

COURS DE CONCEPTION ORIENTÉE OBJET LES INTERFACES ET EXCEPTIONS



Présentée par: Mme HASSAM-OUARI Kahina

Email: kahina.hassam@junia.com

Bureau: T336

Département: SSE(Smart Systems and Energies)

JUNIA HEI

Contexte

- Une classe définit:

- La structure des objets (données, les traitements)
- Le code des traitements (des méthodes)
- La façon dont les objets sont créés (le constructeur de la classe)

Utilisateur



Développeur

Vue
utilisateur

- Les méthodes qu'il peut utiliser sur les objets.
- Le nombre de paramètres des méthodes ainsi que leurs types

Les interfaces

- Une interface permet de définir un ensemble de traitements
- Une interface Java est comme une classe 100% abstraite.
- Comme toutes les méthodes sont abstraites, toute classe qui implémente une interface doit redéfinir toutes les méthodes de cette interface.

Déclarer une première interface

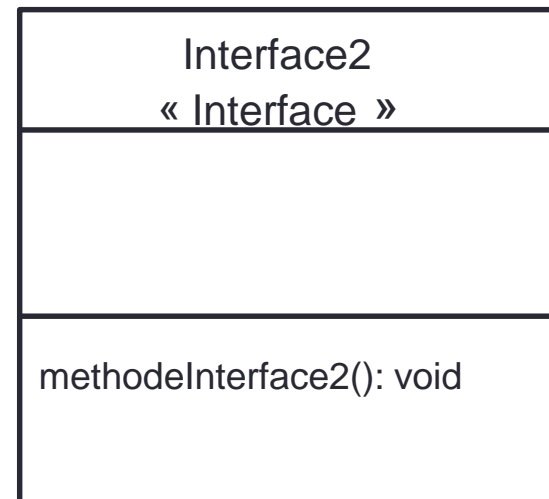
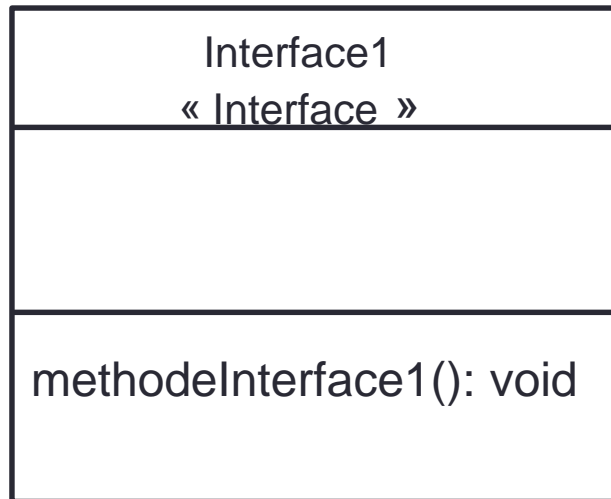
Videothequelf « Interface »
AjouterDvd(String Dvd): void emprunterDvd(): Boolean

```
public interface VideothequeIf{  
    }
```

- Vu qu'une interface est comme une classe 100 % abstraite, il ne vous reste qu'à y ajouter des méthodes abstraites, mais **sans le mot clé abstract**.

```
public interface VideothequeIf{  
    public void AjouterDvd(String Dvd); //Sans  
    {} qui définir le corps de la méthode  
    public boolean emprunterDvd();  
}
```

Exemple



Implémenter une interface

- Pour qu'une classe utilise une interface, il suffit d'utiliser le mot clé **implements**

```
public class MaClasse implements Interface1 {  
    public void methodeInterface1(){  
        //...mettre le code de la methode 1  
    }  
}
```

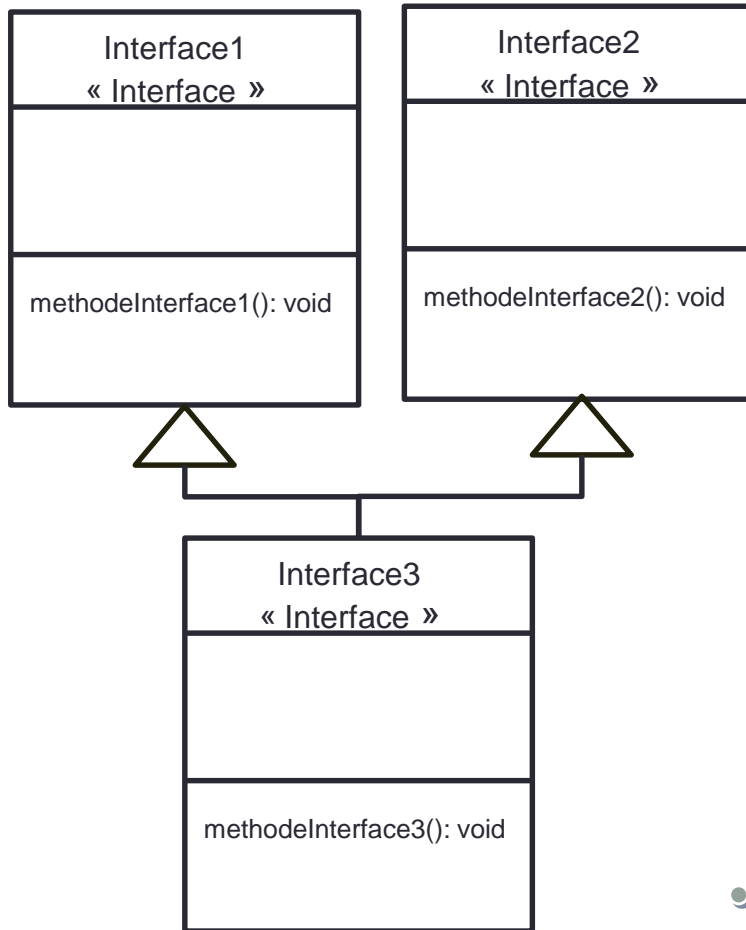
- Lorsque une classe implémente une interface, elle doit obligatoirement redéfinir toutes les méthodes de l'interface implémentée

Contourner l'héritage simple

- Héritage en Java, OUI, Mais Héritage simple.
- Pour faire de l'héritage multiple, il faut qu'une classe implémente plusieurs interfaces.

```
public class MaClasse implements Interface1, Interface2 {  
    public void methodeInterface1(){  
        //...mettre le code de la methode 1  
    }  
    public void methodeInterface2(){  
        //...mettre le code de la methode 2  
    };  
}
```

Héritages et interfaces



```
public interface Interface1 {  
    public void methodeInterface1();  
}
```

```
public interface Interface2 {  
    public void methodeInterface2();  
}
```

```
public interface Interface3 extends  
Interface1, Interface2{
```

```
    public void methodeInterface3();
```

```
}
```

- L'héritage multiple entre interfaces est autorisé

Classe abstraite, sous-classe et interface : faire le bon choix

- Créer une classe simple qui n'étend rien quand elle ne réussit pas le test du EST-UN (Est une sorte de) pour tout autre type.
- Créer une sous-classe uniquement si vous voulez obtenir une version plus spécifique d'une classe, redéfinir ou ajouter des comportements.
- Utiliser une classe abstraite quand vous voulez définir un patron pour un groupe de sous-classes et qu'une partie du code est utilisée par toutes les sous-classes.
- Utiliser une interface pour définir un rôle que d'autres classes puissent jouer, indépendamment de leur place dans la structure d'héritage.

Ce que nous avons vu ...

- Quand vous ne voulez pas qu'une classe soit instanciée, utilisez le mot clé `abstract`.
- Une classe abstraite peut avoir des méthodes abstraites ou non abstraites (Concrètes).
- Une classe qui contient au moins une méthode abstraite doit être abstraite.
- Une méthode abstraite n'a pas de corps. Elle se termine par un `;"`.
- Toute méthode abstraite doit être implémentée par la première classe concrète dans la hiérarchie d'héritage.

Ce que nous avons vu ...

- Toute classe Java est une sous-classe directe ou indirecte de la classe Object.
- Java n'autorise pas l'héritage multiple.
- Une interface est comme une classe 100% abstraite. Une classe peut implémenter plusieurs interfaces (mot clé **implements**).
Toutes les méthodes d'une interface doivent être redéfinies.

En résumé:

Interface OU classe abstraite

Interface

- Décrit la signature des méthodes utilisées

Classe abstraite

- Propriétés et méthodes (avec ou sans code)
- Certaines méthodes doivent être codés dans d'autres classes



Vue utilisateur



Vue developpeur

Exercice de cours

- Certains animaux peuvent crier, d'autres sont muets. On représentera le fait de crier au moyen d'une méthode affichant à l'écran le cri de l'animal.
- Créer une interface « Criant » contenant la méthode permettant de crier.
- Ecrire les classes des chats, des chiens et des lapins et qui implémentent l'interface « Criant » redéfinissez la méthode pour chaque animal.
- Ecrire un programme avec un tableau ou ArrayList, que vous remplissez d'animaux, le remplir avec des chiens et des chats, puis faire crier tous ces animaux.

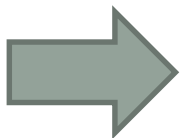
Les exceptions

Introduction

- Il arrive que des erreurs se produisent dans des programmes Java, **erreurs indépendantes de la volonté du programmeur**.
- Le développeur devra gérer différents cas dans les programmes qu'il va écrire, et en particulier, il devra gérer des cas exceptionnels.

- **Exemple:**

Un programme qui calcule le quotient de deux réels, vous devez gérer le cas exceptionnel où l' utilisateur veut faire une division par zéro.



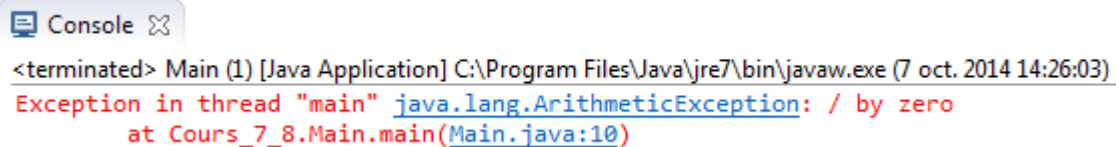
Le mécanisme utilisé en programmation orientée objet et en Java est celui des **exceptions**.

Les exceptions -Exemple

Exemple: la division sur 0

```
public class Main {  
    public static void main(String[] args) {  
        int val=10;  
        int val1= 0;  
        double result= val/val1;  
        System.out.println("Le resultat de la division est"+result );  
    }  
}
```

- Erreur générée à l'exécution:



The screenshot shows a console window titled "Console" with the following text:
<terminated> Main (1) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (7 oct. 2014 14:26:03)
Exception in thread "main" java.lang.ArithmeticException: / by zero
at Cours_7_8.Main.main(Main.java:10)

- Le programme s'arrête brutalement. La ligne ne sera donc pas exécutée, le programme est terminé.
- Le message d'erreur affiché à la console repère le numéro de la ligne de code qui a provoqué l'erreur, de plus, on a le nom de l'erreur qui a été produite, dans notre cas, il s'agit d'une *ArithmeticException*

Les exceptions

- Une exception est une erreur qui survient à l'exécution d'un programme Java.
- Java utilise un mécanisme d'interception des erreurs appelé **traitement des exceptions**
- Quand une erreur survient dans une méthode:
 - ❖ La méthode se termine immédiatement
 - ❖ Elle ne renvoie pas la valeur attendue
 - ❖ Elle ne renvoie aucune valeur.
 - ❖ Le contrôle est donné à des blocs de programmes spécifiques, **les gestionnaires d'exceptions où elle est capturée et traitée**

Les exceptions

Que devrait faire un programme en cas d'erreur?

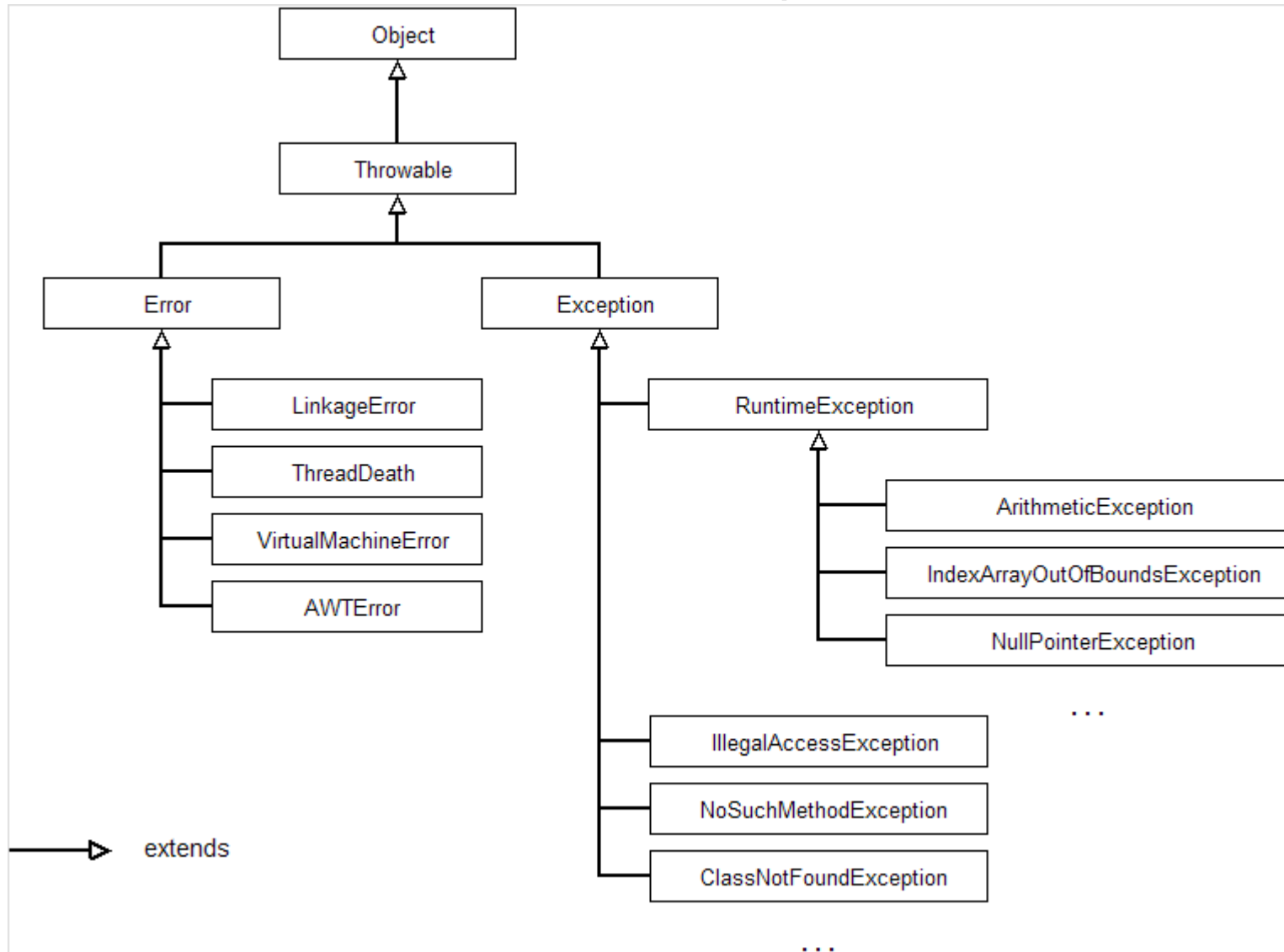
- Envoyer un message.
- Sauvegarder l'ensemble du travail en cours et permettre une sortie correcte du programme.
- Revenir à un état défini et donner à l'utilisateur la possibilité d'exécuter d'autres commandes

Exemples de causes d'erreurs

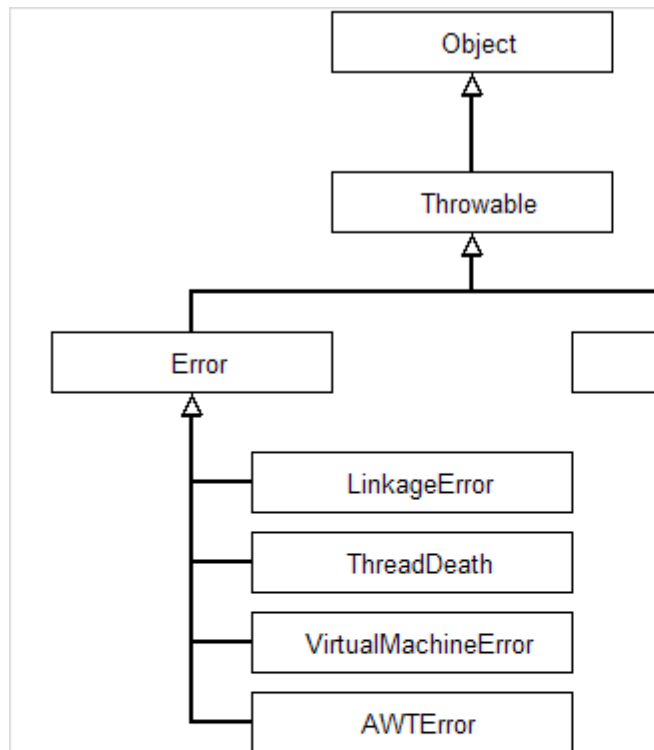
- **Les erreurs dues à l'utilisateur:** qui ne respecte pas les règles métiers de l'application
- **Les erreurs techniques :** connexion réseau défaillante, tentative de modification d'une clé primaire dans une base de données, imprimante déconnectée ou manquant de papier.
- **Les contraintes physique:** disque plein.
- **Les erreurs de programmation :** index d'un tableau invalide, emploi d'une référence à un objet non initialisé.

Classement des exceptions

- Hiérarchie des classes des exceptions



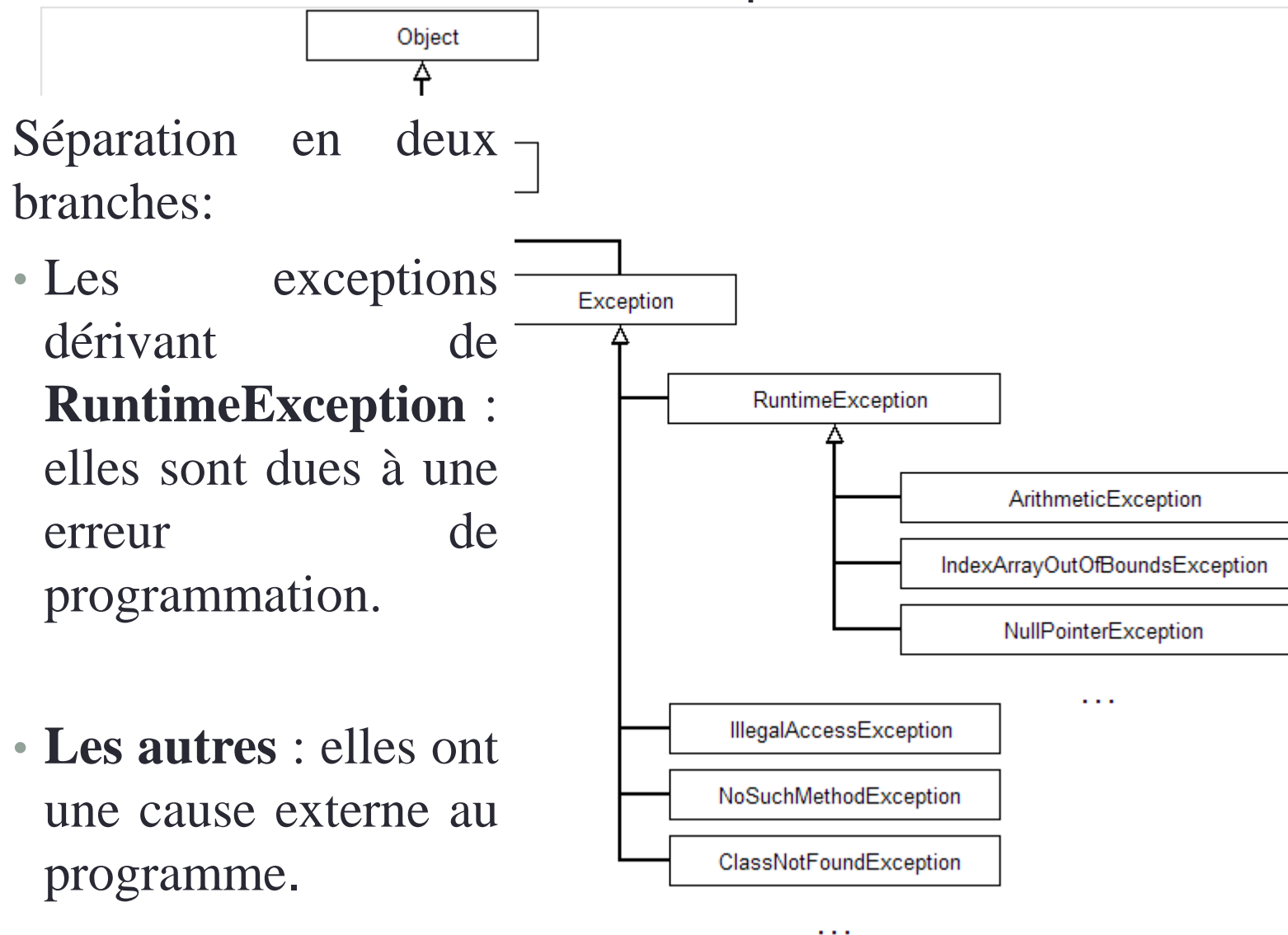
La branche Error



- Décrit les erreurs internes ou le manque de ressources dans le système d'exécution de Java.
- On ne se préoccupe pas de ce type d'exceptions.
- Une telle exception provoque généralement un message et l'arrêt du programme

Classement des exceptions

- Hiérarchie des classes des exceptions



Les sous classes de RuntimeException

- Dans cette branche, on retrouve des erreurs plus communes comme par exemple:
 - `ArithmeticException` qui représente une erreur d'arithmétique comme une division par zéro,
 - `NullPointerException` qui apparait lorsqu'on travaille sur une variable de type objet qui contient la référence null,
 - `ArrayIndexOutOfBoundsException` qui apparait lorsque l'on travaille avec les tableaux et qu'on tente d'accéder à un indice inexistant, ...

Les sous classes de RuntimeException

- Dans cette branche, on retrouve des erreurs plus communes comme par exemple:

- `ArithmeticException` qui est une erreur d'arithmétique comme une division par zéro,
- `NullPointerException` qui apparaît lorsqu'on travaille sur une variable qui contient la référence null,
- `ArrayIndexOutOfBoundsException` qui apparaît lorsque l'on travaille avec les tableaux et qu'on tente d'accéder à un indice inexistant, ...

Ces Exceptions peuvent être traitées

Comment faire pour capturer une exception et la traiter ?

- Il faut mettre dans un bloc **try** toutes les instructions susceptibles de provoquer une erreur.
- On rattache ensuite à ce bloc une ou plusieurs clauses **catch** qui vont chacun gérer un type précis d'exception

Gestion des exceptions avec l'instruction try-catch

```
public class Main {  
    public static void main(String[] args) {  
        int val=10;  
        int val1= 0;  
        double result=0;  
  
        try{  
            result= val/val1;  
            System.out.println("Le resultat de la division est"+result );  
        }  
        catch(ArithmeticException exeption){  
            System.out.println("Attention la division par zéro n'est pas autorisée");  
        }  
  
        System.out.println("Tout s'est bien déroulé!!!");  
    }  
}
```

Gestion des exceptions avec L'instruction try-catch

On a mis la ligne qui réalise la division et la ligne qui affiche le résultat dans un bloc **try**

```
public class Main {  
    public static void main(String[] args) {  
        int val=10;  
        int val1= 0;  
        double result=0;  
  
        try{  
            result= val/val1;  
            System.out.println("Le resultat de la division est"+result );  
        }  
        catch(ArithmeticException exeption){  
            System.out.println("Attention la division par zéro n'est pas autorisée");  
        }  
        System.out.println("Tout s'est bien déroulé!!!");  
    }  
}
```



Gestion des exceptions avec l'instruction try-catch

on ajoute une clause catch pour gérer les erreurs de type arithmétique.

```
public class Main {  
    public static void main(String[] args) {  
        int val=10;  
        int val1= 0;  
        double result=0;  
  
        try{  
            result= val/val1;  
            System.out.println("Le resultat de la division est"+result );  
        }  
        catch(ArithmeticException exeption){  
            System.out.println("Attention la division par zéro n'est pas autorisée");  
        }  
  
        System.out.println("Tout s'est bien déroulé!!!");  
    }  
}
```

L'instruction try-catch

- Que se passe-t'il si une erreur survient dans le bloc try ?
 1. le flux de contrôle est directement transféré vers la clause **catch** correspondant au type d'erreur rencontré,
 2. Le code de la clause catch est exécuté ,
 3. Le programme **continue** après le bloc try-catch.
- Résultat d'exécution du code précédent:

 Console 

```
<terminated> Main (1) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (7 oct. 2014 14:57:09)  
Attention la division par zéro n'est pas autorisée  
Tout s'est bien déroulé!!!
```

L'instruction try-catch

- Syntaxe de la clause catch:

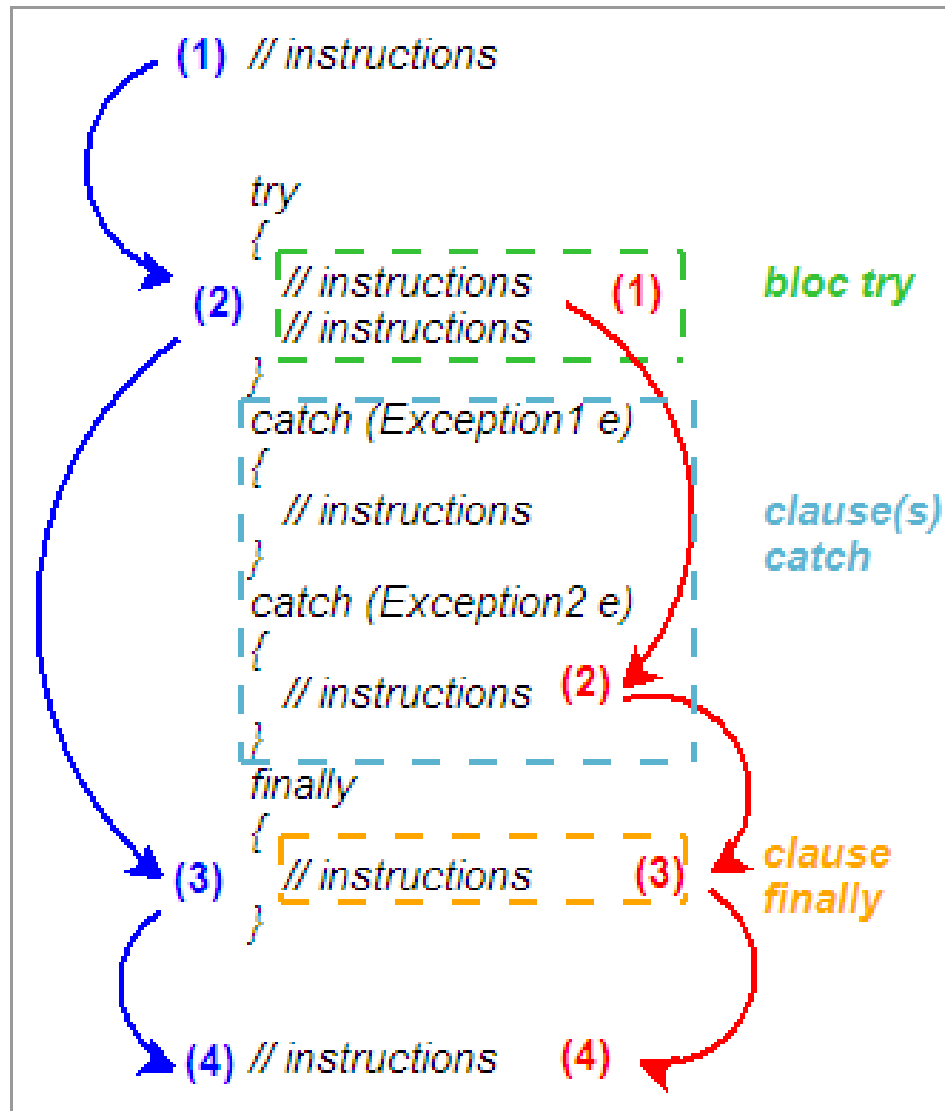
```
1  catch (ClasseException nomvariable)
2  {
3      // ...
4  }
```

- En java, toutes les exceptions possibles sont représentées par une classe.
- donc la clause `catch` reçoit en paramètre un objet qui permet de représenter l'exception qui s'est produite.

L'instruction try-catch : La clause finally

- La clause **finally** représente un bloc de code qui sera exécuté de toute façon, que le bloc **try** se soit terminé sans erreurs ou avec erreurs.

Comportement d'un flux de contrôle face à une instruction de type **try-catch-finally**:



Remarques:

- Il est possible de rattraper plusieurs types d'exceptions en enchaînant les constructions catch;
- Il est également possible d'imbriquer les constructions try-catch

L'instruction try-catch : La clause finally

- Pourriez-vous dire ce qu'il va afficher à la console ?

```
public class Main {  
    public static void main(String[] args) {  
        int val=10;  
        int val1= 0;  
        double result=0;  
  
        try{  
            result= val/val1;  
            System.out.println("Le resultat de la division est"+result );  
        }  
        finally{  
            System.out.println("Dans toute les cas le bloc finally est executé!!");  
        }  
  
        System.out.println("Tout s'est bien déroulé!!!");  
    }  
}
```

L'instruction try-catch : La clause finally

```
public class Main {  
    public static void main(String[] args) {  
        int val=10;  
        int val1= 0;  
        double result=0;  
  
        try{  
            result= val/val1;  
            System.out.println("Le resultat de la division est"+result );  
        }  
        finally{  
            System.out.println("Dans toute les cas le bloc finally est executé!!");  
        }  
  
        System.out.println("Tout s'est bien déroulé!!!");  
    }  
}
```

○ A l'exécution:

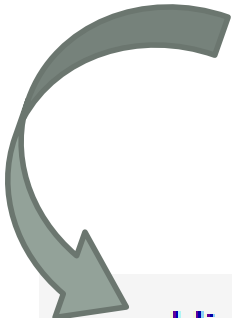
Console

```
<terminated> Main (1) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (7 oct. 2014 15:08:04)  
Dans toute les cas le bloc finally est executé!!  
Exception in thread "main" java.lang.ArithmeticException: / by zero  
    at Cours_7_8.Main.main(Main.java:13)
```

Propagation des exceptions

Exemple de propagation des exceptions

```
1  public class TestPropagation
2  {
3      public static void main (String[] args)
4      {
5          int a = 8;
6          int b = 0;
7          Propagation p = new Propagation();
8          System.out.println (p.divide (a, b));
9      }
10 }
```

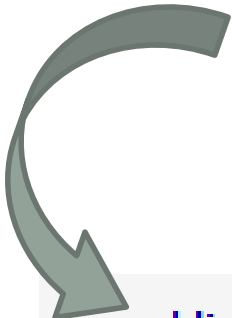


```
1  public class Propagation
2  {
3      public int divide (int a, int b)
4      {
5          int r = a / b;
6          return r;
7      }
8  }
```

Voyons maintenant, lorsqu'une erreur se produit dans une méthode autre que le *main*;

Propagation des exceptions

```
1  public class TestPropagation
2  {
3      public static void main (String[] args)
4      {
5          int a = 8;
6          int b = 0;
7          Propagation p = new Propagation();
8          System.out.println (p.divide (a, b));
9      }
10 }
```



```
1  public class Propagation
2  {
3      public int divide (int a, int b)
4      {
5          int r = a / b;
6          return r;
7      }
8  }
```

- Si on exécute le programme:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Propagation.divide(Propagation.java:5)
    at TestPropagation.main(TestPropagation.java:8)
```

Lever une exception- l'instruction throw

- L'instruction throw permet de lever une exception.
- Pour l'utiliser, il faut lui fournir une référence vers un objet de type Exception.
- Exemple d'utilisation:

```
public int fact (int n)
{
    if (n <= 0) throw new ArithmeticException();

    int result = 1;
    for (int i = 1; i <= n; i++)
    {
        result *= i;
    }
    return result;
}
```

- A l'exécution:

```
Exception in thread "main" java.lang.ArithmeticException
    at Exceptions.Quotient.fact(Quotient.java:21)
    at Exceptions.Quotient.main(Quotient.java:15)
```

Les exceptions personnalisées

- Pour créer sa propre classe d'exception, il faut que la classe créée hérite de la classe `java.lang.Exception` (voir l'api java pour ces constructeurs)
- On crée une nouvelle exception pour
 1. gérer, par programmation (et non par message d'erreur), les utilisations incorrectes d'une méthode.
 2. et surtout prévenir l'utilisateur que certaines des méthodes sont incorrectes et qu'elles doivent être gérées
 3. Dans le programme principale entourez le code concerné par un `try and catch`

Exemple du téléthon:

La somme versée par une ville doit être positive

```
public class Ville {  
  
    private String nom;  
    private int nbHabitants;  
  
    public Ville(String nom,int nbHabitants){  
        this.nom=nom;  
        this.nbHabitants=nbHabitants;  
    }  
  
    public void faireDont(Telethon t, double somme) {  
        t.encaisser(somme);  
    }  
    public String getNom() {  
        return nom;  
    }  
    public double getNbHabitants() {  
        return nbHabitants;  
    }  
    public void setNbHabitants(int nbHabitants) {  
        this.nbHabitants = nbHabitants;  
    }  
    public void setNom(String nom) {  
        this.nom = nom;  
    }  
}
```

C'est dans cette méthode qu'on s'assurera que la somme versée est bien positive

Exemple du téléthon:

Création de la classe DontPositifExeption

```
public class DontPositifExeption extends Exception {  
  
    public DontPositifExeption(String message){  
        super(message);  
        System.out.println(message);  
    }  
  
}
```

La classe
DontPositifExeption hérite de
la classe **Exception**

Exemple du téléthon: Modification de la méthode faireDont

```
public class Ville {  
  
    private String nom;  
    private int nbHabitants;  
  
    public Ville(String nom,int nbHabitants){  
        this.nom=nom;  
        this.nbHabitants=nbHabitants;  
    }  
}
```

Dans la méthode concernée mettre un **throws** le nom de l'exception

```
public void faireDont(Telethon t, double somme) throws DontPositifExeption{
```

```
    if(somme<0) throw new DontPositifExeption("La somme donnée doit être positive, "+this.getNom()+ " C'est pas bien!!");  
    else{
```

```
        t.encaisser(somme);  
    }  
}
```

```
public String getNom() {  
    return nom;  
}
```

```
public double getNbHabitants() {  
    return nbHabitants;  
}
```

```
public void setNbHabitants(int nbHabitants) {  
    this.nbHabitants = nbHabitants;  
}
```

```
public void setNom(String nom) {  
    this.nom = nom;  
}}
```

Instancier votre exception personnalisée

Exemple du téléthon:

Et dans le main

Il faut absolument mettre le code qui peut générer une exception entre un try and catch.

```
public class MainProgram {
    public static void main(String[] args) {

        Ville v= new Ville("Lille", 500000);
        Ville v1= new Ville("Paris", 70000);
        Ville v2= new Ville("Lyon", 70000);

        Telethon frT= new Telethon();
        try{
            v.faireDont(frT, 1000);
            System.out.println("après le dont de la ville, "+v.getNom()+ " la cagnotte vaut "+ frT.getCagnotte());

            v1.faireDont(frT, -2000);
            System.out.println("après le dont de la ville, "+v1.getNom()+ " la cagnotte vaut "+ frT.getCagnotte());

            v2.faireDont(frT, 3000);
            System.out.println("après le dont de la ville, "+v2.getNom()+ " la cagnotte vaut "+ frT.getCagnotte());

        }catch(DontPositifExeption e){ }

        System.out.println("la cagnotte est de"+frT.getCagnotte());

    }
}
```

Console

```
<terminated> MainProgram (1) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (26 sept. 2018 à 16:36:55)
Après le dont de la ville, Lille la cagnotte vaut 1000.0
La somme donnée doit être positive, Paris C'est pas bien!!
la cagnotte est de 1000.0
```