Programmation Orientée Objets

JAVA

Tarek El Falah

tarek_el_falah@hotmail.com

LFIG2

Plan du cours

- Introduction à la POO (concepts de base)
- Eléments syntaxiques de base du langage JAVA
- Classes et Objets dans JAVA
- Collections d'objets dans JAVA
- Héritage et polymorphisme dans JAVA

Introduction à la POO

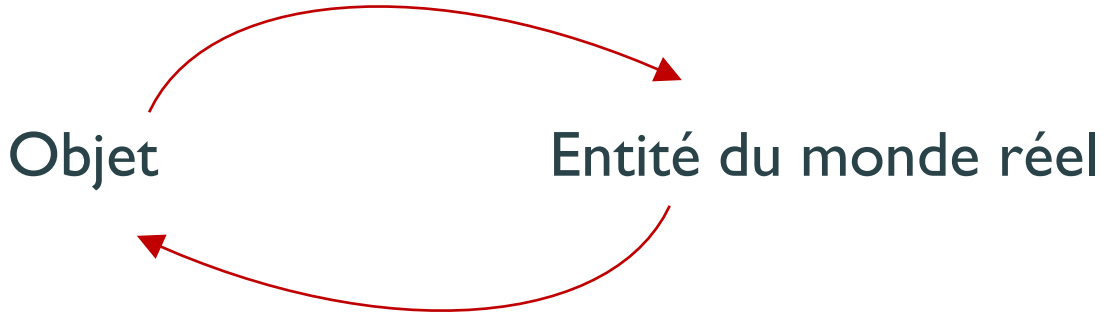
(concepts de base)

Chapitre 1

Approche Orientée Objets

Approche classique de Programmation	Approche Orientée Objets (POO)
Que doit faire le programme ? ↳ raisonnement par tâche, par traitement,…	Sur quoi porte le programme ? Quels sont les objets manipulés par le programme ? ↳ raisonnement par objets

Notion d'objets



- Objet = Etat + Comportement
- Les états sont stockés dans des variables → *attributs*
- Les comportements sont des opérations implémentées à l'aide de fonctions et procédures → *méthodes*

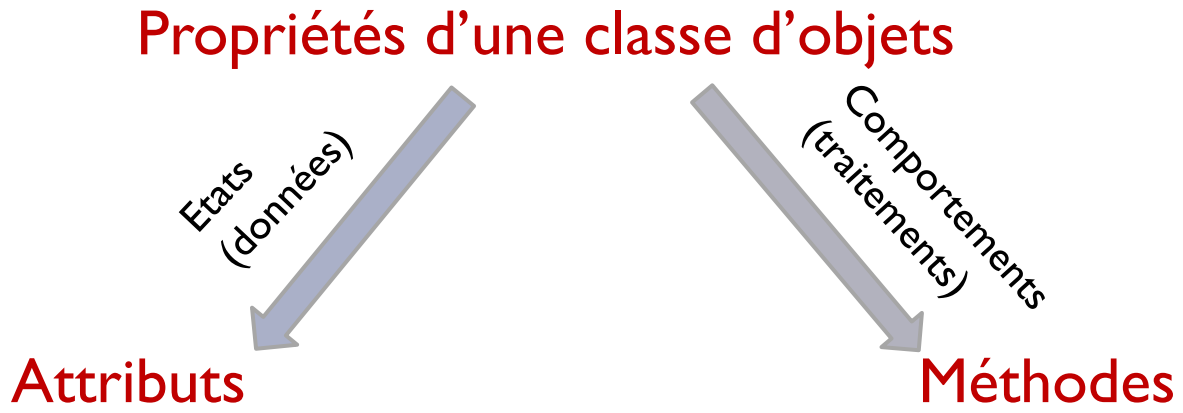
Notion de classes

Définition

• Une classe représente un modèle ou une catégorie d'objets et peut être définie comme un type abstrait de données.

• Une classe donne une description de l'état (*attributs*) et du comportement (*méthodes*) de ces objets

• Les attributs et les méthodes d'une classe sont appelés *propriétés de la classe*



Exemples de classes

Classe Point

Attributs

x : réel

y : réel

Méthodes

Afficher() {…}

Supprimer() {…}

Déplacer(dx, dy) {…}

…

Classe Article

Attributs

ref : entier

libellé : chcar

Qte : entier

PUHT : réel

Méthodes

Calcul-TTC() {…}

Entrée-Stock() {…}

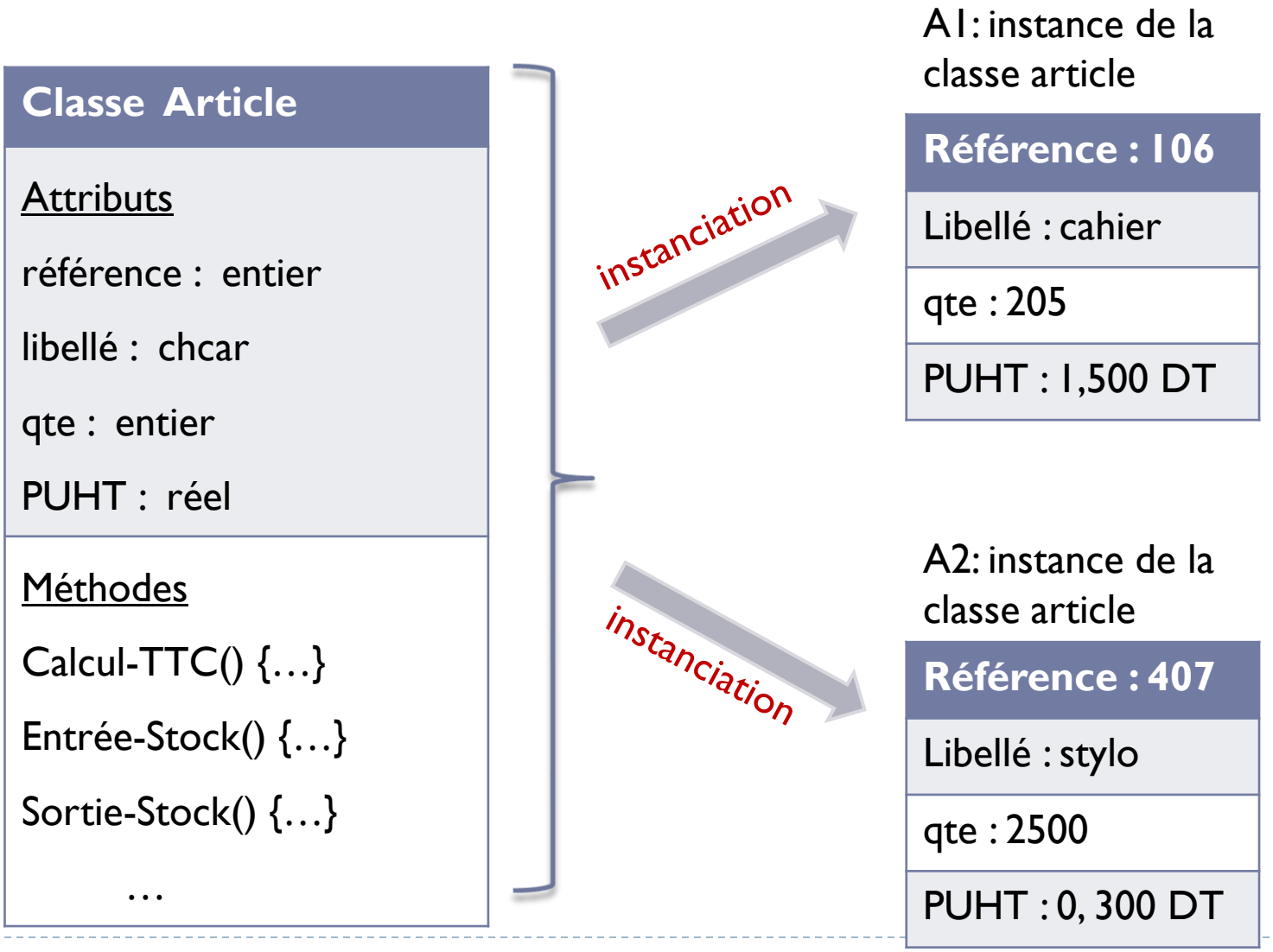
Sortie-Stock() {…}

…

Instanciation

- Tout objet est une *instance* d'une classe
- La création d'une instance s'appelle *instanciation* de la classe
- Une instance :
  - occupe de l'espace en mémoire
  - stocke les valeurs des attributs
  - possède un comportement

Exemple d'instanciation



Encapsulation

Définition

- ✓ Un mécanisme consistant à rassembler données et traitements (attributs et méthodes) dans une même structure (classe) en empêchant l'accès aux données par un autre moyen que les services proposés par la classe (les méthodes)

- ✓ On distingue 2 parties dans la définition d'une classe
  - Partie *privée* accessible uniquement par les méthodes de la classe
  - Partie *publique* accessible à l'extérieur de la classe

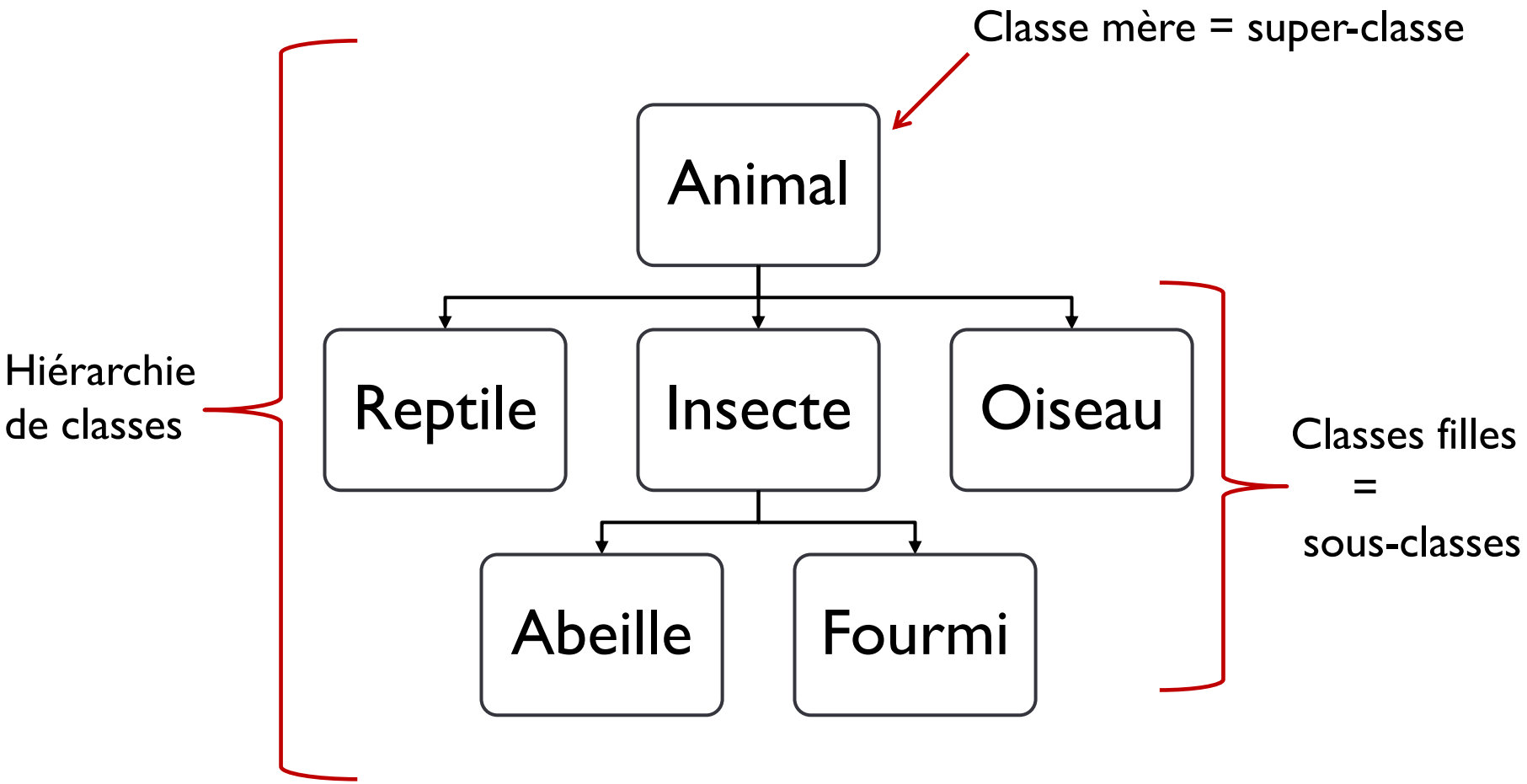
Héritage

- Une classe peut être *spécialisée* en d'autres classes afin d'y ajouter des caractéristiques spécifiques ou,
- Plusieurs classes peuvent être *généralisées* en une seule classe afin de regrouper les caractéristiques communes de ces classes

Définition

- Héritage : mécanisme de transmission automatique des propriétés d'une classe vers une sous-classe
- Une sous-classe hérite de toutes les propriétés de la classe mère tout en ajoutant celles qui lui sont propres

Hiérarchie de classes



Démarche de POO

Conception

1. Identifier les classes d'objets qui modélisent le mieux le problème à résoudre
2. Déterminer les relations entre les diverses classes identifiées (Héritage ou composition)
3. Définir l'interface de chaque classe (services publics)

Implémentation

4. Codification des classes dans un langage OO (JAVA)

Eléments syntaxiques de base du langage JAVA

Chapitre 2

Premier exemple de programme JAVA

droit d'accès des autres classes à cette classe

précise que la méthode main n'est pas liée à une instance particulière de la classe

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

obligatoire pour la compilation

méthode d'affichage dans la fenêtre console

permet de récupérer des arguments au moment du lancement du programme
NB: obligatoire même si on ne passe pas d'arguments

La classe Scanner (lecture à partir du clavier)

```
Import.java.util.Scanner
Scanner S=new Scanner(System.in);
int n = S.nextInt();
float f = S.nextFloat();
double d = S.nextDouble();
short s = S.nextShort();
long l = S.nextLong();
byte b = S.nextByte();
boolean bo = S.nextBoolean();
String ch1 = S.next ();   //lit un mot délimité par l'espace
String ch2 = S.nextLine();    //lit une chaîne pouvant contenir des
                              //espace
```

```
//NB : pas de méthodes pour lire le
//type char

char c;
Scanner S=new Scanner(System.in);
S.o.p ("taper un caractère");
String ch=S.next;
c=ch.charAt(0);
```

Exemple : somme de deux entiers

```
public class Somme
{
   public  static  void main(String [] args)
   {
       int x, y;
       Scanner S=new Scanner(System.in);
       x= S.nextInt();
       y= S.nextInt();
       System.out.println ("la somme de" +x+"et"+y+ "est égale
          à"+(x+y) );
   }
}
```

Les types primitifs

Type		Taille (en octets)	Valeur minimale	Valeur maximale
Entiers	byte	1	Byte.MIN_VALUE (-128)	Byte.MAX_VALUE (127)
	short	2	Short.MIN_VALUE	Short.MAX_VALUE
	int	4	Integer.MIN_VALUE	Integer.MAX_VALUE
	long	8	Long.MIN_VALUE	Long.MAX_VALUE
Flottants	float	4	Float. MIN_VALUE	Float. MAX_VALUE
	double	8	Double. MIN_VALUE	Double. MAX_VALUE

Les types primitifs

- *Le type caractère*

char c1, c2;

constantes : ‘a’, ‘T’, ‘2’, ‘+’,…

- *Le type booléen* sert à représenter une valeur logique du type vrai/faux

Boolean ord;

ord = n < p;

Les 2 constantes de types booléen se notent *true* et *false*

Le mot clé final

```
final int n=20 ;     //la valeur de la variable ne change pas pendant l'exécution
n= n+10;  // faux
Scanner S=new Scanner(System.in);
n= S.nextInt();  // faux

int p;
p=4;
final int n=5*p; //ok

final int n;
Scanner S=new Scanner(System.in);
n= S.nextInt();  // première affectation de n →ok
n++;  // faux
```

Les opérateurs et les expressions

- *Opérateurs arithmétiques* : +, -, *, /, %
- Conversions implicites des types dans les expressions

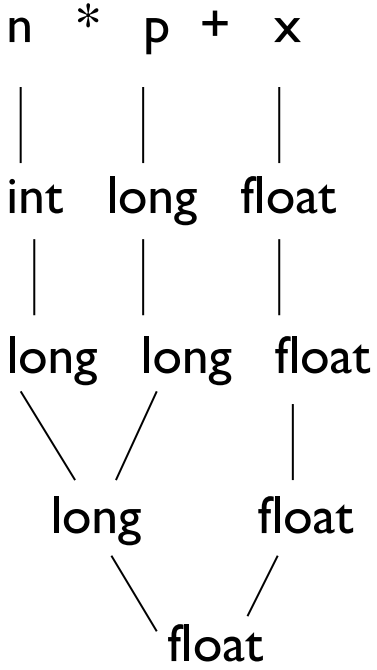
  Byte → short → int → long → float → double

Exemple :

int n;

long p;

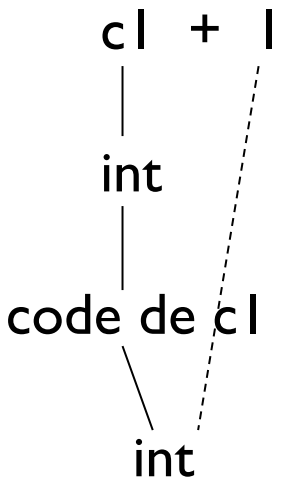
float x;



Les opérateurs et les expressions

- *Cas du type char*

```
char c1;
```



- *Les opérateurs relationnels* : <, <=, >, >=, ==, !=
- *Les opérateurs logiques* :

```
!   : négation
&&  : et
||  : ou
```

Les opérateurs et les expressions

- *Opérateur d'affectation* =
  - associativité de droite à gauche :

```
i = j = 5 ;
```

  - conversion par affectation :

byte → short → int → long → float → double

char → int → long → float → double

  - Exemple :

```
int n;
byte a1, a2;
short p;
n=a1*a2;  // ok
P=a1*a2;  // ko
```

- *Opérateurs d'incrémentation et de décrémentation* : ++ , --

Les instructions de contrôle et les boucles en JAVA

- *L'instruction* ***if***
- *L'instruction* ***switch***
- *L'instruction* ***do … while***
- *L'instruction* ***while***
- *L'instruction* ***for***

Classes et Objets dans JAVA

Chapitre 3

Déclaration des classes

class *Nomclasse*

{

//Déclaration des attributs

Droit d'accès type *NomAttribut* ;

…

// Définition des méthodes

Droit d'accès TypeResRetourné *NomMéthode*(liste paramètres)

{suite d'instructions}

…

} //fin classe

Exemple: Classe Point

```
class Point
{
  //Déclaration des attributs
  private int x ;
  private int y :

  // Définition des méthodes
  public void initialiser (int a, int b)
  {x=a; y=b;}

  public void afficher ()
  {System.out.println ("les coordonnées de ce point sont :" +x+ "et" +y);}

  public void déplacer (int dx, int dy)
  {x+=dx; y+=dy;}
} //fin de la classe Point
```

Encapsulation

Java propose 4 niveaux de protection

- **private** : la propriété est visible et accessible seulement par les méthodes de la classe
- **Public :** la propriété est visible et accessible à l'extérieur de la classe
- **Par défault** : la propriété est visible et accessible par les méthodes de la classe et aussi par toutes les classes du même package
- **Protected** : la propriété est visible et accessible par les méthodes de la classe, par toutes les sous-classes et aussi par toutes les classes du même package

Instanciation de classes (création d'objets)

(1) NomClasse Ref; Création référence

Exemple : Point A;

(2) Ref = new NomClasse (); Allocation mémoire

Exemple : A=new Point ();

(1) et (2) ⟺ Ref = new NomClasse ();

Exemple: Point A=new Point ();

Manipulation d'objets

- *Application de méthodes*

Ref.NomMéthode (paramètres);

```
class Point
{
…
}
```

```
public class MonProg
{
   public static void main(String[] args])
   {
      Point P1=new Point();
      Point P2=new Point();
      P1.initialiser(1,-1);
      P1.afficher();
      P2.intialiser(5,6);
      P2.déplacer(10,5);
      P1= P2;
      P2.déplacer(10,10);
   }//fin main
}//fin MonProg
```

Affectation de référence
et non pas de contenu

Manipulation d'objets

- *Affectation entre objets*

$P_1 = P_2$; Faux

```
class Point
{
      private int x, y;
      …
      public void copier (Point P)
      {
          x = P . x;
          y = P . y;
      }
}
```

Sol : P_1. copier (P_2);

- *Comparaison entre objets*

if(P_1== P_2) Faux

```
class Point
{
      private int x, y;
      …
      public boolean egalite (Point P)
      {
          return((x==P. x) && (y==P. y));
      }
}
```

Sol : if (P_1. egalite(P_2))

Application

1. Définir une classe Point avec les deux attributs x et y et trois méthodes:
   - Saisir
   - Afficher
   - Egalite qui compare deux points
2. Ecrire un programme principal qui crée un tableau de n points, n étant donné par l'utilisateur. Le programme doit saisir puis afficher le tableau, il doit ensuite rechercher un point donné par l'utilisateur

Le mot clé « this »

```
class Point
{
      private int x, y;
      …
      public void déplacer (int dx, int dy)
      {
          x += dx;   ⇔   this.x +=dx;
          y += dy;       this.y +=dy;
      }
}
```

```
public void déplacer (int x, int y)
{
          x += x; !!!
          this.x += x;
          this.y += y;
}
// this est utile pour faire la différence
   entre les attributs x, y et les
   paramètres x, y
```

Rq : dans une méthode, la référence « this » est la référence de l'objet qui appelle cette méthode.

Les constructeurs

- Un constructeur est une méthode automatiquement appelée lors de la création d'une instance d'une classe.
- Règles des constructeurs en java :
  - Les constructeurs en java portent le nom de la classe.
  - Les constructeurs n'ont pas de type de retour.
  - Une même classe peut avoir plusieurs constructeurs qui doivent se distinguer par leurs paramètres.
  - Si une classe hérite d'une autre classe, son constructeur doit appeler impérativement celui de la classe mère en utilisant le mot clef « **super** ».

Surcharge des méthodes

```
class Point
{
      private int x, y;
      Public Point () {x=1; y=-1;}
      Public Point (int a) {x=a; y=a;}
      Public Point (int a, int b) {x=a; y=b;}
      …
      public void afficher ()
      {   S.o.p ("(" + x + "," + y + ")" );  }

      public void afficher (string  msg)
      {    S.o.p (msg );
           S.o.p ("(" + x + "," + y + ")" );  }
}
```

Surcharge du constructeur

```
public static void main(String[] args)
{
      Point P1=new Point ();
      Point P2=new Point (25);
      Point P3=new Point (2,19);
      P1.afficher();
      P2.afficher("Voici le point P2:");
}
```

Surcharge de la méthode afficher

Surcharge des constructeurs

```
class Point
{
      private int x, y;
      …
      public Point (int x, int y)
      {
          this.x += x;
          this.y += y;
      }
      public Point ()
      {  this (2,5);  }

}
```

- Dans JAVA, un constructeur peut être déclenché par un autre constructeur dans une même classe à l'aide du mot clé "this"
- L'appel "this" doit obligatoirement être la première instruction du constructeur

Déclenchement du constructeur paramétré

Propriétés statiques dans une classe

- ***Attributs statiques* (variables de classes) :** attributs partagés par tous les objets de la classe

```
class Point
{
      private int x, y;  // variables d'instance
      private static int NPI=0;  // un attribut statique est initialisé au
          moment de sa déclaration

      public Point ()
      {
          x = 1;
          y= -1;
          NPI++;
      }
      public vois static afficher_NPI ()
      {  S.o.p ("le nombre actuel de points créés est:"  + NPI) ; }
}
```

Propriétés statiques dans une classe

- ***Méthodes statiques* (méthodes de classes) :** elles n'agissent pas sur les variables d'instance mais uniquement sur les attributs statiques.
- Ces méthodes peuvent être appelées sans instancier la classe à l'aide de l'appel suivant: ***NomClasse.NomMéthode ();***

```
public static void main(String[] args)
{
        Point.afficher_NP1(); // 0
        Point A=new Point ();
        Point.afficher_NP1();// 1
ou
        A.afficher_NP1(); // 1
}
```

Préférable car la méthode est statique

Collections d'objets dans JAVA

(Quelques classes prédéfinies)

Chapitre 4

Rappel sur les tableaux classiques

- La création d'un tableau de type *objet* ne génère pas d'objet pour chaque élément mais uniquement des références initialisées à "NULL" par défaut

- La taille d'un tableau est déterminée lors de la création et ne peut être modifiée par la suite

- Un tableau est considéré comme un objet : il possède un attribut publique *length* qui donne la taille du tableau

  exemple : S.o.p (T.length)

La classe java.util.ArrayList
Gestion des tableaux dynamiques

- ArrayList :
  - gère un ensemble d'éléments accessibles par leurs indices
  - Peut contenir des éléments identiques
  - est non limitée en taille → la classe ArrayList ajuste la taille au fur et à mesure que l'on ajoute ou supprime des éléments

Remarque : Les collections ne peuvent stocker que des objets

→ Pas d'éléments de type primitif

Méthodes utiles de la classe ArrayList

ArrayList() // constructeur non paramétré

Exemple : ArrayList T = new ArrayList(); // Création de T initialement vide

ArrayList(int N) // constructeur paramétré

Exemple : ArrayList T2 = new ArrayList(10);

void add(object o) // Ajout d'un objet

Exemple : T.add(A); // Ajout de l'objet A dans T

void add(int i, object o) // Ajout d'un objet à la position i

Exemple : T.add(0,B);

int size() //retourne la taille du tableau

Exemple : S.o.p(" Taille de T :"+ T.size());

boolean isEmpty() // retourne vrai si le tableau est vide

void clear() //vide le tableau

Exemple : T.clear();

void remove(int i) //suppression de l'objet se trouvant à la position i

void remove(object o) //suppression de l'objet o

object get(int i) // retourne l'objet se trouvant à la position i

void set(int i, object o) // remplace l'objet se trouvant à la position i par o

Exemple

```
public class prog
{
public static void main(String[] args)
{
    point p1 = new point(1,1);
    point p2 = new point(2,2);
    point p3 = new point(3,3);
    ArrayList TP =  new ArrayList() ;
    TP.add(P1);
    TP.add(P2);
    TP.add(0,P3); //  P3;P1;P2
    TP.set(0,P1);
    TP.set(1,P2);
    TP.set(2,P3);//P1;P2;P3
    //Affichage
    for(int i=0; I < TP.size(); i++)
    {
        point p = (point) TP.get(i);
        P.afficher();
    }
}
}
```

La classe java.util.HashSet
Gestion d'un ensemble d'objets uniques

- HashSet gère un ensemble d'objets non ordonnés et uniques
- Quelques méthodes utiles :

HashSet () // constructeur non paramétré

Exemple : HashSet E = new HashSet (); // Création d'un ensemble vide

boolean add(object o) // Ajout d'un objet o

int size() //retourne la taille de l'ensemble

boolean contains(object o) // retourne vrai si l'objet o appartient à l'ensemble

void clear() //vide l'ensemble

void remove(object o) //suppression de l'objet o

object[] to-Array() // convertit l'ensemble en un tableau classique

Application

1. On voudrait représenter un nombre rationnel à l'aide d'une classe Rationnel ayant :

   - les attributs suivants : Numérateur, dénominateur (de type entier)
   - les méthodes suivantes:

     Constructeur non paramétré

     Constructeur paramétré

     Accesseur d'attributs

     void Affichage()

     Rationnel Plus (Rationnel R) //ajoute R à un rationnel

     Rationnel Multiplie (Rationnel R) //multiplie un rationnel par R

     Rationnel Divise (Rationnel R) //divise un rationnel par R

2. On voudrait gérer un ensemble de nombres rationnels uniques à l'aide d'un ensemble de type HashSet. Ecrire une classe EnsRationnels, qui prévoie les méthodes d'ajout, suppression et modification d'un rationnel dans l'ensemble.

Héritage et Polymorphisme

Chapitre 5

L'héritage dans JAVA

- Dans JAVA, une classe donnée ne peut posséder qu'une seule super-classe (héritage simple)
- L'héritage est exprimé dans JAVA à l'aide du mot clé "*extends*"
- Syntaxe :

  Class NomSousClasse extends NomClasseMère

Exemple

```
class Point // Super-classe
{
  private int x, y ;
  public Point ()
  {x = 1; y= 1;}
  public void déplacer (int dx, int
  dy)
  {x+=dx; y+=dy;}
  public void afficher ()
  {S.o.p ("(" +x+ "," +y+")"); }

}
```

```
//Sous-Classe
class PointColoré extends Point
{
  private string coul ;
  public Pointcoloré ()
  {coul="Noir";}
 //Redéfinition de la méthode afficher
  public void afficher ()
  {   super.afficher(); // Appelle la méthode
                        //afficher de la classe
  mère
      S.o.p ("couleur=" +coul);
   }
  public void modifCoul (String coul)
  {  this.coul=coul; }
}
```

Exemple

```
public static void main(String[] args])
{
        Point P = new Point () ;
        P.afficher ();
        P.déplacer (2,4);
        PointColoré PC = new PointColoré () ;
        PC.afficher ();
        PC.déplacer (-2,3);
        PC.modifCoul ("rouge");
        P.modifCoul ("rouge");  // ne marche pas
 }
```

Propriétés de l'héritage

- Une sous-classe ne peut pas accéder directement aux propriétés privées de la super-classe
- Elle peut accéder aux propriétés protégées et publiques de la super-classe
- Redéfinition des méthodes ≠ surcharge des méthode (surdéfinition)

Constructeur

```
class Point
{
   private int x, y ;
   public Point (int a, int b)
   {x = a; y= b;}
 }
```

```
public static void main(String[] args])
{
      Point P = new Point (2,3) ;
      PointColoré PC = new
      PointColoré (4, 10, "bleu") ;
 }
```

```
class PointColoré extends Point
{
  private string coul ;
  public Pointcoloré (int a, int b, string c)
  {
  // a et b sont des paramètres nécessaires au
  //déclenchement du constructeur de la super-
  //classe
          super(a,b);
          coul=c;
  }
}
```

Constructeurs

- Lorsque la super-classe possède un constructeur paramétré, la sous-classe doit prévoir aussi un constructeur paramétré respectant les conditions suivantes:

1. Le constructeur de la sous-classe doit comprendre au minimum les paramètres nécessaires à l'exécution du constructeur de la super-classe
2. Le constructeur de la super-classe est déclenché à l'aide de l'instruction *super*

polymorphisme

```
public static void main(String[] args])
{
        Point P = new Point (-,-) ;
        P. afficher();
        PointColoré PC = new PointColoré (-, -, -) ;
        PC.afficher();
        P= new PointColoré (-, -, -) ;
        P.afficher();
 }
```

Le polymorphisme est la capacité d'une même opération (appel de méthode) à pouvoir s'appliquer sur des objets différents appartenant à une même hiérarchie de classes et à pouvoir déclencher des traitements différents.

→ On dit que cette opération est *polymorphe*.

Application-polymorphisme

- Définir la classe Point avec des accesseurs d'attributs, une méthode d'affichage et de saisie.
- Définir la sous-classe PointColoré avec saisie et affichage.
- Ecrire un programme principal qui saisie puis affiche un tableau de points pouvant contenir à la fois des points normaux et des points colorés.

La classe Object

class Point ⟺ class Point extends Object

```
{
…
}
```

- La classe Object est une classe prédéfinie dans JAVA
- Elle représente la classe mère de n'importe quelle classe JAVA prédéfinie ou non
- Elle contient plusieurs méthodes utiles qu'il serait intéressant de les **redéfinir** dans les classes (equals, toString,…)

Redéfinition de la méthode toString

toString : retourne comme résultat la représentation par une chaîne de caractère de la valeur d'un objet donnée

→ utile si on veut afficher des objets à l'aide de S.o.p et sans écrire une méthode d'affichage spécifique

```
class Point
{
   …
   public String toString ()
   {
          String res;
          res = "(" +x+ "," +y+")";
          return(res);
    }
}
```

Méthodes et classes[final]

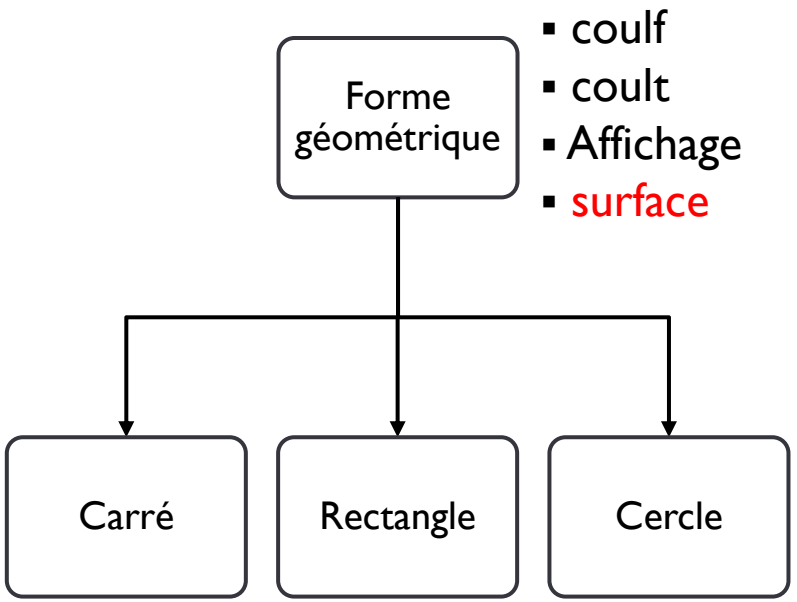
Une méthode déclarée final ne peut plus être redéfinie dans une sous-classe

```
public final type Res NomMeth ()
{
…
}
```

Une classe déclarée final ne peut plus être dérivée en sous-classes

```
final class NomClasse
{
…
}
```

Classes abstraites




```
abstract class Forme
 {
   private String coulf ;
   private String coult ;
   public Forme ()
   {coulf="Blanc"; coult="Noir"; }
   public void Affichage ()
   { S.o.p ("coulf=" +coulf+"coult=" +coult); }
   public abstract double Surface ();
   …
 }
```

```
class Carré extends Forme
 {
   private double coté ;
   public Carré ()
   {coté=1;}
   public void Affichage ()
   {        super.Affichage ();
            S.o.p ("coté=" +coté);
   }
   public double Surface ();
   return(coté*coté);
 }
```

Classes abstraites

- Une méthode déclarée abstract dans une classe mère:
  - n'a pas de corps (juste une entête : signature)
  - n'est pas exécutable
  - doit obligatoirement être redéfinie dans une classe fille
- Dans JAVA, une classe contenant au moins une méthode abstraite doit être déclarée comme une classe abstraite