Université de Paris
UFR Mathématiques et Informatique
L2 Informatique et Mathématiques

POO
(Examen)

Aucun document autorisé

Exercice 1 6 pts

Q1 La classe Table 3.5 pts

```
//0.5 pour la déclaration abstract
public abstract class Table {
        private String couleur;
        private int annee;

        //0.5 pts
   public Table(String couleur, int annee){
      this.couleur = couleur;
      this.annee = annee;
   }
  //Affichage du type de table, des caractéristiques de la table et de sa surface
   //
   //1pt (0.5 pt si non utilistion de getType() pour le polymorphisme)
   public void afficher(){
     System.out.println (getType () + " de couleur : " + couleur + " fabriquée en : " + annee
                    + " . Sa surface est de : " + surface() + " m2 ");
   }
  // 1 pt
  //retourne la surface de la table
   public abstract double surface();

   //0.5 pt
   //Retourne la chaine "Table "
   public String getType() {
        return "Table ";
   }
}
```

Q2 La classe TableRonde 2.5 pts

```
public class TableRonde extends Table{
   private double diametre;

   // 1 pt (héritage et constructeur)
   public TableRonde(String couleur, int annee, double diametre){
     super(couleur, annee);
     this.diametre = diametre;
   }
```

```
    //0.5 pt
     //Affichage du type de table, des caractéristiques de la table et de sa surface
    public void afficher(){
        super.afficher();
        System.out.println("Le diamètre de la table est de : " + diametre + " m");
    }
    //0.5 pt
    //retourne la surface de la table
    public  double surface() {
            double rayon = diametre/2;
            return Math.PI * rayon * rayon;
    }
    //0.5 pt
    //Retourne la chaine : "Table ronde"
    public String getType() {
            return "Table Ronde";
    }
}
```

Exercice 2 **6.5 pts**

Q1 La classe EpargneException

```
//1.5 pts
public class EpargneException extends Exception{

                //constructeur
                public EpargneException(String message){
                        super("Erreur dans la classe Epargne : " + message);
                }
}
```

Q2 La classe Epargne

Q2.1 0.5 pts

Ses attributs doivent être déclarés avec le mot clé ***static*** car il s'agit d'attributs de classe

Q2.2 2.5 pts

```
public class Epargne {
        // le plafond et le taux sont deux variables qui ont exactement les mêmes
        // valeurs pour toutes les instances de la classe Epargne
        // ces valeurs sont pour l'instant fixées respectivement à 30000 euros et 0.1
        // Attention ce ne sont pas des données constantes.
        private static double plafond;
        private static double tauxInteret;
        private String titulaire;
        private double solde;

        //Le dépot initial pour ce compte doit être positif et ne doit pas dépasser le plafond
        //1 pts
        public Epargne(String titulaire, double depotInitial)throws EpargneException {
                this.titulaire=titulaire;
```

```
                this.plafond=30000.0;
                this.tauxInteret=0.03;
                //la prise en compte de l'intérêt reste optionnel pour la correction
                // La solution proposée ici ne correspond pas à la réalité
                this.solde =depotInitial* (1 + tauxInteret) ;
                if(depotInitial < 0 || depotInitial > plafond) {
                        throw new EpargneException ("Dans le constructeur : la valeur du dépot
initial  "
                + depotInitial + " est incorrecte, elle doit être positive et ne doit pas dépasser le
plafond "
                + plafond);
                }
        }

        //On ne doit pas dépasser le plafond
        //0.5 pts
        public void crediter(double montant) throws EpargneException {
                if(montant + solde > plafond) {
                        throw new EpargneException ("Dans crediter :  le plafond de " + plafond
                                        + " ne peut être dépassé");
                }
                solde+=montant * (1 + tauxInteret); //ne correspond pas à la réalité
        }
        //On ne peut pas retirer plus d'argent que le solde du compte.
        //0.5 pts
        public void debiter(double montant) throws EpargneException {
                if(montant > solde) {
                        throw new EpargneException ("Dans debiter :  on ne peut pas retirer un
montant supérieur au "
                + solde);
                }
                solde-=montant;
        }
        //0.5 pts
        public void afficher() {
                System.out.println("Titulaire : " + titulaire);
                System.out.println("Solde : " + solde);
        }
}
```

Q3. La classe TestEpargne 2 pts

```
//1 pts pour Scanner
// 1 pts pour le try catch
import java.util.Scanner;
public class TestEpargne {

        public static void main(String[] args) {
                // TODO Auto-generated method stub
                Scanner sc = new Scanner(System.in);
                System.out.println("Donnez le nom du titulaire du compte ?");
                String titulaire = sc.next();
```

```
                System.out.println("Donnez la valeur du dépot intial du compte ?");
                double depotInitial = sc.nextDouble();

                try {
                        Epargne ep = new Epargne (titulaire, depotInitial);
                        ep.crediter(5000);
                        ep.afficher();
                        ep.debiter(4000);
                        ep.afficher();
                        //ep.debiter(6000);
                }
                catch (EpargneException e){
                        System.out.println("Erreur d'exécution dans le main : " + e.getMessage());
                }
        }
}
```

Exercice 3 7.5 pts

Q1. L'interface Comparable

```
// 1 pts
public interface Comparable {
        //L'objet courant est plus grand que l'objet passé en paramètre
        boolean isPlusGrand(Object o);
}
```

Q2 La classe Employe 3 pts

```
public class Employe implements Comparable{
        private String nom;
        private double salaire;

        // 0.5 pts
        public Employe(String nom, double salaire) {
                this.nom=nom;
                this.salaire = salaire;
        }
        //0.5 pts
        public String toString() {
                return "Nom : " + nom  + "  Salaire : " + salaire;
        }
        //0.5 pts
        public void afficher() {
                System.out.println(this);
        }
        //Un employé courant est plus grand que l'employé passé en argument si son nom est
//plus grand selon l'ordre alphabétique
        // 1.5 pts
        public boolean isPlusGrand(Object o) {
                return o instanceof Employe &&
(this.nom.compareToIgnoreCase(((Employe)o).nom))> 0;
        }}
```

Q3. La classe Personnel 3.5 pts

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

public class Personnel {
        private List<Employe> employes;
        // 0.5 pts
  public Personnel() {
    employes = new ArrayList<Employe>();
  }

  // 0.5 pts
  //Ajouter un employe s'il n'existe pas
  public void ajouter(Employe e) {
    if(!employes.contains(e)){
     employes.add(e);
    }
  }

  // 1.5 pts
  //Algorithme du tri à bulle
  public void trier() {
        boolean permute = true;
                while(permute){
                        permute=false;
                  for (int i=0;i<employes.size()-1; i++){
                    if (employes.get(i).isPlusGrand(employes.get(i+1))){
                      Employe temp = employes.get(i);
                      employes.set(i,employes.get(i+1));
                      employes.set(i+1, temp);
                      permute = true;
                    }
                   }//fin for
                 }//fin while

  }
  //Vous devez utiliser l'interface Iterator pour parcourir la liste
  // 1 pts (enlever 0.5 pts si Iterator n'est pas utilisé)
  public void afficher() {
        System.out.println("Affichage de la liste des employés");
        Iterator<Employe> it = employes.iterator();
        while(it.hasNext()) {
                it.next().afficher();
        }
        System.out.println("******************");
  }
}
```