Exercice 1 (8.14 dans le livre)

Dans le projet DOME du chapitre 8, que doit-on changer dans la classe **`Database`** quand une autre sous-classe de **`Item`** (par exemple **`VideoGame`**) est ajoutée ? Pourquoi ?

Exercice 2 (8.12, 8.13)

Supposons que nous ayons quatre classes : **`Person`**, **`Teacher`**, **`Student`** et **`PhDStudent`**.

`Teacher` et **`Student`** sont toutes deux des sous-classes de **`Person`**.

`PhDStudent` est une sous-classe de **`Student`**.

a) Parmi les affectations suivantes, lesquelles sont légales, et pourquoi ?

```
1. Person p1 = new Student();
2. Person p2 = new PhDStudent();
3. PhDStudent phd1 = new Student();
4. Teacher t1 = new Person();
5. Student s1 = new PhDStudent();
```

b) Supposons que vous ayez les déclarations et affectations valides suivantes :

```
Person p1 = new Person();
Person p2 = new Person();
PhDStudent phd1 = new PhDStudent();
Teacher t1 = new Teacher();
Student s1 = new Student();
```

Quelles sont ci-dessous les affectations légales ? Pourquoi ?

```
1. s1 = p1;
2. s1 = p2;
3. p1 = s1;
4. t1 = s1;
5. s1 = phd1;
6. phd1 = s1;
```

Vérifiez vos réponses à la question précédente en créant les classes mentionnées dans cet exercice et en les testant dans BlueJ.

Exercice 3 (8.18)

Étudiez le code ci-après. Vous avez quatre classes (**`O`**, **`X`**, **`T`** et **`M`**) et une variable pour chacune de celles-ci :

```
O o;
X x;
T t;
M m;
```

Les affectations suivantes sont toutes autorisées (on suppose qu'elles se compilent toutes) :

```
m = t;
m = x;
o = t;
```

Aucune des affectations suivantes n'est autorisée (elles génèrent des erreurs du compilateur) :

```
o = m;
o = x;
x = o;
```

Que pouvez-vous dire sur les relations de ces classes ? Dessinez un diagramme de classes.

Exercice 4 (9.12, 9.13)

Supposons que vous trouviez les lignes de code suivantes :

```
Device dev = new Printer();
dev.getName();
```

`Printer` est une sous-classe de **`Device`**.

1) Lesquelles de ces classes doivent avoir une définition de la méthode **`getName`** pour que ce code se compile ?

2) Si les deux classes ont une implémentation de **`getName`**, laquelle sera exécutée ?

Exercice 5 (9.14, 9.15, 9.16)

Supposons que vous écriviez une classe **`Student`** qui ne possède pas de superclasse déclarée. Vous n'écrivez pas de méthode **`toString`**.

1) Envisagez les lignes de code suivantes :

```
Student st = new Student();
String s = st.toString();
```

Ces lignes se compileront-elles ? Si oui, que se passera-t-il exactement lorsque vous tenterez de les exécuter ?

2) Les lignes suivantes se compileront-elles ? Pourquoi ?

```
Student st = new Student();
System.out.println(st);
```

3) Supposons que **`Student`** redéfinisse **`toString`** de sorte qu'elle renvoie le nom de l'étudiant. Vous avez maintenant une liste d'étudiants. Le code suivant se compilera-t-il ? Si non pourquoi ? Si oui qu'affichera-t-il ? Expliquez en détails ce qui se passe.

```
for(Object st : myList){
    System.out.println(st);
}
```

Exercice 6 (9.17)

Écrivez quelques lignes de code qui créent une situation dans laquelle une variable **`x`** possède le type statique **`T`** et le type dynamique **`D`**.

Exercice 7 (10.59)

Examinez le code qui suit. Vous avez cinq types (classes ou interfaces) : **`U`**, **`G`**, **`B`**, **`Z`** et **`X`**, et une variable de chacun de ces types.

```
U u;
G g;
B b;
Z z;
X x;
```

Les affectations suivantes sont toutes autorisées (on suppose qu'elles se compilent toutes) :

```
u = z;
x = b;
g = u;
x = u;
```

Aucune des affectations suivantes n'est autorisée (elles génèrent des erreurs du compilateur) :

```
u = b;
x = g;
b = u;
z = u;
g = x;
```

Que pouvez-vous dire des types et de leurs relations ? (Quelles relations y a-t-il entre eux ?)

Exercice 8 (10.61)

On trouve parfois des paires classes/interfaces dans la bibliothèque Java standard qui ont exactement les mêmes méthodes. Souvent, le nom de l'interface se termine par **`Listener`** et le nom de la classe par **`Adapter`**.

Exemple : **`PrintJobListener`** et **`PrintJobAdapter`**.

L'interface a quelques méthodes et la classe **`Adapter`** définit ces mêmes méthodes avec un corps vide. Pour quelles raisons a-t-on les deux ?