Java
Exceptions

Exception

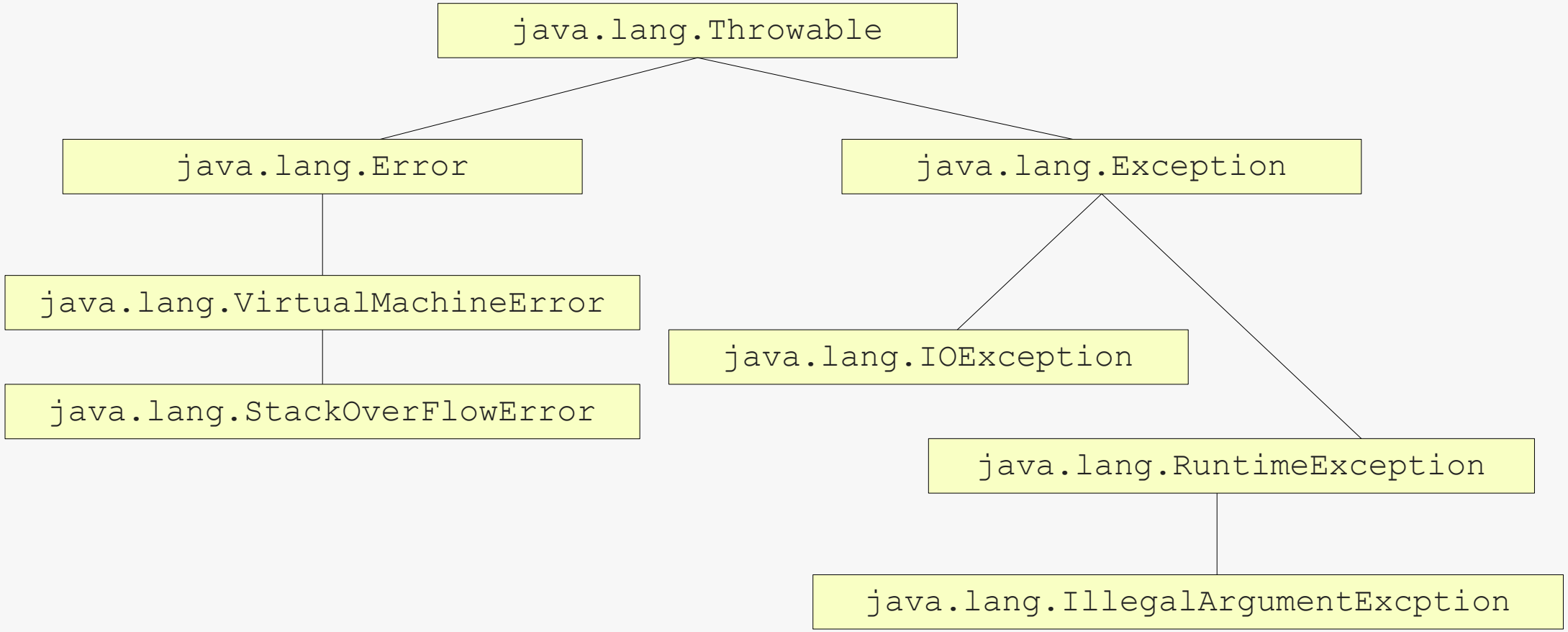
- Signal qui indique la présence d'une situation inhabituelle
- Une exception est de type `java.lang.Throwable`
  - une exception est créee et levée à l'endroit où intervient l'erreur
  - l'exception est passée au gestionnaire d'exception adapté
  - l'exception peut contenir des informations supplémentaires pour gérer l'erreur

Exception

- Les exceptions dérivent généralement de la classe `java.lang.Exception`
- Les exceptions applicatives devraient toujours encapsuler les exceptions "bas niveau"
  - afin d'éviter de faire dépendre les couches hautes de l'application des implémentations
    - utilisation de JDBC ou Hibernate par exemple
- Les classes d'exceptions concrètes dérivent de `java.lang.Exception`
  - `IOException, FileNotFoudException, ...`

Exception

- Schéma simplifié des exceptions Java



Exception

- Trois tâches sont attachées à la gestion des exceptions
  - écriture des classes d'exception pour l'application
  - écriture du code déclarant et levant une exception
  - gestion de l'exception levée

Exceptions

- `java.lang.Error`
  - cette classe, et ses classes dérivées, représente une erreur
    - la gestion n'est pas vérifiée par le compilateur
- `java.lang.RuntimeException`
  - cette classe, et ses classes dérivées, sont des exceptions de type runtime
  - le compilateur n'oblige pas le développeur à les gérer
- Les autres exceptions doivent être gérées par le développeur

Exception

- Les exceptions doivent être utilisées pour des situations anormales
  - utilisent du temps processeur
- Faites une gestion claire des exceptions
  - messages d'erreur, log, ...
  - pas de code mort : `catch(Exception e) {}`
- Regroupez le code susceptible de lever des exceptions dans un même bloc `try`

Écriture de l'exception

- Il suffit de spécialiser la classe `Exception`, ou l'une de ses sous-classes
  - il est préférable de créer une hiérarchie d'exceptions pour votre application

```
public class ApplicationException extends Exception{
     public ApplicationException() {
          super();
     }

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

     public ApplicationException(String message) {
          super(message);
     }

     public ApplicationException(Throwable cause) {
          super(cause);
     }
}
```

Lever l'exception

- Il faut créer une instance de l'exception
- Faire remonter l'exception par la clause `throw`
  - si la méthode levant l'exception ne la gère pas sur place il faut la faire suivre par la clause `throws`
  - si les méthodes appelantes de la méthode levant l'exception ne la gèrent pas, elles doivent la faire suivre avec la clause `throws`
- Si aucune méthode ne gère l'exception, celle-ci remonte à la JVM

Lever l'exception

- Exemple

```
public class Traitement {

     public void getDataFromDS() throws ApplicationDataAccessException{
          throw new ApplicationDataAccessException("Pas d'accès à la base de donnée");
     }

     public void sendMessage(String message) throws ApplicationComputeException{
          throw new ApplicationComputeException("Pas d'accès réseau");
     }
}
```

Gestion de l'exception

- Un bloc `try` encapsule le code susceptible de lever une exception
- Un bloc catch gère l'exception en fonction de son type
  - plusieurs blocs `catch` sont possibles
  - attention à l'ordre des blocs : de l'exception la plus spécialisée à la moins spécialisée
- Un bloc `finally` peut contenir le code qui sera appelé, qu'il y est une exception ou non
  - même si `return` présent dans `try` ou `catch`

Gestion des exceptions

- Exemple

```
Traitement t = new Traitement();

try {
     t.getDataFromDS();
     t.sendMessage("message");
     } catch (ApplicationDataAccessException e) {
          e.printStackTrace();
     }catch (ApplicationComputeException e) {
          e.printStackTrace();
     }finally{
     // code exécuté dans tous les cas
}
```

- Java 7 améliore la succession de catch

```
try {
     t.getDataFromDS();
     t.sendMessage("message");
} catch (ApplicationDataAccessException | ApplicationComputeException e) {
     e.printStackTrace();
}
```

Gestion des exceptions

- Lors de la gestion de ressources le code peut devenir très lourd en cas de gestion de ressources
  - fichier, accès base de données, ...
  - les variables doivent-être en-dehors du bloc `try`
  - le bloc `finally` doit fermer les ressources
    - la fermeture des ressources étant successible de lever une exception

Gestion des exceptions

```
...
     BufferedReader reader = null;

     try {
          URL url = new URL("http://www.yoursimpledate.server/");
          reader = new BufferedReader(new InputStreamReader(url.openStream()));
          String line = reader.readLine();
          SimpleDateFormat format = new SimpleDateFormat("MM/DD/YY");
          Date date = format.parse(line);
     }
     catch (MalformedURLException exception) {
          // handle passing in the wrong type of URL.
     } catch (IOException exception) {
          // handle I/O problems.
     } catch (ParseException exception) {
          // handle date parse problems.
     } finally {
          if (reader != null) {
               try {
                    reader.close();
                     } catch (IOException ex) {
                    ex.printStackTrace();
               }
          }
     }
...
```

Gestion des exceptions

- Java 7 améliore ce code
  - les ressources peuvent être déclarées dans le bloc `try`, elles seront automatiquement fermées à la fin du bloc `try`
    - ARM : Automatic Resource Management
    - fonctionne avec les classes implémentant `AutoCloseable`

```
try (BufferedReader reader = new BufferedReader(
  new InputStreamReader(new URL("http://www.yoursimpledate.server/").openStream())))
 {
  String line = reader.readLine();
  SimpleDateFormat format = new SimpleDateFormat("MM/DD/YY");
  Date date = format.parse(line);
 } catch (ParseException | IOException exception) {
  // handle I/O problems.
 }
```