Problème :

*Inspiré par : https://class.coursera.org/intropoojava-001*

Une montre est un produit composé d'un mécanisme qui peut être digital, analogique ou mixte et de différents accessoires (un boitier, un bracelet, un fermoir, une vitre).

Les mécanismes et les accessoires sont aussi des produits : on peut les acheter séparément.

Tous les produits ont un prix dont le calcul peut varier à partir d'une valeur de base :

* le prix d'un bracelet est sa valeur de base.
* le prix d'une montre est la somme des prix de de ses composants.

Tous les produits doivent pouvoir être affichés selon leur composition.

Exemples d'affichage :

* un bracelet :

bracelet cuir 54.0

* une montre :

Une montre composée de :

\* Mécanisme double (affichage : sur l'écran : 9:15, date 20140401, sous les aiguilles : réveil 7:00), prix de l'ensemble mécanisme double : 468.0

\* bracelet cuir 54.0

\* fermoir acier 12.5

\* boitier acier 36.6

\* vitre quartz 44.8

==> Prix total : 615.9

Le schéma ci-dessous représente les différentes classes et les liens d'héritage entre elles. (Les liens de composition n'y figurent pas).



BAREME : TOTAL Pts

|  |  |  |  |
| --- | --- | --- | --- |
| Question | points/item | Total question | Total devoir |
| 1 | 1 | 1 pt | 1 |
| 2 | 1 + 1.5 | 2.5 pts | 3.5 |
| 3 | 1 | 1 pt | 4.5 |
| 4 | 0.5+1+0.5+1+1 | 4 pts | 8.5 |
| 5 | 1+1+0.5 | 2.5 pts | 11 |
| 6 | 1 + 1 | 2 pts | 13 |
| 7 | 2 | 2 pts | 15 |
| 8 | 2 + (changer =1 +retirer=2) | 5 pts | 20 |

**Question 1 :** Parmi ces classes dites lesquelles vous proposez de déclarer abstraites? Justifier brièvement vos choix.

|  |
| --- |
| les classes Produit , Accessoire et Mecanisme,  toutes classes qui vont être dérivées selon des spécificités. |

La classe Produit est définie comme suit en première approche.

**Question 2 :** Ecrire le code des accesseurs/mutateurs qui vous paraissent pertinents.

|  |
| --- |
| abstract class Produit implements Cloneable {  // un produit a un prix de base et un nom  private **final** double prix;  private **final** String nom;  public Produit(double uneValeur, String unNom) {  this.prix = uneValeur;  this.nom = unNom;  }  public Produit() {  this.prix = 0.0;  this.nom ="";  }  public Object clone() {  try {  return super.clone();  } catch (CloneNotSupportedException e) {  e.printStackTrace();  }  return null;  }  public double getPrix()  {  return prix;  }  public double getNom()  {  return nom;  }  //Pas de mutateur (qualificateur final pour les attributs |

Justifiez vos choix.

|  |
| --- |
| Accesseurs : accéder au prix avec une méthode qui peut être polymorphe (cas de la montre)  Mutateurs : changer le prix d'un produit (action commerciale) ou le nom (erreur de saisie) mais le qualificateur finale est en contradiction avec cette idée. |

Les classes qui décrivent les différents mécanismes sont les suivantes :

//======================================================================

**abstract class Mecanisme extends Produit {**

//...

}

//======================================================================

**class** MecanismeAnalogique **extends** Mecanisme {

// ...

}

//======================================================================

**interface** ReveilDigital{

String toStringReveil();

}

//======================================================================

**class** MecanismeDigital **extends** Mecanisme **implements** ReveilDigital {

// ...

}

//======================================================================

**class** MecanismeDouble **extends** MecanismeAnalogique **implements** ReveilDigital {

// ...

}

**Nous ne les étudierons pas dans ce devoir.** **Par contre, nous allons compléter maintenant les classes Montre, Accessoire et classes dérivées.**

Dans une première approche, ces classes sont définies comme ci-dessous.**.**

//======================================================================

**class** Montre **extends** Produit {

**private** Mecanisme coeur;

**private** List<Accessoire> accessoires;

**public** Montre(Mecanisme depart)

{

coeur = depart.clone();

accessoires = **new** ArrayList<Accessoire>();

}

}

//======================================================================

**abstract** **class** Accessoire **extends** Produit {

**public** Accessoire(**double** valeurDeBase, String unNom){

**super**(valeurDeBase, unNom);

}

}

//------------------------------------------------------------

**class** Bracelet **extends** Accessoire {

**public** Bracelet(**double** valeurDeBase, String unNom) {

**super**(valeurDeBase, "bracelet " + unNom );  
}

}

//------------------------------------------------------------

**class** Fermoir **extends** Accessoire {

**public** Fermoir( **double** valeurDeBase, String unNom) {

**super**(valeurDeBase, "fermoir " + unNom);  
}

}

//------------------------------------------------------------

**class** Boitier **extends** Accessoire {

**public** Boitier( **double** valeurDeBase, String unNom) {

**super**(valeurDeBase, "boitier " + unNom);

}

}

//------------------------------------------------------------

**class** Vitre **extends** Accessoire {

**public** Vitre( **double** valeurDeBase, String unNom) {

**super**(valeurDeBase,"vitre " + unNom);

}

}

**Question 3 :**

L'instruction Produit p = new Produit(); est-elle correcte?

Justifier votre réponse.

|  |
| --- |
| la classe Produit est abstraite : pas d'instanciation possible |

**Question 4 :**

Quelle méthode donne le prix d'un produit, quel que soit le type réel de ce produit?

|  |
| --- |
| getPrix() de la classe Produit |

Dans l'état actuel, est-ce que cela fonctionne pour TOUS les types de produit? Expliquer brièvement.

|  |
| --- |
| Pas pour les montres dont le prix est la somme des prix de leurs composants. |

Sinon, quelle classe devez-vous compléter?

|  |
| --- |
| la classe Montre. |

Ecrire le code de la fonction que vous devez redéfinir dans la classe concernée.

|  |
| --- |
| @Override  public double getPrix() {  // Au départ, le prix est le prix de base plus celui du Mécanisme  double prixFinal = super.getPrix() + coeur.getPrix();  for (Accessoire acc : accessoires) {  prixFinal += acc.getPrix();  }  return prixFinal;  } |

Quel mécanisme avez-vous mis en œuvre?

|  |
| --- |
| le polymorphisme. |

**Question 5 :**

Pour que tous les produits soient affichables mais chacun avec ses spécificités :

* Dans quelles classes faut-il redéfinir la méthode toString() de la classe Objet pour avoir un affichage polymorphe?

|  |
| --- |
| Les classes Montre et(et Mecanisme) |

* Quelle précaution faut-il prendre pour afficher le **prix réel** d'un produit?

|  |
| --- |
| afficher le prix via la méthode getPrix() qui est polymorphe et est adaptée aux spécificités de l'objet. |

* Ecrire le code de la fonction toString() pour la classe Produit.

|  |
| --- |
| public String toString() {  // Par défaut, un produit affiche son nom et son prix.  return this.nom +" " + getPrix();  } |

**Question 6 :**

On veut pouvoir faire une copie d'une montre.

Quelle(s) classe(s) faut-il compléter?

|  |
| --- |
| les classes Montre (et Mecanisme) pour pouvoir dupliquer tous les éléments.  remarque : les accessoires sont bien clonés (pas d'attribut spécifique devant être cloné). |

Ecrire le code la méthode que vous proposez d'ajouter.

|  |
| --- |
| Au choix ajouter un constructeur (meilleure solution) ou/et la redéfinition de la méthode clone)  // Copie PROFONDE d'une Montre  public Montre(Montre autre)  { super(autre);  coeur = (Mecanisme) autre.coeur.clone();  accessoires = new ArrayList<Accessoire>();  for (Accessoire acc : autre.accessoires) {  accessoires.add((Accessoire) acc.clone());  }  }  @Override  public Montre clone(){  return new Montre(this);  } |

**Question 7 :**

On veut pouvoir ajouter un accessoire sur une montre.

|  |
| --- |
| Par exemple : Montre m = **new** Montre(**new** MecanismeDouble(468.00, "9:15", 20140401, "7:00"));  Accessoire braceletCuir =**new** Bracelet(54.0, "cuir");  m.ajouter(braceletCuir);  m.ajouter(**new** Fermoir(12.5, "acier"));  m.ajouter(**new** Boitier(36.60, "acier"));  m.ajouter(**new** Vitre(44.80, "quartz")); |

Ecrire la méthode ajouter() de la classe Montre.

|  |
| --- |
| public void ajouter(Accessoire accessoire)  {  accessoires.add((Accessoire) accessoire.clone ());  } |

**Question 8 :**

On veut aussi pouvoir changer un accessoire, c'est-à-dire en retirer un et le remplacer par un autre.

|  |
| --- |
| Par exemple pour changer le bracelet de la montre précédente :  Accessoire braceletMode = **new** Bracelet(20.0, "mode");  m.changer(braceletCuir, braceletMode); |

Pour identifier l'accessoire qu'on veut retirer de la liste d'accessoires, deux méthodes doivent être ajoutées à la classe Produit? Lesquelles? Pourquoi?

|  |
| --- |
| public int hashCode() {  public boolean equals(Object obj) pour reconnaitre un accessoire selon ses attributs et non sa référence. |

Ecrire les méthodes retirer( ) et changer( ) de la classe Montre qui utilisent les méthodes ci-dessus.

|  |
| --- |
| public void changer(Accessoire oldAccessoire, Accessoire newAccessoire)  {  this.retirer(oldAccessoire);  this.ajouter(newAccessoire);  }  //avec un iterator  public void retirer(Accessoire accessoire){  Iterator<Accessoire> il = accessoires.listIterator();  while (il.hasNext()){  if(il.next().equals(accessoire)){  il.remove();  }  }  }  // methode bricolée  public void retirer(Accessoire accessoire){  int i=-1;  for(Accessoire acc : accessoires){  if(acc.equals(accessoire)){  i=accessoires.indexOf(acc);  }  }  if(i>=0){  accessoires.remove(i);  }  } |

ANNEXES : La classe *Object*

|  |  |
| --- | --- |
| public boolean equals(Object obj) | Retourne *true* si un objet est égal à un autre. |
| public int hashCode() | Retourne une valeur de *hash-code* pour l’objet courant. |
| public String toString() | Retourne une représentation en chaîne de l’objet : la définition pour Object est getClass().getName() + '@' + Integer.toHexString(hashCode()). |
| public Class getClass() | Retourne la classe de l’objet. |
| protected  Object clone() | Création d’une copie de l’objet.   * Si l’objet n’implémente pas l’interface Cloneable, alors une exception  CloneNotSupportedException est levée. * Sinon la méthode crée une nouvelle instance de la classe et initialise tous ses attributs avec les valeurs des attributs de l’objet : si l’attribut est une référence c’est la référence qui est copiée : il n’y a pas copie en profondeur. |

L’interface List

|  |  |
| --- | --- |
| void add(int ind, E elt) | Insère elt à l’indice ind (opération optionnelle) |
| boolean addAll(int ind,  Collection<? extends E> c2) | Insère tous les éléments de la collection c2 à partir de l’indice ind.(opération optionnelle) |
| get(int ind) | Retourne l’élément à la position ind. Peut lever une exception IndexOutOfBoundsException si ind n'est pas correct. |
| int indexOf(Object o) | Retourne l’indice de la première occurrence de o, ou -1 si la list ne contient pas o. |
| int lastIndexOf(Object o) | Retourne l’indice de la dernière occurrence de o, ou -1 si la list ne contient pas o. |
| ListIterator<E> listIterator() | Retourne un ListIterator sur la liste |
| ListIterator<E> listIterator(int ind) | Retourne un ListIterator sur la liste, démarrant à l’indice ind. Peut lever une exception IndexOutOfBoundsException si ind n'est pas correct. |
| E remove(int ind) | Enlève l'élément à l'indice ind, et retourne cet élément. Peut lever une exception IndexOutOfBoundsException si ind n'est pas correct. |

L’interface Iterator

|  |  |
| --- | --- |
| boolean hasNext() | Retourne *true* si l’itérateur n’est pas arrivé en fin de l’ensemble et *false* sinon. |
| E next() | Permet d’avancer l’itérateur, et retourne l’élément qui était obtenu avant ce déplacement.  Lève une exception *NoSuchElementException* si l'itérateur n'a pas d'objet qui le suit. |
| void remove() | Supprime l’objet qui vient d’être obtenu par *next*. Cette méthode est facultative. |

L'interface ListIterator dérive de l'interface Iterator

|  |  |
| --- | --- |
| int  nextIndex() | Retourne l'indice de l'élément qui serait retourné par un appel de next(). |
| void set(E o) | Remplace le dernier élément  retourné par next(), ou par previous(), par l'objet o. |
| void add(E o) | Remplace le dernier élément  retourné par next(), ou par previous(), par l'objet o. |