HAI717I - Programmation par objets (Examen)
13 janvier 2021
16h-18h
Université de Montpellier – Faculté Des Sciences
Master informatique (ICO), géomatique, bioinformatique, Physique numérique

Dans ce sujet, nous élaborons autour des notions d'additif alimentaire et de plat préparé. Un additif alimentaire est un produit ajouté à une préparation afin d'en réduire l'oxydation ou d'en améliorer la conservation, l'aspect, la texture ou le goût. La notion de plat préparé est ici large, et inclut tout ce que l'industrie agro-alimentaire produit pour la consommation humaine. Une énumération décrivant les différentes fonctions possibles pour un additif et le début du code d'une classe **Additif** vous sont donnés ci-dessous.

```
public enum TypeFonction {antioxydant, conservation, aspect, gout, texture}

public class Additif {
  private String nom = "nom inconnu";
  private String code = "Exxx";
  private TypeFonction fonction = TypeFonction.conservation;
  private boolean autoriseFrance = false;

  public Additif() {}
  public Additif(String nom, String code, TypeFonction fonction, boolean autoriseFrance) {
                this.setNom(nom);
                this.setCode(code);
                this.setFonction(fonction);
                this.setAutoriseFrance(autoriseFrance);
  }

  public String getNom() {return this.nom;}
  public void setNom(String nom) {this.nom = nom;}
  public String getCode() {return this.code;}
  public void setCode(String code) {this.code = code;}
  public TypeFonction getFonction() {return this.fonction;}
  public void setFonction(TypeFonction fonction) {this.fonction = fonction;}
  public boolean isAutoriseFrance() {return this.autoriseFrance;}
  public void setAutoriseFrance(boolean autoriseFrance) {this.autoriseFrance = autoriseFrance;}
}
```

Q1 **(4 points).** Deux sous-classes d'additifs seront considérées : les additifs naturels et les additifs de synthèse.

Les *additifs naturels* sont des additifs décrits en plus par le produit naturel d'*origine*, c'est-à-dire dont ils sont extraits (chaîne de caractères) et par le *type* de cette origine, qui prend sa valeur dans l'énumération qui suit : **public enum TypeOrigine {mineral, vegetal, animal, sousProduitAnimal**}. Les sous-produits animaux correspondent par exemple aux œufs et aux laitages.

Les additifs de synthèse sont des additifs créés par un procédé de synthèse chimique. Ils sont décrits par la *stratégie* de synthèse (chaîne de caractères).

Ecrire pour les sous-classes **AdditifNaturel** et **AdditifDeSynthese** :

- Leur entête
- Leur(s) attribut(s)
- Leurs constructeurs avec paramètres permettant d'initialiser tous leurs attributs

On admettra pour la suite que les accesseurs de forme **get** et **set** existent pour les attributs de ces deux classes et que les trois classes disposent d'une méthode **toString**. N'écrivez pas ces méthodes.

```
public class AdditifDeSynthese extends Additif{
        private String strategieSynthese;
```

```
        public AdditifDeSynthese(String nom, String code, TypeFonction fonction, boolean autoriseFrance,
                                String strategieSynthese) {
                super(nom, code, fonction, autoriseFrance);
                this.setStrategieSynthese(strategieSynthese); // ou une affectation
        }
…
public class AdditifNaturel extends Additif {
        private String origine = "origine inconnue";
        private TypeOrigine typeOrigine = TypeOrigine.mineral;
        public AdditifNaturel(String nom, String code, TypeFonction fonction, boolean autoriseFrance, String
        origine, TypeOrigine typeOrigine) {
                super(nom, code, fonction, autoriseFrance);
                this.setOrigine(origine); // ou une affectation
                this.setTypeOrigine(typeOrigine); // ou une affectation
        }
```

Entêtes : 1 point
Attributs : 1 pour ceux d'AdditifNaturel, O,5 pour celui d'AdditifDeSynthese
Constructeurs : les appels à super : 0,75, pour le reste : 0,75

Q2 **(3 points).** Ecrire dans les trois classes une méthode de signature **public void saisie(Scanner clavier)** qui permette de saisir des valeurs pour tous les attributs d'un objet.

1 point pour chaque méthode

```
// Additif
public void saisie(Scanner clavier) {
                System.out.println("Entrer le nom : ");
                this.setNom(clavier.next());
                System.out.println("Entrer le code : ");
                this.setCode(clavier.next());
                System.out.println("Entrer la fonction : ");
                this.setFonction(Fonction.valueOf(clavier.next()));
                System.out.println("Entrer true/false pour l'autorisation en France : ");
                this.setAutoriseFrance(clavier.nextBoolean());
        }
// Additif nature
        public void saisie(Scanner clavier) {
                super.saisie(clavier);
                System.out.println("Entrer l'origine");
                this.setOrigine(clavier.next());
                System.out.println("Entrer le type d'origine");
                this.setTypeOrigine(TypeOrigine.valueOf(clavier.next()));
        }
// Additif de synthese
        public void saisie(Scanner clavier) {
                super.saisie(clavier);
                System.out.println("Entrer la stratégie de synthèse");
                this.setStrategieSynthese(clavier.next());
        }
```

Q3 **(3 points).** Ecrire dans les trois classes une méthode de signature **public TypeRegime regime()** qui retourne le régime alimentaire le plus restrictif autorisant l'additif.
Vous utiliserez l'énumération : **public enum TypeRegime { general, vegetarien, vegetalien }**.

Cette méthode se comporte différemment dans les deux sous-classes, comme décrit ci-dessous :

Additif naturel		*Additif de synthèse*	
Type d'origine	Régime à retourner		Régime à retourner
animal	**general**	Dans tous les cas	**general**
sousProduitAnimal	**vegetarien**		
vegetal	**vegetalien**		
mineral	**vegetalien**		

Après l'avoir écrite, indiquez si vous avez dû modifier quelque chose dans l'entête de la classe **Additif** (expliquer pourquoi cela a été ou non nécessaire). Il y a plusieurs solutions, expliquez simplement votre choix.

1 point
// Additif et si elle est abstract on doit ajouter *public abstract class Additif*
// si elle retourne le régime général, c'est accepté aussi et pas de changement pour l'entête de
// la classe

```
public abstract TypeRegime regime();
```

1,5 point

```
// Additif naturel
public TypeRegime regime() {
        // le régime le plus restrictif autorisant l'additif est retourné
        switch(this.typeOrigine) {
        case mineral:
        case vegetal: return TypeRegime.vegetalien;
        case sousProduitAnimal: return TypeRegime.vegetarien;
        default: return TypeRegime.general;
        }
}
```

0,5 point

```
// Additif de synthèse
public TypeRegime regime() {
        return TypeRegime.general;
}
```

Q4 **(0,5 point).** Ecrire l'instruction d'une méthode **main** par laquelle on crée :
- un additif naturel, l'acide sorbique, de code E200, dont la fonction est de conserver l'aliment. Il est autorisé en France, extrait des baies du sorbier, et donc de type végétal.

```
Additif a3 = new AdditifNaturel("acide sorbique","E200",Fonction.conservation,
                                true,"baies du sorbier",TypeOrigine.vegetal);
```

Q5 **(1,5 points)**. Ecrire l'entête, les attributs et le constructeur avec paramètres d'une classe **PlatPrepare** avec les indications suivantes :
- un plat préparé est décrit par une identification (chaîne de caractères qui est invariable) et une recette (chaîne de caractères),
- il contient une liste d'additifs alimentaires.

```
public class PlatPrepare {
        private final String identification; //0,25
        private String recette; //0,25
        private ArrayList<Additif> listeAdditifs = new ArrayList<> (); //0,75

        public PlatPrepare(String identification, String recette) { //0,25
                this.identification = identification;
                this.recette = recette;
        }
```

Q6 **(3 points).** Ecrire, dans la classe **PlatPrepare,** une méthode **contientCode**, qui prend en paramètre une chaîne de caractères représentant un code et retourne vrai si le plat préparé contient un additif ayant ce code.

```
public boolean contientCode(String code) {
        for (Additif a : this.listeAdditifs)
            if (a.getCode().equals(code))
                return true;
        return false;
    }
```

Q7 **(1,5 points)**. Ecrire, dans la classe **PlatPrepare,** une méthode **ajouteAdditif**, qui prend en paramètre un additif, l'ajoute à la liste des additifs si le plat préparé ne contient pas déjà un additif ayant ce code, et affiche un message d'erreur sinon.

```
public void ajouteAdditif(Additif a) {
        if (this.contientCode(a.getCode()))
            System.out.println("additif déjà présent");
        else
            this.listeAdditifs.add(a);
    }
```

Q8 **(2,5 points)**. Ecrire, dans la classe **PlatPrepare,** une méthode **listeConservateurs**, qui retourne la liste des additifs de ce plat qui ont une fonction de conservation.

```
public ArrayList<Additif> conservateurs(){
        ArrayList<Additif> conservateurs = new ArrayList<> ();
        for (Additif a : this.listeAdditifs)
            if (a.getFonction()==TypeFonction.conservation)
                conservateurs.add(a);
        return conservateurs;
    }
```

Q9 **(1 points)**. Ecrire les instructions permettant de compléter le **main** par : *(1) la création* d'un plat préparé (crème caramel, dont la recette est par cuisson de sucre, lait, œuf et vanille) contenant l'additif créé en question Q4 *puis (2) l'affichage* de la liste des conservateurs qu'il contient. Rappelez-vous que la classe **ArrayList** contient une méthode **toString** qui liste les éléments de la liste entre deux crochets en appelant leur méthode **toString**.

```
PlatPrepare pc = new PlatPrepare("creme caramel","cuisson de sucre, lait, œuf et vanille"); //0,25
p.ajouteAdditif(a3); //0,5
System.out.println(pc.conservateurs()); //0,25
```