CHAPITRE 6 :

LES CLASSES ABSTRAITES ET LES INTERFACES

Objectifs spécifiques

1. Introduire la notion de classe abstraite, les règles de définition et l'intérêt pratique
2. Introduire le concept d'interfaces, leurs intérêts pratiques et leurs relations avec l'héritage

Eléments de contenu

I. Les classes abstraites

II. Les interfaces

Volume Horaire :

Cours : 3 heures

Travaux Pratiques : 3 heures

6.1 Les classes abstraites

6.1.1 Concept des classes abstraites

- Une classe abstraite est une classe qui ne permet pas d'instancier des objets, elle ne peut servir que de classe de base pour une dérivation.
- Dans une classe abstraite, on peut trouver classiquement des méthodes et des champs, dont héritera toute classe dérivée et on peut trouver des méthodes dites « abstraites » qui fournissent uniquement la signature et le type de retour.

Syntaxe

```
abstract class A
{public void f() {……} //f est définie dans A
public abstract void g (int n) ; //g est une méthode abstraite elle n'est pas définie dans A
}
```

Utilisation

```
A a ;
//on peut déclarer une référence sur un objet de type A ou dérivé
a = new A (….) ;
```

```
//Erreur  pas d'instanciation d'objets d'une classe abstraite
```

Si on dérive une classe B de A qui définit les méthodes abstraites de A, alors on peut :

```
a = new B(…) ; //Juste car B n'est pas une classe abstraite
```

6.1.2 Règles des classes abstraites

✈ Dès qu'une classe abstraite comporte une ou plusieurs méthodes abstraites, elle est abstraite, et ce même si l'on n'indique pas le mot clé « abstract » devant sa déclaration.

✈ Une méthode abstraite doit être obligatoirement déclarée « public », ce qui est logique puisque sa vacation est d'être redéfinie dans une classe dérivée.

✈ Les noms d'arguments muets doivent figurer dans la définition d'une méthode abstraite

```
public abstract void g(int) ;
//Erreur : nom argument fictif est obligatoire
```

✈ Une classe dérivée d'une classe abstraite n'est pas obligée de redéfinir toutes les méthodes abstraites, elle peut ne redéfinir aucune, mais elle reste abstraite tant qu'il y a encore des méthodes abstraites non implémentées.

✈ Une classe dérivée d'une classe non abstraite peut être déclarée abstraite.

6.1.3 Intérêt des classes abstraites

Le recours aux classes abstraites facilite largement la conception orientée objet. On peut placer dans une classe abstraite toutes les fonctionnalités dont on souhaite disposer pour les classes descendantes :

- Soit sous la forme d'une implémentation complète de méthodes (non abstraites) et de champs (privés ou non) lorsqu'ils sont communs à tous les descendants,
- Soit sous forme d'interface de méthodes abstraites dont on est alors sûr qu'elles existeront dans toute classe dérivée instanciable.

6.1.4 Exemple

```
abstract class graphique {
private int x, y;
graphique ( int x, int y) { this.x = x ; this.y = y ;}
void affiche () {
System.out.println («  Le centre de l'objet se trouve dans :   » + x + «  et
 » + y) ;}
```

```
Abstract public double surface () ;  // méthode abstraite
} // fin de la classe graphique
//Classe dérivée1
class Cercle extends graphique {
private double rayon =1 ;
void affiche () {
Sustem.out.println (« C'est un cercle  de rayon » + rayon) ;
Super.affiche() ; }
double surface ()
{return ( rayon * 2* 3.14) ;} }
//Classe dérivée2
class Rectangle extends graphique {
private int larg, longueur ;
Rectangle ( int x, int y, int l1, int l2)
{super (x,y) ;
longueur = l1 ; larg = l2 ;}
double surface () {return (longueur*largeur) ;}
}// fin de la classe Rectangle


// Classe de test
class test_poly2 {
public static void main (String [] args) {
graphique [] tab = new graphique [6];
//tab[0] = new graphique (3,2); Erreur car une classe abstraite ne peut
pas être instanciée
tab[0] = new Cercle (3,2,7);
tab[1] = new Cercle (10,7,3) ;
tab[2] = new Rectangle (4,7,8,6) ;
tab[3] = new Rectangle (8,10,12,10);
tab[4] = new Cercle (8,5,3) ;
tab[5] = new Rectangle (10,17,3,8) ;
for (int i=0 ; i <=5 ; i++) {tab[i].affiche();}
}
}
```

=> Une classe abstraite peut ne comporter que des méthodes abstraites et aucun champ. Dans ce cas, on peut utiliser le concept de **l'interface**.

6.2 Les interfaces

6.2.1 Concept d'interface

- Si on considère une classe abstraite n'implantant aucune méthode et aucun champ (sauf les constantes), on aboutit à la notion d'interface.

- Une interface définit les entêtes d'un certain nombre de méthodes, ainsi que des constantes.

- L'interface est plus riche qu'un simple cas particulier des classes abstraites :

    - Une classe peut implémenter plusieurs interfaces (alors qu'une classe ne pouvait dériver que d'une seule classe abstraite).

    - La notion d'interface va se superposer à celle de dérivation, et non s'y substitue.

    - Les interfaces peuvent se dériver.

    - Une classe dérivée peut implémenter 1 ou plusieurs interfaces, elle peut même réimplémenter une interface déjà implémentée par la superclasse.

    - On pourra utiliser des variables de type interface.

6.2.1 Syntaxe

6.2.1.1 Définition d'une interface

```
public interface I
{void f (int n) ; //public abstract facultatifs
void g () ; //public abstract facultatifs
}
```

Toutes les méthodes d'une interface sont abstraites. On peut ne pas mentionner le modificateur de visibilité (par défaut public) et le mot clé « abstract »

6.2.1.2 Implémentation d'une interface

- Lorsqu'on définit une classe, on peut préciser qu'elle implémente une interface donnée en utilisant le mot clé « implements ».

Syntaxe

```
public interface I1
{….}
public interface I2
{…}
class A implements I1, I2
{…..A doit définir les méthodes de I1 et I2}
```

6.2.1.3 Exemple

```
interface Affichable
{void affiche() ;
}
class Entier implements Affichable
{private int val ;
public Entier (int n)
{val = n ;}
public void affiche() {
System.out.println (« Je suis un entier de valeur  » + val) ;}
}
class Flottant implements Affichable
{private float val ;
public Flottant (float n)
{val = n ;}
public void affiche() {
System.out.println (« Je suis un flottant de valeur  » + val) ; }
}
public class TestInterface {
public static void main (String [] args) {
Affichеable [] tab = new Affichеable [2];
tab[0] = new Entier (25);
```

```
tab [1] = new Flottant (1.25);
tab [0].affiche(); tab [1].affiche; } }
```

Résultat de l'exécution

- Je suis un entier de valeur 25
- Je suis un flottant de valeur 1.25

6.2.1.4 Interface et constantes

Une interface peut renfermer aussi des constantes symboliques qui seront accessibles à toutes les classes implémentant l'interface :

```
public interface I
{void f (int n) ; //public abstract facultatifs
void g () ; //public abstract facultatifs
 static final int max = 100 ;
}
```

Les constantes déclarées sont toujours considérées comme « static » et « final »

6.3 Interface et dérivations

Remarque

L'interface est totalement indépendante de l'héritage : Une classe dérivée peut implémenter une ou plusieurs interfaces.

Exemple

```
interface I1 {…..}
interface I1 {…..}
class A implements I1 {……}
class B  extends A implements I1, I2 {…..}
```

Remarque2

On peut définir une interface comme une généralisation d'une autre. On utilise alors le mot clé « extends »

Exemple

```
interface  I1
{void f(int n) ;
static final int max = 100;
```

```
}
interface  I2 extends I1
{void g (int n);
static final int min = 10;
}
```

En fait la définition de I2 est équivalente à

```
interface  I2
{void g (int n);
static final int min = 10;
void f(int n) ;
static final int max = 100;
}
```

Remarque3

- Conflits des noms

Soit l'exemple suivant

```
interface I1
{void f(int n) ;
void g();
}
interface I2
{void f (float x);
void g();
}
class A implements I1, I2
{ /*A doit implémenter la méthode g qui existe dans les 2 interfaces une seule fois*/}
```

->Supposons que la méthode « g » est présente de 2 façons différentes dans I1 et I2

Exemple

```
interface I1
{ void g(); }    // g est une méthode abstraite de I1
interface I2
{ int  g(); }   // g est aussi une méthode abstraite de I2
class A implements I1, I2 {
/* Erreur  car  void g() et int g() ne peuvent pas coexister au sein de la même classe*/
```