Langage JAVA

Prof: R. EL AYACHI

Département d'Informatique

Faculté des Sciences et Techniques

Béni Mellal

Chapitre 1: Introduction à JAVA

1. Caractéristiques de bases de la POO
2. Naissance et développement de JAVA
3. Environnement de JAVA
4. Types de programmes JAVA
5. Paquetages (Packages)
6. Classes en JAVA
7. Utilisation des classes
8. Constructeurs
9. Attributs et méthodes de classes

1. Caractéristiques de bases de la POO

1. Notion de classe et d'objet:

Objet = Données + Traitements

* **Classe**: Abstraction d'un ensemble d'objets qui ont les mêmes caractéristiques (attributs) et les mêmes comportements (méthodes)

* **Objet**: Instance d'une classe

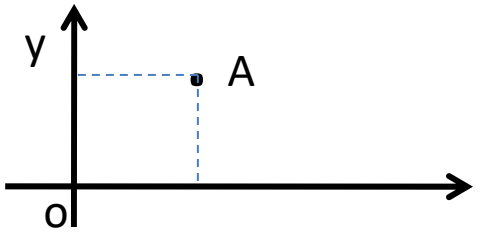
Exemple:

La classe **Point**

Attributs
Abs_x
Ord_y
Méthodes
Dist_orig()

L'objet **A**

Attributs
2
3
Méthodes
$\sqrt{13}$



1. Caractéristiques de bases de la POO

2. <u>Encapsulation des données:</u>

* Les données sont protégées dans chaque objet
* Les données (publiques, secrètes)

3. <u>Héritage:</u>

Une classe dérivée hérite des propriétés d'une classe de base:

* Economie de code
* Amélioration de la fiabilité
* Amélioration de la lisibilité

Exemple: un rectangle peut hériter d'un polygone

1. Caractéristiques de bases de la POO

4. <u>Polymorphisme:</u>

Le polymorphisme permet la redéfinition d'une méthode

5. <u>Multithreading:</u>

THREADS: processus qui s'exécutent simultanément à l'intérieur d'un unique programme

2. Naissance et développement de JAVA

1. Caractéristiques de JAVA:

* JAVA est un langage programmation orientée objet proche de C++
* JAVA fonctionne comme une machine virtuelle (indépendant de toute plate forme)
* JAVA est simple (pas de pointeur, pas d'héritage multiple)
* JAVA autorise le multitâche (multithreading)
* JAVA peut être utilisé sur internet
* JAVA contient une très riche bibliothèques de classes (packages) qui permettent de : créer des interfaces graphiques, utiliser les données multimédia, communiquer à travers les réseaux, …

2. Naissance et développement de JAVA

2. Petit historique de JAVA:

* 1991: Naissance de Java chez Sun Microsystems
* 1995: Réalisation du logiciel HotJava, un navigateur Web écrit par Sun en Java
* Les autres navigateurs Web ont suivi, ce qui a contribué à l'essor du langage sous forme de versions successives :

1996: version 1.01 et 1.02

1998: version 1.1

1999: version 1.2, finalement rebaptisée Java 2

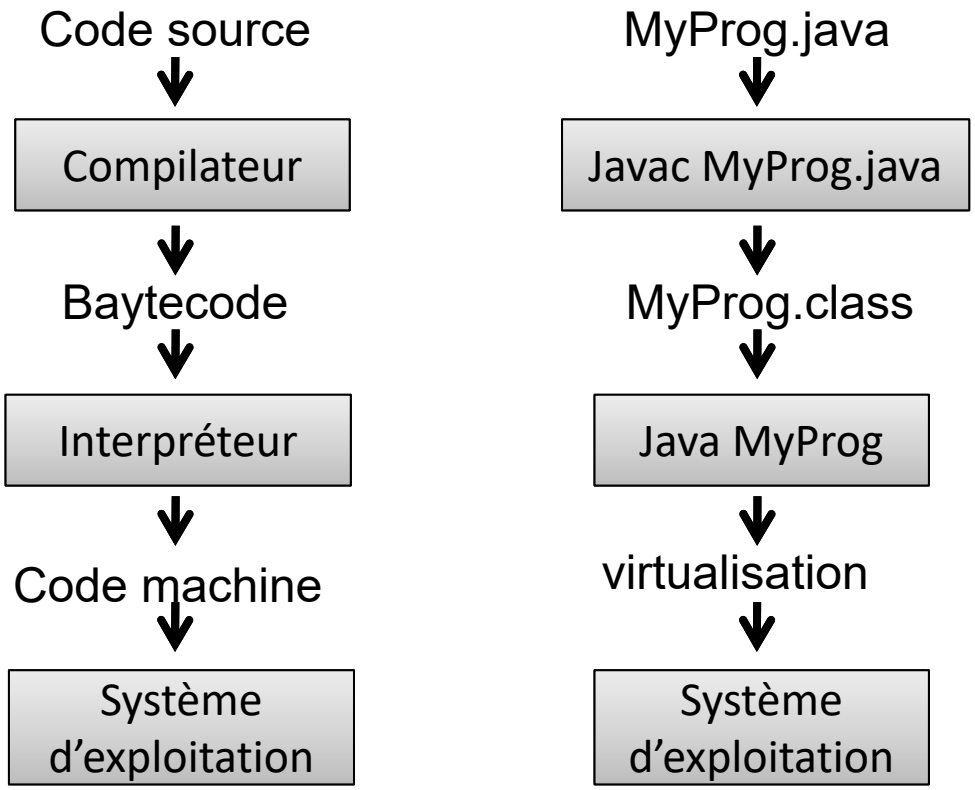
2000: version 1.3, toujours appelée Java 2

…

3. Environnement JAVA

* Java est un langage interprété (un programme compilé n'est pas directement exécuté par le système d'exploitation)

* JDK (Java Development Kit)



4. Types de programmes de JAVA

* **Applications JAVA**: programmes écrits en JAVA
* **Applets JAVA**: programmes lancés à partir d'un programme HTML par un logiciel de navigation

5. Paquetages (packages)

* Organisation hiérarchique des classes en packages
* Regroupement de plusieurs classes qui collaborent entre elles dans une application

1. <u>Construction de package</u>

* Pour ranger une classe dans un package, il faut insérer la ligne suivante au début du fichier source:

package nom_package;

2. <u>Utilisation des classes d'un package</u>

* On peut importer les packages de deux manières:

→ **import nom_package;**

→ **import nom_package.nom_classe;**

5. Paquetages (packages)

3. API (Application Programming Interface) de JAVA

* **java.lang**: contient les classes les plus fondamentales du langage (java.lang.Math, java.lang.String,…)
* **java.awt**: contient les classes pour fabriquer des interfaces graphiques (java.awt.drawline, java.awt.drawRect, …)
* **java.applet**: utile pour faire des applets qui sont des applications utilisables à travers le web
* **java.io**: contient les classes nécessaires aux E/S
* **java.net**: accès aux réseaux
* **java.util**: contient les classes d'utilitaires (Random, Date, …)

6. Classes JAVA

1. <u>Architecture</u>

* Une classe contient des variables et des méthodes, elle se compose de deux parties: **En-tête** et **Corps**

* **En-tête:**

[modificateur**]** **class** <NomClasse>**[extends** <superclass>**]**
[impliments <interface>**]**

[]: optionnel **<>**: obligatoire **gras**: mot clé

* **Corps:**

En-tête
{
déclarations des variables, des méthodes
}

6. Classes JAVA

2. Modificateur

Modificateur	Définition
abstract	Aucun objet ne peut instancier cette classe. Seules les classes abstraites peuvent déclarer des méthodes abstraites
final	Les classes finalises ne peuvent pas être héritables
private	La classe n'est accessible qu'à partir du fichier où elle est définie
public	La classe est accessible par toutes les autres classes des autres packages. Elle est visible partout

6. Classes JAVA

3. Convention d'écriture en JAVA

Type	Convention	Exemple
Nom d'une classe	Débute par une majuscule	class Polygone
Nom d'un objet	Débute par une minuscule	premierObjet
Nom d'une méthode	Débute par une minuscule sauf pour le constructeur	void ecrire()
Constante	S'écrit en majuscule	A_CONSTANT

6. Classes JAVA

4. Premier exemple

```
public class Premier
{
        public static void main(String[] arg)
        {
                System.out.println(‘’bravo’’);
        }
}
```

* **public**: visibilité partout, y compris les autres packages
* **static**: méthode de classe
* **System.out.println**: la méthode **println** écrit sur l’écran la chaine de caractères passe en paramètres

7. Utilisation des classes

Pour utiliser une classe, elle doit être instanciée. Alors, il y aura création d'un objet. Mais avant de créer un objet, il faut le déclarer.

1. Déclaration d'un objet

```
NomClasse  objet;
```

Cette instruction déclare un objet de la classe NomClasse mais ne le crée pas.

2. Création d'un objet

```
objet = new NomClasse();
```

La déclaration et la création d'un objet peuvent être regroupées en une seule instruction:

```
NomClasse  objet = new NomClasse();
```

7. Utilisation des classes

On accède aux méthodes et aux attributs de la classe comme suit:

NomObjet.méthode(arguments de la méthode)

NomObjet.attribut

7. Utilisation des classes

3. Exemple d'utilisation d'objet:

```
class Ecriture
{
  String chaine= ''encore une fois'';
  void ecrire(String autrechaine)
  {
   System.out.println(chaine);
   System.out.println(autrechaine);
  }
}
```

```
public class PourFiliciter
{
public static void main(String[] arg)
  {
    Ecriture ecrivain;
    ecrivain = new Ecriture();
    ecrivain.ecrire(''bravo'');
    ecrivain.chaine=''et pour finir'';
   ecrivain.ecrire(''au revoir'');
  }
}
```

8. Constructeurs

1. Définition

* Le constructeur est une méthode qui est effectuée au moment de la création d'un objet. Il sert à initialiser les variables contenues dans les objets.

* Le constructeur porte le même nom que sa classe. Il ne retourne pas de valeurs et ne mentionne pas **void** au début de sa déclaration.

* Le constructeur est une fonction qui est appelée par le biais de l'opérateur **new.**

monobjet = new constructeur(arguments);

8. Constructeurs

2. Exemple de constructeur:

```
class Incremente
{  int petitPlus;
  int increment;
  Incremente(int i,int petit)
  {   incremente=i;
      petitPlus=petit;
  }
 int additionne(int n)
  {
    return(n+increment+petitPlus);
  }
}
```

```
class Constructeur
{
public static void main(String[] arg)
  {
   Incremente  monAjout =  new
   Incremente(10,1);

  System.out.println(monAjout.ad
  ditionne(5);
   }
}
```

8. Constructeurs

3. Variable this

* La variable **this** sert à référencier dans une méthode l'instance de l'objet en cours d'utilisation. Son emploi peut s'avérer utile dans les constructeurs avec des paramètres lorsque les variables de la classe et celles du constructeur portent exactement le même nom.

8. Constructeurs

4. Exemple d'utilisation de la variable this:

```
class Incremente
{  int petitPlus;
   int increment;
   Incremente(int increment,int petit)
   {   this.increment=increment;
       petitPlus=petit;
   }
  int additionne(int n)
   {
      return(n+increment+petitPlus);
   }
}
```

```
class Constructeur
{
public static void main(String[] arg)
   {
    Incremente  monAjout =  new
    Incremente(10,1);

   System.out.println(monAjout.additi
   onne(5);
    }
}
```

9. Attributs et méthodes

1. <u>Définition</u>

* **Attribut**: un attribut correspond à une variable ou une constante pouvant prendre une valeur différente pour chaque objet instance de la classe. Avant d'utiliser un attribut, il doit être déclaré comme suit:

  <modificateur> type attribut

* **Méthode**: les méthodes sont les opérations qu'on peut appliquer aux objets. Elles permettent de changer l'état des objets ou de calculer des valeurs. Plusieurs méthodes d'une même classe peuvent avoir le même nom, mais des paramètres différents (surcharge).

  Une méthode peut aussi posséder des modificateurs.

9. Attributs et méthodes

2. Quelques modificateurs

Modificateur	Définition
final	Une variable constante Une méthode qui ne peut pas être redéfinie
protected	Méthode ne peut être invoquée que par des sous classes
private	Une variable privée ou une méthode privée ne peut être utilisée que dans les méthodes de la même classe.
public	Variable ou méthode visible par toutes les autres méthodes.
static	Méthode ou variable de classe.

9. Attributs et méthodes

4. Exemple d'utilisation d'attribut et de méthode:

```
class Repetiteur
{  private String nom;
   private int nbRepetitions;
   String message=''travaillons'';
   Repetiteur(String nom,int
   nbRepetitions)
   {   this.nom=nom;
       this.nbRepetitions=nbRepetitions;
   }
Void repete ()
   {  for(int i=0;i<=nbRepetitions;i++)
      System.out.println(message);
   }
```

```
public String ToString()
{return nom+''repete :''+ message;
}
public static void main(String[] arg)
   {int repetition=Integer.parseInt(arg[0]);
    Repetiteur  ecrivain =  new
    Repetiteur(''ali '',repetition);
    ecrivain.repete();
    ecrivain.message=''reposons nous'';
    System.out.println(ecrivain);
    }
}
```