Examen Java SMI S6

Mai 2019

Prof. Abdessamad Belangour

Problème :

Nous souhaitons écrire une petite application en Java qui gère l'association des parents pour un lycée. Un parent peut être un père, une mère ou un tuteur. Tout parent dont l'enfant est inscrit dans ce lycée fait partie automatiquement de l'association. Ces parents sont représentés par un petit groupe de parents qui sont élus et qui sont appelés délégués. Ainsi, on ne peut être délégué si on n'est pas parent. Les différentes classes de l'application sont comme suit :

```
public enum TypeParent {  pere, mere, tuteur }
```

```
public class Eleve implements Comparable{
   private String id;
   private String nom;
   private String prenom;
   private Parent parent;
  public Eleve(String id, String nom, String prenom, Parent parent) { }
  public Eleve(String id, String nom, String prenom) { }
  // on suppose que les getters & setters sont fournis
   @Override
   public int compareTo(Object o) {……}
   }
```

```
public class Parent {

   private String cin;
   private String nom;
   private String prénom;
   private TypeParent type;
   private Set<Eleve> enfants;

   public Parent(String cin, String nom, String prénom, TypeParent type) {
      this.cin = cin;
      this.nom = nom;
      this.prénom = prénom;
      this.type = type;
      this.enfants = ……;
   }
  // on suppose que les getters & setters sont fournis
```

```
public void ajouterEnfant(Eleve eleve) { }
public void ajouterEnfant(String id, String nom, String prenom) { ……. }
public void ajouterEnfants(Set<Eleve> enfants) {}
public Eleve chercherEnfant(String id) {……..}
public boolean supprimerEnfant(String id) {……..}
@Override
   public String toString() {}
```

```
public class CompNbEnfants implements Comparator {
   @Override
   public int compare(Object o1, Object o2) { …….. }
}
```

```
public class ParentInexistantException extends Exception {  }
```

```
public class Lycee {

   private String nom;
   private Set<Parent> parents;
   private Map<String, Parent> délégués;

   public Lycee(String nom) {
      this.nom = nom;
      this.parents =……..;
      this.délégués = ……..;
   }

 // on suppose que les getters & setters sont fournis

   public void ajouterParent(Parent parent) { }
   public Parent chercherParent(String CIN) { ……  }
   public void ajouterDélégué(String CIN) throws ParentInexistantException { …….}
   public void ajouterDélégué(Parent p) throws ParentInexistantException {…… }
   public void ajouterDélégués(String[] tabCIN) throws ParentInexistantException { }
   public void afficherPourcentageTypes() { …. }
   @Override
   public String toString() { }

}
```

```
public class Main {

   public static void main(String[] args) {
      Lycee lycee=new Lycee("Annajah");
      //-------------------------- Parent 1 -------------------------//
      Parent p1=new Parent("X1001", "Alaoui", "Ali", TypeParent.pere);
      lycee.ajouterParent(p1);
      p1.ajouterEnfant("2019/1", "Alaoui", "Adil");
      //-------------------------- Parent 2 -------------------------//
      Parent p2=new Parent("B5003", "Omari", "Salwa", TypeParent.mere);
      lycee.ajouterParent(p2);
      p2.ajouterEnfant("2019/2", "Yousfi", "Hassan");
      p2.ajouterEnfant("2019/3", "Yousfi", "Houssine");
      //-------------------------- Parent 3 -------------------------//
      Parent p3=new Parent("ZR1003", "Zakani", "Yasser", TypeParent.pere);
      lycee.ajouterParent(p3);
      p3.ajouterEnfant("2019/4", "Tahiri", "Samira");
      //-------------------------- Parent 4 -------------------------//
      Parent p4=new Parent("AB1509", "Othmani", "Othman", TypeParent.tuteur);
      lycee.ajouterParent(p4);
      p4.ajouterEnfant("2019/5", "Nassiri", "Youssef");
      p4.ajouterEnfant("2019/6", "Nassiri", "Sara");
      p4.ajouterEnfant("2019/7", "Nassiri", "Ayoub");

         ……..
         ……..
         ……..
         ……..
         ……..
}
}
```

Questions :

1) Fournir le code de la méthode **compareTo** de la classe Eleve permettant de comparer les objets Eleve par ordre alphabétique des noms.
2) Fournir le code d'initialisation de l'attribut **enfants** dans le constructeur de la classe Parent.
3) Fournir le code de la méthode **ajouterEnfant** de la classe Parent.
4) Fournir le code de la méthode **chercherEnfant** de la classe Parent.
5) Fournir le code de la méthode **supprimerEnfant** de la classe Parent.
6) Fournir le code de la méthode **compare** de la classe CompNbEnfants permettant de comparer des objets Parents par nombre d'enfants.
7) Fournir le code de l'initialisation des attributs **parents** et **délégués** dans le constructeur de la classe Lycee, sachant que les parents doivent être ordonnés par nombre d'enfants.
8) Fournir le code de la méthode **chercherParent** de la classe Lycee.
9) Fournir le code de la méthode **ajouterDélégué(String CIN)** de la classe Lycee.
10) Fournir le code de la méthode **ajouterDélégué(Parent p)** de la classe Lycee.
11) Fournir le code de la méthode **afficherPourcentageTypes** de la classe Lycee qui affiche le pourcentage des pères, le pourcentage des mères et le pourcentage des tuteurs parmi les délégués.
12) Terminer la méthode **main()** de la classe Main par ajout des objets p1, p2, p4 comme délégués et afficher l'objet lycee et le pourcetage des types de parents.

N. B. :

- N'écrivez que la partie demandée sur la feuille de l'examen.
- Respectez l'ordre des questions dans la feuille.
- N'oubliez pas de mettre le numéro de la question devant vos réponses.
- Ecrivez avec une écriture claire et lisible.
- Toute tentative de triche découverte dans la correction sera sanctionnée par un zéro.

Bon courage !

```java
package Serie5;


public class Eleve implements Comparable{
	private String id;
	private String nom;
	private String prenom;
	private Parent parent;
	public Eleve(String id, String nom, String prenom, Parent
parent) {

		this.id=id;
		    this.nom=nom;
		    this.prenom=prenom;
		    this.parent=parent;

	 }
	public Eleve(String id, String nom, String prenom) {
		this.id=id;
		    this.nom=nom;
		    this.prenom=prenom;


	 }
	 // on suppose que les getters & setters sont fournis

	// @Override
	 public int compareTo(Object o)
	 {
		 Eleve e=(Eleve)o;
		 return this.prenom.compareTo(e.prenom);
	 }
	public String getId() {
		return id;
	}
	public void setId(String id) {
		this.id = id;
	}
	public String getNom() {
		return nom;
	}
	public void setNom(String nom) {
		this.nom = nom;
	}
	public String getPrenom() {
		return prenom;
	}
```

```
	public void setPrenom(String prenom) {
		this.prenom = prenom;
	}
	public Parent getParent() {
		return parent;
	}
	public void setParent(Parent parent) {
		this.parent = parent;
	}
	@Override
	public String toString() {
		return "Eleve [id=" + id + ", nom=" + nom + ", prenom="
+ prenom  + "]";
	}


	 }

package Serie5;

import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;
import java.util.TreeSet;

public class Parent{
	private String cin;
	private String nom;
	private String prénom;
	private TypeParent type;
	private Set<Eleve> enfants;
	public Parent(String cin, String nom, String prénom,
TypeParent type) {
	this.cin = cin;
	this.nom = nom;
	this.prénom = prénom;
	this.type = type;
	this.enfants = new TreeSet<> ();
	}
	// on suppose que les getters & setters sont fournis

	public void ajouterEnfant(Eleve eleve) {

		enfants.add(eleve);
	}
```

```
     public String getCin() {
          return cin;
     }

     public void setCin(String cin) {
          this.cin = cin;
     }

     public String getNom() {
          return nom;
     }

     public void setNom(String nom) {
          this.nom = nom;
     }

     public String getPrénom() {
          return prénom;
     }

     public void setPrénom(String prénom) {
          this.prénom = prénom;
     }

     public TypeParent getType() {
          return type;
     }

     public void setType(TypeParent type) {
          this.type = type;
     }

     public Set<Eleve> getEnfants() {
          return enfants;
     }

     public void setEnfants(Set<Eleve> enfants) {
          this.enfants = enfants;
     }

     public void ajouterEnfant(String id, String nom, String
prenom) {

     enfants.add(new Eleve(id, nom, prenom, this));
     }
      public void ajouterEnfants(Set<Eleve> enfants) {
```

```
            this.enfants.addAll(enfants);

       }
       public Eleve chercherEnfant(String id) {


            Eleve e = null;
            Iterator it = this.enfants.iterator();
            boolean trouvé = false;
            while (!trouvé && it.hasNext()) {
            Eleve elv = (Eleve) it.next();
            if (elv.getId().equals(id)) { e = elv; trouvé = true;
}
            }
            return e;
            }



       public boolean supprimerEnfant(String id) {
            Eleve eleve = chercherEnfant(id);
       boolean fait = false;
       if (eleve != null) { enfants.remove(eleve); fait = true; }
       return fait;
       }
       @Override
      public String toString() {
            String eleves="mes enfants sont: ";

            for (Eleve e: enfants)
            {

                 eleves +="\n\t\t"+ e.toString();
            }

           return "Parent [cin=" + cin + ", nom=" + nom + ",
prénom=" + prénom + ", type=" + type + ", \n\tenfants=" + eleves
                     + "]";

      }




}

package Serie5;
```

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import java.util.TreeSet;

public class Lycee {
	private String nom;
	private TreeSet<Parent> parents;
	private Map<String, Parent> délégués;
	public Lycee(String nom) {
	this.nom = nom;
	//this.parents = new TreeSet<>(new CompNbEnfants());
	this.parents = new TreeSet<Parent>(new CompNbEnfants());
	//this.parents = new TreeSet<Parent>();
	this.délégués = new HashMap<String, Parent>();
	}

	// on suppose que les getters & setters sont fournis
	public void ajouterParent(Parent parent) {
		//parents.add(parent);
		System.out.println("Parent ajouté : " +
parent.getNom() + " - retour insertion :" + parents.add(parent));
	}
	public Parent chercherParent(String CIN) {
		Parent parent = null;
		Iterator<Parent> it = this.parents.iterator();
		boolean trouvé = false;
		while (!trouvé && it.hasNext()) {
		Parent pr = it.next();
		if (pr.getCin().equals(CIN)) { parent = pr; trouvé =
true; }
		}
		return parent;
	}
	public void ajouterDélégué(String CIN) throws
ParentInexistantException {

	Parent p = this.chercherParent(CIN);
	if (p == null) throw new ParentInexistantException("CIN
Parent non existant");
	this.délégués.put(CIN, p);
	}
	public void ajouterDélégué(Parent p) throws
ParentInexistantException {
```

```
            if (!this.parents.contains(p))
                  throw new ParentInexistantException("Ce n'est pas
un parent");
                   this.délégués.put(p.getCin(), p);

      }
      public void ajouterDélégués(String[] tabCIN) throws
ParentInexistantException { }
      public void afficherPourcentageTypes() {
            int total = this.parents.size();
            int nbMeres = 0, nbPeres = 0, nbTuteurs = 0;
            for (Parent p : parents) {
            if (p.getType() == TypeParent.mere) { nbMeres++; }
            else if (p.getType() == TypeParent.pere) { nbPeres++;
}
            else { nbTuteurs++; }
            }
            System.out.println("Pourcentage Meres " + (nbMeres /
(float)total) * 100 + "%"
            + " Pourcentage Peres " + (nbPeres / (float)total) *
100 + "%"
            + " Pourcentage Tuteurs " + (nbTuteurs / (float)total)
* 100 + "%");
      }


     public String toString() {
            String parent="\nles parents sont: ";



            for (Parent p: parents)
            {
                  parent += "\n\t" + p.toString();
            }
            String delegués="\nles délegués sont: ";

            for (Map.Entry<String,Parent> d:délégués.entrySet())
            {
                  delegués += "\n\t" + d.toString();
            }
           return "Lycee [nom=" + nom + ", \nparents=" + parent +
"\n" + ", \ndélégués=" + "\n" + délégués + "]";
     }

     public TreeSet<Parent> getParents() {
```

```
		return parents;
	}

	public void setParents(TreeSet<Parent> parents) {
		this.parents = parents;
	}

	}
package Serie5;

public class ParentInexistantException extends Exception {

	public ParentInexistantException(String message) {
	super (message);
}
}

package Serie5;

import java.util.Comparator;

public class CompNbEnfants implements Comparator <Parent> {

	public int compare(Parent p1, Parent p2) {

		System.out.println("Inside Comparator - p1 = " +
p1.getNom() + ":" + p1.getEnfants().size() +" - p2 = " +
p2.getNom() + ":" + p2.getEnfants().size());
		if (p1.getEnfants().size() == p2.getEnfants().size())
			return p1.getCin().compareTo(p2.getCin());

		 return (p1.getEnfants().size()-
p2.getEnfants().size());



	}
}



	package Serie5;

public enum TypeParent {
	pere, mere, tuteur
}
```

```java
package Serie5;

public class Test {
     public static void main(String[] args) {

            Lycee lycee=new Lycee("Annajah");
            //--------------------------- Parent 1 --------------
----------//
            Parent p1=new Parent("X1001", "Alaoui", "Ali",
TypeParent.pere);
            p1.ajouterEnfant("2019/1", "Alaoui", "Adil");

            lycee.ajouterParent(p1);
            //--------------------------- Parent 2 --------------
----------//
            Parent p2=new Parent("B5003", "Omari", "Salwa",
TypeParent.mere);
            p2.ajouterEnfant("2019/2", "Yousfi", "Hassan");
            p2.ajouterEnfant("2019/3", "Yousfi", "Houssine");
            lycee.ajouterParent(p2);

            //--------------------------- Parent 5 --------------
----------//
            Parent p5=new Parent("AB1509", "Othmani", "Othman",
TypeParent.tuteur);
            p5.ajouterEnfant("2019/5", "Nassiri", "Youssef");
            p5.ajouterEnfant("2019/6", "Nassiri", "Sara");
            p5.ajouterEnfant("2019/7", "Nassiri", "Ayoub");
            lycee.ajouterParent(p5);
           //--------------------------- Parent 3 ---------------
----------//
            Parent p3=new Parent("ZR1003", "Zakani", "Yasser",
TypeParent.pere);
            p3.ajouterEnfant("2019/4", "Tahiri", "Samira");
            lycee.ajouterParent(p3);
           //--------------------------- Parent 4 ---------------
----------//
            Parent p4=new Parent("LM258974", "bennani", "Othman",
TypeParent.tuteur);
            p4.ajouterEnfant("2019/5", "benali", "Youssef");
            lycee.ajouterParent(p4);
           //--------------------------- Parent 6 ---------------
----------//
            Parent p6=new Parent("BX123", "Radi", "mourad",
TypeParent.pere);
            p6.ajouterEnfant("2019/5", "Radi", "Youssef1");
```

```
            p6.ajouterEnfant("2019/6", "Radi", "Youssef2");
            p6.ajouterEnfant("2019/7", "Radi", "Youssef3");
            p6.ajouterEnfant("2019/8", "Radi", "Youssef4");
            lycee.ajouterParent(p6);

           try { lycee.ajouterDélégué("BJ292");
            lycee.ajouterDélégué(p2);
            lycee.ajouterDélégué(p3);
            }
           catch (ParentInexistantException ex) {
            System.out.println(ex.getMessage());
            }

           System.out.println(lycee);


     System.out.println("=====================================")
;

           System.out.println(lycee.getParents().contains(p1) + "
- " + p1.getNom());
           System.out.println(lycee.getParents().contains(p2) + "
- " + p2.getNom());
           System.out.println(lycee.getParents().contains(p3) + "
- " + p3.getNom());
           System.out.println(lycee.getParents().contains(p4) + "
- " + p4.getNom());
           System.out.println(lycee.getParents().contains(p5) + "
- " + p5.getNom());
           System.out.println(lycee.getParents().contains(p6) + "
- " + p6.getNom());
            //lycee.afficherPourcentageTypes();

           }
           }
```