Classes et méthodes abstraites

- Une classe abstraite ne doit pas être instanciée. Il faut l'étendre pour pouvoir l'instancier.
- Une méthode est dite abstraite si elle est uniquement déclarée et non définie.
- Une méthode abstraite ne peut être déclarée que dans une classe abstraite

Classes et méthodes abstraites

- En java on utilise le mot clé **`abstract`**:

```
abstract class NomClasse{
        ….
        abstract int methode();   // methode
                                  // abstraite
         ….
}
```

Classes et méthodes abstraites

- Utilité des classes abstraites :

Définir des comportements incomplets qui devront être implémentés dans les sous-classes

Factoriser le code et faire apparaître des comportements généraux

Classes et méthodes abstraites

- Une classe abstraite est une description d'objets destinée à être héritée par des classes plus spécialisées
- Pour être utile, une classe abstraite doit admettre des classes descendantes concrètes

Classes et méthodes abstraites

- Toute classe concrète sous-classe d'une classe abstraite doit implémenter toutes les méthodes abstraites de cette dernière
- Une classe abstraite permet de regrouper certaines caractéristiques communes à ses sous-classes et définit un comportement minimal commun

Classes et méthodes abstraites

- La factorisation optimale des propriétés communes à plusieurs classes par généralisation nécessite le plus souvent l'utilisation de classes abstraites

Classes et méthodes abstraites

```
abstract class Forme{
  abstract float perimetre();
  abstract float surface();
  void decritEtalement(){
    float p = this.perimetre();
    if (this.surface() >= (p*p)/16){
      System.out.println("étalement supérieur au carre ");
   }else{
      System.out.println("étalement  inférieur au carre ");
    }
   }
}
```

Classes et méthodes abstraites

```
class Rectangle extends Forme{
  private int longueur,largeur;
  public Rectangle(int lo, int la){
        longueur = lo;   largeur = la;
  }
  public float perimetre(){
        return 2*(longueur+largeur);
  }
  public float surface(){
        return longueur*largeur;
  }
}
```

Règle de visibilité

- *static* : pour définir les attributs et méthodes de classe
- *public* : pour définir des attributs et méthodes visibles de partout
- *protected* : pour définir des attributs et méthodes visibles dans la classe et dans les sous-classes
- *private* : pour définir des attributs et méthodes visibles uniquement dans la classe