Université de Paris **Programmation Orientée Objet**
L2 **Année 2020-2021**

TD - Séance n°6 - Correction

Interfaces et classes abstraites

Exercice 1 *Interfaces vs classes abstraites*

1. Peut-on instancier une interface ? Une classe abstraite ?

   Correction : Non pour les deux.

2. Peut-on y mettre un constructeur ? Un constructeur avec un corps ?

   Correction : Non aux deux pour les interfaces. Oui aux deux pour les classes abstraites, il pourra être appelé avec super(...) comme d'habitude.

3. Est-ce que le code suivant est valide : `A a = new B();`
   — si `A` est une classe abstraite, étendue par la classe `B` ?
   — si `A` est une interface, implémentée par une classe `B` ?
   Correction : Oui pour les deux (si la classe `B` n'est pas abstraite).

4. Une interface/classe abstraite peut-elle contenir des attributs ? Avec quels modificateurs ? Doivent-ils être initialisés ?

   Correction : Les interfaces ne peuvent avoir que des attributs public static final, c'est à dire des constantes ; les modificateurs n'ont pas besoin d'être déclarés. Ces attributs doivent être initialisés directement dans le corps de l'interface. Une classe abstraite peut avoir tout type de champ avec tout modificateur.

5. Une interface/classe abstraite peut-elle contenir des méthodes abstraites ? non-abstraites ? statique et abstraite ?

   Correction : Java < 8 : une interface ne peut contenir que des méthodes publiques abstraites et non-statiques ; les modificateurs public abstract n'ont pas besoin d'être déclarés.

   Java ≥ 8 : Dans une interface : on peut avoir des méthodes abstraites mais également non-abstraites (modificateurs default). On peut aussi avoir des méthodes statiques, mais ils doivent avoir une une definition (donc pas statiques et abstraites en meme temps).

   Dans une classe abstraite on peut avoir la meme chose : méthodes abstraites, méthodes avec définition, méthodes statiques avec definition, mais ici aussi pas de méthodes statiques et abstraites en meme temps.

   Remarquer tout de même une difference : les méthodes statiques d'interface ne sont pas héritées par les classes qui implémentent l'interface (dans ces classes il faut s'y référer comme `I.f()` si `I` est le nom de l'interface). En revanche les méthodes statiques d'une classe abstraite sont héritées par les classes qui étendent la classe abstraite (elle peuvent les invoquer comme `f()`).

6. Une interface peut-elle hériter d'une autre interface ? d'une classe abstraite ?

   Correction : Oui. Non.

7. Une classe abstraite peut-elle hériter d'une autre classe abstraite ? d'une interface ?

   Correction : Oui. Non, elle peut en revanche l'implémenter.

Exercice 2 *Instruments de musique*

Dans cet exercice, on va tenter de modéliser une structure de classes pour des instruments de musique. Ils existent plusieurs façons de classifier les instruments.

Une première consiste à différencier selon le procédé qui permet de produire le son. Certains sont dits mécaniques, dans le sens où le son provient d'une vibration mécanique d'une pièce ou d'une masse d'air (tous les instruments traditionnels, mais aussi la guitare électrique ou les pianos électriques type orgue Hammond, Rhodes, etc...) et d'autres, dits électroniques, dont le son est produit par un générateur électronique oscillant (les synthétiseurs).

Une seconde consiste à différencier selon la manière dont le son est amplifié. L'amplification peut à nouveau être mécanique, par une caisse de résonnance, ou bien électrique à l'aide d'un microphone. Une guitare électrique, par exemple, n'est pas un synthétiseur, le son provient bien de la vibration d'une corde, mais il est amplifié a l'aide de microphones qui transforment cette vibration en signal électrique.

Les productions mécaniques de son sont séparés en trois grandes familles,

— Les Cordes, qui peuvent être pincées (guitare), frappées (piano) ou frottées (violon).
— Les Vents divisés en Bois (le son vient de la vibration de l'air sur une pièce mécanique), et Cuivres (le son vient de la vibration des lèvres à l'embouchure).
— Les Percussions (on frappe une peau ou une pièce de bois ou de métal).

1. Fournir une architecture de classes et/ou interfaces pour représenter ceci. Toutes les classes descendront du type Instrument. Tous les instruments ont en commun de pouvoir être joués. On munira donc Instrument d'une méthode abstraite **abstract public void** play(), qui devra être implémentée par chaque instrument. Ils ont aussi en commun d'avoir un nom : Instrument disposera donc d'un champ String name. Les instruments à cordes disposent d'un champ indiquant le nombre de cordes. Les instruments à amplification électrique disposent d'une méthode retournant le type de prise. Écrire les classes et/ou interfaces permettant de modéliser cette hiérarchie. Donner la déclaration (l'en-tête) de la classe Orgue, de la classe Saxophone, de la classe GuitareElectrique et de la classe PianoSynthetiseur.
2. On aimerait bien aussi pouvoir dire qu'un piano, un orgue, un piano synthétiseur, appartiennent tous à la famille des claviers. Comment faire cela ?

Exercice 3 *Java for Rocket Scientists*

On a les classes suivantes :

```
public interface Spationef {
    public default int equipageMax() {
       return 0;
    }
    public String typeSpationef();
}
```

```
public interface Propulsion {
    public String typePropulsion();
}
```

```
public abstract class NavetteSpatiale
                implements Spationef, Propulsion {
    public String typeSpationef() {
       return "navette spatiale";
    }

    public abstract String typePropulsion();
}
```

```
public class NavetteSpatialeAmericaine extends NavetteSpatiale {
    public int equipageMax() {
      return 8;
    }
    public String typeSpationef() {
      return "navette spatiale americaine";
    }
    public String typePropulsion() {
      return "propulsion propre";
    }
}
```

```
public class NavetteDiscovery extends NavetteSpatialeAmericaine {
    public int equipageMax() {
      return 7;
    }
    public String typeSpationef() {
      return "navette Discovery";
    }
}
```

```
public class NavetteSpatialeRusse extends NavetteSpatiale {
    public String typeSpationef() {
       return "navette spatiale russe";
```

```
    }

    public int equipageMax() {
        return 0;
    }

    public String typePropulsion() {
        return "par fusee";
    }
}
```

```
public class SatelliteMeteo implements Spationef {
    public String typeSpationef() {
        return "satellite meteo";
    }
}
```

Dans le code suivant, dites quelles sont les lignes qui compilent, qui s'exécutent. En cas d'erreur, expliquez pourquoi. Si l'on commente les lignes qui posent problème, qu'affiche l'exécution du code ?

```
Spationef u1 = new NavetteSpatiale();
NavetteSpatiale u2 = new NavetteDiscovery();
Propulsion u3 = new NavetteDiscovery();
NavetteSpatialeRusse u4 = new NavetteDiscovery();
Spationef u5 = new NavetteSpatialeRusse();
SatelliteMeteo u6 = new SatelliteMeteo();
```

```
System.out.println(u2.equipageMax());
System.out.println(u3.equipageMax());
System.out.println(u5.equipageMax());
System.out.println(u6.equipageMax());
System.out.println(u2.typeSpationef());
System.out.println(
  ((NavetteSpatialeAmericaine)u2).typeSpationef());
System.out.println(u3.typePropulsion());
System.out.println(u6.typePropulsion());
```

```
NavetteSpatiale u7 = u5;
SatelliteMeteo u8 = (SatelliteMeteo) u5;
NavetteSpatialeRusse u9 = (NavetteSpatialeRusse) u5;
Spationef u10 = u2;
Spationef u11 = u3;
Spationef u12 = (NavetteSpatiale) u3;
```

Correction :

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

```
    //Spationef u1 = new NavetteSpatiale();
4   //la ligne precedente echoue a la compilation :
    // NavetteSpatiale is abstract; cannot be instantiated

    NavetteSpatiale u2 = new NavetteDiscovery();
8   Propulsion u3 = new NavetteDiscovery();

10  //NavetteSpatialeRusse u4 = new NavetteDiscovery();
    //la ligne precedente echoue a la compilation:
12  // incompatible types:
    // NavetteDiscovery cannot be converted to
14  // NavetteSpatialeRusse

16  Spationef u5 = new NavetteSpatialeRusse();
    SatelliteMeteo u6 = new SatelliteMeteo();

    System.out.println(u2.equipageMax()); // 7

    //System.out.println(u3.equipageMax());
22  //la ligne precedente echoue a la compilation:
    // cannot find symbol
24  // symbol: method equipageMax()
    // location: variable u3 of type Propulsion

    System.out.println(u5.equipageMax()); // 0
28  System.out.println(u6.equipageMax()); // 0
    System.out.println(u2.typeSpationef()); // navetteDiscovery
30  System.out.println(((NavetteSpatialeAmericaine)u2).typeSpationef());
        // navetteDiscovery
32  System.out.println(u3.typePropulsion()); // propulsion propre

34  //System.out.println(u6.typePropulsion());
    //la ligne precedente echoue a la compilation:
36  // cannot find symbol
    // symbol: method typePropulsion()
38  // location: variable u6 of type SatelliteMeteo

40  //NavetteSpatiale u7 = u5;
    //la ligne precedente echoue a la compilation:
42  // incompatible types: Spationef cannot be converted to
    // NavetteSpatiale

    //SatelliteMeteo u8 = (SatelliteMeteo)u5;
46  //la ligne precedente echoue a l'execution:
    // class NavetteSpatialeRusse cannot be cast to class
48  // SatelliteMeteo

50  NavetteSpatialeRusse u8 = (NavetteSpatialeRusse) u5;
    Spationef u10 = u2;
```

```
        //Spationef u11 = u3;
54      //la ligne precedente echoue a la compilation:
        // incompatible types: Propulsion cannot be converted to
56      // Spationef

58      Spationef u12 = (NavetteSpatiale) u3;
    }
60  }
```