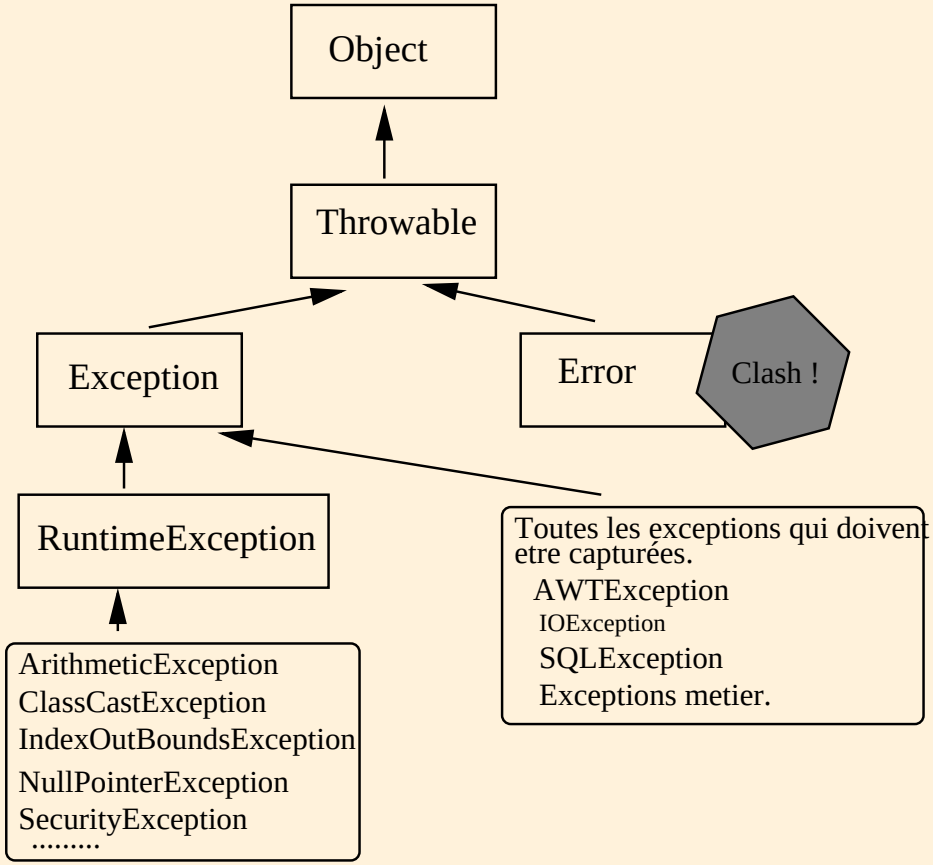
Hiérarchie des exceptions I



Qu'est ce qu'une exception ? I

- C'est un objet qui est instancié lors d'un incident :
- le traîtement du code de la méthode est interrompu
  $\Rightarrow$ l'exception est propagée à travers la pile d'exécution.
- L'exécution se termine par une erreur si
  $\Rightarrow$ l'exception n'est capturée par aucune méthode

Exemple I

- Dans cette méthode, il y une division par 0

```
1  class Exception1{
   public static void methode()
3  {
    int d , a ;
5   d=0;
    a=10/d;
7  }
```

- Appel de la méthode dans le main

```
1  public static void main()String args[])
   {
3   Exception1.methode();
   }
5  }
```

- A l'exécution, l'erreur se propage

Exemple II

```
1  Exception in thread "main" java.lang.ArithmeticException:
    / by zero
3          at Exception1.methode(Exception1.java:5)
           at Exception1.main(Exception1.java:8)
```

Qu'est ce qu'une exception ? I

- Le principe repose sur
  $\Rightarrow$ l'obligation de prévoir un traitement d'erreur
  $\Rightarrow$ sur les instructions susceptibles de les provoquer.
- Les avantages de ce mécanisme :
  1. Séparation
     $\Rightarrow$ de la gestion des erreurs du code normal
  2. Propagation
     $\Rightarrow$ possible des erreurs dans la pile des appels
  3. Centralisation
     $\Rightarrow$ des erreurs : la gestion se fait par type.

Capturer l'exception avec *try* et *catch* I

- la clause try définit un bloc d'instructions
  $\Rightarrow$ pour lequel on désire capturer les exceptions levées.
- L'exécution du bloc est interrompue dés que la première exception est levée.
  $\Rightarrow$ Le contrôle est alors passé à la clause catch.
- la clause catch définit l'exception à capturer en
  $\Rightarrow$ référençant l'objet de cette exception par un paramétre,
  $\Rightarrow$ puis le bloc à exécuter en cas de capture.

Capturer l'exception avec *try* et *catch* I

- Voici le même programme avec un bloc try/catch

```
   public static void main(String args[]s) {
2  int d,a;
   try{
4     d=0; a=23/d;
      System.out.println("Ceci ne sera jamais imprime");
6     }
   catch(ArithmeticException e){
8  System.out.println("Division par zero");
     }
10 }
```

- Voici le résultat de l'exécution du programme ci-dessus.

```
   Division par zero
```

Gérer une exception dans le bloc catch I

- Les clauses catch bien construites cherche à résoudre la condition exceptionnelle comme s'il n y avait jamais eu d'erreur.

```
1  public static void main(String args[]) {
   int a=0, b=0, c=0;
3  Random r=new Random();
   for(int i=0;i<100;i++){
```

- Cotrole du bloc try :

```
   try {
2     b=r.nextInt();c=r.nextInt();a=12345/(b/c);
      }
```

- Capture de l'erreur :

Gérer une exception dans le bloc catch II

```
1   catch(ArithmeticException e){
        System.out.println("Division␣par␣zero");
3       a=0; // met a a zero et continnue
      }
5   System.out.println("␣a␣:"+ a);}}
```

Clause catch multiple I

- Un code peut générer plusieurs exceptions.
  - plusieurs clauses catch
  - chaque clause capture une exception différente
  - lorsqu'une exception est lancée,
    $\Rightarrow$ chaque instruction catch est examinée
  - une et une seule clause est exécutée,
    $\Rightarrow$ les autres clauses sont ignorées.

Exemple avec catch multiple I

```
1  class MultiCatch{
   public static void main(String args[]) {
3  try{int a=args.length;
       System.out.println("a␣:␣"+ a);
5      int b=24/a;
       int c[]={1};
7      c[24]=99;
       }
```

```
   catch(ArithmeticException e)
2      {
        System.out.println("Division␣par␣zero␣:"+ e);
4      }
```

```
   catch(ArrayIndexOutOfBoundsException e){
2       System.out.println("Index␣du␣tableau␣:␣",+e);
        }
4      System.out.println("Apres␣les␣blocs␣try/catch");}}
```

Exemple avec catch multiple II

```
java MultiCatch
a : 0
Division par zero :java.lang.ArithmeticException: / by zero
Apres les blocs try/catch
```

```
java MultiCatch 3
a : 1
Index du tableau:java.lang.ArrayIndexOutOfBoundsException
    :24
Apres les blocs try/catch
```

Calcul de la moyenne de notes entières I

```
class ExceptionCatch{
static int moyenne(String[] tab){
int somme = 0,note, nbNotes = 0;
for (int i = 0; i < tab.length; i++){
try{
   note = Integer.parseInt(tab[i]);
   somme += note; nbNotes++;
   }
catch(NumberFormatException e){
   System.out.println("La " + (i+1)+" eme note n'est"
   +"pas entiere");}
}
return somme/nbNotes;}
public static void main(String[] argv){
System.out. println("La moyenne est "+moyenne(argv));
}
}
```

```
La 1 eme note n'est pas entiere
La moyenne est 13
```

Calcul de la moyenne de notes entières II

```
Pour : java ExceptionCatch ha 15.5
La 1 eme note n'est pas entiere
La 2 eme note n'est pas entiere
java.lang.ArithmeticException: / by zero
        at ExceptionCatch.moyenne(ExceptionCatch.java:22)
        at ExceptionCatch.main(ExceptionCatch.java:27)
```

Définir sa propre exception I

- Si on veut signaler un événement exceptionnel d'un type non prévu par l'API,
  $\Rightarrow$ il faut étendre la classe java.lang.Exception
- la classe étendue contient en général
  $\Rightarrow$ un (ou plusieurs) constructeur(s) et éventuellement une redéfinition de la méthode toString.
- Lors du lancement de l'exception, à l'aide du mot réservé throw,
  $\Rightarrow$ on crée une instance de la classe définie.

Définir sa propre exception : Exemple I

```
class AucuneNoteValide extends Exception
{
AucuneNoteValide()
{
  System.out.println("Une␣exception␣AucuneNoteValide␣s'est␣
      produite");
}
public String toString()
{
return "Aucune␣note␣n'est␣valide";
}
}
```

Calcul de la moyenne I

- Pour cet exemple et le suivant, l'objectif est de calculer une moyenne de notes entières envoyées en arguments par la ligne de commande. Les arguments non entiers (et donc erronés) doivent être éliminés. On utilise l'exception AucuneNoteValide.
- Si aucune notes n'est valide :

```
1  java ExceptionThrows ha 15.5
   La 1 eme note n'est pas entiere
3  La 2 eme note n'est pas entiere
   Aucune note n'est valide
```

- la méthode moyenne est susceptible de lever une excepption "AucuneNoteValide" $\Rightarrow$throws

```
   static int moyenne(String[] liste)throws AucuneNoteValide{
2  int somme=0,x, nbNotes=0; int i;
   for(i=0;i<liste.length;i++){
4  try{
     x=Integer.parseInt(liste[i]);
6    somme=somme+x;
     nbNotes++;
8    }
   catch(NumberFormatException e)
10  {
     System.out.println("La␣"+(i+1)+"␣eme␣note␣n'est␣"+
12   "pas␣entiere");
   }
14 }
   if (nbNotes==0) throw new AucuneNoteValide();
16 return somme/nbNotes;
   }
```

- la méthode main capture l'exception "AucuneNoteValide"

```
1  public static void main(String[] argv){
   try{
3     System.out.println("La moyenne est " + moyenne(argv));
      }
5  catch(AucuneNoteValide e)
     {
7    System.out.println(e);
     }
9  }
```

Exemple d'utilisation des exceptions I

- Ecrivons la fonction int evalue (int x, char op, int y) :
  - calcule l'expression arithmétique $x \quad op \quad y$ et la renvoie.
  - *op* est supposé être un caractère parmi $+, -, *$, et $/$.
- La fonction est dans l'incapacité de renvoyer un résultat dans deux cas :
  - si op n'appartient pas à l'ensemble $+, -, *, /$
  - si $op = /$ et $y = 0$
- Le deuxième cas est déjà prévu par java qui définit la classe ArithmeticException qui hérite de Exception.

Classe ErreurOperateur I

- Pour gérer l'erreur declenchée à la suite d'un opérateur inconnu, créons la classe ErreurOperateur :

```
1  class ErreurOperateur extends RuntimeException
   //RuntimeException derive aussi de Exception
3  {
   public ErreurOperateur(char op) // constructeur
5  {super("operateur␣inconnu␣" + op);}
   }
```

- La fonction evalue doit indiquer avec le mot clé throws qu'elle est susceptible de lancer des exceptions.

Classe ErreurOperateur II

```
    public static int evalue ( int x, char op, int y)
2   throws ErreurOperateur, ArithmeticException
    {
4   switch(op)
    {
6   case '+' : {return x + y; /*break;*/}
    case '-' : {return x - y; /*break;*/}
8   case '*' : {return x * y; /*break;*/}
    case '/' : {
10  if (y == 0) throw new ArithmeticException("division par
        zero");
    return x / y;
12              }
    default : throw new ErreurOperateur(op);//lancement d'
        exception
14  }
    }
```