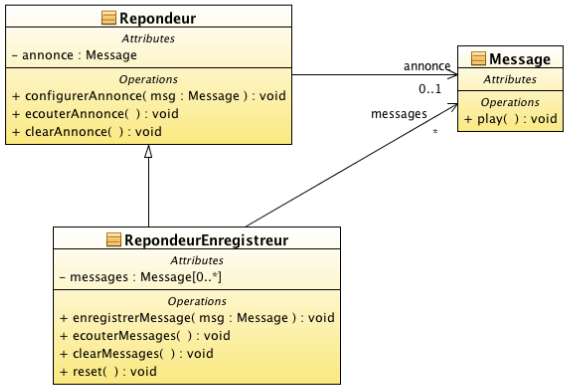
EXCEPTIONS LOGICIELLES 2

Walter Rudametkin
Maître de Conférences
Bureau F011
Walter.Rudametkin@polytech-lille.fr

Advantages of Exceptions

- Advantage 1: Separating Error-Handling Code from "Regular" Code
- Advantage 2: Propagating Errors Up the Call Stack
- Advantage 3: Grouping and Differentiating Error Types
- https://docs.oracle.com/javase/tutorial/essential/exceptions/advantages.html

Exemple



Exercise

- Rajouter une class Switchboard qui contient un tableau avec des Repondeurs et RepondeursEnregistreurs.
- Réalisez du code pour rajouter des repondeurs et pour les effacer:
  - void addRepondeur(Repondeur rep) ;
  - void removeRepondeur(int index);
- Gérez les exceptions:
  - Index invalide, repondeur plein, …
- Ajouter des méthodes pour compter les repondeurs
  - `public int countRepondeurs()`
  - `public int countRepondeursEnregistreurs();`
- Créez un code (`main(…)`) pour tester votre switchboard

Exception Wrapping

```
public class FooException extends Exception{

    public FooException() {
        super();
    }
    public FooException(String message){
        super(message);
    }
    public FooException(String message, Throwable cause){
        super(message, cause);
    }
    public FooException(Throwable cause) {
        super(cause);
    }
}
```


Exemple: Sans Exception Wrapping

```
public class RepondeurInexistantException extends Exception {}
```

```
public class Switchboard {
    …
    public Repondeur getRepondeur(int index) throws
                                    RepondeurInexistantException {
        try{
            return repondeurs[index];
        }catch(IndexOutOfBoundsException ex){
            throw new  RepondeurInexistantException();
            //infos sur l'origine de l'exception sont perdus!
        }
    }
```

```
//RUNTIME, exemple sortie sur console:
RepondeurInexistantException
        at Switchboard.getRepondeur(Switchboard.java:35)
        at Switchboard.removeRepondeur(Switchboard.java:40)
        at TestSwitchboard.main(TestSwitchboard.java:21)
```

Exemple: Avec Exception Wrapping

```
public class RepondeurInexistantException extends Exception{
    public RepondeurInexistantException() {
        super();
    }
    public RepondeurInexistantException(String message){
        super(message);
    }
    public RepondeurInexistantException(String message,
                                        Throwable cause){
        super(message, cause);
    }
    public RepondeurInexistantException(Throwable cause) {
        super(cause);
    }
}
```


Exemple: Avec Exception Wrapping

```
public class Switchboard {
    …
    public Repondeur getRepondeur(int index) throws
                                    RepondeurInexistantException {
        try{ return repondeurs[index]; }
        catch(IndexOutOfBoundsException ex){
            throw new  RepondeurInexistantException(
                "Repondeur id=" + index + " n'existe pas", ex);
        }
    }
}
```

```
//RUNTIME: Sortie sur la console
RepondeurInexistantException: Repondeur id=5 n'existe pas
        at Switchboard.getRepondeur(Switchboard.java:35)
        at Switchboard.removeRepondeur(Switchboard.java:40)
        at TestSwitchboard.main(TestSwitchboard.java:21)
Caused by: java.lang.ArrayIndexOutOfBoundsException: 5
        at Switchboard.getRepondeur(Switchboard.java:33)
        … 2 more
```


Exemple: N'oubliez pas d'utiliser les nouveaux infos

```
…
 public static void main (String[] args){
   try{
      System.out.println("\nCommence à enlever des repondeurs");
      sb.removeRepondeur(3);
      sb.removeRepondeur(5);
      sb.removeRepondeur(7);
   }catch(RepondeurInexistantException e){
      e.printStackTrace();

       //optionellement, rajouter un petit message pour clarifier
       //l'erreur
      System.out.println("\n*** dans main: Mauvais index?\n");
   }
…
```


Try, catch, finally with ressources

```
…
        try {
              operation_risquée1;
              opération_risquée2;
        } catch (ExceptionInteressante e) {
              //traitements
        } catch (ExceptionParticulière e) {
              //traitements
        } catch (Exception e) {
              //traitements
        } finally {
              //traitement_pour_terminer_proprement;
        }
…
```


Try, catch, finally with ressources

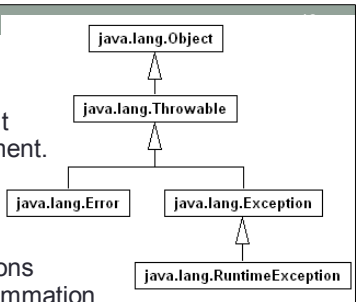
```
 private static void printFile() throws IOException {

     InputStream input = null;

     try {
         input = new FileInputStream("file.txt");

         int data = input.read();
         while(data != -1){
             System.out.print((char) data);
             data = input.read();
         }
     } finally {
         if(input != null){
             input.close();
         }
     }
 }
```

3 types d'exceptions



- **Error :** ces exceptions concernent des problèmes liés à l'environnement. Elles héritent de la classe Error (exemple : OutOfMemoryError)
- **RuntimeException :** ces exceptions concernent des erreurs de programmation qui peuvent survenir à de nombreux endroits dans le code (exemple : NullPointerException, ClassCastException). Elles héritent de la classe RuntimeException.
- **Checked exception :** ces exceptions doivent être traitées ou propagées. Toutes les exceptions qui n'appartiennent pas aux catégories précédentes sont de ce type.