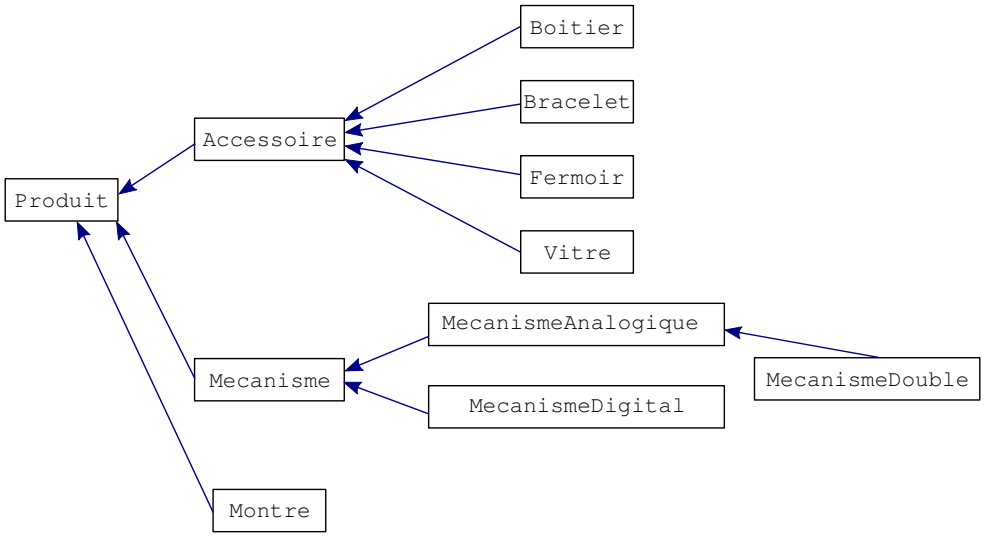
Rappel du problème

- Les montres ont un *mécanisme* de base [...]
- Les *mécanismes* [...] sont aussi des produits
- Il existe trois sortes de *mécanismes* : *analogiques*, *digitaux* et *doubles*.
- Pour les *mécanismes doubles*, on supposera ici qu'ils n'indiquent qu'une seule heure, mais se comportent sinon à la fois comme des *mécanismes analogiques* et comme des *mécanismes digitaux*

```
class Mecanisme extends Produit {
}

class MecanismeAnalogique extends Mecanisme {
}

class MecanismeDigital extends Mecanisme {
}

class MecanismeDouble extends Mecanisme {
}

// ========================================
class Montre extends Produit {
  // ...
  private Mecanisme coeur;
  // ...
}
```

Hiérarchie de classe



Première révision de la hiérarchie

```
class Mecanisme extends Produit {
   private String heure;
}
class MecanismeAnalogique extends Mecanisme {
   private int date;
}

class MecanismeDigital extends Mecanisme {
   private String reveil;
}

// Un mécanisme double est avant tout un mécanisme
// analogique auquel on ajoute des caractéristiques
// d'un mécanisme digital
class MecanismeDouble extends MecanismeAnalogique
{
   // duplication incontournable ici
   private String reveil;
}
```

Construction d'un `Mecanisme`

Pour le moment, il n'y a qu'un constructeur par défaut, mais fixons la construction des `Mecanisme` :

- initialisation de la valeur de base (`Produit`)
- initialisation de l'heure

```
class Mecanisme extends Produit {
   private  String heure;

   public Mecanisme(double valeurDeBase, String uneHeure) {
      super(valeurDeBase);
      heure = uneHeure;
   }
   public Mecanisme(double valeurDeBase){
      super(valeurDeBase);
      heure = "12:00";
   }
}
```

Constructeurs des sous-classes

```
class MecanismeAnalogique extends Mecanisme {
   private int date;
   public MecanismeAnalogique(double valeurDeBase, String uneHeure, int uneDate) {
      super (valeurDeBase, uneHeure);
      date = uneDate;
   }
}
class MecanismeDigital extends  Mecanisme {
   private String reveil;
   public  MecanismeDigital(double valeurDeBase, String uneHeure, String heureRev) {
      super(valeurDeBase, uneHeure);
      reveil = heureRev;
   }
}
// ...
```

Constructeurs des sous-classes (suite)

```
class MecanismeDouble extends MecanismeAnalogique {
   private String reveil;
   public MecanismeDouble(double valeurDeBase, String uneHeure,
                          int uneDate, String heureReveil) {
      super(valeurDeBase, uneHeure, uneDate);
      reveil = heureReveil;
   }
}
```

Gestion de la valeur par défaut de la super-classe

```
class MecanismeDouble extends MecanismeAnalogique implements ReveilDigital {

   public MecanismeDouble(double valeurDeBase, String uneHeure, int uneDate,
                          String heureReveil) {
      super(valeurDeBase, uneHeure, uneDate);
      reveil = heureReveil;
   }

   // gestion propre de la valeur par défaut de l'heure (super-classe)
   public MecanismeDouble(double valeurDeBase, int uneDate,
                          String heureReveil) {
      super(valeurDeBase, uneDate);
      reveil = heureReveil;
   }
}
```

Gestion de la valeur par défaut de la super-classe

```
public MecanismeDouble(double valeurDeBase, String uneHeure, int uneDate,
                           String heureReveil)
{
 super(valeurDeBase, uneHeure, uneDate);
 reveil = heureReveil;
}

public // gestion propre de la valeur par défaut de l'heure (super-classe)
MecanismeDouble(double valeurDeBase, int uneDate, String heureReveil) {
  super(valeurDeBase, uneDate);
  reveil = heureReveil;
}
```

Construction d'une `Montre`

Complétons maintenant nos constructeurs pour la classe `Montre` :

```
class Montre extends Produit {
   private Mecanisme coeur;
   private ArrayList<Accessoire> accessoires;
   //..
   public Montre(Mecanisme depart)
   {
      coeur = depart; // nous y reviendrons
      accessoires = new ArrayList<Accessoire>();
   }
   //..
}
```

Affichage des `Mecanisme`s

Supposons que l'on veuille :

- que tous les mécanismes s'affichent suivant le **même** schéma, **imposé** et non modifiable

  Par exemple :
  affichage du type de mécanisme, suivi d'un affichage du cadran (heure, date, heure de réveil, ...), suivi du prix
- mais que chaque partie de ce schéma soit adaptable
- offrir une version par défaut, utilisable dans les sous-classes, de l'affichage du cadran (par exemple affichage de l'heure)
- imposer la redéfinition de l'affichage du type de mécanisme

☞ Comment faire ?

Affichage des `Mecanisme`s : niveau général

```
abstract class Mecanisme extends Produit {
   //...
      // Tous les mécanismes DOIVENT s'afficher comme ceci
      public final String toString()  {
         String result = "mécanisme ";
         result += toStringType();
         result += " (affichage : ";
         result += toStringCadran();
         result += "), prix : ";
         result += super.toString();
         return result;
      }
      // on veut offrir la version par défaut aux sous-classes et aux classes
   // du même paquetage. Par défaut, on affiche juste l'heure.
      protected String toStringCadran() {
         return heure;
      }
      // Un mécanisme, ici à ce niveau, est abstrait (= classe abstraite)
      protected  abstract String toStringType();
}
```

Affichage des `Mecanisme`s : sous-classes

```
class MecanismeDigital extends  Mecanisme  {
   //..
   @Override
   protected String toStringType()  {
      return "digital";
   }

   @Override
   protected String toStringCadran() {
   // On affiche l'heure (façon de base)...
   // ...et en plus l'heure de réveil.
      return super.toStringCadran() + ", " + toStringReveil();
   }

   protected String toStringReveil() {
         return " réveil " + reveil;
   }
}
```

Affichage des Mecanismes : sous-classes

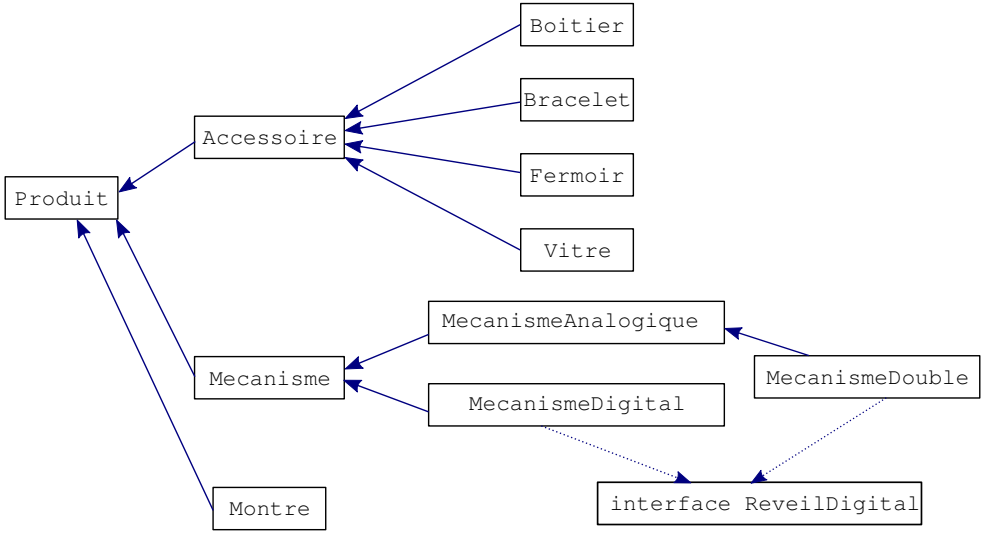
```
class MecanismeDouble extends MecanismeAnalogique {
    //...
   @Override
   protected String toStringCadran() {
      // Par exemple...
      String result = "sur l'écran : ";
      result += super.toStringCadran();
      result += ", sous les aiguilles : ";
      result += toStringReveil();
      return result;
   }
   @Override
   protected String toStringType() {
      return "double";
   }
   // propres aux mécanismes digitaux
   protected String toStringReveil() {
      return " réveil " + reveil;
   }
}
```

Affichage des Mecanismes : sous-classes

Il serait bien :

- d'imposer aux mécanismes ayant (aussi) un comportement digital d'implémenter la méthode `toStringReveil` ;
- d'expliciter le lien entre les mécanismes ayant un comportement digital.

Hiérarchie de classe



Révision de la hiérarchie

```
//=========================================================================
interface ReveilDigital
{
   String toStringReveil();
}
//=========================================================================
class MecanismeDigital extends  Mecanisme implements ReveilDigital {
   //...
   // propres aux mécanismes digitaux
   public String toStringReveil() {
      return " réveil " + reveil;
   }
}
//=========================================================================
class MecanismeDouble extends MecanismeAnalogique implements ReveilDigital {
   //..
   // propres aux mécanismes digitaux
   public String toStringReveil() {
      return " réveil " + reveil;
   }
}
```

Test : un exemple de méthode `main()`

```
// test de l'affichage des mécanismes
MecanismeAnalogique v1 = new MecanismeAnalogique(312.00, 20141212);
MecanismeDigital    v2 = new MecanismeDigital(32.00, "11:45", "7:00");
MecanismeDouble     v3 = new MecanismeDouble(543.00, "8:20", 20140328, "6:30");
System.out.println(v1);
System.out.println(v2);
System.out.println(v3);
// Test des montres
Montre m = new Montre(new MecanismeDouble(468.00, "9:15", 20140401, "7:00"));
m.ajouter(new Bracelet("cuir", 54.0));
m.ajouter(new Fermoir("acier", 12.5));
m.ajouter(new Boitier("acier", 36.60));
m.ajouter(new Vitre("quartz", 44.80));
System.out.println('\n' + "Montre m :");
m.afficher();
```

Le code complet à ce stade (298 lignes) peut être téléchargé sur le site du cours :

`https://d396qusza40orc.cloudfront.net/java-fr/complements/Montres02.java`