CHAPITRE 1

Implémentation des ensembles

Copiez dans votre workspace le fichier que vous trouverez sur moodle dans le dossier ensembles. Décompressez ce fichier et importez-le dans Eclipse
Le fichier **EnsembleInterface.java** est une « interface »Java que l'on vous demande d'implémenter. Toute instance d'une classe implémentant EnsembleInterface est « moralement »un sous-ensemble de l'Univers des Elt. Pour rappel, la classe Elt fournit les constructeurs

```
public Elt(int i)
 // construit un Elt de valeur i;
 // produit une IllegalArgumentException si i n'est pas dans l'Univers.

public Elt(Elt e)
 // constructeur par recopie.
 // lance une IllegalArgumentException si e est null.
```

et les méthodes :

```
public int val()
 // renvoie la valeur numerique du Elt courant

public String toString()
 // description d'un Elt par sa valeur numerique.

public boolean equals(Object o)
 // renvoie true si l'Elt courant est egal a o.

public boolean hashCode()
 // calcule et renvoie un hashCode associe a l'Elt courant.

public Elt succ()
 // renvoie le successeur du Elt courant;
 // le successeur de 1 est 2, celui de 2 est 3, ...
 // celui de MAXELT est 1.

public Elt pred()
 // renvoie le predecesseur du Elt courant ;
 // le predecesseur de 32 est 31, ..., celui de 1 est MAXELT.
```

L'interface EnsembleInterface expose les méthodes que vous allez devoir compléter dans les classes que vous allez récupérer :

```
public interface EnsembleInterface {

  // renvoie true ssi l'ensemble courant est vide
      public boolean estVide();

  // renvoie un element de l'ensemble s'il n'est pas vide
  // lance une MathException si l'ensemble est vide
      public Elt unElement();

  // renvoie true ssi e appartient a l'ensemble courant
  // lance une IllegalArgumentException en cas de parametre invalide
      public boolean contient(Elt e);

  // ajoute e (eventuellement) a l'ensemble courant
  // lance une IllegalArgumentException en cas de parametre invalide

      public void ajouter(Elt e);

  // enleve e (eventuellement) de l'ensemble courant
  // lance une IllegalArgumentException en cas de parametre invalide

      public void enlever(Elt e);

  // remplace l'ensemble courant par son complementaire
      public void complementer();

  // renvoie le cardinal de l'ensemble courant
      public int cardinal();

  // renvoie une chaine de caractere decrivant this en extension
      public String toString();
} // EnsembleInterface
```

1. Complétez les cl[illegible] Ens1 et Ens2 et testez-les à l'aide du pro[illegible]me TestEnsemble.
   Ens1 implément[illegible]emble en gardant comme attribut un [illegible] de booléens et le cardinal de l'ensemble. Ens2 [illegible]mente un ensemble en gardant c[illegible]tribut un tableau de Elt et le cardinal de l'ensem[illegible]
2. On veut maintenant pe[illegible]la « cohabitation » [illegible]mbles créés par des constructeurs différents. Afin de réaliser c[illegible]vous demande[illegible]
   (a) compléter les méthodes pa[illegible]t qui s[illegible]nt dans EnsembleInterface.java.
   (b) compléter les méthodes inclus[illegible]qui se trouvent dans la classe EnsembleAbstrait.

   Testez ces méthodes au moyen du p[illegible]stEnsembleBis.
3. Ajoutez à vos classes *Ens i* les c[illegible]teur[illegible]

```
public Ensi(Ensembl[illegible]ace a)
//cree un ensembl[illegible] les memes elements

public Ensi(E[illegible]
//fait de t[illegible]singleton{e}.
```

CHAPITRE 3

Implémentation des relations binaires

Copiez dans votre workspace le fichier que vous trouverez sur moodle dans le dossier Relations. Décompressez ce fichier et importez-le dans Eclipse.

Vous disposez d'une classe Couple dont les instances sont les couples d'éléments de l'Univers. Cette classe fournit les constructeurs :

```
public Couple (Elt x, Elt y)
  // IllegalArgumentException si x ou y est null.
public Couple (int i, int j)
  // IllegalArgumentException si i et/ou j n'appartient pas a l'Univers.
```

et les méthodes :

```
public Elt getx()
public Elt gety()
  // renvoient resp.  la premiere et la seconde composante.
public String toString()
  // description sous la forme : <(x,y)>.
public Couple reciproque()
  // renvoie le couple reciproque du couple courant.
public boolean equals(Object o)
  // renvoie vrai si le couple courant est egal a l'objet passe en parametre
public boolean hashCode()
  // renvoie un hashcode associe au couple courant
```

Relation.java est l'esquisse d'une classe destinée à implémenter le concept de relation binaire entre sous-ensembles de l'Univers. Cette classe hérite de la classe abstraite RelationDeBase. Vous aurez essentiellement, dans ces exercices, à compléter cette classe Relation ainsi que la classe RelationAbstraite. Vous n'avez pas à vous préoccuper de la manière dont les relations sont implémentées ; en effet, ce choix est fait dans RelationDeBase qui est pour vous une boîte noire qui hérite de la classe abstraite RelationAbstraite et, par conséquent, implémente l'interface RelationInterface :

```
public interface RelationInterface extends Iterable<Couple>{

// renvoie true ssi la Relation courante est vide
    public boolean estVide();

// renvoie true ssi le couple c appartient a la Relation courante
// lance une IllegalArgumentException si c est null
    public boolean contient(Couple c);

// ajoute le couple c=(x,y) (eventuellement) a la Relation courante
// lance une IllegalArgumentException si c est null
// lance une IllegalArgumentException si x n'est pas dans l'ensemble
// de depart de la relation courante
// lance une IllegalArgumentException si y n'est pas dans l'ensemble
// d'arrivee de la relation courante
    public void ajouter(Couple c);

// enleve le couple (x,y) (eventuellement) de la Relation courante
// lance une IllegalArgumentException si c est null
// lance une IllegalArgumentException si x n'est pas dans l'ensemble
// de depart de la relation courante
// lance une IllegalArgumentException si y n'est pas dans l'ensemble
// d'arrivee de la relation courante
    public void enlever(Couple c);

// renvoie une copie de l'ensemble de depart de la Relation courante
    public EnsembleAbstrait depart();

// renvoie une copie de l'ensemble d'arrivee de la Relation courante
    public EnsembleAbstrait arrivee();

// renvoie une chaine de caracteres decrivant la Relation courante
    public String toString();

// renvoie un iterateur sur la Relation courante
    public Iterator<Couple> iterator();
} // RelationInterface
```

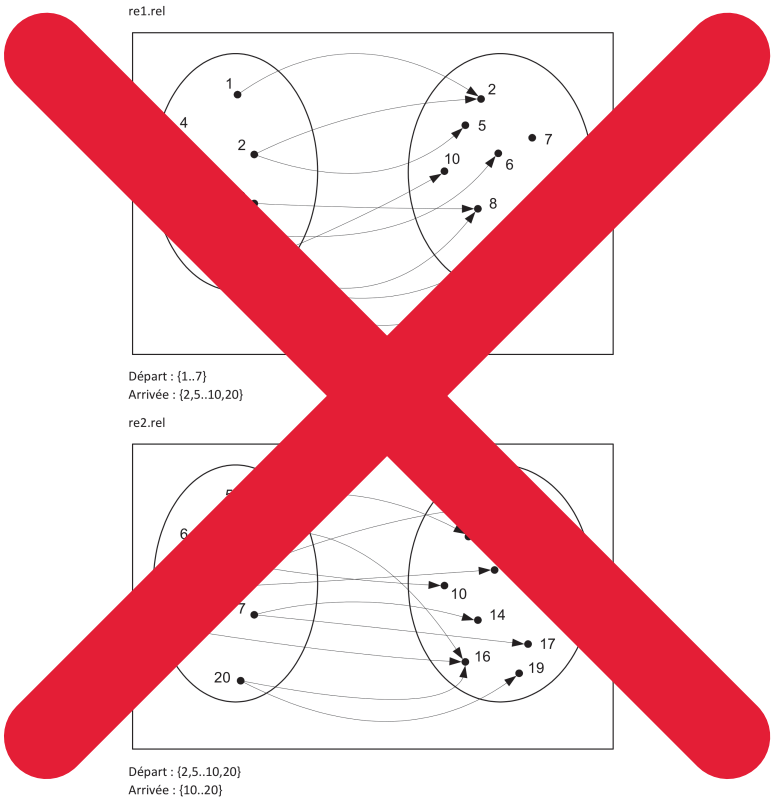
Outre ces méthodes, la classe RelationDeBase contient les méthodes suivantes qui lanceront une IllegalArgumentException si le paramètre e est null.

```
// ajoute eventuellement e a l'ensemble de depart de la relation
    public void ajouterDepart(Elt e)

// ajoute eventuellement e a l'ensemble d'arrivee de la relation
    public void ajouterArrivee(Elt e)

// supprime eventuellement e de l'ensemble de depart de la relation
// ainsi que toutes les fleches partant de e
    public void supprimerDepart(Elt e)

// supprime eventuellement e de l'ensemble d'arrivee de la relation
// ainsi que toutes les fleches aboutissant a e
    public void supprimerArrivee(Elt e)
```

et les deux constructeurs

```
// construit la Relation vide sur l'ensemble vide
   public RelationDeBase()

// construit la Relation vide de l'ensemble dep vers l'ensemble arr
// lance une IllegalArgumentException si un des parametres est null
   public RelationDeBase(EnsembleAbstrait dep, EnsembleAbstrait arr)
```

Vous utiliserez le fichier TestRelation.java pour tester vos méthodes. Les relations suivantes sont utilisées dans cette classe test :



1. Implémentez les méthodes permettant de trouver le domaine et l'image de la relation :

```
public EnsembleAbstrait domaine() {
}
// renvoie le domaine de la relation

public EnsembleAbstrait image() {
}
// renvoie l'image de la relation
```

2. Implémentez les opérations sur les relations en écrivant les méthodes ci-dessous. Ces méthodes génèreront une IllegalArgumentException dans le cas où l'opération souhaitée ne peut pas être exécutée.

```
public Relation complementaire()
// renvoie la complementaire de la Relation courante

public Relation reciproque()
// renvoie la reciproque de la Relation courante

public void ajouter(RelationInterface r)
// si possible, remplace la Relation courante par son union avec r

public void enlever(RelationInterface r)
// si possible, remplace this par sa difference avec r

public void intersecter(RelationInterface r)
// si possible, remplace this par son intersection avec r

public Relation apres(RelationInterface r)
// si possible, renvoie la composee "this apres r"
```

3. Dans la classe RelationAbstraite, complétez les méthodes booléennes :

```
public boolean inclusDans(RelationAbstraite r)
// renvoie true si this est inclue dans r

public boolean equals(Object o)
// renvoie true si this=o
```

Les deux questions qui suivent ne concernent que les relations sur un ensemble. Les méthodes demandées génèreront donc une MathException lorsque l'ensemble de départ ne coïncide pas avec l'ensemble d'arrivée.

4. Ecrivez les méthodes de clôture suivantes :

```
public void cloReflex()
 // remplace this par sa cloture reflexive

public void cloSym()
 // remplace this par sa cloture symetrique

public void cloTrans()
 // remplace this par sa cloture transitive (Warshall)
```

5. Ecrivez les méthodes booléennes suivantes :

```
public boolean reflexive()
// renvoie true ssi this est reflexive

public boolean antireflexive()
// renvoie true ssi this est antireflexive

public boolean symetrique()
// renvoie true ssi this est symetrique

public boolean antisymetrique()
// renvoie true ssi this est antisymetrique

public boolean transitive()
// renvoie true ssi this est transitive
```

6. Ajoutez à la classe relation un constructeur par recopie :

```
public Relation(RelationInterface r){
}
// construit une copie de r
// lance une IllegalArgumentException en cas de parametre invalide
```

et les méthodes statiques :

```
public static Relation identite(EnsembleAbstrait e)
// renvoie l'identite sur e
// lance une IllegalArgumentException si e est null

public static Relation produitCartesien(EnsembleAbstrait a,
    EnsembleAbstrait b)
// renvoie le produit cartesien de a et b
// lance une IllegalArgumentException si l'un des parametres est null
```