UEF4.3. Programmation Orientée Objet

CONTROLE FINAL

(2h- Documents et téléphones portables interdits)

Remarques :

1) Ceci est un corrigé type. Toute solution correcte n'y figurant pas est acceptée
2) Les réponses aux questions sont parfois longues et détaillées volontairement pour servir de révision. Le minimum attendu est ce qui est surligné en jaune.

Exercice 1 (10,25 points)

1.a) Quel est le principe fondamental de la POO qui n'est pas respecté dans ce code ? Expliquez.

- Le principe fondamental de la POO qui n'est pas respecté dans ce code est l'encapsulation **(0.25)** car le modificateur des attributs est `public`. **(0.25)**

1.b) Définissez ce principe et expliquez comment son application peut améliorer la qualité d'un programme ?

- L'encapsulation consiste à regrouper au sein d'une même classe les attributs et méthodes d'une classe et d'en réglementer l'accès. **(0.25)** Le principe de l'encapsulation est que, vu de l'extérieur, un objet se caractérise uniquement par ses méthodes visibles. Les détails de l'implémentation des méthodes sont masqués aux autres objets ce qui permet de concevoir des programmes plus sûrs, plus lisibles et plus faciles à maintenir **(0.25)**, car une modification de la structure des données d'un objet ne se répercute pas sur les objets qui communiquent avec lui (invoquent ses méthodes).

1.c) Donnez une version améliorée du programme.

Rendre les attributs private : 0.25
Ajouter getVitesse() et setVitesse() : 0.25
Remplacer moteur.vitesse par moteur.getVitesse : 0.25

Remarques: les getters et setters des autres attributs peuvent être ajoutés mais nous n'avons mis ici que ce qui est nécessaire pour le moment. Toute autre solution correcte est acceptée.

2.a) Le code du constructeur de la classe Voiture est-il correct ? Si la réponse est « non » expliquez pourquoi et corrigez-le.

Le code est incorrect **(0.25)** au niveau du constructeur. Pour le corriger il faut soit ajouter un constructeur sans arguments dans la classe `Vehicule` ou bien modifier le constructeur de Voiture comme suit :

```
public Voiture(Moteur m,int vmax){
        super(m);
        vitesse Maximale = vmax;
    }
```
(0.25)

Remarque : Toute erreur fait perdre le point y compris le fait de ne pas mettre `super(m);` en premier

2.b) Est-ce une surdéfinition (surcharge) ou une redéfinition ?

C'est une surdéfinition **(0.25)**

- Donnez le code de cette méthode en indiquant les éventuelles modifications à apporter au code existant.

```
public void accelerer(int vitesse){
    try{
    double v = getMoteur().getVitesse();;
    while(v < vitesse) {
          v++;    (0.25)
(0.25)    if (v > vitesseMaximale) throw new DepassementVitesseException();(0.25)
          super.accelerer(v);   (0.25)
          }
    }
    catch(DepassementVitesseException e){
    System.out.println("Impossible  de dépasser la vitesse Maximale !" ); (0.25)
    }
}
```

Modifications

1/ ici il faut ajouter getMoteur dans la classe Vehicule ou bien il faut rendre cet attribut « protected » **0.25**:

2/ Ajouter la classe

class DepassementVitesseException extends Exception{} //pas de note car elle est donnée dans l'énoncé

2.c) Est-ce une surdéfinition ou une redéfinition ?

C'est une redéfinition **(0.25)**

- Donnez le code de cette méthode.

```
public void afficherVitesse(){
super.afficherVitesse();(0.25)
System.out.println("Vitesse à ne pas dépasser"+ vitesseMaximale);  (0.25)
    }
```

3.a) Qu'affiche ce code ? **(0.25)** *8 lignes = 2pts

```
Le véhicule portant le matricule 1 roule à une vitesse de 200.0Km/h
-----------------
Vous devez entrer une valeur entière !
Le véhicule portant le matricule 2 roule à une vitesse de 0.0Km/h
Vitesse à ne pas dépasser: 160Km/h
-----------------
****************
Le véhicule portant le matricule 1 roule à une vitesse de 200.0Km/h
-----------------
Vous devez entrer une valeur entière !
Le véhicule portant le matricule 2 roule à une vitesse de 0.0Km/h
Vitesse à ne pas dépasser: 160Km/h
-----------------
```

3.b) Il existe une incohérence dans le résultat retourné par ce code. A quel niveau réside cette incohérence et à quoi est-elle due ?

L'incohérence réside au niveau de l'affichage produit par la deuxième boucle for pour vehicules[1] **(0.25)** qui est de type effectif Voiture. En effet, on remarque que même si l'argument de la méthode accélérer est de type int, c'est la méthode accélérer ayant comme argument un paramètre de type double qui est exécutée et ce malgrè l'existence d'une méthode plus appropriée au niveau de la classe Voiture. **(0.25)** Ceci est dû au fait que le choix de la signature et du type de retour de la méthode se fait sur la base du type (qui est Vehicule) et non pas sur la base du type effectif (qui est Voiture) et que dans la classe Vehicule la seule méthode acceptable est `void accelerer(double).` **(0.25)** A l'exécution, c'est la méthode `void accelerer(double)` de la classe Voiture qui a été exécutée selon le fonctionnement de la ligature dynamique **(0.25)**

4.

Instruction	Erreur à la compilation ?	Correction ?	Correction	Erreur à l'execution ?

v1 = h1;
v2 = h2;
h1 = v1;
h2 = h1;
v1 = h2;
	Oui
	Oui
	Non **(0.25)**
	Non **(0.25)**
	Oui
		Oui
		Oui
		
		Oui
			v1=(Voiture)h1;
			v2=(Voiture)h2;
			
			v1=(Voiture)h2;
				Oui **(0.5)pour toute la ligne**
				Non **(0.5) pour toute la ligne**
				
				Non **(0.5) pour toute la ligne**

Remarque : accorder 0.25 si une ligne est correcte partiellement

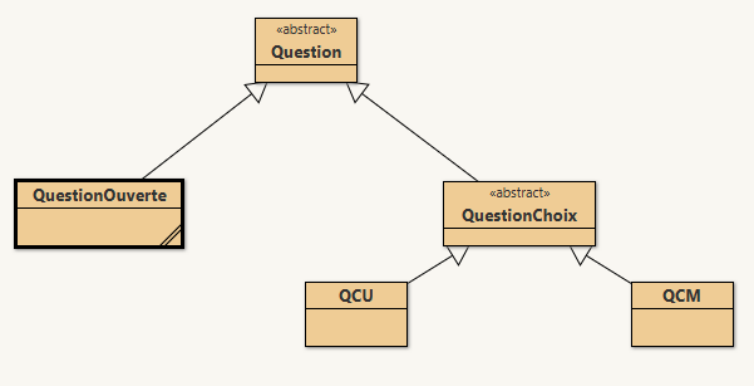
5. Si dans ce programme seule la classe Vehicule utilise la classe Moteur, quelle modification pourriez-vous apporter à la conception et pourquoi ?

Dans ce cas, on peut concevoir Moteur comme étant une classe interne à la classe Vehicule **(0.25)**.
Ceci augmentara l'encapsulation, la lisibilité et facilitera la maintenance du code **(0.25)**

Exercice 2 (10.75)

1. Proposez une modélisation orientée objet pour implémenter les différents types de questions en respectant au mieux les principes fondamentaux de la POO.
   a. Tracez le diagramme des classes en précisant le type des classes et les liens entre elles.

Classes : 4*0.25 (QuestionChoix n'est pas indispondable)
Héritage 3*0.25 (0.25 pour chaque sous classe de Question)
Abstract (pour Question) : 0.25



Remarque : accepter aussi la solution qui fait dériver QCU et QCM de Question. Dans ce cas la classe QuestionChoix est inutile.

b. Précisez pour chacune des classes ses attributs et ses méthodes (hormis les constructeurs, les getters et les setters) en utilisant le tableau suivant.

Abstract class Question	
Liste des attributs	
Modificateur d'accès Type de l'attribut nom de l'attribut	signification
private/protected String enonce **0.25** private/protected String solution **0.25** private/protected double nbPoints **0.25**	…
Liste des méthodes	
public abstract double evaluer(String reponse) **0.25**	Evalue la réponse de l'étudiant et retourne la note correspondante. Dans le cas d'une QCM les réponses sont concaténées et séparées par des virgules.

Abstract class QuestionOuverte	
Liste des attributs	
Modificateur d'accès Type de l'attribut nom de l'attribut	signification
	…
Liste des méthodes	
public double evaluer(String reponse) **0.25**	redéfinition

Abstract class QuestionAChoix	
Liste des attributs	
Modificateur d'accès Type de l'attribut nom de l'attribut	signification
Private/protected ArrayList<String> propositions **0.25**	Un tableau contenant les propositions
Liste des méthodes	

class QCU	
Liste des attributs	
Modificateur d'accès Type de l'attribut nom de l'attribut	signification
	…
Liste des méthodes	
Public double evaluer(String reponse) **0.25**	redéfinition

class QCM	
Liste des attributs	
Modificateur d'accès Type de l'attribut nom de l'attribut	signification
Private int penalite **0.25**	Nombre de points à soustraire dans le cas d'une réponse erronnée
Liste des méthodes	
Public double evaluer(String reponse) **0.25**	redéfinition

2.a) Quel est le concept Orienté objet qui permet de regrouper toutes les questions au sein de la même collection ?

Polymorphisme (d'objets) **0.25**

2.b) En sachant qu'une question ne doit pas figurer plus d'une fois dans le même quiz, quel est le type de collections le plus adéquat ? Donnez l'instruction permettant de créer cette collection.

HashSet **0.25**

HashSet<Question> questions = new HashSet<Question>(); **0.25**

Si la collection choisie est TreeSet

TreeSet<Question> questions = new TreeSet<Question>();

2.c) Que faut-il faire pour garantir que le type de collections choisi fonctionne correctement ? Expliquez en donnant le code java nécessaire.

Il faut redéfinir les méthodes equals et hashcode de la classe object

```
public int hashCode(){
        return enonce.hashCode();
    }  0.25

    public boolean equals(Object q){
        if (!(q instanceof Question)) return false;
        else return (enonce.equals(((Question)q).enonce));
    }  0.25
```

Remarque : si la collection choisie est TreeSet il faut que la classe Question implémente Comparable et redéfinisse compareTo pour définir un critère d'ordonnancement des questions. Une solution simple et à moindre coût consiste à ordonner les questions selon l'ordre alphabétique de l'énoncé mais toute autre solution correcte est acceptée.

3.a) A quoi sert l'instruction à la ligne 14 ?

Cette instruction crée un ArrayList à partir d'un Set **0.25**

3.b) Est-ce que la collection choisie en réponse à la question 2.b garantit que le résultat retourné par la méthode `evaluerQuiz` est correct ? Expliquez.

1. Si la collection choisie est un HashSet NON **0.25** car les questions seront stockées dans l'objet questions de type ArrayList selon l'ordre de leur stockage dans le HashSet questions de la class Quiz qui n'est pas forcément celui de leur apparition sur le questionnaire **0.25**. La correspondance peut donc être erronée et par conséquent l'évaluation aussi.
2. Si la collection choisie depuis le début est TreeSet la réponse est Oui **0.25** car les questions sont ordonnées selon l'ordre défini par compareTo **0.25**

3.c) Si votre réponse à la question précédente est non, proposez un autre type de collections plus adéquat en expliquant les modifications à apporter à votre programme.

1. Il faut transformer le HashSet en TreeSet **0.25**
2. Il faut que la classe Question implémente Comparable **0.25** et redéfinisse compareTo **0.25**pour définir un critère d'ordonnacement des questions. Une solution simple et à moindre coût consiste à ordonner les questions selon l'ordre alphabétique de l'énoncé.

Remarque : si la réponse était OUI car TreeSet a été prévu depuis le début accorder 0.75

3.d) Complétez la méthode `evaluerQuiz.`

```
public double evaluerQuiz(){
 double note = 0;
 ArrayList<Question> questions = new
 for(int i = 0; i<questions.size(); i++)  0.25
  note = note + (questions.get(i)).evaluer(reponses.get(i));
 return note;
}
```

```
// cumul des notes  0.25
//Appel à la méthode
evaluer  0.25
//appel à la méthode get(i) :
0.25
```

4. Le principe fondamental qui nous a simplifié ici l'écriture de la méthode evaluerQuiz est le polymorphisme **0.25** et plus précisément, le polymorphisme de méthode. Il existe une méthode abstraite evaluer dans Question qui est redénie dans toutes les sous-classes ce qui nous a permis d'écrire une simple boucle pour parcourir l'ensemble des questions et appeler cette méthode **0.25**. Grâce à la ligature dynamique **0.25**, la méthode correspondant au bon type est appelée à chaque fois.

```
GridPane root = new  GridPane() ; 0.25// aucun autre layout n'est accepté car
```

```
        root.add(enonce,0,1);
        root.add(reponseLabel,0,2);
        root.add(reponse,0,3);
        root.add(submitButton,0,4);
```

0.25 pour l'ajout de tous les composants
+ 0.25 si l'emplacement des indices des lignes et colonnes est correct

```
        //Gestion de l'évènement associé au click sur le bouton
        submitButton.setOnAction(new EventHandler<ActionEvent>()  0.25 {
            public void handle(ActionEvent e) 0.25 {
             System.out.println(reponse.getText())0.25;
            }
        });
        // ou bien en utilisant une expression lambda
        submitButton.setOnAction((ActionEvent e) -> 0.5{
             System.out.println(reponse.getText())0.25;
        });
```