Programmation objets en java

Programmation objets en Java

- **Introduction :**
- Dans la programmation orienté objet, on s'intéresse d'abord aux données qui constituent la partie la plus stable, puis ensuite on leur associe des traitements.
- Cette nouvelle façon de programmer nécessite des langages spécifiquement conçus pour placer les données au centre des préoccupations. Pour cela, on introduit les concepts suivants :
-
- 1-) classes et objets
- 2-) méthodes et attributs
- 3-) encapsulation
- 4-) héritage
- 5-) polymorphisme

Programmation objets en Java

- **Déclaration des classes**
- **La syntaxe générale**
- [modificateurs] **class** NomClasse {
- [déclaration des attributs/propriétés privés]
- [déclaration des méthodes d'accès]
- [déclaration des constructeurs]
- [déclaration des méthodes utilitaires]
- }

Programmation objets en Java

- **Création des objets**

- Les objets sont des instances de classe. Si on a défini une classe (Cercle, Personne, Étudiant, ..), c'est pour créer ensuite des objets de cette classe.
- En Java, il ne suffit pas de **nommer** les variables objets, il faut les **créer** explicitement et les **initialiser**.

- Syntaxe :

- 1-) déclare le nom d'un objet de la classe sans définir l'objet lui-même
  - **NomClasse** nomObjet;
- 2-) construit l'objet
- nomObjet = **new constructeurClasse**([liste de paramètres]) ;
- 3-) permet de fusionner les 2 étapes
- nomClasse nomObjet = new constructeurClasse([liste de paramètres]) ;

Programmation objets en Java

- Exemple : Classe Personne
- public class Personne {
- private int num, age;
- private String nom, prenom;
- private char genre;
- public Personne(){
- this (0,"ahmed", "Bounouar", ‘M’, 35);
- }

- public Personne(int num, String nom, String prenom, char genre, int age){
- this.num=num;
- this.nom=nom;
- this.prenom=prenom;
- this.genre=genre;
- this.age=age;
- }

Suite de la classe Personne

- public String **getNom**() { return this.nom;}
- public void **setNom (**String value) {this.nom=value.toUpperCase();}
- public String getPrenom() { return prenom; }
- public void setPreom(String prenom) {this.prenom=prenom.toUpperCase();}
- public void setGenre(char genre) {this.genre=genre;}
- public char getGenre( ) {return this.genre;}
- public void setAge(int age) {this.age=age;}
- public int getAge( ) {return this.age;}

- public Override String **toString**(){
- return "numero="+this.num+"\nnom="+this.nom+"\nprenom="+
- this.prenom+" \n Genre="+this.genre+"\n age="+this.age;
- }

Héritage

- L'avantage essentiel d'un langage orienté-objet est que le code est **réutilisable**. Grâce à l'héritage, on peut faire dériver une nouvelle classe d'une classe existante et ainsi en récupérer les propriétés et méthodes, sans avoir à la réécrire de nouveau complètement.
- On dit que la classe Enfant hérite de la classe Parent ou de Base, qu'elle étend cette ancienne classe.

- Syntaxe
- [modificateur] class NomclassEnfant extends NomclassParent {
  - Les nouvelle propriétés et méthodes
- }
- Remarque : Pour redéfinir une méthode du parent, vous devez utiliser l'identificateur **Override dans la signature de la méthode.**

Héritage (suite)

- Remarque :
- Utiliser une méthode de la classe parent qui a été redéfinie dans l'enfant :

- Pour pouvoir faire appel à une méthode du parent, on écrit le mot-clé **super** devant le nom de la méthode. **super.nomMethod**(), fait appel à la méthode définie dans la classe Parent et non à celle d'Enfant.
-

- Utiliser le constructeur de la classe parent :

- On peut faire appel à un constructeur de la super-classe en utilisant le mot spécial **super()**
- L'appel au constructeur super() doit être la **première instruction** du bloc.

Héritage (suite)

- Exemple :
- public **class** Employe **extends** Personne{
- private double sal;
- public Employe(){
- this(0,"ahmed","Bounouar",'M',35,1000);
- }
- public Employe(int num, String nom, String prenom, char genre, int age, double sal){
- **super**(num, nom, prenom, genre, age);// appel au const du parent
- **this.sal=sal**;
- }
- @override
- public String toString(){
- return **super**.toString()+" salaire:"+this.sal;
- }

Programmation objet en Java

- **Classes et méthodes abstraites**

- - Une classe abstraite est une classe qui contient au moins une méthode abstraite.
- - Une classe abstraite n'ayant pas d'instances. Elle n'est utilisable qu'à travers sa descendance. Autrement dit, une telle classe ne peut pas être instanciée directement, et doit toujours être dérivée pour pouvoir générer des objets.
- - Elle sert uniquement à regrouper des attributs et méthodes communes à un groupe de classes pour constituer une véritable "factorisation" de ces classes.

Classe abstraite (suite)

- **Déclaration d'une classe abstraite**:

- public abstract class MaClasseAbstraite {
- ...
- public abstract type nomMethod(…) ;
- …
-
- }