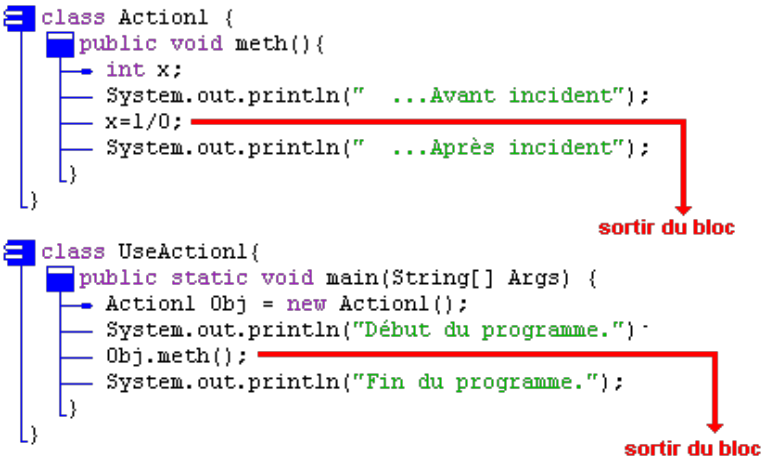
Les exceptions
et la programmation défensive

Les exceptions

- La programmation **défensive** est une attitude de pensée consistant à prévoir que le logiciel sera soumis à des défaillances dues à certains paramètres externes ou internes et donc à prévoir une réponse adaptée à chaque type de situation
- L’objectif principal est d’améliorer la qualité de "**robustesse**" d’un logiciel
- Une **exception** est chargée de signaler un comportement exceptionnel (mais prévu) d’une partie spécifique d’un logiciel ; Le mécanisme d’exceptions permet aussi d’**interrompre** proprement un programme

Exemples



int i=10 ;System.out.println(i/0);

⇒ **ArithmeticException**

System.out.println(t[-1]);

⇒ **ArrayIndexOutOfBoundsException**

String s = null; s.length();

⇒ **NullPointerException**

⇒ le programme s'arrête et la JVM signale une erreur

Conséquence

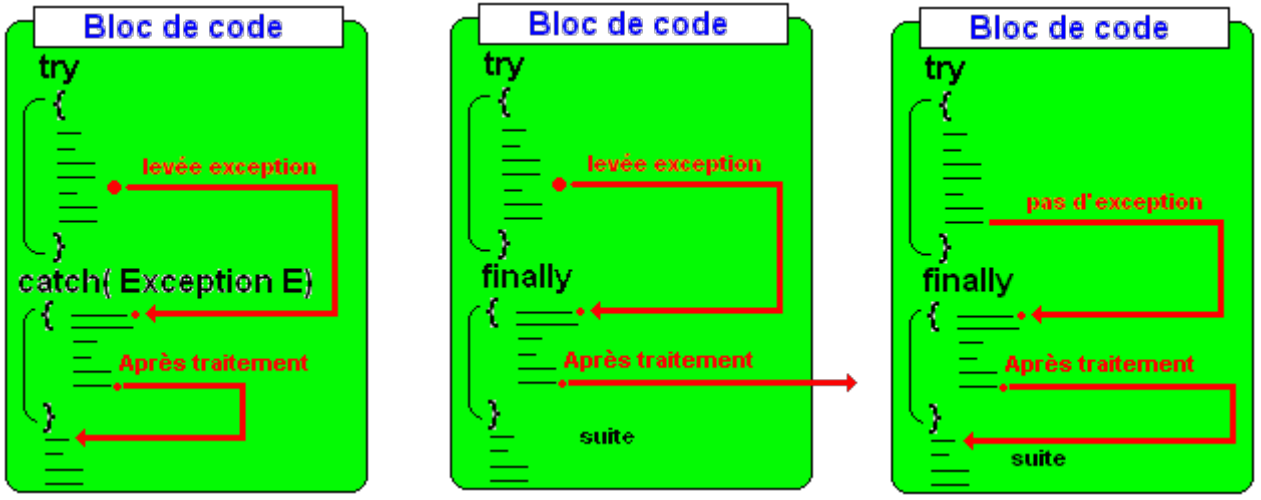
- Effet de vague :
  - Lorsqu’une exception se produit, l’instruction courante est interrompue
  - L’exécution de la méthode dans laquelle se trouvait l’instruction est également interrompue, elle ne renvoie aucune valeur
  - L’exception se propage en cascade jusqu’à la méthode initialement invoquée.
  - Finalement l’environnement reçoit l’exception et affiche le message associé
- Nous allons voir comment **intercepter** (on dit aussi "**attraper" - to catch**) cette exception afin de faire réagir notre programme afin qu'il ne s'arrête pas « brutalement »

Exemple

```
public static void main(String[] args) {
    int j = 20, i = 0;
    try {
      System.out.println(j/i);
      } catch (ArithmeticException e) {
            System.out.println("Division par zéro !");
            e.printStackTrace();
      }
   System.out.println("coucou ici !") ;
}
```

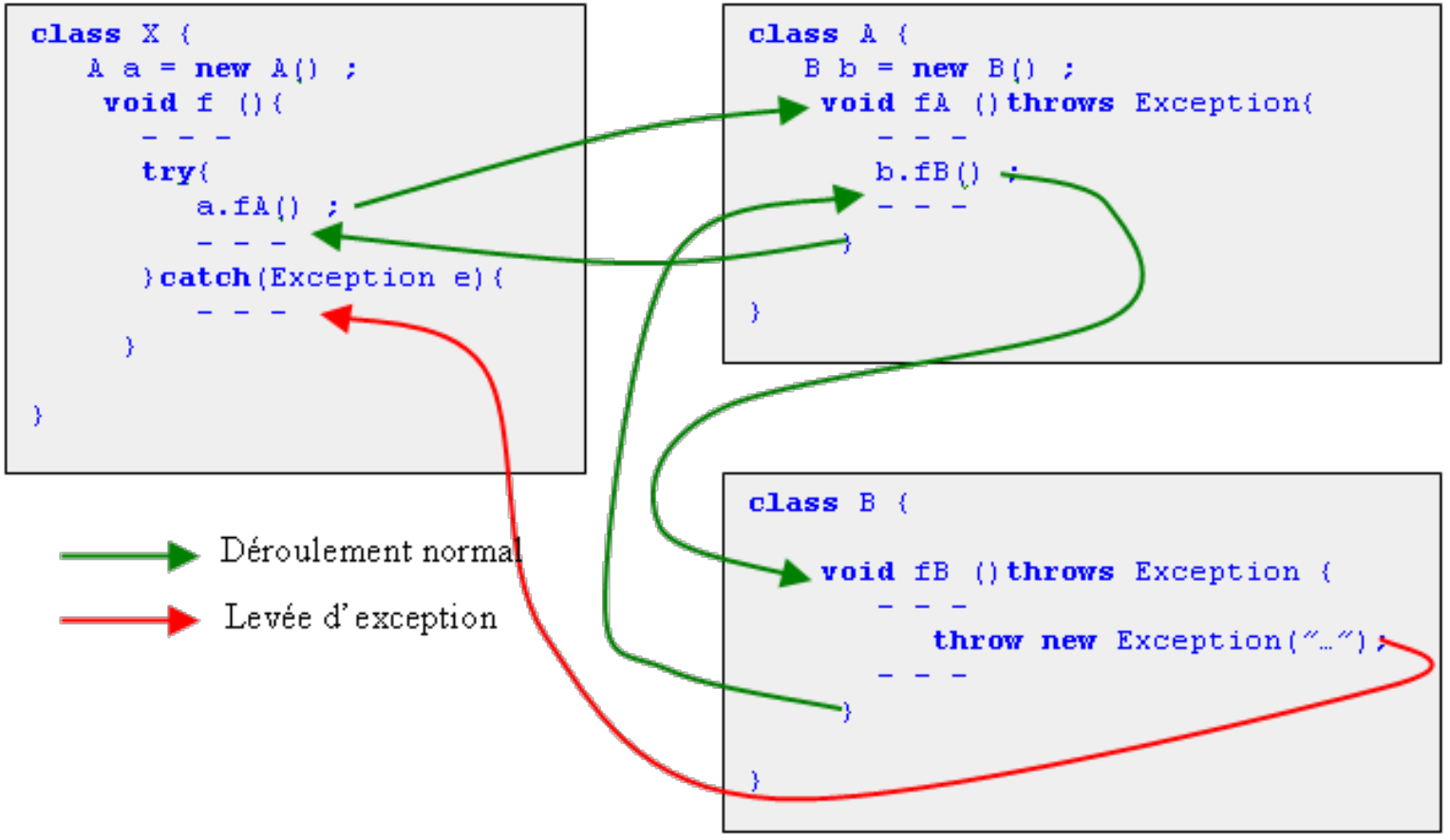
En général

```
try{
      I1 ;
      I2 ;
} catch(Exception1 e1){
       DuCodeJava...
} catch(Exception2 e2){
       UnAutreCodeJava...
} catch(Exception3 e3 | Exception4 e4) {
} finally{
            FaireToujoursCeCode
   }
```



La **clause finally** est toujours exécutée, qu'il y ait eu une levée d'exception ou non.

Elle est aussi exécutée si un catch ou le bloc try se termine par un return, un break ou un continue.


class X {
A a = new A() ;
void f (){
- - -
try{
a.fA() ;
- - -
}catch(Exception e){
- - -
}
}
class A {
B b = new B() ;
void fA ()throws Exception{
- - -
b.fB() ;
- - -
}
}
class B {
void fB ()throws Exception {
- - -
throw new Exception("…");
- - -
}
}
Déroulement normal
Levée d'exception

Exemples

```
class Action1 {
  public void meth(){
    int x;
    System.out.println("  ...Avant incident");
    x=1/0;
    System.out.println("  ...Après incident");
  }
}
```

engendre une exception

```
class UseAction1{
  public static void main(String[] Args) {
    Action1 Obj = new Action1();
    System.out.println("Début du programme.");
    try{
      Obj.meth();
    }
    catch(ArithmeticException E){
      System.out.println("Interception exception");
    }
    System.out.println("Fin du programme.");
  }
}
```

levée d'ArithmeticException

traitement, puis poursuite de l'exécution

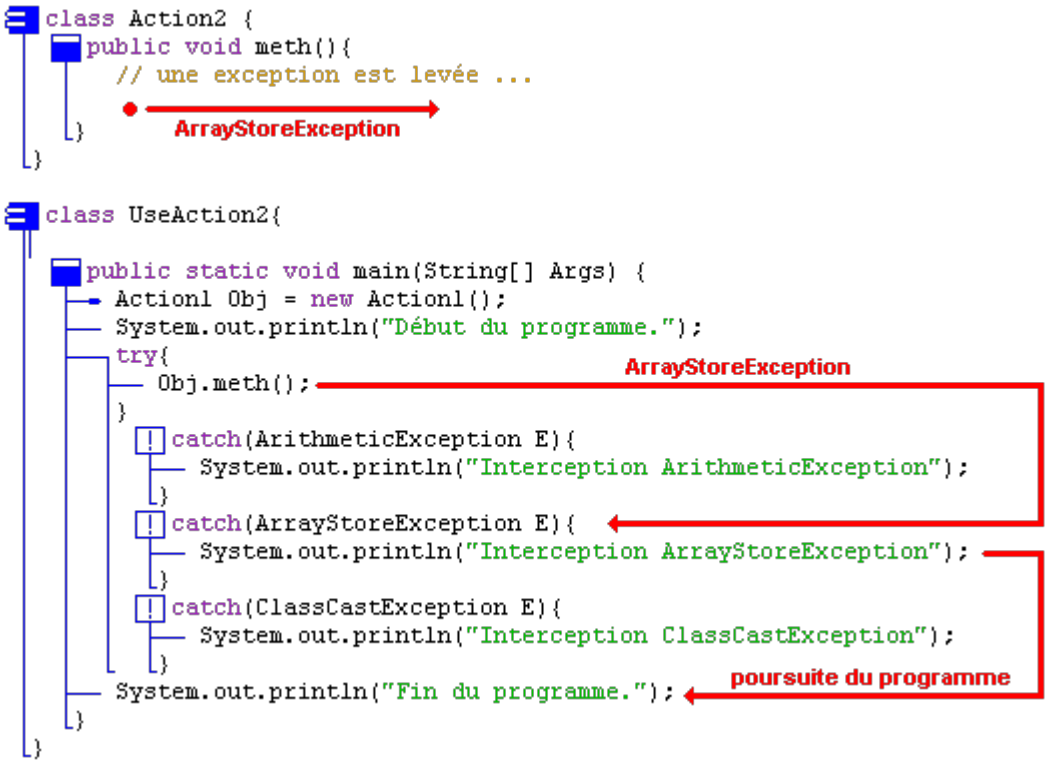
- Try {

  if (true) {a=i/0}

  else {a=t[-1]}

  }

  catch (Arith e | OutBoud e) { S.O.Pln(« coucou »)


class Action2 {
public void meth(){
// une exception est levée ...
}
}
ArrayStoreException
class UseAction2{
public static void main(String[] Args) {
Action1 Obj = new Action1();
System.out.println("Début du programme.");
try{
Obj.meth();
}
catch(ArithmeticException E){
System.out.println("Interception ArithmeticException");
}
catch(ArrayStoreException E){
System.out.println("Interception ArrayStoreException");
}
catch(ClassCastException E){
System.out.println("Interception ClassCastException");
}
System.out.println("Fin du programme.");
}
}
ArrayStoreException
poursuite du programme

Déclenchement manuel d'une exception existante

La JVM peut aussi **lever (déclencher)** une exception à votre demande suite à la rencontre d'une instruction **throw.**

```
class ArithmeticExceptionPerso extends ArithmeticException{
    ArithmeticExceptionPerso(String s){
        super(s);
    }
}

class Action3 {
    public void meth(){
        int x=0;
        System.out.println("  ...Avant incident");
        if (x==0)
            throw new ArithmeticExceptionPerso("Mauvais calcul !");
        System.out.println("  ...Après incident");
    }
}

class UseAction3{
    public static void main(String[] Args) {
        Action3 Obj = new Action3();
        System.out.println("Début du programme.");
        try{
            Obj.meth();
        }
            catch(ArithmeticExceptionPerso E){
                System.out.println("Interception exception : "+E.getMessage());
            }
        System.out.println("Fin du programme.");
    }
}
```

Créer ces propres exceptions

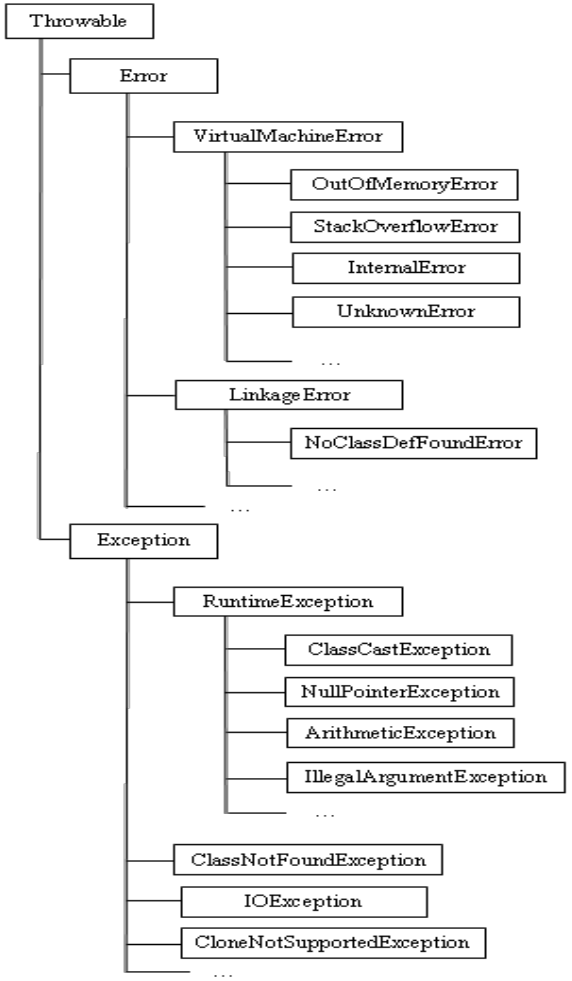
```
class NombreHabitantException extends Exception{
  public NombreHabitantException(){
    System.out.println("Nombre habitants (<0)!");
  }
}
```

Utilisation

```
public static void main(String[] args)
{
  Ville v = null;
  try {
    v = new Ville("Rennes", -1, "France");
  }
catch (NombreHabitantException | NomVilleException e2){
   System.out.println(e2.getMessage());
    e.printStackTrace();
 }
  finally{
    if(v == null)
      v = new Ville();
  }
  System.out.println(v.toString());
}
```

```
public Ville(String pNom, int pNbre, String pPays)
  throws  NombreHabitantException, NomVilleBidon
  {
    if(pNbre < 0)
      throw new NombreHabitantException(pNbre);
   if(pNom.length() < 3)
    throw new NomVilleException("le nom ville bidon »+ pNom);
    else
    {
      //Le code de construction d'une ville
    }
  }
```

Hiérarchie des exceptions




```
class UseAction2{
    public static void main(String[] Args) {
        Action1 Obj = new Action1();
        System.out.println("Début du programme.");
        try{
            Obj.meth();
        }
        catch(ArithmeticException E){
            System.out.println("Interception ArithmeticException");
        }
        catch(ArrayStoreException E){
            System.out.println("Interception ArrayStoreException");
        }
        catch(ClassCastException E){
            System.out.println("Interception ClassCastException");
        }
        catch(RuntimeException E){
            System.out.println("Interception RuntimeException");
        }
        System.out.println("Fin du programme.");
    }
}
```

La classe parent doit être placée après ses classes filles

Des méthodes avec exceptions

```
public static div(int x, int y) throws ArithmeticException
{
    if (y==0) {throw ArithmeticException(« pas bien »)
    else return x/y
}
```

Exceptions et assertions

- Quand l'assertion (une formule truc logique a tester dans le programme) dans le est violée, une exception est émise avec le message demandé par l'utilisateur

  L'instruction **assert (valeur >= 0 && valeur < limite);**

- Pourrait se traduire par

```
if (valeur < 0 || valeur >= limite) {
    // émettre une exception
}
```

- Les violations d'assertions sont plus graves que des exceptions, elles sont décrites par la classe Error au lieu de la classe Exception
- Utile pour la programmation et le test d'applications sensibles (avions, voiture, etc.)