Module
II.102 Algorithme et Programmation

Projet Assisté (2ème séance): Fonctionnalités avancées et tests unitaires

isep
École d'ingénieurs du numérique

Sommaire

I. Lecture / Ecriture fichiers externes

- En Java, il peut parfois être utile de lire le contenu d'un fichier pour récupérer des données. Qu'il s'agisse d'un fichier que nous avons déjà sauvegardé ou d'un autre fichier, il est possible de le lire, ligne par ligne

- Il peut être intéressant pour nos programmes Java de pouvoir écrire dans un fichier pour plusieurs raisons, comme par exemple pour traiter des fichiers de données (XML, CSV, etc.).

L'idée est d'ajouter une fonctionnalité de lecture et d'écriture de fichiers sur votre projet Java.

Ceci pourrait être fait grâce à l'utilisation des classes fournies par les deux packages :

- java.io ou,
- java.nio.file

- La lecture du contenu des fichiers se fait grâce à un "**BufferedReader**" qui va lire le fichier ligne par ligne.
- L'écriture de texte dans un fichier se fait avec un "**BufferedWriter**".

I. Lecture / Ecriture fichiers externes

❖ Lire un fichier

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ExempleLectureFichier {
    public static void main(String[] args) {
        String pathFile = "cheminPath/fichier.txt";

        try (BufferedReader reader = new BufferedReader(new
                FileReader(filePath))) {
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            System.err.println("Erreur lors de la lecture du fichier : " +
e.getMessage());
        }
    }
}
```

❖ Ecrire dans un fichier

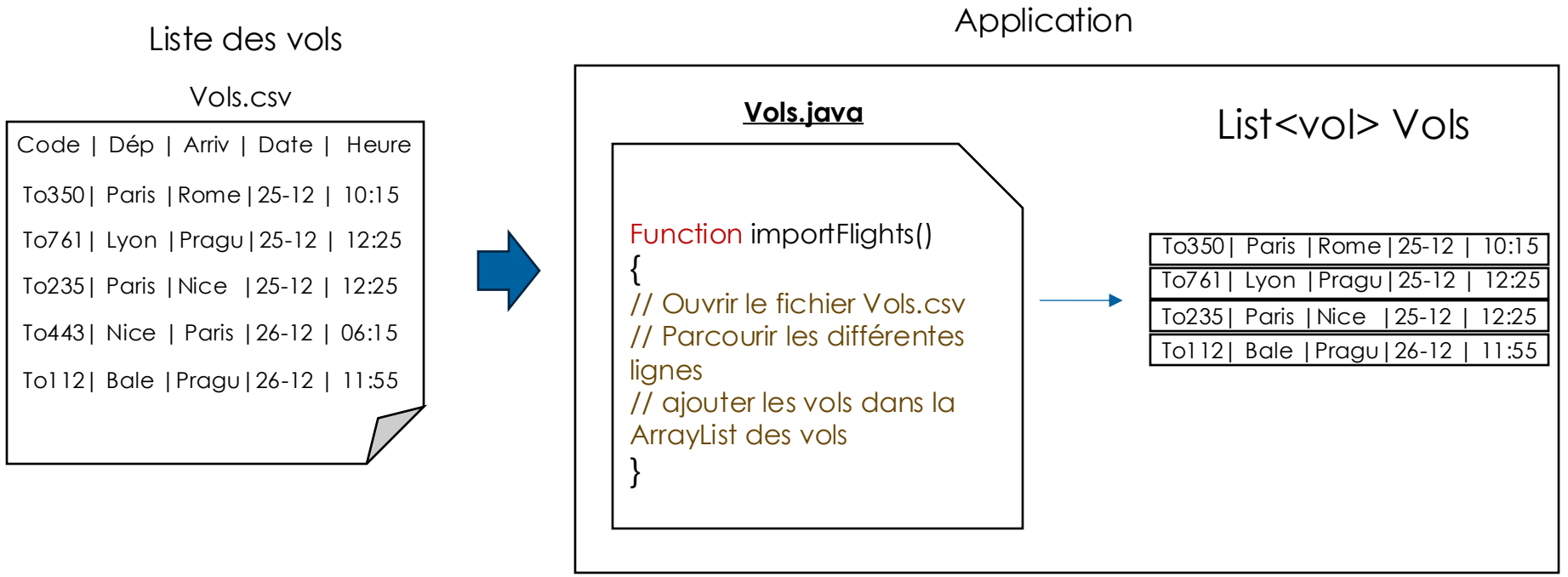
```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class ExempleEcritureFichier {
    public static void main(String[] args) {
        String filePath = "cheminPath/fichier.txt";

        try (BufferedWriter writer = new BufferedWriter(new
FileWriter(filePath, true))) // `true` pour ajouter à la fin
                {
            writer.write("Ceci est un exemple de texte.");
            writer.newLine(); // Ajoute une nouvelle ligne
            writer.write("Une autre ligne de texte.");
                } catch (IOException e) {
            System.err.println("Erreur lors de l'écriture dans le fichier : " +
e.getMessage());
        }
    }
}
```

[Tutoriel sur la manipulation des fichiers avec Java]

I. Lecture / Ecriture fichiers externes

Exemples d'usage des fichiers dans le projet

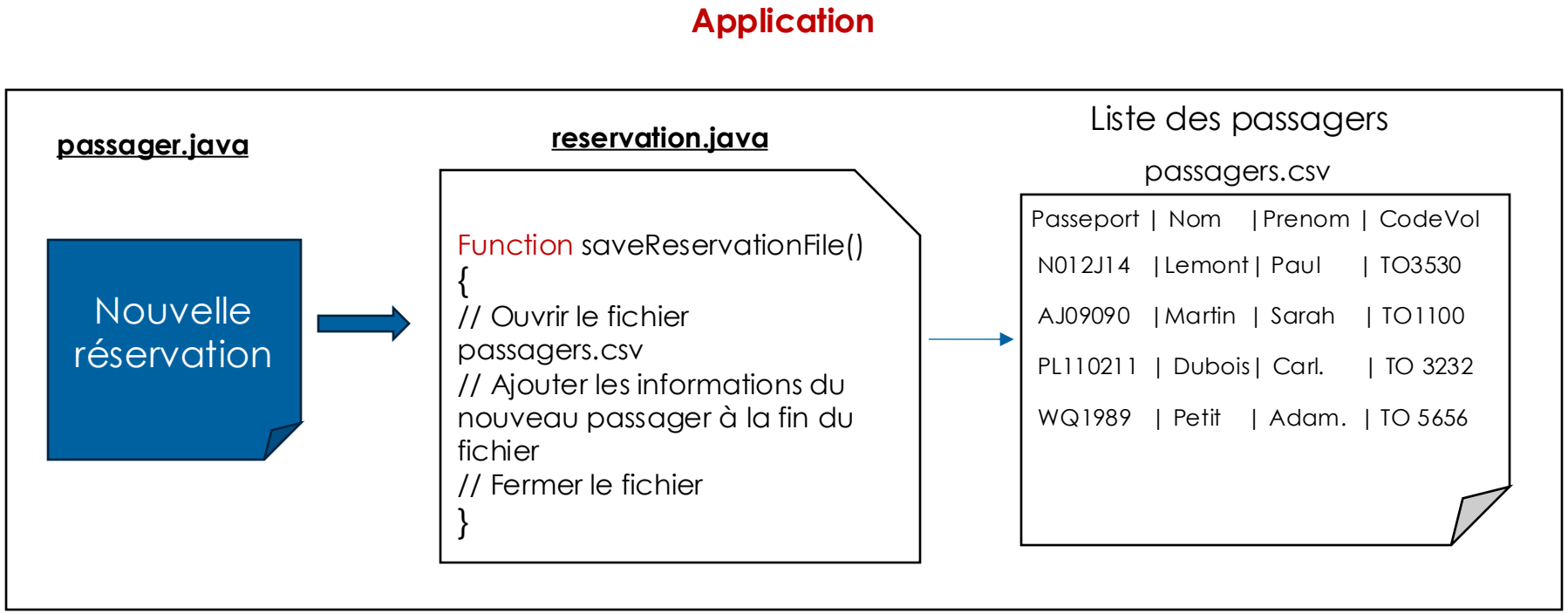
- Lecture depuis un fichier



I. Lecture / Ecriture fichiers externes

Exemples d'usage dans le projet

- Ecriture dans un fichier



I. Lecture / Ecriture fichiers externes

Recommendations

- **Gestion des exceptions :** Toujours encapsuler la lecture/écriture dans un bloc **try-catch** pour éviter des plantages inattendus.
- **Encodage :** Utilisez explicitement un encodage, par exemple en utilisant **FileReader** ou **BufferedWriter** avec des paramètres personnalisés.
- **Performance :** Préférez **BufferedReader** et **BufferedWriter** pour les gros fichiers, car ils sont optimisés pour réduire les appels d'E/S.

II. API Java

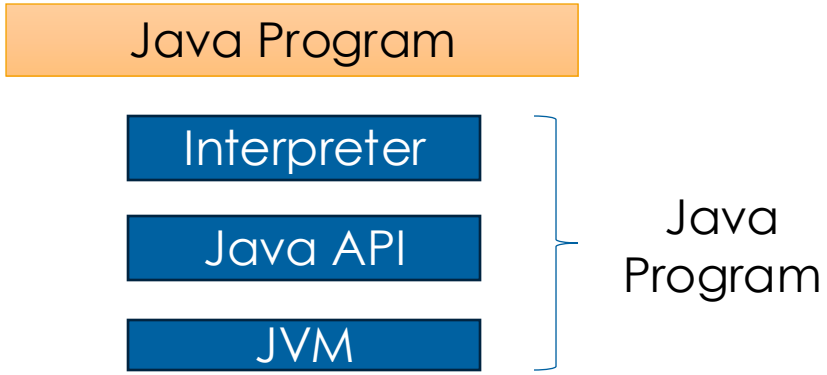
Définitions ?

JDK consiste en trois composants majeurs

- Un compilateur Java est un programme prédéfini qui convertit le langage de code de haut niveau écrit par l'utilisateur en un langage de code d'octet de bas niveau compréhensible par l'ordinateur pendant la compilation.
- JVM (Java Virtual Machine) traite le code d'octet obtenu du compilateur et génère une sortie dans un format lisible par l'utilisateur
- Les API Java sont des progiciels intégrés fournis avec JDK. L'objectif principal de l'API est d'établir une communication entre l'application

II. API Java

Architecture



Application Hardware

Composents fondamentaux de API Java

II. API Java

Fonctionnement

- Dans le développement de logiciels, les API servent de ponts qui relient différents éléments de logiciels, les aidant ainsi à fonctionner ensemble de manière fluide.

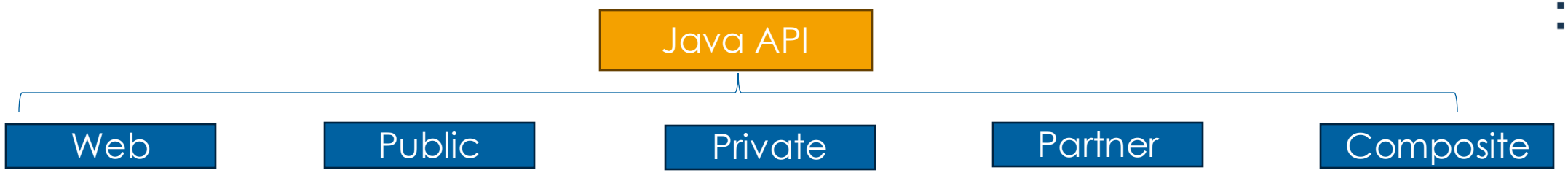
- L'écosystème API robuste de Java permet aux développeurs de créer des applications puissantes et riches en fonctionnalités en utilisant des fonctionnalités prédéfinies.

- Que vous interagissiez avec des API Java standard ou que vous intégriez des API REST tierces, comprendre comment utiliser efficacement les API est une compétence vitale pour tout développeur Java.

II. API Java

Types des API Java

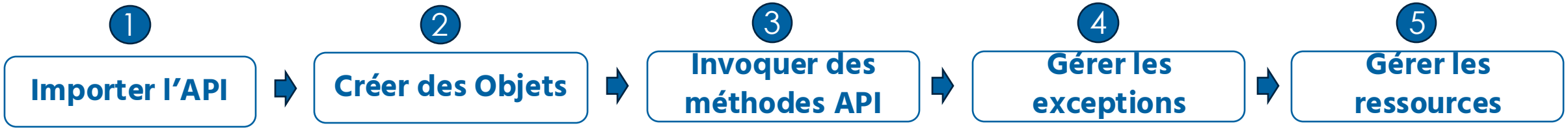
Il existe principalement cinq types d'API en Java



- Les API Web sont accessibles à l'aide du protocole HTTP. Ils incluent les API utilisées pour communiquer avec le navigateur. Il peut s'agir de services tels que les notifications Web et le stockage Web.
- Les API publiques (ou ouvertes) sont des API Java fournies avec le JDK. Ils n'ont pas de restrictions strictes sur la manière dont les développeurs les utilisent
- Les API privées (ou internes) sont développées par une organisation spécifique et ne sont accessibles qu'aux employés qui travaillent pour cette organisation
- Les API partenaires sont considérées comme des API tierces et sont développées par des organisations pour des opérations commerciales stratégiques.
- Les API composites sont des microservices et les développeurs les construisent en combinant plusieurs API de service

II. API Java

Processus d'utilisation d'une API



1. On doit commencer par importer les classes et packages nécessaires pour accéder aux fonctionnalités d'une API.
2. Créer des objets à partir des classes fournies par l'API. Ces objets permettent d'accéder aux méthodes et propriétés proposées par l'API.
3. Utiliser les méthodes fournies par l'API pour effectuer des tâches spécifiques. Il faut lire la documentation de l'API pour comprendre les méthodes disponibles, leurs paramètres, etc.
4. Les API peuvent générer des exceptions pendant l'exécution. Il faut utiliser des mécanismes de gestion des exceptions, tels que les blocs try and catch, pour gérer les erreurs.
5. Certaines API nécessitent une gestion appropriée des ressources, comme la fermeture des connexions ou des flux de base de données.

II. API Java

Protocoles de Services API

- HTTP : c'est un moyen très simple d'accéder aux données, mais si vous demandez beaucoup de données, elles ne vous reviendront pas dans un joli format. Il utilise votre navigateur Internet comme client et vous récupérez un document texte en langage de codage à trier.

- Formats de texte : XML et JSON sont les principaux langages d'accès aux données via une API. Lorsque vous recevrez vos données, vous devrez parcourir le code XML ou JSON pour comprendre ce que le serveur vous a donné.

Processus d'utilisation

- La plupart des API nécessitent une clé API. Une fois qu'on commence d'utiliser une API, on doit vérifier les conditions d'accès dans la documentation.

- Ensuite, on doit probablement passer par une vérification d'identité. Après cela, une chaîne unique de lettres et de chiffres à utiliser est obtenue lors de l'accès à l'API.

- Le moyen le plus simple de commencer à utiliser une API consiste à trouver un client HTTP en ligne, comme REST-Client, Paw ou Postman.

Types de requêtes API

Il existe de nombreux types de requêtes que l'API pourrait gérer, mais voici les plus utilisées :

- GET – renvoie les données du serveur. D'ailleurs, la demande la plus populaire.
- POST – ajoute de nouvelles données sur le serveur. Ce type est souvent utilisé pour l'enregistrement ou le téléchargement de fichiers.
- PUT/PATCH – met à jour les données, requête utile mais pas si recherchée.
- DELETE – supprime les données du serveur.

II. API Java

Top Java API

API	Utilité
JUnit et Mockito	Bibliothèque de test unitaire
Jackson et Gson	Bibliothèques d'analyse JSON
Log4j2 et SLF4j	Bibliothèques de journalisation (Logging)
Xerces et JAXB	Bibliothèques d'analyse XML
Apache POI	Bibliothèques de lecture Excel
JMS et MQ	Bibliothèques de messagerie
Java Mail API	Bibliothèque de mailing

II. API Java

Exemples d'usage dans le projet : Amadeus Flight API

- L'intégration de l'API Amadeus dans le projet vous permet d'interagir avec divers services de voyage, notamment la recherche de vols, les réservations d'hôtel, etc.

- Il faut créer un compte sous https://developers.amadeus.com/ pour pouvoir récupérer des clefs d'essai (API key)
- Une vidéo explicative du mode d'utilisation :

Tutoriel d'utilisation de "Amadeus API"

NB : Vous êtes sur un mode de test, la clef d'essai a un nombre maximum d'utilisation.

Exemples d'usage dans le projet : Amadeus Flight API

Pom.XML

```
<dependencies>

   <dependency>
      <groupId>com.squareup.okhttp3</groupId>
      <artifactId>okhttp</artifactId>
      <version>4.9.3</version>
   </dependency>
   <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-databind</artifactId>
      <version>2.15.2</version>
   </dependency>
</dependencies>
```

II. API Java

Exemples d'usage dans le projet : Amadeus Flight API

Fichier FlightSearch.java

```
import okhttp3.*;
import com.fasterxml.jackson.databind.ObjectMapper;

import java.io.IOException;
import java.util.Map;

public class FlightSearch {
    private static final String API_KEY =
"YK24COXbNx1Dm3t89M8vw5Z3LUCx";
    private static final String API_URL =
"https://test.api.amadeus.com/v2/shopping/flight-offers";

    public static void main(String[] args) throws IOException {
        OkHttpClient client = new OkHttpClient();

        HttpUrl.Builder urlBuilder = HttpUrl.parse(API_URL).newBuilder();
        urlBuilder.addQueryParameter("originLocationCode", "JFK");
        urlBuilder.addQueryParameter("destinationLocationCode", "LAX");
        urlBuilder.addQueryParameter("departureDate", "2024-12-15");
        urlBuilder.addQueryParameter("adults", "1");
```

```
Request request = new Request.Builder()
                .url(urlBuilder.build().toString())
                .addHeader("Authorization", "Bearer " + API_KEY)
                .build();

        Response response = client.newCall(request).execute();
        if (response.isSuccessful()) {
            String jsonResponse = response.body().string();
            ObjectMapper mapper = new ObjectMapper();
            Map<String, Object> flightData =
mapper.readValue(jsonResponse, Map.class);
            System.out.println(flightData);
        } else {
            System.err.println("Error: " + response.code() + " - " +
response.message());
        }
    }
}
```

III. Les tests unitaires en Java : JUnit API

Junit et les tests unitaires

- En Java, les tests unitaires se font le plus souvent avec la bibliothèque JUnit (qui doit s'installer).
- Supposons que l'on a la classe suivante (dans un fichier Weapon.java) :

```
public class Weapon {
        private String name ;
        public Weapon(String name){
                this.name = name ;
        }
        public String getName(){return name ;}
}
```

Junit et les tests unitaires

Dans un fichier WeaponTest.java, on a le code suivant. La ligne "@Test" indique que la fonction qui suit est un test. Le "@" désigne une annotation.

```
import org.junit.jupiter.api.Test ;
import static org.junit.jupiter.api.Assertions.*
class WeaponTest{
        @Test
        void testCorrectName(){
                Weapon weapon = new Weapon("sword") ;
                assertEquals("sword", weapon.getName()) ;
        }
}
```

Junit et les tests unitaires

Structure d'un projet JUnit :

```
src
├── main
│   ├── java
│   │   └── UnitTestDemo.java
│   └── resources
└── test
    └── java
        └── UnitTestDemoTest.java
```

Junit et les tests unitaires

- Un test unitaire doit contenir une assertion : c'est elle qui détermine si le test passe ou non.
- Il existe plusieurs types d'assertions. Les plus courants sont assertEquals, assertFalse, assertArrayEquals et assertThrows.
- La plupart du temps, les paramètres de ces assertions sont la valeur attendue et la valeur obtenue.
- Le cas particulier d'assertThrows sera vu lorsque l'on traitera des Exceptions (il fait appel aux lambda expressions)

III. Les tests unitaires en Java : JUnit API

Junit et les tests unitaires

Tutoriel JUnit sur JetBreains :

- https://www.jetbrains.com/help/idea/junit.html

Exemple de manipulation de JUnit :

- https://www.jetbrains.com/guide/java/tutorials/marco-codes-junit/introduction/

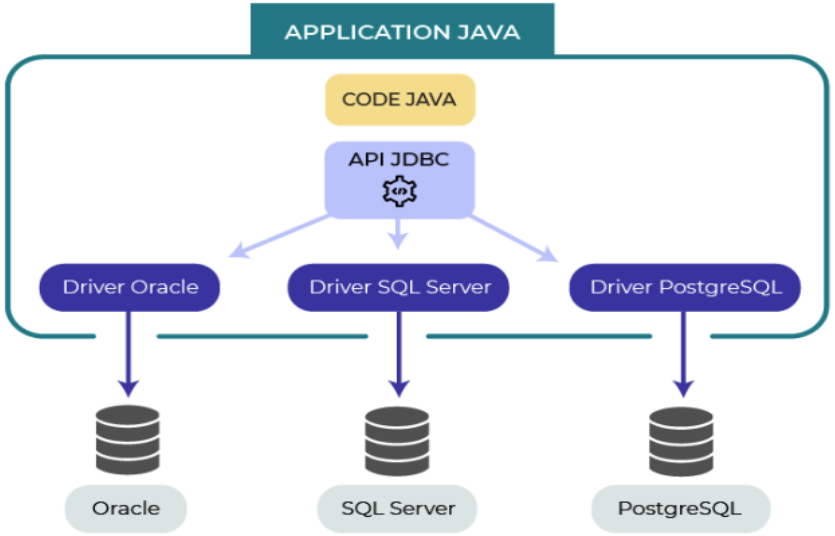
IV. Bonus : Bases de données avec Java

Les BD servent à assurer la pertinence des données d'une application informatique.

> Les bases de données sont des programmes informatiques qui ont pour but de stocker des données de façon structurée.

- Face à la multitude des bases de données existantes (Oracle, SQL Server, PostgreSQL, MySQL et bien d'autres), une solution globale s'est avérée nécessaire.

- Au sein du langage Java, cette solution est l'API JDBC (Java Database Connectivity). Cette API est un ensemble de classes permettant de communiquer avec des bases de données, indépendamment du modèle de bases de données utilisé.



IV. Bonus : Bases de données avec Java

1. Tutoriel de la connexion d'une BD avec Intellij Idea :
https://www.jetbrains.com/help/idea/connecting-to-a-database.html

2. Tutoriel de la configuration d'une BD avec Intellij Idea :
https://www.jetbrains.com/help/idea/configuring-database-connections.html

3. Vidéo : Connecter MySQL à Java dans IntelliJ 2024
https://www.youtube.com/watch?v=9ntKSLLDeSs

IV. Bonus : Bases de données avec Java

Exemples d'usage dans le projet

- Ecriture dans un fichier