UNIVERSITE LIBRE DE BRUXELLES
Faculté des Sciences Appliquées

Juin 2017

INFO-H-2001 – Programmation Orientée-Objet – Examen (3h30)
Hugues Bersini

Partie pratique (10 Points)

Question 11 : Syntaxe et Orienté-Objet (3 Point(s))

Corrigez, à même le code, le programme fourni de manière à ce qu'il puisse s'exécuter sans erreur. Indiquez les erreurs qui se produiraient que ce soit à la compilation ou à l'exécution et/ou les éléments manquants. Expliquez brièvement chaque erreur.

```
public class Main {
    private static Vehicule favoriteVehicule;
    private static Plane myPlane;
    private static Flying myVehicule;

    public static void main(String[] args) {
    Vehicule vehicule = new Car(15, "Rouge");    // Car(int, int)
        myPlane = new Plane();
        myVehicule = new Plane();
        favoriteVehicule = new Flying();         // implémentation d'interface

        myPlane.accelerate(15);
        myVehicule.accelerate(10);               // type Flying:
                                                 // pas de méthode accelerate
        vehicule.accelerate(5);

        favoriteVehicule = vehicule;
        ((Plane) favoriteVehicule).move();       // casting de Car en Plane
        ((Plane) myVehicule).move();
    }
}
```

```
public class Vehicule {
    private final int speed;                      // protected int
    private final int position = 0;               // private int

    public Vehicule(int speed){
        this.speed = speed;
    }

    public void move(){
        position += speed;
    }

    public abstract void accelerate(int a);       // on ne peut pas avoir de
                                                  // méthode abstraite dans une
                                                  // classe non abstraite
                                                  // -> rendre la classe abstraite
}
```

```java
public class Car extends Vehicule {
    private int color;

    public Car(int speed, int color) {
    this.speed = speed;                              // manque le super(int)
        this.color = color;
    }

    public void accelerate(int a){
        speed += a;
    }
}
```

```java
public class Plane extends Vehicule implements Flying{
    private boolean flying = false;

    public Plane() {
    this.speed = 0;                                  // manque le super(int)
    }

    public void accelerate(int a){
        int newSpeed = speed + a;
        if(flying && newSpeed < 300){
            return false;                            // retourn d'un booléen
                                                     // dans une méthode void
        }
        speed = newSpeed;
    }

    public void takeoff() {
        flying = true;
    }

    public void land() {
        flying = false;
    }
                                                     // il manque isFlying()
}
```

```java
public interface Flying {
    public void takeoff();
    public void land();
    private boolean isFlying();                  // méthode ne peut pas
                                                 // être privée dans une
                                                 // interface.
}
```

Question 12 : Ecriture de code et diagramme UML (3.5 Point(s))

Vous êtes un berger et possédez un troupeau composé de moutons, chèvres et vaches. Réalisez dans une syntaxe approximative (mais respectant les principes "Orienté-Objet") un code Java contenant les classes suivantes : une classe "Troupeau" avec ses attributs "animeaux"et "nomDuTroupeau" ; une classe "Mouton" avec ses attributs "productionDeLaine" et "productionDeFromage" ainsi qu'une classe "Chevre" et une classe "Vache" possédant chacune un attribut "productionDeFromage". Implémentez aussi les méthodes suivantes :

1. Une méthode "productionLaine" qui devra permettre de calculer la production totale de laine du troupeau.
2. Une methode "productionFromage" qui devra permettre de calculer la production totale de fromage du troupeau.

Vous pouvez ajouter les classes, attributs et méthodes que vous jugez nécessaires. Votre proposition devra être brièvement justifiée. **Enfin, réalisez un diagramme de classes modélisant votre programme.**

```java
public class Troupeau {
    private ArrayList<Animal> animaux;
    private String name;

    public Troupeau(ArrayList<Animal> animaux, String name) {
        this.animaux = animaux;
        this.name = name;
    }

    public int productionLaine() {
        int total = 0;
        for (Animal animal: animaux) {
            if (animal instanceof ProducteurDeLaine) {
                total += ((ProducteurDeLaine) animal).getProductionLaine();
            }
        }
        return total;
    }

    public int productionFromage() {
        int total = 0;
        for (Animal animal: animaux) {
            if (animal instanceof ProducteurDeFromage) {
                total += ((ProducteurDeFromage) animal).getProductionFromage();
            }
        }
        return total;
    }
}
```

```java
public class Animal {}
```

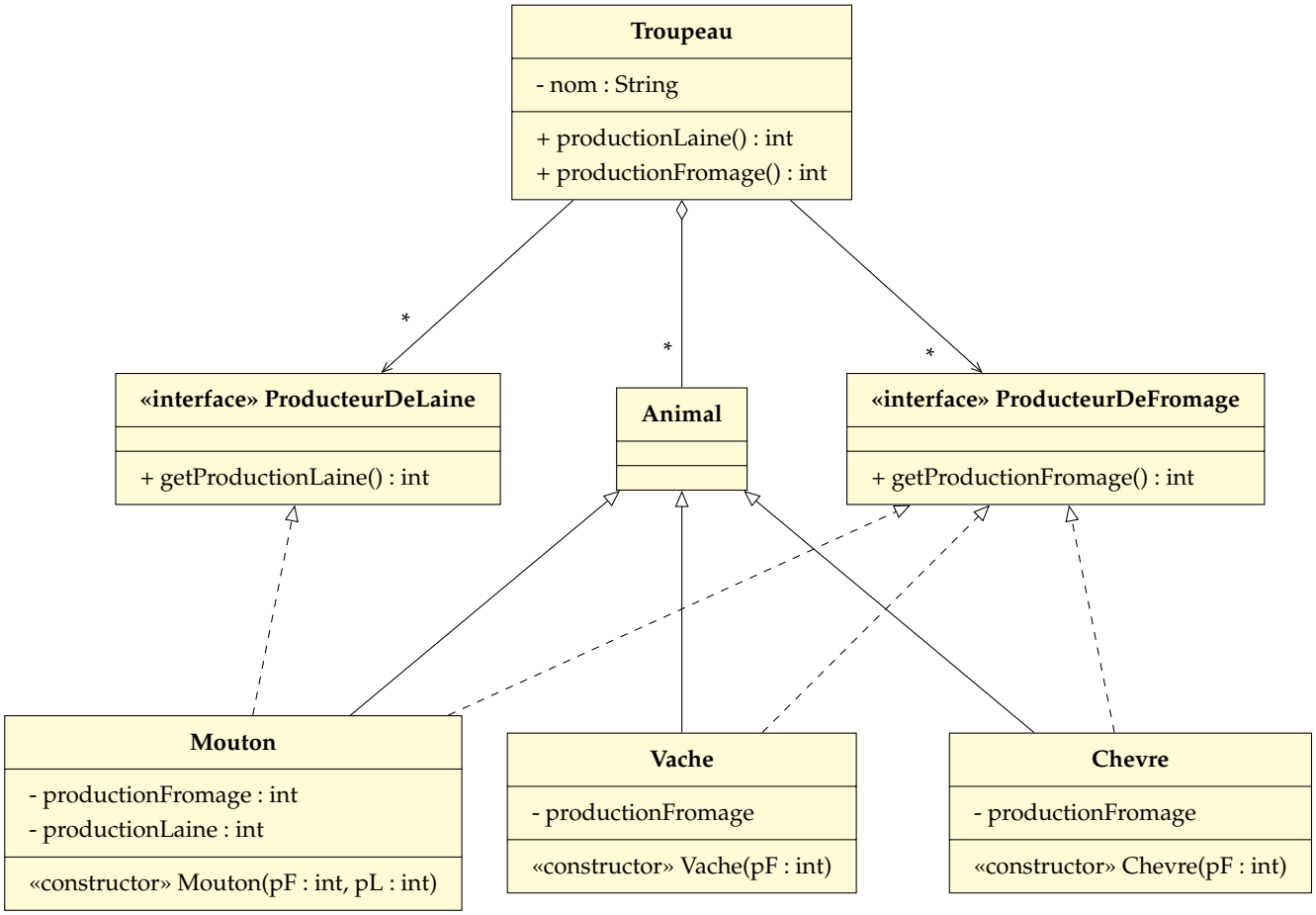
```java
public interface ProducteurDeFromage {
        public int getProductionFromage();
}
```

```java
public interface ProducteurDeLaine {
        public int getProductionLaine();

}
```

```java
public class Mouton extends Animal implements ProducteurDeLaine, ProducteurDeFromage {
        private int productionFromage;
        private int productionLaine;

        public Mouton(int pF, int pL) {
                productionFromage = pF;
                productionLaine = pL;
        }

        @Override
        public int getProductionFromage() {
                return productionFromage;
        }

        @Override
        public int getProductionLaine() {
                return productionLaine;
        }

}
```

```java
public class Chevre extends Animal implements ProducteurDeFromage{
        private int productionFromage;

        public Chevre(int pF) {
                productionFromage = pF;
        }

        @Override
        public int getProductionFromage() {
                return productionFromage;
        }

}
```

```java
public class Vache extends Animal implements ProducteurDeFromage{
        private int productionFromage;

        public Vache(int pF) {
                productionFromage = pF;
        }

        @Override
        public int getProductionFromage() {
                return productionFromage;
        }

}
```


Troupeau
- nom : String
+ productionLaine() : int
+ productionFromage() : int
*
*
*
«interface» ProducteurDeLaine
+ getProductionLaine() : int
Animal
«interface» ProducteurDeFromage
+ getProductionFromage() : int
Mouton
- productionFromage : int
- productionLaine : int
«constructor» Mouton(pF : int, pL : int)
Vache
- productionFromage
«constructor» Vache(pF : int)
Chevre
- productionFromage
«constructor» Chevre(pF : int)

Question 13 : Lecture de code et diagramme UML (3.5 Point(s))

Le code Java fourni ne présente aucune erreur. Pour ce dernier, veuillez répondre aux questions suivantes :

1. Donnez la sortie console de ce programme. (i.e : Ce qui s'imprime à l'écran)
2. Donnez le diagramme de séquence représentant l'appel à la fonction : boulangerie.getPrix(type);

```
public class Main {
    public static void main(String[] args){
        Boulangerie boulangerie = new Boulangerie(100);
        Client client = new Client(15);

        boulangerie.addPain(new Pain("Baguette"));
        boulangerie.addPain(new Pain("Sandwich"));
        boulangerie.addPain(new Pain("Traditionnel"));
        boulangerie.addPain(new Pain("Baguette"));
        boulangerie.addPain(new Pain("Traditionnel"));
        boulangerie.addPain(new Pain("Baguette"));

        client.acheteUnPain(boulangerie, "Traditionnel");
}}
```

```
public class Client {
    private double solde;
    private Pain monPain = null;

    public Client(double argent){
        solde = argent;
    }

    public void acheteUnPain(Boulangerie boulangerie, String type){
        System.out.println("Combien coute un " + type);
        double prix = boulangerie.getPrix(type);
        if(prix < solde){
            System.out.println("Très bien, je le prends");
            monPain = boulangerie.vend(this);
        }
    }

    public double pay(double prix){
        solde = solde - prix;
        return prix;
    }
}
```

```
public class Pain {
    private String type;
    private double prix;

    public Pain(String type){
        this.type = type;
        switch (type){
        case "Baguette":
            prix = 2.5; break;
        case "Traditionnel":
            prix = 3; break;
        case "Sandwich":
            prix = 1.5;
        }
    }
```

```
    public double getPrix(){ return prix; }

    public String getType(){ return type; }
}
```

```
import java.util.ArrayList;

public class Boulangerie {
    private double solde;
    private Pain painEnCours;
    protected ArrayList<Pain> stock;

    public Boulangerie(double s){
        solde = s;
        stock = new ArrayList<Pain>();
    }

    public void addPain(Pain p){
        stock.add(p);
    }

    public double getPrix(String type){
        deposePain();
        for(Pain pain: stock){
            String painType = pain.getType();
            if(painType.equals(type)){
                prendPain(pain);
                System.out.println("Il coute " + pain.getPrix() + " euros");
                return pain.getPrix();
            }
            else{
                System.out.println("heum, pas celui-ci, c'est un " + painType);
            }
        }
        return 0;
    }

    public void prendPain(Pain pain){
       painEnCours = pain;
    }

    public void deposePain(){
        painEnCours = null;
    }

    public Pain vend(Client client) {
        solde = solde + client.pay(painEnCours.getPrix());
        stock.remove(painEnCours);
        Pain painVendu = painEnCours;
        painEnCours = null;
        System.out.println("Merci");
        return painVendu;
    }
}
```

La sortie du programme est la suivante :
Combien coute un Traditionnel
heum, pas celui-ci, c'est un Baguette
heum, pas celui-ci, c'est un Sandwich
Il coute 3.0 euros
Très bien, je le prends
Merci