UEF4.3. Programmation Orientée Objet

Contrôle Intermédiaire

1H45 - Documents & Téléphone interdits

Exercice 1 (4,5pts):

Indiquez pour chacun des codes suivants:

- S'il compile ou pas.
- Si la réponse est oui, indiquez ce qu'il affiche à son exécution,
- Si la réponse est non :
  - expliquez pourquoi,
  - proposez, quand c'est possible, des corrections puis indiquez ce qu'il affiche à son exécution.

i./

```
class A{
int i;
A(int i){ this.i = i; }
}
class B extends A{
A a;
int i = 2;
B(){ this.a = this; }
B(A a){
super(3);
this.a = a;
}
```

```
public static void main(String[]
args){
A a2 = new A(5);
B b1 = new B(a2);
B b2 = new B();
System.out.println(b1.i);
System.out.println(((A) b1).i);
System.out.println(b1.a.i);
System.out.println(b2.i);
System.out.println(((A) b2).i);
System.out.println(b2.a.i);
}
}
```

Solution 1pts

le code ne compile pas: car il faut un constructeur sans paramètre dans A (pour que le constructeur sans paramètre de B puisse faire l'appel automatique au super-constructeur sans paramètre).0,25

correction: ajouter un constructeur sans paramètre dans A 0,25

affichage: 0,5 (le tout ou rien)

2

3

5

2

0

0

ii./

```
public class Test {
public static void main ( String
args[ ]) {
int k ; int j=0;
try {
k = 1 / j;
throw new
  IndexOutOfBoundsException();
}
catch(RuntimeException ex ) {
System.out.println(" Runtime
  Exception");
}
catch(IndexOutOfBoundsException
ex){
System.out.println("Dépassemnt
de la capacité" );
}
catch(ArithmeticException ex){
System.out.println("Exception
  arithmétique " );
}
finally {
System.out.println(" fin");
 }
System.out.println(" suite");
}
}
```

Solution 1,25pts

le code ne compile pas, l'ordre des catch n'est pas adéquat.0,25

correction: mettre le premier catch `catch(RuntimeException ex )` **en dernier juste avant finally 0,25**

affichage: 0,25 pour l'exception+ 0,5 pour fin et suite (0 si l'une manque)

```
Exception arithmétique
fin
suite
```

iii./

```
package z;
public class Test {
public static void main ( String args[ ]) {
    D d=new D(3);
    System.out.println(d.getI());
}
}
```

```
package p;
interface I {
public int getI();
}
```

```
package q;
abstract class C
implements I {
private int i;
C(int i) {
this.i = i;
}
}
```

```
package q;
public class D
extends C {
D(int x) {
super(x);
this.i = 0;
}
}
```

Solution 2,25pts

le code ne compile pas: 0,75 (toutes les explications)

l'interface I est invisible en dehors de son paquetage et ne peut donc pas être implémentée par la classe C. (qui doit une fois ce pb résolu implémenter la méthode getI())

L'attribut "i" est invisible en dehors de sa classe et ne peut donc être accédé dans la classe D.

Il manque les import des classes entre les packages.

Le constructeur de D doit être public

correction 1,25:

-changer le modes d'accès de l'interface I qui devient public,

-implémenter la méthode getI soit dans la classe abstraite C ou bien dans la classe D

-soit modifier le mode d'accès de l'attribut i ou ajouter un setter et l'utiliser.

-importer l'interface I dans le code de la classe C

-importer la classe D dans le code de la classe Test

+ rendre le constructeur de D public

Affichage: 0,25

0

Exercice 2 (8pts)

Nous souhaitons gérer une hiérarchie de moyens de transport, où les classes *Moto*, *Voiture*, *Bus* dérivent de la classe *MoyenTransportRoutier* et les classes *Tramway*, *Metro*, *Train* dérivent de la classe *MoyenTransportFerrovier*. Les classes *MoyenTransportRoutier* et *MoyenTransportFerrovier* dérivent de la classe *MoyenTransport*.

1. Quelles sont les affections correctes parmi ces quatre propositions? : **(1pts = 0,25*4)**

a. MoyenTransportRoutier [] tab = new Bus [10] ; **vrai**

b. Voiture [] tab = new Tramway [10] ; **faux**

c. MoyenTransport[] tab = new MoyenTransport [10] ; **vrai**

d. MoyenTransportRoutier [] tab = new MoyenTransport [10] ; **faux**

2. Laquelle des propositions précédentes permet de créer un tableau pouvant contenir des objets de type Voiture et Tramway (a, b, c ou d)? **(0,5pts)**

réponse: c

3. Nous supposons que la méthode afficher() est définie dans la classe *MoyenTransport* et redéfinie dans les classes *MoyenTransportRoutier*, *Tramway* et *Train* . Dites quelles méthodes *afficher()* seront appelées suite à l'exécution du code suivant : **(2,25pts)**

```
public class Test {
public static void main(String args[]) {
MoyenTransport m1 = new MoyenTransport(); MoyenTransport m2 = new Train();
MoyenTransportRoutier m3 = new Moto();
MoyenTransportFerrovier m4 = new Tramway();  Train m5 = new Train();
MoyenTransportFerrovier m6 = new Metro();    Moto m7=new Moto();
m1.afficher(); m2.afficher(); m3.afficher();
m4.afficher(); m5.afficher(); m6.afficher(); m7.afficher();
((MoyenTransport)m3).afficher();((MoyenTransportFerrovier)m5).afficher();
}
}
```

corrigé

Instruction	**méthode appelée:** afficher de la classe
m1.afficher()	MoyenTransport
m2.afficher()	Train
m3.afficher()	MoyenTransportRoutier
m4.afficher()	Tramway
m5.afficher()	Train
m6.afficher()	MoyenTransport
m7.afficher()	MoyenTransportRoutier
((MoyenTransport)m3).afficher()	MoyenTransportRoutier
((MoyenTransportFerrovier)m5).afficher();	Train

4. En considérant les initialisations faites dans le précédent code, indiquez pour chacune des instructions suivantes : **(4,25pts)**

- si elle est correcte ou si elle provoque une erreur à la compilation ou à l'exécution.
- si on peut résoudre le problème de la compilation par un cast (et donnez ce cast), et si oui, s'il reste une erreur à l'exécution ou non.

```
1. m1 = m2;          2. m4 = m2;       3. m3 = m1;    4. m4 = m5;          5.
m5 = (Train)m2;    6. m6 = m7;       7. m7=m3;      8. m6=(Metro)m2;
```

Corrigé 4,25Pts

Instruction	**Compilation :** ***OK*** **ou** ***Erreur*** **avec explication**	**Correction** ***en cas d'erreur***	**Exécution :** ***OK*** **ou** ***Erreur*** **avec explication**
m1 = m2; **0,25**	ok	-	ok
m4 = m2; **0,75**	erreur sous type	m4 = (MoyenTransport Ferrovier)m2;	ok
m3 = m1; **0,75**	erreur sous type	m3 = (MoyenTransport Routier)m1;	erreur d'execution m1 ne pointe pas vers un objet de type MoyenTransportRou tier
m4 = m5; **0,25**	ok	-	ok
m5 = (Train)m2; **0,5**	ok	-	ok
m6 = m7; **0,5**	erreur de compilation, types non compatibles	non	-
m7=m3; **0,75**	erreur sous type	m7 = (Moto)m1;	ok
m6=(Metro)m2; **0,5**	ok	-	erreur d'execution m1 ne pointe pas vers un objet de type Metro

Exercice 3 (7,5 points)

On veut écrire un programme gérant le salaire des enseignants d'une université. Chaque enseignant a un nom, un prénom et un certain nombre d'heures de cours assurées dans l'année. Il existe plusieurs catégories d'enseignants. Les enseignants-chercheurs sont payés avec un salaire fixe plus le revenu des heures complémentaires (un revenu fixe pour chaque heure effectuée). Les heures complémentaires sont les heures effectuées au-delà des 192h. Ainsi, un enseignant-chercheur qui donne 200h de cours dans l'année recevra (salaireFixe*12+ revenuHeureComplémentaire*8) DA sur l'année. Les vacataires sont des enseignants venant d'un organisme extérieur à l'université. Cet organisme est décrit par une chaîne de caractères. Les vacataires sont payés de l'heure selon un revenu fixe particulier (revenuHeureVacataire). Les étudiants en post-graduation (doctorants) peuvent assurer des cours et sont payés selon un revenu fixe propre à leur statut (revenuHeureDoctorant), mais ne peuvent dépasser un certain nombre d'heures de cours (nombreHeuresMax). Un doctorant qui fait plus d'heures que le nombre maximum ne recevra que le salaire correspondant au nombre

maximum de cours. Un début de code a été écrit par un programmeur qui débute dans la POO.

```
public class Enseignant {

  public String nom;
  public String prenom;
  public int nbHeures;
   int type; // 0: enseignant-Chercheur, 1: enseignant vacataire, 2:
doctorant


public final double salaireFixe ;
public final double revenuHeureComplementaire;
public final double revenuHeureVacataire;
public final double revenuHeureDoctorant;
public final int nbHeuresMax ;

// Constructeur

public Enseignant (String nom, String prenom, int nbHeures, int type) {
      this.nom = nom;
      this.prenom = prenom;
      this.nbHeures = nbHeures;
      this.type = type;
      }

public double calculerSalaire() {
      if (type == 0 ){.../* calculer le salaire d'un enseignant-chercheur;
*/}
      else if (type == 1) //calculer le salaire d'un enseignant vacataire
           else if (type == 2) //calculer le salaire d'un doctorant
                  else return 0; // si le type est différent de 0, 1 et 2

      }
}
```

1. Quels sont les principes de la POO non respectés dans ce code ? expliquez (Toute réponse sans explication sera rejetée)
2. Proposez un nouveau programme conforme aux principes de la POO qui calcule le salaire de chaque enseignant.
3. Expliquez les avantages du programme que vous avez proposé dans la question précédente.
4. Donnez l'instruction qui permet de déclarer un tableau contenant à la fois des références vers des enseignants-chercheurs, des enseignants vacataires et des doctorants. Quel principe OO avez-vous utilisé ?
5. Donnez l'instruction permettant de calculer les salaires de tous les éléments du tableau précédent. Quelle notion OO est utilisée dans cette instruction. ?
6. On désire pouvoir trier le tableau d'enseignants suivant l'ordre décroissant du nombre d'heures effectuées en utilisant la méthode sort de la classe Arrays. Quels sont les modifications à apporter au code source ?

7. On souhaite que le programme affiche un message lorsqu'un doctorant dépasse le nombre d'heures maximum. Expliquez (sans donner le code source) par quel moyen cela pourrait être implémenté.

Solution

1. Quels sont les principes de la POO non respectés dans ce code ? expliquez (Toute réponse sans explication sera rejetée). **2 points**

   Dans ce code source plusieurs principes de la POO ne sont pas respectés:

   - le principe d'abstraction: A la lecture de l'énoncé, nous repérons trois types différents qui sont Enseignant-chercheur, Enseignant vacataire et Doctorant. Ces trois types doivent chacun être représentés par une classe au lieu de les distinguer au moyen d'une variable entière prenant trois valeurs différentes. **(0.5 point)**
   - Les principes d'héritag et de polymorphismee (lié au principe d'abstraction): puisque les trois types partagent des attributs et un comportement communs, ces derniers devraient être regroupés au sein d'une classe mère abstraite Enseignant qui contiendrait la méthode abstraite calculerSalaire() qui serait redéfinie au niveau des classes dérivées EnseignantChercheur, EnseignantVacataire et Doctorant. **0.5 *2 point**
   - Le principe d'encapsulation: les attributs de la classe Enseignant sont publics ce qui est contraire aux bonnes pratiques de la POO qui préconisent de garder l'implémentation cachée. **0.5 point**

Remarque: citer le principe sans fournir une explication valable vaut 0

2. Proposez un nouveau programme conforme aux principes de la POO qui calcule le salaire de chaque enseignant. **3.25 points**

A la lumière de l'analyse critique faite en réponse à la question 1, voici une version du programme respectant les principes de la POO

(abstract: 0.5, extends dans les 3 sous-classes : 0.5, la méthode abstraite calculerSalaire dans la classe Enseignant 0.5, attributs spécifiques des 3 sous-classes: 0.5, appel au constructeur de la classe mère dans les 3 sous-classes: 0.5. redéfinition correcte de la méthode abstraite calculerSalaire dans chaque sous classe: 0.25*3= 0.75)

```java
public abstract class Enseignant {

    protected String nom;
    protected String prenom;
    protected int nbHeures;

    // Constructeur
    public Enseignant (String nom, String prenom, int nbHeures) {
        this.nom = nom;
        this.prenom = prenom;
        this.nbHeures = nbHeures;
    }

    public abstract double calculerSalaire();
}
```

```java
public class EnseignantChercheur extends Enseignant {
    public final double salaireFixe ;
    public final double revenuHeureComplementaire ;

    public EnseignantChercheur (String nom, String prenom, int nbHeures, double salaireFixe,
            double  revenuHeureComplementaire ) {
        super(nom,prenom,nbHeures);
        this.salaireFixe = salaireFixe;
        this.revenuHeureComplementaire =  revenuHeureComplementaire;
    }

    public double calculerSalaire() {
        return (salaireFixe*12 + (nbHeures - 192) * revenuHeureComplementaire);
        }
}
```

```java
public class EnseignantVacataire extends Enseignant  {

public String organisme;
public final double revenuHeureVacataire;

public EnseignantVacataire (String nom, String prenom, int nbHeures,
        String organisme, double revenuHeureVacataire) {
    super(nom,prenom,nbHeures);
    this.organisme = organisme;
    this.revenuHeureVacataire = revenuHeureVacataire;
}

public double calculerSalaire() {
    return (nbHeures * revenuHeureVacataire);
    }
}
```

```
public class Doctorant extends Enseignant {

    public final double revenuHeureDoctorant;
    public final int nbHeuresMax;

    public Doctorant (String nom, String prenom, int nbHeures,
            double revenuHeureDoctorant, int nbHeuresMax) {
        super (nom, prenom, nbHeures);
        this.revenuHeureDoctorant = revenuHeureDoctorant;
        this.nbHeuresMax = nbHeuresMax;
    }

    public double calculerSalaire() {
    if (nbHeures<= nbHeuresMax) return (nbHeures * revenuHeureDoctorant);
    else return (nbHeuresMax * revenuHeureDoctorant);
    }
}
```

3. Expliquez les avantages du programme que vous avez proposé dans la question précédente.

Le programme proposé en réponse à la question précédente respecte les principes fondamentaux de la POO ce qui le rend plus maintenable, plus évolutif et plus réutilisable. Ces trois critères font de lui un programme de meilleure qualité **0.75 point**

4. Donnez l'instruction qui permet de déclarer un tableau contenant à la fois des références vers des enseignants-chercheurs, des enseignants vacataires et des doctorants. Quel principe OO avez-vous utilisé ?

Enseignant tabEnseignants []; **0.25**

Le principe OO est le polymorphisme (d'objets) **0.25**

5. Donnez l'instruction permettant de calculer les salaires de tous les éléments du tableau précédent. Quelle notion OO est utilisée dans cette instruction. ?

```
for (int i = 0; i<tabEnseignants.length; i++)
                    tabEnseignants.calculerSalaire();
```
0.25

La notion OO est la ligature dynamique **0.25**

6. On souhaite que le programme affiche un message lorsqu'un doctorant dépasse le nombre d'heures maximum. Expliquez (sans donner le code source) par quel moyen cela pourrait être implémenté. **0.5 point**

Ceci peut être implémenté en utilisant une exception personnalisée **0.25 point** que l'on appellera par exemple NbHeuresMaxException. Dans la méthode calculerSalaire de la classe Doctorant, le calcul du salaire se fera dans un bloc try catch et une exception de type NbHeuresMaxException sera lancée si le nombre d'heures enseignées par le doctorant est dépassé. **0.25 point**