Cours JAVA - FMS

Initiation à la programmation

avec le langage Java

Partie 2 : Les Concepts Objets

Stéphane MASCARON
Architecte Logiciels Libres
http://mascaron.net

Cours JAVA - FMS

- La programmation orientée objet
  - L'idée de base de la programmation orientée objet est de rassembler dans une même entité appelée objet les données et les traitements qui s'y appliquent.
  - Pour cela on utilise le concept de **Classe**, qui représente le modèle utilisé pour construire les objets.

Cours JAVA - FMS

- Le concept de classe
  - Une classe est le support de l'encapsulation :
    - C'est un ensemble de données et de fonctions regroupées dans une même entité.
    - Une classe est une description abstraite d'un objet.
    - Les fonctions qui opèrent sur les données sont appelées des méthodes.
    - Instancier une classe consiste à créer un objet sur son modèle.
    - Une classe comporte sa déclaration, des variables et les définitions de ses méthodes.

Cours JAVA - FMS

- Le concept de classe (suite)
  - Une classe se compose de deux parties : un en-tête et un corps :
    - Le corps peut être divisé en 2 sections :
      - la déclaration des données et des constantes
      - la définition des méthodes.

NB : Les méthodes et les données sont pourvues d'attributs de visibilité qui gèrent leur accessibilité par les composants hors de la classe.

Cours JAVA - FMS

- La syntaxe de déclaration d'une classe

```
modificateurs class NomDeClasse [extends classe_mere] [implements interfaces] {...}
```

  - Les modificateurs de classe (ClassModifiers) sont :

Modificateur	Rôle
abstract	La classe contient une ou des méthodes abstraites, qui n'ont pas de définition explicite. Une classe déclarée abstract ne peut pas être instanciée : il faut définir une classe qui hérite de cette classe et qui implémente les méthodes nécessaires pour ne pas être abstraite.
final	La classe ne peut pas être modifiée, sa redéfinition grâce à l'héritage est interdite. Les classes déclarées final ne peuvent donc pas avoir de classes filles.
private	La classe n'est accessible qu'à partir du fichier où elle est définie
public	La classe est accessible partout

Cours JAVA - FMS

- La syntaxe de déclaration d'une classe (suite)
  - Les modificateurs abstract et final ainsi que public et private sont mutuellement exclusifs.
  - Marquer une classe comme « **final** » peut permettre au compilateur et à la JVM de réaliser quelques petites optimisations.
  - Le mot clé « **extends** » permet de spécifier une **superclasse** éventuelle : ce mot clé permet de préciser la classe mère dans une relation d'héritage.
  - Le mot clé « **implements** » permet de spécifier une ou des interfaces que la classe implémente. Cela permet de récupérer quelques avantages de l'héritage multiple et de mettre en œuvre le polymorphisme.

Cours JAVA - FMS

- Les objets
  - Ils contiennent des attributs et des méthodes.
  - Les attributs sont des variables ou des objets nécessaires au fonctionnement de l'objet.
  - En Java, une application est un objet.
  - La classe est la description d'un objet.
  - Un objet est une instance d'une classe.
  - Pour chaque instance d'une classe, le code est le même, seules les données sont différentes à chaque objet.

Cours JAVA - FMS

- La création d'un objet : instancier une classe
  - Il est nécessaire de définir la déclaration d'une variable ayant le type de l'objet désiré. La déclaration est de la forme :

```
NomDeClasse nomDeVariable ;
```

  - L'opérateur new se charge de créer une instance de la classe et de l'associer à la variable.

Exemple

```
MaClasse m = new MaClasse();
```

  - Chaque instance d'une classe nécessite sa propre variable. Plusieurs variables peuvent désigner un même objet.

Cours JAVA - FMS

- La création d'un objet : instancier une classe (suite)
  - En Java, tous les objets sont instanciés par allocation dynamique.
  - Dans l'exemple, la variable m contient une référence sur l'objet instancié, c'est à dire l'adresse mémoire de l'objet qu'elle désigne.
  - Attention toutefois, il n'est pas possible de manipuler ou d'effectuer des opérations directement sur cette adresse mémoire comme en C.
    - Si m2 désigne un objet de type MaClasse, l'instruction **m2 = m** ne définit pas un nouvel objet mais m et m2 désignent tous les deux le même objet.

Cours JAVA - FMS

- L’opérateur new
  - Est un opérateur de haute priorité qui permet d'instancier des objets et d'appeler une méthode particulière de cet objet : le **constructeur**.
  - Il fait appel à la machine virtuelle pour obtenir l'**espace mémoire nécessaire** à la représentation de l'objet puis appelle le **constructeur** pour initialiser l'objet dans l'emplacement obtenu.
  - Il renvoie une valeur qui référence l'objet instancié.
  - Si l'opérateur new n'obtient pas l'allocation mémoire nécessaire, il lève l'exception **OutOfMemoryError.**

Cours JAVA - FMS

- Remarque sur les objets de type String :
  - Un objet String est automatiquement créé lors de l'utilisation d'une constante chaîne de caractères sauf si celle-ci est déjà utilisée dans la classe.

Exemple

```
public class TestChaines1 {
    public static void main(String[] args) {
        String chaine1 = "bonjour";
        String chaine2 = "bonjour";
        // on teste ici l'égalité des références en mémoire pas la valeur
        System.out.println("(chaine1 == chaine2) = " + (chaine1 == chaine2) );
    }
}
```

```
Resultat :
(chaine1 == chaine2) = true  // donc c'est bien le même objet
```

  - Pour obtenir une seconde instance de la chaine, il faut explicitement demander sa création en utilisant l'opérateur **new**. (Exo : écrivez-le, quel est le résultat)

Cours JAVA - FMS

- La durée de vie d'un objet
  - Les objets ne sont pas des éléments statiques et leur durée de vie ne correspond pas forcément à la durée d'exécution du programme.
    - La durée de vie d'un objet passe par trois étapes :
      - La déclaration de l'objet et l'instanciation grâce à l'opérateur new
      - L'utilisation de l'objet en appelant ses méthodes
      - La suppression de l'objet : elle est automatique en Java grâce à la machine virtuelle. La restitution de la mémoire inutilisée est prise en charge par le récupérateur de mémoire (garbage collector). Il n'existe pas d'instruction delete comme en C++.

Cours JAVA - FMS

- Les références et la comparaison d'objets
  - Les variables de type objet que l'on déclare ne contiennent pas un objet mais une référence vers cet objet.
  - Lorsque l'on écrit c1 = c2 (c1 et c2 sont des objets), on copie la référence de l'objet c2 dans c1.
  - c1 et c2 font référence au même objet : ils pointent sur le même objet.
  - L'opérateur == compare ces références. Deux objets avec des propriétés identiques sont deux objets distincts :

Exemple

```
Rectangle r1 = new Rectangle(100,50);
Rectangle r2 = new Rectangle(100,50);
if (r1 == r1) { ... } // vrai
if (r1 == r2) { ... } // faux
```

Cours JAVA - FMS

- Pour s'assurer que deux objets sont de la même classe, il faut utiliser la méthode getClass() de la classe Object dont toutes les classes héritent.

Exemple
boolean egaux = (obj1.getClass().equals(obj2.getClass()) ;

- Le littéral « null » :
  - Le littéral « null » est utilisable partout où il est possible d'utiliser une référence à un objet.
  - Il n'appartient pas à une classe mais il peut être utilisé à la place d'un objet de n'importe quelle type ou comme paramètre.
  - « null »ne peut pas être utilisé comme un objet normal : il n'y a pas d'appel de méthodes et aucune classe ne peut en hériter.
  - Le fait d'affecter « null » à une variable référençant un objet pourra permettre au ramasse-miettes de libérer la mémoire allouée à l'objet si aucune autre référence n'existe encore sur lui.

Cours JAVA - FMS

- Les variables de classes
  - Elles ne sont définies qu'une seule fois quel que soit le nombre d'objets instanciés de la classe.
  - Leur déclaration est accompagnée du mot clé static

Exemple

```
public class MaClasse() {
    static int compteur = 0;
}
```

  - L'appartenance des variables de classe à une classe entière et non à un objet spécifique permet de remplacer le nom de la variable par le nom de la classe.

Exemple

```
MaClasse m = new MaClasse();
int c1 = m.compteur;
int c2 = MaClasse.compteur;
// c1 et c2 possèdent la même valeur.
```

Cours JAVA - FMS

- La variable this
  - Cette variable sert à référencer dans une méthode l'instance de l'objet en cours d'utilisation. **this** est un objet qui est égal à l'instance de l'objet dans lequel il est utilisé.

Exemple

```
private int nombre;
public maclasse(int nombre) {
      nombre = nombre;
      // variable de classe = variable en paramètre du constructeur
}
```

Exemple

```
(…)
 this.nombre = nombre;
(…)
```

Cours JAVA - FMS

- L'opérateur instanceof
  - L'opérateur instanceof permet de déterminer la classe de l'objet qui lui est passé en paramètre. La syntaxe est objet instanceof classe :

Exemple

```
void testClasse(Object o) {
    if (o instanceof MaClasse )
      System.out.println(" o est une instance de la classe MaClasse ");
    else System.out.println(" o n'est pas un objet de la classe MaClasse ");
}
```

  - Dans le cas ci-dessus, même si o est une instance de MaClasse, il n'est pas permis d'appeler une méthode de MaClasse car o est de type Object.

Exemple

```
System.out.println(o.getChaine());  // erreur à la compil
```

Cours JAVA - FMS

- L'opérateur instanceof (suite)
  - Pour résoudre le problème, il faut utiliser la technique du casting (conversion).

Exemple

```
void afficheChaine(Object o) {
    if (o instanceof MaClasse){
        MaClasse m = (MaClasse) o;
        System.out.println(m.getChaine());
        // OU System.out.println( ((MaClasse) o).getChaine() );
    }
}
```

Cours JAVA - FMS

- Les modificateurs d'accès
  - Ce sont les mots réservés public, protected, private, static, final, abstract, synchronized, volatile, native qui se placent devant :
    - Les classes, les méthodes et les attributs (variables globales à la classe)

NB : Ils ne peuvent pas être utilisés pour qualifier des variables locales aux méthodes. Seules les variables d'instance et de classe peuvent en profiter.

  - Ils assurent le contrôle des conditions d'héritage, d'accès aux éléments et de modification de données par les autres objets.

Cours JAVA - FMS

- Les mots clés qui gèrent la visibilité des entités
  - Les attributs de visibilité réglementent l'accès aux classes et aux objets, aux méthodes et aux données.
  - Il existe 3 modificateurs qui peuvent être utilisés pour définir les attributs de visibilité des entités (classes, méthodes ou attributs) : public, private, protected.
  - Leur utilisation permet de définir des niveaux de protection différents, présentés dans un ordre croissant de niveau de protection offert dans le tableau suivant :

Cours JAVA - FMS

Modificateur	Rôle
public	Une variable, méthode ou classe déclarée public est visible par tous les autres objets. Depuis la version 1.0, une seule classe public est permise par fichier et son nom doit correspondre à celui du fichier. Dans la philosophie orientée objet aucune donnée d'une classe ne devrait être déclarée publique : il est préférable d'écrire des méthodes pour la consulter et la modifier
Par défaut, sans modificateur	Il n'existe pas de mot clé pour définir ce niveau, qui est le niveau par défaut lorsqu'aucun modificateur n'est précisé. Cette déclaration permet à une entité (classe, méthode ou variable) d'être visible par toutes les classes se trouvant dans le même package. (protected des classes)
protected	Si une méthode ou une variable est déclarée protected, seules les méthodes présentes dans le même package que cette classe ou ses sous-classes pourront y accéder. **On ne peut pas qualifier une classe avec protected.**
private	C'est le niveau de protection le plus fort. Les composants ne sont visibles qu'à l'intérieur de la classe : ils ne peuvent être modifiés que par des méthodes définies dans la classe et prévues à cet effet. Les méthodes déclarées private ne peuvent pas être en même temps déclarées abstract car elles ne peuvent pas être redéfinies dans les classes filles.

Cours JAVA - FMS

- Le mot clé static
  - Le mot clé static s'applique aux variables et aux méthodes.
  - Les variables d'instance sont des variables propres à un objet. Il est possible de définir une variable de classe qui est partagée entre toutes les instances d'une même classe : elle n'existe donc qu'une seule fois en mémoire.
  - Une telle variable permet de stocker une constante ou une valeur modifiée tour à tour par les instances de la classe. Elle se définit avec le mot clé static.

Cours JAVA - FMS

- Le mot clé static

Exemple d'une classe

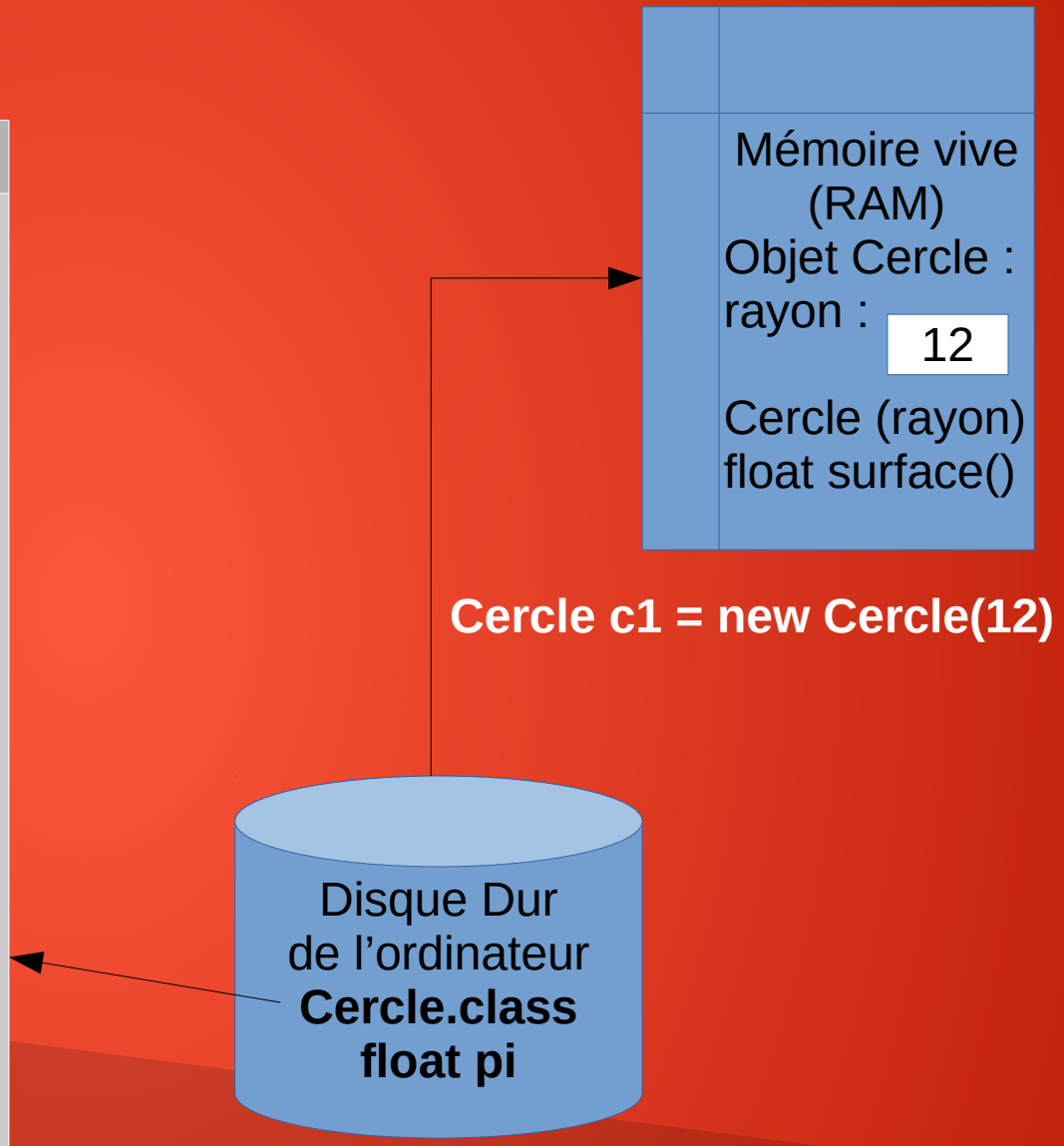
```
public class Cercle {

        // variable de classe
        static float pi = 3.1416f;

        // variable d'instance
        float rayon;

        //méthode d'instance *
        public Cercle(float rayon) {
            this.rayon = rayon;
        }

        //méthode d'instance
        public float surface() {
            return rayon * rayon * pi;
        }
}
```



Cours JAVA - FMS

- Le mot clé static (suite)
  - Une méthode static est une méthode qui n'agit pas sur des variables d'instance mais uniquement sur des variables de classe.
  - Ces méthodes peuvent être utilisées sans instancier un objet de la classe.

Exemple d'utilisation de notre classe Cercle

```
(…)
      public static void main (String [] args) {
          Cercle c1 = new Cercle(12) ;
          float laSurface = c1.surface() ;
          System.out.println("Pour la valeur de PI : " + Cercle.pi);
          System.out.println("Surface du cercle : " + laSurface);
      }
```

NB : Il n'est pas possible d'appeler une méthode d'instance ou d'accéder à une variable d'instance depuis une méthode de classe statique.

Cours JAVA - FMS

- Le mot clé **final**
  - Le mot clé final s'applique aux variables de classe ou d'instance ou locales, aux méthodes, aux paramètres d'une méthode et aux classes.
  - Il permet de rendre l'entité sur laquelle il s'applique non modifiable une fois qu'elle est déclarée pour une méthode ou une classe et initialisée pour une variable.
  - Une variable qualifiée de final signifie que la valeur de la variable ne peut plus être modifiée une fois que celle-ci est initialisée.

Cours JAVA - FMS

- Le mot clé **final (suite)**
  - Une vérification est opérée par le compilateur.

Exemple d'utilisation de final

```
public class Constante1 {
     public static final int constante = 0;

     public Constante1() {
          Constante = 10;  //tentative modification variable finale
     }
}
```

Résultat

```
$ javac Constante1.java
Constante1.java:6: cannot assign a value to final variable constante
constante = 10;
^
1 error
```

NB : Les constantes sont qualifiées avec les modificateurs final et static. (**`public static final float PI = 3.141f`**)

Cours JAVA - FMS

- Le mot clé **final** (suite) :
  - Une méthode déclarée final ne peut pas être redéfinie dans une sous-classe.
  - Lorsque le modificateur final est ajouté à une classe, il est interdit de créer une sous-classe pour cette classe finale.
  - Cela peut être fait dans un souci d'efficacité ou pour empêcher l'utilisateur d'en faire une utilisation non appropriée dans une classe dérivée.
  - De nombreuses classes de la bibliothèque standard Java sont déclarées comme **final** telles que java.lang.System ou java.lang.String.

Cours JAVA - FMS

- Le mot clé **abstract** :
  - Le mot clé abstract s'applique aux méthodes et aux classes.
  - **Abstract** indique que la classe ne pourra être instanciée telle quelle. De plus, toutes les méthodes de cette classe abstract ne sont pas implémentées et devront être redéfinies par des méthodes complètes dans ses sous-classes.
  - **Abstract** permet de créer une classe qui sera une sorte de « patron ». Toutes les classes dérivées pourront profiter des méthodes héritées et n'auront à implémenter que les méthodes déclarées abstract.

Cours JAVA - FMS

- Le mot clé **abstract** :

Exemple d'utilisation de abstract

```
abstract class FiguresGeometriques {
     public static final float PI = 3.14159f;
     abstract void dessiner() ;  // méthode asbtraite
     abstract double surface() ; // méthode asbtraite
}
public class Rectangle extends FiguresGeometriques {
     private double largeur ;
     private double longueur ;
     //Constructeur de la classe que l'on verra plus loin
     public Rectangle(double longueur, double largeur) {
        this.longueur = longueur ;
        this.largeur = largeur ;
     }
     public void dessiner() {
          //ici le code permettant de dessiner un rectagle
     }
     public double surface() {
          return this.longueur * this.largeur ;
     }
} //fin de la classe rectangle
```

Cours JAVA - FMS

- Exercice sur le mot clé **abstract** :
  - Ecrire une classe Cercle qui hérite de la classe « FiguresGeometriques » permettant de calculer la surface d'un cercle grâce à l'implémentation de la méthode abstraite « surface ».

Exemple d'utilisation de abstract

```
abstract class FiguresGeometriques {
     public static final float PI = 3.14159f;
     abstract void dessiner() ;  // méthode asbtraite
     abstract double surface() ; // méthode asbtraite
} //fin de la classe
```

Cours JAVA - FMS

- Exercice :
  - Ecrire une classe Personne permettant de gérer des individus avec :
    - nom, prénom, age, adresse1, adresse2, codePostal, ville, estMarie, nbEnfants
  - Vous devrez définir les types de données adaptées aux attributs, implémenter la classe Personne, et les methodes d'accès aux attributs (getter, setter), ainsi qu'une methode permettant de calculer l'année de naissance nommée « getAnneeNaissance() ».

Cours JAVA - FMS

- Le mot clé **abstract** (suite) :
  - Une méthode abstraite est une méthode déclarée avec le modificateur abstract et sans corps.
  - Elle correspond à une méthode dont on veut forcer l'implémentation dans une sous-classe.
  - Une classe sans définition explicite d'une ou des méthodes abstraites et sans le modificateur abstract génère une erreur de compilation.
  - Une classe est automatiquement abstraite dès qu'une de ses méthodes est déclarée abstraite.

Cours JAVA - FMS

- Le mot clé **synchronized**
  - Il permet de gérer l'accès concurrent aux variables et méthodes lors de traitements de threads (exécution « simultanée » de plusieurs petites parties de code du programme). On le verra plus en détail lors de l'étude des Threads.
- Le mot clé **volatile**
  - Le mot clé volatile s'applique aux variables.
  - L'utilisation du mot clé volatile force l'écriture de la valeur d'une variable en mémoire ainsi que sa relecture : cela permet de garantir que la lecture de la donnée par un thread retournera la valeur la plus récente en mémoire. Le mot clé volatile ne réalise aucune opération pour garantir la gestion des accès concurrents : elle offre juste une garantie sur la visibilité.

Cours JAVA - FMS

- Le mot clé **native**
  - Une méthode native est une méthode qui est implémentée dans un autre langage. L'utilisation de ce type de méthode limite la portabilité du code mais permet une vitesse d'exécution plus rapide.
  - C’est hors du champ du cours initiation, car nécessite la mise en œuvre de Java Native Interface pour accéder à des codes assembleur, C ou C++.
  - CurrentTimesMillis() est une méthode native.

Cours JAVA - FMS

- Les propriétés ou attributs
  - Les variables d'instances
    - Une variable **d'instance** nécessite simplement une déclaration de la variable dans le corps de la classe.

Exemple d'utilisation de notre classe Cercle

```
(…)
public class MaClasse {
    public int valeur1 ;
    int valeur2 ;
    protected int valeur3 ;
    private int valeur4 ;
}
```

    - Chaque instance de la classe a accès à sa propre occurrence de la variable.

Cours JAVA - FMS

- Les variables de classes
  - Les variables de classes sont définies avec le mot clé static

Exemple

```
(…)
public class MaClasse {
    static int compteur ;
    static float PI = 3.14159f ;
}
```

```
  float surface = MaClasse.PI * rayon * rayon ;
```

  - Chaque instance de la classe partage la même variable.
  - Pour accéder à la variable on utilise la Classe à la place de l'instance.

Cours JAVA - FMS

- Les constantes
  - Les constantes sont définies avec le mot clé final : leur valeur ne peut pas être modifiée une fois qu'elles sont initialisées.

Exemple

```
(…)
public class MaClasse {
    static final float PI = 3.14159f ;
}
```

```
  float surface = MaClasse.PI * rayon * rayon ;
```

Cours JAVA - FMS

- Les **Méthodes** : sont des fonctions qui implémentent les traitements de la classe.
  - La syntaxe de la déclaration

```
modificateurs type nom_méthode (type arg1, ... ) {
 // définition des variables locales
 // et du bloc d'instruction
}
```

  - Le type retourné peut être élémentaire ou correspondre à un objet. Si la méthode ne retourne rien, alors on utilise void.
  - Le type et le nombre d'arguments déclarés doivent correspondre au type et au nombre d'arguments transmis. Il n'est pas possible d'indiquer des valeurs par défaut dans les paramètres.

Cours JAVA - FMS

- Les **Méthodes** (suite)
  - Les arguments sont passés par valeur : la méthode fait une copie de la variable qui lui est locale.
  - Lorsqu'un objet est transmis comme argument à une méthode, cette dernière reçoit une référence qui désigne l'emplacement mémoire d'origine de l'objet et qui est une copie de la variable.
  - Il est possible de modifier l'objet grâce à ses méthodes mais il n'est pas possible de remplacer la référence contenue dans la variable

Cours JAVA - FMS

- Les modificateurs de méthodes

Modificateur	Rôle
public	La méthode est accessible aux méthodes des autres classes
private	L'usage de la méthode est réservé aux autres méthodes de la même classe.
protected	la méthode ne peut être invoquée que par des méthodes de la classe ou de ses sous-classes
final	La méthode ne peut être modifiée (redéfinition lors de l'héritage interdite)
static	La méthode appartient simultanément à tous les objets de la classe (comme une constante déclarée à l'intérieur de la classe). Il est inutile d'instancier la classe pour appeler la méthode mais la méthode ne peut pas manipuler de variable d'instance. Elle ne peut utiliser que des variables de classes.
synchronized	la méthode fait partie d'un thread. Lorsqu'elle est appelée, elle barre l'accès à son instance. L'instance est à nouveau libérée à la fin de son exécution.
native	le code source de la méthode est écrit dans un autre langage

Cours JAVA - FMS

- Les modificateurs de méthodes (suite)
  - Sans modificateur, la méthode peut être appelée par toutes autres méthodes des classes du package auquel appartient la classe.
  - La valeur de retour de la méthode doit être transmise par l'instruction **return**. Elle indique la valeur que prend la méthode et termine celle-ci : toutes les instructions qui suivent return sont donc ignorées.

Exemple

```
int add(int a, int b) {
      return a + b;
}
```

Cours JAVA - FMS

- Les modificateurs de méthodes (suite)
  - Il est possible d'inclure une instruction **return** dans une méthode de type **void** : cela permet de **quitter la méthode**.
  - La méthode **main()** de la classe principale d'une application **doit être déclarée** de la façon suivante :

Exemple

```
public class MonApp1 {
	public static void main(String[] args) {
		System.out.println("Bonjour");
	}
}
```

  - Cette déclaration de la méthode **main()** est **imposée par la machine virtuelle** pour reconnaitre le point d'entrée d'une application.

Cours JAVA - FMS

- Les modificateurs de méthodes (suite)
  - Si la déclaration de la méthode **main() diffère**, **une exception sera levée** lors de la tentative d'exécution par la machine virtuelle.

Exemple

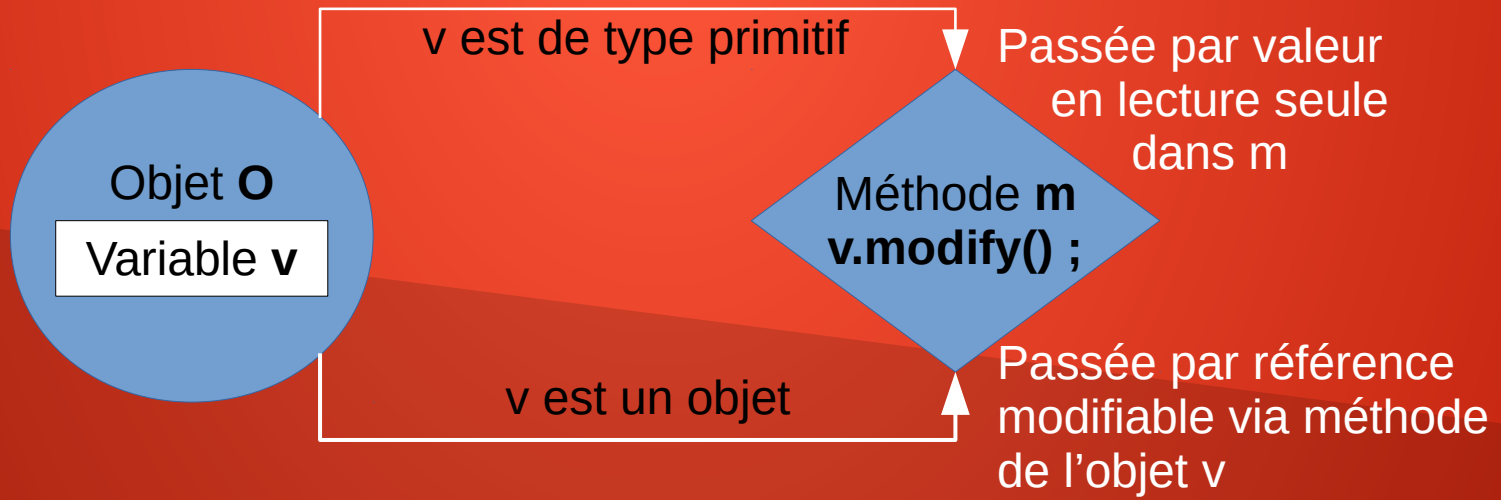
```
public class MonApp2 {
     public static int main(String[] args) {
         System.out.println("Bonjour");
         return 0;
     }
}
```

Résutat :

```
$ javac MonApp2.java
$ java MonApp2
Exception in thread "main" java.lang.NoSuchMethodError: main
```

Cours JAVA - FMS

- La transmission de paramètres
  - Lorsqu'un objet est passé en paramètre, ce n'est pas l'objet lui-même qui est passé mais une référence sur l'objet.
  - **La référence est bien transmise par valeur et ne peut pas être modifiée** mais l'objet peut être modifié par un **message** (appel d'une méthode).



Cours JAVA - FMS

- L'émission de messages
  - Un message est émis lorsqu'on demande à un objet d'exécuter l'une de ses méthodes.
  - La syntaxe d'appel d'une méthode est :

```
identificateur_objet.nom_méthode(param, ... ) ;
```

  - Si la méthode appelée ne contient aucun paramètre, il faut laisser les parenthèses vides :

```
identificateur_objet.nom_méthode() ;
```

Cours JAVA - FMS

- L'enchaînement de références à des variables et à des méthodes :

Exemple

```
public class MonApp1 {
     public static void main(String[] args) {
          System.out.println("Bonjour");
     }
}
```

  - Deux classes sont impliquées dans l'instruction : **System** et **PrintStream** :
    - La classe **System** possède une variable nommée **out** qui est un objet de type **PrintStream**.
    - **println()** est une méthode de la classe **PrintStream**.

L'instruction signifie : « utilise la méthode **println()** de la variable **out** de type **PrintStream** de la classe **System** ».

Cours JAVA - FMS

- La surcharge de méthodes
  - La surcharge d'une méthode permet de définir plusieurs fois une même méthode avec des arguments différents.
  - Le compilateur choisi la méthode qui doit être appelée en fonction du nombre et du type des arguments. (appelé « signature » d'une méthode).
  - Ceci permet de simplifier l'interface des classes vis à vis des autres classes.
  - Une méthode est surchargée lorsqu'elle exécute des actions différentes selon le type et le nombre de paramètres transmis.

Cours JAVA - FMS

- La surcharge de méthodes (suite)
  - Il est donc possible de donner le même nom à deux méthodes différentes à condition que les signatures de ces deux méthodes soient différentes.
  - La signature d'une méthode comprend :
    - le nom de la classe,
    - le nom de la méthode,
    - les types des paramètres.
  - Il n'est pas possible d'avoir deux méthodes de même nom dont tous les paramètres sont identiques et dont seul le type retourné diffère.

Cours JAVA - FMS

Exemple : surcharge de la méthode afficheValeur (...)

```
class affiche{
     public void afficheValeur(int i) {
          System.out.println(" nombre entier = " + i);
     }
     public void afficheValeur(float f) {
          System.out.println(" nombre flottant = " + f);
     }
}
```

Exemple

```
class affiche{
     public float convert(int i){
          return((float) i);
     }
     public double convert(int i){
          return((double) i);
     }
}
```

```
$ javac Affiche.java
Affiche.java:5: Methods can't be redefined with a different return
type: double convert(int) was float convert(int)
public double convert(int i){
^
1 error
```

Cours JAVA - FMS

- Les constructeurs
  - La déclaration d'un objet est suivie d'une sorte **d'initialisation** par le moyen d'une méthode particulière appelée **constructeur** pour que les **variables** aient une **valeur de départ**. Elle est invoquée lors de la création d'un objet.
  - Le **constructeur** suit la définition des autres méthodes excepté que **son nom doit obligatoirement correspondre à celui de la classe** et qu'il **n'est pas typé**, **pas même void**, donc il ne peut **pas y avoir d'instruction return** dans un constructeur.
  - On peut surcharger un constructeur.

Cours JAVA - FMS

- Les constructeurs
  - La définition d'un **constructeur** est **facultative**. Si aucun constructeur n'est explicitement défini dans la classe, **le compilateur va créer un constructeur** par défaut sans argument.
  - Dès qu'un constructeur est explicitement défini, le compilateur considère que le programmeur prend en charge la création des constructeurs et que le mécanisme par défaut, qui correspond à un constructeur sans paramètres, n'est pas mis en oeuvre.
  - Si on souhaite un cnstructeur sans paramètres, il faut définir explicitement un constructeur sans paramètres en plus des autres constructeurs avec paramètres.

Cours JAVA - FMS

- Les constructeurs (suite)
  - Il existe plusieurs manières de définir un constructeur :
    1. Le constructeur simple : ce type de constructeur ne nécessite pas de définition explicite : son existence découle automatiquement de la définition de la classe.
    2. Le constructeur avec initialisation fixe : il permet de créer un constructeur par défaut.
    3. Le constructeur avec initialisation des variables : pour spécifier les valeurs de données à initialiser on peut les passer en paramètres au constructeur

Cours JAVA - FMS

- Les constructeurs (suite)

Exemple 1 : le constructeur simple : il y a rien a faire c'est automatique

```
public MaClasse() { … }
```

Exemple 2 : le constructeur avec initialisation fixe

```
public MaClasse() {
      Nombre = 5;
}
```

Exemple 3 : le constructeur avec initialisation des variables

```
public MaClasse(int nombre) {
      this.nombre = nombre;
}
```

Cours JAVA - FMS

- Le destructeur
  - Un destructeur permet d'exécuter du code lors de la libération, par le garbage collector, de l'espace mémoire occupé par l'objet.
  - En Java, les destructeurs appelés finaliseurs (finalizers), sont automatiquement invoqués par le garbage collector.
  - Pour créer un finaliseur, il faut redéfinir la méthode **finalize()** héritée de la classe **Object**.

Cours JAVA - FMS

- Les accesseurs & l’encapsulation
  - L'encapsulation permet de sécuriser l'accès aux données d'une classe.
  - Ainsi, les données déclarées private à l'intérieur d'une classe ne peuvent être accédées et modifiées que par des méthodes définies dans la même classe.
  - Si une autre classe veut accéder aux données de la classe, l'opération n'est possible que par l'intermédiaire d'une méthode de la classe prévue à cet effet.
  - Un accesseur est une méthode publique qui donne l'accès à une variable d'instance privée.
  - Par convention, les accesseurs en lecture commencent par get et les accesseurs en écriture commencent par set.

Cours JAVA - FMS

- Les accesseurs & l'encapsulation

Exemple :

```
public class MaClasse

     private int valeur = 13;

     public int getValeur(){
          return(valeur);
     }

     public void setValeur(int val) {
          valeur = val;
     }
}
```

- Pour un attribut de type booléen, il est possible de faire commencer l'accesseur en lecture par is au lieu de get.

Cours JAVA - FMS

- Exercice :
  - Ecrire une classe java qui permet de modéliser un employé qui possède :
    - Un Nom, prénom, date de naissance, telephone, adresse, code_postal, ville, libelle_poste, date_entrée, type_employé, et les autres infos que vous trouverez pertinentes.
    - Les attributs (variables d'instance) seront privée et des accesseurs permettrons de les lire ou les modifier.

Cours JAVA - FMS

- L'héritage
  - L'héritage est un mécanisme qui facilite la **réutilisation du code** et la gestion de **son évolution**. Elle définit une relation entre deux classes :
    - Une classe mère ou super-classe
    - Une classe fille ou sous-classe qui hérite de sa classe mère
- Le principe de l'héritage
  - Grâce à l'héritage, les objets d'une classe fille ont accès aux données et aux méthodes de la classe parente et peuvent les étendre.
  - Les sous-classes peuvent redéfinir les variables et les méthodes héritées.

Cours JAVA - FMS

- Le principe de l'héritage
  - Pour les variables, il suffit de les redéclarer sous le même nom avec un type différent.
  - Les méthodes sont redéfinies avec le même nom, les mêmes types et le même nombre d'arguments, sinon il s'agit d'une surcharge.
  - L'héritage successif de classes permet de définir une hiérarchie de classe qui se compose de super-classes et de sous-classes.
  - Une classe ne peut avoir qu'une seule classe mère : il n'y a pas d'héritage multiple en Java.
  - Object est la classe parente de toutes les classes en Java. Toutes les variables et méthodes contenues dans Object sont accessibles à partir de n'importe quelle classe car par héritages successifs toutes les classes héritent d'Object.

Cours JAVA - FMS

- La mise en oeuvre de l'héritage
  - On utilise le mot clé extends pour indiquer qu'une classe hérite d'une autre. En l'absence de ce mot réservé associé à une classe, le compilateur considère la classe Object comme classe mère.

Exemple :

```
public class Fille extends Mere { … }
```

  - Pour invoquer une méthode d'une classe mère, il suffit d'indiquer la méthode préfixée par super. Pour appeler le constructeur de la classe mère, il suffit d'écrire super(paramètres) avec les paramètres adéquats.
  - Le lien entre une classe fille et une classe mère est géré par la plate-forme Java.
  - En Java, il est obligatoire dans un constructeur d'une classe fille de faire appel explicitement ou implicitement au constructeur de la classe mère.

Cours JAVA - FMS

- L'accès aux propriétés héritées
  - Les variables et méthodes définies avec le modificateur d'accès public restent publiques à travers l'héritage et toutes les autres classes.
  - Une variable d'instance définie avec le modificateur private est bien héritée mais elle n'est pas accessible directement, elle l’est par les méthodes héritées (getter, setter).
  - Une variable définie avec le modificateur protected sera héritée dans toutes les classes filles qui pourront y accéder librement ainsi que les classes du même package.