Institut Supérieur d'Electronique
et de Communication de Sfax

Programmation Orientée Objet en Java (POO)

Responsable

Tarak CHAARI

tarak.chaari@gmail.com

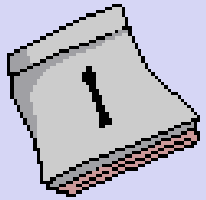
Objectifs de ce cours

- Découverte du Java et de son approche Objet
- Maîtriser la syntaxe du langage
- Savoir utiliser les classes de base et les objets

Contenu

1. C'est quoi Java?
2. Les bases du langage
3. Concept Objet du langage Java
4. Classes et Objets en Java
5. Programmation orientée objet avancée Java
6. Les classes de bases
7. Exercices et corrigés

Chapitre



C'est quoi Java ?

1- C'est quoi Java

1-1 Historique du langage Java (1/2)

- Idée originale de James Gosling
- L'objectif était de commander sa machine à café à distance (pour trouver un café chaud lors de sa rentrée à la maison)
- Il a voulu développer cette commande en C
- Après plusieurs mois: il a trouvé plusieurs difficultés dû à la complexité du langage C et à sa dépendance du système d'exploitation

  => il faut un langage plus simple et mieux adapté à la communication sur internet
- Cette idée a été reprise par SUN MicroSystems après avoir recruté James Gosling

1- C'est quoi Java

1-1 Historique du langage Java (2/2)

- **En 1990: première version du langage Java sous la direction de Bill Joy et James Gosling.**
- **En 1993, avec l'intérêt grandissant d'Internet, ce langage, se métamorphose en langage dédié à Internet :**
  - SUN diffuse le premier browser HotJava qui permet d'exécuter des programmes Java encapsulés dans des pages WEB (i.e. des applets Java) de plus en plus «vivantes ».
- **1996 : Les Java Développement Kits (JDK) ou (J2SDK) sont disponibles gratuitement pour la plupart des machines du marché.**
- **2009: Java a été racheté par Oracle**

1- C'est quoi Java

1-2 Caractérisation du langage Java (1/8)

- **Le langage Java est familier :**
  - Java est un langage familier très proche du langage C, C++. Par exemple :
    - Même types de base que C++ (int, float, double, etc.),
      - Même formes de déclarations que C++,
      - Même structure de contrôle que C++ (if, while, for, etc.).

- **Le langage Java est simple :**
  - Java est un langage simple par rapport au langage C et C++.
    - Il n'y a plus de pointeurs et des manipulations les concernant ;
    - Java se charge (presque) de restituer au système les zones mémoire inaccessibles et ce sans l'intervention du programmeur.

1- C'est quoi Java

1-2 Caractérisation du langage Java (2/8)

- **Le langage Java est orienté objet :**
  - Paquetage pour la réutilisation :
    - java.lang : classes de base
    - java.awt : interfaces graphiques
    - java.net : communication réseaux (socket et URL)
    - java.applet : API Applet
- **Le langage Java est distribué :**
  - Supporte des applications réseaux (protocoles de communication java.net)
    - URL : permet l 'accès à des objets distants
    - RMI : Remote Method Invocation

1- C'est quoi Java

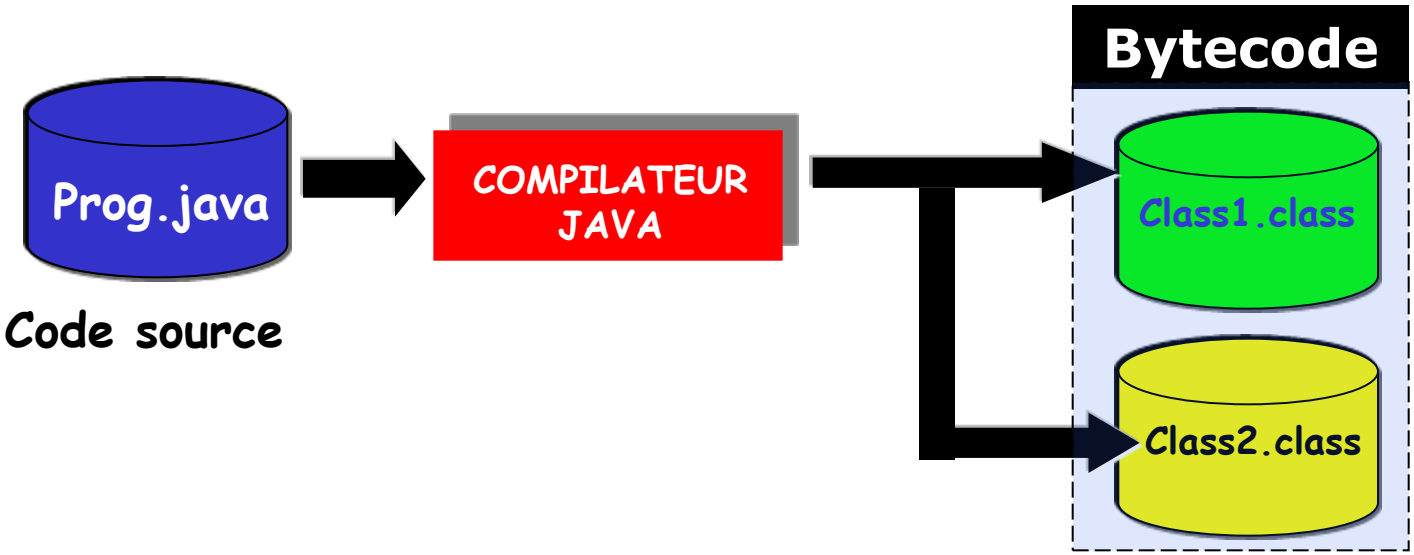
1-2 Caractérisation du langage Java (3/8)

- **Le langage Java est un langage intermédiaire:**
  - Un programme Java n'est pas compilé en code machine ;
    - Il sera compilé en code intermédiaire interprété nommé ByteCode.
    - Lors de l'exécution le ByteCode sera interprété à l'aide d'une machine dite virtuelle JVM (Java Virtual Machine).
- **Le langage Java est portable et indépendant des plates-formes :**
  - Le code intermédiaire produit «ByteCode » est indépendant des plates-formes.
    - Il pourra être exécuté sur tous types de machines et systèmes pour peu qu'ils possèdent l'interpréteur de code Java « JVM ».

1- C'est quoi Java

1-2 Caractérisation du langage Java (4/8)

Production d'une application Java standard

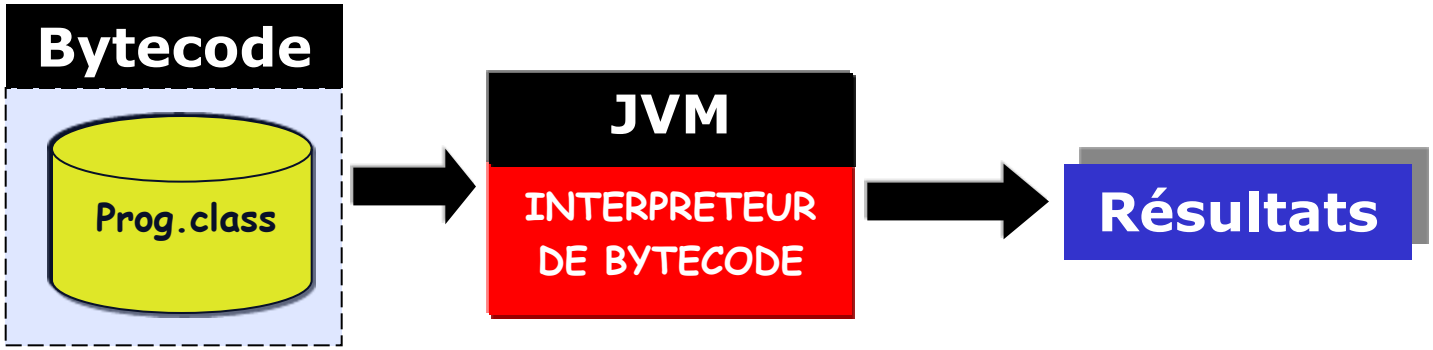


Bytecode = Pseudo code machine qui contrairement à du code binaire natif, n'est pas exécuté directement par le processeur – ce qui a pour conséquence de ralentir son exécution.

1- C'est quoi Java

1-2 Caractérisation du langage Java (5/8)

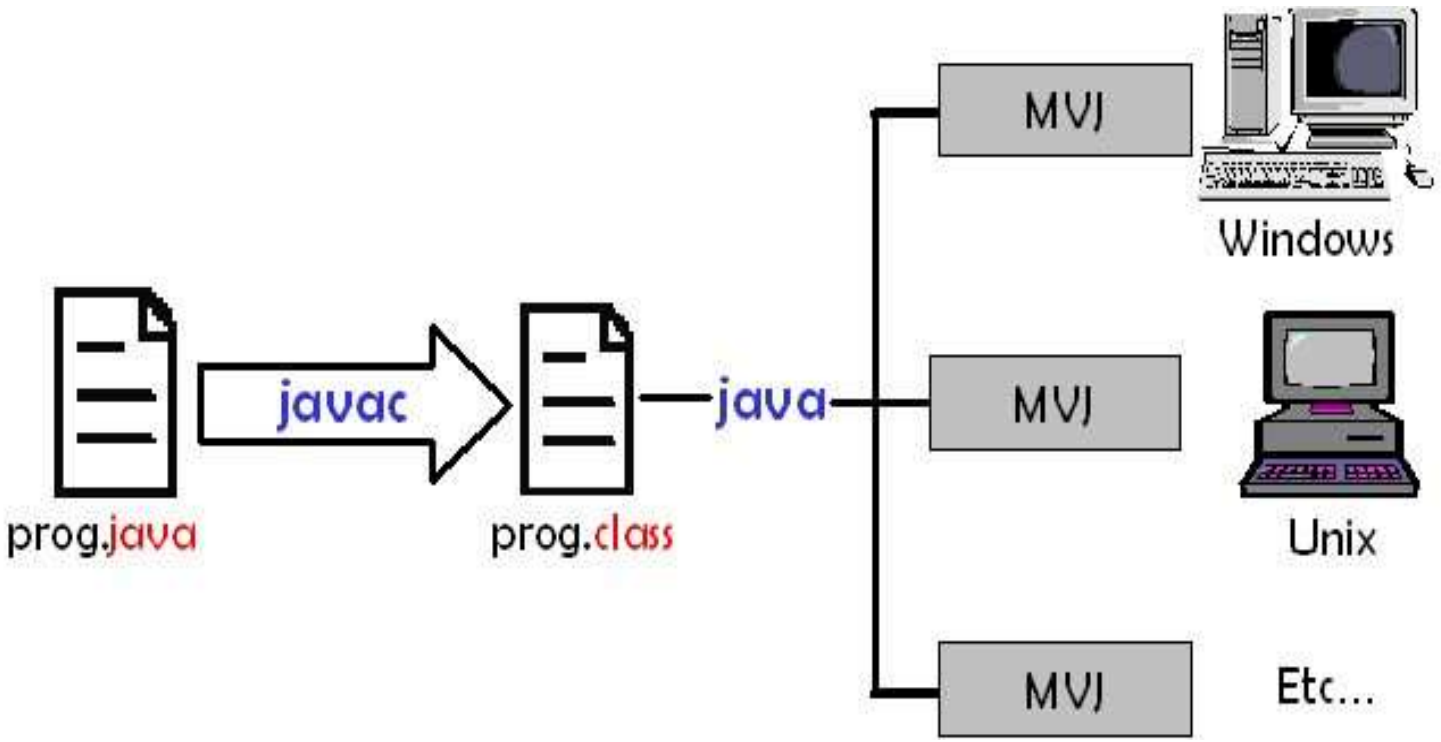
Exécution d'une application Java standard



JVM = Programme capable d'interpréter les instructions contenues dans les fichiers ByteCode Java afin de les exécuter.

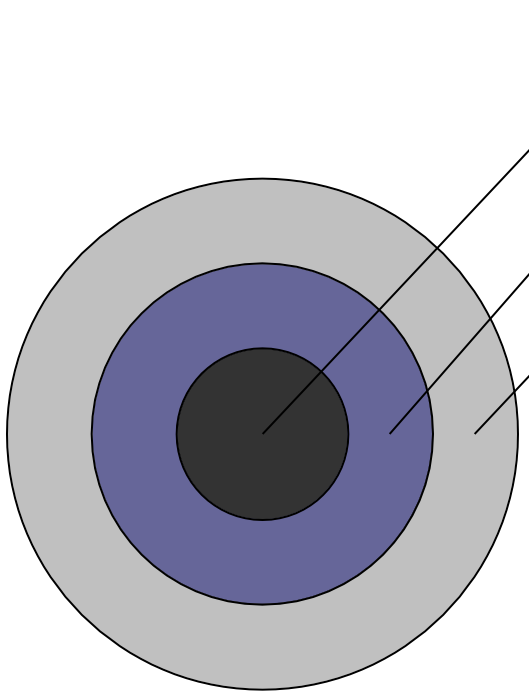
1- C'est quoi Java

1-2 Caractérisation du langage Java (6/8)



1- C'est quoi Java

1-2 Caractérisation du langage Java (7/8)



Système d'exploitation

MVJ

Code Java

1-2 Caractérisation du langage Java (8/8)

- **Le langage Java est robuste et sûr :**
  - Détection des erreurs d'invocation des méthodes :
    - java est fortement typé
    - seules des conversions sûres sont automatiques
- **Fiabilité de la gestion de la mémoire**
  - pas de pointeurs
  - ramasse miettes automatique
  - contrôle automatique d'accès aux tableaux et aux chaînes de caractères.
- **Le langage Java est multithread :**
  - JAVA permet l'exécution simultanée de plusieurs processus légers (thread)
    - Classe java.lang.thread avec les méthodes permettant de :
      - Démarrer, Exécuter, Stopper ces processus.
    - contrôler les synchronisations et l'état cohérent des données.

1- C'est quoi Java

1-3 Java vs C++

- Un programme Java est une classe.
- Une méthode `main()` avec un type de retour `void`.
  => main(String[] args)
- Une constante est définie au sein d'une classe à l'aide de l'instruction `final`.
- L'instruction `include` est remplacée par l'instruction `import` suivie par un nom complet de la classe.
- Représentation des caractères et des identificateurs sur 16 bits (Unicode).
- Pas de pointeurs, seulement des références.

Fichier «XXX.java »

// Commentaire

Définition d'une classe XXX

Méthode main ( )

- **Une application Java minimale doit contenir une classe :**
  - Portant le même nom que le fichier ayant l'extension « .**java** » dans lequel elle est enregistrée.
  - Comportant (au moins) une méthode :
    - appelée main,
    - de type public et static,
    - ayant un argument de type String[ ],

Programme Java minimal :

Fichier «PremierProgramme.java »

```
// Ce programme se contente d'afficher le message « Ca Marche »
class PremierProgramme
{
  public static void main(String args[] )
  {
        System.out.println("Ca Marche");
  }
}
```

Fichier «PremierProgramme.java »

```
// Ce programme se contente d'afficher le message « Ca Marche »
```

- **La première ligne du programme « PremierProgramme » est une ligne de commentaire :**
  - Elle commence par //.
  - Tout ce qui est compris entre ces deux caractères et la fin de la ligne est ignoré par le compilateur.
  - En revanche, d'autres outils peuvent utiliser le texte contenu dans ce type de commentaire.

1- C'est quoi Java

1-4 Java par l'exemple

```
class PremierProgramme
{
}
```

- **Le mot class veut dire que nous allons définir une nouvelle classe Java, suivi du nom de cette classe.**
- **En Java, les majuscules et les minuscules sont considérés comme des caractères différents.**
  - PremierProgramme n'est pas identique à PREMIERProgramme.
- **Les caractères « { « et « } » marquent le début et la fin du bloc d'instructions à réaliser par la classe.**

1- C'est quoi Java

1-4 Java par l'exemple

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

 }
```

- **main() { : signifie que nous allons définir une méthode appelée main.**
- **Le mot main indique que cette méthode est la méthode principale de la classe.**
- **Un interpréteur Java a pour fonction d'exécuter les instructions contenues dans le bloc d'instruction de la méthode principale main, du programme qu'on lui soumet.**
- **Une méthode est une sorte de procédure (ensemble d'instructions) appartenant à une classe.**

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

 }
```

- le mot **void** signifie que la méthode **main** ne renvoie aucune valeur.
- **args[ ]** est le paramètre d'entrée de type **String** de la méthode **main.**
- les mots **public** et **static** décrivent chacun une caractéristique de la méthode (**voir plus loin**).

```
public static void main(String args[] )
{
      System.out.println("Ca Marche");
 }
```

- **System.out.println** est une commande permettant d'afficher la chaîne de caractère « Ca MARCHE » sur la sortie par défaut de votre machine qui est l'écran.

1-5 Exemple d'une application Java standard

Fichier «PremierProgramme.java »

```
// Ce programme se contente d'afficher le message « Ca Marche »

class PremierProgramme
{
    static void affiche( )
    {
            System.out.println("Ca Marche");
     }

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

1-6 Le Java developper Kit : JDK

- **Le JDK ou J2SDK (Java 2 Software Development Kit):**
  - est l'ensemble des outils nécessaires pour développer et exécuter une application Java.

- **Des environnement intégrés de développement Java sont actuellement :**
  - commercialisés (Visual j++, Symnatec café, etc.)
  - distribués gratuitement à travers le Web (Eclipse).

1- C'est quoi Java

1-7 Outils principaux fournis par le JDK

- **Le compilateur javac:**
  - Compile les fichiers sources java de nom XXX.**java**.
  - Traduit les fichiers sources en ByteCode :
    - Produit autant de fichier .class qu'il y a de classe définie dans le fichier .java.
- **L'interpréteur java:**
  - Prend en paramètre le nom de la classe.
  - Cherche le ou les fichiers .class qui lui correspondent.
  - Appelle la méthode main de la classe.
- **Le documenteur javadoc:**
  - Génère automatiquement une documentation sous la forme de fichiers html à partir des fichiers sources commentés.

1-8 J2RE (Java 2 Runtime Environment)

- **Contient tout ce qui est nécessaire pour diffuser vos applications.**
  - En cas d'une application devant être diffusée au public sur un CD-ROM, vous ne pouvez pas supposer que chaque utilisateur sera équipé d'une JVM.
    - Il est nécessaire d'accompagner chaque application d'une machine virtuelle .
    - C'est à cette fin que *Oracle* met à la disposition de tous, gratuitement le J2RE. Celui-ci contient tous les éléments nécessaires pour faire fonctionner une application Java, par opposition au J2SDK, qui contient tout ce qui est nécessaire pour développer une application.
    - Le J2RE peut être diffusé librement sans payer aucun droit à Oracle.

1- C'est quoi Java

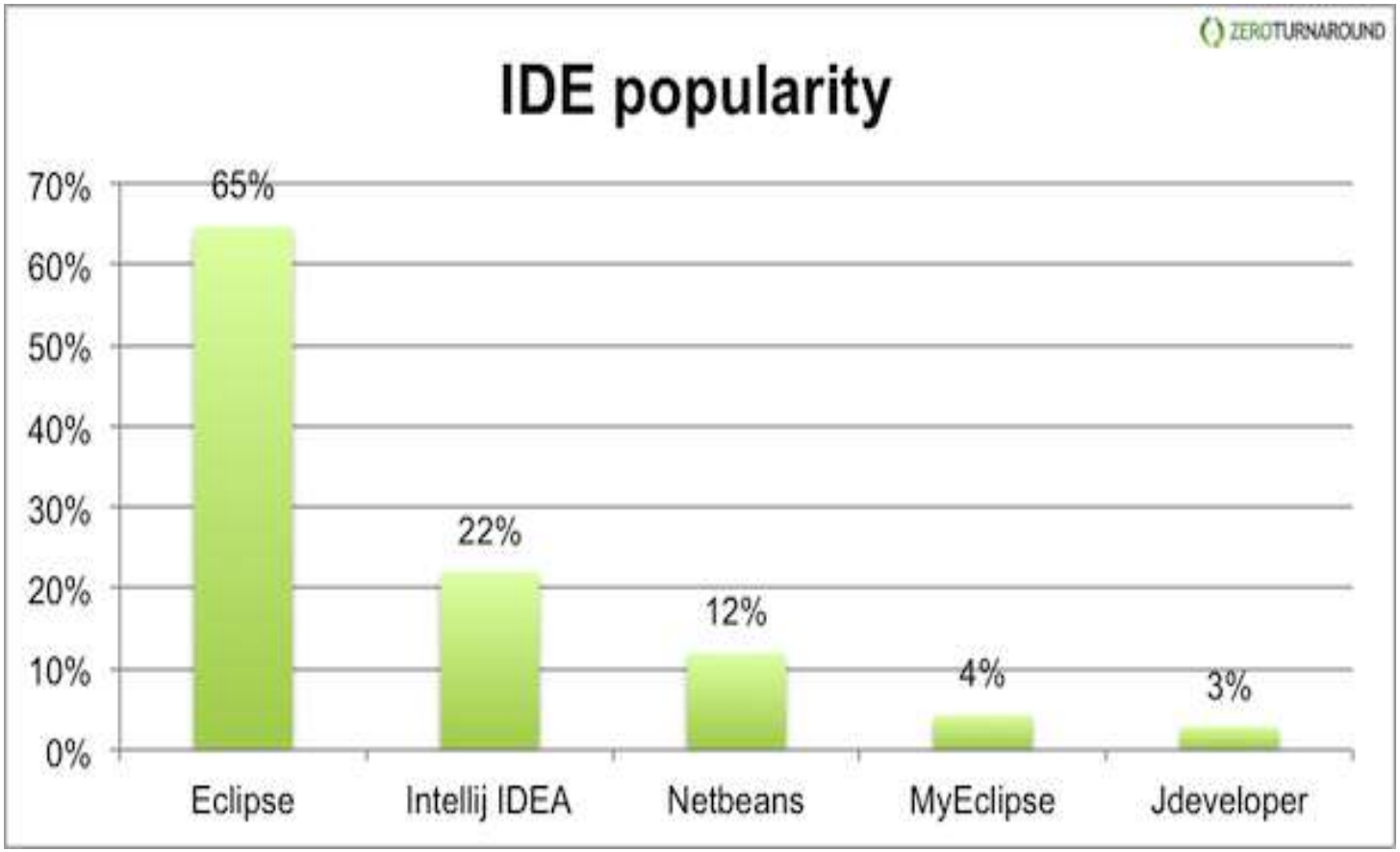
1-9 Ecrire une application Java en ligne de commandes

- **Pré requis**
  - Un éditeur de texte

- **Etapes :**
  - Ecrire le code source du programme **PremierProgramme.java** dans un fichier portant le nom de la classe.
  - Respecter la casse des caractères (majuscules/minuscules).
  - Compiler le code source: **javac PremierProgramme.java**.
  - Un fichier **PremierProgramme.class** contenant du **ByteCode** sera généré.
  - Exécuter le **ByteCode** : **java PremierProgramme**

1-10 Ecrire une application Java avec un IDE

- **Simple éditeurs ou environnement de développement**

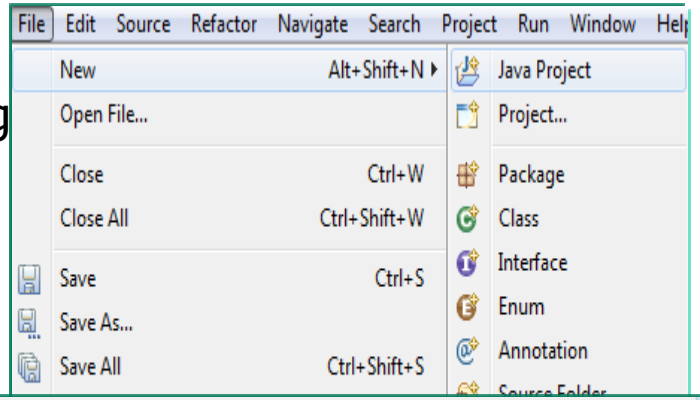


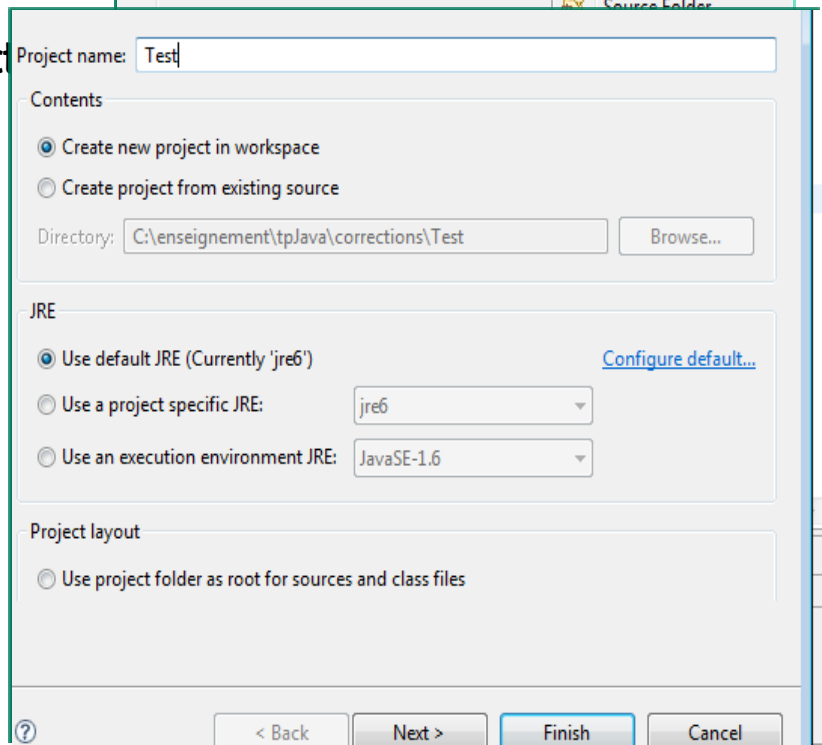
1- C'est quoi Java

1-10 Ecrire une application Java avec un IDE (1/4)

- **Pré requis**
  - ECLIPSE : www.eclipse.org

- **Etapes :**
  - Créer un projet.
    - File → new Java Projec
    - Project name→ "Test "

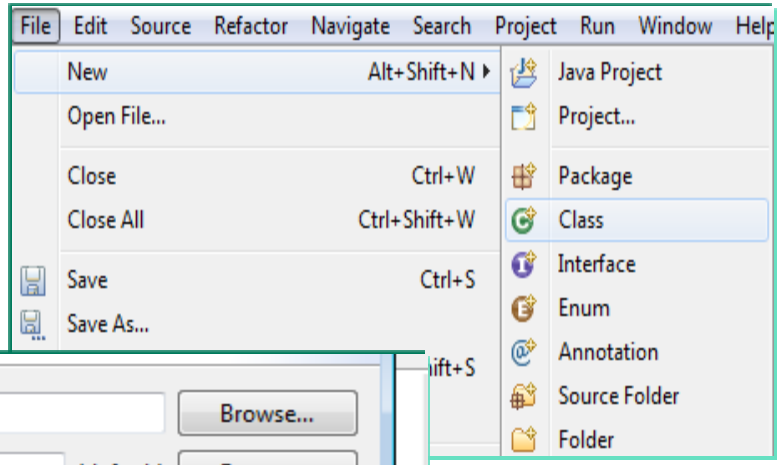


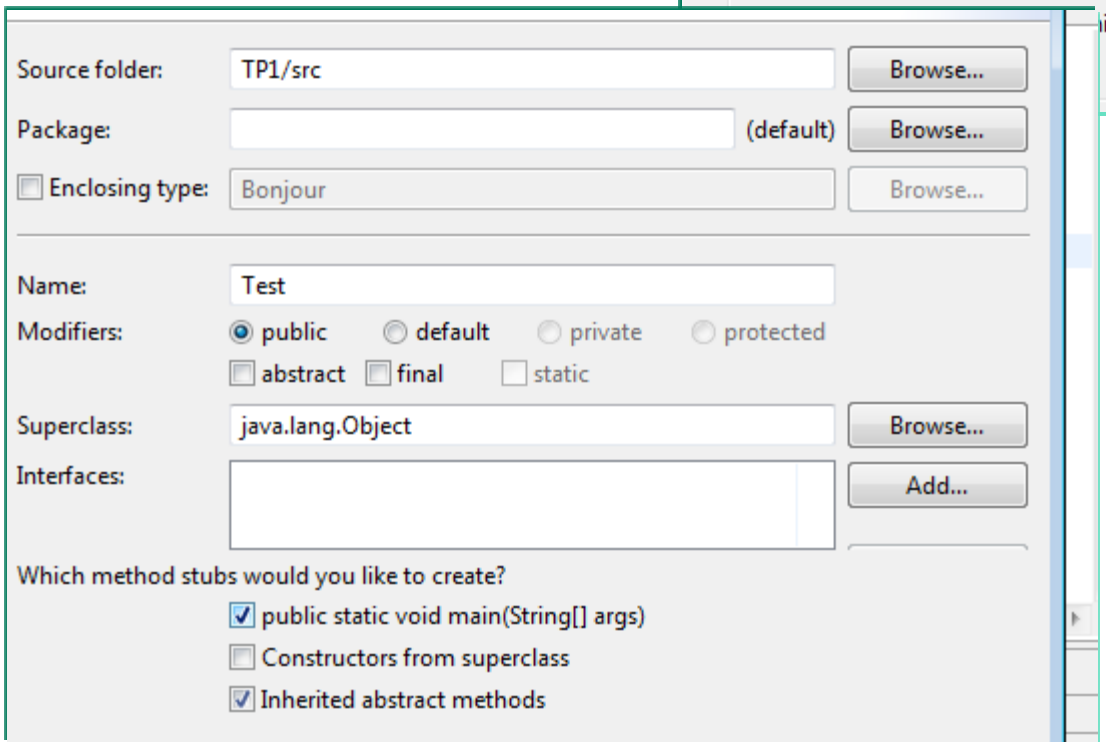


1- C'est quoi Java

1-10 Ecrire une application Java avec un IDE (2/4)

- **Etapes (suite) :**
  - Créer une classe.
    - File → new Class
    - Name→ "Test"





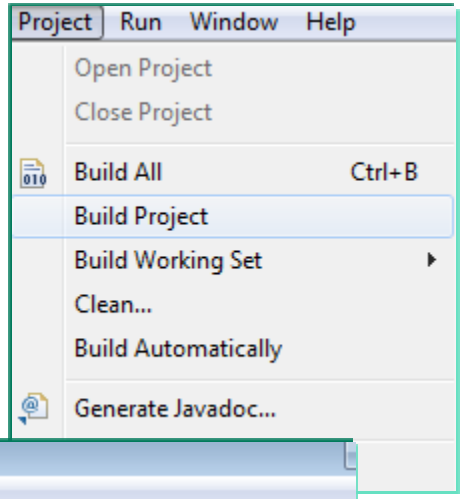
1-10 Ecrire une application Java avec un IDE (3/4)

- **Etapes (suite) :**
  - Insérer l'instruction « System.out.println("Ca Marche"); » dans le corps de la méthode main.

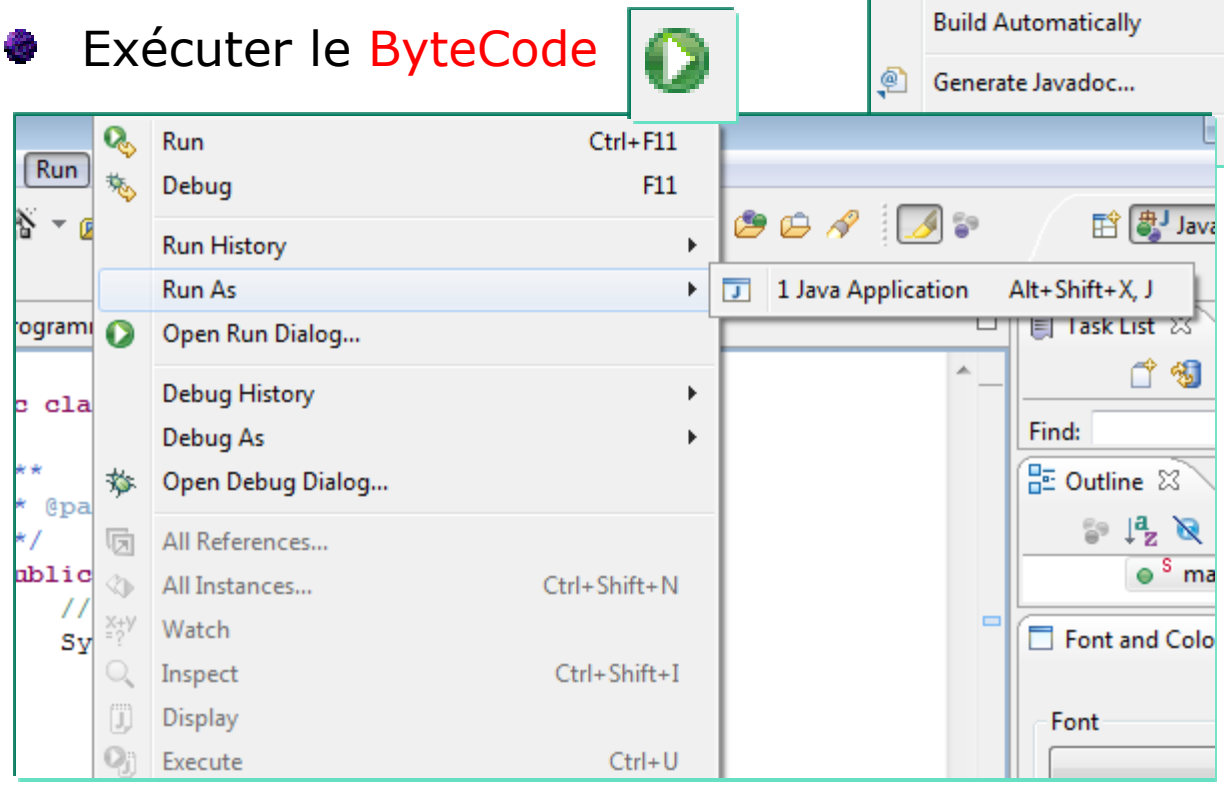
```
public class Test {

    /**
     * @param args
     */
    public static void main(String[] args) {
        System.out.println("ça marche");

    }

}
```

1-10 Ecrire une application Java avec un IDE (4/4)

- **Etapes (suite) :**
  - Compiler le code source.



  - Exécuter le ByteCode



1-11 Point faible de Java

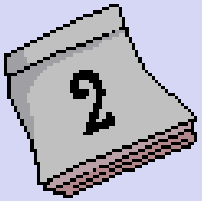
- Pas aussi rapide qu'un programme natif.
- Gourmand en mémoire.

Sites Web :

- Site officiel Java (JDK et doc.) :
  - ~~http://www.javasun.com~~
  - http://www.oracle.com/technetwork/java/javase/documentation/index.html
- Info sur Java :
  - http://www.javaworld.com.

POO : Java

Chapitre



Les bases du langage

T. CHAARI

2- Les bases du langage

2.1 Les commentaires (1/4)

- **Tout programme (grand ou petit, simple ou complexe) contient (ou devrait contenir) des commentaires.**

- **Ils ont pour but d'expliquer :**
  - Ce qu'est sensé faire le programme,
  - Les conventions adoptées,
  - Tout autre information rendant le programme lisible à soi même et surtout à autrui.

- **Java dispose de trois types de commentaires :**
  - Les commentaires multilignes,
  - Les commentaires lignes,
  - Les commentaires de type documentation.

Commentaires en lignes

- Les commentaires lignes débutent avec les symboles ``//'' et qui se terminent à la fin de la ligne.

```
// Ce programme imprime la chaîne
// de caractères " bonjour " à l'écran
...
```

- Ils sont utilisés pour des commentaires courts qui tiennent sur une ligne.

- **Commentaires multilingnes :**
  - Un commentaire multiligne commence par les caractères ``/*'' et se terminent par ``*/''.

```
/* Ce programme imprime la chaîne
de caractères "bonjour" à l'écran
*/
```

  - A l'intérieur de ces délimiteurs toute suite de caractères est valide (sauf évidemment ``*/'').

- **Commentaires de types documentation :**
  - Ces commentaires, appelés aussi commentaires javadoc, servent à documenter les classes que l'on définit.
  - Ces commentaires sont encadrés entre ``/**'' et ``*/'' .

```
/** Documentation de la classe .
*/
```

  - Java exige que ces commentaires figurent avant la définition de la classe, d'un membre de la classe ou d'un constructeur.
  - Ces commentaires serviront à produire automatiquement (avec l'outil javadoc) la documentation sous forme HTML à l'image de la documentation officielle de Oracle.

2- Les bases du langage

2.3 Les types de données élémentaires

- **Le langage Java est un langage fortement typé :**
  - Chaque variable et chaque expression possède un type bien défini.
  - Le choix d'un type est en fonction de l'étendu de valeur souhaitée pour vos variables.
- **Les types de données de Java sont divisés en deux grands groupes :**
  - Les types primitifs (ou de base)
    - Exemple : Les types numériques, le type booléen, Le type caractère.
  - Les types références.

2.4 Les types primitifs numériques (1/2)

Les types numériques entiers				
	byte	short	int	long
Taille en bits	8	16	32	64
Etendue	-128 .. 127	-32768 .. 32767	-2^{31} .. $2^{31}-1$	-2^{63} .. $2^{63}-1$

- **Le bit y est utilisé pour décrire les tailles.**
  - Un bit ne peut prendre que deux valeurs (0 ou 1).
  - **n** bits ne peuvent définir que 2^n valeurs.
  - Un **octet** ou **byte** en anglais = 8 bits.
  - Un octet peut donc contenir 2^8 (soit 256) valeurs distinctes.

2.4 Les types primitifs numériques (2/2)

Les types numériques flottants

- De même que pour les types numériques, il existe deux types pour les nombres flottants :
  - float,
  - double.
- la seule différence résidant dans la taille utilisée pour stocker une valeur de ce type.

	float	double
Taille en bits	32	64
Exemple de valeur	3.25	3.25

2.5 Le type primitif booléen

- **Ce type est introduit en Java par le mot clé booléen.**
- **Ce type est un vrai type booléen**
  - il accepte seulement deux états :
    - l'un est nommé true :
      - Symbolise un état d'acceptation,
    - l'autre, nommé false,
      - Symbolise un état de réfutation.

- **Attention, contrairement au langage C, le type booléen en Java n'est pas un sous-type numérique :**

  - En C par exemple, la valeur 0 est considérée comme fausse et les autres valeurs entières comme vraies.

2.6 Le type primitif caractère

- Ce type, introduit par le mot clé **char**, et permet la gestion des caractères.
- **Java** utilise le codage de caractères universel **Unicode** qui est un extension du codage **ASCII**.
- Le codage **ASCII** utilise **8** bits et permet de représenter seulement **128** caractères.

2.7 Les types références

- Tout type non primitif, est un type de référence.
- Le rôle d'un type de référence est de référencer ou repérer une zone mémoire.
- Un type de référence a un rôle analogue à celui des pointeurs du langage C .

Plus de détails plus loin …

2.8 Déclaration et initialisation des variables

- **Syntaxe :**
  - **Type identificateur [= constante ou expression];**
    - int NbredeMois = 12 ; int NbredeMois = 4*3 ;
    - int nbrDeDoigts = 012 ;
    - boolean Unboolean = false ;
    - float Unfloatant = 1.3f ;
    - char Uncaractère = 'c' ; char Uncaractère = '\n' ;
    - String Unstring= " bonjour \n " ;
  - Et éventuellement, un « modificateur d'accès ou de visibilité » : final float pi=3.14159

String **n'est pas un type primitif, mais une classe**

- **Les opérateurs dans Java sont regroupés par :**
  - type d'opérations :
    - d'affectation
    - numérique,
    - de comparaison,
    - logique,
    - sur les chaînes de caractères,
    - de manipulations binaires.
  - le nombre d'opérandes :
    - unaire,
    - binaire,
    - ternaire.

2- Les bases du langage

2.10- Opérateurs unaires

Opérateurs unaires	Action	Exemple
-	négation	i=-j
++	incrémentation de 1	i=j++ ou i=++j
--	décrémentation de 1	i=j-- ou i=--j

- **++ et -- peuvent préfixer ou postfixer la variable.**
  - i = j++ : post-incrémentation
    - La valeur en cours de j est affectée à i et ensuite la valeur de j est incrémentée de 1.
  - i = ++ j : pré-incrémentation
    - La valeur en cours de j est incrémentée de 1 et ensuite la valeur de j est affectée à i.

2- Les bases du langage

2.11- Opérateurs binaires

Opérateurs binaires	Action	Exemple	Syntaxe équivalente
+	addition	i = j+k;	
+=		i += 2;	i= i + 2
-	soustraction	i = j - k;	
-=		i -= j;	i= i - j
*	multiplication	x=2*y;	
*=		x *=x;	x= x * x
/	division (tronque si les arguments sont entiers)	i =j/k;	
/=		x /= 10;	x= x /10
%	modulo	i = j %k;	
%=		i %=2	i= i %2

2.12- Opérateurs relationnels

- Le résultat d'une comparaison est une valeur booléenne (vrai ou faux).
- Dans le langage Java, le résultat d'une comparaison est true ou false.

Opérateurs relationnels	Action	Exemple
<	plus petit que	x<i;
>	plus grand que	i>100;
<=	plus petit ou égal que	j<=k;
>=	plus grand ou égal que	c>='a';
==	égal à	i==20;
!=	différent de	c!='z';

2- Les bases du langage

2.13- Opérateurs logiques

Opérateurs logiques	Action	Exemple	Syntaxe équivalent
!	négation	!p;	
&	ET	p & (i<10)	
|	OU	p | q	

2.14- Opérateurs ternaire

- **Un unique opérateur ternaire.**
- **Cette expression est une sorte de si-alors-sinon sous forme d'expression :**
  - a = (condition e) ? x : y
    - si la condition e est vraie alors a vaut x sinon elle vaut y.
  - Exemple : a = (v==2) ? 1 : 0;
    - Cette expression affecte à la variable a la valeur 1 si v vaut 2, sinon affecte à la variable a la valeur 0.

2- Les bases du langage

2.15 structures de contrôle : if

- **Syntaxe :**

```
if (expression booléene ) instruction ;
```

- **Exemple :**

```
public class IfApplication1 {

        public static void main(String[ ] args)
        {
                int i = 4 ;
            if ( i %2 != 0 ) System.out.print (" i est impair ");
        }
}
```

Résultat d'affichage

Attention : L'expression logique attendue est obligatoirement de type boolean.

- si l'on écrit if (i = 1), le compilateur détectera une erreur, car le type de l'expression logique est alors int.

2- Les bases du langage

2.15 structures de contrôle : if - else

Syntaxe

```
if (expression booléenne )
{
      instruction 1 ;
      ……
      instruction i ;
}
else
{
      instruction j ;
      ……
      instruction n ;
}
```

Exemple

```
public class If_elseApplication {

   public static void main(String[ ] args)
   {
      int i = 4 ;
      if (i % 2 == 0 )
      {
         System.out.println (" i est pair " );
      }
      else
      {
        System.out.println (" i est impair " );
      }
   }
}
```

Résultat d'affichage

```
i est pair
```

2- Les bases du langage

2.15 structures de contrôle : if – else - if

POO : Java

Syntaxe

```
if (expression booléene 1)
{
      instruction 1 ;
      ……
      instruction i ;
}
else if (expression booléene 2 )
{
      instruction j;
      ……
      instruction m ;
}
else
{
      instruction m+1 ;
      ……
}
```

Exemple

```
public class if_else_ifApplication {

  public static void main(String[ ] args)
  {
    int i = 1, j =2 ;
    if (i == j )
    {
      System.out.println (" i est égal à j " );
    }
    else if ( i>j)
    {
     System.out.println (" i est supérieur à j" );
    }
    else
    {
     System.out.println (" i est positive " );
    }
  }
}
```

Résultat d'affichage

```
i est positive
```

2- Les bases du langage

2.15 structures de contrôle : switch

Syntaxe

```
switch (variable)
{
   case valeur 1 :
      instr1_1; instr1_2; …
      break;
   …
   case valeur N :
      instrN_1; instrN_2;
      break;
   default: /* optionnel */
     instrD_1; instrD_2; …
     break;
}
```

Exemple

```
public class SwitchApplication {

   public static void main(String[ ] args)
   {
      int i = 1,
      switch (i)
      { case 0 :
                  System.out.println (" 0 " );
                  break;
        case 1 :
                  System.out.println (" 1 " );
                  break;
        case 2 :
                  System.out.println (" 2 " );
                  break;
        default :
                  System.out.println (i);
                  break;
      }
   }
}
```

Résultat d'affichage

```
1
```

2- Les bases du langage

2.15 structures de contrôle : for

Exemple :

```
public class ForApplication {

        public static void main(String[ ] args)
        {
                int somme = 0;
                for ( int compteur = 1, max = 4 ; compteur < max ; compteur++)
                {
                    somme = somme + compteur;
                }
                System.out.println (somme );
        }
}
```

Résultat d'affichage
6

Fonctionnement :

- initialisation du compteur,
- comparaison avec max,
- réalisation des instructions,
- Incrémentation du compteur et on recommence.

2- Les bases du langage

2.15 structures de contrôle : while

Exemple :

```
public class WhileApplication {

        public static void main(String[ ] args)
        {
                int compteur =1; int max = 4 ; int somme = 0;

                while (compteur < max )
                {
                    somme = somme + compteur ;
                    compteur++;
                }

                System.out.println (somme );
        }
}
```

Résultat d'affichage

```
6
```

2- Les bases du langage

2.15 structures de contrôle : do - while

Exemple :

```
public class Do_WhileApplication {

        public static void main(String[ ] args)
        {
                int compteur =1; int max = 4 ; int somme = 0;

                do
                {
                     somme = somme + compteur ;
                     compteur++;
                }
                while (compteur < max ) ;

                System.out.println (somme );
        }
}
```

Résultat d'affichage

```
6
```

2.15 structures de contrôle : break

- **Utilisation :**
  - Pour sortir d'une structure de boucle avant que la condition du test soit remplie.
    - Quand la boucle rencontre une instruction break, elle se termine immédiatement en ignorant le code restant.

```
public class BreakApplication {

        public static void main(String[ ] args)
        {
                int compteur =0;

                while (compteur < 10 )
                {
                    System.out.println (compteur );
                      compteur++;
                      if (compteur == 5) break;
                }
        }
}
```

Résultat d'affichage

```
0
1
2
3
4
```

2.15 structures de contrôle : continue

- **Utilisation :**
  - Pour ignorer le reste de la boucle et reprendre l'exécution à l'itération suivante de la boucle.

```
public class ContinueApplication {

        public static void main(String[ ] args)
        {
                int compteur =0;

                for(compteur=0 ; compteur< 6 ; compteur++ )
                {
                    if (compteur == 4) continue;
                    System.out.println (compteur );
                }
        }
}
```

Résultat d'affichage

```
0
1
2
3
5
```

2.16 Les conversions des types (1/8)

- **Il y a 2 catégories de conversions possibles :**
  - Conversions explicites :
    - **celles faites sur une demande explicite par un programmeur.**
  - Conversions implicites :
    - **celles faites automatiquement par un compilateur :**
      - lors d'une affectation,
      - lors d'une promotion arithmétique,
      - lors d'un passage de paramètres (lors de l'invocation d'une méthode),

2.16 Les conversions des types (2/8)

- **Conversion explicite :**
  - Objectif :
    - changer le type d'une donnée si besoin.
  - Comment ? :
    - Préfixer l'opérande par le type choisi.
    - Encadrer le type choisi par des parenthèses.
  - Exemple :
    - double d = 2.5 ;
    - long l = (long) d ;

2.16 Les conversions des types (3/8)

- **Conversion implicite lors d'une affectation :**
  - Objectif :
    - changer automatiquement le type d'une donnée si besoin.
  - Comment ? :
    - **Exemple :**
      - Un type char peut être utilisé partout où une valeur de type int est permise.

2.16 Les conversions des types (4/8)

- **Conversion implicite lors d'une affectation :**
  - Illustration pour l'exemple 1:

```
public class ProgrammeAffectation {

    public static void main(String[ ] args)
    {
        int i;
        short j = 2 ;
        i   =   j ;

        float k= 1.2 f ;
        i   =   k;
    }
}
```

Conversion implicite

Erreur de compilation

  - Solution :

```
        i   = (int)     k;
```

- **Conversion implicite lors d'une affectation :**
  - Illustration pour l'exemple 2:

```
public class ProgrammeSoustraction {

        public static void main(String[ ] args)
        {
                int i;
                i    =   'A';

                System.out.print( i ) ;
        }
}
```

Conversion implicite

  - Résultat d'affichage : **65**

- **Conversion implicite lors d'une promotion arithmétique :**
  - Objectif :
    - Si un opérateur s'applique sur deux arguments de type différent un des deux arguments sera converti dans le type de l'autre.
  - Exemple :

```
public class ProgrammeSoustraction {

        public static void main(String[ ] args)
        {
                int i;
                i    =   'A'-1;

                System.out.print( i ) ;
        }
}
```

Conversion implicite

    - Résultat d'affichage : 64

2.16 Les conversions des types (7/8)

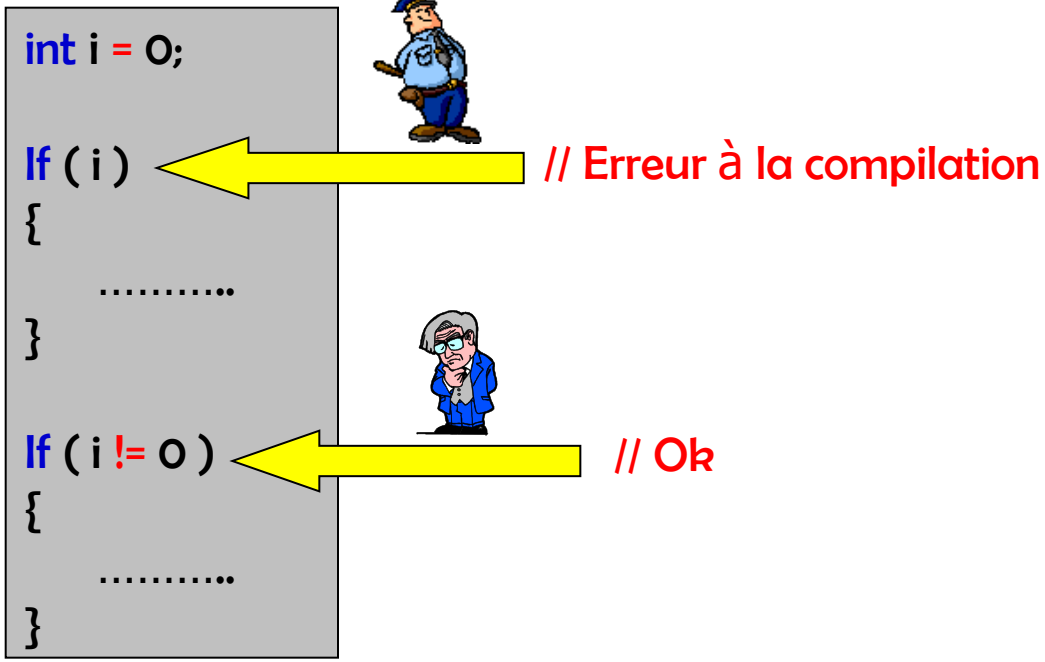
- **Conversion implicite lors d'un passage de paramètres (lors de l'invocation d'une méthode) :**

```
public class ProgrammeSoustraction {

        static short entierShort;

        static void Afficher( int entier )
        {
            System.out.print( entier ) ;
        }

        public static void main(String[ ] args)
        {
                entierShort = 2 ;
                Afficher(entierShort ) ;
        }
}
```

Invocation de la méthode Afficher

Attention :

- Il n'y a pas de conversion possible (implicite ou explicite) entre un type entier et le type boolean :



2.17 Les tableaux (1/2)

- **Déclaration**
  - monodimensionnel
    - `int[ ] tableau_Entier;`
      - équivalent à : `int tableau_Entier[ ];`
  - multidimensionnel
    - `Color rgb_cube[ ][ ][ ];`

- **Création et initialisation**
  - `int[] primes = {1, 2, 3, 5, 7};`
  - `int[] tableau_Entier = new int[50];`
  - `tableau_Entier[0]= 3;`
  - `rgb_cube = new Color[256][256][256];`

- Les indices des tableaux commencent par 0.
- Exemple d'utilisation des tableaux :

```
public class TabeauApplication {

        public static void main(String[ ] args)
        {
                int[ ] tableau_Entier = new int[50]; // création
                System.out.println(tableau_Entier.length);
                tableau_Entier[51]=10;

        }
}
```

Résultat d'affichage

```
50
java.lang.ArrayIndexOutOfBoundsException: 51
```

Exercice 1

- Ecrire le code d'une méthode *afficherTableau* permettant d'afficher le contenu d'un tableau donné de type entiers
- Ecrire le code de la méthode main permettant d'invoquer la méthode *afficherTableau* .

```
public class Tableau {

public static void afficherTableau(int []tab)
{
for(int i=0;i<tab.length;i++)
System.out.println(tab[i]);
}

public static void main(String[] args)
{

int[] tableau_Entier = new int[2];

tableau_Entier[0]=2;
tableau_Entier[1]=5;
afficherTableau(tableau_Entier);
}
}
```

Exercice 2

Rappel: String[] args : est un tableau de chaîne de caractères nommé args qui permet de passer des arguments saisis par l'utilisateur avant l'exécution du programme

- **Ecrire le code de la méthode main permettant d'afficher les arguments donnés en exécution en ordre inversé.**

```
public class Chaine{

public static void main(String[] args)
{

For (int i=args.length-1;i>=0;i--)

System.out.println(args[i]);
}
}
```

POO : Java

Chapitre

Le Concept Objet de Java

T. CHAARI

Principe POO

- Programmation par Objets
  - Unité logique : l'objet
  - Objet est défini par
    - un état
    - un comportement
    - une identité

maVoiture
- couleur = bleue
- vitesse = 100

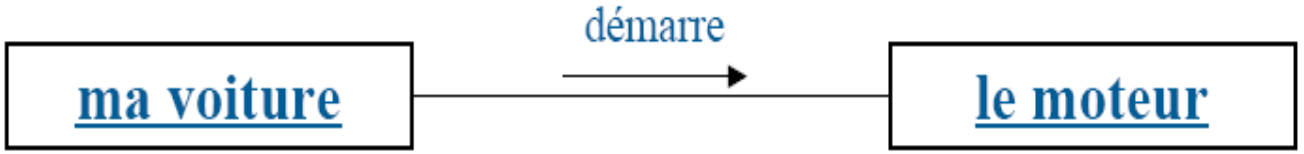
- État : représenté par des attributs (variables) qui stockent des valeurs
- Comportement : défini par des méthodes (procédures) qui modifient des états
- Identité : permet de distinguer un objet d'un autre objet

Principe POO

- Les objets communiquent entre eux par des messages
  - Un objet peut recevoir un message qui déclenche :
    - une méthode qui modifie son état

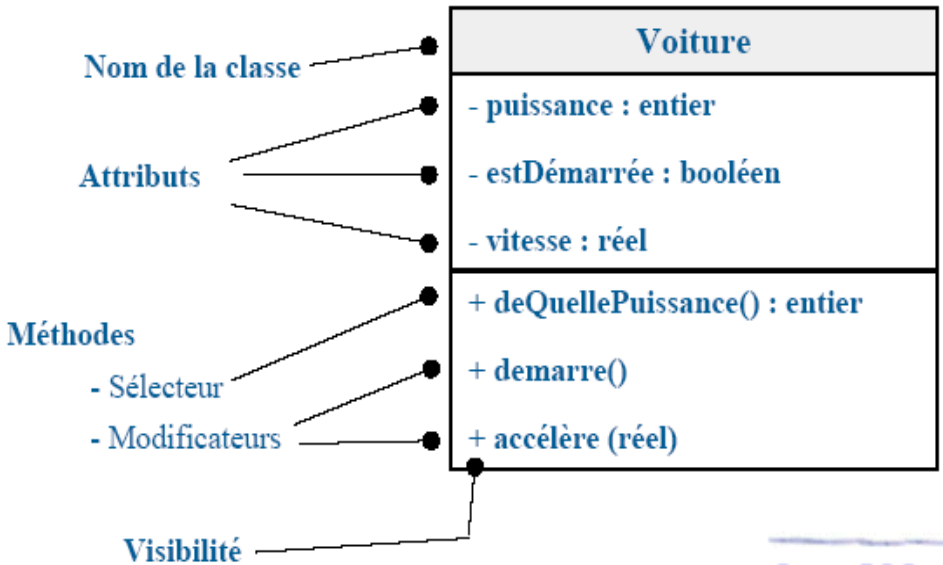
    et / ou

    - une méthode qui envoie un message à un autre objet



Principe POO

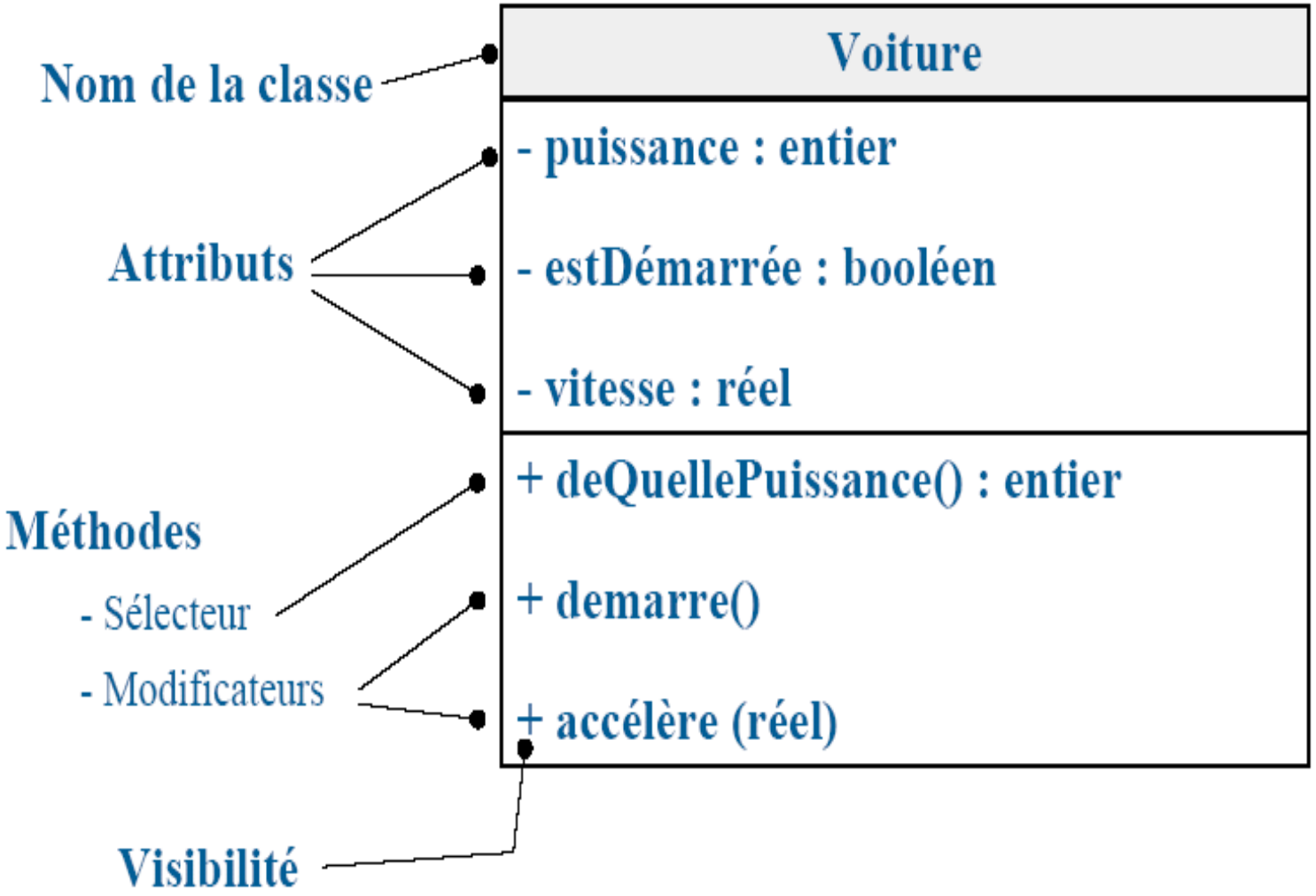
- Notion de classe
  - On regroupe les objets qui ont les mêmes états et les même comportements : c'est une classe
  - Les classes servent de « moules » pour la création des objets : un objet est une « instance » d'une classe
  - Un programme OO est constitué de classes qui permettent de créer des objets qui s'envoient des messages



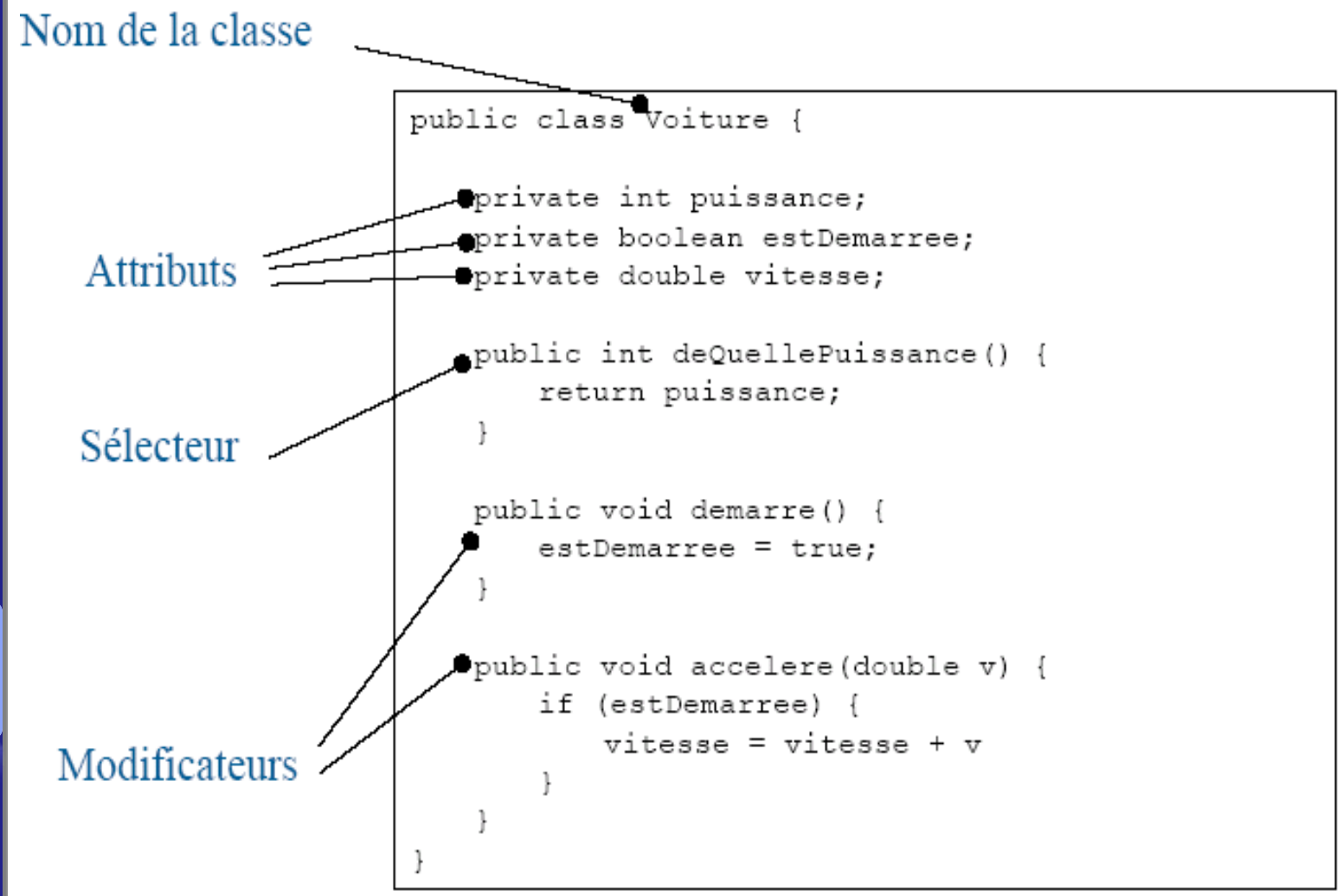
Classe : Définition

- Une classe est constituée :
  - Données ce qu'on appelle des **attributs**
  - Procédures et/ou des fonctions ce qu'on appelle des **méthodes**
- Une classe est un modèle de définition pour des objets
  - Ayant même structure (même ensemble d'attributs)
  - Ayant même comportement (même méthodes)
  - Ayant une sémantique commune
- Les **objets** sont des représentations dynamiques (**instanciation**), du modèle défini pour eux au travers de la classe
  - Une classe permet d'**instancier** (créer) plusieurs objets
  - Chaque objet est instance d'une classe et une seule

Classe : Notations



Codage de la classe « Voiture »



Classe : Attributs

- Caractéristique d'un attribut :
  - Variables « globales » de la classe
  - Accessibles dans toutes les méthodes de la classe

```
public class Voiture {

    private int puissance;
    private boolean estDemarree;
    private double vitesse;

    public int deQuellePuissance() {
        return puissance;
    }

    public void demarre() {
        estDemarree = true;
    }

    public void accelere(double v) {
        if (estDemarree) {
            vitesse = vitesse + v
        }
    }
}
```

Attributs visibles dans les méthodes

Classe : Attributs et variables

- Caractéristique d'une variable :
  - Visible à l'intérieur du bloc qui le définit

```
public class Voiture {

    private int puissance;
    private boolean estDemarree;
    private double vitesse;

    public int deQuellePuissance() {
        return puissance;
    }

    public void demarre() {
        estDemarree = true;
    }

    public void accelere(double v) {
        if (estDemarree) {
            double avecTolerance;
            avecTolerance = v + 25;
            vitesse = vitesse + avecTolerance
        }
    }
}
```

Variable visible uniquement dans cette méthode

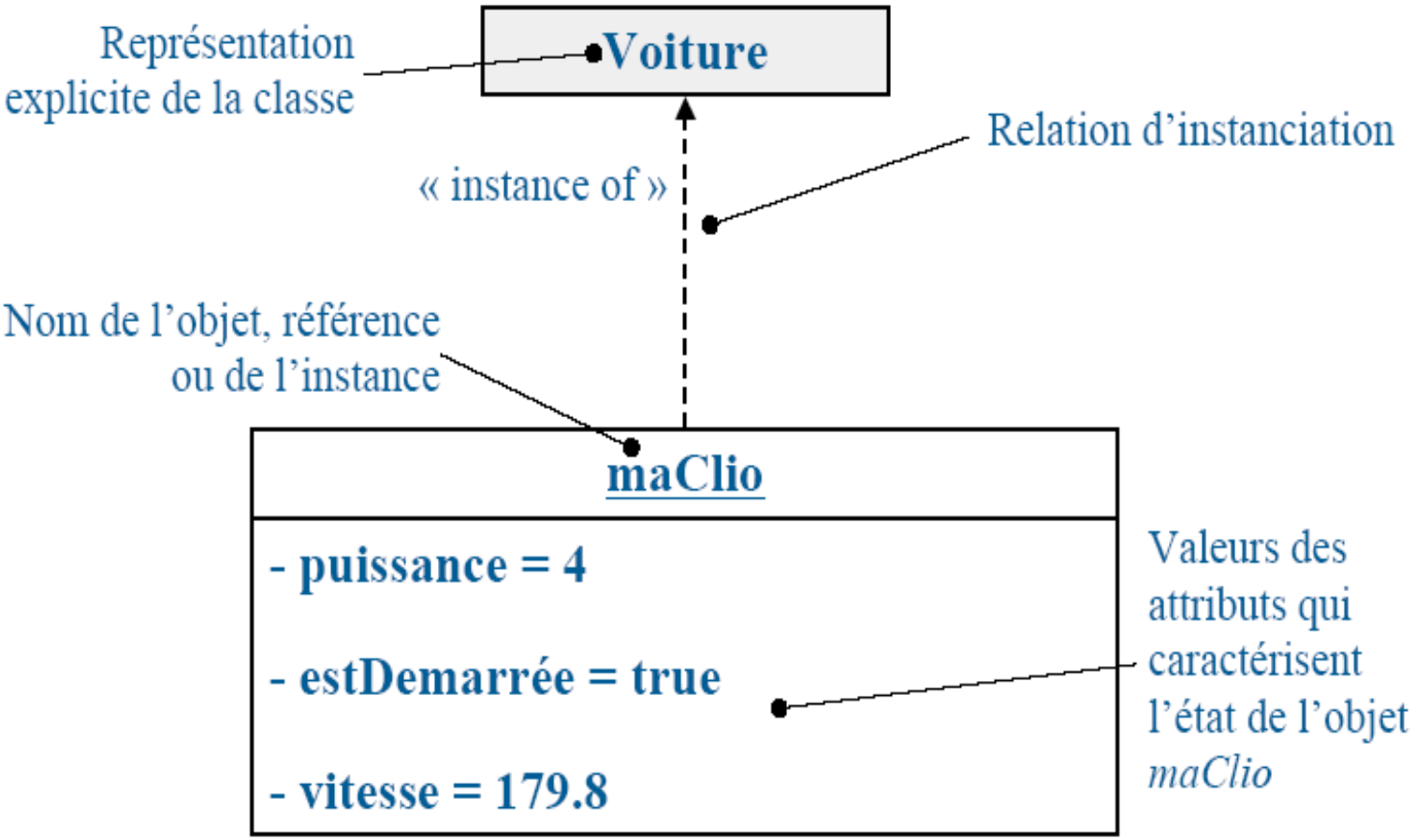
Variable peut être définie n'importe où dans un bloc

Objet : Définition

- Un objet est **instance** d'une seule classe :
  - Se conforme à la description que celle-ci fournit
  - Admet une valeur propre à l'objet pour chaque attribut déclaré dans la classe
  - Les valeurs des attributs caractérisent l'**état** de l'objet
  - Possibilité de lui appliquer toute opération (**méthode**) définie dans la classe
- Tout objet est manipulé et identifié par sa référence

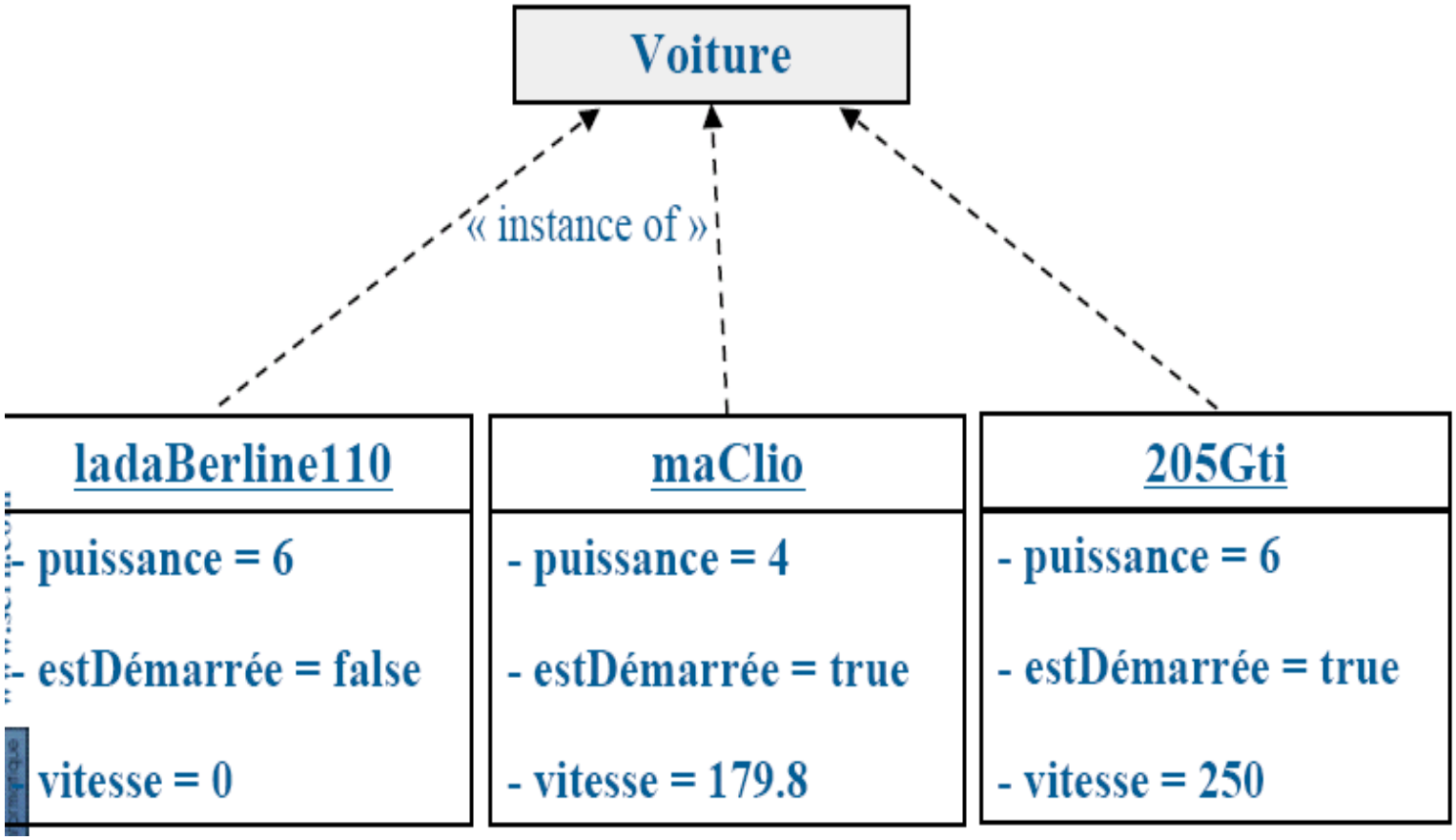
Objet : Notations

- maClio est une instance de la classe *Voiture*



Etats des objets

- Chaque objet qui est une instance de la classe *Voiture* possède ses propres valeurs d'attributs



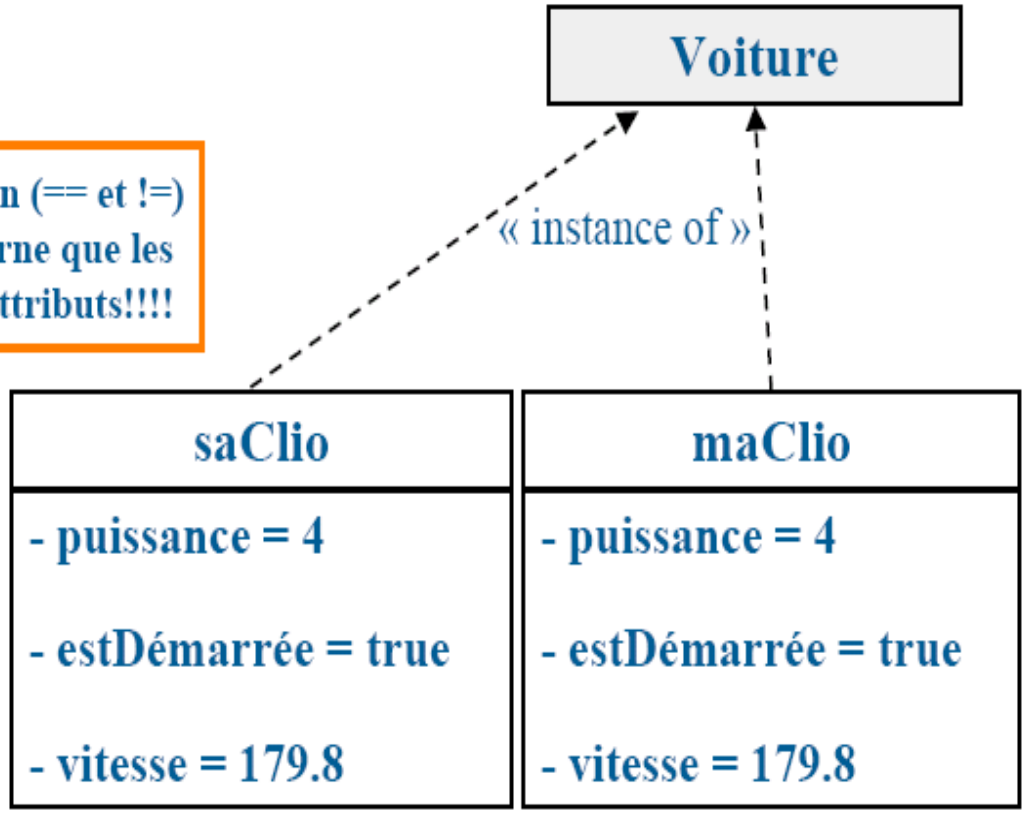
Affectation et comparaison

- Affecter un objet
  - « a = b » signifie a devient identique à b
  - Les deux objets a et b sont identiques et toute modification de a entraîne celle de b
- Comparer deux objets
  - « a == b » retourne « true » si les deux objets sont identiques
  - C'est-à-dire si les références sont les mêmes, cela ne compare pas les attributs

Affectation et comparaison

- L'objet **maClio** et **saClio** ont les mêmes attributs (états identiques) mais ont des références différentes
  - maClio != saClio

Le test de comparaison (== et !=) entre objets ne concerne que les référence et non les attributs!!!!



Cycle de vie d'un objet

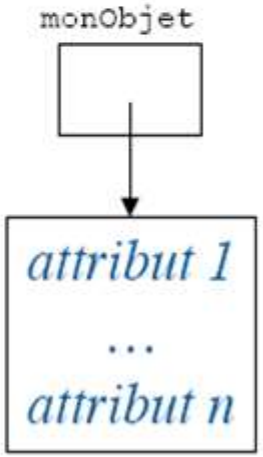
- Création
  - Usage d'un Constructeur
  - L'objet est créé en mémoire et les attributs de l'objet sont initialisés
- Utilisation
  - Usage des Méthodes et des Attributs (non recommandé)
  - Les attributs de l'objet peuvent être modifiés
  - Les attributs (ou leurs dérivés) peuvent être consultés

L'utilisation d'un objet non construit provoque une exception de type *NullPointerException*

- Destruction et libération de la mémoire

Création d'objets

- La création d'un objet à partir d'une classe est appelée une **instanciation**.
- L'objet créé est une **instance** de la classe
- Déclaration
  - Définit le nom et le type de l 'objet
  - Un objet seulement déclaré vaut « **null** » (mot réservé du langage)
- Création et allocation de la mémoire
  - Appelle de méthodes particulières : les constructeurs
  - La création réserve la mémoire et initialise les attributs
- Renvoi d'une référence sur l'objet maintenant créé
  - monObjet != **null**



Création d'objets

- La création d'un objet est réalisée par **new** Constructeur(paramètres)
  - Il existe un constructeur par défaut qui ne possède pas de paramètre (si aucun autre constructeur avec paramètre n'existe)

Les constructeurs portent le même nom que la classe

Constructeur avec un paramètre

```
public class Voiture {

    private int puissance;

    private boolean estDemarree;

    private double vitesse;

    public Voiture(int p) {
        puissance = p;
        estDemaree = false;
        vitesse = 0;
    }
    ...
}
```


Création d'objets

Déclaration

Création et allocation mémoire

```
public class TestMaVoiture {

    public static void main (String[] argv)  {

        // Déclaration puis création
        Voiture maVoiture;
        maVoiture = new Voiture();

        // Déclaration et création en une seule ligne
        Voiture maSecondeVoiture = new Voiture();
    }
}
```

Création d'objets

- Exemple : construction d'objets

```
public class TestMaVoiture {

    public static void main (String[] argv) {

        // Déclaration puis création
        Voiture maVoiture;
        maVoiture = new Voiture();

        // Déclaration d'une deuxième voiture
        Voiture maVoitureCopie;
        // Attention!! pour l'instant maVoitureCopie vaut null


        // Test sur les références.
        if (maVoitureCopie == null) {

            // Création par affectation d'une autre référence
            maVoitureCopie = maVoiture
            // maVoitureCopie possède la même référence que maVoiture
        }
        ...
    }}
```

Déclaration

Affectation par référence

POO : Java

T. CHAARI

Constructeur de « Voiture »

- Actuellement
  - On a utilisé le constructeur par défaut sans paramètre
  - On ne sait pas comment se construit la « Voiture »
  - Les valeurs des attributs au départ sont indéfinies et identiques pour chaque objet (puissance, etc.)

Les constructeurs portent le même nom que la classe et n'ont pas de valeur de retour

- Rôle du constructeur en Java
  - Effectuer certaines initialisations nécessaire pour le nouvel objet créé

- Toute classe Java possède au moins un constructeur
  - Si une classe ne définit pas explicitement de constructeur, un constructeur par défaut sans arguments et qui n'effectue aucune initialisation particulière est invoquée

Constructeur de « Voiture »

- Le constructeur de « Voiture »
  - Initialise « vitesse » à zéro
  - Initialise « estDémaree » à faux
  - Initialise la « puissance » à la valeur passée en paramètre du constructeur

Constructeur avec un paramètre

```
public class Voiture {

    private int puissance;

    private boolean estDemarree;

    private double vitesse;

    public Voiture(int p) {
        puissance = p;
        estDemaree = false;
        vitesse = 0;
    }
    ...
}
```

Construire une voiture de 7CV

- Création de la « Voiture » :
  - Déclaration de la variable « maVoiture »
  - Création de l'objet avec la valeur 7 en paramètre du constructeur

Déclaration

Création et allocation mémoire avec Voiture(int)

```
public class TestMaVoiture {

    public static void main(String[] argv)   {

        // Déclaration puis création
        Voiture maVoiture;

        maVoiture = new Voiture(7);

        Voiture maSecVoiture;
        // Sous entendu qu'il existe
        // explicitement un constructeur : Voiture(int)

        maSecVoiture = new Voiture(); // Erreur
    }
}
```

Constructeur sans arguments

- Utilité :
  - Lorsque l'on doit créer un objet sans pouvoir décider des valeurs de ses attributs au moment de la création
  - Il remplace le constructeur par défaut qui est devenu inutilisable dès qu'un constructeur (avec paramètres) a été défini dans la classe

```
public class Voiture {

    private int puissance;
    private boolean estDemarree;
    private double vitesse;

    public Voiture() {
        puissance = 4;
        estDemaree = false;
        vitesse = 0;
    }

    public Voiture(int p) {
        puissance = p;
        estDemaree = false;
        vitesse = 0;
    }...
```

```
public class TestMaVoiture {

    public static void main (String[] argv) {

        // Déclaration puis création
        Voiture maVoiture;
        maVoiture = new Voiture(7);
        Voiture maSecVoiture;
        maSecVoiture = new Voiture(); // OK
    }
}
```

Exercice

- Ecrire le code d'une classe *Point* représentant un point cartésien sur un plan à deux dimensions.
- Avec cette classe, on doit pouvoir créer un point dans des coordonnées données.
- On doit pouvoir aussi créer un point par défaut aux coordonnées (0,0).
- Cette classe offre une méthode *deplacer* permettant de déplacer le point d'une distance *dx* sur l'axe des x et d'une distance *dy* sur l'axe des y

Solution


```
class Point
{
double x,y;

Point (double a, double b)
{
x=a;
y=b;
}

Point ()
{
x=0;
y=0;
}
```

```
void deplacer

(double dx, double dy)


{
x=x+dx;
y=y+dy;
}

}
```

Accès aux attributs

- Pour accéder aux données d'un objet on utilise une notation pointée

```
identificationObjet.nomAttribut
```

```
public Voiture() {
    puissance = 4;
    estDemaree = false;
    vitesse = 0;
}
```

```
public class TestMaVoiture {

    public static void main (String[] argv) {

        // Déclaration puis création
        Voiture v1 = new Voiture();
        Voiture v2 = new Voiture();

        // Accès aux attributs en écriture
        v1.puissance = 110;

        // Accès aux attributs en lecture
        System.out.println("Puissance de v1 = "  + v1.puissance);
    }
}
```

Il n'est pas recommandé d'accéder directement aux attributs d'un objet

Appel de méthodes

- Pour « demander » à un objet d'effectuer un traitement il faut lui **envoyer un message**
- Un message est composé de trois parties
  - Une référence permettant de désigner l'objet à qui le message est envoyé
  - Le nom de la méthode ou de l'attribut à exécuter
  - Les éventuels paramètres de la méthode

```
identificationObjet.nomDeMethode(« Paramètres éventuels »)
```

- Envoi de message similaire à un appel de fonction
  - Le code défini dans la méthode est exécuté
  - Le contrôle est retourné au programme appelant

Appel de méthodes

Ne pas oublier les parenthèses pour les appels aux méthodes

```
public class TestMaVoiture {

    public static void main (String[] argv)  {

        // Déclaration puis création
        Voiture maVoiture = new Voiture();

        // La voiture démarre
        maVoiture.demarre();

        if (maVoiture.deQuellePuissance() == 4) {
            System.out.println("Pas très Rapide…");
        }

        // La voiture accélère
        maVoiture.accélère(123.5);
    }
```

Voiture
- …
+ deQuellePuissance() : entier + demarre() + accélère (réel) + …

Envoi d'un message à l'objet *maVoiture*

Appel d'un modificateur

Envoi d'un message à l'objet *maVoiture*

Appel d'un sélecteur

Objet « courant »

- L'objet « courant » est désigné par le mot clé **this**
  - Permet de désigner l'objet dans lequel on se trouve
  - **self** ou **current** dans d'autres langages
  - Désigne une référence particulière qui est un membre caché

Ne pas tenter d'affecter une nouvelle valeur à this !!!!

```
this = ... ; // Ne pas y penser
```

- Utilité de l'objet « courant »
  - Rendre explicite l'accès aux propres attributs et méthodes définies dans la classe

Objet « courant » : Méthodes

- Désigne des variables ou des méthodes définies dans une classe

```
public class Voiture {

    ...
    private boolean estDemarree;
    private double vitesse;

    public int deQuellePuissance() {
        ...
    }

    public void accelere(double vitesse) {
        if (estDemarree) {
            this.vitesse = this.vitesse + vitesse;
        }
    }
}
```

Désigne la variable *vitesse*

Désigne l'attribut *vitesse*

Désigne l'attribut *demarree*

this n'est pas nécessaire lorsque les identificateurs de variables ne présentent aucun équivoque

Exercice: retour à la classe Point

- Modifier le constructeur Point (double a, double b) en Point (double x, double y)
- Ajouter un constructeur permettant de créer un point par copie d'un autre point donné
- Ajouter une méthode afficher permettant d'afficher les coordonnées du point
- Ajouter une méthode distance permettant de calculer la distance entre le point actuel et un autre point donné
- Ecrire une classe TestPoint contenant un main permettant de tester tous les constructeurs et toutes les méthodes de la classe Point

Solution(1/2)

```
class Point {
 double x,y;

 Point (double x, double y) {
  this.x=x;
  this.y=y;
 }

 Point (Point p) {
  this.x=p.x;
  this.y=p.y;
 }

Point () {
  this.x=0;
  this.y=0;
}
```

```
void afficher(){
  System.out.println("("+
    this.x+","+this.y+")");
 }

double distance(Point p){
  return Math.sqrt((this.x-
    p.x)*(this.x-p.x)+(this.y-
    p.y)*(this.y-p.y));
 }

 void deplacer(double dx, double
    dy){
  this.x=this.x+dx;
  this.y=this.y+dy;
 }
}
```

Solution(2/2)


```
class TestPoint {
 public static void main(String[] arguments);
 {
  Point p1 = new Point(3,1); // création (appel constructeur)
  Point p2 = new Point(); // création (appel constructeur)
  Point p3 = new Point(p1); // création (appel constructeur)

 System.out.println("P2: "+p2.x+","+p2.y);//accès aux attributs

  p3.afficher(); // (3,1)

  System.out.println(p1.distance(p2)); // appel de méthode

  p3.deplacer(-3,-2); // appel de méthode

  p3.afficher(); // (0,-1)
 }
}
```

Passage de paramètres

- Un paramètre d'une méthode peut être
  - Une variable de type simple
  - Une référence d'un objet typée par n'importe quelle classe
- En Java tout est passé par valeur
  - Les paramètres effectifs d'une méthode
  - La valeur de retour d'une méthode (si différente de « void »)
- Les types simples
  - Leur valeur est recopiée
  - Leur modification dans la méthode n'entraîne pas celle de l'original
- Les objets
  - Leur référence est recopiée et non pas les attributs
  - Leur modification dans la méthode entraîne celle de l'original!!!

Exemple passage de paramètres

```
void affichageDouble (int a)
{ a=a*2;
System.out.println (a);}
-----------------------------
 int a=4;
affichageDouble (a);
// a=8
System.out.println (a);
// a=4
```


Exemple passage de paramètres


```
void deplacerPoint (Point p, double dx, double dy)
{
p.deplacer (dx,dy);
}
-------------------------
Point p=new Point(1,1);
p.afficher();
// (1,1)
deplacerPoint (p,2,2);
p.afficher();
//(3,3)
```

```
void deplacer(double dx,
double dy)
{
 this.x=this.x+dx;
 this.y=this.y+dy;
}
```

```
void afficher(){
 System.out.println("("+
   this.x+","+this.y+")");
}
```

Encapsulation

- Possibilité d'accéder aux attributs d'une classe Java mais pas recommandé car contraire au principe d'encapsulation
  - Les données (attributs) doivent être protégés et accessibles pour l'extérieur par des sélecteurs
- Possibilité d 'agir sur la visibilité des membres (attributs et méthodes) d'une classe vis à vis des autres classes
- Plusieurs niveaux de visibilité peuvent être définis en précédant d'un modificateur la déclaration d'un *attribut*, *méthode* ou *constructeur*
  - private
  - public
  - protected

A revoir dans la partie suivante

Encapsulation

	+ public	- private
classe	La classe peut être utilisée par n'importe quelle classe	Utilisable uniquement par les classes définies à l'intérieur d'une autre classe. Une classe privée n'est utilisable que par sa classe englobante
attribut	Attribut accessible partout où sa classe est accessible. N'est pas recommandé du point de vue encapsulation	Attribut restreint à la classe où est faite la déclaration
méthode	Méthode accessible partout où sa classe est accessible.	Méthode accessible à l'intérieur de la définition de la classe

Encapsulation

Exemple

```
public class Voiture {

    private int puissance;
    ...

    public void demarre() {
        ...
    }

    private void makeCombustion() {
        ...
    }
}
```

Une méthode privée ne peut plus être invoquée en dehors du code de la classe où elle est définie

```
public class TestMaVoiture {

    public static void main (String[] argv) {
        // Déclaration puis création de maVoiture
        Voiture maVoiture = new Voiture();

        // Démarrage de maVoiture
        maVoiture.demarre();

        maVoiture.makeCombustion(); // Erreur
    }
}
```

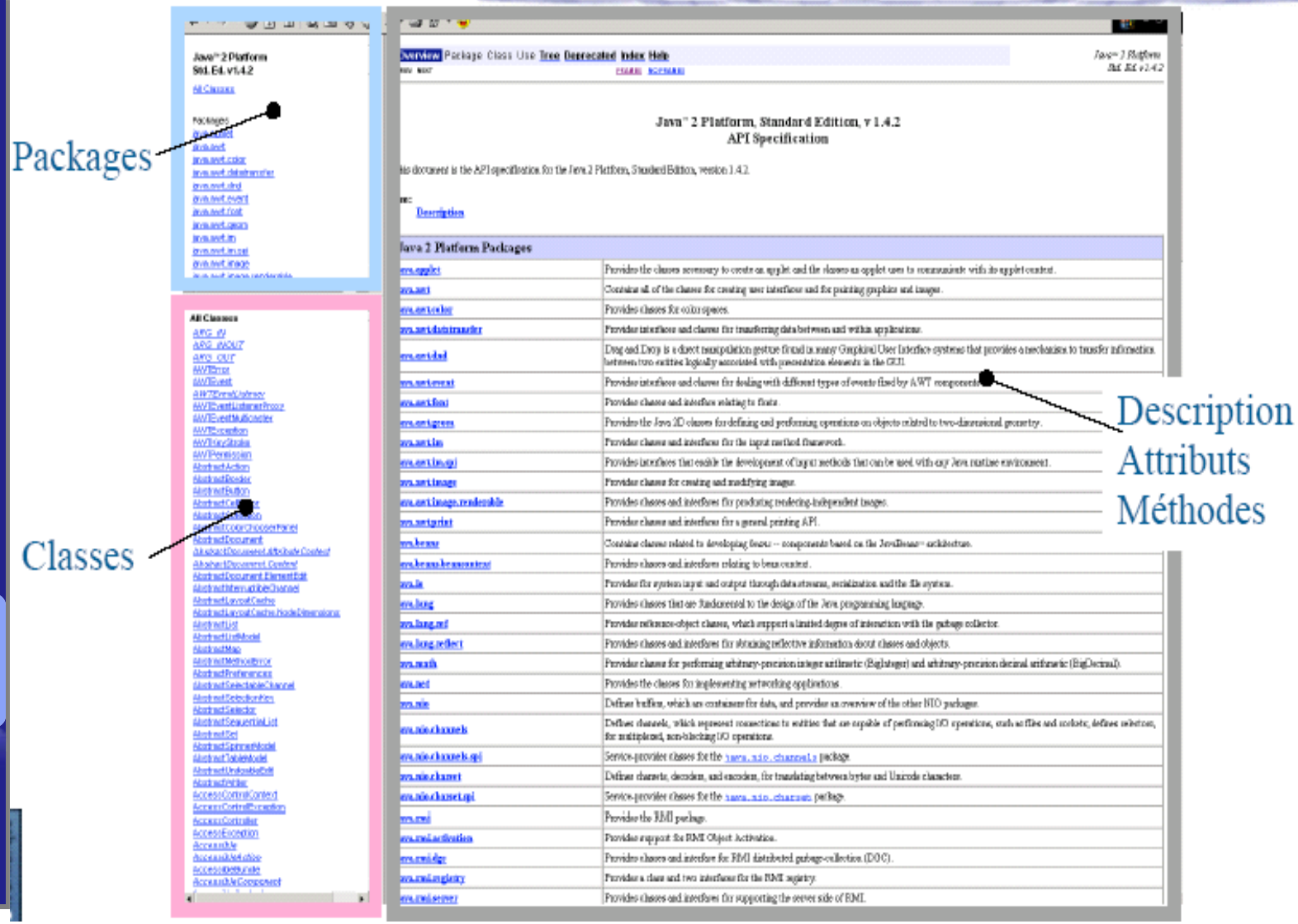
Précision sur « System.out. »

- Usages : affichage à l'écran
  - « System.out.println(…) » : revient à la ligne
  - « System.out.print(…) » : ne revient pas à la ligne
- Tout ce que l'on peut afficher…
  - Objets, nombres, booléens, caractères, …
- Tout ce que l 'on peut faire…
  - Concaténation sauvage entre types et objets avec le « + »

API Java

POO : Java

T. CHAARI



Chaînes des caractères « String »

POO : Java

- Ce sont des objets traités comme des types simples...
- Initialisation

```
String maChaine = "Bonjour!"; // Cela ressemble à un type simple
```

- Longueur

```
maChaine.length(); // Avec les parenthèses car c'est une méthode
```

- Comparaison

```
maChaine.equals("Bonjour!"); // Renvoi vrai
```

- Concaténation

```
String essai = "ess" + "ai";
String essai = "ess".concact("ai");
```

Faites attention à la comparaison de chaînes de caractères.

```
maChaine == "toto";
```

Comparaison sur les références !!

Opérations de bases sur « String »

POO : Java

```
String s = "C' est " ; // création
String t = s + " le moment " ; // concaténation
String t1 = s + " " + 2.54 ; // conversion
int len = t . length ( ) ; // t a i l l e d ' une chaîne
String sub = t . substring ( 3 , 5 ) ; // extraction d'une sous-chaîne
char c = t . charAt ( 2 ) ; // extraction d'un caractère
boolean b1 ;
b1 = t . equals ( " h e l l o " ) ; // test d'égalité
int r ; r = t . compareTo ( " bonjour " ) ; // comparaison alphabétique
r = t . indexOf ( 't' ) ; // index d'éléments
r = t . indexOf ( 't' , r+1) ;
r = t . lastIndexOf ( 't' ) ;
r = t . lastIndexOf ( 't' , r-1);
String nouv = t . replace ( ' a ' , ' h ' ) ;  //remplacement sans modification de
    la chaîne initiale
```

Variables de classe

- Il peut être utile de définir pour une classe des attributs indépendamment des instances : nombre de Voitures créées

- Utilisation des Variables de classe comparables aux « variables globales »

- Usage des **variables de classe**
  - Variables dont il n'existe qu'un seul exemplaire associé à sa classe de définition
  - Variables existent indépendamment du nombre d'instances de la classe qui ont été créés
  - Variables utilisables même si aucune instance de la classe n'existe

Variables de classe

- Elles sont définies comme les attributs mais avec le mot-clé **static**

```
public static int nbVoitureCreees;
```

Attention à l'encapsulation. Il est dangereux de laisser cette variable de classe en public.

- Pour y accéder, il faut utiliser non pas un identificateur mais le nom de la classe

```
Voiture.nbVoitureCreees = 3;
```

Il n'est pas interdit d'utiliser une variable de classe comme un attribut (au moyen d 'un identificateur) mais fortement déconseillé

Constantes de classe

- Usage
  - Ce sont des constantes liées à une classe
  - Elles sont écrites en MAJUSCULES

Une constante de classe est généralement toujours visible

- Elles sont définies (en plus) avec le mot-clé final

```
public class Galerie {
    public static final int MASSE_MAX = 150;
}
```

- Pour y accéder, il faut utiliser non pas un identificateur d 'objet mais le nom de la classe (idem variables de classe)

```
if (maVoiture.getWeightLimite() <= Galerie.MASSE_MAX) {...}
```

Variable et constantes de classe

Exemple

```
public class Voiture {

    public static final int PTAC_MAX = 3500;
    private int poids;
    public static int nbVoitureCreees;
    ...

    public Voiture(int poids, ...) {
        this.poids = poids;
        ...
    }
}
```

Dangereux car possibilité de modification extérieure...

```
public class TestMaVoiture {

    public static void main (String[] argv) {
        // Déclaration puis création de maVoiture
        Voiture maVoiture = new Voiture(2500);
        ...

        System.out.println("Poids maxi:" +
            Voiture.PTAC_MAX);
        System.out.println(Voiture.nbVoitureCreees);
        ...
    }
}
```

Utilisation de Variables et Constantes de classe par le nom de la classe Voiture

Méthode de classe

- Usage
  - Ce sont des méthodes qui ne s'intéressent pas à un objet particulier
  - Utiles pour des calculs intermédiaires internes à une classe
  - Utiles également pour retourner la valeur d'une variable de classe en visibilité *private*

- Elles sont définies comme les méthodes d'instances, mais avec le mot clé **static**

```
public static double vitesseMaxToleree() {
    return vitesseMaxAutorisee*1.10;
}
```

- Pour y accéder, il faut utiliser non pas un identificateur d'objet mais le nom de la classe (idem variables de classe)

```
Voiture.vitesseMaxToleree()
```

Méthode de classe

Exemple

```
public class Voiture {

    private static int nbVoitureCreees;
    ...

    public static int getNbVoitureCreees(){
        return Voiture.nbVoitureCreees;
    }
}
```

Déclaration d'une variable de classe privée. Respect des principes d'encapsulation.

Déclaration d'une méthode de classe pour accéder à la valeur de la variable de classe.

```
public class TestMaVoiture {

    public static void main (String[] argv) {
        // Déclaration puis création de maVoiture
        Voiture maVoiture = new Voiture(2500);
        ...

        System.out.println("Nbre Instance :" +
                Voiture.getNbVoitureCreees());
    }
}
```

Méthode de classe : erreur classique

- Exemple

```
public class Voiture {

    private Galerie laGalerie;
    ...

    public Voiture(Galerie g) {
        laGalerie = g;
        ...
    }

    public static boolean isGalerieInstall() {
        return (laGalerie != null)
    }
}
```

Déclaration d'un objet Galerie non statique

Erreur : Utilisation d'un attribut non statique dans une zone statique

On ne peut pas utiliser de variables d'instance dans une méthode de classe!!!!

Exercice

- Ecrire le code d'une classe Division qui offre deux méthodes *diviser*
  - La première est une méthode d'instance
  - La deuxième est une méthode de classe
- Ecrire le code d'une classe *TestDivision* qui permet de tester les deux méthodes *diviser. Laquelle est meilleure dans ce cas?*

Solution (1/2)

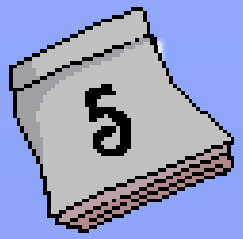
```
Public class Division
{
private double a,b;
Division (double a, double b)
{ this.a=a; this.b=b;}

public double diviser ()
{return (this.a/this.b);}

public static double diviser (double a, double b)
{return (a/b);}
}
```

Solution (2/2)

```
Public class TestDivision {

Public static void main (String [] args) {
Division d1= new Division (7,7);
double resultat=d1.diviser();
System.out.println (resultat);
//1
double resultat=Division.diviser(7,7);
System.out.println (resultat);
//1
}
}
```

POO : Java

Chapitre



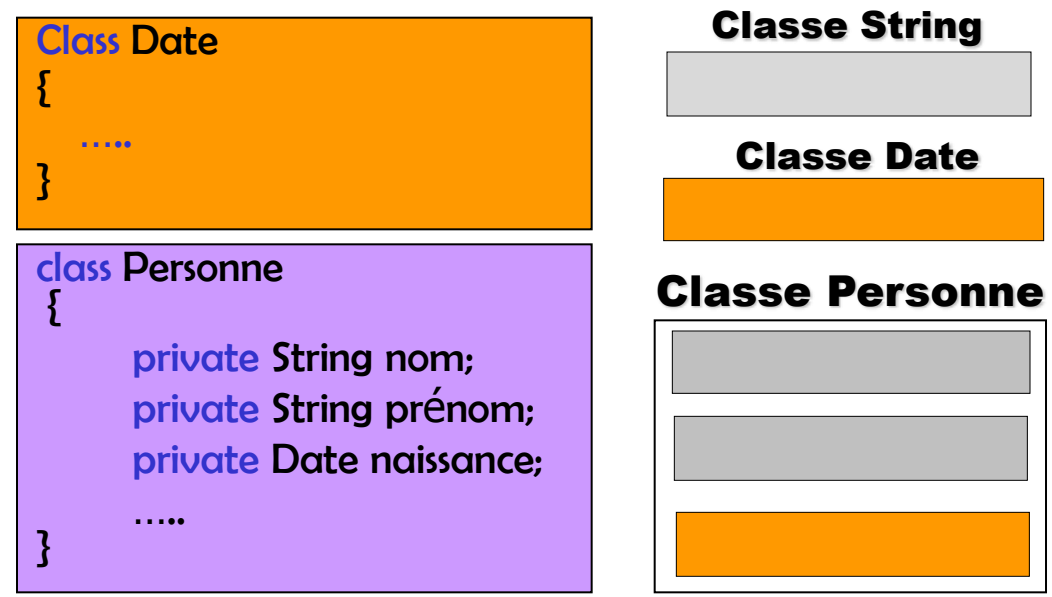
Programmation Orientée Objet Avancée Java

5.1 Composition de classes (1/4)

- **Principe :**
  - La réutilisation par composition :
    - **Quand une classe est testée et elle est opérationnelle on peut l'utiliser aussi en tant que nouveau type dans une autre classe.**
      - Les attributs d'une classe peuvent être des instances d'autres classes.

```
Class Date
{
   …..
}
```

```
class Personne
 {
        private String nom;
        private String prénom;
        private Date naissance;
        ….
}
```



Classe Date :

```
class Date {

        private int j;        // jour
        private int m;        // mois
        private int a;        // an

        Date( )
        { j=1 ; m=1 ; a=1900 ; }

        Date(int j, int m, int a)
        { this.j = j ; this.m = m ; this.a = a ; }

        void initialise(int jour, int mois, int an)
        {  this.j=jour; this.m=mois; this.a=an;  }

        void affiche( )
         {  System.out.println(j + "/" + m +  "/" + a) ;  }
}
```

Classe Personne :

```
class Personne {

        private String nom;
        private String prénom;
        private Date naissance;
        Personne(String nom, String prénom, int jour, int mois, int an)
        {
           this.nom=nom;
           this.prénom=prénom;
           naissance=new Date(jours,mois,an);
        }
        void affiche()
        {
            System.out.println("Identité Personne : ");
            System.out.println(nom+ " " +prénom);
            naissance.affiche();
        }
}
```

Classe ApplicationPersonne :

- ApplicationPersonne est une classe de lancement.
  - Elle sert d'enveloppe à la méthode main qui sert de « programme principal ».

Résultat d'affichage

```
Identité Personne :
Jacques DUPONT
1/2/1947
```

```
class ApplicationPersonne {

    public static void main(String args[ ])
    {
        Personne p = new Personne("Jacques", "DUPONT", 1,2,1947);
        p.affiche();
    }
}
```

5.2 Héritage (1/10)

- **Besoins** : éviter de dupliquer du code (attributs et méthodes) dans différentes classes qui partagent des caractéristiques communes
- faciliter les modifications futures

⇒elles n'ont besoin d'être faites qu'à un seul endroit

Principe :

- L'héritage permet de créer une sous catégorie d'objets à partir d'une autre catégorie (classe) plus générale. Par exemple, Voiture est une sous catégorie de Vehicule. Etudiant est une sous catégorie de Personne.
- Dans ce cas, on dit que la classe fille (Etudiant) est une extension par héritage de sa classe mère (Personne).
- La classe fille reprend toutes les caractéristiques de sa classe mère et y ajoute d'autres plus spécifiques.
  - Par exemple, dans la classe Personne on trouve les attributs nom et prénom.
  - La classe Etudiant ajoute numéro d'inscription en plus de nom et prénom qui sont déjà automatiquement repris par héritage de la classe Personne.

- **En Java, une classe ne peut hériter que d'une seule classe.**
  - Syntaxe :

```
class Héritière extends mère { ... }
```

```
class Etudiant extends Personne { ... }
```

- **Les classes dérivent, par défaut, de java.lang.Object.**

- **Une référence d'une classe C peut contenir des instances de C ou des classes dérivées de C.**

  Personne p = new Etudiant("Jean" , "Dupont", 203435);

- **Exemple :**
  - Une classe mère « Compteur », qui permet de créer des compteurs classiques.
    - Dispose de méthodes d'incrémentation et de décrémentation.

```
class Compteur {
        private  int valeur ;
        Compteur() { valeur = 0 ; }
        void inc( ) {valeur ++ ; }
        void dec( ) {valeur --; }
        int vaut() { return valeur ; }
}
```

Exemple :

- Une classe dérivée « CompteurControlé » de la classe mère « Compteur » permet de créer des compteurs capables de contrôler l'incrémentation des compteurs classiques.

```
class Compteur {
        private  int valeur ;
        Compteur() { valeur = 0 ; }
        void inc( ) {valeur ++ ; }
        void dec( ) {valeur --; }
        int vaut() { return valeur ; }
}
```

```
class Compteur_Controlé extends Compteur
{
        private int maxval;
        Compteur_Controlé (int maxval)
        {
          super();          // explication diapo 5/10
          this.maxval = maxval;
        }
        void inc()          // Redéfinition de la méthode inc
        {
          if( vaut( ) < maxval ) super.inc( );    // diapo 6/10
        }
        int get_maxval( ) { return maxval; }
}
```

Utilisation de « super() » :

```
Compteur_Controlé(int maxval)
{
  super( );
  this.maxval = maxval;
}
```

- Le constructeur de Compteur_Controlé appelle le constructeur de la classe Compteur à l'aide de la fonction super().
- L'appel du constructeur de la classe mère doit être la première instruction du constructeur de la classe dérivée.
- super() représente un appel au constructeur de la classe mère.

Utilisation de « super » et masquage des données :

```
void inc()
{
  if( vaut( ) < maxval ) super.inc( );
}
```

- Une partie du traitement de la méthode inc() de Compteur_Controle peut être effectuée par la méthode inc() de Compteur, c'est pourquoi inc() de Compteur_Controle appelle inc() de Compteur en utilisant super.inc().
- super désigne l'instance en cours de la classe mère.
- La méthode inc() de la classe Compteur_Controle, masque (remplace) la méthode inc() héritée de la classe Compteur.

- **Exemple :**
  - TestControle_Compteur, une classe de lancement.

```
class TestControle_Compteur {
{
        public static void main(String args[])
        {   Compteur_Controle c = new Compteur_Controle(6);
            c.inc( );                    // Première incrémentation
            c.inc( );                   // Deuxième Incrémentation
            System.out.println("valeur : "+c.vaut( ) );
            c.dec( );                    // Première décrémentation
            c.dec( );                   // Deuxième décrémentation
            System.out.println("valeur : "+c.vaut( ) );
        }
}
```

- **Accès à des attributs privés de la classe mère :**

```
class Compteur {
        private  int valeur ;
}       ……
```

```
class Compteur_Controle extends Compteur
{
        ……..
        void inc()
        {
          if( vaut( ) < maxval ) super.inc( );
        }
}
```

  - L'attribut « valeur » est déclaré private.
  - seules les méthodes de la classe Compteur ont accès à l'attribut « valeur ».
  - nécessité d'utiliser la fonction d'accès publique vaut().

- **Accès à des attributs privés de la classe mère :**

```
class Compteur {
        protected  int valeur ;
}       ……
```

```
class Compteur_Controle extends Compteur
{
        ……
        void inc()
        {
          if( valeur < maxval ) super.inc( );
        }
}
```

- L'attribut « valeur » est déclaré protected.
- L'attribut « valeur » est hérité par la classe Compteur_Controle et il est donc accessible par cette classe.

Héritage et conversion : utilisation de « instanceof »

```
Compteur c1 = new Compteur( );
Compteur_Controle c2 = new Compteur_Controle(1000);

int maximum = c1. get_maxval( ) ; // Erreur

System.out.println((c1 instanceof Compteur_Controle));  // false
System.out.println((c1 instanceof Compteur)); // true
System.out.println((c2 instanceof Compteur_Controle));  // true
System.out.println((c2 instanceof Compteur )); // true

c1 = c2;

System.out.println((c1 instanceOf Compteur_Controle));  // true
System.out.println((c1 instanceOf Compteur)); // true

c2 = c1; // Erreur
```

- **Exercice:**

- **Est-ce qu'on peut définir une relation d'héritage entre les classes Animal et Lion?**
- **Est-ce qu'on peut définir une relation d'héritage entre les classes Point et Cercle?**
- **Est-ce qu'on peut définir une relation d'héritage entre les classes Point et PointAvecNom?**

5.5 Les classes abstraites (1/3)

- **Contexte: quel est le problème?**
  - Dans certains cas, certains comportements (méthodes) d'une classe ne peuvent pas être complétements définis
  - Exemple: essayez d'écrire le code de la méthode périmètre de la classe FormeGeometrique
  - Cause: le code est encore flou (abstrait)
- **Solution: développer des classes abstraites**

5.5 Les classes abstraites (2/3)

- Une classe **abstraite** sert à mettre en facteur **des attributs** et **des méthodes** pour des classes dérivées concrètes.
- Une classe **abstraite** est introduite par le mot clé « **abstract** » et elle a **au moins** une méthode **abstraite**.
- Une méthode abstraite est déclarée par le mot clé **abstract**, elle n'a pas de corps.
- Une classe abstraite **ne peut pas** être instanciée (**new**).
- Si une classe dérivée d'une classe abstraite ne redéfinit pas toutes les méthodes abstraites alors elle est aussi abstraite.
- Pour utiliser une classe abstraite on doit définir une classe **héritière** qui fournit les **réalisations** des méthodes abstraites de la classe abstraite.

5.5 Les classes abstraites (3/3)

- **Exemple :**

```
class abstract FormeEuclidienne
{
   public abstract double perimetre();
}
```

```
class Cercle extends FormeEuclidienne
{
   ...
   public double perimetre() { return 2 * Math.PI * r ; }
}
```

```
class Rectangle extends FormeEuclidienne
{
   ...public double perimetre() { return 2 * (height + width); }
}
```

Exemple :

- Chaque case du tableau peut avoir différentes formes possibles → polymorphisme

```
FormeEuclidienne[ ] formes = {  new Circle(2),
                                new  Rectangle(2,3),
                                ……..
                               };
```

- L'avantage du polymorphisme est de pouvoir définir un code générique (applicable sur tous les cas particuliers)

```
double  somme_perimetres = 0;

for (int i = 0; i < formes.length; i++)

     somme_perimetres += formes[i].perimetre( );
```

- **Principe de l'interface :**
  - Une interface est introduite par le mot clé «interface», et se comporte *comme une classe* dont toutes les méthodes sont abstract et dont tous les attributs sont final.
  - Les mots clés final et abstract n'apparaissent pas dans la définition d'une interface.
  - Une interface est un moyen de préciser les services qu'une classe peut rendre. C'est un modèle d'implémentation.
  - Une interface est inutilisable en elle-même.
  - Une classe doit implémenter l'interface, c'est à dire définir les corps des méthodes abstraites de l'interface.

- **Une interface est utilisée lorsque nous avons seulement besoin de savoir qu'une classe doit avoir les méthodes déclarées dans l'interface, exemple :**
  - On peut imaginer une interface appareilElectrique qui définit les méthodes estEnclenche() et alimente(boolean).
  - Si une classe implémente l'interface appareilElectrique, on sait donc qu'elle possède au moins ces deux méthodes, qu'il s'agisse d'une radio, d'une lampe ou d'un autre appareil.
  - Lorsque l'on désire indiquer à un appareil qu'il est alimenté, on exécute app.alimente(true), et pour lui indiquer qu'il a été débranché, on exécute app.alimente(false).

- **Exemple d'interface :**
  - Appareil_Electrique

```
public interface Appareil_Electrique
{
   /** teste si l'appareil est enclenché */
   public boolean estEnclenche( );

  /** on appelle cette methode lorsque l'on branche  l'appareil dans une source de courant active avec true, ou false si la source est inactive */
public void alimente(boolean alim);

}
```

- **Exemple d'utilisation de l'interface Appareil_Electrique.**
  - classe Radio.

```
class Radio implements Appareil_Electrique
{
  final static int FREQ_INIT= 1007; // 100.7 MHz
  int freq;
  boolean allumee = false;
  public boolean estEnclenche()    { return allumee; }

  public void alimente(boolean a)
  {
     allumee = a;
     if (allumee) freq = FREQ_INIT;
  }
 public boolean changeFreq(int freq)
 {
    if (allumee == true) { this.freq = freq; return allumee; }
    return false;
 }
```

- **Le mot clé « implements » indique que la classe Radio doit forcément donner une définition précise pour les méthodes spécifiées dans l'interface.**
- **Une classe peut implémenter plusieurs interfaces.**
  - Elle doit dans ce cas fournir des définitions pour toutes les méthodes spécifiées dans l'interface.

```
interface Truc
{
   void e( );
}
```

```
interface
      Bidule
{
   void f( );
}
```

```
class Machin implements Truc, Bidule
{
   void e( ) {- - -; }
   void f( ) {- - -; }
}
```

- **Plusieurs <u>classes différentes</u> peuvent implémenter de manières différentes une <u>même interface</u>.**
  - Plusieurs réalisations possibles d'une même spécification (polymorphisme).

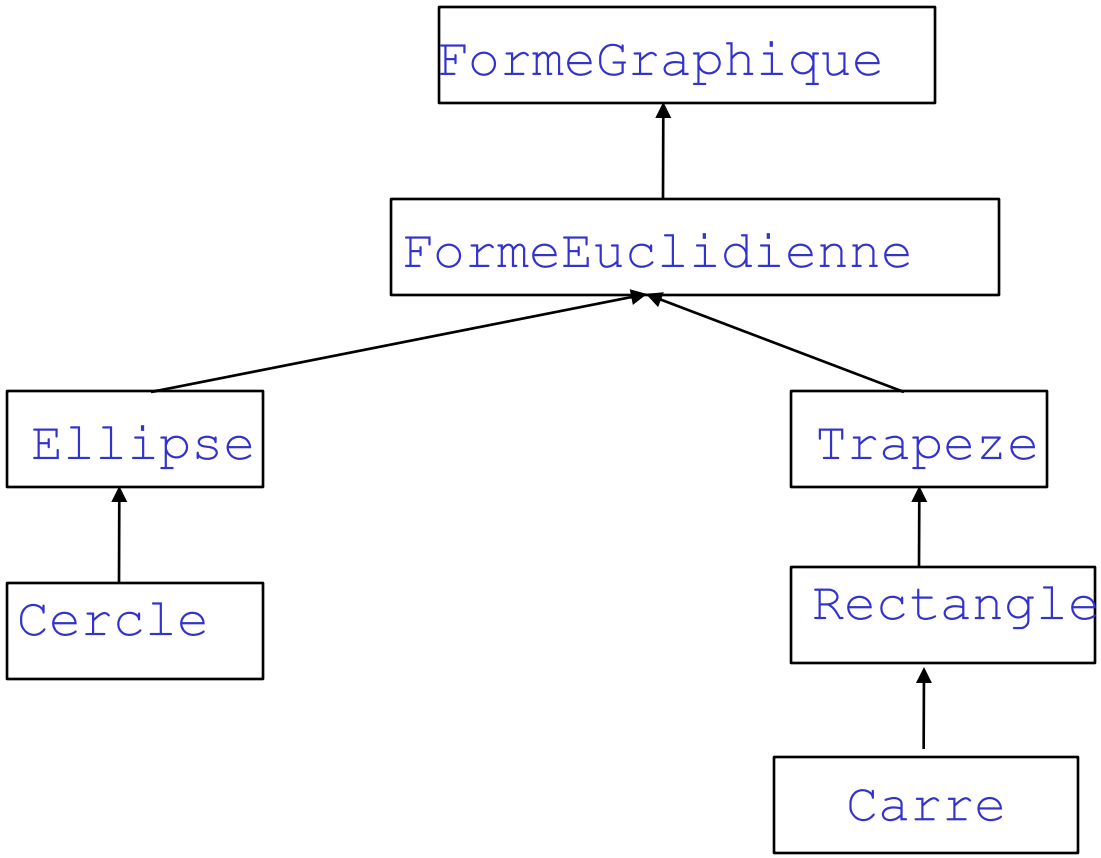
Rôles des interfaces :

- Conceptuel : factoriser un comportement commun entre plusieurs classes différentes. Ce comportement est représenté par un ensemble de méthodes.
  - Exemple 1 : interface Dessinable qui contient la méthode dessiner (factorisée entre tous les objets dessinables (Cercle, Maison…)
  - Exemple 2 : interface Deplaçable qui contient la méthode déplacer (factorisée entre tous les objets qu'on peut déplacer (Voiture, Personne…)
- Technique : obliger le développeur à implémenter toutes les méthodes définies dans l'interface. Ainsi, il doit respecter la signature de ces méthodes → utile pour la programmation à plusieurs. Ex : affich, affiche(), afficher()…

Exercice

L'héritage - Hiérarchie

Soit l'hiérarchie suivante:



Exercice

- Ecrire le code des classes suivantes:
  - l'interface Affichable qui contient une méthode afficher()
  - FormeEuclidienne qui implémente l'interface Affichable et qui contient les méthodes double perimetre() et double surface()
  - *Ellipse* et *Trapeze* qui héritent de la classe *FormeEuclidienne*
  - *Cercle qui hérite de Ellipse*
  - *Rectangle qui hérite de Trapèze*
  - *Carre qui hérite de Rectangle*
  - *GestionFormes qui contient les méthodes statiques:*
    - *void afficheSurfaces(FormeEuclidienne [] tab) permettant d'afficher la surface des formes du tableau tab en utilisant la méthode surface*
    - *void afficheTout(Affichable[] tab) permettant d'afficher les éléments du tableau tab*
    - *main qui permet de tester les méthodes précédentes*

Indications

- Perimetre Ellipse = (axe1+axe2) * Math.PI
- Surface Ellipse = axe1 * axe2 * Math.PI
- Perimetre Trapeze = base1 + base2 + cote1 + cote2
- Surface Trapeze = (base1+base2) * hauteur/2