TP n°7 - Correction

Exceptions et Classes Abstraites

Exercice 1 [Utilisation des Exceptions]
La méthode `parseInt` est spécifiée ainsi :

```
public static intparseInt(Strings)
                  throws NumberFormatException

    Parses the string argument as a signed decimal integer.
    The characters in the string must all be decimal digits,
    except that the first character may be an ASCII minus
    sign '-' ('\u002D') to indicate a negative value. The
    resulting integer value is returned, exactly as if the
    argument and the radix 10 were given as arguments to the
    parseInt(java.lang.String, int) method.

    Parameters:
        s - a String containing the intrepresentation to be parsed
    Returns:
        the integer value represented by the argument in decimal.
    Throws:
        NumberFormatException - if the string does not contain a parsable integer.
```

Utilisez cette méthode pour faire la somme de tous les entiers donnés en argument de la ligne de commande, les autres arguments étant ignorés.
Correction :

```
class Somme {
  public static void main(String[] args) {
    int somme = 0;
    for(int i=0;i<args.length;i++) {
      try {
        somme += Integer.parseInt(args[i]);
      }
      catch (NumberFormatException e) {}
    }
    System.out.println(somme);
  }
}
```

Exercice 2 [Création des Exceptions]
Écrire une classe `Entreprise`. Une entreprise a un nombre d'employés, un capital, un nom, une mission, et une méthode `public String mission()` qui renvoie la mission de l'entreprise

et qui déclare le lancement de l'exception `SecretMissionException`. On aura également une méthode `public int capital()` qui renvoie le capital et qui déclare le lancement de l'exception `NonProfitException`.
Écrire une classe `EntrepriseSecrete` qui hérite d'`Entreprise` et dont la méthode `mission` lance l'exception `SecretMissionException`. Écrire une classe `EntrepriseSansProfit` qui hérite d'`Entreprise` et dont la méthode `capital` lance l'exception `NonProfitException`.
Écrire une méthode qui prend en entrée un tableau d'entreprises et affiche la mission et le capital de toutes les entreprises (quand cela est possible). Tester la méthode sur les entreprises "Ford", "CIA", "Spectre", "CroixRouge", "Microsoft", "ParisDiderot".

Correction :

```
//fichier Entreprises
class SecretMissionException extends Exception {};
class NonProfitException extends Exception {};

class Entreprise{
    private String nom, mission;
    private int nombre_employes, capital;

    Entreprise(String nom, String mission, int nombre_employes, int capital){
        this.nom = nom;
        this.mission = mission;
        this.nombre_employes = nombre_employes;
        this.capital = capital;
    }
    public String mission() throws SecretMissionException{ return mission; }
    public int capital() throws NonProfitException{ return capital;    }
}

class EntrepriseSecrete extends Entreprise{
    EntrepriseSecrete(String nom, String mission, int nombre_employes, int capital){
        super(nom, mission, nombre_employes, capital);
    }
    public String mission() throws SecretMissionException{
        throw new SecretMissionException();
    }
}

class EntrepriseSansProfit extends Entreprise{
    EntrepriseSansProfit(String nom, String mission, int nombre_employes, int capital){
        super(nom, mission, nombre_employes, capital);
    }
    public int capital() throws NonProfitException{
        throw new NonProfitException();
    }
}

public class Entreprises{
    public static void tousLesEntreprises(Entreprise[] e){
        for (int i = 0; i < e.length; i++ ){
            try{
                System.out.println("Mission" + i + " = " + e[i].mission());
            }catch(SecretMissionException a){
```

```
            }try{
                System.out.println("Capital" + i + " = " + e[i].capital());
            }catch(NonProfitException b){
            }
        }
    }
    public static void main(String[] args){
        Entreprise [] t = new Entreprise[4];
        Entreprise Micro = new Entreprise("Microsoft", "Destroy the world", 1000, 1900);
        Entreprise Ford = new Entreprise("Ford", "Conquer the world", 2000, 1500);
        Entreprise CIA = new EntrepriseSecrete("CIA", "Spy Bill Gates", 23000, 35500);
        Entreprise CroixRouge =
            new EntrepriseSansProfit("CroixRouge", "look after yours health", 20, 500);
        t[0] = Micro;
        t[1] = Ford;
        t[2] = CIA;
        t[3] = CroixRouge;
        tousLesEntreprises(t);
    }
}
```

Exercice 3 [Utilisation des exceptions dans les constructeurs]
`Toutou` est une classe avec deux propriétés privées `String nom` et `int nombrePuces`.
Écrire un constructeur `public Toutou (String n, int np)` qui propage des exceptions de type `IllegalArgumentException` lorsque le nom `n` est `null` ou lorsque le nombre de puces `np` est négatif. Utiliser ce constructeur dans une méthode `main` pour contrôler les appels `new Toutou ("milou", 4)` et `new Toutou ("medor", -11)` et afficher les erreurs éventuelles lors de l'exécution des constructeurs.
Correction :

```
public class Toutou {
  private String nom;
  private int nombrePuces;
  public Toutou (String n, int np) throws IllegalArgumentException {
    if (n == null)
      throw new IllegalArgumentException("pas de nom !");
    this.nom = n;
    if (np < 0)
      throw new IllegalArgumentException("nombre negatif de puces !");
    this.nombrePuces = np;
  }

  public String toString() {
    return nom + " a " + nombrePuces + " puces.";
  }

  public static void main(String[] args) {
    try {
      System.out.println("creation d'un premier toutou");
      Toutou milou = new Toutou ("milou", 4);
      System.out.println("le voici : " + milou);
      System.out.println("creation d'un second toutou");
      Toutou medor = new Toutou ("medor",  -11);
```

```
      System.out.println("le voici : " + medor);
    }
    catch  (IllegalArgumentException e) {
      System.out.println("un toutou rate !! " + e);
    }
  }
}
```

Exécution :

```
creation d'un premier toutou
le voici : milou a 4 puces.
creation d'un second toutou
un toutou rate !! java.lang.IllegalArgumentException: nombre negatif de puces !
```

Exercice 4 [Capture d'exceptions et rôle de `finally`.]
Executez la classe suivante, et expliquez la raison de son comportement.

```
import java.io.*;

public class Except1 {
  public void methodeA(String args[]) {
    System.out.println("  methodeA : debut");
    try {
      System.out.println("  methodeA : appel de methodeB");
      this.methodeB(args);
      System.out.println("  methodeA : retour de methodeB");
      if (args.length > 99)
        throw new IOException();
    } catch (IOException e) {
      System.out.println("  methodeA : capture : "+ e);
    } finally {
      System.out.println("  methodeA : execute finally");
    }
    System.out.println("  methodeA : fin");
  }

  public void methodeB(String args[]) {
    System.out.println("    methodeB : debut");
    try {
      System.out.println("    methodeB : tente d'acceder a args[99]");
      String s = args[99];
      System.out.println("    methodeB : a reussi a acceder a args[99]");
    } catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("    methodeB : capture : "+ e);
    } finally {
      System.out.println("    methodeB : execute finally");
    }
    System.out.println("    methodeB : fin");
  }
```

```
  public static void main(String args[]) {
    System.out.println("main : debut");
    Except1 ex = new Except1();
    try {
      System.out.println("main : appel de methodeA");
      ex.methodeA(args);
      System.out.println("main : retour de methodeA");
    } catch (Exception e) {
      System.out.println("main : capture : "+e);
    }
    System.out.println("main : fin");
  }
}
```

Correction :

```
main : debut
main : appel de methodeA
  methodeA : debut
  methodeA : appel de methodeB
    methodeB : debut
    methodeB : tente d'acceder a args[99]
    methodeB : catch une Exception : java.lang.ArrayIndexOutOfBoundsException: 99
    methodeB : execute finally
    methodeB : fin
  methodeA : retour de methodeB
  methodeA : execute finally
  methodeA : fin
main : retour de methodeA
main : fin
```

L'exception `ArrayIndexOutOfBoundsException` est capturée par la `methodeB` donc la `methodeA` se déroule normalement.
Les post-traitements `finally` sont exécutés à chaque fois.

Exercice 5 [Classes abstraites.]

1. Construisez une classe abstraite `TabTrie` qui correspond à un tableau trié d'objets. Cette classe doit notamment contenir :
   – un tableau d'`Object`, `tab`, initialisé avec une `capacite` définie par défaut (il faut penser aussi à stocker le nombre d'`Object` contenus dans le tableau),

   et différentes méthodes qui peuvent être implantées ou abstraites :
   – une méthode `plusGrand` qui compare deux objects et renvoie `true` si le premier est plus grand que le deuxième,
   – une méthode `ajouter` qui insère un objet dans le tableau en respectant l'ordre croissant,
   – une méthode `toString` qui renvoie une chaîne de caractères représentant le tableau.

   Lorsque la `capacite` du tableau est atteinte, l'insertion d'un nouveau element lancera une exception `TabPlein`.
2. Construisez la classe `TabTriCouple` qui hérite de `TabTri` et ordonne des objects de type `Couple` lexicographiquement.

3. Une solution était-elle envisageable uniquement avec des interfaces ? Quel est l'intérêt ici des classes abstraites ?

Correction :

```
//fichier TabTriCouple
class TabPlein extends Exception {}

abstract class TabTrie {
    Object[] tab;
    static int capacite = 10;
    int pos;

    TabTrie(){
        tab = new Object[capacite];
        pos = 0;
    }

    abstract boolean plusGrand(Object o1, Object o2);

    void ajouter(Object o) throws TabPlein{
        Object temp;
        int i = pos;
        try{
            tab[pos] = o;
            while((i > 0) && plusGrand(tab[i-1], tab[i])){
                temp = tab[i-1];
                tab[i-1] = tab[i];
                tab[i] = temp;
                i--;
            }
            pos++;
        }
        catch (ArrayIndexOutOfBoundsException e){ throw new TabPlein(); }
    }

    public String toString(){
        String s = "[";
        for (int i = 0; i < pos; i++){ s += tab[i].toString() + "; "; }
        s += "]\n";
        return s;
    }

}

class Couple {
    int x, y;

    Couple(int x, int y){
        this.x = x;
        this.y = y;
    }

    public String toString(){
        return "(x = " + x + ", y = " + y + ")";
```

```
    }
};

public class TabTriCouple extends TabTrie{

    boolean plusGrand(Object o1, Object o2){
        Couple c1 = (Couple) o1;
        Couple c2 = (Couple) o2;
        return (c1.x > c2.x) || ((c1.x == c2.x) && (c1.y > c2.y));
    }

    public static void main(String[] args) throws TabPlein{
        TabTriCouple t = new TabTriCouple();
        for (int i = 0; i < capacite ; i++){
            t.ajouter(new Couple(((int) (Math.random() * 10)),
                                 ((int) (Math.random() * 10))));
        }
        System.out.println(t.toString());
    }
};
```