TP5 JAVA – Programmation Orientée Objets

Rafael COLARES

Maitre de Conférences ISIMA

Bureau D104 – email:rafael.colares_borges@uca.fr

Leçons apprises du TP4

- Classes abstraites
  - Modélisation des concepts
    - Instanciation interdite

```
Animal a = new Animal();
```

  - Peut avoir des attributs et méthodes concrets
    - Tout **`Animal`** possède un nom
    - Tout **`Animal`** est capable de fournir son nom

Animal.java

```
public abstract class Animal {
        private String name;
        public String getName() {
                return this.name;
        }
}
```

Leçons apprises du TP4

- Classes abstraites
  - Modélisation des concepts
    - Instanciation interdite

```
Animal a = new Animal();
```

  - Peut avoir des attributs et méthodes concrets

- Méthodes abstraites
  - Impose que la classe soit abstraite
  - Déclaration sans implémentation
  - A quoi ça sert?

Animal.java

```
public abstract class Animal {
        private String name;
        public String getName() {
                return this.name;
        }
        public abstract void makeNoise();
}
```

Dog.java

```
public class Dog extends Animal{
        @Override
        public void makeNoise() {
                System.out.println(« Woof woof »);
        }
}
```

Leçons apprises du TP4

- Classes abstraites
  - Modélisation des concepts
    - Instanciation interdite

```
Animal a = new Animal();
```

  - Peut avoir des attributs et méthodes concrets
- Méthodes abstraites
  - Impose que la classe soit abstraite
  - Déclaration sans implémentation

Animal.java

```
public abstract class Animal {
        private String name;
        public String getName() {
                return this.name;
        }
        public abstract void makeNoise();
}
```

Dog.java

```
public class Dog extends Animal{
        @Override
        public void makeNoise() {
                System.out.println(« Woof woof »);
        }
}
```

Cat.java

```
public class Cat extends Animal{

}
```

Erreur: Classe Cat doit implémenter la méthode makeNoise()

Impose toute sous-classe concrète de fournir une implémentation chaque méthode abstraite

Leçons apprises du TP4

- Interfaces
  - Modélisation d'un contrat de comportement
  - Par défaut:
    - Tout attribut est **`public`**, **`static`** et **`final`**: CONSTANTES
    - Toute méthode est **`public`** et **`abstract`**

Animal.java

```
public abstract class Animal {
        private int health;
        private String name;
        public Animal(int health, String name) {
                this.health = health;
                this.name = name;
        }
        public Animal(String name) {
                this(0, name);
        }
        public String getName() {
                return this.name;
        }
        public boolean isDead(){
                return health == 0;
        }
}
```

Carnivore.java

```
public interface Carnivore {
        boolean IS_PREDATOR = true;
        Animal hunt();
}
```

Animal.java

```
public abstract class Animal {
        private int health;
        private String name;
        public Animal(int health, String name) {
                this.health = health;
                this.name = name;
        }
        public Animal(String name) {
                this(0, name);
        }
        public String getName() {
                return this.name;
        }
        public boolean isDead(){
                return health == 0;
        }
}
```

Cat.java

```
public class Cat extends Animal implements Carnivore{
        public Cat(int health, String name) {
                super(health, name);
        }
        @Override
        public Animal hunt() {
                return new Souris(0, "Jerry");
        }
}
```

Souris.java

```
public class Souris extends Animal{
        public Souris(int health, String name) {
                super(health, name);
        }
}
```

Carnivore.java

```
public interface Carnivore {
        boolean IS_PREDATOR = true;
        Animal hunt();
}
```

App.java

```
public class App{
        public static void main(String[] args) {
                Cat tom = new Cat(100, "Tom");
                Souris deadJerry = tom.hunt();
        }
}
```

Toute classe respectant le contrat, implémente l'interface

➔ Doit fournir une implémentation pour chaque méthode abstraite

Animal.java

```java
public abstract class Animal {
        private int health;
        private String name;
        public Animal(int health, String name) {
                this.health = health;
                this.name = name;
        }
        public Animal(String name) {
                this(0, name);
        }
        public String getName() {
                return this.name;
        }
        public boolean isDead(){
                return health == 0;
        }
}
```

Cat.java

```java
public class Cat extends Animal implements Carnivore{
        public Cat(int health, String name) {
                super(health, name);
        }
        @Override
        public Animal hunt() {
                return new Souris(0, "Jerry");
        }
}
```

Souris.java

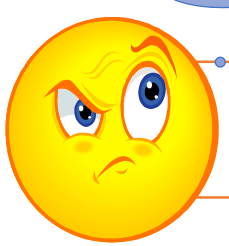
```java
public class Souris extends Animal{
        public Souris(int health, String name) {
                super(health, name);
        }
}
```

App.java

```java
public class App{
        public static void main(String[] args) {
                Cat tom = new Cat(100, "Tom");
                Souris deadJerry = tom.hunt();
        }
}
```

Carnivore.java

```java
public interface Carnivore {
        boolean IS_PREDATOR = true;
        Animal hunt();
}
```



Interfaces vs.
Classes Abstraites ?

Toute classe respectant le contrat, implémente l'interface
➔ Doit fournir une implémentation pour chaque méthode abstraite

Leçons apprises du TP4

- Interfaces vs. Classes Abstraites
  - PAS D'HÉRITAGE MULTIPLE EN JAVA !!
  - Pas de limite pour l'implémentation d'interfaces

Cat.java

```
public class Cat extends Animal implements Carnivore, OtherInterface, AnotherInterface {
        ...
}
```

  - Classe abstraite peut avoir des attributs d'instance
    - Chaque animal a son propre **`name`**, son propre **`health`**...
  - Interfaces peuvent avoir uniquement des constantes (**`static final`**)
    - Commun à tous les objets qui l'implémentent: tous les carnivores sont prédateurs (**`IS_PREDATOR = true`**)

TP 5

- Contenu:
  - ✓ Exceptions

- Diapos disponibles sur: https://github.com/rafael-colares/teaching/

- Aller voir *https://perso.isima.fr/loic/java/tp_05.php*