 octet), short (2 octets), **int** (4 octets), long (8 octets)
  - nous utiliserons les int (attention les entiers sont toujours signés)
  - format : 17, -4, 42, 0xb0, 0b01110
  - opérateurs : +, -, *, /, % (et d'autres)
- Nombres réels
  - float (4 octets), **double** (8 octets)
  - nous utilisons les double
  - format : 17.0, -4.5, 1.2e3
  - opérateurs : +, -, *, /

fppt.com

# Types primitifs (suite)

- Caractère
  - char (2 octets). jeu de caractère unicode : https://www.unicode.org
    Entre 2 apostrophes
    Tous les caractères accentués du français sont présents.
  - format : 'a', '1', '\u03c0' → π, ...
  - '\t' tabulation
  - '\n' retour à la ligne
- types indépendants de l'architecture
  - un int fait toujours 4 octets quelle que soit l'architecture sous-jacente

fppt.com

# Le type String

- Ce n'est pas un type primitif, c'est une classe (d'où la présence de la majuscule)
- Nous reviendrons plus tard sur les classes
- Comme c'est une classe, on utilise le mot-clef new pour créer un objet, on parle d'une instance de classe de type String
  - String str = new String() ;
  - str = "ceci est une phrase" ;
- la chaîne de caractères de met entre guillemets

fppt.com

# Le type String (suite)

- D'autres possibilités plus classiques de déclaration d'un objet de type String
  - String str ;
  - str = "ceci est une phrase" ;
- ou
  - String str = "ceci est une phrase" ;

- Mais la classe String est un cas particulier, toute utilisation de classe nécessite une instanciation (mot-clef new)

fppt.com

# Casts entre types primitifs

- Conversion automatique
  - entier → réel ou petit entier → grand entier
  - Pas de conversion automatique en booléen comme en C (tout ce qui est différent de 0 n'est pas vrai)
- Attention à la perte de précision dans l'autre sens
  - conversion d'un long en float

fppt.com

# Illustration

- Les affectations entre types primitifs peuvent utiliser un *cast* implicite s'il n'y a pas de perte de précision
  - int i = 180 ;
    double v = 4.0*i ;
- Sinon le *cast* doit être explicite
  - short s = 65 ;
    s = 10000 ; erreur de compilation
    s = (short) 100000 ; ok mais valeur erronée

fppt.com

- Les plus utilisés
  - Arithmétiques
    - + - * /
    - % modulo
    - ++ – (pré ou post incrémentation/décrémentation) cf cours C
  - Logique
    - && (et) || (ou) ! (négation), comme en langage C
  - Relationnels
    - >= <= == != < > (toujours inspiré du C)
  - Affectation
    - = += -= /= *=

fppt.com

# Ordre de priorité des opérateurs

Ordre de priorité des opérateurs

<table>
<tr><td>Postfixés</td><td>[] . (params) expr++ expr--</td></tr>
<tr><td>Unaires</td><td>++expr --expr +expr -expr ~ !</td></tr>
<tr><td>Création et cast</td><td>new (type)expr</td></tr>
<tr><td>Multiplicatifs</td><td>* / %</td></tr>
<tr><td>Additifs</td><td>+ -</td></tr>
<tr><td>Décalages bits</td><td><< >></td></tr>
<tr><td>Relationnels</td><td>< > <= >= instanceof</td></tr>
<tr><td>Egalité</td><td>== !=</td></tr>
<tr><td>Bits, et</td><td>&</td></tr>
<tr><td>Bits, ou exclusif</td><td>^</td></tr>
<tr><td>Bits, ou inclusif</td><td>|</td></tr>
<tr><td>Logique, et</td><td>&&</td></tr>
<tr><td>Logique, ou</td><td>||</td></tr>
<tr><td>Conditionnel</td><td>? :</td></tr>
<tr><td>Affectation</td><td>= += -= *= /= %= &= ^= |= <<= >>=</td></tr>
</table>

Les opérateurs d'égales priorités sont évalués de gauche à droite, sauf les opérateurs d'affectation, évalués de droite à gauche


© P. Itey - INRIA


fppt.com

# Déclarations

- Comme en C, toute variable doit être déclarée avant d'être utilisée
- Peut être déclarée à n'importe quel niveau dans le code
- Une variable est accessible (visible) dans le bloc où elle a été déclarée jusqu'à la fin du bloc où elle a été déclarée

fppt.com

- Syntaxe identificateur = expression ;
- En java, comme en C il est possible de réaliser des affectations multiples
- *cf* Cours5.java

fppt.com

# Bloc d'instructions

- permet de grouper un ensemble d'instructions en lui donnant la forme syntaxique d'une seule instruction
  - syntaxe { }
  - int k;

```
{
    int i = 1 ;
    int j = 12 ;
    j = j + i ;
    k = j*k ;
}
```

fppt.com

# Raccourcis arithmétiques

```
int i=5;
System.out.println(i++ + ++i);
```

**TABLE 3.1:**
Increment/decrement Java shortcuts.

| (*expression*) | (*meaning*) |
|---|---|
| $i++$ or $++i$ | increment $i$ by 1 |
| $i--$ or $--i$ | decrement $i$ by 1 |
| $i+=5$ | increment $i$ by 5 |
| $i-=5$ | decrement $i$ by 5 |

fppt.com

# Flot de contrôle

- Structure conditionnelle – le if

- Syntaxe **if (** *expression booléenne* **)** *instruction1*

  ou bien

  **if (** *expression booléenne* **)**
  *instruction1*
  **else**
  *instruction2*

- exemple

```
if (i==j){
    j = j -1;
    i = 2 * j;
}
else
    i = 1;
```

Un bloc car *instruction1* est composée de deux instructions

fppt.com

# Flot de contrôle

- structure conditionnelle : le switch

```
1   switch (/*variable*/)
2         {
3           case /*argument*/:
4             /*action*/;
5             break;
6
7           case /*argument*/:
8             /*action*/;
9             break;
10
11          case /*argument*/:
12            /*action*/;
13            break;
14
15           default:/*action*/;
16        }
```

Principe : on évalue la variable après le switch et on exécute les actions en vis à vis du case

fppt.com

# Illustration : le switch

```
1   int nbre = 5;
2
3         switch (nbre)
4         {
5           case 1: System.out.println("Ce nombre est tout petit");
6             break;
7
8           case 2: System.out.println("Ce nombre est tout petit");
9             break;
10
11          case 3: System.out.println("Ce nombre est un peu plus grand");
12            break;
13
14          case 4: System.out.println("Ce nombre est un peu plus grand");
15            break;
16
17          case 5: System.out.println("Ce nombre est la moyenne");
18            break;
19
20          case 6: System.out.println("Ce nombre est tout de même grand");
21            break;
22
23          case 7: System.out.println("Ce nombre est grand");
24            break;
25
26         default: System.out.println("Ce nombre est très grand, puisqu'il est compris entre 8 et 10");
27
28        }
```

fppt.com

# Flot de contrôle

- Boucles : instruction while

- Syntaxe **while (** *expression booléenne* **)**
  *instruction*

- Exemple

```
int i = 0;
int somme = 0;
while (i <= 10){
    somme += i;
    i++;
}
System.out.println("Somme des 10 premiers entiers" + somme);
```

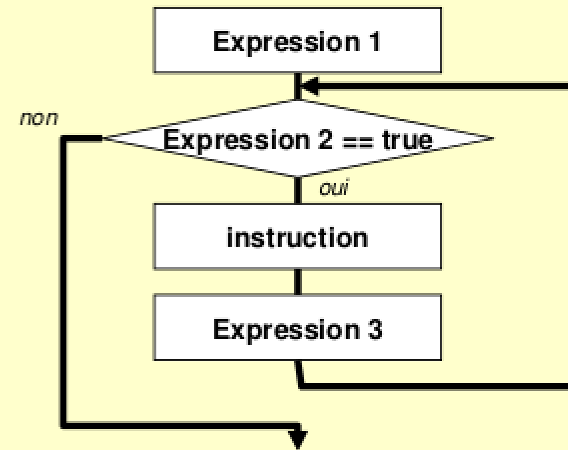
fppt.com

# Flot de contrôle

- boucle – le pour for

- Syntaxe **for (***expression1* **;** *expression2***;** *expression3***)**
  *instruction*

- Exemple




```
int  i;
int somme = 0;
for (i = 0; i <= 10; i++)
   somme += i;

System.out.println("Somme des 10 premiers entiers" + somme);
```

fppt.com

# Affichage sur la console

- System.out.println(chaîne) ou System.out.print(chaîne)
- chaîne est ici :
  - une constante chaîne de caractères (String)
  - une expression convertie automatiquement en String si c'est possible
  - une concaténation d'objets de type String ou convertis en String séparés par le symbole +
  - cf Cours6.java

fppt.com

# lecture au clavier

- Pas de moyen simple de faire une lecture au clavier !
- Il faut utiliser la classe Scanner
  - classe comprenant beaucoup de méthodes
  - cf Cours7.java

fppt.com

# Illustration

```
import java.util.Scanner ;

public class CalculatorPanel {
 public static void main(String[] args) {
  int i;
  System.out.println("Entez un entier: ");
  Scanner clavier = new Scanner(System.in);
  i = clavier.nextInt();
  System.out.println("Vous avez entré : "+i);
 }
}
```

fppt.com

# Méthode de Scanner

- nextInt() - gets the next integer
- nextBoolean() - gets the next Boolean
- nextDouble() - gets the next double
- nextFloat() - gets the next float
- nextShort() - gets the next short
- next() - gets the next string (a line can have multiple strings separated by space)
- nextLine() - gets the next line

fppt.com

# Exercice

- Écrire un programme qui convertit les degrés Celsius en Fahrenheit
- Demande à l'utilisateur de rentrer un nombre en degré celsius
- Calcul en utilisant la formule $y=32+(9/5)*x$ (x est la température en ° celsius)
- Affiche la valeur convertie à l'écran

fppt.com

# Jeu de la multiplication

- Pour apprendre la multiplication aux enfants
- Demander à l'utilisateur de multiplier deux chiffres et vérifier le résultat
- Algorithme :
  - Création de 2 nombres aléatoires entiers x et y
  - Affichage des 2 entiers et demande le résultat du produit de ces deux nombres
  - Lecture au clavier du résultat z
  - Si z = x*y, féliciter l'utilisateur sinon suggérer de mieux apprendre ses tables

fppt.com

# Comment créer des nombres aléatoires

Utilisation de la classe Random

https://docs.oracle.com/javase/8/docs/api/java/util/Random.html

```
import java.util.Random;
public class Test {
   public static void main(String [] args){
Random r = new Random();


int a;
a = r.nextInt(100);
System.out.println(a + "\t" + r.nextInt(20));
   }
```

fppt.com

- L'ordinateur choisit un nombre entier entre 1 et 1000
- Le joueur doit le deviner en maximum 10 essais
- Pour chaque tentative, on indique si le nombre à trouver est plus grand ou plus petit que le nombre rentré par l'utilisateur

fppt.com

# Classes et Objets

fppt.com

- L'OO en quelques mots
  - Une boite noire
  - Un objet sans classe n'a pas de classe
  - Les objets parlent aux objets
  - Héritage
  - Le modèle devient la référence → UML
  - Avantages de l'OO

fppt.com

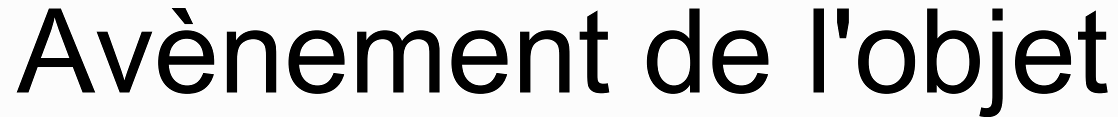
- Programmation dictée par le fonctionnement des processeurs
- Programme = succession d'instructions
- Organisation du programme et nature des instructions le plus proche possible de la façon dont le processeur les exécute
  - Modification des données mémorisées
  - Déplacement des données d'un emplacement à un autre
  - Opérations arithmétiques et de logique élémentaires
- Programmation en langage « machine »
- Exemple : « c = a + b » se programme
  - LD A, REG1
  - LD B, REG2
  - ADD REG3, REG1, REG2
  - MV c, REG3

fppt.com

# Langages procéduraux

- Mise au point d'algorithmes plus complexes
- Nécessité de simplifier la programmation
  - Distance par rapport au fonctionnement des processeurs
- Apparition des langages procéduraux
  - Cobol, Fortran, C, etc...
- Le raisonnement reste néanmoins conditionné par la conception des traitements et leur succession
  - ➔ Éloigné de la manière humaine de poser et résoudre les problèmes
  - ➔ Mal adapté à des problèmes de plus en plus complexes (sauf pour les étudiants très brillants!)

fppt.com



# Avènement de l'objet

- Libérer la programmation des contraintes liées au fonctionnement des processeurs
- Rapprocher la programmation du mode cognitif de résolution des problèmes
- Mise au point d'un nouveau style de langage de programmation
  - Simula, Smalltalk, C++, Eiffel, Java, C#, Delphi, python
- Idée centrale :
  - Placer les entités, objets ou acteurs du problème à la base de la conception
  - Étudier les traitements comme des interactions entre les entités
  - Penser aux données AVANT de penser aux traitements

fppt.com

# Classe

- Une classe est constituée de descriptions de :
  - données : que l'on nomme attributs
  - procédures et fonctions : que l'on nomme méthodes
- Une classe est uniquement un **modèle** de représentation d'objets
  - ayant même structure (même ensemble d'attributs)
  - ayant même comportement (même ensemble de méthodes)
- Les objets sont des représentations, on parle **d'instances** du modèle défini dans les classes
  - une classe permet d'instancier (créer) plusieurs objets
  - en général, un objet est instance d'une classe

fppt.com

# Un exemple de classe

```
public class Carte {

   /* constantes et attributs */

   public final static int PIQUES = 0,       // Codes pour les 4
couleurs
                      COEURS = 1,
                      CARREAUX = 2,
                      TREFLES = 3;


   private  int couleur;   //    PIQUES, COEURS,
CARREAUX, TREFLES.

   private  int valeur;  // valeur de la carte

   public Carte(int valeur, int couleur) {
      this.valeur = valeur;
      this.couleur = couleur;
   }


   public Carte(){
      valeur = 1;
      couleur = PIQUES;
   }

   public int getCouleur() {
      return couleur;
   }

   public int getValeur() {
      return valeur;
   }

   public String getCouleurCommeChaine() {
      switch ( couleur ) {
        case PIQUES:   return "Piques";
        case COEURS:   return "Coeurs";
        case CARREAUX: return "Carreaux";
        case TREFLES:    return "Trefles";
        default:       return "??";
      }
   }


   @Override
   public String toString() {
      return getValeur() + " de " + getCouleurCommeChaine();
   }


} //  classe Carte
```

fppt.com

- Déclaration
  - les attributs correspondent aux valeurs et éléments définissant la classe (ici la valeur et la couleur pour une Carte)
  - la code source e la classe Carte doit **obligatoirement** se trouver dans un fichier de même nom Carte.java
  - En Java, une classe = un fichier. Un programme de 100 classes comportera donc 100 fichiers
- Attributs
  - chaque Carte a deux attributs de type entier : sa valeur et sa couleur
  - chaque **instance** aura (a priori) des valeurs différentes
  - nos attributs sont (par défaut) privés : private et ne sont pas accessibles en dehors de la classe
  - ils sont accessibles en lecture par nos méthodes : public int getCouleur() et public int getValeur()

fppt.com

- Constructeur
  - le constructeur indique comment initialiser une instance de la classe Carte
  - obligatoirement le même nom que la classe
  - des arguments optionnels mais aucune valeur de retour
  - Dans notre cas, plusieurs constructeurs avec des arguments différents
- Note sur this
  - toto.x accède à l'attribut x de toto
  - this dénote l'objet courant
  - par défaut, x dénote l'attribut x de l'objet courant
    → x et this.x sont synonymes
  - x peut être masqué par une variable locale
    → il faut utiliser this.x dans ce cas pour accéder à l'attribut

fppt.com

# Carte construction

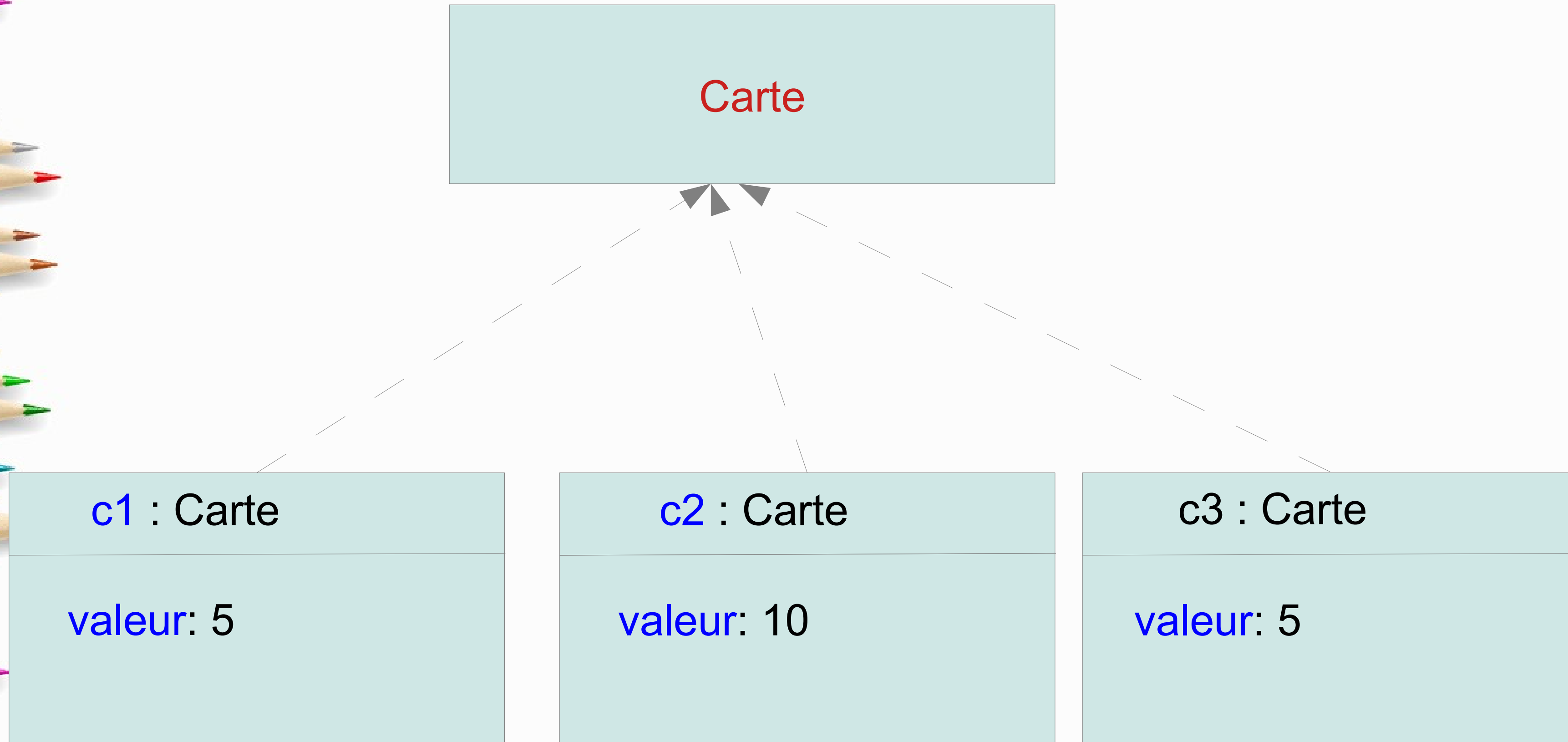
- Le mot-clef new permet de créer une nouvelle instance de la classe Carte :
  - renvoie une nouvelle instance (un nouvel objet) de type Carte
  - doit être stocké dans une variable de type Carte (ou une variable de type sur-classe que nous verrons plus loin)
  - effectue un appel au constructeur de la classe
  - le constructeur appelé est celui qui correspond en nombre et type aux arguments du constructeur

```
Carte c1 = new Carte() ;
Carte c2 ;
c2 = new Carte(10,2) ;
```

fppt.com

# Objets: Carte

- Chaque instance de la classe Carte possédera ses propres attributs (nom, prénom, ...)



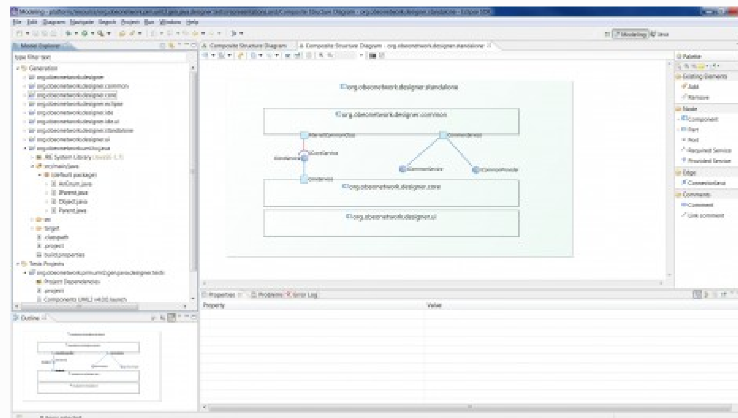

fppt.com

- Les méthodes correspondent aux actions que l'on peut effectuer la une Carte (obtenir sa valeur, l'afficher,…)
- Appel
  - toto.méthode(expr1,…,exprN) pour appeler la méthode de l'objet toto
  - à l'intérieur de l'objet méthode(expr1,…,exprN) est équivalent ) this.méthode(expr1,…,exprN)
- Note :
  - private .. méthode(…) est une méthode privée à usage uniquement interne
  - public int getValeur() est une méthode publique disponible dans toutes les classes
  - les attributs sont souvent privés et les méthodes publiques (mais ce n'est pas une généralité)

fppt.com

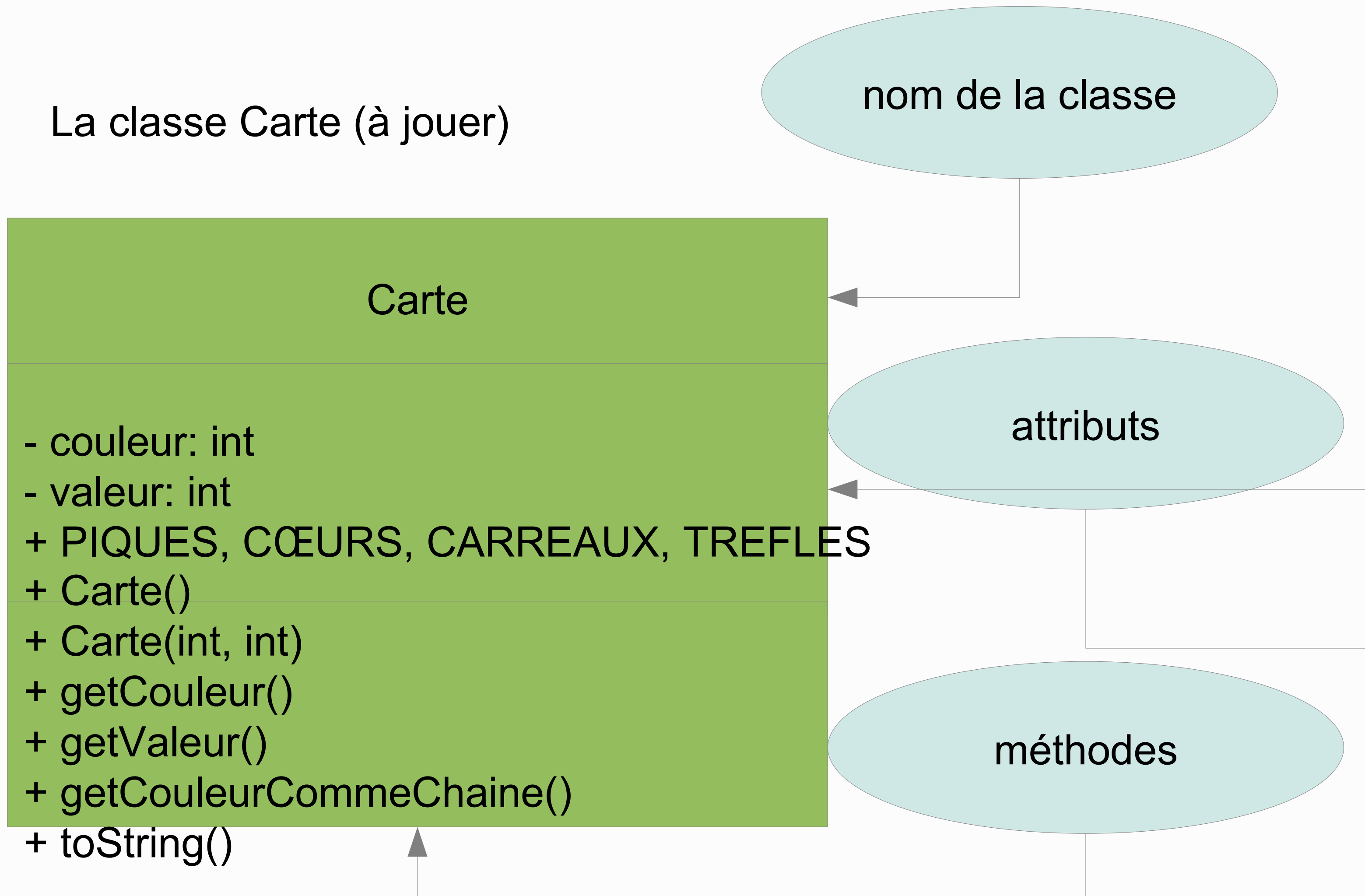
# Diagrammes de classes UML

- UML : unified Modeling Language
  - notation commune et standardisés pour la modélisation orientée objet
  - permet une modélisation graphique des classes et de leurs relations en faisant abstraction de l'implantation du langage
  - modélisation UML → conversion en C++, Java etc.
  - plugin de transcription automatique de UML vers un langage https://marketplace.eclipse.org/content/uml-java-generator



fppt.com

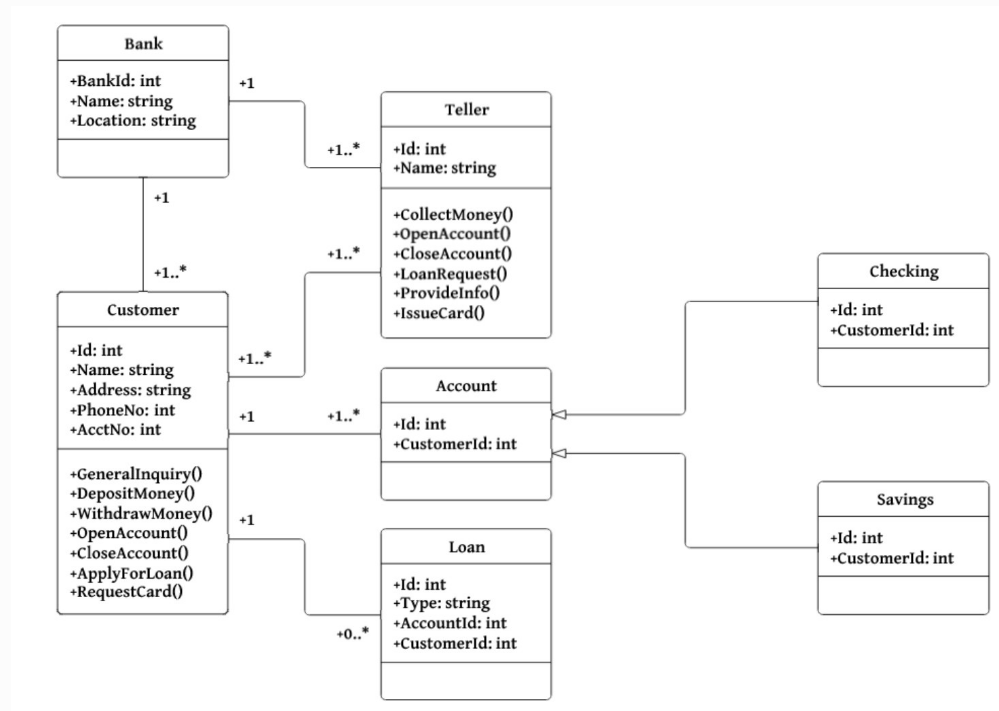
# Exemple de classe




fppt.com

# Diagrammes de classes UML

- 3 blocs
  - nom de la classe
  - liste des attributs et type
  - liste des méthodes et constructeurs avec type de retour
- Visibilité
  - + : publique (public)
  - - : privée (private)
  - ~ : package (non vu)




fppt.com

# Déclaration des méthodes

- Une méthode définit du code exécutable qui peut être invoqué par les objets
- On distingue 3 types de méthodes:
  - Les constructeurs : méthodes servant à créer des objets
  - Les accesseurs : méthodes servant à accéder aux données de nos objets
  - Les méthodes de classe : méthodes servant à la gestion des objets

fppt.com

# Constructeurs

- Méthode appelée automatiquement lors de la création d'une instance de classe (mot-clef new)
- Une classe permet de créer un nombre quelconque d'objets (instance)
- Chaque instance est engendrée par une classe
  - Pour notre classe Carte
    - Un constructeur générique (nom inconnu, …)
    - Un constructeur explicite. On fournit dans ce cas le nom de l'étudiant, son prénom, sa date de naissance, ...

fppt.com

# Accesseurs

- Permet d'accéder aux différents attributs d'une classe
- Récupération des attributs d'une classe
- Modification des attributs d'une classe
  - Pour la classe Carte :
    - Récupération de la couleur
    - Récupération de la valeur
    - Modification de la carte

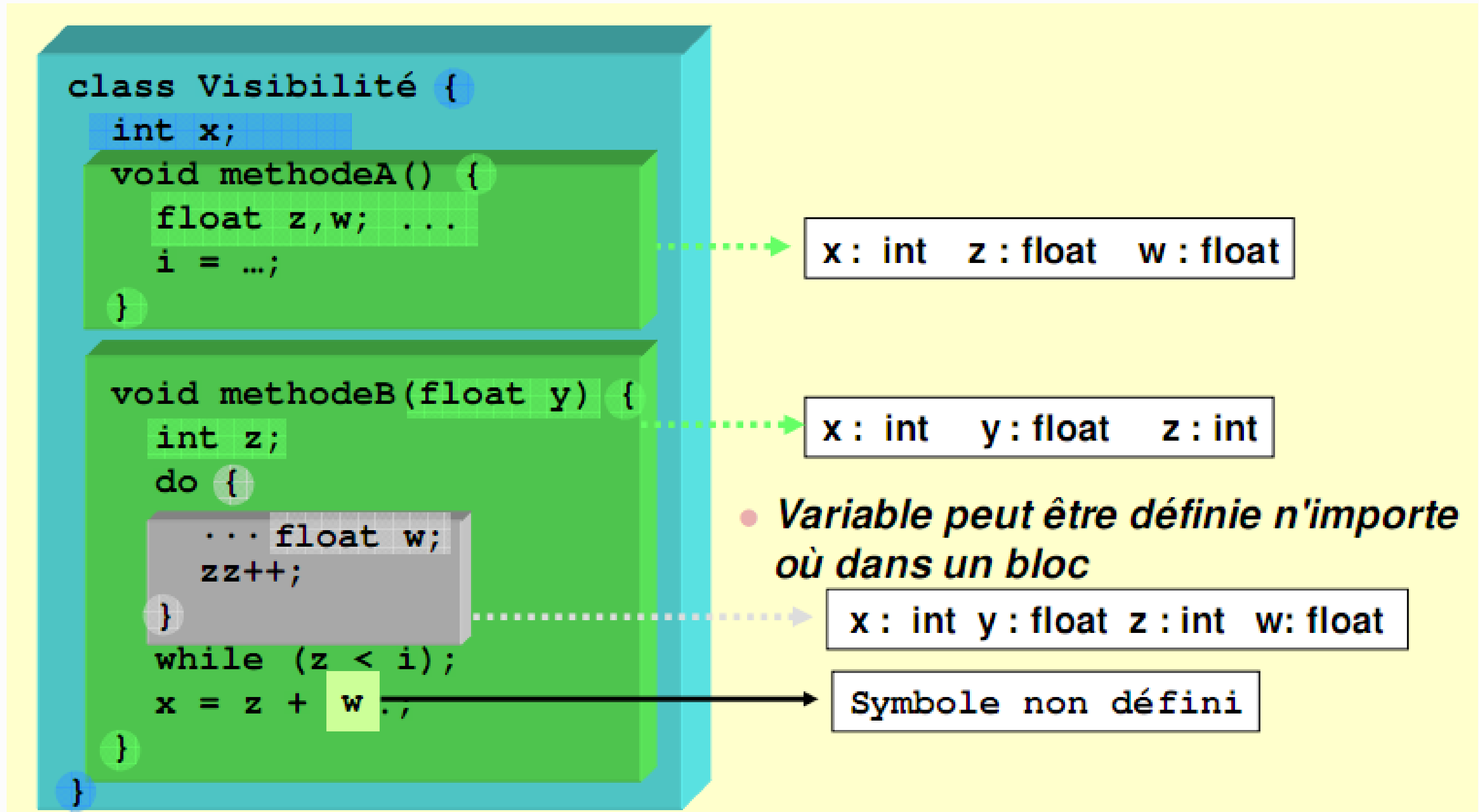
fppt.com

# Autres méthodes

- Permet de travailler directement sur les instances de classe
- Gestion, édition, affichage, calcul sur les objets
- Pour notre classe Carte :
  - affichage de la carte
  - transformation de la couleur en chaîne de caractères
- Pour la classe String
  - toUpper() mise en majuscules
  - subString() récupération d'une sous-chaîne

fppt.com

# Visibilité des variables




fppt.com

# Objets

- Un objet est une instance d'une seule classe :
  - Chaque attribut déclaré dans la classe de l'objet admet une valeur **propre** dans chaque instance
  - Ces valeurs caractérisent **l'état** de l'objet
  - Les méthodes de classes agissent sur les instances. Attention, les méthodes statiques sont indépendantes des objets. Ce sont en général des boites à outils

fppt.com

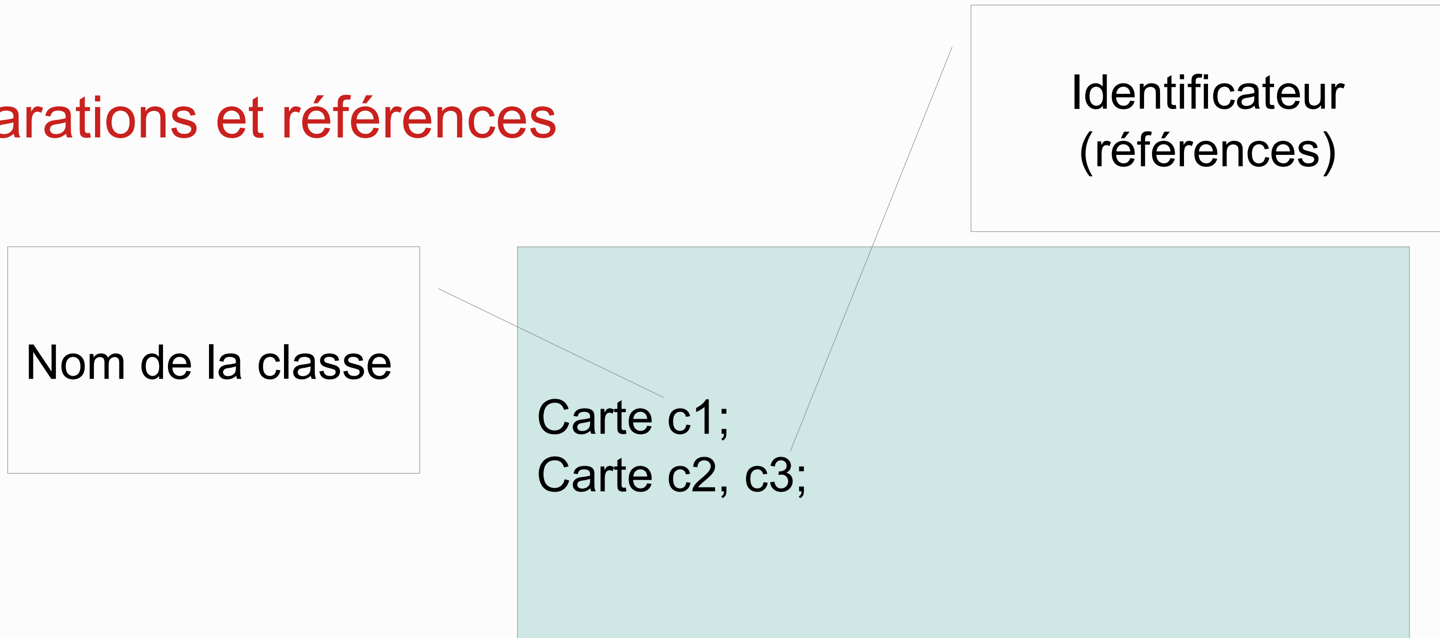
# Gestion de la mémoire

- L'instanciation provoque une allocation dynamique de la mémoire
- En Java, le programmeur n'a pas à se soucier de la gestion de la mémoire
  - Si un objet n'est plus référence (n'est plus utilisé par aucune référence), la mémoire allouée est libérée automatiquement. Ce processus de libération de la mémoire est appelé « garbage collector » ou « ramasse-miettes »

fppt.com

# Références en Java

- Déclarations et références



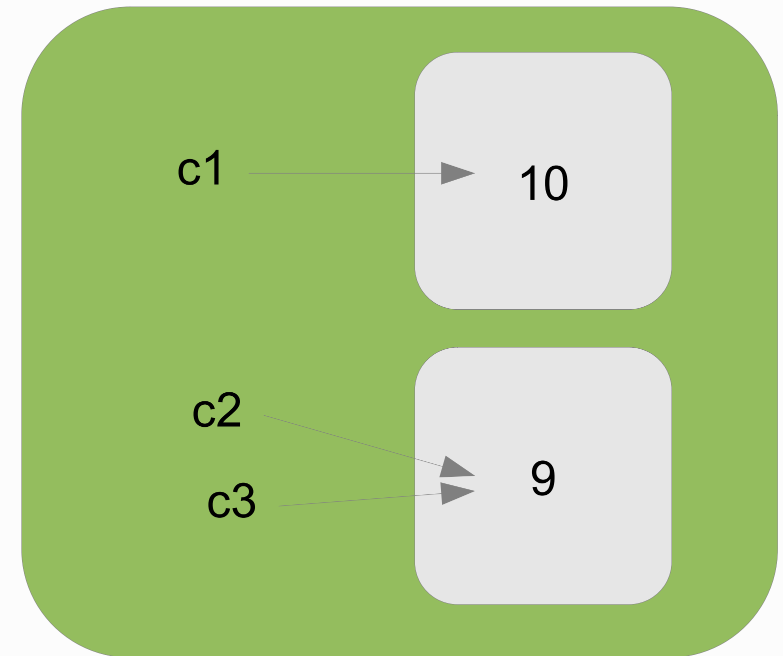

- Par défaut à la déclaration une référence vaut **null**
  - Carte c1; ↔ Carte c1 = null;

fppt.com

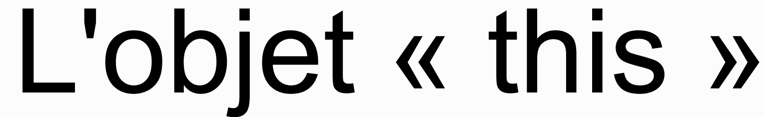
# Accès aux attributs d'un objet

- Pour accéder aux attributs d'un objet, on utilise la notation :
  - **nomObjet.nomAttribut**

```
Carte c1;
c1 = new Carte();
Carte c2 = new Carte();
Carte c3 = c2;
c1.valeur = 10;
c2.valeur = 9;
c3.couleur = 2;
```




fppt.com



# L'objet « this »

- En java, l'accent est mis sur l'objet (et non pas sur l'appel de fonction)
  - e1.ajouterNote(17)
  - e1.plusVieux(e2)
- L'objet qui reçoit un message est implicitement passé comme argument à la méthode invoquée
- Cet argument implicite défini par le mot-clef **this**
  - Désigne l'objet courant (objet récepteur du message)
  - Peut être utilisé pour rendre explicite l'accès aux propres attributs et méthodes définis dans la classe
  - Est en général facultatif

fppt.com

# Encapsulation

- Accès direct aux variables d'un objet possible en java
- Mais... n'est pas recommandé car contraire aux principes des langages orientés-objets
  - Les données d'un objet doivent être privées (c'est-à-dire protégées et accessibles qu'au travers de méthodes prévues à cet effet)
- En Java, on peut définir de manière assez précise la visibilité des membres (attributs et méthodes) d'une classe vis-à-vis d'une autre
  - private
  - public
  - protected

fppt.com

# Encapsulation

- public/private

| | public | private |
|---|---|---|
| classe | La classe peut être utilisée dans n'importe quelle classe | |
| attribut | Attribut accessible depuis n'importe quelle classe<br>Classe.attribut | Attribut accessible uniquement dans la classe où il est défini |
| méthode | Méthode pouvant être invoquée depuis n'importe quelle classe<br>Classe.methode(...) | Méthode utilisable uniquement dans la classe où elle est définie |

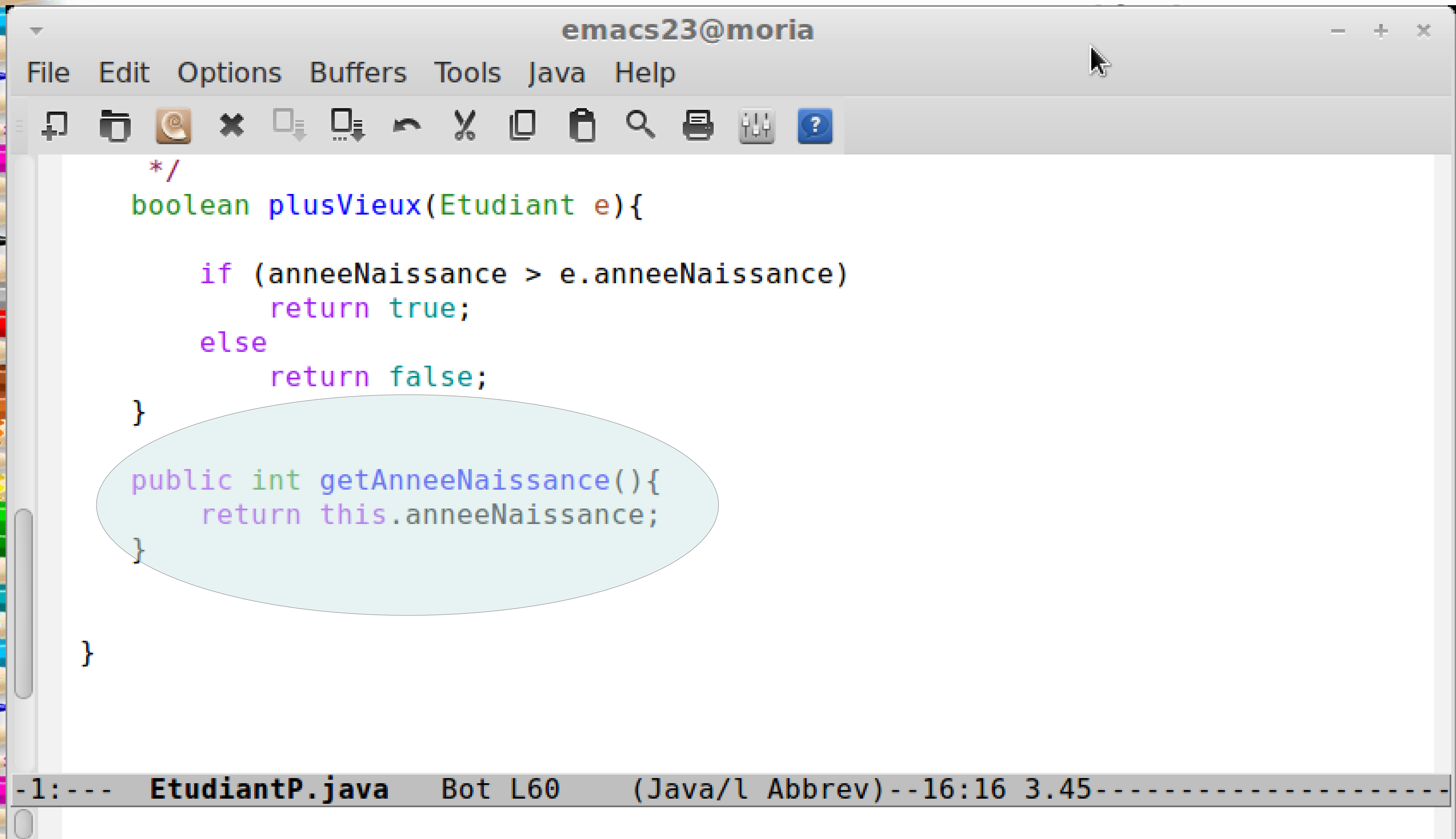
fppt.com

# Intérêt

- On accède aux données seulement à travers des méthodes définies dans le code de la classe. On obtient ainsi un code robuste
- Cela masque l'implémentation et facilite l'évolution du logiciel
- Les méthodes qui accèdent aux attributs sont appelées accesseurs
- Les méthodes qui modifient les attributs sont appelées modificateurs

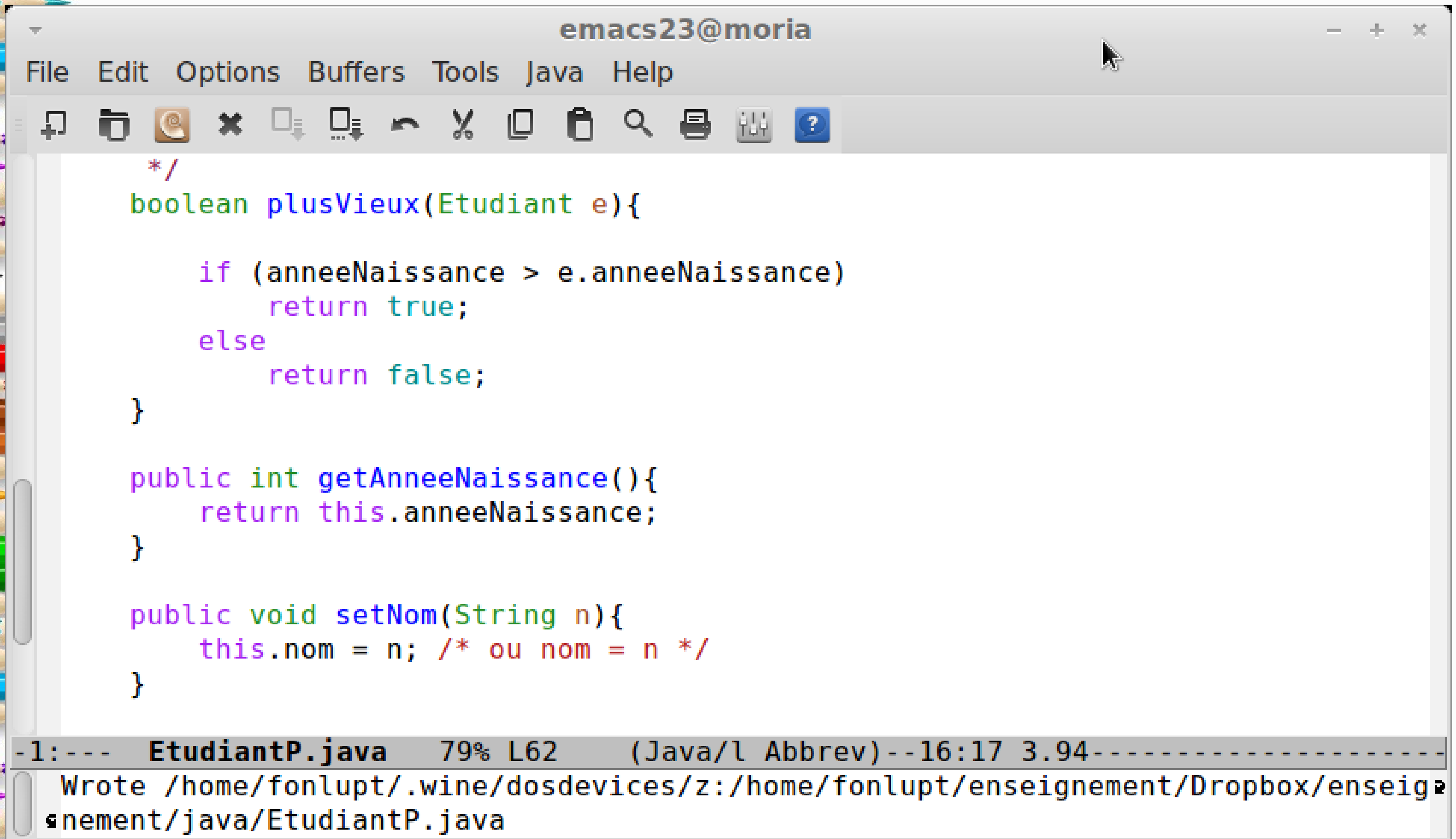
fppt.com

# Accesseurs




fppt.com

# Modificateurs




fppt.com

# Rappels

emacs@CYRIL-ALIEN

File Edit Options Buffers Tools Java Help

```java
public class Etudiant {

    /* on commence par définir les attributs de notre classe Etudiant */
    String nom;
    String prenom;
    int anneeNaissance;
    int nbNotes;
    int [] notes;


    public Etudiant(String n, String p, int a){

        System.out.println("Création de l'étudiant " + p + " " + n);
        nom = n;
        prenom = p;
        anneeNaissance = a;
        nbNotes = 0;
        notes = new  int[20];
    }

    /**
     * rajoute une note dans la liste de notes de l'étudiant
     *
     * @param n note à rajouter
     *
     */
    void ajouterNote(int n){

        System.out.println("Ajout d'une note à l'étudiant " + prenom + " " + nom
);

        notes[nbNotes] = n;
        nbNotes++;
    }

    /**
     * teste si un étudiant est plus vieux que l'étudiant courant
     *
     * @param e étudiant
```

-1(Unix)**- **Etudiant.java** Top L10 (Java/l Abbrev)

classe

attributs

constructeur

méthodes

- Inutile d'aller plus loin
  - Quelle est la différence entre une classe et une instance ?
  - Qu'est-ce qu'un constructeur ?
  - Qu'est-ce qu'une méthode ?
  - Un constructeur est-il une méthode ?
  - Qu'est-ce qu'un attribut ?
- Construire une classe Voiture
  - Année (millésime)
  - Marque
  - Couleur
  - Neuve (booléen)

fppt.com

# Rappels

```java
/**
 * classe Montre
 */

public class Montre {

    // attributs
    private int heures;
    private int minutes;
    private int secondes;

    // Méthodes
    /**
     * méthode de mise à l'heure
     *@param h heure à mettre
     */
    public void setHeures(int h){
        heures = h;
    }
```

Nom de la classe

attributs

Minutes

fppt.com

# Rappels

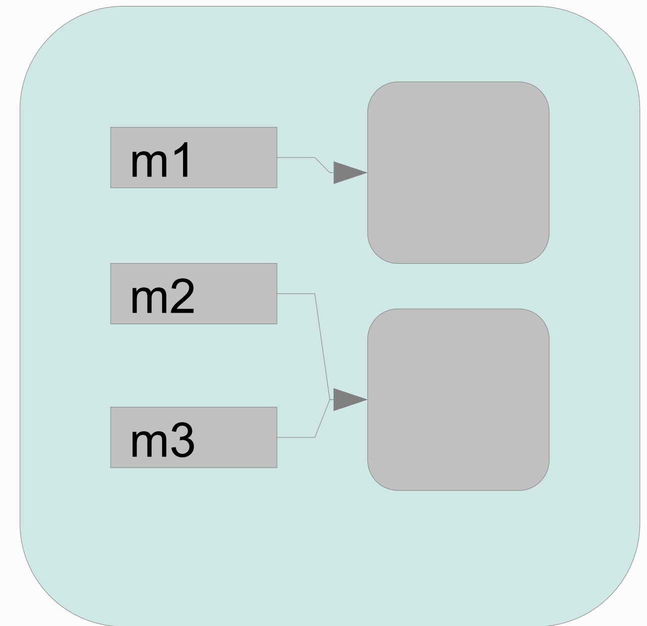
- les attributs déclarés comme privés (**private**) sont inaccessibles depuis l'extérieur de la classe
- principe d'encapsulation

```
Montre m1 = new Montre() ;
Montre m2 =  new Montre() ;
m1.heures = 10 ;
m2.secondes = 55 ;
```

fppt.com

# Rappels

```
Montre m1 ;
m1 = new Montre() ;
Montre m2 = new Montre() ;
Montre m3 = m2 ;
```




égalité de référence

```
m1 == m2
m2 == m3
```

toujours faux
toujours vrai

```
m1.egale(m2)
```

pour tester l'égalité de 2 objets
il faut écrire une méthode

fppt.com

# Rappels : constructeurs

- Constructeurs d'une classe :
  - méthodes particulières pour la création d'objets de cette classe
  - méthodes dont le nom est <u>obligatoirement</u> identique au nom de la classe **sans** valeur de retour
- Rôle d'un constructeur
  - effectuer certaines initialisations de manière « automatique » à partir de paramètres quand un nouvel objet est créé
- Toute classe JAVA possède au moins un constructeur implicite
  - si une classe ne définit pas explicitement de constructeur, un constructeur par défaut sans arguments et qui n'effectue aucune initialisation particulière est invoqué.

fppt.com

# Rappels : constructeur explicite

```
/**
 * classe Montre
 */

public class Montre {

    // attributs
    private int heures;
    private int minutes;
    private int secondes;

    // Constructeur explicite

    public Montre(int h, int m){
        this.heures = h; // ou  heures = h
        this.minutes = m; // ou minutes = m
    }
```

Montre m1 = new Montre(16,55) ;

fppt.com

# Rappels : constructeurs

- Il est possible de définir plusieurs constructeurs

```
/**
 * classe Montre
 */

public class Montre {

    // attributs
    private int heures;
    private int minutes;
    private int secondes;

    // Constructeur explicite

    public Montre(int h, int m){
        this.heures = h; // ou  heures = h
        this.minutes = m; // ou minutes = m
    }

    public Montre(){
        this.heures = this.minutes = 0;
    }
```

chaque constructeur possède le même nom (nom de la classe)
le compilateur distingue les constructeurs en fonction du nombre, du type des arguments
On parle de **SURCHARGE**

fppt.com

- La surcharge n'est pas limitée aux constructeurs, elle est possible pour n'importe quelle méthode
- Il est donc possible de définir des méthodes possédant le même nom mais dont les arguments diffèrent
- Lorsqu'une méthode surchargée est invoquée
  - le compilateur sélectionne automatiquement la méthode dont le nombre et le type des arguments correspondent au nombre et au type des paramètres passés dans l'appel de la méthode
- Des méthodes surchargées peuvent avoir des types de retour différents
  –

fppt.com

# Surcharge des méthodes

```java
    private int heures;
    private int minutes;
    private int secondes;

    // Constructeur explicite

    public Montre(int h, int m){
        this.heures = h; // ou  heures = h
        this.minutes = m; // ou minutes = m
    }

    public Montre(){
        this.heures = this.minutes = 0;
    }

    // Méthodes
    public int nbSecondes(){
        return 3600*heures + 60*minutes + secondes;
    }

    public int nbSecondes(Montre m){
        int s;

        s = 3600*heures + 60*minutes + secondes - 3600*m.getHeures() - 60*m.getM
inutes() - m.getSecondes();
        return s;
    }
```

```java
Montre m1 =
new Montre(10,50) ;
Montre m2 =
new Montre(12,55) ;
System.out.println(
m1.nbSecondes() );
System.out.println(
m1.nbSecondes(m2)) ;
```

fppt.com

# Attributs de classe

- Comment comparer deux heures approximativement (à x secondes près) dans une montre ?
- Ajouter une variable (un attribut) proche à la classe avec une méthode « accesseur » et une méthode « modifieur »
- cf classe Montre.java

```
public boolean EstEgale(Montre m){
    int s1, s2;

    s1 = nbSecondes();
    s2 = m.nbSecondes();

    if (Math.abs(s1 - s2) < proche)
        return true;
    else
        return false;
}
```

fppt.com

# Variables de classe

m1

| heures | 10 |
|---|---|
| minutes | 15 |
| secondes | 25 |
| proche | 5 |

Montre m1 = new Montre(10,15,25)

Montre m2 = new Montre(10,15,23) ;
m1.setProche(5) ;
m2.setProche(1) ;

m2

| heures | 10 |
|---|---|
| minutes | 15 |
| secondes | 23 |
| proche | 1 |

Que donnent m1.estEgale(m2)
et m2.estEgale(m1) ?

Comment représenter proche en dehors de la classe Montre ?

fppt.com

# Variables de classe

- Pas de variable globale en java et rien ne peut être défini en dehors des classes
- En utilisant le mot clef **static** on peut définir des variables de classe qui seront valables et identiques pour toutes les instances de la classe

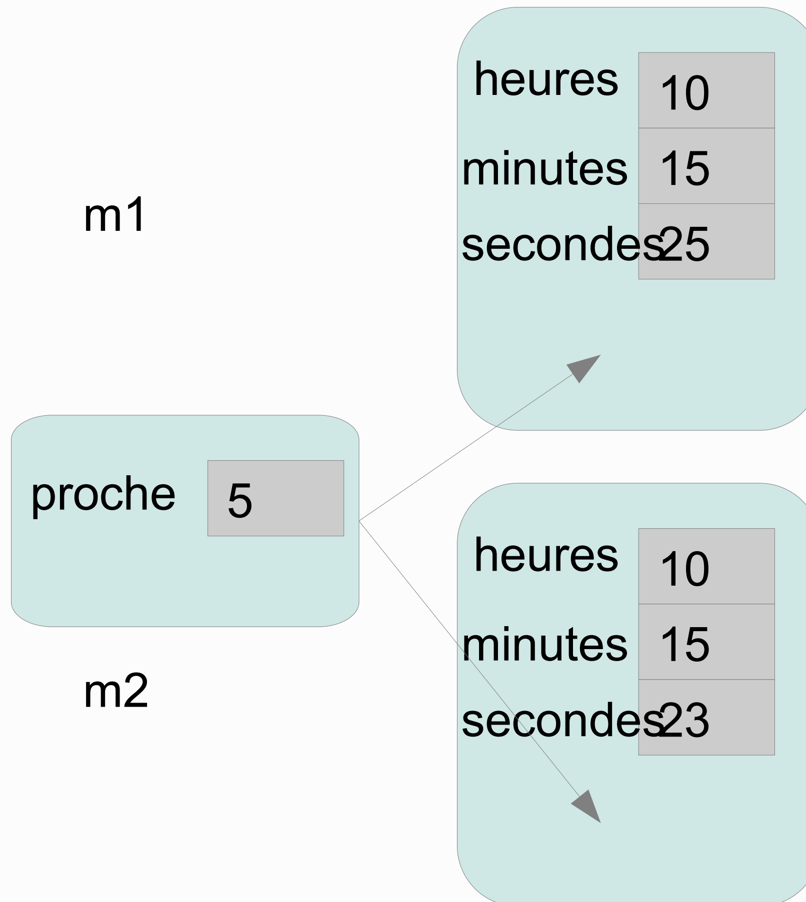
```
private static int proche = 5 ;
```

C'est ici une attribut de classe et non pas un attribut d'instance

**cf** transparent suivant

fppt.com

# Variables de classe




Montre m1 = new Montre(10,15,25)

Montre m2 = new Montre(10,15,23) ;

m1.estEgale(m2)
et m2.estEgale(m1) donnent
la même chose

Comment représenter proche en dehors de la classe Montre ?

fppt.com

# Accès aux variables de classe

- Pour accéder aux attributs d'un objet, la syntaxe utilisée est la suivante : InstanceObjet.Attribut (ex m1.heures)
- Pour accéder aux variables de classe, on utilise la syntaxe suivante : Classe.nomVariable (ex Montre.proche)

Montre m1 = new Montre() ;

m1.proche erreur proche n'est pas un attribut de l'objet m1

**mais** de la classe Montre

Il faut écrire Montre.proche

fppt.com

# Définition de méthodes de classes

- Il s'agit de méthodes statiques pour l'ensemble de la classe

```
/**
 * classe Montre
 */

public class Montre {

    // attributs d'instance
    private int heures;
    private int minutes;
    private int secondes;
    private static int proche = 5; // attribut de classe

    // Constructeur explicite

    public Montre(int h, int m){
        this.heures = h; // ou  heures = h
        this.minutes = m; // ou minutes = m
    }

    public Montre(){
        this.heures = this.minutes = 0;
    }

    // mise à jour de la variable de classe proche
    public static setProche(int p){
        proche = p;
    }

    // Méthodes
    public int nbSecondes(){
        return 3600*heures + 60*minutes + secondes;
```

fppt.com

- Comment utiliser notre méthode de classe ?

Montre m1 = new Montre( ) ;

~~m1.setProche(2) ;~~ NON

setProche() n’est pas une méthode de l’objet m1 mais de la classe Montre.

Il faut utiliser Montre.setProche(2) ;

fppt.com

# Rappels

```
public class Montre {

    // attributs
    private int heures;
    private int minutes;
    private int secondes;
    private int proche;

    // Constructeur explicite

    public Montre(int h, int m){
        this.heures = h; // ou  heures = h
        this.minutes = m; // ou minutes = m
    }
```

fppt.com

# Constructeurs

- Un constructeur peut être appelé par un autre constructeur

```
// Constructeur explicite

public Montre(int h, int m){
    this.heures = h; // ou  heures = h
    this.minutes = m; // ou minutes = m
}

public Montre(){
    this.heures = this.minutes = 0;
}

// ou

public Montre(){
    this(0,0);
}
```

factorisation du code

fppt.com

# Rappel Méthodes

- 2 types de méthodes en java :
- Méthodes d'instances (les plus nombreuses)
  - Opérations sur les objets :
    - String str = "coucou" ;
    - str.toUpperCase() ; // mise en majuscules
- Méthodes de classe
  - Opérations statiques (méthodes de classe)
    - Math.sin(double a) // méthode sin de la classe Math
    - Math.max(int a, int b) // méthode max de la classe Math
    - Files.delete(c:\temp\toto.txt") // méthode delete
  -

fppt.com

# Appel des méthodes statiques

- Déclaration

```
static <typeRetour> nomMethode ( <liste de paramètres>) {
<corps de la méthode>
}
```

- Appel

```
nomMethode( <liste des paramètres effectifs>)
```

  - <liste des paramètres effectifs>
  - liste d'expressions séparées par des virgules et dont le nombre et le type correspond au nombre et au type des paramètres de la méthode

| déclaration | Appel |
|---|---|
| static void sauteLignes(int a) {<br>} | sauteLignes(10) ; |
| static int abs(int a){<br>} | n = abs(-5) ;<br>b = abs(x) ; |
| static int [ ] fibonacci(int n){<br>} | int [ ] t = new int[10] ;<br>t = fibonacci(6) ; |

fppt.com

# Appel des méthodes

- Toute méthode statique d'une classe peut être invoquée depuis n'importe quelle autre méthode statique de la classe
  - l'ordre de déclaration des méthodes n'a pas d'importance
  - pour invoquer une méthode d'une autre classe, il faut la préfixer par le nom de la classe où elle est déclarée
  - ex :
    - Math.random()   Arrays.fill(t, 17)
  - Il est donc possible d'avoir des méthodes de nom identique dans des classes différentes

```
public class A {
    static void m1() {
        ...
    }

    static void m2() {
        m1();
        m3();
    }

    static void m3() {
        ...
    }
}
```

Nom des classes où les méthodes sont déclarées

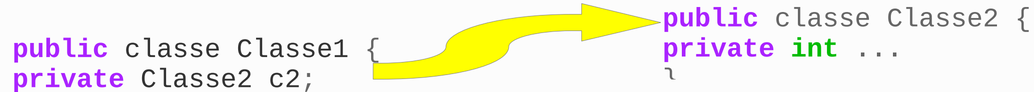
fppt.com

- En java, les méthodes dites « statiques » doivent être vues comme des outils offerts par une classe
- Une méthode statique ne s’applique pas sur un objet à la différente des méthodes d'objets
- Les méthodes statiques s'appliquent en dehors des objets
- En UML, les méthodes et attributs statiques sont soulignés
- C'est ce qui se rapproche le plus des fonctions des langages impératifs (C, Ada, ...)

fppt.com

# La délégation en POO

- Il est très commun  qu'un objet d'une classe soit utilisé comme attribut d'une autre classe
- Il s'agit de la délégation

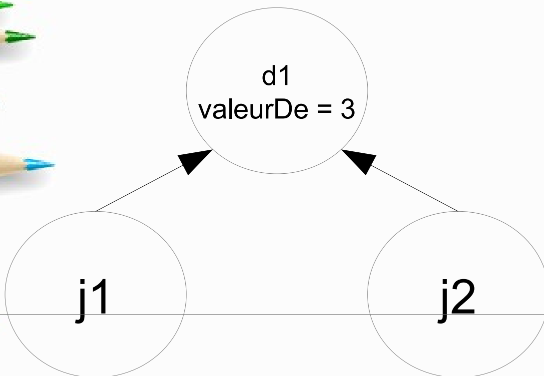



fppt.com

# Subtilités

- Dans cet exemple, le dé a un cycle de vie indépendant
- Il peut être utilisé par plusieurs joueurs
- Attentions aux effets de bords

```
Dé d1 = new Dé() ;
Joueur j1 = new Joueur(d1,"Alain");
Joueur j2 = new Joueur(d1,"Pierre");
```




```
public class Joueur {

    // Dé du joueur
    private Dé monDé;

    // Nom du joueur
    private String nom;

    // Constructeur de joueur
    public Joueur(Dé dé, String n){
        monDé = dé;
        nom = n;
    }
    public void lancer(){

        monDé.lancerDe();

    }
```

d1.lancerDé() affecte j1 et j2 !

fppt.com

# Comment faire ?

- L'objet de la classe Dé est indépendant
- Chaque instance de dé n'est lié qu'à un unique joueur

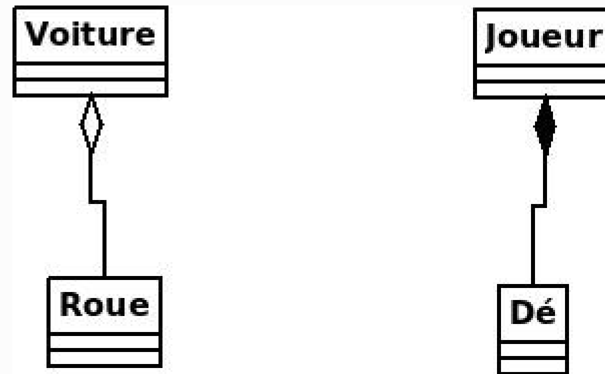
```
// Constructeur de joueur
   public Joueur( String n){
       monDé = new Dé(dé);
       nom = n;
   }
...

// Constructeur de Dé dans Dé
   public De(De d){
       valeurDe = d.getValue();
   }
```

j1.lancer() n'affecte pas le joueur j2
Au niveau du cycle de vie, lorsque le joueur j1 est détruit le dé associé l'est également

fppt.com

# Agrégation / Composition

- Les deux exemples précédents traduisent deux nuances sémantiques de l'association a-un entre le Joueur et le Dé
- UML distingue ces deux sémantiques en définissant deux type de relations




**Agrégation**
L'élément agrégé (Roue) a une existence autonome
Cycle de vie indépendant

**Composition/Agrégation forte**
Le Dé n'existe pas en dehors
Du Joueur

© Philippe Genoud

fppt.com

# Agrégation la classe Poker

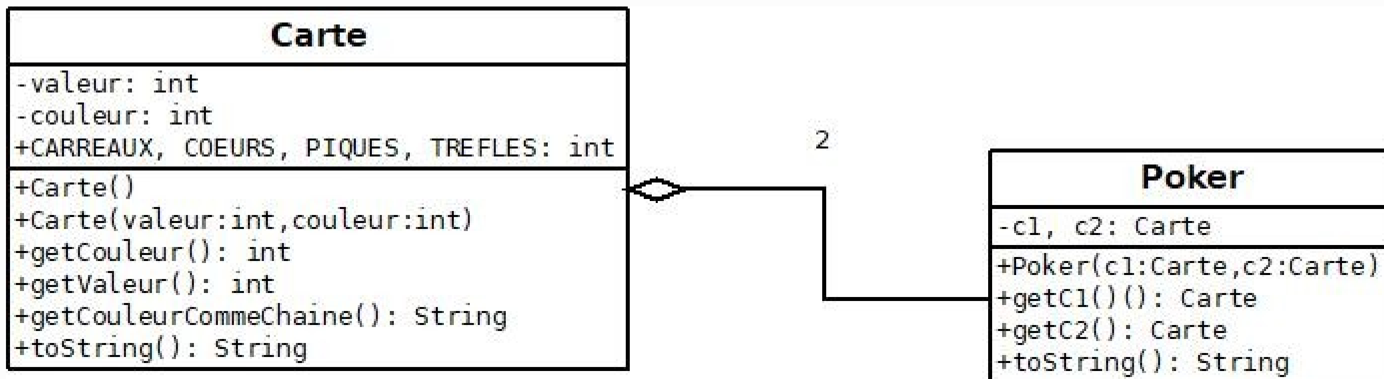
- Agrégation
  - la classe Poker (très simplifiée !) est composée de deux Cartes
  - les opérations d'affichage des cartes sont déléguées à la classe Carte

```
public class Poker {

    private Carte c1, c2;

    public Poker(Carte c1, Carte c2){
        this.c1 = c1;
        this.c2 = c2;
    }

    public Carte getC1(){ return c1; }

    public Carte getC2() { return c2; }

    @Override
    public String toString(){
        return c1 + " et " +  c2;
    }

}
```

fppt.com

# Agrégation représentation

- Pour matérialiser l'agrégation, UML utilise une flèche avec un losange vide côté agrégat (appartenance faible)
- Étiquetée par la multiplicité : * indiquant un tableau




fppt.com

# Objets, références et égalité

- Les instances créées sont passées **par référence** :
  - new Carte crée une nouvelle instance
  - l'affectation c1 = … stocke une référence sur l'objet dans c1
  - l'appel à Poker(c1, c2) passe une référence sur l'objet au constructeur

    p1.c1 et p2.c1 correspondent au même objet
    → si jamais on modifie c1 dans p1 cela le modifiera dans p2
- Opérateurs == teste si les références sont identiques (attention ce n'est pas du tout un test d'égalité!)
  - → à utiliser avec précaution ou à ne pas utiliser p1.c1 == p2.c1 p2.c3

```
Carte c1 = new Carte(10,2);
Carte c2 = new Carte(8,3);
Carte c3 = new Point(10,2);
Poker p1 = new Poker(c1,c2);
Poker p2 = new Poker(c1,c3);
```

fppt.com

# Que donne ?

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

    Carte c1 = new Carte(10,2);
    Carte c2 = new Carte(8,3);
    Carte c3 = new Carte(10,2);

    Poker p1 = new Poker(c1,c2);
    Poker p2 = new Poker(c1,c3);

    c1.incrementeValeur();
    System.out.println(p1);
    System.out.println(p2);


}
```

fppt.com

# Attributs immutables : final

- Le mot-clef final indique qu'un attribut est immutable (constant)
  - Il est souvent initialisé dans le constructeur
  - Il ne peut jamais être modifié ensuite
  - On pourrait mettre ici les attributs de toutes nos cartes à final
- Si jamais une variable ayant la propriété final est modifiée erreur de compilation

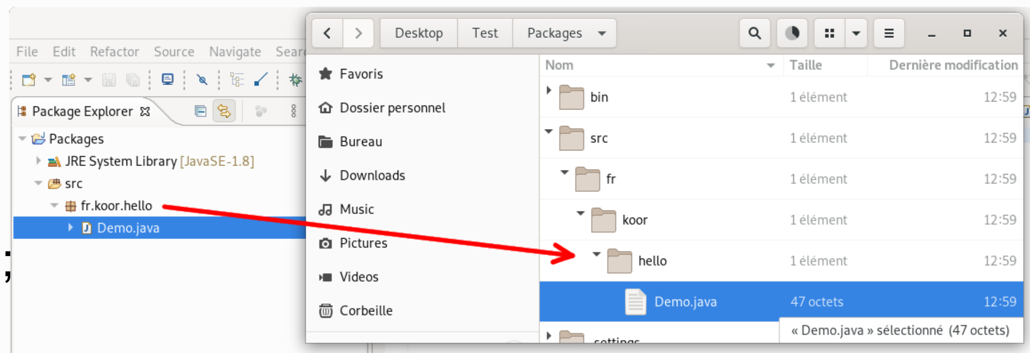
```
public class Carte {

    /* constantes et attributs */

    public final static int PIQUES = 0,       // Codes pour les 4 couleurs
                            COEURS = 1,
                            CARREAUX = 2,
                            TREFLES = 3;


    private final  int couleur;   //    PIQUES, COEURS, CARREAUX, TREFLES.


    private final  int valeur;  // valeur de la carte
```

fppt.com

# package et import

- Les packages et les importations correspondent à la manière dont Java organise ses fichiers (attention un peu complexe)
- Les packages regroupent plusieurs classes (ex java.util)
- Notre classe Carte
  - Appartient à un package (package par défaut du répertoire courant sinon)
  - Peut être appelé par toutes les classes du même package (ex Poker)
  - En cas d'arborescence, on doit indiquer tout le chemin complet

package fr.koor.hello ;




fppt.com

# package et import

- Pour utiliser une classe à partir d'une autre classe
  - Implicite si la classe est dans le même package (même répertoire si package non précisé)
  - Sinon utilisation de import
    - Import chemin complet
    - Ici au début de Test.java
      package com.test ;
    - Si le fichier Prod.java est dans le répertoire src de com on aura package com.src ;
    - Pour utiliser la classe Test dans la classe Prod, il faudra utiliser l'import suivant :
      import com.test ; ou import com.test.Test ;

```
└── com
  └── test
    └── Test.java
```

fppt.com

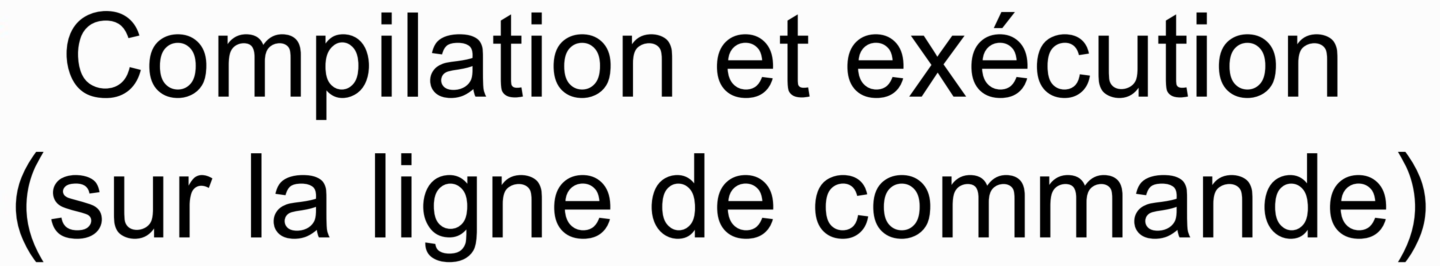
# Notre point d'entrée main

- Un programme Java est généralement constitué de nombreuses classes
- Le point d'entrée dans le programme est la méthode main d'une classe (il peut y avoir plusieurs main)

  public static void main(String[] args)
  - public  indique que la méthode est visible
  - static car elle n'est pas liée à la classe
  - String [ ] args : tableau comprenant la liste des arguments

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

    Carte c1 = new Carte(10,2);
    Carte c2 = new Carte(8,3);
    Carte c3 = new Carte(10,2);
}
```

fppt.com



# Compilation et exécution (sur la ligne de commande)

- Compilation : javac Test.java en Test.class (byte-code)
- Exécution java Test (pas d'extension derrière)
- Test doit contenir une méthode main
- Sur un IDE comme IntelljIDEA ces deux phases sont cachées

En utilisant les packages

- Compilation javac ing1/exercices/Programme1.java
  Génère le fichier Programme1.class si package exercice ;
- Exécution java ing1.exercices.Programme

fppt.com

# Les tableaux en java

fppt.com

# Introduction

- Principe assez proche du C++
- Plus faciles à manipuler qu'en C
- Les tableaux peuvent être des ensembles ordonnées de :
  - types de bases (int, char, …)
  - d'objets !
- Note : en Java, les tableaux sont des objets

fppt.com

# Déclaration

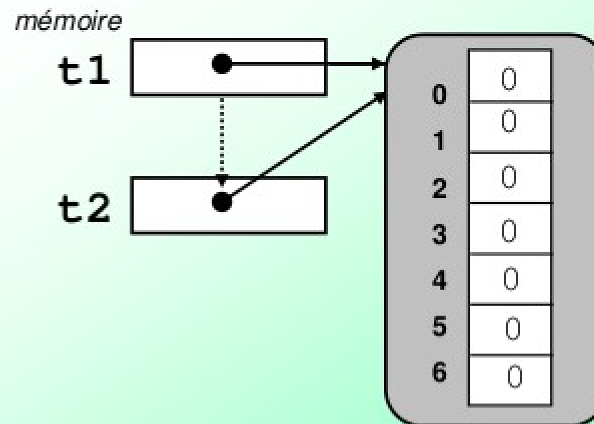
- syntaxe : type [ ] nomDuTableau ;
- int [ ] tableau ;
  - la déclaration ne fait que déclarer le tableau mais ne crée pas le tableau en mémoire. La variable déclarée permet de référence tout tableau de même type.
- L'opérateur new permet de créer le tableau
  - int [] tableau ;
    tableau = new int[50] ;
- La taille donnée est fixe, elle ne peut plus être modifiée

fppt.com

# Visualisation

- La création d'un tableau par **`new`**
  - *alloue la mémoire nécessaire en fonction*
    - *du type du tableau*
    - *de la taille spécifiée*
  - *initialise le contenu du tableau*
    - *type simple : 0*
    - *type complexe (classe) :* **`null`**

```
int[] t1;
t1 = new int[7];
int[] t2 = t1;
```




fppt.com

# Accès aux éléments

- principe : tableau[*expression*]
  - Comme en C, les éléments d'un tableau de taille n sont numérotés de 0 à n-1
  - *expression* est une expression donnant une valeur entière bien sûr !
  - Java vérifie dynamiquement que la valeur de l'élément existe sinon
    - erreur : ArrayIndexOutOfBoundsException
    - beaucoup mieux que le C

fppt.com

# Attribut du tableau

- Il y a des attributs puisque c'est un objet !
- tableau.length donne la taille du tableau
  - int [ ] monTableau = new int[10] ;
  - monTableau.length → 10 taille du tableau
  - monTableau.length – 1 → indice max du tableau
- cf *Cours8.java* utilisation du tableau args[ ]

fppt.com

# Avantage du tableau
# Jeu du Yahtzee

**ancienne version :**

```
public class Yahtzee{
   int d1 , d2 , d3 , d4 , d5 ; //les 5 dés
   ...
}
```

**nouvelle version :**

```
public class Yahtzee{
   int[] dice = new int[5];
   ...
}
```

fppt.com

# Avantage du tableau

- Le code est plus compact et plus adaptable (marche avec un nombre quelconque de dés)

ancienne version :

```
d1 = getRandomDieValue();
d2 = getRandomDieValue();
d3 = getRandomDieValue();
d4 = getRandomDieValue();
d5 = getRandomDieValue();
}
```

nouvelle version :

```
public static void rollDice(){
  for(int i=0; i < dice.length; i++){
    dice[i] = getRandomDieValue();
  }
}
```

fppt.com

```
public static int getRandomDieValue(){
  return Random.nextInt(6) + 1 ;
}
```

Cette Méthode retourne une valeur entre 1 et 6.

Nous reverrons les méthodes ultérieurement

fppt.com

# Refactorisation du Yahtzee les constantes

- Refactoriser signifie simplifier le programme sans changer son comportement ;
- Exemple:

CONSTANTE

```
class Yahtzee{
   public static final int NUMBRE_DE_DES = 5;
   int[] dice = new int[NOMBRE_DE_DES];
   ...
}
```

- 5 est un **nombre magique**. Il ne faut pas créer un tableau int[] dice = new int[5]
- Cela rend les changements difficiles si on utilise 10 dés

fppt.com

Utilisation d'une boucle basée sur NOMBRE_DE_DES

```
for(int i = 0; i < NOMBRE_DE_DES; i++){
  dice[i] = getRandomDieValue();
}
```

- Le code est plus général et fonctionne quel que soit le nombre de dés
- Le code est plus lisible

fppt.com

# Autres possibilités de créer des tableaux

- comme en C, on peut donner explicitement la liste de ses éléments à la déclaration (liste de valeurs entre accolades)
  - int [ ] t = {17, 8, 9, 23, 17} ;
  - char [ ] voyelles = { 'a', 'e', 'i', 'o', 'u', 'y' } ;
- L'allocation mémoire (équivalent de new) est réalisée implicitement

fppt.com

- Un tableau est une suite d'éléments du même **type**
- Le mot-clef **new** est utilisé pour allouer de l'espace mémoire
- Lors de l'allocation mémoire, il est nécessaire de préciser la taille du tableau
- Une fois que le tableau est créé, **sa taille ne peut être modifiée**

fppt.com

# Tableaux et boucles

- affecter un tableau aux valeurs : {2,4,...,100}
- nous pouvons affecter directement chaque élément, mais cela représente trop de travail
- on utilise plutôt une boucle !

```
int[] a = new int[500];
for(int i=2, j=0; i<=1000; i+=2, j++){
  a[j] = i;
}
```

- Les boucles sont très pratiques pour le parcours dans un tableau

fppt.com

# La classe Arrays

- Java dispose d'une classe Arrays qui propose des méthodes pour trier et rechercher des éléments dans des tableaux
- import java.util.Arrays ; pour l'utiliser
- cf *Cours9.java* que fait ce code ?
- Exercice :

A partir d'un tableau trié de 100 éléments (cf Cours9.java) demander à l'utilisateur de rentrer une valeur entre 0 et 1000 et afficher si cette valeur est trouvée dans le tableau et si oui à quelle position

- cf Cours10.java recherche dans un tableau

fppt.com

# Retour sur la classe String

- Comme indiqué précédemment, les chaînes de caractères sont des instances de la classe String
- Ce sont des objets immutables (impossible à modifier) → constantes
- Le mot-clef new est facultatif pour créer une nouvelle chaîne String t = « toto »
- Rappel : == ne vérifie que l'égalité en mémoire (≠ Python)
  - On utilise chaine1.equals(chaine2)
- Concaténation : opérateur +
- Taille d'une chaîne chaine.length()
- Affichage System.out.println(chaine)

fppt.com

```
public class Test {

    public static void main(String [] args){

        String t1 = "toto";
        String t2 = "toto";

        System.out.println(t1 == t2);
        System.out.println(t1.equals(t2));

        t1 = t1 + "1";
        System.out.println(t1);
    }


}
```

fppt.com

- Les constantes sont écrites en MAJUSCULES (mot-clef final)
  public static final PI = 3,14159 ;
- Les classes commencent pas une majuscule unique
  public class Complexe
- Les attributs et méthodes commencent par une minuscule. Si ces objets comportent plusieurs mots, les mots suivants commenceront pas une majuscule
  int valeur = 2 ;
  int maValeurPréférée = 7 ;
  public String metEnMajuscule(String s)

fppt.com

- Par convention en Java, les constantes seront écrites en MAJUSCULES
- la déclaration s'effectue en début de classe :

  public static final int MAX = 50 ;
- public indique que la constante peut être utilisée en dehors de la classe
- static indique que c'est une variable de classe
- final que la valeur ne peut jamais être modfiée
- utilisation :

  MAX dans la classe           Classe.MAX à l'extérieur de la classe
- 
- valeur de $\pi$ → Math.PI constante définie dans la classe Math

fppt.com

- La méthode toString est implicite lorsque :
  - on affiche un objet System.out.println(object) ;
  - on concatène un objet à une chaîne
    - String s = objet + "123" ;
      est **équivalent à**
    - String s = objet.toString() + "123" ;
- En d'autres mots quand Java s'attend à une chaîne, il convertit automatiquement l'objet en chaîne

fppt.com