Exercice 1

Question 1.1

Consulter la documentation de la méthode `parseInt` de la classe `Integer`. Quelle exception peut-elle renvoyer ?

> Bien faire remarquer aux étudiants que l'information en question se trouve dans la documentation détaillée de la méthode, et pas dans le résumé.

Question 1.2

écrire un petit programme java qui prendra comme *arguments* (c'est à dire dans l'argument `args` de `main`) une série d'entiers, et affichera leur somme. On considère que si l'utilisateur entre un argument qui n'est pas un entier, le programme s'arrête sur l'exception :

```
$ java MonProg 15 7 a 3
Exception in thread "main" java.lang.NumberFormatException: a
at java.lang.Integer.parseInt(Integer.java, Compiled Code)
at java.lang.Integer.parseInt(Integer.java, Compiled Code)
at MonProg.main(MonProg.java:7)
```

```
public class MonProg {
    public static void main(String args[]) {
        int result= 0;
        for(int i=0; i< args.length; i++)
            {
                result += Integer.parseInt(args[i]);
            }
        System.out.println("résultat " + result);
    }
}
```

> Notez qu'il n'y a pas de `throws`. C'est dû au fait que les `NumberFormatException` sont des `RuntimeException`, et que le `throws` est optionnel dans ce cas. Bien entendu, ce n'est pas une erreur d'en mettre un.

Question 1.3

même chose, mais on veut un message d'erreur :

```
$ java MonProg 15 7 a 3
erreur : entrez uniquement des entiers
```

```
public class MonProg {
    public static void main(String args[]) {
        int result= 0;
        try {
            for(int i=0; i< args.length; i++)
                {
                    result += Integer.parseInt(args[i]);
                }
            System.out.println("résultat " + result);
```

```
            } catch (NumberFormatException e) {
                System.out.println(
                  "erreur : entrez uniquement des entiers");
            }
        }
    }
```

Question 1.4

même chose, mais cette fois-ci, en cas d'erreur, on veut afficher la somme partielle jusqu'à l'erreur :

```
$ java MonProg 15 7 a 3
22
```

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

Question 1.5

même chose, mais on ignore les arguments erronés :

```
$ java MonProg 15 7 a 3
25
```

```
    public class MonProg {
        public static void main(String args[]) {
            int result= 0;
            for(int i=0; i< args.length; i++)
                {
                    try {

                        result += Integer.parseInt(args[i]);
                    } catch (NumberFormatException e) {}
                }
            System.out.println("résultat " + result);
        }
    }
```

Exercice 2

Pour lire une ligne de texte sur l'entrée standard, il faut :

– construire à partir de celle-ci un objet de classe `BufferedReader` :

```
BufferedReader in
   = new BufferedReader(new InputStreamReader(System.in));
```

– utiliser celui-ci pour lire les données :

```
String ligne;
...
ligne= in.readLine();
```

On désire écrire une classe nommée `MaConsole`, pour disposer d'entrées/sorties simples.

Question 2.1 *écriture (pas d'exception ici)*

écrire une méthode :

`void` **`ecrire`** `(String s)`
: écrit `s` sur la console

et une méthode

`void` **`ecrire`** `(int i)`
: écrit `i` sur la console

Question 2.2 *lecture de ligne*

écrire la méthode :

`public String` **`lireLigne`** `(String question)`
: affiche `question` puis lit une ligne sur l'entrée/sortie standard. exemple :

```
MaConsole c= new MaConsole();
...
s= c.lireLigne("Bonjour quel est votre nom ?");
```

Cette méthode ne doit pas renvoyer d'exception. En cas de problème (fin de fichier ou autre), elle renvoie `null`.

Question 2.3 *lecture d'entier*

écrire la méthode :

`public int` **`lireEntier`** `(String question)`
: affiche `question,` puis lit un entier sur l'entrée/sortie standard. Si la ligne de texte tapée ne correspond pas à un entier, repose la question jusqu'à ce que la réponse soit bonne.

Dans le cas où une fin de fichier est rencontrée, lireEntier doit renvoyer 0.

Question 2.4 *lecture d'entier avec erreur possible*

écrire la méthode

`public int` **`lireCommeEntier`** `(String question)`
: affiche `question,` puis lit la ligne entrée par l'utilisateur, et essaye de l'interpréter comme un entier. Si ce n'est pas le cas, une exception est propagée.

Attention !! du fait d'une certaine incohérence dans la gestion des chaînes en java, la méthode `parseInt,` si on lui fournit comme argument une chaîne égale à `null`, renvoie un `NumberFormatException`

au lieu d'un `NullPointerException` qui serait plus logique. Dans `lireCommeEntier`, si le fichier est terminé (et que la chaîne lue est `null`), on veut renvoyer une `NullPointerException` ou une `EOFException` au choix.

Question 2.5 *Correction de `lireEntier`*

Il est assez maladroit de renvoyer 0 en cas de fin de fichier, puisque c'est une valeur possible pour un entier d'une part, et que d'autre part c'est une méthode de traitement d'erreur « artisanale ».

Modifiez `lireEntier` pour qu'elle renvoie une `EOFException` (type déjà existant) dans le cas où il n'y a plus de texte à lire.

Question 2.6

On voudrait créer pour notre classe une exception spécifique, qui serait nommée `MauvaiseEntreeException`. Les données associées seraient :

- un message d'erreur, "Mauvaise entree : " + l'entrée erronée
- l'entrée erronée ;
- la question posée.

Implémentez cette classe, et utilisez-la pour `lireCommeEntier`.

Question 2.7

écrivez une méthode `lireCommeLigneNonVide`, qui lise une ligne non vide. Si la ligne est vide, la méthode doit renvoyer une `MauvaiseEntreeException`.

La correction ne suit pas complètement l'ordre des question, pour proposer une architecture plus élégante où `lireEntier` utilise `lireCommeEntier`. La classe `MauvaiseEntreeException` doit être dans un autre fichier.

MaConsole.java

```
import java.io.*;

public class MaConsole {
    private BufferedReader in;

    public MaConsole() {
        in= new BufferedReader(new InputStreamReader(System.in));
    }

    public void ecrire(String s) {
        System.out.println(s);
    }

    public void ecrire(int i) {
        System.out.println(i);
    }


    public String lireCommeLigneNonVide(String question)
       throws MauvaiseEntreeException, EOFException {
        String result;
```

```
        result= lireLigneWithException(question);
        if (result.equals("")) {
            throw new MauvaiseEntreeException("ligne vide",
                                              question);
        }
        return result;
    }

    public String lireLigne(String question) {
        String result= null;
        ecrire(question);
        try {
            result= in.readLine();
        } catch (IOException e) {
        }
        return result;
    }

    // Version plus logique de lireLigne :

    public String lireLigneWithException(String question)
     throws EOFException {
        String result= lireLigne(question);
        if (result == null)
           throw new EOFException("fin de ligne");
        return result;
    }

    //Version 1
    //      public int lireCommeEntier(String q)
    //        throws NumberFormatException, EOFException {
    //          int result;
    //          String txt= lireLigne(q);
    //          if (txt == null)
    //              throw new EOFException("fin de fichier");
    //          result= Integer.parseInt(txt);
    //          return result;
    //      }

    public int lireCommeEntier(String q)
      throws MauvaiseEntreeException, EOFException {
        int result;
        String txt= null;
        try {
            txt= lireLigneWithException(q);

            // Ce test étaient nécessaires avec lireLigne simple :
            // if (txt == null)
            //    throw new EOFException("fin de fichier");

            result= Integer.parseInt(txt);
            return result;
        } catch (NumberFormatException e) {
            throw new MauvaiseEntreeException(txt, q);
        }
    }

    public int lireEntier(String q) throws EOFException
    {
        boolean pasDeValeur= true;
        int result= 0;
```

```
        try {
            result= lireCommeEntier(q);
            pasDeValeur= false;
        } catch (MauvaiseEntreeException e) {}

        while (pasDeValeur) {
            ecrire("il nous faut un entier !");
            ecrire(q);
            try {
                result= lireCommeEntier(q);
                pasDeValeur= false;
            } catch (MauvaiseEntreeException e) {}
        }
        return result;
    }

    public static void main(String args[])
      throws EOFException, MauvaiseEntreeException {
        MaConsole c= new MaConsole();
        String ligne1= c.lireLigne("entrez une ligne");
        c.ecrire("lu "+ ligne1);
        c.ecrire(3);
        int i= c.lireCommeEntier("entrez un entier");
        c.ecrire("vous avez tapé " + i);
        i= c.lireEntier("Encore !");
        c.ecrire("vous avez tapé " + i);
    }
}
```

MauvaiseEntreeException.java

```
public class MauvaiseEntreeException extends Exception {
    private String entree;
    private String question;

    public MauvaiseEntreeException(String entree,
                                   String question)
    {
        super("mauvaise réponse à la question : "
              + question + " : " + entree);
        this.entree= entree;
        this.question= question;
    }

    public String getEntree() {
        return entree;
    }

    public String getQuestion() {
        return question;
    }

}
```