Annales POO

Robin VAN DE MERGHEL

Table des matières

Note : Il se peut que je ne soit pas sûr de certaines réponses, n'hésitez pas à me contacter si vous pensez que je me suis trompé. Je noterai les questions où je ne suis pas sûr d'une étoile.

1 Annale 2020-2021

1.1 Exercice 1

1.1.1 Question 1

En programmation orientée objet, qu'est-ce qu'un objet ?

Un objet est une entité qui possède des attributs et des méthodes. Il peut être instancié et possède un état. Il peut être modifié par des méthodes.

On peut le voir comme une sorte de "boîte" qui contient des informations et des fonctions.

1.1.2 Question 2

Expliquez le but et le fonctionnement des exceptions.

Les exceptions sont des erreurs qui peuvent survenir lors de l'exécution d'un programme comme une division par zéro. Elles sont gérées par des blocs `try/catch` qui permettent de gérer les erreurs (pouvoir choisir un comportement en cas d'erreur).

1.1.3 Question 3

Pourquoi faut-il toujours redéfinir hashCode (ou équivalent) quand on redéfini equals (ou équivalent) ?

Si on redéfinit `equals`, on doit redéfinir `hashCode` pour que les deux méthodes soient cohérentes. En effet, si deux objets sont égaux (sans avoir la même référence), ils doivent avoir le même `hashCode`.

Il en est de même si on redéfinit `compareTo`.

1.1.4 Question 4

Quelle est la différence entre la redéfinition de méthodes et la surcharge?

La redéfinition de méthodes est une spécialisation d'une méthode d'une classe parente, par exemple un `Animal` qui redéfinit la méthode `manger` pour qu'elle soit plus spécifique à un `Chat`.

La surcharge est une méthode qui a le même nom mais qui a des paramètres différents. Par exemple, on peut avoir une méthode `manger` qui prend en paramètre un `Animal` et une autre qui prend en paramètre un `Nourriture`.

1.1.5 Question 5

Quel est l'intérêt d'une machine virtuelle (dans le contexte de Java)?

La machine virtuelle permet d'exécuter du code Java sur n'importe quel système d'exploitation. Par exemple, un programme Java écrit sur Windows peut être exécuté sur Linux, voire même sur un téléphone Android.

Elle permet aussi une portabilité du code Java, car le code compilé est le même peu importe l'OS.

1.1.6 Question 6

En Java, quelle est la différence entre l'opérateur == et la méthode equals?

L'opérateur `==` compare les références des objets (leur emplacement en mémoire). La méthode `equals` compare les valeurs des objets (leur contenu) et peut-être redéfinie.

Donc deux `Etudiant` peuvent avoir la même valeur (même nom, prénom, etc.) mais ne pas être égaux (car ils ont des références différentes $\leftrightarrow$ ils ne sont pas de la même famille).

1.1.7 Question 7*

Note : je ne suis pas sûr.

Pourquoi est-il préférable de ne dépendre que des interfaces?

Cela permet de ne pas dépendre d'une implémentation particulière. Par exemple, on peut avoir une classe `Etudiant` qui dépend de l'interface `Personne` et qui peut donc être utilisée avec n'importe quelle implémentation de `Personne` (une classe `Etudiant` ou une classe `Professeur` par exemple).

1.1.8 Question 8

Donner les critères permettant de garantir la substituabilité d'un sous-type.

Un sous-type doit pouvoir être utilisé à la place de son type parent. Pour cela, il doit implémenter toutes les méthodes de son type parent et ne pas en ajouter.

De plus, il doit pouvoir être utilisé dans un contexte où le type parent est attendu. Exemple : Un triangle est une forme, si une forme est dessinable, alors un triangle est dessinable.

1.1.9 Question 9

Que signifie le mot-clé Java `final` utilisé sur : un attribut, une méthode, une classe.

1.1.9.1 Attribut Un attribut `final` ne peut pas être modifié après sa création. Il doit être initialisé dans le constructeur. C'est une constante.

1.1.9.2 Méthode

Une méthode `final` ne peut pas être redéfinie dans une classe fille.

1.1.9.3 Classe

Une classe `final` ne peut pas être étendue (héritée). On peut dire que c'est une classe "scellée" ou "élémentaire".

1.1.10 Question 10*

Note : peut-être mieux formuler ?

Qu'est-ce que l'héritage? Quelles alternatives peut-on utiliser?

L'héritage est une relation entre deux classes où une classe fille hérite des attributs et méthodes de sa classe parente. Cela permet de réutiliser du code et d'organiser le code en classes plus petites.

On peut utiliser la composition à la place de l'héritage. Cela consiste à créer une classe qui contient une instance d'une autre classe. Par exemple, on peut avoir une classe `Etudiant` qui contient une instance de la classe `Personne`.

1.1.11 Question 11

Expliquez ce que sont les classes génériques.

Les classes génériques sont des classes qui peuvent prendre en paramètre un type. Par exemple, on peut avoir une classe `Liste` qui peut prendre en paramètre un type `T` et qui contient une liste de `T`.

On peut ainsi faire une `Liste` de `String`, de `Integer`, de `Etudiant`, etc. sans avoir à créer une classe pour chaque type.

1.1.12 Question 12

Quels sont les intérêts d'utiliser des accesseurs?

Les accesseurs permettent de contrôler l'accès aux attributs d'une classe. Par exemple, on ne veut pas donner l'accès à la référence d'un étudiant, donc on peut créer un accesseur qui renvoie juste une information (le nom par exemple).

1.2 Exercice 2

Soient les deux classes Java suivante:

```
class A {
    private int a;

    A (int v) {
        a = v;
    }

    public int foo(A other) {
        return a + other.a;
    }

}

class B extends A {
    private int b;

    B (int v) {
        super(v);
```

```
    }

    @Override
    public int foo(B other) {
        return 42;
    }
}
```

Ce programme affiche une erreur à la compilation. Pourquoi?

La méthode `foo` de la classe `B` écrase la méthode `foo` de la classe `A`. Or les types passé en paramètre sont différents. La méthode `foo` de la classe `A` prend en paramètre un objet de type `A`, alors que la méthode `foo` de la classe `B` prend en paramètre un objet de type `B`.

1.3 Exercice 3

Soit le diagramme de classe conceptuel suivant :

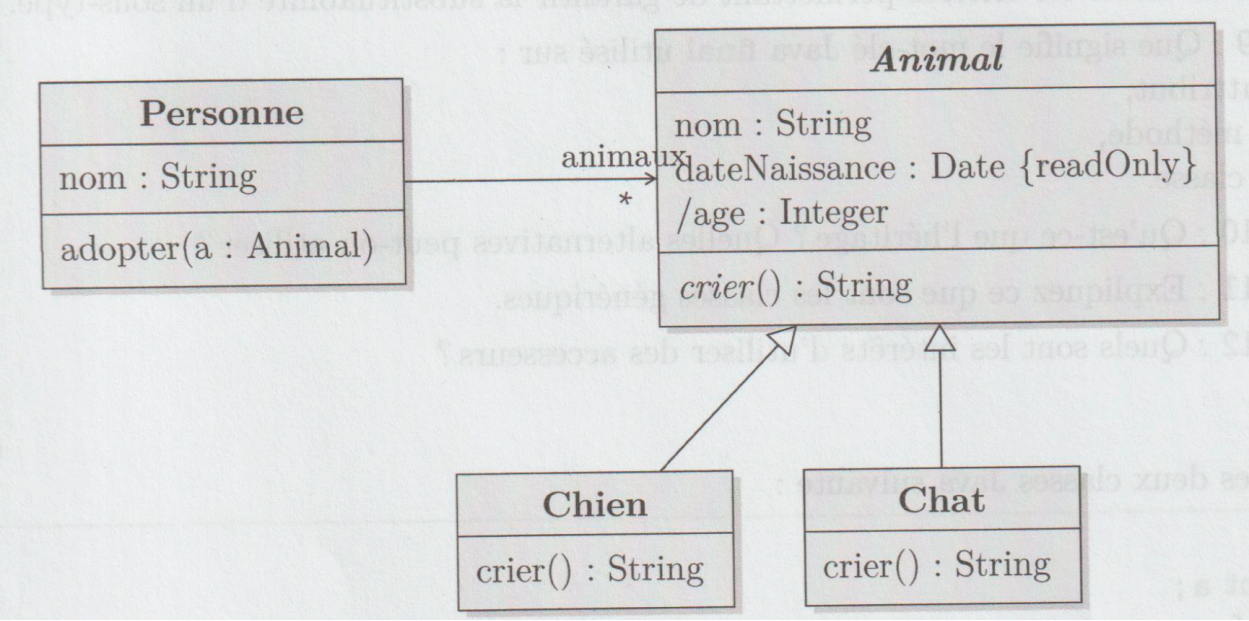

Figure 1: Diagramme

Donnez le code Java correspondant, en respectant les conventions du langage (attention, le diagramme n'est pas au niveau d'abstraction de l'implémentation). Vous pourrez utiliser la class `java.time.LocalDate` pour représenter la date. La méthode statique now de cette classe retourne la date courrante. Le détail de son utilisation n'est cependant pas important ici.

```
import java.time.LocalDate;

class Personne {
    private String nom;
    private Animal animal;

    public Personne(String nom) {
        this.nom = nom;
    }
```

```
    public void adopter(Animal animal) {
        this.animal = animal;
    }
}

abstract class Animal {
    private String nom;
    private LocalDate dateNaissance;

    public Animal(String nom, LocalDate dateNaissance) {
        this.nom = nom;
        this.dateNaissance = dateNaissance;
    }

    public String getNom() {
        return nom;
    }

    public LocalDate getDateNaissance() {
        return dateNaissance;
    }

    public int getAge() {
        return LocalDate.now().getYear() - dateNaissance.getYear();
    }

    public abstract String crier();
}

class Chien extends Animal {
    public Chien(String nom, LocalDate dateNaissance) {
        super(nom, dateNaissance);
    }

    @Override
    public String crier() {
        return "Wouaf";
    }
}

class Chat extends Animal {
    public Chat(String nom, LocalDate dateNaissance) {
        super(nom, dateNaissance);
    }

    @Override
    public String crier() {
        return "Miaou";
    }
}


class Main {
```

```
    public static void main(String[] args) {
        Personne personne = new Personne("Jean");
        Animal chien = new Chien("Rex", LocalDate.of(2010, 1, 1));
        Animal chat = new Chat("Felix", LocalDate.of(2015, 1, 1));

        personne.adopter(chien);
        personne.adopter(chat);

        System.out.println(personne.animal.crier());
    }
}
```