



Rapport de travaux pratiques. Exceptions

Réalisé par : Oussama EDDAHRI.

Le 09/02/2020.

Année Universitaire 2019-2020

Introduction:

Une exception est une erreur se produisant dans un programme qui conduit le plus souvent à l'arrêt de celui-ci.

Ce rapport présente la correction des exercices des travaux pratiques des exceptions.

Exercice 1:

EX1 : (Déclenchement et traitement d'une exception)

- Réaliser une classe **EntNat** permettant de manipuler des entiers naturels (positifs ou nuls). Pour l'instant, cette classe disposera simplement :
 1. d'un **constructeur** à un argument de type `int` qui générera une exception de type **ErrConst** (type classe à définir) lorsque la valeur reçue ne conviendra pas.
 2. d'une méthode **getN** fournissant sous forme d'un `int`, la valeur encapsulée dans un objet de type `EntNat`.

Question : Ecrire un petit programme d'utilisation qui traite l'exception **ErrConst** en affichant un message et en interrompant l'exécution.

Solution:

```
package Exceptions;

public class EntNat {
    int N;

    public EntNat(int N) throws ErrConst{
        if(N<0) throw new ErrConst("Nombre negative");
        this.N = N;
    }

    public int getN() {
        return N;
    }
}

package Exceptions;

public class ErrConst extends Exception{

    public ErrConst(String message) {
        super(message);
    }
}
```

Class de test :

```
package Exceptions;

public class Main {

    public static void main(String[] args) {
        try {
            EntNat entNat = new EntNat(-5);
        } catch (ErrConst e) {
            // TODO Auto-generated catch block
            System.out.println(e.getMessage());
        }
    }
}
```

Affichage :

Nombre negative

Exercice 2:

EX2: (Transmission d'information au gestionnaire)

- Adapter la classe **EntNat** de l'exercice 1 et le programme d'utilisation de manière à disposer dans le gestionnaire d'exception du type **ErrConst** de la valeur fournie à tort au constructeur.

Problème : (Synthèse du chapitre)

- Réaliser une classe permettant de manipuler des entiers naturels (positifs ou nuls) et disposant :
 1. d'un constructeur à un argument de type int ; il générera une exception **ErrConst** si la valeur de son argument est négative.
 2. de méthodes statiques de somme, de différence et de produit de deux naturels ; elles généreront respectivement des exceptions **ErrSom**, **ErrDiff** et **ErrProd** lorsque le résultat ne sera pas représentable ; la limite des valeurs des naturels sera fixée à la plus grande valeur du type int.
 3. Une méthode d'accès **getN** fournissant sous forme d'un int la valeur de l'entier naturel. On s'arrangera pour que toutes les classes exceptions dérivent d'une classe **ErrNat** et pour qu'elles permettent à un éventuel gestionnaire de récupérer les valeurs ayant provoqué l'exception.

Questions : Ecrire deux exemples d'utilisation de la classe :

1. l'un se contentant d'intercepter sans discernement les exceptions de type dérivé de **ErrNat**.
2. l'autre qui explicite la nature de l'exception en affichant les informations disponibles.

Les deux exemples pourront figurer dans deux blocs try d'un même programme.

Remarque : la plus grande valeur entière est définie par : **Integer.MAX_VALUE**

Solution :

```
package Exceptions;
```

```
public class EntNat {
    int N;

    public EntNat(int N) throws ErrConst{
        if(N<0) throw new ErrConst("Nombre negative");
        this.N = N;
    }

    public int getN() {
        return N;
    }

    public static int somme(int a, int b) throws ErrSom {
        if(a+b<0) throw new ErrSom("Somme Plus grand");
        return a+b;
    }

    public static int difference(int a, int b) throws ErrDiff{
        if(a-b<0 && a>b) throw new ErrDiff("difference plus grand");
        return a-b;
    }

    public static int produit(int a, int b) throws ErrProd {
        if(a*b<0) throw new ErrProd("produit plus grand");
        return a*b;
    }
}
```


```
package Exceptions;
```

```
public class ErrNat extends Exception{

    public ErrNat(String message) {
        super(message);
        // TODO Auto-generated constructor stub
    }
}
```

```
package Exceptions;
```


```
public class ErrConst extends ErrNat{
```



```
    public ErrConst(String message) {  
        super(message);  
    }  
}
```

```
package Exceptions;
```


```
public class ErrSom extends ErrNat{
```



```
    public ErrSom(String message) {  
        super(message);  
    }  
}
```

```
package Exceptions;
```


```
public class ErrProd extends ErrNat{
```



```
    public ErrProd(String message) {  
        super(message);  
    }  
}
```

```
package Exceptions;
```

```
public class ErrDiff extends ErrNat{
```



```
    public ErrDiff(String message) {  
        super(message);  
    }  
}
```

Class de test :

```

package Exceptions;

public class TEST{

    public static void main(String[] args) {
        try {
            EntNat entNat = new EntNat(5);
            System.out.println(EntNat.somme(5, 10));
            System.out.println(EntNat.difference(5, 10));
            System.out.println(EntNat.produit(5, 10));

        } catch (ErrNat e) {
            // TODO Auto-generated catch block

            System.err.println(e.getMessage());
        }
    }
}

```

Affichage :

15
-5
50

Test d'exception

```
package Exceptions;
```

```
public class TEST{
```

```
    public static void main(String[] args) {
```

```
        try {
```

```
            EntNat entNat = new EntNat(5);
```

```
            System.out.println(EntNat.somme(1000000000*100000000, 10));
```

```
        } catch (ErrNat e) {
```

```
            // TODO Auto-generated catch block
```

```
            System.err.println(e.getMessage());
```

```
        }
```

```
    }
```

```
}
```

run:

Somme Plus grand

BUILD SUCCESSFUL (total time: 0 seconds)

|

