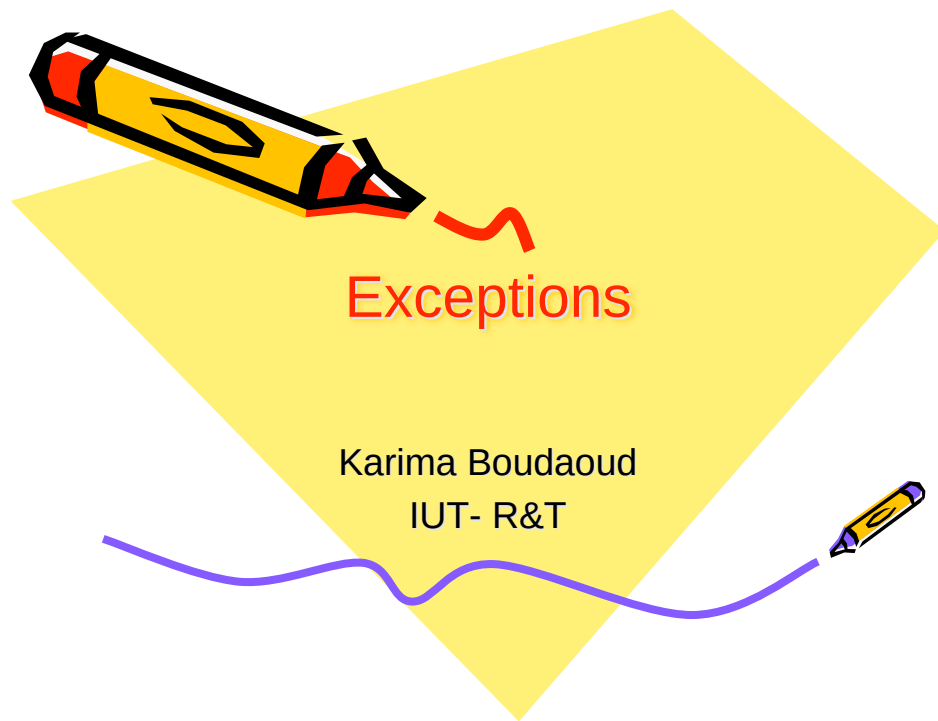




# Exceptions et Erreurs

## ○ Définition

- les exceptions/erreurs sont utilisées pour traiter un fonctionnement anormal d'une partie d'un code (provoqué par une erreur ou un cas exceptionnel)
  - ✓ Plus de mémoire libre
  - ✓ Mauvais URL
  - ✓ Division par 0
  - ✓ Référence null...
- en java, une erreur ne provoque pas l'arrêt brutal du programme mais la création d'un objet, instance d'une classe spécifiquement créée pour être associée à des erreurs/exceptions



Karima Boudaoud

IUT GTR-Sophia Antipolis

2

## L'approche maladroite (1)

- code non-spécialisé
  - ✓ où le traitement des erreurs est mélangé dans le code
- pour le renvoi de condition d'erreur
  - ✓ comment retourner condition et résultat ?
- mais cela est difficile à comprendre
- et difficile à maintenir



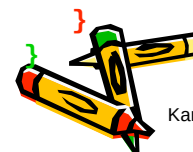
Karima Boudaoud

IUT GTR-Sophia Antipolis

3

## L'approche maladroite (2)

```
int codeErr = ouvrirFichier();
if (codeErr == FICHIER_INEXISTANT) { /*traiter l'erreur */}
else if (codeErr==OUVERTURE_INTERDITE){ /*traiter l'err */}
else { // ok, fichier ouvert
    codeErr = lireFichier();
    if (codeErr == ERREUR_DE_LECTURE) { /* traiter l'erreur */}
    else { // ok, on l'a lu
        codeErr = testerFormat();
        if (codeErr == FORMAT_INCONNU) { /* traiter l'erreur */}
        else { // c'est bon, continuons
            ...
        }
    }
}
```



Karima Boudaoud

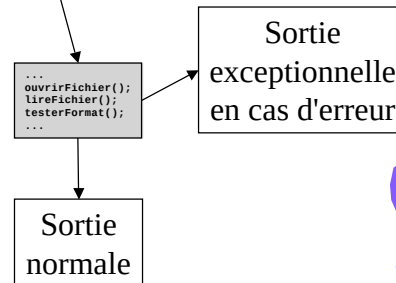
IUT GTR-Sophia Antipolis

4

## Spécialisation du Code (1)

- le traitement des exceptions en Java inspiré par C++, pour traiter un fonctionnement anormal
- en java, la java virtual machine (JVM) fait le travail
- pour gérer les exceptions, il existe des structures spécialisées

- ✓ **try {...}**  
qui indique risque d'erreur
- ✓ **catch () {...}**  
qui attrape l'erreur
- ✓ **throw ()**  
qui lève l'erreur



Karima Boudaoud

IUT GTR-Sophia Antipolis

5

## Spécialisation du Code (2)

```
try {  
    ouvrirFichier();  
    lireFichier();  
    testerFormat();  
}  
catch(FichierInexistant e) { /*traiter l'erreur */}  
catch(OuvertureInterdite e) { /* traiter l'erreur */}  
catch(ErreurDeLecture e) { /* traiter l'erreur */}  
catch(FormatInconnu e) { /* traiter l'erreur */}
```



Karima Boudaoud

IUT GTR-Sophia Antipolis

6

## Vocabulaire sur les erreurs/exceptions

- une instruction, une méthode peut lever ou lancer une exception : une anomalie de fonctionnement provoque la création d'une exception
- une méthode peut attraper, saisir, traiter une exception par une clause **catch** d'un bloc **try-catch**
- une méthode peut laisser se propager une exception :
  - ✓ elle ne l'attrape pas par une clause **catch**
  - ✓ l'erreur «remonte» alors vers la méthode appelante qui peut elle-même l'attraper ou la laisser remonter



Karima Boudaoud

IUT GTR-Sophia Antipolis

7

## Signature d'une méthode

- comprend aussi le traitement d'exceptions
  - **public static void sleep(long ms)**  
**throws InterruptedException**
- code *doit* traiter l'exception

```
try {Thread.sleep(1000);}
catch(InterruptedException e) {
    System.err.println("Sommeil interrompu");}
```

- sinon...
  - FileWithGetLine.java:8: Warning: Exception java.io.InterruptedExcepion must be caught, or it must be declared in throws clause of this method.



Karima Boudaoud

IUT GTR-Sophia Antipolis

8

## Traitement (1)

- lorsqu'une Exception est levée...
  - ❑ le traitement linéaire s'arrête
  - ❑ la JVM récupère la main
    - ✓ la JVM cherche à passer le contrôle
- deux possibilités
  - ❑ traitement « sur place » si la méthode en est capable
    - ✓ **catch** suit **try**
  - ❑ traitement « reporté »
    - ✓ méthode rejette la responsabilité
    - ✓ indiqué dans sa signature par **throws**
    - ✓ possibilité de traitement par une méthode plus douée



Karima Boudaoud

IUT GTR-Sophia Antipolis

9

## Traitement (2)

```
...
try {
    k = foo();
} catch (Exception e) {
    // traitement de
    // l'exception
}
...

void foo() throws Exception {
    ...
    if (...) {
        throw new Exception();
    };
    // code jamais
    // exécuté si exception
    ...
}
```



La JVM récupère  
la main et passe  
le contrôle

Karima Boudaoud

IUT GTR-Sophia Antipolis

10

## Traitement sur place (1)

```
try {
    int i = m();
    // à exécuter si m() n'a pas levé
    // d'exception
    . . .
} catch (TrucException e) {
    // traitement de l'exception e levée
    // dans la méthode m
    . . .
}
```



Karima Boudaoud

IUT GTR-Sophia Antipolis

11

## Traitement sur place (2)

- le traitement général des exceptions se fait de la manière suivante :

```
try {
    /* code à risque */
} catch (ClasseException1 e) {
    ...
} catch (ClasseException2 e) {
    ...
}
```

- le contrôle ne passe que dans **un seul** bloc **catch**
  - ✓ qui correspond à la première exception levée



Karima Boudaoud

IUT GTR-Sophia Antipolis

12



## Erreur du débutant



```
try {
    int n = ParseInt(args[0]);
    etagere.add(livres[n]);
} catch (NumberFormatException e) {
    System.err.println("Mauvais numéro de livre");
} catch (EtagerePleineException e) {
    System.err.println("Etagère pleine : déjà "
        + e.etagere.contenance() + " livres");
}
n++; // provoque une erreur à la compilation
```



Karima Boudaoud

IUT GTR-Sophia Antipolis

13

## Traitement reporté



- c'est pas moi, c'est l'autre...

```
public maMéthode(long ms)
    throws InterruptedException {
    ...
    Thread.sleep(millis); // ni try ni catch
    ...
}
```

- la JVM prend la main
  - ✓ cherche **catch** correspondant à l'exception
  - ✓ remonte la pile de méthodes



Karima Boudaoud

IUT GTR-Sophia Antipolis

14

## Traitement par défaut



- si aucune méthode ne traite l'exception, la JVM
  - ❑ affiche un message d'erreur
    - ✓ avec la pile des méthodes traversées par l'exception
  - ❑ stoppe l'exécution du *thread*
    - ✓ peut planter la JVM



Karima Boudaoud

IUT GTR-Sophia Antipolis

15

## Création d'exception (1)



- dérivation de **java.lang.Exception**

```
class MonException extends Exception {}
class MaClasse {
    ...
    public void maMéthode() throws MonException {
        ...
        throw new MonException();
        ...
    }
    ...
}
```



Karima Boudaoud

IUT GTR-Sophia Antipolis

16

## Création d'exception (2)

- par convention, nom se termine par **Exception**

```
class EtagerePleineException
    extends Exception {
    private Etagere etagere;
    EtagerePleineException(Etagere et) {
        etagere = et;
    }
    Etagere getEtagere() {
        return etagere;
    }
}
```

Karima Boudaoud

IUT GTR-Sophia Antipolis

17

## Utilisation de l'exception

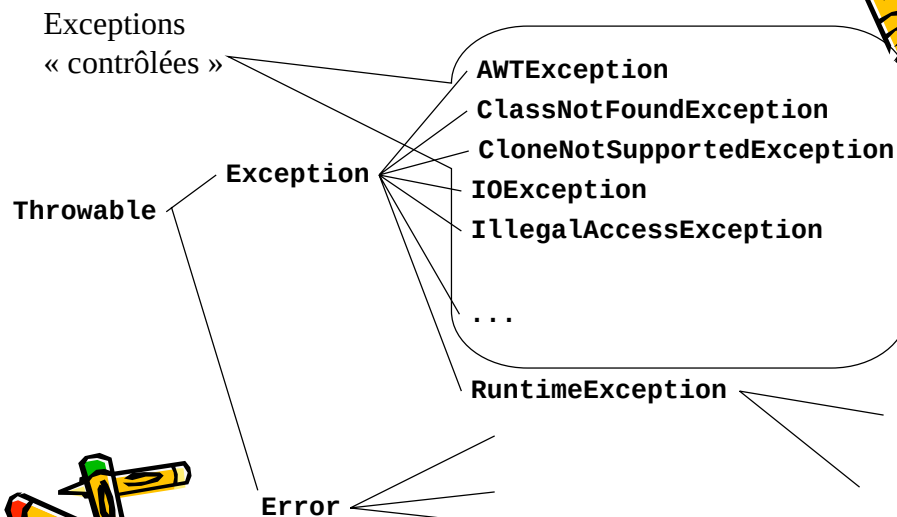
```
public class Etagere {
    int nbLivres = 0,
    contenanceEtagere = 0;
    . . .
    public void ajouterLivre(Livre
    livre) throws EtagerePleineException {
        if (nbLivres == contenanceEtagere) {
            throw new EtagerePleineException(this);
        }
        else // ajout du livre dans l'étagère
            . . .
    }
}
```

Karima Boudaoud

IUT GTR-Sophia Antipolis

18

## Structure hiérarchique



Karima Boudaoud

IUT GTR-Sophia Antipolis

19

## Exceptions Contrôlées

- Exceptions dites "*non-runtime*"
  - ☐ Prévisibles et rares
  - ☐ Traitement est rentable
  - ☐ Traitables par l'utilisateur
    - ✓ héritent de la classe `java.lang.Exception`
    - ✓ pas de `java.lang.RuntimeException`
- toute méthode qui peut lancer une exception contrôlée *doit* la déclarer dans sa signature

```
public int m(double d) throws TrucException
```

Karima Boudaoud

IUT GTR-Sophia Antipolis

20

## Méthodes de la classe Exception

- **toString()**

- ✓ existe pour tous les objets Java

```
catch(ArithmeticException e) {  
    System.out.println("exception : "  
        + e + " rencontrée");  
}
```



Karima Boudaoud

IUT GTR-Sophia Antipolis

21

## Méthodes de la classe Throwable

- **printStackTrace()**

- ✓ imprime les appels ayant conduit à lever l'exception
- ✓ appelée par défaut lors de l'acquittement d'une exception par java

```
catch(ArithmeticException e) {  
    e.printStackTrace()  
}
```

- affiche les informations suivantes :

```
java.lang.ArithmeticException: / by zero  
    at Vector.normalize(Vector.java: 25)  
    at Vector.getMatrix(Vector.java: 140)...
```

- **StackTraceElement[] getStackTrace()**



Karima Boudaoud

IUT GTR-Sophia Antipolis

22

## Exceptions RuntimeException

- trop fréquentes
  - ✓ traitement n'en vaut pas la peine
- par exemple...
  - ✓ **ArithmeticException**
  - ✓ **ArrayIndexOutOfBoundsException**
  - ✓ **NullPointerException**
  - ✓ ...



Karima Boudaoud

IUT GTR-Sophia Antipolis

23

## Erreurs : java.lang.Error

- les erreurs indiquent des problèmes dans le système
- elles ne sont pas traitables par l'utilisateur
- elle peuvent être ignorées (par l'utilisateur)



Karima Boudaoud

IUT GTR-Sophia Antipolis

24



## Erreur du débutant

```
try {  
    ...  
} catch (Exception e) { // attrape tout  
    ...  
} catch (NumberFormatException e) { // jamais ici  
    ...  
} catch (EtagerePleineException e) { // jamais ici  
    ...  
}
```



## Après l'exception (1)

- le bloc **finally{}**
  - ☐ exécuté quoiqu'il arrive
    - ✓ aucune exception
    - ✓ exception traité dans catch
    - ✓ exception traité ailleurs

```
...  
try { uneMéthodeQuiJette(); }  
catch (ExceptionJettée e) {...}  
finally { /* fait le ménage dans tous les cas */ }  
...
```



## Après l'exception (2)

- le bloc **finally{}**
  - ☐ utilisé principalement pour fermer des ressources
    - ✓ fichiers ouverts
    - ✓ sockets
    - ✓ ...



## Exception ou Traitement Normal ?

- éviter d'utiliser une exception à la place d'un traitement normal
  - ✓ Mauvais style
  - ✓ Coûteux
- exemples :
  - ✓ Si on lit un fichier du début à la fin, ne pas utiliser **EOFException**
  - ✓ Si on rencontre la fin du fichier avant d'avoir lu les 10 valeurs dont on a besoin, utiliser **EOFException**



# Exception et Constructeurs

- lancer une exception si un constructeur n'a pu faire correctement son travail
  - ✓ vaut mieux une exception à la création qu'un objet défectueux créé

```
public Toto() throws TotoNotCreatedException
{
    ...
    if (pb) {
        throw new TotoNotCreatedException();
    }
}
```



# Quel Type d'Exception ?

- faut pas lancer des erreurs (classe **Error**)
- pour obliger une méthode à traiter le problème...
  - ☐ lancer une exception
    - ✓ mais qui n'est pas dérivée de **RuntimeException**
  - ☐ lancer l'exception la plus spécifique possible au problème

