BTS
BREVET DE TECHNICIEN SUPÉRIEUR
LYCÉE QUALIFIANT MOHAMED V
ESSAOUIRA

# PROGRAMMATION ORIENTÉE OBJET (JAVA)

# Partie 2: POO (java)

## Héritage et polymorphisme

1. Définition de l'héritage
2. Déclaration de l'héritage
3. Propriétés de l'héritage
4. Référence à la super classe
5. Premier exemple
6. Masquage des attributs et redéfinition des méthodes
7. Le Polymorphisme
8. Classe abstraite
9. Interfaces

# 1. Définition de l'héritage

* L'héritage est l'un des aspects les plus utiles de la POO. Il permet à une classe de transmettre ses attributs et ses méthodes à des sous classes.
* Si une classe **B** hérite d'une classe **A**, Alors la classe **B** contient implicitement l'ensemble des attributs non privés de **A** et peut invoquer les méthodes non privées de **A**.

# 2. Déclaration de l'héritage

* En java, pour déclarer l'héritage, on utilise le mot clé **extends**.

* Syntaxe:

**class B extends A**

# 3. Propriétés de l'héritage

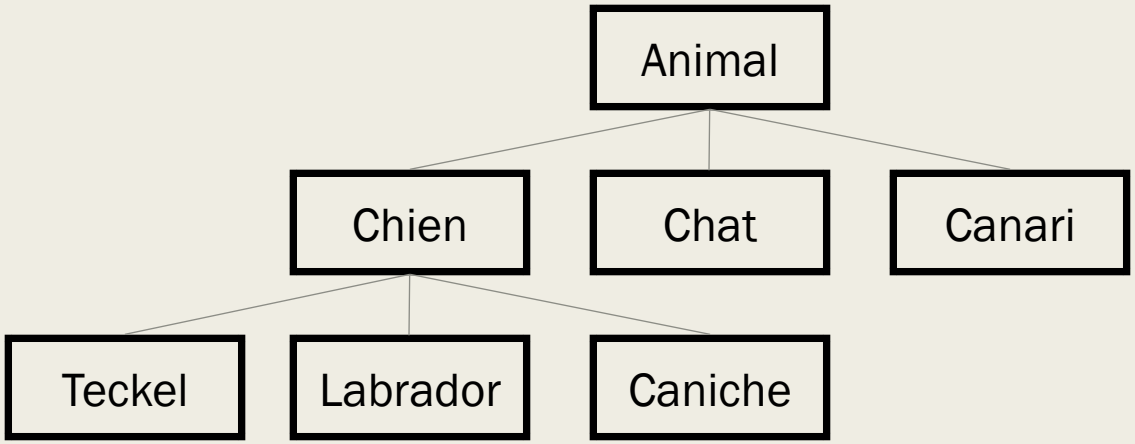
* La classe dérivée hérite tous les attributs et les méthodes non privés de sa super classe.
* Tout objet de la classe dérivée est aussi un objet de la super classe.
* Si une classe n'hérite d'aucune autre classe, elle hérite par défaut de la super classe **Object** définie dans le package **java.lang**
* Si la déclaration d'une classe est précédée de **final**, alors cette classe ne peut pas être une classe dérivée.

# 4. Référence à la super classe

* Pour faire référence à la classe parente (super classe), il suffit d'utiliser le mot clé **super**.
* Ce mot permet d'invoquer un attribut ou une méthode de la super classe.
* On utilise **super(arguments)** pour invoquer le constructeur de la super classe ayant les arguments correspondants.

# 5. Premier exemple

* Prenons la hiérarchie suivante:



# 5. Premier exemple

```
class Animal
{  boolean vivant;
   int age;
   Animal(int a)
   { age=a; vivant=true;
   System.out.println("Un animal de"+a+"an(s) vient
   d'être crée");}
   void vieillit()
   { ++age;
     System.out.println("C'est l'anniversaire de cet
   animal");
     System.out.println("Il a maintenant"+age+"an(s)");
    }
   void meurt()
   {vivant=false;
    System.out.println("Cet animal est mort");
   }
   void crie()
   {
   }
}
```

```
class Canari extends Animal
{
   Canari(int a)
   {super(a);}
   void crie()
   { System.out.println("Le canard crie Cui Cui!");}
}
class Chien extends Animal
{
    Chien(int a)
   {super(a);}
   void crie()
   { System.out.println("Le chien crie Hab Hab!");}
}
class Caniche extends Chien
{
    Caniche(int a)
   {super(a);}
}
```

# 5. Premier exemple

```
class Animaux
{
   public static void main(String[] arg)
   {
      System.out.println(″Utilisation d'un objet Canari″);
      Canari titi = new Canari(3);
      titi.vieillit();
      titi.crie();

      System.out.println(″Utilisation d'un objet Chien″);
      Chien bobi = new Chien(2);
      bobi.vieillit();
      bobi.crie();

      System.out.println(″Utilisation d'un objet Caniche″);
      Caniche teddy = new Caniche(1);
      teddy.vieillit();
      teddy.crie();
   }
}
```

A l'exécution, on aura:

```
Utilisation d'un objet Canari
Un animal de 3 an(s) vient d'être crée
C'est l'anniversaire de cet animal
Il a maintenant 4 an(s)
Le canard crie Cui Cui !

Utilisation d'un objet Chien
Un animal de 2 an(s) vient d'être crée
C'est l'anniversaire de cet animal
Il a maintenant 3 an(s)
Le chien crie Hab Hab !

Utilisation d'un objet Caniche
Un animal de 1 an(s) vient d'être crée
C'est l'anniversaire de cet animal
Il a maintenant 2 an(s)
Le chien crie Hab Hab !
```

# 6. Masquage des attributs et redéfinition des méthodes

* On dit qu'un attribut d'une classe masque un attribut d'une super classe s'il a le même nom qu'un attribut de la super classe.
* On dit qu'une classe redéfinit une méthode d'une super classe si elle définie une méthode ayant même nom, même suite de types d'arguments et même valeur de retour qu'une méthode de la super classe.

# 7. Le polymorphisme

Le terme **polymorphisme** décrit la caractéristique d'un élément qui peut prendre plusieurs formes, comme l'eau qui se trouve à l'état solide, liquide ou gazeux.

En programmation Objet, on appelle polymorphisme

- le fait qu'un objet d'une classe puisse être manipulé comme s'il appartenait à une autre classe.

- le fait que la même opération puisse se comporter différemment sur différentes classes de la hiérarchie.

- Le polymorphisme constitue la troisième caractéristique essentielle d'un langage orienté objet après l'abstraction des données (encapsulation) et l'héritage.

En utilisant le polymorphisme en association à la liaison dynamique plus besoin de distinguer différents cas.

- Développement **plus rapide**
- Plus grande **simplicité et meilleure organisation du code**
- Programmes plus **facilement extensibles**
- Maintenance du code plus aisée

# 7. Le polymorphisme

```
class A
{
   int attr=10;
   void meth()
   {
      System.out.println("je suis la méthode de A");
   }
}
class B extends A
{
   int attr=20;
   void meth()
   {
      System.out.println("je suis la méthode de B");
   }
}
```

```
public  class Masque
{
   public static void main(String[] arg)
   {
        A a;
        B b = new B();
        a = b; // car b est aussi un objet de la classe A
        System.out.println("b.attr = "+b.attr);
        System.out.println("a.attr = "+a.attr);
        b.meth();
        a.meth();
   }
}
```

---

A l'exécution:

```
b.attr = 20
a.attr = 10
je suis la méthode de B
je suis la méthode de B
```

# 8. Classe abstraite

* Une classe abstraite est une classe qui contient au moins une méthode abstraite, elle doit être déclarée avec le mot **abstract**.
* Une méthode est dite abstraite si seul son prototype figure. Elle n'a pas de définition explicite.
* Une classe abstraite ne peut pas être instanciée, il faut l'étendre pour pouvoir l'utiliser.

# 8. Classe abstraite

```
abstract class Forme //————————————————
{   abstract float perim();
    abstract float surf();
    abstract void affiche(); }
class Disque extends Forme //————————————
{   private int rayon;
    Disque(int rayon)
    { this.rayon=rayon; }
    float perim() {return (float) (2*Math.PI*rayon);}
    float surf()    {return (float) (Math.PI*rayon*rayon);}
     void affiche() {System.out.println("Disque de rayon
    :"+rayon+"Perim :"+perim()+"Surf:"+surf());}
}
class Rectangle extends Forme //—————————————
{   private int long,lar;
    Rectangle(int long,int larg)
    { this.long=long; this.larg=larg; }
    float perim()  {return 2*(long+larg);}
    float surf()  {return long*larg;}
 void affiche() {System.out.println("Rectangle de: long=
    "+long+"larg="+larg+"Perim :"+perim()+"Surf:"+surf());}
}
```

```
class EssaiFormes
{
    public static void main(String[] arg)
    {
        Forme[] desFormes = new Forme[4];
        desFormes[0] = new Disque(2);
        desFormes[1] = new Disque(5);
        desFormes[2] = new Rectangle(2,3);
        desFormes[3] = new Rectangle(5,7);

       for(int i=0;i<4;i++)
        {
           desFormes[i].affiche();
        }
    }
}
```

# 9. Interfaces

## 1. **Définition:**

Une interface est une sorte de classe qui ne contient que des prototypes de méthodes et des constantes ayant les modificateurs **static** et **final**. Elle sert à regrouper des constantes et à traiter des objets qui ne sont pas tous de la même classe mais qui ont en commun une partie. Cette partie sera gérée par l'interface.

# 9. Interfaces

## 2. Présentation:

Une interface se compose de deux parties: **En-tête** et **Corps**.

[modificateur] **interface** <Nom> [**extends** <interface>]

{

déclarations des constantes et des prototypes de méthodes

}

Remarque: Une interface peut dériver de plusieurs autres interfaces (héritage multiples pour les interfaces et non pas pour les classes)

# 9. Interfaces

**3. Propriétés:**

→ Pour dériver une classe d'une interface, on doit utiliser le mot : **implements**.

Exemple: class A implements I

- Toutes les méthodes figurant dans l'interface I doivent être déclarées publiques (avec le modificateur public) et définies par la classe A.

→ Si une classe A implémente une interface I, les sous classes de A implémentent aussi automatiquement I.

# 9. Interfaces

## 4. Exemple d'utilisation d'une interface:

```
interface AfficheTexte()
{
    void affiche();
}
Class Appartement implements AfficheTexte
{
    public void affiche()
    {
        System.out.printn("Bel appartement");
    }
}
Class Maison implements AfficheTexte
{
    public void affiche()
    {
        System.out.printn("Maison n'est pas mal non plus");
    }
}
```

```
public  class UtiliseInterface
{
    public static void main(String[] arg)
    {
        Appartement appart = new Appartement();
        Maison demeure = new Maison();
        appart.affiche();
        demeure.affiche();
    }
}
```

---

A l'exécution:

Bel appartement

Maison n'est pas mal non plus