Les Exceptions

Introduction informelle aux exceptions

Considérons l'exemple suivant :

```
Avion A = new Avion
("AirbusA380","PISTE_6");

 A.decolle();

A.voleJusquaProchaineEtape();

A.atterit();
```

Ici la situation normale est que l'avion ne soit pas confronté à un accident

Exemple 2:

```
public class test {

        public static void main(String args[]) {

                Scanner lec= new Scanner(System.in);

                Int nombre

                nombre = lec.nextInt();

                x = 35 * 246;

                int result = x / nombre;

                System.out.println("Enter your name");

                String nom  =lect.next();

                System.out.println("Bye " + nom );

                lec.close();}}
```

Ici la situation normale est que l'utilisateur introduise un entier différent de zéro.

LES EXCEPTIONS

Le programme se termine immédiatement en état d'erreur avec le message suivant dans la fenêtre console : "*Exception in thread main": java.lang.ArithmeticException / by zero at test.main si on introduit zéro et InputMismatchException si on introduit « abc ».*

Notion d’exception :

Une **exception** est un signal qui indique que quelque chose d'exceptionnel est survenue en cours d'exécution. Deux solutions alors :

- laisser le programme se terminer avec une erreur,
- essayer, malgré l'exception, de continuer l'exécution normale.

Une exception est un objet ou une instance d'une classe d'exception. Il provoque la sortie d'une méthode. Il correspond à un type d'erreur et contient des informations sur cette erreur.

Les Exceptions standards

On a un certain nombre d'exceptions standards

- Il en existe une quarantaine.
- Chaque type d'exception est une classe.
- Nous retiendrons les suivantes qui dérivent de la classe Exception:
    - ArithmeticException
    - NumberFormatException
    - IndexOutOfBoundsException
    - NegativeArraySizeException
    - NullPointerException

LES EXCEPTIONS

Ces classes sont dans le paquetage j**ava.lang** importé par défaut**.** D'autres classes d'exceptions standards telles que IOException et FileNotFoundException existent et sont dans le paquetage **java.io.IOException** qui doit être importé explicitement.

Remarque : Les exceptions construites par l'utilisateur doivent étendre la classe Exception

Gestion des exceptions

Toutes les exceptions prédéfinies provoquent l'arrêt du programme (en état d'erreur) à moins que le programmeur mette en place un mécanisme de gestion des exceptions. Une telle gestion se fait avec *try/catch*. Ainsi nous décidons nous-même ce qui est à faire en cas d'exception. On doit placer le code à tester dans un bloc *try.*

- *Le bloc try contient le code à exécuter quand tout fonctionne* ***correctement****.*

  Exemple :

```
try {
        A.decolle();
        A.voleJusquaProchaineEtape();
        A.atterit(); }
```

- On doit faire suivre le bloc *try* immédiatement d'un ou plusieurs bloc(s) *catch* dans lequel on place les instructions à exécuter en cas d'exception.

LES EXCEPTIONS

Exemple :

```
try {
   A.decolle();
   A.voleJusquaProchaineEtape();
   A.atterit(); }
catch(ProblemeTechniqueException e)
{   A.atteritDurgence();
    if(!A.peutAtterir()) {
      A.getPilote().annonce("Vos Parachutes et sautez !! vite !!");
      A.getPilote().sauteEnParachute(); }}
```

LES EXCEPTIONS

```
public class test {

public static void main(String args[]) {

Scanner lec= new Scanner();

try{

1. Int i= nombre.nextInt();
2. int x, nombre, result;
3. x = 35 * 246;
4. result = x / nombre;
5. System.out.println("Result:" + result);}

6.Catch(Exception e)

{7. System.out.println(“Objet exception: ”+e.getClass());}

8. String nom  =lect.next("Enter your name");

9. System.out.println("Bye " + nom ); }}
```

LES EXCEPTIONS

Format général d'un gestionnaire d'exception

```
try {
    <lignes de code à protéger>
}catch ( UneException1  e ) {
    <lignes de code réagissant à l'exception UneException1 >
}
}catch ( UneException2  e ) {
    <lignes de code réagissant à l'exception UneException2 >
} …
```

finally

{ Lignes de code s'exécutant quelque soit la situation même si dans l'un des blocs contient l'instruction return }

LES EXCEPTIONS

Exemple 1(ArithmeticException) : se produit lors d'erreurs d'arithmétique, telle la division par zéro

```
nombre = 0;
try {
        result = x / nombre;
       System.out.println( "Résulat:" + result);
}
catch (ArithmeticException erreur)
{ System.out.println( "Division par zero" );}
```

LES EXCEPTIONS

Exemple 2 (NumberFormatException) : se produit lors d'une conversion de chaîne à numérique impossible à réaliser

```
Import java.util.*;

int nombre;

String reponse;

Scanner lec=new Scanner();

System.out.println("Entrez un nombre entier");

reponse = lec.next();

try { nombre  = Integer.parseInt(reponse);}

catch (NumberFormatException e)  {

 System.out.println( "Ceci n'est pas un entier!!");

 System.exit(0);}

 double result = nombre * nombre;
```

L'exception est levée si l'usager entre *abc* ou *1.4* par exemple.

Remarques :

Avec *Double.parseDouble*, l'exception NumberFormatException ne survient pas lorsque l'usager entre un entier (il y a conversion implicite).

On pourrait placer ce code dans une boucle permettant ainsi de reposer la question tant que l'usager n'entre pas un entier valide

LES EXCEPTIONS

Exemple 3 (IndexOutOfBoundsException) Se produit quand un index est hors-limite dans une chaîne ou un tableau.

Exemple avec un tableau:

```
int notes[ ] = new int[5];
int n=0;
try {
    for (int k=0; k<5; k++) {
            notes[ k ] = 100 - k;}
    n = lec.nextInt();
    System.out.println("La note est: " + notes[n] );
}
catch (IndexOutOfBoundsException e)  {
     System.out.println( "Ce numéro ne correspond pas à une note!");
 }
```

LES EXCEPTIONS

Exemple avec une chaîne

```
String phrase = ‘’JAVA ‘’;
int n = lect.nextInt();
try {
    char lettre = phrase.charAt( n );
    System.out.println("Le caractère numéro " + n "  est " + lettre );
}
catch (IndexOutOfBoundsException e)  {
            System.out.println( "Ce numéro ne correspond pas un caractère!");
 }
```

LES EXCEPTIONS

```
Point p1, p2=null;
p1 = new Point(5, 8);
try{
        String reponse =lec.next();
        if ( reponse.equals("99" )) {
                p2 = p1;
                p2.affiche( ); (1)
        }}
    catch(NullPointerException  e){System.out.println(“Reference null”);   }
```

Checked Exceptions (exceptions vérifiées)

Une **checked exception** est une exception que le compilateur Java oblige à **gérer explicitement**. Le programmeur doit soit :

- l'intercepter avec un bloc try-catch
- soit la déclarer avec throws dans la signature de la méthode

Exemples :

•IOException
•SQLException
•FileNotFoundException

Exemple:

```
void lireFichier() throws IOException {
FileReader f = new FileReader("data.txt");
}
```

Ce code définit une méthode **lireFichier()** qui tente d'ouvrir un fichier nommé "data.txt" à l'aide d'un **FileReader**. Comme l'ouverture d'un fichier peut échouer, cette opération peut générer une **IOException**, qui est une ***checked exception***. C'est pourquoi la méthode est déclarée avec **throws IOException.**

LES EXCEPTIONS

Unchecked Exceptions (exceptions non vérifiées) est une exception que le compilateur **n'oblige pas à gérer et** surviennent généralement à cause d'erreurs de programmation (bugs).

Exemples :

- NullPointerException
- ArithmeticException
- IndexOutOfBoundsException
- NumberFormatException

Principe de fonctionnement de l'interception

Dès qu'une **exception est levée** (instanciée), la Java machine **stoppe immédiatement** l'exécution normale du programme à la **recherche d'un gestionnaire** d'exception (try—catch) susceptible d'intercepter (saisir) et traiter cette exception. Dans ce dernier cas, nous avons :

- les clauses catch sont examinées l'une après l'autre dans le but d'en trouver une qui traite cette classe d'exceptions (ou une superclasse).
- Les clauses catch doivent donc traiter les exceptions de la plus spécifique à la plus générale.
- Si une clause catch convenant à cette exception a été trouvée et le bloc exécuté, l'exécution du programme reprend son cours.
- Un bloc finally permet au programmeur de définir un ensemble d'instructions qui est toujours exécuté, que l'exception soit levée ou non, capturée ou non.

Remarque : La seule instruction qui peut faire qu'un bloc finally ne soit pas exécuté est System.exit().

Déclenchement manuel d'une exception existante (prédéfinie)

- La Java machine peut déclencher une exception automatiquement comme dans l'exemple de la levée d'une ArithmeticException lors de l'exécution de l'instruction "x = 1/0 ;".
- La Java machine peut aussi déclencher une exception à votre demande suite à la rencontre d'une instruction **throw**. Le programme qui suit lance une ArithmeticException avec le message "Mauvais calcul !" dans la méthode **calcul(int y)** .
- Quand on va lancer la méthode calcul, si on introduit une valeur négative, le programme va s'arrêter. Si on veut que le programme continue à s'exécuter, il suffit de rajouter le bloc try/catch

LES EXCEPTIONS

```
public class Trajet{
double distance;
public Trajet (double d ) throws Exception {

{ System.out.println("…Avant Incident  ) ;

if( d <0) {

   throw new Exception(" Distance ne peut être négative !");}
   System.out.println("Apres Incident… " ) ;
}
 public static void main(String[] args) throws Exception  {
          Scanner lec = new Scanner(System.in);}
 }
```

Création d'une classe exception personnalisée

- On a la possibilité de créer une nouvelle classe d'exception, pour ce faire, il faut qu'elle **hérite obligatoirement de la classe Exception** ou de n'importe laquelle de ses sous-classes.
- Créons une classe d'exception que nous nommerons ArithmeticExceptionPerso héritant de la classe desArithmeticException puis exécutons ce programme :

LES EXCEPTIONS

```
public class  ArithmeticExceptionPerso {
int x =8;
public ArithmeticExceptionPerso(String  s){ super(s) ;}}
class A {
int calcul(int y)throws  ArithmeticExceptionPerso
{ System.out.println("…Avant Incident " ) ;
if( y <0) throw new  ArithmeticExceptionPerso (Mauvais Calcul");
 System.out.println("Apres Incident… " ) ;

return x+y;}
public static void main(String[] args)   {

 A  O = new A();
 Scanner lec = new Scanner(System.in);
 int  x= lec.nextInt();
 System.out.println("Debut propramme:") ;
 try { System.out.print( O.calcul(x))};
catch( Exception e) { System.out.println("Interception exception :"+     e.getMessage()) ;}
System.out.println("Fin programme:") }
```