1 Interfaces

Question 1 *Extraction d'une interface. On suppose connue une classe* `Personne` *et ci-dessous on vous donne le code d'une classe représentant des files d'attente de personnes. Proposez une interface décrivant le type de cette classe.*

```
public class FileAttente
{
     private String nomFile;
     private static int nbPersonnesEntreesTotal = 0;
     private ArrayList<Personne> contenu;
     public FileAttente(){contenu=new ArrayList<Personne>();}
     public void entre(Personne p){contenu.add(p); nbPersonnesEntreesTotal++;}
     public Personne sort()
          {
               Personne p=null;
               if (!contenu.isEmpty())
                    {p=contenu.get(contenu.size()-1);
                    contenu.remove(contenu.size()-1);}
               return p;
          }
     public int nbElements(){return contenu.size();}
     public boolean estVide(){return contenu.isEmpty();}
     public String toString(){return ""+descriptionContenu();}
     private String descriptionContenu()
     {
          String resultat = "";
          for (Personne p:this.contenu)
               resultat += p + " ";
          return resultat;
     }
}
```

Réponse 1 *On peut donner différentes interfaces, le plus important est d'extraire dans l'interface ce que l'on veut donner aux utilisateurs extérieurs de la file d'attente. Une solution peut être la suivante. On a éliminé les attributs (sauf s'il y en avait eu des "public static final"), les méthodes statiques et tout ce qui est privé en général. Comme ce corrigé vous est donné après le cours sur la généricité, on a ajouté un paramètre de généricité.*

```
public interface IFileAttente<E> {
     void entre(E p);
     E sort();
     boolean estVide();
     String toString();
     int nbElements();
}
```

Question 2 *Comment modifiez-vous la classe* `FileAttente` *pour qu'elle implémente l'interface que vous avez créée ?*

Réponse 2 *On modifie simplement son entête.*

```
public class FileAttente<E> implements IFileAttente<E>
```

QUESTION 3 *Proposez un type (une interface) file d'attente avec des statistiques. Deux opérations supplémentaires permettent de connaître respectivement la taille de la liste et le nombre total d'opérations réalisées (en cumulant les entrées et les sorties). Puis proposez une classe implémentant cette interface. Si le nombre total est cumulé pour une file,* `nbTotalOp` *est mis comme attribut d'instance (comme ci-dessous). Si on veut le cumuler pour toutes les files, on en fait un attribut static.*

RÉPONSE 3 *On commence par définir une nouvelle interface.*

```
public interface IFileAttenteAvecStatistiques<E> extends IFileAttente<E>{
     int tailleFile();
     int nbTotalOperations();
}
```

Puis on peut en donner une implémentation (l'énoncé ne le demande pas explicitement puisqu'il demande juste un type).

```
public class FileAttenteAvecStatistiques<E> extends FileAttente<E>
implements IFileAttenteAvecStatistiques<E>
{
private int nbTotalOp = 0;

public FileAttenteAvecStatistiques() {}

@Override
public int tailleFile() {
return this.nbElements();
}

@Override
public int nbTotalOperations() {
return this.nbTotalOp;
}

public void entre(E p)
{super.entre(p); nbTotalOp++;}

public E sort()
{
nbTotalOp++;
return super.sort();
}
}
```

QUESTION 4 *Proposez une autre implémentation* `Rectangle_tab` *de l'interface* `Irectangle` *dans laquelle on stocke la hauteur et la largeur dans un tableau de* `float` *de taille 2. Voyez-vous des parties communes dans cette implémentation qui pourraient suggérer une super-classe des deux implémentations ? Si oui, écrivez-la.*

Analyser le programme suivant avec les classes que vous avez créées.

```
    public static void main(String[] arg)
    {
        Irectangle r1, r2, r3;
        r1 = new Irectangle();
        r1 = new Rectangle();
        r2 = new Rectangle_2float(2,4);
        r3 = new Rectangle_tab(3,5);
        System.out.println(r1+"\n"+r2+ "\n"+ r3);
        System.out.println(r1.perimetre()+"\n"+r2.perimetre()+ "\n"+ r3.perimetre());
        System.out.println(r1.largeur()+"\n"+r2.largeur()+ "\n"+ r3.largeur());
    }
```

RÉPONSE 4 *On rappelle les interfaces du cours, IRectangle et IQuadrilatère.*

```
public interface Iquadrilatere {
   public static final int nbCotes = 4;
   public abstract double perimetre();
   public abstract double surface();
}

public interface Irectangle extends Iquadrilatere{
    int angle = 90;
    double getLargeur();
    void setLargeur(double l);
    double getHauteur();
    void setHauteur(double h);

    default double perimetre()
    {return 2*this.getLargeur()+2*this.getHauteur();}

    default double surface()
    {return this.getLargeur()*this.getHauteur();}

    default String description() // on ne peut pas utiliser le nom "toString"
    {return "largeur ="+this.getLargeur()+" hauteur ="+this.getHauteur();}

    static boolean egal(Irectangle r1, Irectangle r2)
    {
        return r1.getLargeur()==r2.getLargeur()
            && r1.getHauteur()==r2.getHauteur();
    }
}
```

Pour représenter le comportement commun à toutes les implémentations de rectangle, on introduit parfois une classe abstraite. Cela évite de dupliquer du code entre les différentes implémentations. Ce n'est nécessaire ici que pour la méthode ***`toString()`*** *qui n'a pas pu être introduite comme une méthode par défaut dans l'interface ou pour une méthode retournant l'angle. C'est un procédé très classique que l'on va retrouver dans l'implémentation des classes représentant les collections en Java.*

```
public abstract class Rectangle implements Irectangle{
    public float angle(){return Irectangle.angle;}
```

```
    public String toString(){return "rectangle "+description();}

    public static void main(String[] arg)
    {
     // on pourra déclarer des variables de type Irectangle
     Irectangle r1, r2, r3;
     //r1 = new Rectangle(); Impossible car la classe est abstraite
     r2 = new RectangleAT(2,4);
     r3 = new Rectangle_tab(3,5);
     System.out.println(r2+ "\n"+ r3);
    }
}

 public class RectangleAT implements Irectangle {
 private double largeur, hauteur;
 public RectangleAT() {}
 public RectangleAT(double largeur, double hauteur) {this.largeur = largeur; this.haute
 public double getLargeur() {return this.largeur;}
 public void setLargeur(double l) {this.largeur = l;}
 public double getHauteur() {return this.hauteur;}
 public void setHauteur(double h) {this.hauteur = h;}
}

 public class Rectangle_tab extends Rectangle {
 private double[] tab = new double[2];
 public Rectangle_tab(){}
 public Rectangle_tab(double l, double h){tab[0]=l;tab[1]=h;}
 public double getLargeur(){return tab[0];}
 public void setLargeur(double l) {tab[0]= l;}
 public double getHauteur(){return tab[1];}
 public void setHauteur(double h) {tab[1] = h;}
}
```

QUESTION 5 *Ecrivez une classe pour représenter des stocks de rectangles et une méthode retournant la somme de leurs périmètres. Qu'en déduisez-vous sur l'un des intérêts des interfaces ?*

RÉPONSE 5 *L'intérêt des interfaces est de pouvoir écrire un code très général, sans savoir comment les rectangles seront implémentés (avec un paire d'attributs réels ou avec un tableau de 2 réels). On le voit dans le code ci-dessous. La somme des périmètres est écrite simplement grâce à la connaissance de l'existence de la méthode* `perimetre` *dans l'interface Irectangle.*

```
public class StockRectangles {

private ArrayList<Irectangle> stock = new ArrayList<Irectangle>();

// ....

public double sommePerimetres()
```

```
{
double somme = 0;
for (Irectangle r : this.stock)
somme += r.perimetre();
return somme;
}
}
```

2 Listes

Le code ci-dessous est un extrait très simplifié de l'interface décrivant les listes.

```
public interface List extends Collection
{
    boolean add(Object element);
    boolean contains(Object o);
    Object get(int index);
    int size();
}
```

L'implémentation basée sur les tableaux consiste à stocker les éléments dans un tableau de plus en plus grand au fur et à mesure des besoins. On mémorise dans un attribut `nbrElements` le nombre d'éléments insérés. Le tableau est créé d'une certaine taille (`tailleInitiale`) et les cases du tableau sont remplies entre 0 et `nbrElements`-1. On ne retire pas d'élément dans cette première version. On décide que lorsque le tableau est plein, on l'agrandit d'une quantité stockée dans un attribut `incrementTaille`. Testez vos méthodes au fur et à mesure.

QUESTION 6 *Proposez une implémentation de liste basée sur un tableau.*

RÉPONSE 6 *Dans l'implémentation proposée ici, vous noterez que* ***`ListeTableau`*** *dérive réellement de l'interface* ***`List`*** *mais sans paramètre de généricité.* ***`ListeTableau`*** *hérite de* ***`AbstractList`*** *qui contient des redéfinitions par défaut de beaucoup de méthodes (un peu comme la classe Rectangle abstraite de l'exercice précédent).*

```
package genericiteNonBornee;

import java.util.*;

public class ListeTableau extends AbstractList implements List {
    private int nbrElements=0;
    private int tailleInitiale=10;
    private int incrementTaille=10;
    private Object[] contenu;

    public ListeTableau(){
        this.contenu = new Object[this.tailleInitiale];}

    public ListeTableau(int tailleInitiale){
```

```
    this.tailleInitiale=tailleInitiale;
    this.contenu = new Object[this.tailleInitiale];}

@Override
public boolean add(Object arg0) {
    if (this.nbrElements==contenu.length)
        {  // garder une référence vers le contenu actuel
            Object[] actuel = this.contenu;
            //agrandir le tableau
            this.contenu = new Object[this.contenu.length+this.incrementTaille];
            //recopier les valeurs précédentes
            for (int i=0; i<actuel.length; i++)
                {this.contenu[i] = actuel[i];}
            System.out.println("copie "+this);
        }
        // maintenant il y a de la place
        this.contenu[this.nbrElements]=arg0;
        nbrElements++;
        return true;
}

@Override
public boolean contains(Object arg0) {
    for (int i=0; i<this.nbrElements; i++)
        if (this.contenu[i].equals(arg0))
            return true;
        return false;
}

@Override
public Object get(int arg0) {
    if (this.nbrElements==0)
        return null;
    if (arg0 >=0 && arg0 <this.nbrElements)
        return this.contenu[arg0];
    return null;
    }

    @Override
    public boolean isEmpty() {
        return this.nbrElements==0;
    }

    @Override
    public int size() {
        return this.nbrElements;
    }

    public String toString()
```

```
        {
            String s="[";
            for (int i=0; i<this.nbrElements; i++)
                s+=this.contenu[i].toString()+", ";
            return s+"]";
        }

        public static void main(String[] a)
        {
            ListeTableau t1 = new ListeTableau(3);
            System.out.println(t1);
            t1.add("le");
            System.out.println(t1);
            t1.add("chat");
            System.out.println(t1);
            t1.add("dort");
            System.out.println(t1);
            t1.add("au");
            System.out.println(t1);
            t1.add("soleil");
            System.out.println(t1);
            System.out.println(t1.contains("dort"));
            System.out.println(t1.contains("lune"));
        }
}
```

QUESTION 7 *Ecrivez une méthode* **`equals`** *qui vérifie l'égalité de contenu entre deux listes (attention au type du paramètre de* **`equals`** *de manière à bien profiter de la liaison dynamique).*

RÉPONSE 7 *On peut ne rien écrire et s'appuyer sur la méthode* **`equals`** *de la class* ***`AbstractList`*** *qui effectue ceci : Compares the specified object with this list for equality. Returns true if and only if the specified object is also a list, both lists have the same size, and all corresponding pairs of elements in the two lists are equal. (Two elements e1 and e2 are equal if (e1==null ? e2==null : e1.equals(e2)).) In other words, two lists are defined to be equal if they contain the same elements in the same order. Mais ici, on vous montre comment l'écrire (si on n'en héritait pas) pour que cela fasse un exercice d'algorithmique.*

```
@Override
public boolean equals(Object autreListe)
{
    System.out.println("methode equals redefinie dans ListeTableau");
    // si autreList n'est pas une liste, on retourne faux
    if (! (autreListe instanceof List)) return false;
    List autreL = (List) autreListe;
    // si les deux listes ne sont pas de la même taille, on retourne faux
    if (autreL.size() != this.size()) return false;
    // sinon on parcourt les deux listes, dont on sait qu'elles sont de même taille
    for(int i=0; i<this.nbrElements; i++)
    if(! (autreL.get(i).equals(this.get(i))))
```

```
        return false; // on s'arrête à la première paire différente
    // si on arrive à ce point, les deux listes sont identiques en contenu
    return true;
}
```

QUESTION 8 *Ecrivez une interface* `IListeAvecStats` *qui dispose de 4 méthodes retournant le nombre d'appels de chacune des méthodes* `add`*,* `contains`*,* `get`*,* `size` *effectués dans un programme. Ecrivez une sous-classe* `ListeTableauAvecStats`*qui implémente* `IListeAvecStats`*. Proposez une méthode* `equals` *qui vérifie l'égalité de contenu et de statistiques de deux listes avec statistiques (attention au type du paramètre de* `equals` *de manière à bien profiter de la liaison dynamique).*

RÉPONSE 8

```
public interface IListeAvecStats {
  int getNbAdd();
  int getNbContains();
  int getNbGet();
  int getNbSize();
}

public class ListeTableauAvecStats extends ListeTableau
          implements IListeAvecStats{
     private  int nbAdd=0;
     private  int nbContains=0;
     private  int nbGet=0;
     private  int nbSize=0;

     public ListeTableauAvecStats() {}
     public ListeTableauAvecStats(int tailleInitiale) {super(tailleInitiale);}
     @Override
     public int getNbAdd() {
          return this.nbAdd;
     }
     @Override
     public int getNbContains() {
          return this.nbContains;
     }
     @Override
     public int getNbGet() {
          return this.nbGet;
     }
     @Override
     public int getNbSize() {
          return this.nbSize;
     }

     @Override
     public boolean add(Object arg0)
```

```
    {this.nbAdd++; return super.add(arg0);}

    @Override
    public boolean contains(Object arg0)
    {this.nbContains++; return super.contains(arg0);}

    @Override
    public Object get(int arg0)
    {this.nbGet++; return super.get(arg0);}

    @Override
    public int size()
    {this.nbSize++; return super.size();}

    @Override
    public boolean equals(Object autreListe)
    {
        System.out.println("methode equals redefinie dans ListeTableauAvecStats");
        // Les deux listes doivent être avec statistiques
        if (! (autreListe instanceof IListeAvecStats)) return false;
        IListeAvecStats autreL = (IListeAvecStats)autreListe;
        boolean contenusEgaux = super.equals(autreListe);
        // les deux listes doivent être égales en contenus
        if (! contenusEgaux) return false;
        // les statistiques doivent être identiques
        if (this.nbAdd != autreL.getNbAdd()) return false;
        if (this.nbContains != autreL.getNbContains()) return false;
        if (this.nbGet != autreL.getNbGet()) return false;
        if (this.nbSize != autreL.getNbSize()) return false;
        return true;
    }

}
```

QUESTION 9 *Vérifiez le comportement des instructions suivantes et notamment quelle est la méthode* ***`equals`*** *appelée dans chaque cas.*

```
System.out.println("-----------------");
ListeTableau l1 = new ListeTableauAvecStats();
ListeTableau l2 = new ListeTableauAvecStats();
ListeTableauAvecStats l3 = new ListeTableauAvecStats();
ListeTableauAvecStats l4 = new ListeTableauAvecStats();
System.out.println("--------l1.equals(l2)---------");
System.out.println(l1.equals(l2));
System.out.println("--------l1.equals(l3)---------");
System.out.println(l1.equals(l3));
System.out.println("--------l3.equals(l1)---------");
System.out.println(l3.equals(l1));
System.out.println("--------l3.equals(l4)---------");
System.out.println(l3.equals(l4));
```

RÉPONSE 9 *Le programme affiche ceci à l'exécution.*

```
-----------------
--------l1.equals(l2)---------
methode equals redefinie dans ListeTableauAvecStats
methode equals redefinie dans ListeTableau
true
--------l1.equals(l3)---------
methode equals redefinie dans ListeTableauAvecStats
methode equals redefinie dans ListeTableau
false
--------l3.equals(l1)---------
methode equals redefinie dans ListeTableauAvecStats
methode equals redefinie dans ListeTableau
false
--------l3.equals(l4)---------
methode equals redefinie dans ListeTableauAvecStats
methode equals redefinie dans ListeTableau
false
```

En effet, `nbSize` *est modifié à chaque appel de* `equals` *au moment où on teste les tailles. Ce pourrait être amélioré en remplaçant*
`if (autreL.size() != this.size())`
par :
`if (((ListeTableau) autreL).nbrElements != this.nbrElements).`

Par ailleurs si on n'avait pas pris le soin de bien garder le paramètre de la méthode `equals` *de type* `Object`*, c'est-à-dire que l'on aurait utilisé la signature* `public boolean equals(Object autreListe)` *et si on l'avait spécialisée en mettant :*
`public boolean equals(List autreListe)` *dans* `ListeTableau`
`public boolean equals(IListeAvecStats autreListe)` *dans* `ListeTableauAvecStats`
On aurait obtenu ce résultat qui n'est pas correct dans les cas 2 et 3 (la quatrième égalité `System.out.println(l3.equals(l4));` *ne peut d'ailleurs même pas compiler) :*

```
-----------------
--------l1.equals(l2)---------
methode equals redefinie dans ListeTableau
true
--------l1.equals(l3)---------
methode equals redefinie dans ListeTableau
true
--------l3.equals(l1)---------
methode equals redefinie dans ListeTableau
true
```

Point de cours : quand on redéfinit une méthode "A m(B bb) throws X, Y" (redéfinir au sens "pour que la liaison dynamique s'applique") on a seulement le droit de :

- mettre un type de retour plus spécialisé, donc remplacer A par une sous-classe de A,
- enlever des exceptions signalées ou les spécialiser, par exemple enlever X et remplacer Y par une sous-classe de Y,
- alléger la visibilité.

On n'a pas le droit de :

— changer les types des paramètres, même pour les spécialiser, par exemple c'est interdit de remplacer B par une sous-classe de B.

Sinon, la liaison est seulement statique (et non dynamique) avec des effets secondaires tels que racontés dans la correction ci-dessus.