Université Paris Descartes,
UFR Mathématiques et Informatique
L2 Informatique

Examen POO
(1ere session)
Correction

Aucun document autorisé

Exercice 1
Répondez par vrai ou par Faux aux questions suivantes :
Q1 Une interface peut hériter d'une ou de plusieurs interfaces Vrai
Q2 Une classe abstraite est une classe qui contient au plus une méthode abstraite Faux
Q3 Les instructions contenue dans un bloc ***finally*** seront exécutées qu'il y ait ou non une exception Vrai
Q4. Une classe ne peut implémenter plusieurs interfaces Faux
Q5. Toute classe qui implémente une interface est sous type de cette interface Vrai
Q6. Le code ci-dessous affiche le texte « *Hello from Class1* » Vrai

```
public class Class1 {
        public Class1 () {
                System.out.println("Hello from class1");
        }
}

class Class2 extends Class1 {}

public class TestClass2 {
        public static void main (String[] args){
                Class2 c2 = new Class2();
        }
}
```

Q7. Une classe qui implémente une interface hérite automatiquement de toutes ses constantes Vrai
Q8. Implémenter l'interface *Serializable* revient à implémenter toutes les méthodes qu'elle contient Faux, L'interface serializable n'expose aucune méthode, implémenter cette interface consiste donc juste à déclarer cette implémentation.

Exercice 2 Corrigez les morceaux de code ci-dessous

```
public class  FormeGeometrique1 {
        double posX, posY;

        double surface();
        double perimetre();

        public deplacer(double x, double y) {
                posX=x;
                posY=y;
        }
        public afficher() {
                System.out.println("position : (" +posX+","+posY+")");
        }
}
```

```
public abstract class  FormeGeometrique1 {
	double posX, posY;

	abstract double surface();
	abstract double perimetre();

	public void deplacer(double x, double y) {
		posX=x;
		posY=y;
	}
	public void afficher() {
		System.out.println("position : (" +posX+","+posY+")");
	}
}
```

Une classe personnalisée, *RectangleException*, a été créée comme sous-classe de *Exception*.

```
class Rectangle {
	int longueur;
	int largeur;

	public Rectangle(int lo, int la) throw RectangleException {
		if (lo < 0 || la < 0)
			throws new RectangleException();
		else{
			longueur = lo;
			largeur = la;
		}
	}
}
```

```
class Rectangle {
	int longueur;
	int largeur;

	public Rectangle(int lo, int la) throws RectangleException {
		if (lo < 0 || la < 0)
			throw new RectangleException();
		else{
			longueur = lo;
			largeur = la;
		}
	}
}
```

Exercice 3

Expliquez (en précisant si c'est juste ou faux) le morceau de code ci-dessous

```
//Vecteur d'entiers
Public static void main(String[] args){
	Vector<Integer> v = new Vector<Integer>();
	int valeur = 0;
	int nbVal = Saisie.lireEntier("Donnez le nombre total de valeurs") ;
	for(int i=0 ; i< nbVal; i++){
		valeur = Saisie.lireEntier("Donnez une valeur ?");
		v.add(valeur);
	}
	System.out.println(v) ;
}
```

Liste des erreurs :
- Public s'écrit en minuscule (public).
- D'après le cours, pour Vector il faut utiliser addElement, cependant add fonctionne aussi

Fonctionnement du programme:
Un vecteur de type Vector est créé, l'utilisateur entre le nombre n d'entiers qu'il veut entrer dans ce vecteur, il entre n entiers, puis le programme fini en affichant le contenu du vecteur.

Exemple d'exécution :

```
Donnez le nombre total de valeurs2
Donnez une valeur ?3
Donnez une valeur ?4
[3, 4]
```

Exercice 4
Q1 : Que se passe t'il si on exécute le code ci-dessous en ligne de commande, de la manière suivante : « ***java Principale 0 5*** ».

Q2 : Corrigez le problème en proposant deux solutions distinctes.

```
public class Equation {
	private int a,b;

	public Equation(int a, int b){
		this.a=a; this.b=b;
	}

	public void afficher() {
		System.out.println(a+" * X = "+b);
	}

	public int solution() {
		return b/a;
	}
}
```

```
public class Principale {
	public static void main(String args[]){
		int a=Integer.valueOf(args[0]).intValue();
		int b=Integer.valueOf(args[1]).intValue();
		Equation equation = new Equation(a,b);
		equation.afficher();
		System.out.println("resultat : X = "+ equation.solution());
	}
}
```

Q1 : Une exception est levée et le programme s'arrete car il y a une division par 0 (java.lang.ArithmeticException)

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at testexam.Equation.solution(Equation.java:15)
        at testexam.Principale.main(Principale.java:9)
```

Q2 : Solutions au problème :

Solution 1, la plus simple mais n'utilisant pas les principes de la POO :

```
public int solution() {
        if(a == 0){
                System.out.println("ERREUR, DIVISION PAR 0");
                System.exit(-1);
                return(-1);
        } else return b/a;
}
```

Solution 2, en traitant l'exception :

```
public class Equation {
        private int a,b;

        public Equation(int a, int b){
                this.a=a; this.b=b;
        }

        public void afficher() {
                System.out.println(a+" * X = "+b);
        }

        public int solution() throws Exception {
                if(a == 0)
                        throw new Exception();
                else
                        return b/a;

        }
}
```

```
public class Principale {
        public static void main(String args[]){
                try{
                        int a=Integer.valueOf(0).intValue();
                        int b=Integer.valueOf(5).intValue();
                        Equation equation = new Equation(a,b);
                        equation.afficher();
                        System.out.println("resultat : X = "+
equation.solution());
                } catch(Exception e) {
                        System.out.println("Erreur: " + e.getMessage());
                }

        }
}
```

Exercice 5 Gestion d'une collection de meubles.

Complétez ou corrigez le squelette de code ci-dessous

```
public class Meuble {
        int annee;

        public Meuble(...){
                //TO DO
        }

        //Affichage des attributs
        public void afficher(){
                //TO DO
        }

        //Retourne la surface du meuble
                //TO DO
        public double surface()....
}
```

```
/*Une table est un meuble particulier*/
class Table {
        String couleur;

        public Table(....){
                //TO DO
        }

        //Affichage des attributs
        public void afficher(){
                //TO DO
        }

        //retourne la surface de la table
                //TO DO
        public double surface()...
}
```

```java
/*Une table ronde est une table particulère
 * qui possède un rayon */
class TableRonde {
	private double rayon;

	public TableRonde(...){
		//TO DO
	}

	//Affichage des attributs
	public void afficher(){
		//TO DO
	}

	//retourne la surface de la table
		//TO DO
	public double surface()...
}
```

```java
//On souhaite gérer une collection de meubles (tables, table rondes, ...). Expliquez en un mot, le principe de la POO qui est appliqué ici

public class CollectionMeuble {
	private List<Meuble> meubles;

	public CollectionMeuble(){
		//TO DO;
	}

	//Afficher la surface totale de tous les meubles
	//Attention!!! Il faut utiliser l'interface Iterator pour cette question

	public void afficherSurfaceTotale(){
		//TO DO
	}
}
```

```java
public abstract class Meuble {
	int annee;

	public Meuble(int annee) {
		this.annee = annee;
	}

	//Affichage des attributs
	public void afficher() {
		System.out.println("Année : "+annee);
	}

	//Retourne la surface du meuble
	public abstract double surface();
}
```

```java
/*Une table est un meuble particulier*/
public abstract class Table extends Meuble {
	String couleur;

	public Table(int annee, String couleur) {
		super(annee);
		this.couleur = couleur;
	}

	public void afficher() {
		super.afficher();
		System.out.println(" Couleur : "+couleur);
	}

	//retourne la surface de la table
	public abstract double surface();
}
```

```java
/*Une table ronde est une table particulière
 *qui possède un rayon */

public class TableRonde extends Table {
	private double rayon;

	public TableRonde(int annee, String couleur, double rayon) {
		super(annee, couleur);
		this.rayon = rayon;
	}

	//affichage des attributs
	public void afficher() {
		super.afficher();
		System.out.println(" Rayon : "+rayon);
	}

	public double surface() {
		return rayon*rayon*3.14;
	}
}
```

```java
import java.util.Iterator;
import java.util.List;
import java.util.Vector;

/*On souhaite gérer une collection de meubles (tables, tables rondes,...).
 *Expliquer en un mot le principe de POO qui est appliqué ici
 */

public class CollectionMeuble {
	private List<Meuble> meubles;

	public CollectionMeuble() {
		meubles = new Vector<Meuble>();
	}

	public void ajouterMeuble(Meuble meuble){
		meubles.add(meuble);
	}

	//Afficher la surface de tous les meubles
	//Attention !!! Il faut utiliser l'interface Iterator pour cette question

	public void afficherSurfaceTotale() {
		double sum = 0;
		Iterator<Meuble> it = meubles.iterator();
		do {
			sum += it.next().surface();
		}
		while (it.hasNext());
		System.out.println(sum);
	}
}
```

Le principe qui aura été appliqué tout au long de l'exercice est l'Héritage