Programmation Objet – Initiation à Java

Letícia SEIXAS PEREIRA

Cours 08 – Interfaces + Classes abstraites

Concepts de base de l'OO

1. Encapsulation
2. Héritage
3. Polymorphisme
4. **Abstraction**

Interfaces

- Un mécanisme qui permet de réunir au sein d'une même classe la mise en œuvre de spécifications d'origines multiples.

Interfaces

- Un mécanisme qui permet de réunir au sein d'une même classe la mise en œuvre de spécifications d'origines multiples.
- Un prototype de classe sans aucune implémentation:

Interfaces

- Un mécanisme qui permet de réunir au sein d'une même classe la mise en œuvre de spécifications d'origines multiples.
- Un prototype de classe sans aucune implémentation:
  - Son but est de spécifier de façon formelle une capacité, un comportement dont l'implémentation est laissée aux classes qui le nécessitent;
  - Ainsi, la définition d'une Interface ne contient que des en-têtes de méthodes, des variables **final** et aucun constructeur.

Interfaces

- Une Interface:
  - contient des signatures de méthodes;
  - peut hériter d'une ou plusieurs interface(s);
  - ne peut pas être instanciée;
- Une classe peut implémenter plusieurs Interfaces.

Interfaces

- **Syntaxe:**

```
public interface NomInterface{
    //…
}
```

Interfaces

- **Syntaxe:**

```
public interface NomInterface{
    //…
}
public interface NomInterface extends AutreInterface{
    //…
}
```

Interfaces

- **Syntaxe:**

```
public interface NomInterface{
   //…
}
public interface NomInterface extends AutreInterface{
   //…
}
public class NomClass implements NomInterface{
   //…
}
```

Interfaces

- **Exemple:**

- Dans notre exemple, Forme peut être une Interface décrivant les méthodes qui doivent être implémentées par les classes Rectangle et Carre.

```
public interface Forme {
    //…
}
```

Interfaces

- **Exemple:**

- Dans notre exemple, Forme peut être une Interface décrivant les méthodes qui doivent être implémentées par les classes Rectangle et Carre.

```
public interface Forme {

    //…

}
```

```
Forme.java:3: error: interface
abstract methods cannot have body
    public void surface(){
                          ^
1 error
```

Interfaces

- **Exemple:**

- Dans notre exemple, Forme peut être une Interface décrivant les méthodes qui doivent être implémentées par les classes Rectangle et Carre.

```
public interface Forme {

    //…

}
```

```
Forme.java:3: error: interface
abstract methods cannot have body
    public void surface(){
                          ^
1 error
```

« Une interface décrit un ensemble de signatures de méthodes, sans implémentation »

Interfaces

- **Exemple:**

- Fichier: Forme.java

```
public interface Forme {
        public int surface() ;
        public int perimetre() ;
}
```

Interfaces

- **Exemple:**

```
public interface Forme {
        public int surface() ;
        public int perimetre() ;
}
```

- Fichier: Rectangle.java

```
public class Rectangle implements Forme {

public int surface(){
            //…
    }
    public int perimetre(){
            //…
    }
}
```

Interfaces

```
public interface Forme {
        public int surface() ;
        public int perimetre() ;
}
```

- **Remarques:**
  - Le polymorphisme s'applique aussi pour les interfaces;
  - Les variables déclarés dans une interface sont implicitement **public**, **static** et **final**;
  - Les méthodes déclarées dans une interface ne peuvent être ni **static** ni **final**;
  - Les méthodes déclarées dans une interface ne contiennent pas d'implémentation;
    - (Elles se terminent par ";" et non pas par "{}")
  - **Une classe implémentant une interface doit implémenté toutes ses méthodes.**

Exercice d’application 1

```java
public class Forme {

}
```

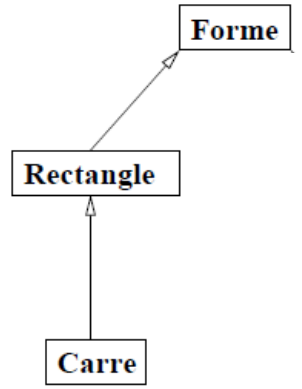
```java
public class Rectangle extends Forme{
	private int longueur ;
	private int largeur ;

	public Rectangle(int longueur, int largeur){
		this.longueur = longueur;
		this.largeur = largeur;
	}

	public int getLongueur(){ return this.longueur; }
	public int getLargeur(){ return this.largeur;}

	public void setLongueur(int longueur){ this.longueur = longueur;}
	public void setLargeur(int largeur){ this.largeur = largeur;}

	public String toString(){
		return "Rectangle: " + this.longueur + "x" + this.largeur;
	}
	public int surface () {
		return this.longueur * this.largeur ;
	}
}
```




```java
public class Carre extends Rectangle {

	private int cote;

	public Carre (int cote) {
		super (cote, cote);
		this.cote = cote;
	}

	public int getCote(){ return this.cote; }

	public void setCote(int c){ this.cote = c; }

	public String toString(){
		return "Carre: " + this.cote;
	}
}
```

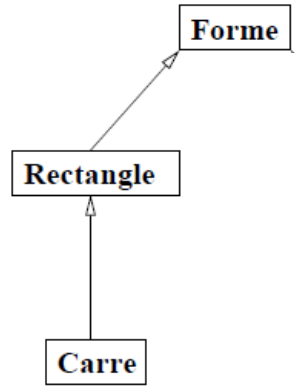
Exercice d'application 1

1. Transformez la classe **Forme** dans une **Interface**;
2. Ajoutez la signature des méthodes **surface()** et **perimetre()** dans l'interface **Forme**;
3. Effectuez les modifications nécessaires dans les classes **Rectangle** et **Carré**;
4. Écrivez une application de test pour créer des Rectangles et des Carrés.

Exemple de sortie:

```
Rectangle: 5x10
Surface : 50
Perimetre : 30
Carre: 3
Surface : 9
Perimetre : 12
```

```java
public interface Forme {
	int surface();
	int perimetre();
}
```




```java
public class Rectangle implements Forme{
	//...
	public int perimetre(){
		return  2*(this.longueur + this.largeur);
	}
}
```

```java
public class Carre extends Rectangle {

	private int cote;
	public Carre (int cote) {
		super (cote, cote);
		this.cote = cote;
	}
	public int getCote(){ return this.cote; }
	public void setCote(int c){ this.cote = c; }
	public String toString(){
		return "Carre: " + this.cote;
	}
}
```

```java
public class MainFormes {
	public static void main(String[] args) {

		Rectangle rectangle1 = new Rectangle(5,10);
		Carre carre1 = new Carre(3);

		System.out.println(rectangle1);
		System.out.println("Surface : " + rectangle1.surface());
		System.out.println("Perimetre : " + rectangle1.perimetre());

		System.out.println(carre1);
		System.out.println("Surface : " + carre1.surface());
		System.out.println("Perimetre : " + carre1.perimetre());

	}
}
```

Classes abstraites

Classes abstraites

- Le concept de classe abstraite se situe entre celui de classe et celui d'interface;
  - Une classe abstraite peut contenir:
    - Des variables;
    - Des méthodes implémentées;
    - Des signatures de méthodes à implémenter (abstraites).

Classes abstraites

- Le concept de classe abstraite se situe entre celui de classe et celui d'interface;
  - Une classe abstraite peut contenir:
    - Des variables;
    - Des méthodes implémentées;
    - Des signatures de méthodes à implémenter (abstraites).
- Donc, elles sont des **classes qui ne peuvent pas être directement instanciées** car certaines de ses méthodes peuvent ne pas être implémentées.

Classes abstraites

- Une classe abstraite peut:
  - implémenter des interfaces;
  - hériter d'une classe ou d'une classe abstraite.

Classes abstraites

- **Syntaxe:**
  - Le mot-clé **abstract** est utilisé:
    - Devant le mot-clé class pour déclarer une classe abstraite;
    - Dans les signatures des méthodes à implémenter.

Classes abstraites

- **Syntaxe:**
  - Le mot-clé **abstract** est utilisé:
    - Devant le mot-clé class pour déclarer une classe abstraite;

```
public abstract class NomClasse {



}
```

Classes abstraites

- **Syntaxe:**
  - Le mot-clé **abstract** est utilisé:
    - Devant le mot-clé class pour déclarer une classe abstraite;
    - Dans les signatures des méthodes à implémenter.

```
public abstract class NomClasse {
    //…



    public abstract <type> nomMethode();
}
```

Classes abstraites

- **Syntaxe:**
  - Le mot-clé **abstract** est utilisé:
    - Devant le mot-clé class pour déclarer une classe abstraite;
    - Dans les signatures des méthodes à implémenter.

```
public abstract class NomClasse {
     //…
     public <type> methodeImplemente(){
          //...
     }
     public abstract <type> nomMethode();
}
```

Classes abstraites

- Lorsqu'une classe hérite d'une classe abstraite, elle doit :

```
public abstract class NomClasse {
    //…
    public <type> methodeImplemente(){
        //...
    }
    public abstract <type> methodeNonImplemente();
}
```

```
public class ClasseFille extends NomClasse{
    //…

}
```

Classes abstraites

- Lorsqu'une classe hérite d'une classe abstraite, elle doit :
  - Soit implémenter les méthodes abstraites de sa classe mère en les dotant d'un corps ;

```
public abstract class NomClasse {
    //…
    public <type> methodeImplemente(){
        //...
    }
    public abstract <type> methodeNonImplemente();
}
```

```
public class ClasseFille extends NomClasse{
    //…
    public <type> methodeNonImplemente(){
        //…
    }
}
```

Classes abstraites

- Lorsqu'une classe hérite d'une classe abstraite, elle doit :
  - Soit implémenter les méthodes abstraites de sa classe mère en les dotant d'un corps ;
  - Soit être elle-même abstraite si au moins une des méthodes abstraites de sa classe mère reste abstraite.

```
public abstract class NomClasse {
    //…
    public <type> methodeImplemente(){
        //...
    }
    public abstract <type> methodeNonImplemente();
}
```

```
public abstract class ClasseFille extends NomClasse{
    //…
    public abstract <type> methodeNonImplemente();
}
```

Classes abstraites

- **Remarques:**
  - Une classe peut être marqué abstraite même si toutes ses méthodes sont concrètes;
  - Une classe abstraite n'a pas de constructeur (car elle ne peut être instanciée);

Classes abstraites

- **Remarques:**
  - Une classe peut être marqué abstraite même si toutes ses méthodes sont concrètes;
  - Une classe abstraite n'a pas de constructeur (car elle ne peut être instanciée);
  - Une méthode abstraite ne peut être marqué ni **final** ni **private**;
  - Une classe abstraite ne peut être marqué **final:**

Classes abstraites

- **Remarques:**
  - Une classe peut être marqué abstraite même si toutes ses méthodes sont concrètes;
  - Une classe abstraite n'a pas de constructeur (car elle ne peut être instanciée);
  - Une méthode abstraite ne peut être marqué ni **final** ni **private**;
  - Une classe abstraite ne peut être marqué **final:**
    - L'unique objectif d'une classe abstraite est d'être héritée.

Classes abstraites

- Exemple:
  - Fichier: Forme.java
  - Imaginons que l'on souhaite attribuer deux variables, origineX et origineY, à tout objet représentant une forme:

```
public abstract class Forme {
     protected int origineX;
     protected int origineY;

     public Forme (int origineX, int origineY) {
          this.origineX = oroginex;
          this.origineY = orogineY;
     }
     public abstract int surface() ;
     public abstract int perimetre() ;
}
```

Classes abstraites

- Exemple:
  - Fichier: Forme.java
  - Imaginons que l'on souhaite attribuer deux variables, origineX et origineY, à tout objet représentant une forme:

```
public abstract class Forme {
     protected int origineX;
     protected int origineY;

     public Forme (int origineX, int origineY) {
          this.origineX = oroginex;
          this.origineY = orogineY;
     }
     public abstract int surface() ;
     public abstract int perimetre() ;
}
```

```
public class Rectangle extends Forme{
     //…
}

public class Carre extends Rectangle{
     //…
}
```

Interfaces vs classes abstraites

Interfaces vs classes abstraites

- Un interface peut être considérée comme une forme de classe abstraite qui:
  - Ne dispose d'aucun attribut (à l'exception des attributs statiques constants);
  - Ne dispose d'aucun constructeur ou blocs d'initialisation;

- Comme une classe abstraite une interface ne peut pas être déclarée **final**;

- **Une interface peut hériter d'une ou plusieurs autres interfaces MAIS une interface ne peut pas hériter d'une classe.**

Interfaces vs classes abstraites

	Interface	Classe abstraite
déclaration	`interface I {/*…*/}`	`abstract class A {/*…*/}`
héritage simple	`interface J extends I {/*…*/}`	`abstract class B extends A {/*…*/}`
héritage multiple	`interface K extends I, J {/*…*/}`	**INTERDIT**

attribut publique statique constant	`// déclarations équivalentes:`
	`public static final int i = 0;`
	`static final int i = 0;`
	`final int i = 0;`
	`int i = 0;`
		`// déclarations NON-équivalentes:`
		`public static final int i = 0;`
		`static final int i = 0; // NON PUBLIC`
		`final int i = 0; // NON STATIQUE`
		`int i = 0; // NON CONSTANT`

attribut protégée, ou d'instance ou non constant	**INTERDIT**	`protected int i;`
		`final int i;`
		`static int k;`

code d'initialisation	**INTERDIT**	`A() {/*Constructeur*/}`

Lequel utiliser, classes abstraites ou interfaces?

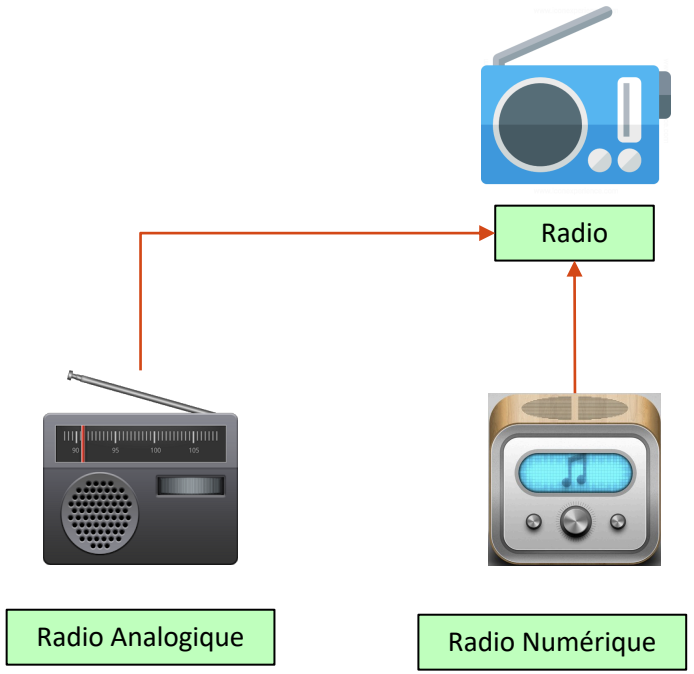
Lequel utiliser, classes abstraites ou interfaces?

- Utilisez des classes abstraites si:
  - On veut partager du code entre plusieurs classes liées;
  - Les classes qui héritent de cette classe abstraite aient beaucoup de méthodes ou d'attributs communs, ou nécessitent des modificateurs d'accès autres que publics (tels que *protected* et *private*);
  - On veut déclarer des attributs *non-statiques* ou *non-finaux* (Cela permet de définir des méthodes qui peuvent accéder et modifier l'état de l'objet auquel elles appartiennent – *attributs d'instance*).

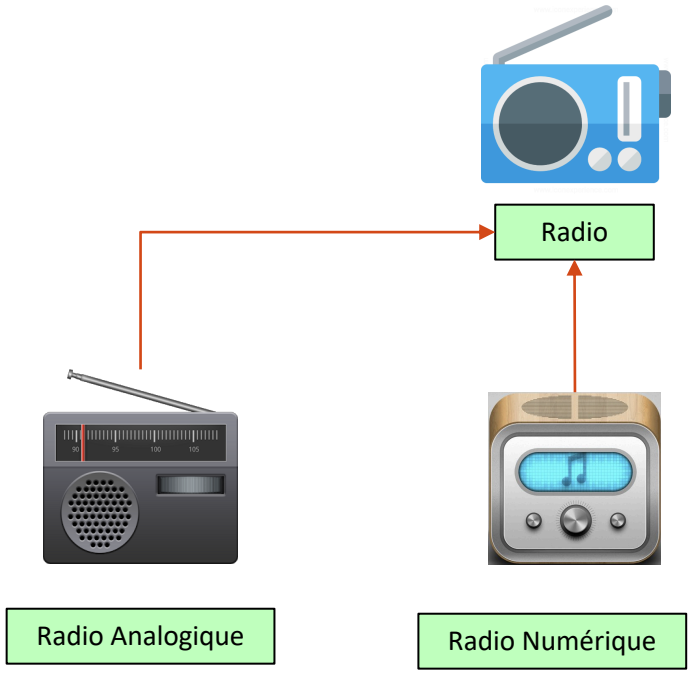
Lequel utiliser, classes abstraites ou interfaces?

- Utilisez des classes abstraites si:
  - On veut partager du code entre plusieurs classes liées;
  - Les classes qui héritent de cette classe abstraite aient beaucoup de méthodes ou d'attributs communs, ou nécessitent des modificateurs d'accès autres que publics (tels que *protected* et *private*);
  - On veut déclarer des attributs *non-statiques* ou *non-finaux* (Cela permet de définir des méthodes qui peuvent accéder et modifier l'état de l'objet auquel elles appartiennent – *attributs d'instance*).
- Utilisez des interfaces si:
  - Il est prévu que les classes non liées mettraient en œuvre cette interface;
  - On veut spécifier le comportement d'un type de données particulier, mais ne pas se préoccuper de savoir qui implémente son comportement;
  - On veut profiter de l'héritage multiple.

Interfaces et classes abstraites

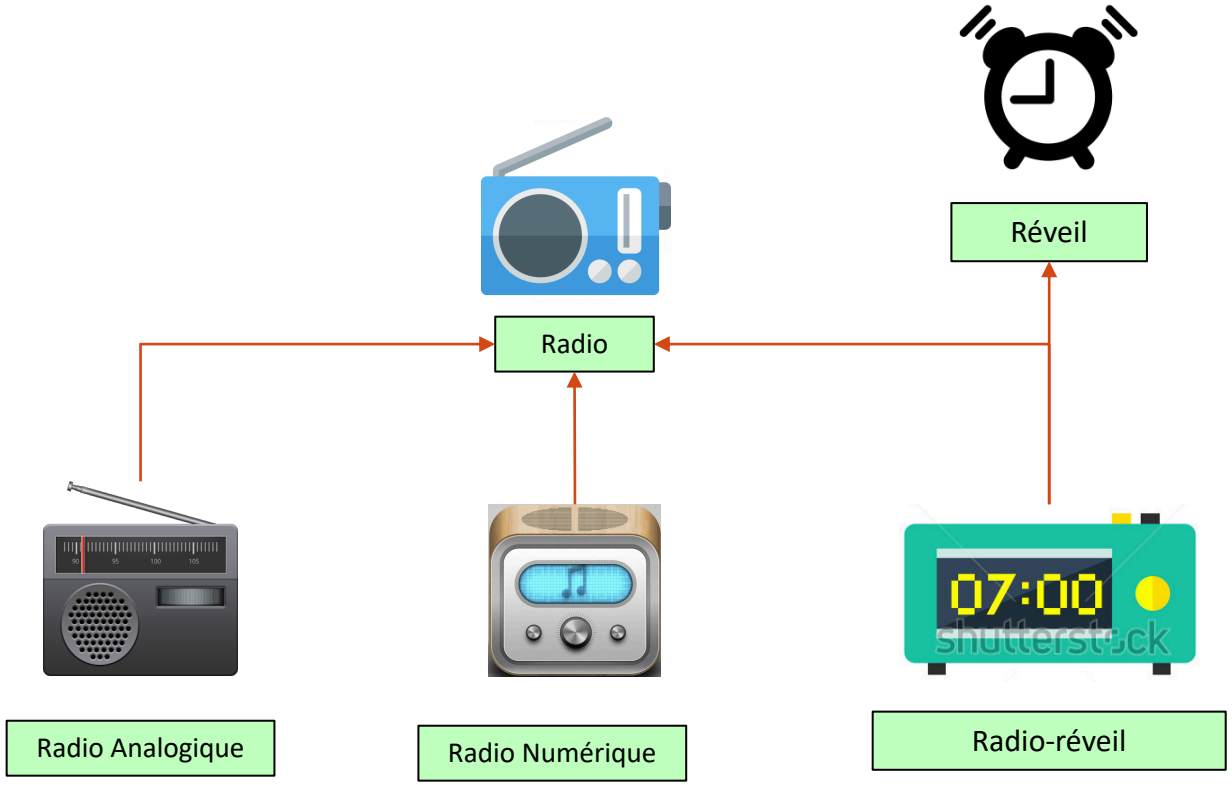


Interfaces et classes abstraites

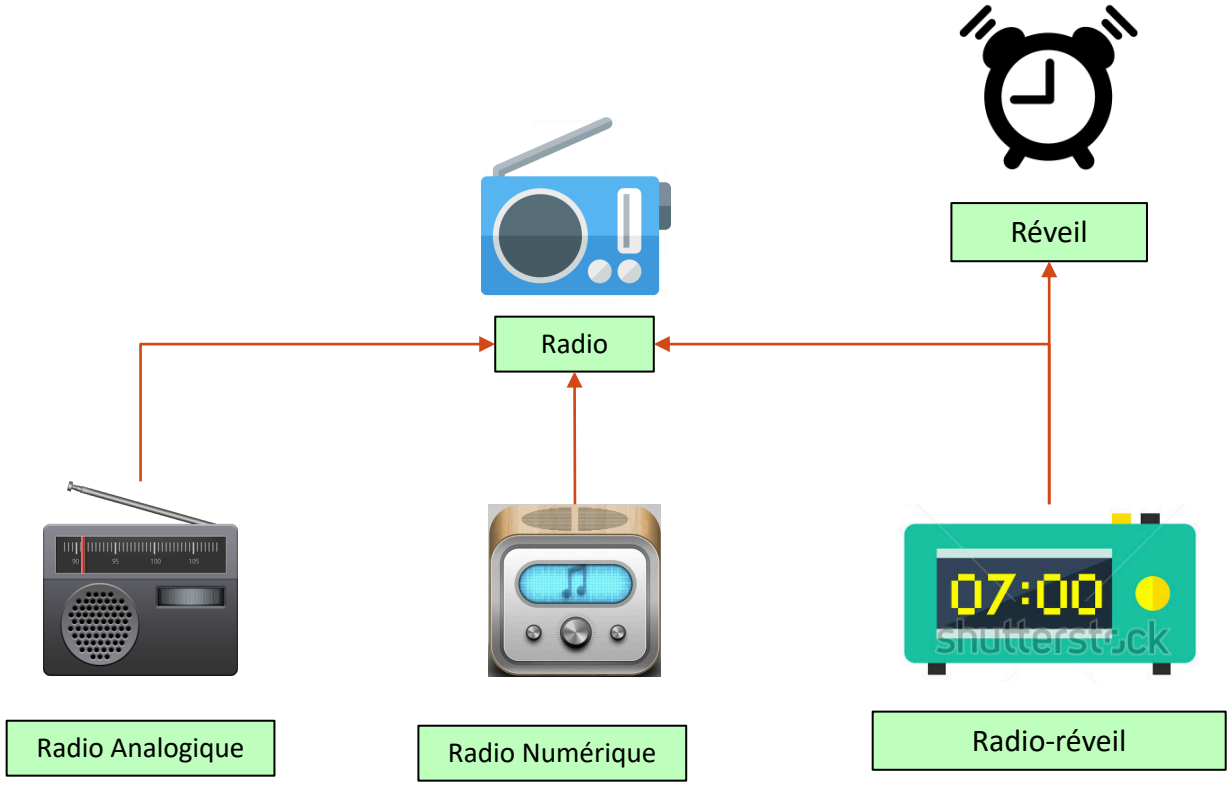



```
abstract class Radio { }
class RadioAnalogique extends Radio { }
class RadioNumerique extends Radio { }
```

Interfaces et classes abstraites



Interfaces et classes abstraites




```
interface Radio { }
interface Reveil { }
class RadioAnalogique implements Radio { }
class RadioNumerique implements Radio { }
class RadioReveil implements Radio, Reveil { }
```

Exercice d'application 2

1. Transformez la classe **Forme** dans une **Classe Abstraite**;

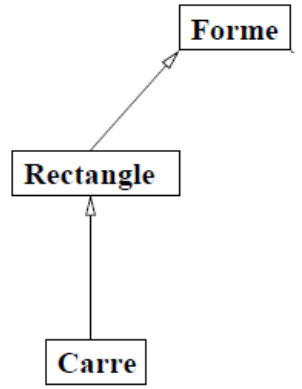
2. Effectuez les modifications nécessaires dans les classes **Rectangle** et **Carré**;

3. Effectuez les modifications nécessaires dans l'application de test avec des Rectangles et des Carrés.

Exemple de sortie:

```
Rectangle: 5x10
Surface : 50
Perimetre : 30
Carre: 3
Surface : 9
Perimetre : 12
```

```java
public abstract class Forme {
    abstract int surface();
    abstract int perimetre();
}
```




```java
public class Rectangle extends Forme{
    //...
    public int perimetre(){
        return  2*(this.longueur + this.largeur);
    }
}
```

```java
public class Carre extends Rectangle {

    private int cote;
    public Carre (int cote) {
        super (cote, cote);
        this.cote = cote;
    }
    public int getCote(){ return this.cote; }
    public void setCote(int c){ this.cote = c; }
    public String toString(){
        return "Carre: " + this.cote;
    }
}
```

```java
public class MainFormes {
    public static void main(String[] args) {

        Rectangle rectangle1 = new Rectangle(5,10);
        Carre carre1 = new Carre(3);

        System.out.println(rectangle1);
        System.out.println("Surface : " + rectangle1.surface());
        System.out.println("Perimetre : " + rectangle1.perimetre());

        System.out.println(carre1);
        System.out.println("Surface : " + carre1.surface());
        System.out.println("Perimetre : " + carre1.perimetre());

    }
}
```