Exception

- On ne peut pas toujours tester qu'un traitement ou une opération sera possible.
- Il faut alors tenter l'opération.
- Si elle échoue, une exception est levée.
- L'exception doit être prise en compte :
  - Propagée si on ne sait pas la traiter,
  - Traitée

Tenter et récupérer si nécessaire

```
try {
  //opération délicate
} catch( ExceptionType1 e1){
  e1.printStackTrace();
} catch( ExceptionType2 e2){
  System.err.println(e2.getMessage());
} catch( ExceptionType3 e3){
  throw e3; // propage l'exception
            // vers le bloc try/catch englobant
} finally {
  //traitement de fin de bloc try, toujours exécuté
}
```

exceptions ordonnées de la + pertinente à la - pertinente

Multi-catch

```
try {
  //opération délicate
} catch( ExceptionType1 | ExceptionType2 e1){
  System.err.println(e1.getMessage());
} catch( ExceptionType3 e2){
  throw e2; // propage l'exception
            // vers le bloc try/catch englobant
} finally {
  //traitement de fin de bloc try, toujours exécuté
}
```

try-catch avec ressources

```
try (Classe objet = new Classe(…)) {
  //instructions avec objet
}
```

à la fin du try-catch,
objet sera clos par un appel à la méthode close().
Classe doit implanter l'une des interfaces
AutoCloseable ou Closeable !

Levée d'exception

- Une méthode susceptible de lever des exceptions l'indique dans son entête :

```
type NomDeMethode(Arguments)
throws liste d'exceptions
```

- La levée proprement dite comporte :
  - Une instanciation d'exception
  - La propagation de l'exception

```
throw new ClasseException(Paramètres)
```

Interception d'exception

- Une méthode dont les traitements peuvent engendrer des exceptions doit :
  - Soit les propager (throws)
  - Soit les traiter (try/catch)

Exemple

```
public class CompteBancaire {
    private int solde;
    public int solde() { return solde; }
    public CompteBancaire() { solde = 0; }
    public void crediter(int montant)
    throws ExceptionMontantIncorrect {
      if (montant <= 0)
        throw new ExceptionMontantIncorrect(montant);
      else
        solde += montant;
    }
    public void debiter(int montant)
    throws ExceptionMontantIncorrect,
           ExceptionMontantTropGrand {
      if (montant <= 0)
        throw new ExceptionMontantIncorrect(montant);
      else if (montant > solde)
        throw new ExceptionMontantTropGrand(montant, solde);
      else
        solde -= montant;
    }
}
```

Exemple …

```
public class ExceptionMontantIncorrect
extends Exception {
  public ExceptionMontantIncorrect(int montant) {
    super("Montant incorrect (" + montant +
          ") : crédit ou débit d'un montant positif seulement.");
  }
}

public class ExceptionMontantTropGrand
extends Exception {
  public ExceptionMontantIncorrect(int montant, int solde) {
    super("Montant trop grand (" + montant +
          ") : supérieur au solde (" + solde + ").");
  }
}
```