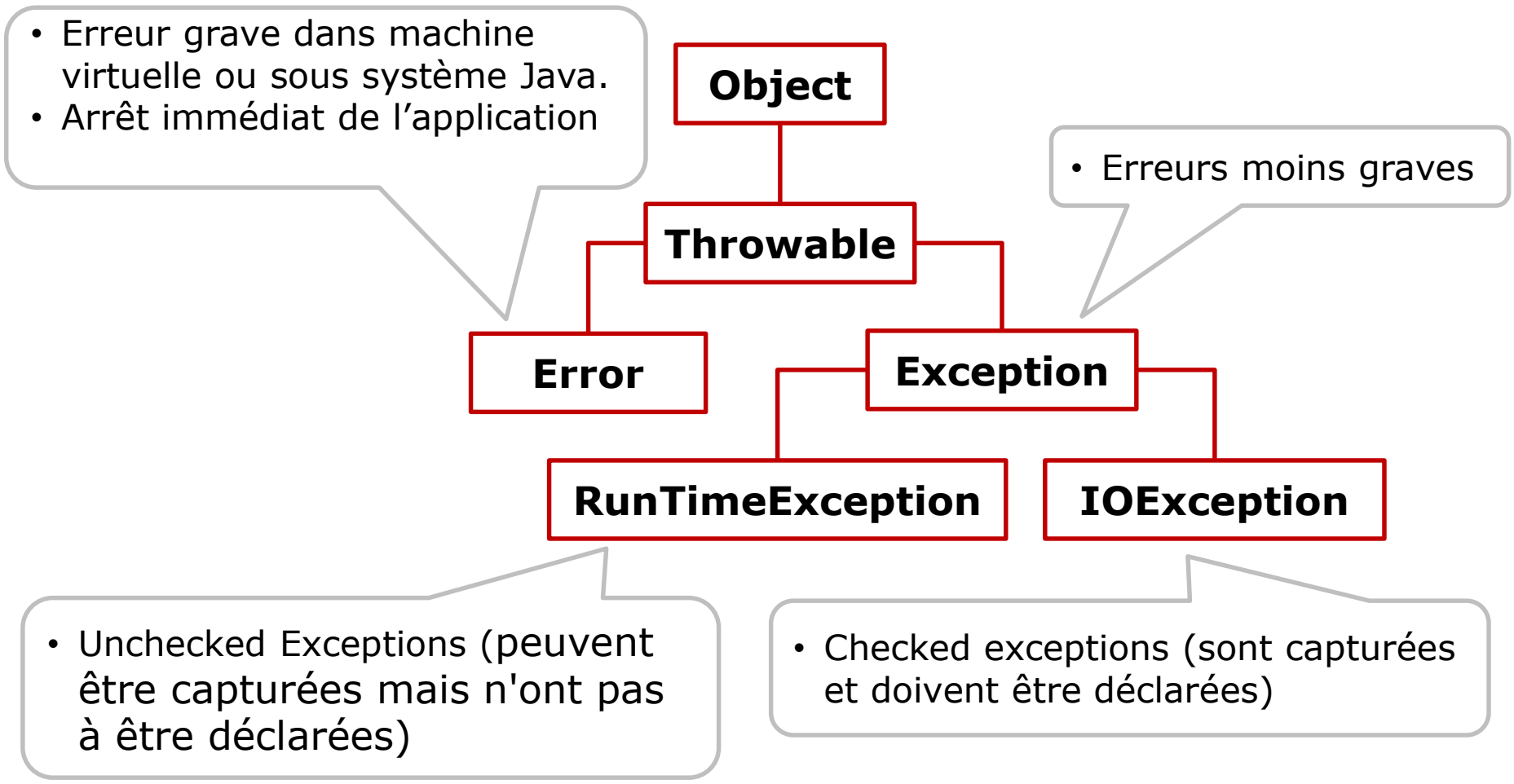
Chapitre 4

Gestion des exceptions

Introduction

- Une exception est une erreur qui se produit lors de l'exécution d'un programme.
- Le langage Java offre un mécanisme qui permet de :
  1. Isoler la partie du code générant l'erreur
  2. Dissocier la détection et le traitement de cette erreur.
- Lorsqu'une erreur est **détectée**, un objet Exception est créé : on dit qu'une exception est **levée**
- Lorsqu'elle est **traitée** on dit qu'elle est **capturée**.

Hiérarchie des exceptions



Rôle d’un objet exception

- Un objet exception fournit une description de l'erreur qui s’est produite
- Son rôle est de :
  - Signaler qu’une erreur s’est produite
  - Fournir un message décrivant l’erreur
  - Fournir le numéro de la ligne où l’erreur s’est produite
  - Fournir l’enchainement des appels des fonctions ou l’erreur s’est produite (pile des appels ou Stack trace)

Rôle d'un objet exception

- La classe exception
  - Le Constructeur Exception(String message): prend en paramètre le message à afficher en cas d'erreur
  - Quelques méthodes héritées de Throwable:
    - String **getMessage( )** : retourne le message d'erreur
    - String **toString()** : retourne le nom de l'exception et le message d'erreur
    - void **printStackTrace( )** : affiche le nom de l'exception, le message d'erreur et l'enchainement des appels des fonctions avec les numéros des lignes de code ou se trouve l'erreur.

Hiérarchie des exceptions

- **Exemple**: cas de division par zéro
  - getmessage : / by zero
  - toString : java.lang.ArithmeticException: / by zero
  - printStackTrace : java.lang.ArithmeticException: / by zero
    at Main.main(Main.java:24)

Exceptions prédéfinis

- Java dispose d'un ensemble d'exceptions prédéfinies
- La plupart appartiennent au package java.lang et héritent de la classe RuntimeException
- Exemples :
  - ArithmeticException : erreurs arithmétiques comme dans la division par zéro
  - ArrayIndexOutOfBoundsException : index d'un tableau endehors de ses limites
  - NullPointerException : tentative d'utiliser un objet null
  - ClassCastException : Cast invalide.

Exceptions personnalisées

- Il existe des situations ou nous aimerions détecter un type personnalisé d'erreurs
- Exemple :
  - Une note en dehors de l'intervalle [0,20]
  - Un nom ne respectant pas une forme particulière
  - …etc
- Dans ce cas il faut définir une classe pour cette exception.
- Cette dernière dois hériter de la classe **Exception**.

Exceptions personnalisées

- Exemple :
  - Classe Exception pour Une note en dehors de l'intervalle [0,20]:

```
public class NoteDebordanteException extends Exception {

    public NoteDebordanteException (String message){
        super(message);
    }
}
```

Lever une exception

- Dans une méthode, la détection d'une erreur est faite grâce au code suivant : **throw new ClasseDeLexception()**
- Cette méthode doit aussi l'indiquer dans sa signature comme suit : **throws ClasseDeLexception**
- Pour permettre d'informer les méthodes appelantes
- Exemple :

```
public int diviser(int x, int y)   throws    ArithmeticException   {
        if (y==0)
                throw new ArithmeticException();
        return (x/y);
}
```

Lever une exception

- Remarque :
  - Une méthode peut détecter plusieurs exceptions et doit les déclarer dans son en-tête comme suit :
  - …methode (arguments) throws Exception1, Exception2,..{ }

Lever une exception personnalisée

- Exemple d'Utilisation :

```
public void setNote (float note) throws NoteDebordanteException {
 if ((note<0)||(note>20))
     throw new NoteDebordanteException(" Erreur  note qui
     déborde" );
 this.note=note;
}
```

Comportement face à une exception

- Soit une méthode *methode1()* qui lève une exception
- Et soit une méthode *methode2()* appelant *methode1()*
- Methode2 a deux choix :
  1) **Ne pas traiter l'exception** : Dans ce cas elle doit déclarer l'exception dans sa signature avec « throws »
  2) **Capturer ou traiter l'exception** : Dans ce cas elle doit recourir à try/catch

Capture d'une exception

- La gestion d'une exception se fait selon le schéma suivant :

```
try{
   appel de la fonction susceptible de générer  l'exception
}
catch (Exception e){
   traiter l'exception e
}
instruction suivante
```

- Remarque 1 :
  - En cas d'exception les instructions qui suivent le lancement de l'exception sont ignorées.

Capture d'une exception

- Remarque 2 :
  - Le block Catch() est aussi facultatif.
  - s'il est omis, dans ce cas, si une exception survient dans le bloc try, elle se propagera dans la pile d'appels, et il incombera au code appelant ou à un gestionnaire d'exceptions de plus haut niveau de la traiter.

Capture d'une exception : Exemple

- Soit Etudiant une classe composée de :
  - Attributs : nom de type String et moyenne de type float
  - Constructeur paramétré
  - Getters et setters
  - toString()
- Ecrire une Classe *MoyenneDebordanteException* permettant de signaler la non appartenance de la note à l'intervalle [0,20]
- Ecrire une classe Main pour le test.

```
public class MoyenneDebordanteException extends Exception {
   public MoyenneDebordanteException(String msg) {
      super(msg);
   }
}
```

Capture d'une exception : Exemple

```
public class Etudiant {
 private String nom;
 private float moyenne;
 public Etudiant(String nom, float moyenne) throws MoyenneDebordanteException {
   if ((moyenne<0)||(moyenne>20))
      throw new MoyenneDebordanteException("La moyenne déborde de l'intervalle 0,20");
  this.nom = nom;
  this.moyenne = moyenne;
 }
 public String getNom() { return nom; }
 public void setNom(String nom) {this.nom = nom;}
 public float getMoyenne() {return moyenne;}
 public void setMoyenne(float moyenne) throws MoyenneDebordanteException {
  if ((moyenne<0)||(moyenne>20))
    throw new MoyenneDebordanteException("La moyenne déborde de l'intervalle 0,20");
  this.moyenne = moyenne;
 }
 @Override
 public String toString() {
    return "Etudiant{ nom = " + nom + ", moyenne=" + moyenne + '}';
 }
}
```

Capture d'une exception : Exemple

```
public class Main {
    public static void main(String[] args) {
        try {   Etudiant etudiant1=new Etudiant("Omar",16);
                System.out.println(etudiant1.toString());
                Etudiant etudiant2=new Etudiant("Ali",22);
                System.out.println(etudiant2.toString());
        } catch (MoyenneDebordanteException ex) {
                ex.printStackTrace();
        }
    }
}
```

- Résultat de l'exécution :

```
Etudiant{nom=Omar, moyenne=16.0}
testexceptions.MoyenneDebordanteException: La moyenne déborde de l'intervalle 0,20
        at testexceptions.Etudiant.<init>(Etudiant.java:7)
        at testexceptions.Main.main(Main.java:7)
BUILD SUCCESSFUL (total time: 1 second)
```

Capture de plusieurs exceptions

- Cas d'une instruction levant plusieurs exceptions :

```
try {
fonction susceptible de générer l'exception }
catch (Exception1 e){ traitement 1 }
catch (Exception2 e){traitement 2 }
…
catch (ExceptionN e){traitement N }
instruction suivante
```

Capture de plusieurs exceptions

- Depuis la version 7 de Java, il est possible de réécrire le code précédent sous la forme :

```
try {
fonction susceptible de générer l'exception }
catch (Exception1 | ... | ExceptionN e){
   traitement ;
}
instruction suivante
```

- Remarque : Dans catch il faut commencer par exceptions les plus précises en premier sinon

  Un message d'erreur est émis par le compilateur.

Capture de plusieurs exceptions

- Cas de plusieurs instructions levant plusieurs exceptions :

```
try {   }
catch (){  }
try {   }
catch (){  }
try {   }
catch (){  }
…
```

Le bloc finally

- Certaines ressources peuvent rester non libérée après qu'une exception sois levée.
- Pour forcer le compilateur à exécuter certaines instructions, qu'il y ait exception ou non, ils doivent être placées dans un bloc **finally**.
- Dans ce cas le bloc finally est exécuté avant que l'exception soit capturée.
- Remarque :
  - Ce bloc est facultatif !

Le bloc finally

- Structure

```
try {
 }
catch (){
}
finally {
}
instruction suivante
```

Try avec ressources

- La déclaration « Try avec ressources » ou **try-with-resources** permet de simplifier la gestion des ressources en fermant automatiquement les ressources qui implémentent l'interface AutoCloseable ou Closeable.
- Cette déclaration aide à éviter les fuites de ressources et rend le code plus propre et plus lisible.

try-with-resources

- Syntaxe:

```
try (TypeDeRessource ressource1 = new TypeDeRessource();
TypeDeRessource ressource2 = new TypeDeRessource()) {
   // Code utilisant les ressources
} catch (TypeException e) {
   // Code pour gérer les exceptions
}
```

try-with-resources

- Dans cette syntaxe :
  - TypeDeRessource est le type de ressource que vous souhaitez gérer (comme les fichiers par exemple)
  - Les ressources sont déclarées entre parenthèses après le mot-clé try.
  - Ces ressources doivent implémenter l'interface AutoCloseable ou Closeable, qui définit une méthode close().
  - Lorsque le bloc try est quitté, les ressources sont automatiquement fermées.

try-with-resources

- Exemple :

```
import java.util.Scanner;
public class ExempleScanner {
    public static void main(String[] args) {
        // Utilisation de try-with-resources avec Scanner
        try (Scanner scanner = new Scanner(System.in)) {
            System.out.print("Entrez votre nom: ");
            String nom=scanner.next();
            System.out.println("Bonjour: " + nom);
        } catch (Exception e) {
            // Gère les exceptions, le cas échéant
            System.err.println("Erreur : " + e.getMessage());
        }
    }
}
```

Chaînage des exceptions

- Dans le traitement d'une exception on ne se contente pas que d'afficher le message d'erreur.
- Souvent on a recours à un traitement durant lequel nous faisons appel à une instruction qui peut lever une autre exception.
- Cette nouvelle exception doit être liée avec l'exception d'origine pour conserver l'empilement des exceptions levées durant les traitements.

Chaînage des exceptions

- Il y a deux façons de chaîner deux exceptions :

1) Utiliser le constructeur de Throwable qui attend un paramètre Throwable représentant la cause.

Exemple :

```
catch(Exception1 e1) {
    ....
            throw new Exception2(e1);
   // ou throw new Exception2 ("un message" , e);
 }
```

2) Utiliser la méthode initCause() d'une instance de Throwable

Exemple :

```
catch(Exception1 e1) {
            throw (Exception2) new Exception2().initCause(e1);
}
```

Chaînage des exceptions

- **Exemple** :

```
package testchainageexception;

public class MonException extends Exception{
    // constructeur 1
    MonException(String message){
            super(message);
    }
   // constructeur 2
    MonException(String message , Throwable cause){
            super(message, cause);
    }
}
```

Chaînage des exceptions

```
package testchainageexception;

public class TestChainageException {

   public static int Division (int x, int y) throws MonException{

      int resultat=0;

      try { resultat=x/y; }

      catch(ArithmeticException e){

         throw new MonException("Attention Il y a une exception !!",e);

      }

      return resultat;

      }
```

Chaînage des exceptions

```
public static void main(String[] args) {
        int a=3,b=0,z;
        try { z=Division(a, b); }
        catch(MonException e){
            e.printStackTrace();
            // e.getCause().printStackTrace(); pour n'afficher que la cause
        }
    }
}
```

Chaînage des exceptions

- Résultat :

testchainageexception.MonException: Attention Il y a une exception !! **at** testchainageexception.TestChainageException.Division(TestChainageException.java:27) **at** testchainageexception.TestChainageException.main(TestChainageException.java:15)

Caused by: java.lang.ArithmeticException: / by zero **at** testchainageexception.TestChainageException.Division(TestChainageException.java:25)

... 1 more