Exceptions, assertions

Objectifs

- Comprendre ce qu'est une exception (en particulier en Java)
- Savoir appeler des méthodes susceptibles de jeter des exceptions en choisissant à bon escient entre la capture et la propagation d'exceptions
- Savoir créer de nouveaux types d'exceptions, avec des méthodes susceptibles de les jeter
- Savoir utiliser les assertions et comprendre la programmation par contrats

---

Exercice 1 *Contrôle et capture des exceptions*

Soit les deux classes d'exception suivantes :

```
class Exc1 extends RuntimeException { .... }
class Exc2 extends IOException { .... }
```

Soit la classe suivante :

```
public class TestException {
        public void f1(){throw new Exc1(); }
        public void f2(){throw new Exc2(); }
        .....
}
```

Question 1. Expliquer pourquoi f1() se compile sans problème, tandis qu'une erreur de compilation se produit lors de l'analyse de f2() ?

Question 2. Proposer une solution pour résoudre ce problème de compilation.

Soient maintenant les classes d'exception suivantes :

```
class Exc3 extends Exception { ... }
class Exc31 extends Exc3 { ... }
class Exc311 extends Exc31 { ... }
class Exc32 extends Exc3 { ... }
class Exc321 extends Exc32 { ... }
```

Et soit le programme suivant :

```
public class TestException {

        public void f3() throws Exc3 {
                System.out.println("debut\_f3");
                try {
                        f4();
                }
                catch(Exc31 e) {System.out.println(e.getClass().getName());}
                finally {System.out.println("finally\_f3");}
                System.out.println("suite\_f3");
        }

        public void f4() throws Exc3 {
                System.out.println("debut\_f4");
                try {
                        f5();
                }
                catch(Exc321 e){System.out.println(e.getClass().getName());}
                catch(Exc32 e){System.out.println(e.getClass().getName());}
                finally {System.out.println("finally\_f4");}
                System.out.println("suite\_f4");
        }

        public void f5() throws Exc3 {
                System.out.println("debut\_f5");
                Exc3 e = new ..........;
                throw e;
```

```
        }
    public static void main(String[] args) throws Exc3 {
            TestException t = new TestException();
        t.f3();
        }
}
```

Indiquez ce qu'affiche le programme suivant si l'exception signalée e est une instance de :

- Exc3
- Exc31
- Exc311
- Exc32
- Exc321

Exercice 2 *Une classe Pile (faire la trace d'une exécution)*

Soient les trois classes suivantes :

```
public class PileVideException extends Exception {}
public class PilePleineException extends Exception {}

public class Pile {

        private static final int TAILLE_MAX = ...;// taille maximum de la pile
        private int t[];        // tableau stockant les elements de la pile
        private int nb; // nombre d elements dans la pile

        public Pile() {
                t = new int[TAILLE_MAX];
                nb = 0;
        }

        public boolean estVide(){       return nb == 0;  }

        public void empiler(int i) throws PilePleineException {
                if (nb == TAILLE_MAX) throw new PilePleineException();
                t[nb] = i;
                nb++;
                System.out.println("sortie de empiler");
        }

        public int sommet() throws PileVideException {
                if (nb == 0) throw new PileVideException();
                System.out.println("sortie de sommet");
                return t[nb-1];
        }

        public void depiler() throws PileVideException {
                if (nb == 0) throw new PileVideException();
                nb--;
                System.out.println("sortie de depiler");
        }

        public void depilerTout() {
                try
                   while(true) {
                        System.out.println(sommet());
                        depiler();
                    }
               catch (PileVideException e)
                   {System.out.println("le depilerTout est passe par la");}
                System.out.println("sortie du depilerTout");
        }

        public static void main(String[] args) {
                Pile p = new Pile();
                try {
                    p.empiler(2);
                    p.empiler(4);
```

```
                p.depilerTout();
                System.out.println(p.sommet());
            }
            catch (PileVideException e)
               {System.out.println("Le main pense que la pile est vide");}
            catch (PilePleineException e)
               {System.out.println("Le main pense que la pile est pleine");}
            catch (Exception e)
               {System.out.println("Le main est passe par la");}
            System.out.println("sortie du main");
        }
}
```

Question 3. Indiquer ce qui s'affiche dans la fenêtre console quand on exécute la méthode main :

- lorsque TAILLE_MAX = 0
- lorsque TAILLE_MAX = 1
- lorsque TAILLE_MAX = 2

Question 4. La méthode depilerTout() utilise abusivement le mécanisme des exceptions puisqu'elle provoque délibérément une exception qu'elle pourrait éviter. Remplacez-la par une méthode mieux conçue. Cette méthode doit, comme la méthode depilerTout(), afficher la liste des éléments selon l'ordre de dépilement et dépiler toute la pile.

Exercice 3 *Gestion d'exceptions d'E/S (capture et traitement d'exceptions)*

On veut écrire une application qui lit un fichier texte supposé exister, affiche son contenu, ainsi que le nombre total de caractères autres que les espaces (ou toute autre propriété du contenu : nombre de lignes, etc...). Le chemin d'accès au fichier à ouvrir est demandé à l'utilisateur.

Question 5. Écrire l'application demandée sans chercher à capturer les exceptions susceptibles d'être générées. Pour créer un fichier texte, vous pouvez utiliser n'importe quel éditeur de texte. Vous pouvez aussi écrire une autre application qui crée un fichier texte. Voir les programmes suivants qui vous donnent des exemples de lecture et écriture de fichiers textes.

Testez en particulier le cas où le nom (ou chemin) entré par l'utilisateur ne correspond pas à un fichier existant.

Question 6. On veut maintenant gérer l'exception "FileNotFoundException" en capturant cette exception, en signalant l'erreur à l'utilisateur, et en lui demandant d'entrer un nouveau nom de fichier, et ce, jusqu'à ce que l'ouverture aboutisse ou que l'utilisateur abandonne la tentative d'ouverture.

```
/* Exemple de programme lisant  un fichier texte nomme "essai.txt" */

import java.io.*;
public class FicText2 {
        public static void main(String args []) throws IOException {
           BufferedReader lectureFichier = new BufferedReader( new
                                           FileReader ("essai.txt"));
           System.out.println("Lecture du fichier essai.txt");
           String s = lectureFichier.readLine();

           /* readLine() retourne :
la ligne lue jusqu'au retour chariot (lu mais non retourne)
donc une chaine vide si la ligne ne comporte qu'un RC
la valeur null s il n y a rien a lire (fin du flux de donnees)
        */
           while (s!= null) {
                System.out.println(s);
                s = lectureFichier.readLine();
           }
           lectureFichier.close();
           System.out.println("Fin du fichier");
```

```
        }
}

/* Exemple de programme creant un fichier texte nomme "essai.txt" */

import java.io.*;
public class FicTexte1 {
        public static void main(String args []) throws IOException {
           BufferedReader lectureClavier = new BufferedReader( new
                                           InputStreamReader (System.in));
           BufferedWriter ecritureFichier = new BufferedWriter( new
                                            FileWriter ("essai.txt"));

   System.out.println("Entrez des lignes (Return pour terminer)");
           String s = lectureClavier.readLine();
           while (s.length() != 0)
           { ecritureFichier.write(s); // TQ pas chaine vide
              ecritureFichier.newLine();
                  s = lectureClavier.readLine();
           }
           ecritureFichier.close(); // ferme le fichier associe
           System.out.println("Fin saisie");
}
}
```

Exercice 4 *assertions : retour sur les listes ...*

Reprenez le sujet sur les listes avec classes internes, ou utilisez et adaptez n'importe quelle implémentation de liste que vous auriez sous la main.

Question 7. Ajoutez à la méthode d'ajout de noeud une assertion vérifiant que la taille de la liste après l'ajout est égale à la taille avant l'ajout, augmentée de 1.

Question 8. Ajoutez à la méthode renverser une assertion vérifiant que la taille de la liste retournée est la même que la taille de la liste receveur.

Question 9. Activez la vérification des assertions. Testez en exécutant les méthodes d'ajout et de renversement, il ne doit rien se passer de particulier (sauf bug dans votre implémentation, auquel cas corrigez les bugs). Ajoutez des bugs judicieux, et vérifiez que les assertions sont violées. Puis enlevez les bugs !

Exercice 5 *Programmation par contrats et Java*

Comme vu en cours, Java ne propose pas nativement de mécanisme permettant de faire de la programmation par contrat, hormis les assertions. Nous allons ici utiliser un outil développé initialement par Google, cofoja (pour COntracts FOr JAva), afin que vous puissiez quelque peu expérimenter la programmation par contrats.

Question 10. Installez cofoja sous Eclipse (les instructions sont aussi sur moodle) :

1. Téléchargez le jar de cofoja (`https://github.com/nhatminhle/cofoja/releases/download/v1.3/cofoja.asm-1.3-20160207.jar`) et placez-le, dans votre projet, dans un répertoire lib (que vous créerez, au même niveau que src et bin).
2. Dans la configuration de votre projet, ajoutez ce même jar au Java Build Path, puis configurez la façon dont les annotations sont prises en compte, puis configurez le "factory path" (voir captures ci-dessous)
3. Activez l'agent permettant de traiter les contrats à l'exécution : dans une configuration d'exécution (run configuration), ajoutez comme argument de VM : -javaagent :lib/cofoja.asm-1.3-20160207.jar

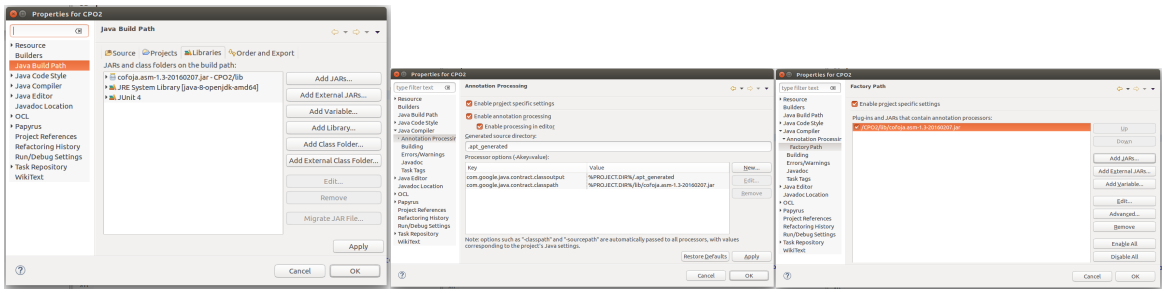

FIGURE 1 – Configurer Eclipse avec Cofoja

Question 11. Développez une classe Compte, en y ajoutant les contrats adéquats (invariants de classe, préconditions, post-conditions). La classe compte a un solde et un découvert autorisé. Le découvert autorisé est positif ou nul, il indique de quelle somme maximum le compte peut avoir un découvert. Si le découvert est nul, cela revient à ne pas avoir de découvert autorisé, et donc le solde ne peut pas devenir négatif. Le compte est muni d'un constructeur paramétré, d'une méthode créditer qui ajoute un montant (forcément positif ou nul) au solde, et d'une méthode débiter qui débite le solde d'un certainement montant (forcément positif ou nul) si et seulement si le débit ne mène pas à outrepasser le découvert autorisé. Dans le cas où le débit ne peut pas être effectué, la méthode ne fait rien et retourne faux. Si le débit a lieu, la méthode retourne vrai. Testez !

Exercice 6 *Les Entiers Naturels (générer des exceptions)*

Réaliser une classe EntNat permettant de gérer des entiers naturels (positifs ou nuls) et disposant :

- d'un constructeur avec un argument de type int ; il générera une exception de type ErrConst si la valeur de son argument est négative ;
- un accesseur en lecture getN() qui fournira sous forme d'un int la valeur encapsulée dans un objet de type EntNat ;
- un accesseur en écriture setN() qui modifiera la valeur de l'entier naturel grâce à un int passé en paramètre ; cette méthode générera une exception de type ErrModif si la valeur passée en paramètre est négative ;
- une méthode décremente() qui décrémente de 1 un objet EntNat ; cette méthode devra pouvoir lever une exception de type ErrModif ;
- une méthode de classe – statique donc – décremente(EntNat e) qui décrémente de 1 l'objet passé en paramètre (c'est juste pour que vous travaillez sur les méthodes de classe, il serait en effet normal d'en faire une méthode d'instance . . .)

Écrire une méthode main qui utilise les méthodes de la classe EntNat, en capturant les exceptions susceptibles d'être générées.

S'il vous reste du temps :

- Organisez vos classes d'exception pour qu'elles dérivent toutes d'une classe ErrNat.
- Une exception doit mémoriser la valeur erronée qui a entrainé sa génération. Modifiez vos classes d'exception de façon à ce qu'elles permettent le stockage de cette valeur, et fournissent une méthode permettant de consulter cette valeur. Testez.
- Telle qu'elle est écrite, la classe EntNat est très contraignante : par exemple, lors de la création d'une instance de EntNat, on est obligé de prendre en compte l'exception susceptible d'être générée par le constructeur, même si l'on sait que la valeur passée en paramètres est correcte. Comment rendre optionnelle la prise en compte de ces exceptions ?