Cours 7

•Exceptions

1. Une exception est un **signal**
- qui indique que quelque chose d'exceptionnel

(par exemple une erreur) s'est produite

- qui interrompt le flot d'exécution normal du programme

2. lancer (throw) une exception consiste à la signaler

3. attraper (catch) une exécution permet d'exécuter les actions nécessaires pour traiter cette situation

```
public class TSTExcp1
{public static void main (String[] args)
   {int tabEnt[]={1,2,3,4};
   tabEnt[4]=3; //erreur et arret
   System.out.print("fin de main ");
 }}
```

```
java TSTExcp1
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
      at TSTExcp1.main(TSTExcp1.java:5)
```

```
public class TSTExcp2
{public static void main (String[] args)
   {int tabEnt[]={1,2,3,4};
   try {tabEnt[4]=3;}
   catch(Exception e){System.out.println("Exception: "+e);}
   System.out.print("fin de main ");
 }}
```

```
java TSTExcp2
Exception: java.lang.ArrayIndexOutOfBoundsException: 4
fin de main
```


try-throw-catch

- bloc **try** le code lorsque “tout va bien”
  et on lance l’exception à partir de ce bloc

```
try
{CodeThatMayThrowAnException}
catch(Exception e)
{
ExceptionHandlingCode
  }
```

```
throw new ExceptionClassName(PossiblySomeArguments);
```

exécution du bloc **try** block est arrêtée, normalement
le contrôle est transféré au bloc **catch**

- bloc **catch** a un paramètre (objet exception créé)

```
catch(Exception e)
{
  ExceptionHandlingCode
}
```

Lorsque le bloc try est exécuté:

1. Aucune exception est lancée dans ce bloc:

- le code du bloc catch n'est pas exécuté
- l'exécution continue avec le bloc suivant

2-Une exception est lancée:

- l'exécution du try est interrompu
- le bloc du catch est exécuté
- l'exécution continue avec le bloc suivant de catch

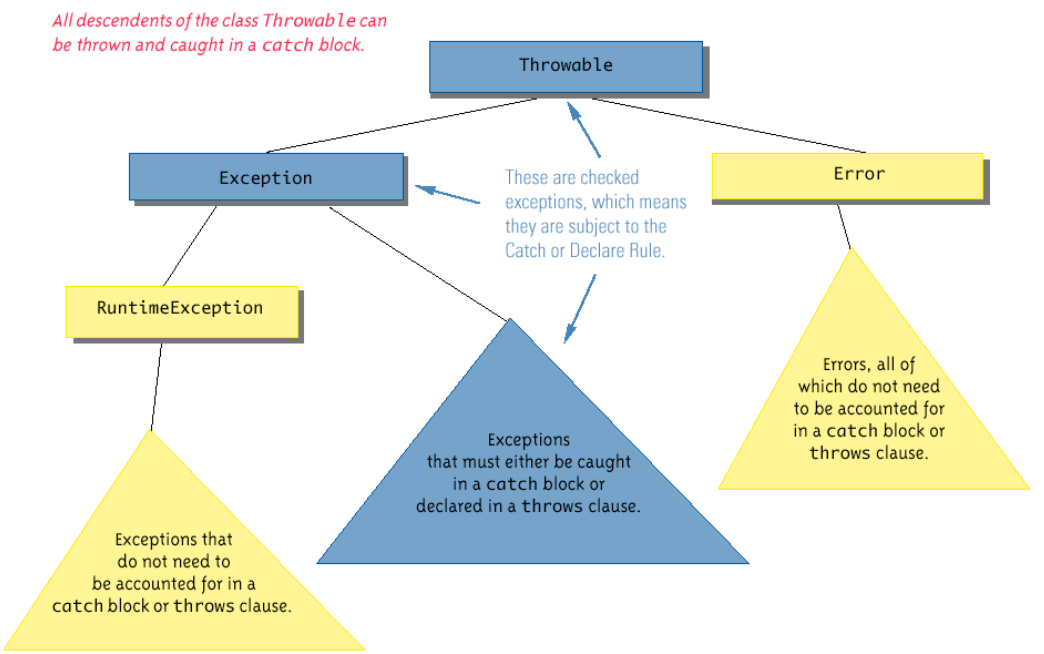
- si une exception n'est pas attrapée dans le bloc où elle a été lancée elle est transmise au bloc de niveau supérieur (récursivement)

- si une exception n'est pas attrapée dans la méthode qui la lance, elle est propagée dans la méthode qui invoque cette dernière (récursivement)

- si une exception n'est jamais attrapée
  1- propagation jusqu'à la méthode main()
  2- affichage des messages d'erreurs et la pile d'appels
  3- arrêt de l'exécution du programme

Hierarchy of Throwable Objects

Display 9.10 **Hierarchy of Throwable Objects**



En JAVA, les exceptions sont des objets

Throwable(java.lang.Throwable)

Exception récupérable

Exceptions usuelles
NullPointerException
NegativeArraySizeException
IndexOutofBoundsException
ArrayIndexOfBoundsException
StringIndexOutOfBoundsExcpetion
IllegalArgumentException
ArithmeticException
NumberFormatException
IOException
InterruptedExcpetion

Error
non récupérable
erreur système
(plus de mémoire,
problème de chargement
dynamique, etc.

Puisque les exceptions sont des objets, elles contiennent

•attributs (variables) particuliers
•méthodes particulières

aussi attributs et méthodes standards

String getMessage()
void printStacktrace()
…

Les exceptions étant des objets, leur structuration en une hiérarchie des classes permet de les traiter à différent niveaux d'abstraction (via polymorphisme)


```
class EntNat
{private int n;

public EntNat(int m) throws ErrConst
{if (m<0) throw new ErrConst(m);     n=m;}

public int getn()  {return n;}
}

class ErrConst extends Exception
{private int errval;

public ErrConst(int valeur)     {errval=valeur;}
public int getValeur()      {return errval;}
}
public class TestExcep2
{ public static void main (String[] args)
         {try
         {EntNat n1= new EntNat(20);
         System.out.println("n1=  "+n1.getn());
         EntNat n2= new EntNat(-12);
         System.out.println("n2 = "+n2.getn());
         }
         catch (ErrConst e)
         {System.out.println("Erreur de construction avec valeur "+e.getValeur());
         System.exit(-1);
         }
}}
```

•un bloc try est suivi d'un ou plusieurs blocs de catch
qui permettent d'attraper les exceptions dont le type est spécifié
et d'exécuter le code spécifique

•un seul bloc de catch peut être exécuté
(le premier susceptible d'attraper l'exception)

1- chaque clause catch est déclarée avec un argument de type Throwable ou sous-classe de Throwable

2- quand une exception est levée dans le bloc try, la première clause dont le type d'argument correspond à celui de l'exception levée est invoquée

•l'ordre des blocs de catch est important

Classes d'Exception

•Il y a plusieurs classes d'exception prédéfinies. Elles ont toutes
- •un constructeur avec un argument de type string
- •une méthode accesseur getMessage() pour récupérer cet argument

•Plusieurs classes d'exception ont définies dans différents packages standards

```
import java.io.IOException;
```

•La classe Exception définie dans java.lang est la classe de base de toute classe d'exception

•Une classe d'exception doit être dérivée d'une classe d'exception

finally

•permet de rassembler dans un seul bloc un ensemble d'instructions autrement auraient du être dupliquées

•permet d'effectuer des traitements après le bloc try,
même une exception a été déclenchée ou non et attrapée ou non

•optionnel


```
class Erreur extends Exception { }
class Erreur1 extends Erreur  {  }
class Erreur2  extends Erreur  {  }

class A
{public A (int n) throws Erreur
{ try{   if (n==1) throw new Erreur1();
         if (n==2) throw new Erreur2();
         if (n==3) throw new Erreur();
       }
         catch ( Erreur1 e)
         {System.out.println(" Exception Erreur1 dans  A");}
}}

public class TestExcep3bis
{ public static void main (String[] args)
{int n;
for (n=1; n<=3; n++)
         { try {A a=new A(n);}
           catch (Erreur1 e)
           {System.out.println("Exception Erreur1 dans main");}
           catch (Erreur2 e)
           {System.out.println("Exception Erreur2 dans main");}
           catch (Erreur e)
           {System.out.println("Exception Erreur dans main");}
          System.out.println("-----------------");
          }
 System.out.println("fin de main");
}}
```

```
Exception Erreur1 dans  A
-----------------
Exception Erreur2 dans main
-----------------
Exception Erreur dans main
-----------------
fin de main
```

```
class Erreur extends Exception { }
class Erreur1 extends Erreur  {  }
class Erreur2  extends Erreur  {  }

class A
{public A (int n) throws Erreur
{ try{   if (n==1) throw new Erreur1();
         if (n==2) throw new Erreur2();
         if (n==3) throw new Erreur();}
         catch ( Erreur1 e)
         {System.out.println("Exception Erreur1 dans  A");}
         catch (Erreur e)
         {System.out.println("Exception Erreur dans  A");
          throw(e);}
}}

public class TestExcep3
{ public static void main (String[] args)
{int n;
for (n=1; n<=3; n++)
     {   try {A a=new A(n);}
         catch (Erreur1 e)
         {System.out.println("Exception Erreur1 dans main");}
         catch (Erreur2 e)
         {System.out.println("Exception Erreur2 dans main");}
         catch (Erreur e)
         {System.out.println("Exception Erreur dans main");}
    System.out.println("-----------------");}
 System.out.println("fin de main");
}}
```

```
Exception Erreur1 dans  A
-----------------
Exception Erreur dans  A
Exception Erreur2 dans main
-----------------
Exception Erreur dans  A
Exception Erreur dans main
-----------------
fin de main
```



```
class EntNat
{private int n;

public EntNat(int m) throws ErrConst
{if (m<0) throw new ErrConst();    n=m;}

public int getn()  {return n;}
}

class ErrConst extends Exception
{}
public class TestExcep4
{ public static void main (String[] args)
         {try
         {EntNat n1= new EntNat(20);
         System.out.println("n1=  "+n1.getn());
         EntNat n2= new EntNat(-12);
         System.out.println("n2 = "+n2.getn());
         }
         catch (ErrConst e)
         {System.out.println("Erreur de construction");
         //System.exit(-1);
         }
         finally
         {System.out.println("terminaison");}
}
}
```

```
n1=  20
Erreur de construction
terminaison
```