Corrigé-Type de l'EMD - Programmation Orientée Objet (L2 RN)-

Les réponses doivent respecter les principes de l'orienté objet, en particulier les deux derniers exercices, toute solution qui ne prend pas en considération ce paradigme elle est rejetée même si elle est correcte.

Exercice 1 : Donnez le résultat si le code suivant est correct. Justifiez votre réponse.

```
class X {
public double g (double x) {return f(x)*f(x);}
public double f (double x) {return x + 1.0;}
}
class Y extends X {
public  double f(int x) {return x + 2.0; }
public double f(double x) { return x + 3.0; }
}
class Exo1 {
public static void main (String args[]) {
X x = new X();
Y y = new Y();
x=y;
System.out.println(x.f(3));
System.out.println(x.g(3.0));
} }
```

Le code est correct, il affiche sur écran :

6.0

36.0

On accepte la réponse en sa globalité avec explication, l'explication consiste à dérouler le programme à la main.

(3 pts)

Exercice 2 : **(4pts)**

Soit le programme suivant :

```
abstract class Crayon {public abstract String ecrire();}
interface Effacable {public void effacer();}
class CrayonGraphite extends Crayon implements Effacable {public void effacer() {System.out.println("effacer trait");}
public String ecrire() {return "Graphite";}}
class CrayonGraphiteDur extends CrayonGraphite {
public String ecrire() {return "Gris";} }
class CrayonGraphiteGras extends CrayonGraphite {}
class CrayonCouleur extends Crayon {
public String ecrire() {return "Couleur";}}
class CrayonBleu extends CrayonCouleur {
public String ecrire() {return "Bleu";}}
class CrayonBleuCiel extends CrayonBleu {
public String ecrire() {return super.ecrire() + " Ciel";}
} }
```

Question1: Tracer le diagramme de classes en précisant les types des classes et les relations entre elles.

(1 pt) pour la totalité du schéma

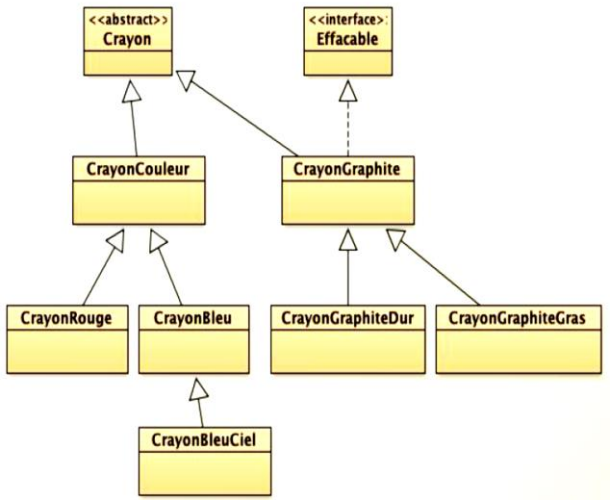
Question2: Qu'affiche le code suivant : **(2 pts pour la totalité de l'affichage avec explication)**

Crayon Bleu Ciel
Crayon Gris
Crayon Couleur
Crayon Graphite
Crayon Bleu

Question3:

Ligne	Solution
CrayonCouleurc0 = trousse[0];	Erreur compilation =>cast**0,25** CrayonCouleur c0 = (CrayonCouleur) trousse[0];**0,25**
CrayonRougec2 = trousse[2];	Erreur compilation CrayonRouge c2 = (CrayonRouge) trousse[2];**0,25**
CrayonBleuc3= (CrayonBleu) trousse[3];	Erreur exécution Crayon Graphite n'est pas un Crayon bleu
Crayon c4=trousse[4];	Ok**0,25**

(0.25 pt pour chaque instruction * 4)



Exercice 3 : Donnez exactement qu'affiche le programme suivant. **(2pts)**

```
class A{
int i=0;
A(int i){ this.i=i;}
int m(A a){ return a.i; } }
class B extends A{ int i=1;
B(){ super(3);
this.i=4;
}
int m(A a){ return super.m(this);}
public static void main(String[] truc){
A a = new B();
System.out.println(a.i);
System.out.println(((B) a).i);
System.out.println(a.m(a));
System.out.println(((B) a).m(a));
A b = new A(5);
System.out.println(b.m(a));
} }
```

Ce programme est correct, il affiche :

3

4

3

3

3

2 pts pour l'affichage en sa globalité avec explication

Exercice 4 : **(2pts)**

Sachant qu'un cheval peut se comporter comme un Animal et il peut se comporter comme un moyen de transport.

Proposez un mécanisme orienté Objet (en java) permettant de représenter ces entités. Expliquez votre démarche.

En utilisant les interfaces. Nous définissions deux interfaces MoyenDeTransport dans laquelle nous allons déclarer tous les comportements des moyens de transport. Une autre interface Animal dans laquelle nous allons également déclarer tous les comportements de la classe Animal. Puis nous définissons une classe Cheval qui va implémenter les deux interfaces précédentes.

Exercice 5 : **(2pts)**

Les classes suivantes sont placées dans différents répertoires : B et C sont dans un répertoire p1, A est dans un répertoire p3 qui est un sous-répertoire de p1 et D est dans un répertoire p2.

```
package p1.p3;
public class A{
protected int i;
int j;
void m(){System.out.println("Je suis un A");}
}
```

Question : Les classes compilent-elles et si non, quelles sont les erreurs? (Souligner et expliquer les erreurs)

Erreurs à la compilation. La méthode m() de B doit avoir une visibilité supérieure ou égale à celle de D.

Dans la classe D, il faut ajouter import p1.p3.A; en tête de fichier, ou nommer A avec son nom complet.

```
package p1;
class B extends p2.D{
private int i=3;
void m(){System.out.println("Je suis un B");}
}
```

```
package p1;
class C extends p1.p3.A{
static protected int i = 1;
}
```

```
package p2;
public class D extends A{
protected void m(){System.out.println("Je suis un D");}
}
```

(2pts)

Exercice 6 : **(3 pts)**

La suite de Fibonacci est définie par :

$$f_n = \begin{cases} 1 & si\ n = 0 \\ 1 & si\ n = 1 \\ f_{n-1} + f_{n-2} & si\ \ n \geq 2 \end{cases}$$

On vous demande de créer une classe pour calculer fn avec n un entier introduit par l'utilisateur et de valeur positive (>=2).

Votre classe doit intégrer un mécanisme de gestion des erreurs survenant pendant l'exécution.

Le programme ne doit s'arrêter que si une valeur correcte est introduite.

Pour la suite de Fibonacci, j'ai considéré une exception uniquement le cas où le nombre introduit est inférieur à 0, ceci est un corrigé-type, toute solution correcte est acceptable : le plus important de cet exercice est de savoir créer une exception prédéfinie et l'utiliser, introduire une valeur via le clavier et boucler le processus jusqu'à introduction d'une valeur correcte pour délivrer le résultat et quitter.

Toute d'abord, voilà comment créer une exception prédéfinie : **(1.5 pts)**

```
public class FiboException extends Exception {
public FiboException() {
System.out.println("Le nombre introduit doit être supérieure ou égale à 0");
} }
```

Et voilà maintenant le programme qui déclenche l'exception :

(1.5 pts)

```
import java.util.Scanner;
public class Fibo {
public static int fibo(int n) throws FiboException {
if (n < 0) throw new FiboException();
else if (n == 0 || n == 1)return 1;
    else return fibo(n - 1) + fibo(n - 2);
 }
 public static void main(String[] args) {
 boolean t=true;
 do {
 Scanner sc = new Scanner(System.in);
 System.out.println("Introduire un entier");
 int n = sc.nextInt();
 try {
    System.out.println("fibo(" + n + ") = " + fibo(n));
    break;
    }
 catch (Exception e) {
     System.out.println("Erreur de saisie, recommencer");
    }
}
 while (t=true);
}
 }
```

Exercice 7 : **(4 pts)**

Nous disposons d'une classe Etudiant (Matricule : String, Nom : String, Age : int, moyenne : double) dans le cadre de la gestion de scolarité. Il a été demandé de classer les étudiants pour choisir ceux qui pourront bénéficier d'un stage à l'étranger.

On veut donc classer ces étudiants par ordre décroissant de la moyenne et par ordre croissant de l'âge.

Donner le code java nécessaire pour réaliser cette tâche en utilisant les concepts orientés objet vus au cours. Expliquez

// On commence par définir la classe Etudiant :

```
import java.util.Comparator;
public class Etudiant {
String matricule;
String nom;
int age;
double moyenne;
 public Etudiant(String matricule,String nom, int age, double moyenne) {
       this.matricule = matricule;
       this.nom = nom;
       this.age = age;
       this.moyenne = moyenne;
   }
 @Override
   //cette méthode affiche les informations de l'employé
   public String toString() {
      return "[matricule=" + this.matricule + ", nom=" + this.nom + ", age=" + this.age + ", moyenne=" +
            this.moyenne + "]";
   }
  //  Comparator pour le tri des étudiants par âge
   public static Comparator<Etudiant> ComparatorAge = new Comparator<Etudiant>() {
     @Override
      public int compare(Etudiant e1, Etudiant e2) {
         return (e1.age - e2.age);
      }
   };

   //  Comparator pour le tri des étudiants par moyenne
   public static Comparator<Etudiant> ComparatorMoyenne = new Comparator<Etudiant>() {
   @Override
```

// la méthode comprare renvoie un int, on doit faire un transtypage vers le type int

```
   public int compare(Etudiant e1, Etudiant e2) {
         return ((int)e1.moyenne - (int)e2.moyenne);
      }
   };
}
```

// Maintenant, on définit la classe du tri :

```
import java.util.ArrayList;
import java.util.Collections;
public class Tri {
   public static void main(String[] args) {
  ArrayList<Etudiant> Etudiants = new ArrayList<Etudiant>();
   Etudiants.add(new Etudiant("123456","mohamed", 19, 15.60));
  Etudiants.add(new Etudiant("123457","katia", 20, 15.62));
  Etudiants.add(new Etudiant("123458","ali", 20, 16.00));
  Etudiants.add(new Etudiant("123459","faten", 19, 15.70));
  System.out.println("/////////// avant le tri ///////////");
  for(Etudiant e: Etudiants)
  System.out.println(e);
  System.out.println("/////////// après le tri ///////////");
```

```
// Maintenant, on applique les deux critères du tri, d'abord par moyenne, ensuite par âge
  Collections.sort(Etudiants, Etudiant.ComparatorMoyenne);
  Collections.sort(Etudiants, Etudiant.ComparatorAge);

  for(Etudiant e: Etudiants)
   System.out.println(e);
 }
 }
```