01/04/2023
Version 3

OBJET

JÉRÉMY PERROUAULT

GÉNÉRIQUES

LES TEMPLATES GÉNÉRIQUES

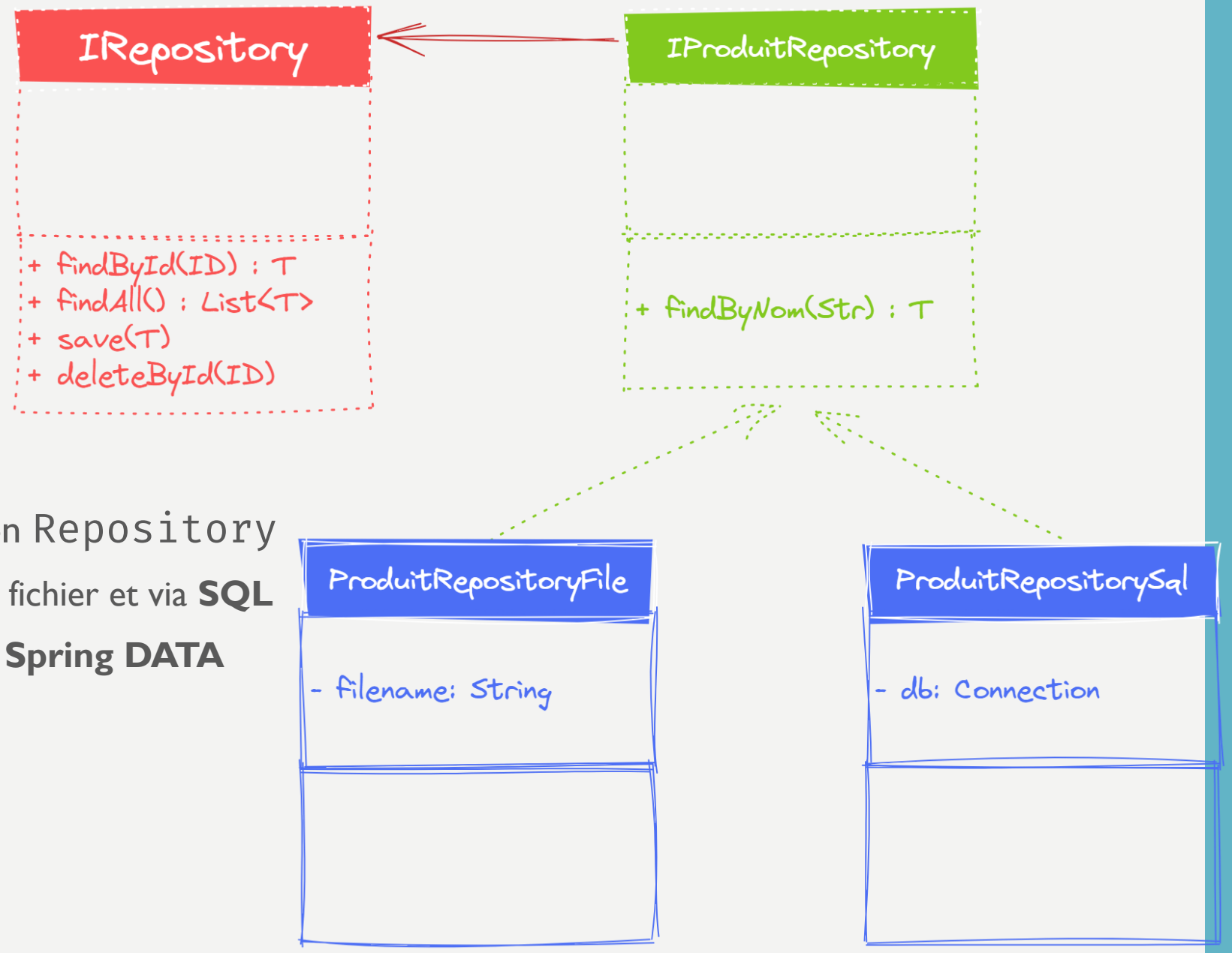
TEMPLATE GÉNÉRIQUE

- Classe qui permet d'utiliser un type d'objet générique
  - Ne connait pas le type d'objet qu'on va utiliser
  - Permet de définir les signatures

TEMPLATE GÉNÉRIQUE

- Démonstration classe soigneur générique
  - Avec Object
  - Avec généricité
  - Avec interfaces

EXERCICE



- Démonstration et explication `Repository`
  - Peut gérer un stockage via fichier et via **SQL**
  - Evolutif vers **Hibernate** / **Spring DATA**

BASE

ÉLÉMENTS DE BASE

ELÉMENTS DE BASE

- Les classes mises à disposition par Java se trouvent dans des packages particuliers
  - Pour pouvoir manipuler ces classes, il faut **importer** le ou les packages concernés
  - D'une manière générale, si la classe n'est pas accessible MAIS qu'elle existe belle et bien
    - Eclipse vous propose de l'importer pour la manipuler

```
import java.util.ArrayList;

import java.util.*;
```

ELÉMENTS DE BASE

- On peut généraliser et faire des imports statiques
  - Permet ainsi de ne pas réécrire l'instruction complète, mais simplement utiliser « out »

```
import static java.lang.System.out;
```

  - Fonctionne sur les variables et sur les méthodes statiques

ELÉMENTS DE BASE

- Les annotations sont à positionner selon les cas
  - Sur une classe
  - Sur un attribut
  - Sur une méthode

- Elles sont préfixées de @
  - `@Autowired`
  - `@Override`

- Elles confèrent un comportement
  - A la lecture du code
  - A la compilation du code
  - A l'exécution du code

ELÉMENTS DE BASE

- Arguments variables en dernière position

```
public void multiArgs(String arg1, int... nombres) {
  // nombres est en fait un int[] (Array) !
}
```

```
multiArgs("test", 1, 2, 3);
```

- A partir de Java 1.5, c'est le cas de System.out.printf() !

EXCEPTIONS

LANCER ET GÉRER DES ERREURS

LES EXCEPTIONS

- Permet de contrôler les erreurs sans interrompre le programme
  - `try` ... `catch` ! Et `finally` ...
- Classe `Exception`
- Possibilité de lancer une exception avec le mot-clé `throw`

```
throw new Exception();
```

LES EXCEPTIONS

- Comment créer une `Exception` spécialisée ?
  - Il suffit de créer une **classe JAVA** qui hérite de la classe `Exception` ou d'un autre dérivé

LES EXCEPTIONS

- Possibilité d'attraper plusieurs exceptions et gérer les cas

```
try {
  //...
}
catch (MonException mex) {
  //...
}
catch (Exception ex) {
  //...
}
```

```
try {
  //...
}
catch (Exception ex) {
  //...
}
finally {
  //...
}
```

- Possibilité d'exécuter un code que l'exception soit levée ou non avec `finally`
  - S'exécute même si l'instruction « return » est présente dans le try ou dans le catch

LES EXCEPTIONS

try

1 On tente l'exécution d'un code qui peut potentiellement lever une Exception

Si une Exception est levée, le bloc try s'arrête immédiatement

catch

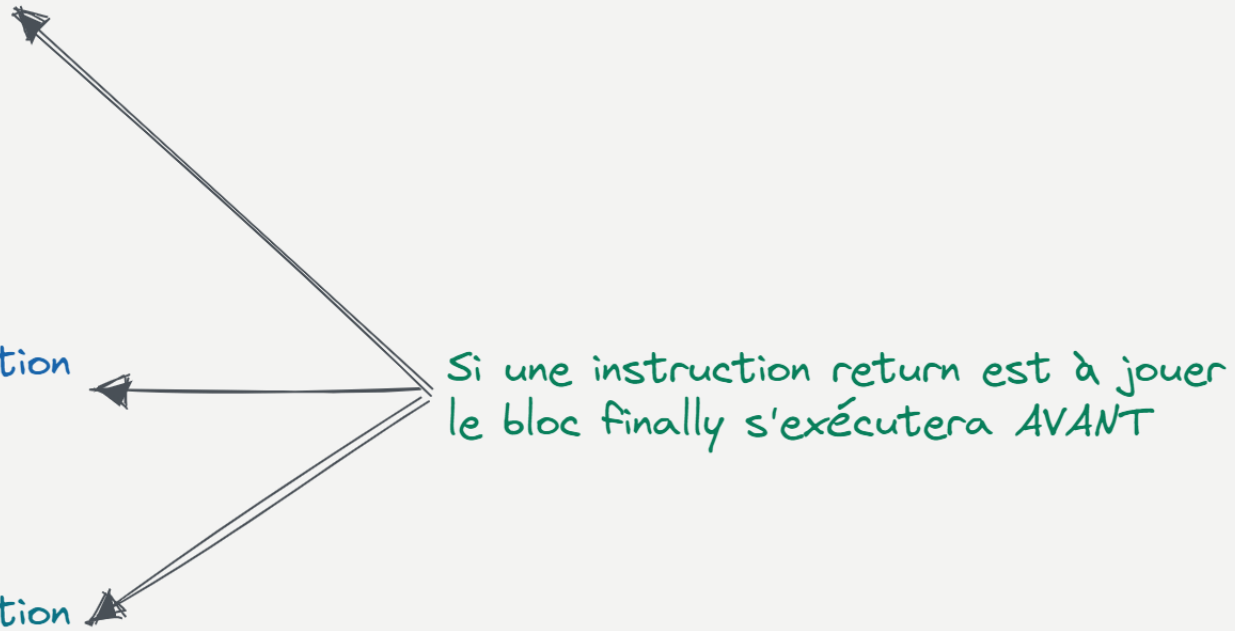
2 On exécute le code associé à la résolution du type d'exception levée

catch

2 On exécute le code associé à la résolution du type d'exception levée

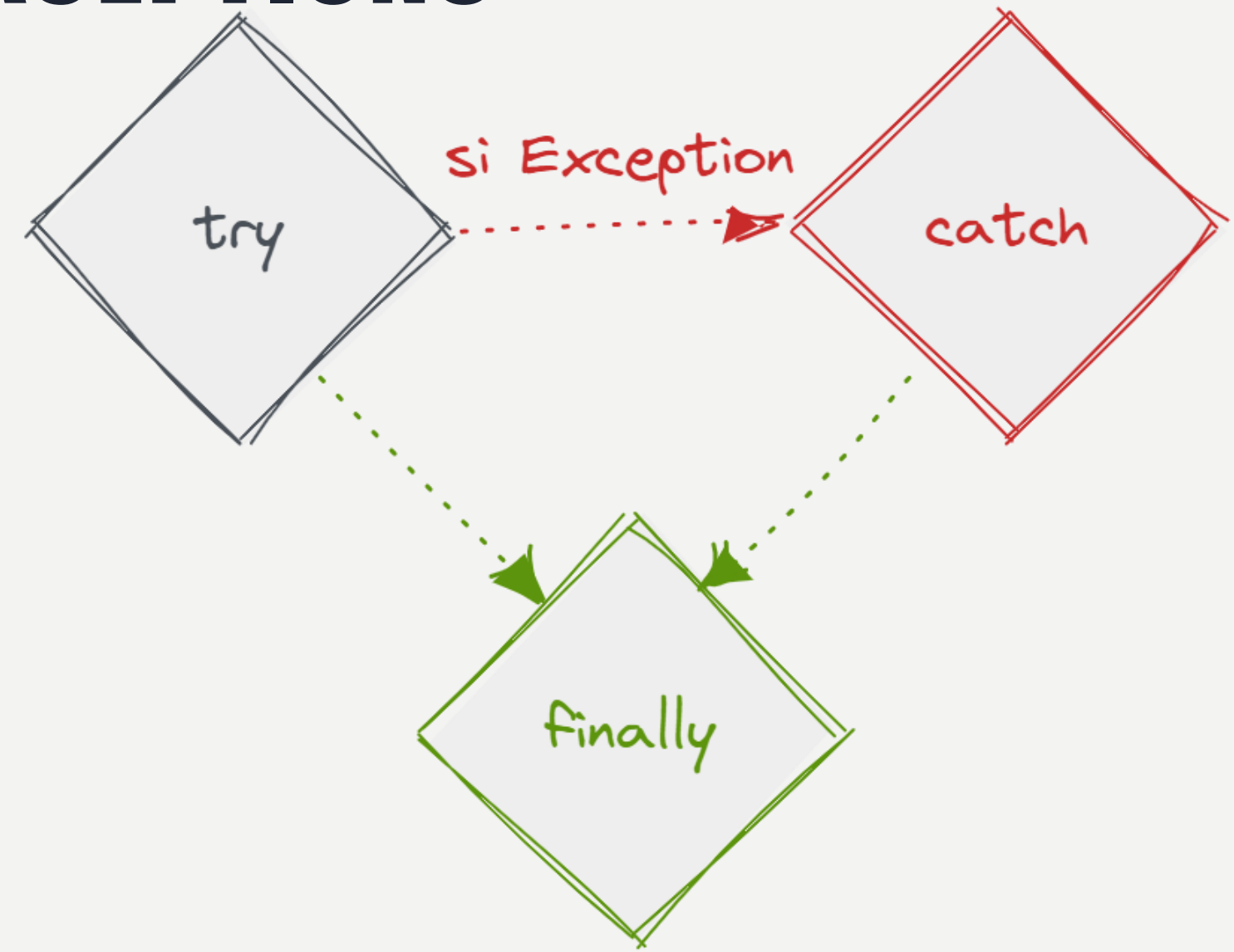
finally

3 On exécute ce code dans tous les cas

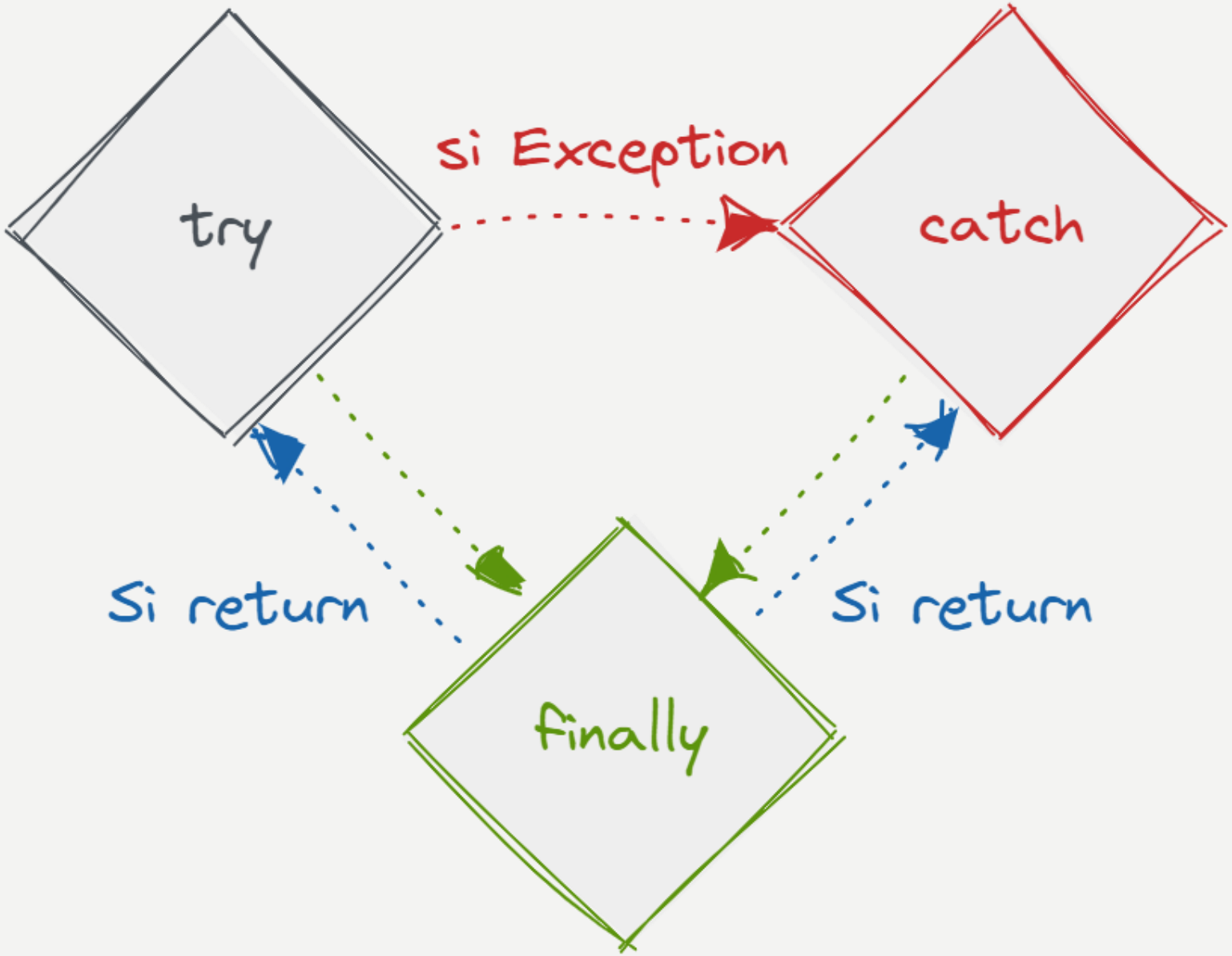


Si une instruction return est à jouer le bloc finally s'exécutera AVANT

LES EXCEPTIONS



LES EXCEPTIONS



LES EXCEPTIONS

- En **JAVA**, les méthodes qui – potentiellement – déclenchent une `Exception`
  - Doivent signer qu'elles le font avec le mot-clé `throws`

```java
public void uneMethode() throws Exception {
  throw new Exception();
}
```

```java
public int uneAutreMethode() throws UneException, UneAutreException {
  //...
}
```

EXERCICE

- Dans un programme principal
  - Afficher « saisir un chiffre »
  - Attendre la saisie d'un chiffre avec `Scanner.nextInt()`
  - Saisir une lettre
    - L'application va crasher
  - Faire en sorte que l'application continue de s'exécuter
    - Demander à l'utilisateur la saisie d'un chiffre, jusqu'à ce que ce soit OK !
    - Utiliser `Scanner.next()` dans le catch pour réinitialiser la mauvaise saisie

EXERCICE

- Alors que `nextInt()` peut retourner une `Exception`
  - **JAVA** ne nous oblige pas à la gérer
  - Il existe 3 types d'`Exception`
    - `Checked Exception`
    - `Unchecked Exception`
    - `Error`

EXCEPTIONS

- `Checked Exception`
  - `Exception` interne à l'application qui peut être anticipée
  - `Exception` qui doit être gérée dans le code
    - Par un bloc `try .. catch`
    - Par l'utilisation du mot-clé `throws` sur la signature d'une méthode
  - Dérive de `Throwable`
  - **Classe** `Exception`
  - Toute les exceptions sont `checked`, sauf celles qui héritent de
    - `RuntimeException`
    - `Error`

EXCEPTIONS

- `Unchecked Exception`
  - `Exception` interne à l'application qui ne peut être anticipée
  - Dérive de `Exception`
  - **Classe** `RuntimeException`

EXCEPTIONS

- `Error`
  - `Exception` externe à l'application dont l'anticipation est difficile
    - Exemple : `OutOfMemoryError`, `StackOverflowError`
  - Dérive de `Throwrable`
  - **Classe** `Error`

EXERCICE

- Créer une classe de lecture de chiffre **LireChiffre**
  - Avec une méthode *positif()*
- Créer une exception **LireChiffreFormatException**
- Créer une exception **LireChiffreNegatifException**
- Le programme principal fait appel à **LireChiffre**
  - Si la saisie est incorrecte, **LireChiffre** lève une exception **LireChiffreFormatException**
  - Si le chiffre est négatif, **LireChiffre** lève une exception **LireChiffreNegatifException**
  - Le programme principal intercepte les 2 exceptions
    - Il a un comportement différent selon le type d'exception

LES EXCEPTIONS – LES FLUX

- Un flux ouvert devrait <u>toujours</u> être fermé après son utilisation
  - Dans le deuxième exemple, le bloc `try catch` ferme automatiquement le flux
  - C'est possible lorsque la classe implémente l'interface `Closeable`

```
Scanner sc = new Scanner(System.in);

try {
  //...
}

catch (Exception e) {
  //...
}

finally {
  sc.close();
}
```

```
try (Scanner sc = new
Scanner(System.in)) {
  //...
}

catch (Exception e) {
  //...
}
```

BIBLIOTHÈQUES

ECRIRE UN CODE RÉUTILISABLE

LES BIBLIOTHÈQUES

- Tout projet **JAVA** peut être exporté en **JAR** (**J**ava **AR**chive)
- Tout **JAR** peut être importé dans un projet **JAVA**
  - Ce **JAR** devient une dépendance vis-à-vis du projet qui l'inclu et qui l'utilise !

LES BIBLIOTHÈQUES

- Démonstration
  - Création d'un nouveau projet **JAVA** par un volontaire
  - Création d'une classe **Personne** (nom, prénom, age)
  - Exportation du projet en **JAR**
  - Intégration de ce **JAR** dans un autre projet

BDD

ECRIRE ET LIRE UNE BASE DE DONNÉES

BASE DE DONNÉES – ORM

- **Object Relational Mapping**
- Classes métier représentent une projection de la base de données

Modèle objet	Modèle relationnel
Graphe d'objets	Base de données relationnelle
Instances de classes	Enregistrements dans une table
Références	Relations (FK → PK)
« Clé primaire » optionnelle	
Héritage	

BASE DE DONNÉES – ORM

- Exemple de mapping

Modèle objet (une classe)	Modèle relationnel (une table)
Personne.java	Personne
int id	PER_ID
String nom	PER_NOM
String prenom	PER_PRENOM
String adresse	PER_ADRESSE

BASE DE DONNÉES – ACCÈS

- L'accès le plus bas niveau avec Java
  - Driver **JDBC** adapté au serveur **SQL** manipulé
  - Implémentation de la bibliothèque « **connecteur MySQL** »
- On doit charger ce driver adapté
  - La classe doit être présente dans le *classpath*

```
try {
  Class.forName("com.mysql.cj.jdbc.Driver");
}
catch (ClassNotFoundException e) {
  //...
}
```

BASE DE DONNÉES – ACCÈS

- L'accès le plus bas niveau avec Java
  - Driver **JDBC** adapté au serveur **SQL** manipulé
  - Implémentation de la bibliothèque « **connecteur PostgreSQL** »
- On doit charger ce driver adapté
  - La classe doit être présente dans le *classpath*

```
try {
  Class.forName("org.postgresql.Driver");
}
catch (ClassNotFoundException e) {
  //...
}
```

BASE DE DONNÉES – ACCÈS

- Pour s'y connecter
  - Il faut connaître l'URL de connexion

```
try {
  Connection myConnection =
    DriverManager.getConnection("jdbc:mysql://localhost:3306/nom_base", "username", "password");
}

catch (SQLException e) {
  //...
}
```

```
try {
  Connection myConnection =
    DriverManager.getConnection("jdbc:postgresql://localhost:5432/nom_base", "username", "password");
}

catch (SQLException e) {
  //...
}
```

BASE DE DONNÉES – ACCÈS

- Exécuter des requêtes
  - Création d'un `Statement`
  - Récupération du résultat dans un `ResultSet` avec la méthode `executeQuery()`
    - Il existe aussi `execute()` et `executeUpdate()`

```
try {
  Statement myStatement = myConnection.createStatement();
  ResultSet myResult = myStatement.executeQuery("SELECT PER_ID, PER_NOM, PER_PRENOM, PER_ADRESSE FROM personne");

  while (myResult.next()) {
    System.out.println(myResult.getString("PER_NOM"));
    //...
  }
}

catch (SQLException e) {
  //...
}
```

BASE DE DONNÉES – ACCÈS

- Exécuter des requêtes
  - Création d'un `Statement` préparé
  - Récupération du résultat dans un `ResultSet` avec la méthode `executeQuery()`
    - Il existe aussi `execute()` et `executeUpdate()`

```
try {
  PreparedStatement myStatement =
  myConnection.prepareStatement("INSERT INTO produit (PRO_NOM, PRO_PRIX, PRO_FOURNISSEUR_ID) VALUES (?, ?, 3)");

  myStatement.setString(1, "GoPRO HERO 6");
  myStatement.setFloat(2, 499.99f);

  myStatement.execute();
}

catch (SQLException e) {
  //...
}
```

BASE DE DONNÉES – ACCÈS

- Pour contrôler les *transactions SQL* (auto-commitées par défaut)
  - *Les instructions de transaction sont accessibles en **JAVA***

```
//On désactive l'auto-commit
myConnection.setAutoCommit(false);

//On joue la transaction
Statement myStatement = myConnection.createStatement();
//...

//On valide la transaction
myConnection.commit();

//On aurait pu l'annuler avec 'myConnection.rollback()'
```

BASE DE DONNÉES – ACCÈS

- Pour aller plus loin, vous pouvez consulter la documentation officielle
  - *https://docs.oracle.com/javase/tutorial/jdbc/basics/*

BASE DE DONNÉES – EXERCICE

- Créer une classe `Produit`
  - `id`, `nom`, `prix`
- Dans un programme principal
  - Lister les produits
  - Ajouter un produit
- Créer une classe `ProduitRepositorySql`
  - La manipuler depuis le programme principal
- Implémenter la composition `Repository` vue précédemment
  - Interface `IRepository<T, ID>`
  - Interface `IProduitRepository` qui hérite de l'**interface** `IRepository<T, ID>`
  - Classe `ProduitRepositorySql` qui implémente l'**interface** `IProduitRepository`