INTRODUCTION AU DÉVELOPPEMENT EN JAVA

DIDIER ERIN

ARTISAN DÉVELOPPEUR

DIDIER.ERIN@ERINTEK.COM

version 202406

DÉROULEMENT DE L'INITIATION

Plan

- INTRODUCTION
- LES CONCEPTS DE BASE
- UN PEU DE MODÉLISATION
- LA ROUE EXISTE DÉJÀ
- L'INCONTOURNABLE SPRING BOOT
- POUR ALLER PLUS LOIN

Fil rouge

DoctoCrest

Application Doctolib simplifiée pour les habitants de Crest

INTRODUCTION

INTRODUCTION

- C’est quoi Java ?
- Et la JVM ?
- Un peu d’histoire
- Avantages et inconvénients de Java
- Environnement de développement
  - Le JDK
  - IntelliJ / Eclipse
  - Maven
  - Git
  - Docker

INTRODUCTION

- C’est quoi Java ?
- Et la JVM ?
- Un peu d’histoire
- Avantages et inconvénients de Java
- Environnement de développement
  - Le JDK
  - IntelliJ / Eclipse
  - Maven
  - Git
  - Docker

INTRODUCTION

Et la JVM ?

- Java Virtual Machine
- Exécution du byte code
- Garbage Collector (GC)
- Optimisation Just-In-Time (JIT)
- Sécurité

- HotSpot JVM (Oracle)
- OpenJ9 (Eclipse)
- GraalVM

- Class Loader : chargement des classes en mémoire
- Execution Engine : exécution du code traduit en byte code
- Runtime Data Areas : gestion des zones de mémoire

INTRODUCTION

Un peu d’histoire

- Initié par James Sun Microsystems en 1991 : projet Oak
- Renommé Java en 1994, la marque Oak existait déjà
- Première version officielle : mai 1995

INTRODUCTION

Un peu d’histoire

- Initié par James Sun Microsystems en 1991 : projet Oak
- Renommé Java en 1994, la marque Oak existait déjà
- Première version officielle : mai 1995

INTRODUCTION

Avantages

- Multiplateforme
- Robuste et fiable
- Sécurité
- Multithreading
- Gestion automatisée de la mémoire
- Vaste librairie standard et écosystème
- Rétrocompatibilité
- Communauté

Inconvénients

- Courbe d’apprentissage longue
- Ecosystème pléthorique
- Verbosité
- Performance … quoi que
- Coût en mémoire
- Développement d’interface graphique

INTRODUCTION

Environnement de développement

- JDK
  - Java Development Kit embarque :
    - Le compilateur
    - La JVM
    - Les bibliothèques standard
    - Les outils de développement
- Maven
  - Gestion de projet Java
    - Gestion des dépendances
    - Compilation
    - Tests
    - Génération des livrables
    - Déploiement
  - pom.xml : fichier de configuration
  - Alternative : Gradle

INTRODUCTION

Environnement de développement

- IntelliJ
  - IDE avec des fonctionnalités très poussées dans l'écriture de code Java
- Git
- Docker ?
- Postman
  - Pour tester les API web

FIL ROUGE

Installation de l'environnement de développement

- Tout au long du fil rouge, nous allons poser les bases pour développer une application de type Doctolib : DoctoCrest
- https://github.com/ZandoliDev/doctocrest
  - Chaque branche correspond à une partie du cours

FIL ROUGE

Installation de l’environnement de développement

- Installation du JDK
- Installation de IntelliJ version Community
- Installation de Maven
- Installation de Git

FIL ROUGE

Initialisation du projet

- Avec Maven

```
$ mvn archetype:generate -DgroupId="com.example" -DartifactId=doctocrest
-DarchetypeArtifactId="Maven-archetype-quickstart" -DinteractiveMode=false
```

  - Qu'à générer MAVEN ?

- Vérifier que ça fonctionne

```
$ cd doctocrest

$ java src/main/java/com/example/App.java

Hello World!
```

FIL ROUGE

Initialisation du projet

- Initialisation du versionning

```
$ git init
Initialized empty Git repository in …./doctocrest/.git/
$  git config user.name "Didier Erin"
$ git config user.email "didier.erin@erintek.com"
$ $ git commit –a -m "Initialisation du projet doctocrest"
```

- Importer le projet dans IntelliJ
- Analyse du fichier App.java

CONCEPTS DE BASE

CONCEPTS DE BASE

- Variables, types de données, opérateurs
- Structures de contrôle
- Fonctions
- Gestion des entrées / sorties
- Gestion des erreurs
- Structures de données de base

FIL ROUGE

Variables, types de données, opérateurs

- Afficher la tracer suivante

Bienvenue sur DoctoCrest !

Patient : Amina Lopez

Âge : 28 ans

Taille : 1.70 mètres

A un rendez-vous : true

Médecin : Dr. Chen Wong (Cardiologue)

Durée du rendez-vous : 45 minutes

Coût du rendez-vous : 80.0 euros

Le coût total des rendez-vous cette année est de 240.0 euros.

Le patient est-il éligible pour le traitement ? true

Le patient a-t-il un rendez-vous aujourd'hui et est-il éligible pour le traitement ? true

Nombre de rendez-vous restants aujourd'hui : 5

Un rendez-vous vient de se terminer. Rendez-vous restants : 4

FIL ROUGE

Structures de contrôle

- Rendre le code plus dynamique avec
  - Des conditions *If*
  - Des boucles *For*
  - Des boucles *While*
  - Des boucles *Do / While*
  - Des instructions *Switch / Case*

CONCEPTS DE BASE

Les fonctions

- Bloc autonome de code réutilisable
- Peut prendre en entrée des paramètres
- Peut renvoyer une valeur, de type *void* si absente
- Peut améliorer la lisibilité du code

```
<type de retour> <nom de fonction>(<type de paramètre 1> <nom du paramètre 1> ...) {
    corps de la fonction
    return <valeur de retour> // facultatif si méthode de type void
}
```

FIL ROUGE

Les fonctions

- Utiliser du code réutilisable grâce à des méthodes :
  - Sans valeur de retour et sans paramètres
  - Sans valeur de retour et avec des paramètres
  - Avec valeur de retour et sans paramètres
  - Avec valeur de retour et des paramètres

- Comparer la lisibilité de code avant et après l'utilisation des méthodes

CONCEPTS DE BASE

Gestion des entrées (in) / sorties (out)

- Communiquer avec les systèmes externes
  - Console ⇨ System.in ou Scanner / System.out
  - Système de fichiers ⇨ FileReader / FileWriter
  - Flux de bytes (fichiers binaires, images, ...) ⇨ InputStream / OutputStream
  - Réseau ⇨ Socket (client réseau) / ServerSocket (serveur réseau)
- Améliorer les performances en utilisant la mémoire
  - Mémoire ⇨ ByteArrayInputStream / ByteArrayOutputStream
  - Buffer ⇨ BufferReader / BufferWriter

FIL ROUGE

Gestion des entrées / sorties

- Permettre à l'utilisateur de saisir des informations
- Que se passe-t-il si vous saisissez une chaîne de caractères à la place de l'âge du patient ?
- IMPORTANT : PENSEZ A FERMER UNE RESSOURCE OUVERTE
  - Avec la méthode close()

CONCEPTS DE BASE

Gestion des erreurs

- Erreurs de compilations
- Erreurs irrécupérables : Error
  - Erreurs graves au niveau de la JVM
    - InternalError / StackOverFlowError / OutOfMemoryError
- Erreurs récupérables : Exception
  - Erreurs prévisibles qui peuvent être gérées pour l'application continue de fonctionner normalement

CONCEPTS DE BASE

Gestion des erreurs

- Erreurs de compilations
- Erreurs irrécupérables : Error
  - Erreurs graves au niveau de la JVM
    - InternalError / StackOverFlowError / OutOfMemoryError
  - Entraîne systématiquement le crash de l'application
- Erreurs récupérables : Exception
  - Erreurs prévisibles qui peuvent être gérées pour l'application continue de fonctionner normalement
  - RuntimeException : erreur prévisible non gérée, elle entraîne de le crash de l'application

CONCEPTS DE BASE

Gestion des erreurs

- Récupérer une erreur : try / catch / finally
- finally permet de réaliser des traitements qui seront forcément exécutés, même en cas d'erreur
    - Intéressant pour fermer les ressources ouvertes, même en cas d'erreur

```
try {
    traitement pouvant générer une erreur de type <type d'exception>
} catch (<type d'exception> ex) {
    comportement à adopter en cas d'erreur
} finally {
    traitement exécuté dans tous les cas, même en cas d'erreur
}
```

CONCEPTS DE BASE

Gestion des erreurs

- Indiquer une erreur potentielle

```
public void maMethode() throws <type d'exception> {
    traitement pouvant générer une erreur de type <type d'exception>
}
```

  - Le traitement qui appel cette méthode doit au choix
    - gérer l'erreur avec un try / catch
    - Déclarer l'erreur potentielle à son tour avec throws

FIL ROUGE

Gestion des erreurs

- Modifier l’application pour qu’elle invite l’utilisateur à saisir de nouveau l’âge du patient s’il est mal renseigné

CONCEPTS DE BASE

Structures de données de base

- Manière d’organiser les données manipulées
- Optimiser l’accès aux données
- Optimiser la manipulation des données

- Les tableaux
  - Collection d'éléments de même type, stockés contiguëment en mémoire
  - Accès rapide aux éléments par leur index
  - Ex: int[] nombres = {1, 2, 3, 4, 5};

CONCEPTS DE BASE

Structures de données de base

- Les listes
  - Collection ordonnée d'éléments pouvant contenir des doublons
  - Manipulation flexible des collections de données (ajout, suppression, tri)
  - Ex: List<String> noms = new ArrayList<>();
- Les ensembles
  - Collection d'éléments uniques, sans ordre particulier
  - Éliminer les doublons, tester l'appartenance
  - Ex: Set<String> ensembleNoms = new HashSet<>();

CONCEPTS DE BASE

Structures de données de base

- Les tables de hachage
  - Collection de paires clé-valeur
  - Association rapide de valeurs à des clés, recherche par clé
  - Ex: Map<String, Integer> annuaire = new HashMap<>();
- Les piles
  - Collection suivant le principe LIFO (Last In, First Out)
  - Gestion des appels de fonctions, navigation (par exemple, retour en arrière)
  - Ex: Stack<String> pile = new Stack<>();

CONCEPTS DE BASE

Structures de données de base

- Les files
  - Collection suivant le principe FIFO (First In, First Out)
  - Gestion des tâches, file d'attente de traitement
  - Ex: Queue<String> file = new LinkedList<>();
- Les Deques (Double-ended Queues)
  - File d'attente où les éléments peuvent être ajoutés ou retirés à la fois du début et de la fin
  - Structures flexibles pour la manipulation de collections d'éléments
  - Ex: Deque<String> deque = new ArrayDeque<>();

CONCEPTS DE BASE

Structures de données de base

- Les énumération (enum)
  - Type particulier de données
  - Permet de figer la liste des valeurs possibles pour un type de données

FIL ROUGE

Structures de données de base

- Cas de l’énumération (enum)

```
enum Specialite {
    CARDIOLOGUE,
    DERMATOLOGUE,
    PEDIATRE,
    NEUROLOGUE,
    GYNECOLOGUE,
    GENERALISTE,
    ORTHOPEDISTE,
    ;
  }
```

- Permet de gérer une liste finie et figée de constantes connues

FIL ROUGE

Structures de données de base

- Utiliser deux tableaux
  - Un pour stocker le nom des praticiens
  - Un pour stocker la spécialité correspondante à chaque praticien

- Générer la création de 100 000 000 de rendez-vous sur des spécialités aléatoires
  - Random random = new Random();
    int value = random.nextInt(max + min) + min
  - Penser à désactiver les messages dans la console
  - Enlever tout le code superflux

FIL ROUGE

Structures de données de base

- Calculer le temps de traitement pour les 1M de rendez-vous

```
Instant debut = Instant.now();
...
Instant fin = Instant.now();
Duration duree = Duration.between(debut, fin);
long secondes = duree.getSeconds() % 60;
long millis = duree.toMillis() % 1000;
System.out.printf("Durée de l'opération : %ds %dms", secondes, millis)
```

- Utiliser une table de hachage pour faire correspondre la spécialité au praticien
- Utiliser une table de hachage pour faire correspondre les prix à la spécialité
- Comparer la taille du code et le temps d’exécution

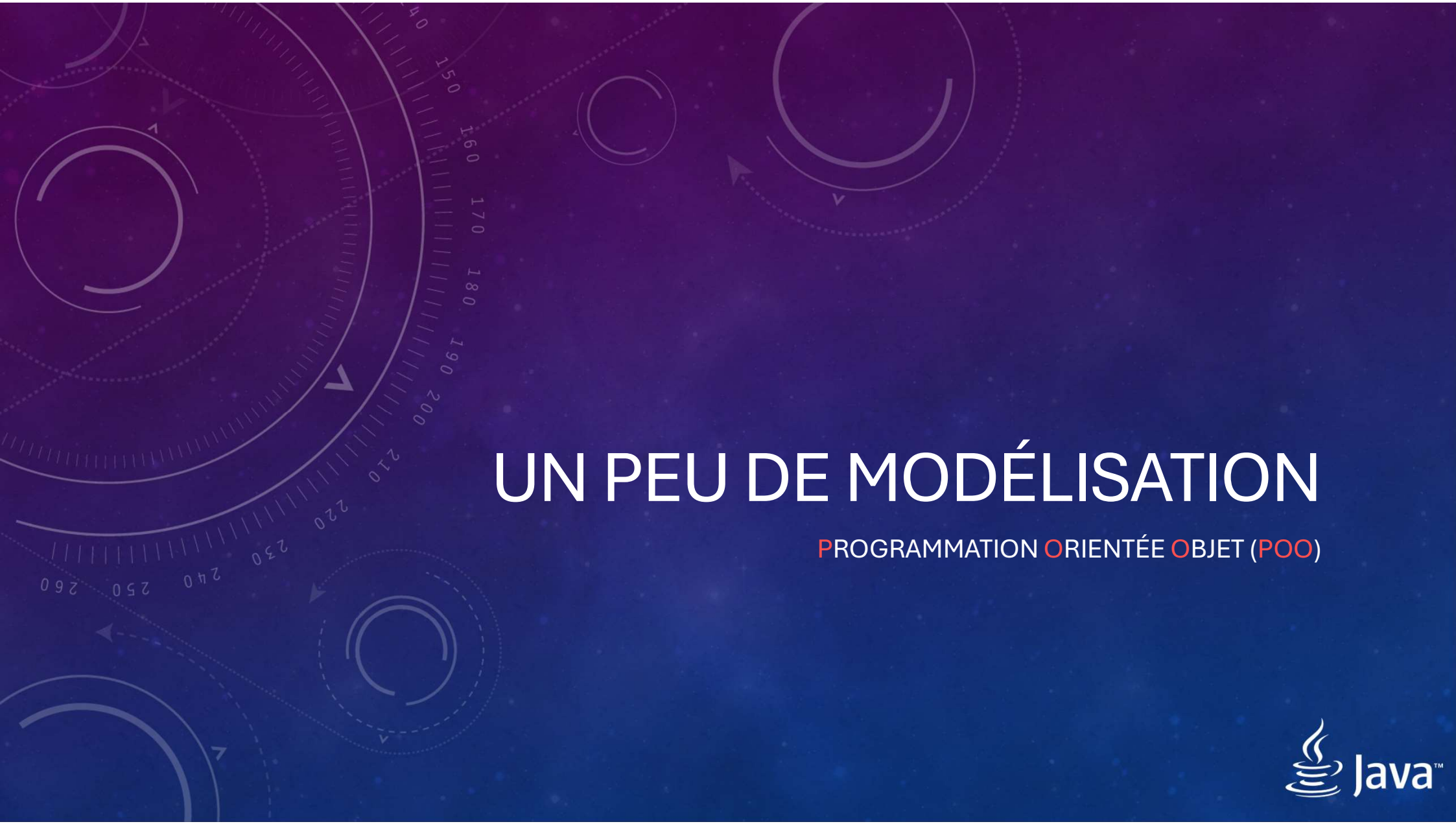
FIL ROUGE

Structures de données de base

- Gérer la liste possible des spécialités avec des enum

UN PEU DE MODÉLISATION

PROGRAMMATION ORIENTÉE OBJET (POO)



UN PEU DE MODÉLISATION

- Pourquoi modéliser ?
- Principes de base de la POO
  - Encapsulation
  - Abstraction
  - Héritage
  - Polymorphisme

UN PEU DE MODÉLISATION

Pourquoi modéliser ?

- Les limites de la programmation procédurale
  - Gestion de la complexité
  - Réutilisabilité du code
  - Maintenance et évolution
- Les solutions de la programmation orientée objet (POO)
  - Décomposition du système en petits objets gérables
  - Structuration des responsabilités
  - Facilitation de la réutilisation du code grâce à une organisation modulaire
  - Facilitation de la représentativité de l'application

UN PEU DE MODÉLISATION

Principes de base de la POO - Encapsulation

- Encapsulation
  - Regrouper des données et des comportements dans une entité autonome
  - Intégrité des données
- En Java
  - On crée des classes : class
    - Avec des attributs
    - Avec un/des constructeur(s)
    - Avec des méthodes
      - Getter : retourne un attribut de la classe
      - Setter : modifie la valeur d'un attribut de la classe

```
public class MaClasse {
        String propriete1
        Integer propriete2

        public MaClasse(String propriete1, Integer
propriete2) {
                this.propriete1 = propriete1;
                this.propriete2 = propriete2;
        }

        public void comportement1() {
                ...
        }
}
```

UN PEU DE MODÉLISATION

Principes de base de la POO - Encapsulation

- Une classe est la description d'un type de données personnalisé
- On y décrit :
  - Les données stockées par la classe et leur type
  - La façon de créer nouvelle instance
  - Les traitements que peuvent réaliser la classe

UN PEU DE MODÉLISATION

Principes de base de la POO - Encapsulation

- La classe
  - Les propriétés
    - Toutes les informations relatives à l'objet représentée par la classe
  - Le(s) constructeur(s)
    - Définit la/les manières d'initialiser une instance de la classe
    - Par défaut, toute classe possède un constructeur par défaut, il ne prend pas de paramètre
  - Le(s) méthodes
    - Regroupe tous les traitements réalisables par la classe
    - Elles peuvent servir en interne (private)
    - Elles peuvent être appelée de l'extérieur (public / package)

UN PEU DE MODÉLISATION

Principes de base de la POO - Encapsulation

- Cas particulier des enum en Java
- Dans une enum, on définit
  - Les seules instances possibles
  - Les éléments d'une classe
    - Attributs
    - Constructeurs
    - Méthodes

```
public enum MonEnum {
        INSTANCE1("valeur1", 10), // => new MonEnum INSATANCE1(...)
        INSTANCE2("valeur2", 20), // => new MonEnum INSATANCE2(...)
        ...;

        public MonEnum(String propriete1, Integer propriete2) {
                this.propriete1 = propriete1;
                this.propriete2 = propriete2;
        }

        public void comportement1() {
                ...
        }
}
```

FIL ROUGE

Principes de base de la POO - Encapsulation

- Créer une classe qui représente un rendez-vous avec les informations suivantes :
  - Le patient
  - Le praticien
- Cette classe fournit
  - Une méthode qui permet d'afficher le récapitulatif du rendez-vous
  - Une méthode qui donne le prix du rendez-vous

UN PEU DE MODÉLISATION

Principes de base de la POO – Abstraction

- Abstraction
  - Masquer les détails d'implémentation
  - Masquer la complexité
  - Faciliter la compréhension des objets
- En Java
  - On crée des interfaces : interface
    - Les méthodes à implémenter
  - On crée des classes qui implémentent l'interface
    - class MaClasse implements MonInterface

```
public interface MonInterface {
        void comportement1();
        String comportement2(int param);
}

public class MaClasse implements MonInterface {
        public void comportement1() {
                ...
        }
        public String comportement2() {
                ...
        }
}
```

FIL ROUGE

Principes de base de la POO – Abstraction

- Rajouter la notion d'*Utilisateur* : un utilisateur de l'application peut
  - fournir son nom : *String getNom()*
  - Afficher ses informations : *void afficherInformations()*
- Modifier les classes Patient et Praticien pour qu'elles deviennent des implémentations de l'interface *Utilisateur*
- Qu'observez-vous ?

UN PEU DE MODÉLISATION

Principes de base de la POO - Héritage

- Héritage
  - Permet de hiérarchiser des concepts du plus générique au plus précis
    - Rectangle > Carré
  - La classe fille hérite de la classe mère
    - Les attributs
    - Les méthodes
  - La classe fille peut spécialiser modifier le comportement défini par sa classe mère
- En Java
  - On étend une autre classe : extends
    - class MaClasse implements MonInterface

```
public class Rectangle {
        private int longueur;
        private int largeur;
        public Rectangle(int longueur, int largeur) {
                ...
        }
        public int superficie() {
                return longueur * largeur;
        }
}
public class Carre extends Rectangle {
        public Carre(int longueur) {
                super(longueur, longueur);
        }
}
```

UN PEU DE MODÉLISATION

Principes de base de la POO - Héritage

- En java
  - Les attributs de la classe mère sont accessibles par la classe fille si la classe mère le permet
    - Possible pour les attributs avec les visibilités public, package ou protected
    - Impossible pour les attributs private
  - Même principe pour l'accès aux méthodes de la classe mère
  - Les constructeurs de la classe fille doivent utiliser un des constructeurs de la classe mère
  - La classe fille peut s'enrichir en définissant des attributs, constructeurs, méthodes propres

UN PEU DE MODÉLISATION

Principes de base de la POO - Polymorphisme

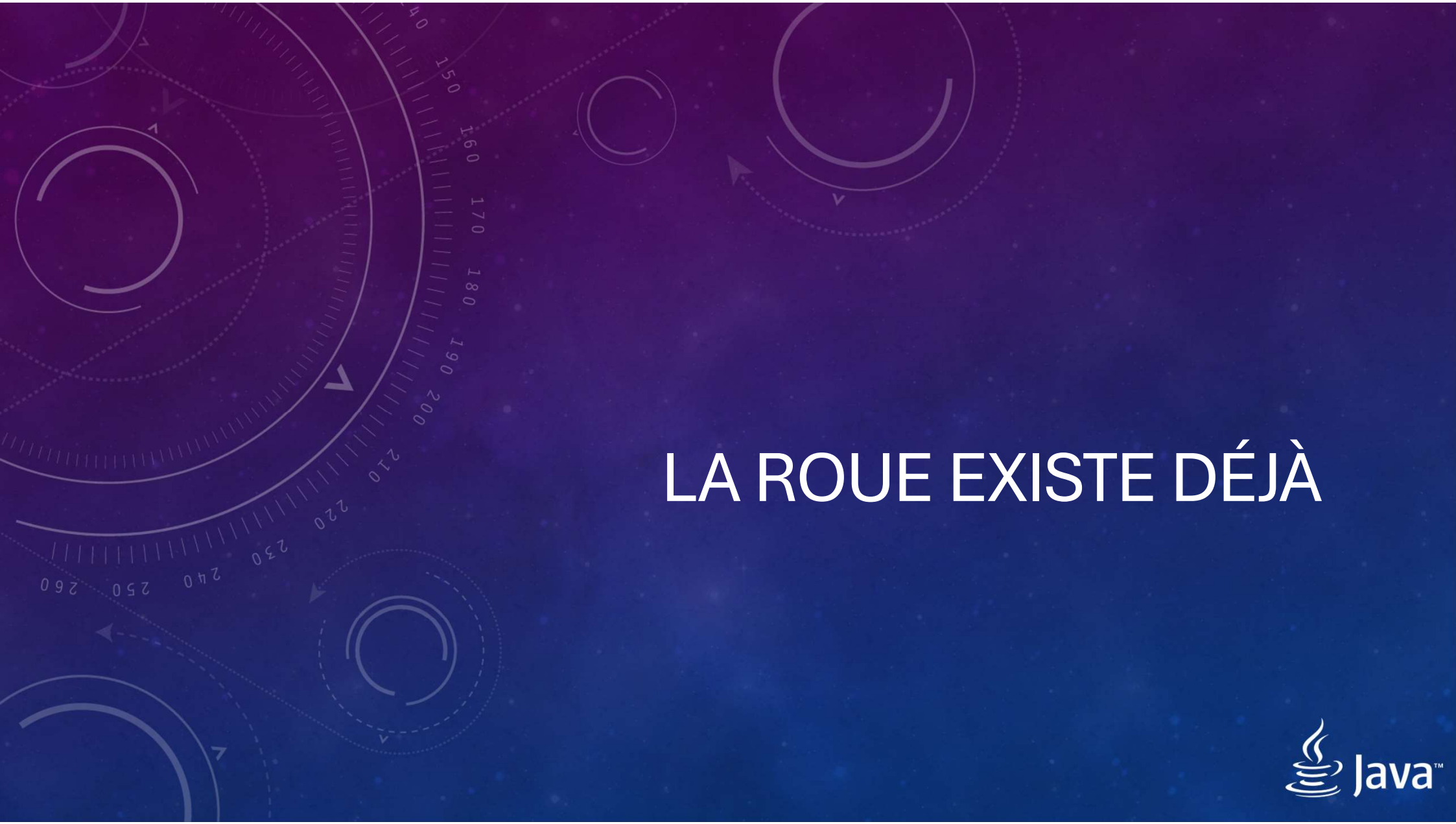
- Polymorphisme
  - Permet d'utiliser une même interface pour différentes classes
  - Chaque classe pourra définir le comportement à adopter par les méthodes de l'interface
  - A l'exécution, le code de la forme la plus spécialisée sera exécutée

FIL ROUGE

Principes de base de la POO

- Créer la classe *Personne* qui implémente l'interface Utilisateur
- Modifier les classes *Patient* et *Praticien* pour qu'elles étendent la classe *Personne*
- Changer les variables de type Patient en Personne dans la classe App
- Changer les variables de type Praticien en Personne dans la classe App
- Que se passe-t-il ?

LA ROUE EXISTE DÉJÀ



LA ROUE EXISTE DÉJÀ

- Un tour d’horizon des APIs Java
- La gestion des logs
- Aperçu de l’API Stream
- Les lambdas

LA ROUE EXISTE DÉJÀ

Un tour d’horizon des APIs Java

- API : Application Programming Interface
- Façade d’utilisation d’une application
- Masque la complexité d’implémentation
- Améliore la modularité et la réutilisabilité
- Accroît l’interopérabilité

LA ROUE EXISTE DÉJÀ

Un tour d'horizon des APIs Java

- API des collections : fournit de nombreux outils de base pour structurer les données
  - Listes, tables de hachage, ensembles, … (voir sujet abordé dans les concepts de base)
- API de concurrence : pour gérer les traitements en parallèle
  - Executor, Future, Semaphore
- API de flux d'entrée/sortie : pour la lecture et l'écriture
  - InputStream, OutputStream, Reader, Writer, …
- API de gestion de fichiers : opérations sur le système de fichier
  - Path, Files, FileSystem, …
- API de la bibliothèque standard : pour toutes opérations basiques (math, …)
  - Math, String, Object, …

LA ROUE EXISTE DÉJÀ

La gestion des logs

- Pourquoi gérer des logs ?
  - Comprendre ce qui a pu se passer en cas de bug
  - Difficile d'exploiter les logs à travers la console en cours d'exécution d'une application … et après son exécution ?
  - Garder une trace des évènements de l'application
  - Permet de faire du monitoring d'application

LA ROUE EXISTE DÉJÀ

La gestion des logs

- Grâce à des framework de logging
  - Log4j / Logback
  - Offre différentes fonctionnalités de gestion
    - Définition des différentes ressources où écrire les logs (console, fichiers, ...)
      - Stratégie possible de gestion de la taille des fichiers générés
    - Définition du formatage des logs
    - Définition du niveau de logs à tracer (DEBUG, INFO, WARN, ERROR, FATAL)

FIL ROUGE

La roue existe déjà

- Ajouter la configuration nécessaire pour gérer des logs avec Log4j2
- Remplacer les appels à System.out.printxxx par l'utilisation du framework de logging
- Voir branche 03-01-la-roue-existe-logs

LA ROUE EXISTE DÉJÀ

Aperçu de l'API Stream

- Stream : abstraction de traitement sur des collections
  - Filtrage, tri, transformation de données
- Pipeline d'opérations
  - Un traitement par étape
- Opération intermédiaire
  - Applique un traitement pour fournir un nouveau Stream en sortie
    - Filter, map, sort, ...
  - Opération stockée en mémoire en attendant une opération terminale
- Opération terminale
  - Applique un traitement pour fournir un objet non-Stream
  - Déclenche toute la chaîne de traitements intermédiaires

LA ROUE EXISTE DÉJÀ

Aperçu de l'API Stream

- Stream : abstraction de traitement sur des collections
  - Filtrage, tri, transformation de données
- Pipeline d'opérations
  - Un traitement par étape
- Opération intermédiaire
  - Applique un traitement pour fournir un nouveau Stream en sortie
    - Filter, map, sort, ...
  - Opération stockée en mémoire en attendant une opération terminale
- Opération terminale
  - Applique un traitement pour fournir un objet non-Stream
  - Déclenche toute la chaîne de traitements intermédiaires

LA ROUE EXISTE DÉJÀ

Les lambdas

- Manière concise de décrire une implémentation d'interface qui ne propose qu'une seule méthode
- Améliore
  - La lisibilité
  - La concision
  - La flexibilité
- Notations :
  - (paramètre) -> { traitement }
  - paramètre -> traitement sur une ligne
  - (Type paramètre) -> ...

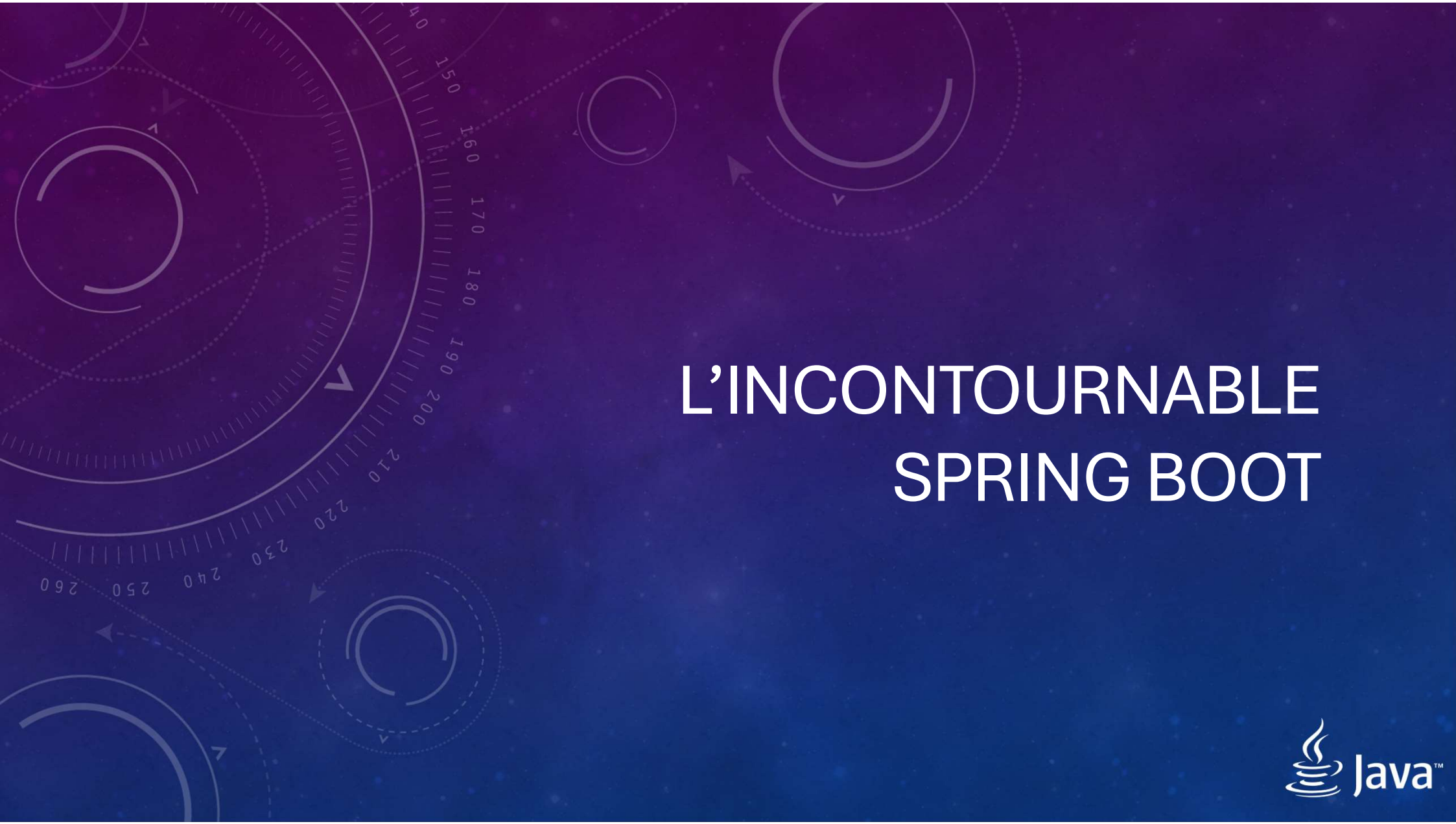
```
Interface Formattable<Personne, String> {
        String formater(Personne personne);
}
Formattable personneFormattable = (personne) -
> {
        return "%s %s".formatted(personne.prenom,
personne.nom);
}
Ou
Formattable personneFormattable = personne ->
"%s %s".formatted(personne.prenom,
personne.nom);
```

FIL ROUGE

La roue existe déjà

- Modifier l’application pour utiliser les Stream à la place des boucles

L'INCONTOURNABLE SPRING BOOT



L'INCONTOURNABLE SPRING BOOT

- Présentation de Spring et Spring Boot
- Principes de base d'une API REST
- Exposer une API REST avec Spring Boot
- Communications avec la base de données

L'INCONTOURNABLE SPRING BOOT

Présentation de Spring et Spring Boot

- Qu'est-ce qu'un framework ?
  - Outil qui fournir une structure dans le but de faciliter et accélérer la création d'une application
  - C'est le framework qui appelle le code
- Qu'est-ce qu'une librairie ?
  - Outil réutilisable pour réaliser un traitement donné
  - Le code appelle la librairie

L'INCONTOURNABLE SPRING BOOT

Présentation de Spring et Spring Boot

- Spring
  - Framework open source pour faciliter le développement d'applications Java
- Plusieurs composants
  - Inversion of Contrôle Container (IoC Container) : permet de gérer le cycle des objets qui composent l'application
  - Dependency Injection (DI) : permet de configurer l'implémentation à utiliser en fonction d'un contexte
  - Spring MVC : facilite le développement d'application web (API par exemple)
  - Spring BOOT : accélère la configuration des applications pour permettre un développement plus rapide
  - Spring data : facilite l'échange avec les bases de données
  - Spring Security : facilite la mise en place de mécanisme de sécurité (authentification, autorisations, ...)

L'INCONTOURNABLE SPRING BOOT

Présentation de Spring et Spring Boot

- Spring Boot
  - Facilite le développement d'applications
  - Amène une configuration complète par défaut pour l'ensemble des composants Spring utilisés
  - Configuration Maven simple
  - Détection automatique des fichiers de configuration
    - application.properties
      - Au format clé=valeur
    - Ou
    - application.yaml ou application.yml
      - Au format YAML

L’INCONTOURNABLE SPRING BOOT

Présentation de Spring et Spring Boot

- Spring MVC
  - Facilite le développement d’applications web selon le modèle MVC
- MVC (Model-View-Controller)
  - Design-pattern :
  - Model : les données et la logique métier de l'application
  - View : la présentation des données (interface web par exemple)
  - Controller : Gère les requêtes entrantes, les traite et retourne les réponses adéquates en coordination avec le modèle et la vue

L'INCONTOURNABLE SPRING BOOT

Principes de base d'une API REST

- API (Application Programming Interface)
  - Interface permettant de faire communiquer deux systèmes
  - Couche d'abstraction sur la complexité des traitements de l'application

L'INCONTOURNABLE SPRING BOOT

Principes de base d'une API REST

- REST (Representational State Transfer)
  - Architecture de conception d'application web basé sur le protocole HTTP
  - Permet la création de systèmes distribués efficaces et extensibles
  - Plusieurs principes et contraintes
    - Stateless : chaque requête contient toutes les informations (on ne stocke aucune donnée de session)
    - Client / Serveur : client et serveur sont distincts. Le serveur gère la logique métier et la persistance des données
    - Cacheable : certaines réponses peuvent être mis en cache par le client pour limiter les appels réseaux
    - Layered System : le serveur qui expose l'API peut masquer une couche de serveurs, chacun jouant un rôle spécifique (sécurité, cache, répartition de la charge, ...)
    - Uniform Interface : les différentes ressources traitées par l'application sont gérées de manière uniforme
      - Simplification et décomposition de l'architecture
      - L'interface de chaque ressource peut évoluer de manière indépendante

L'INCONTOURNABLE SPRING BOOT

Principes de base d'une API REST

- Choix du verbe HTTP
  - GET : récupérer les informations d'une ressource
  - POST : créer une nouvelle ressource
  - PUT : mettre à jour une ressource
  - DELETE : supprimer une ressource
  - PATCH : mettre à jour partiellement une ressource
- Exemples d'URL avec leur signification
  - GET http://adresse.serveur/livres : récupérer la liste les livres
  - POST http://adresse.serveur/livres : ajouter un ou plusieurs nouveau(x) livre(s)
  - GET http://adresse.serveur/livres/{id} : récupérer les informations du livre identifié par {id}
  - PUT http://adresse.serveur/livres/{id} : mettre à jour le livre identifié par {id}
  - DELETE http://adresse.serveur/books/{id} : supprimer le livre identifié par {id}

L'INCONTOURNABLE SPRING BOOT

Principes de base d'une API REST

- Représentation des ressources
  - Les ressources sont le plus souvent représentées par le format JSON ou XML
- Format JSON (JavaScript Object Notation)
  - Format permettant de représenter de données
  - Léger et facile à lire et écrire pour les humains
  - Format texte
  - Représentation d'objet : { "clé1": "valeur1", "clé2": "valeur2" , ... }
  - Représentation de tableaux de valeurs : ["valeur1", "valeur2", ...]
  - Représentation de tableaux d'objets :
    [{ "clé1": "valeur11", "clé2": "valeur21" }, { "clé1": "valeur12", "clé2": "valeur22" }, ...]

L'INCONTOURNABLE SPRING BOOT

Communications avec la base de données

- Spring s'appuie sur les annotions pour identifier les éléments qui permettent de construire l'API
- @Controller : identifie une classe qui expose une ressource
- @RequestMapping : définit le chemin d'accès à la ressource
- @GetMapping, @PostMapping, ... : définit le verbe HTTP et le chemin d'accès à la ressource
- @PathVariable : définit le paramètre qui va accueillir une partie dynamique du chemin d'accès à la ressource
- @RequestParam : définit un paramètre d'URL
- @ResquestBody : définit le paramètre qui va accueillir le corps de la requête reçue

L'INCONTOURNABLE SPRING BOOT

Communications avec la base de données

- La réponse s'appuie sur les getters de la classe retournée par la méthode

```
public class Livre {
  private String auteur;
  private String titre;
        private String extrait;
        public String getAuteur() {
                return auteur;
        }
        public String getTitre() {
                return titre;
        }
        // pas de getter sur extrait
        // => { "auteur":"XXX", "titre":"YYY" }
}
```

L'INCONTOURNABLE SPRING BOOT

Exposer une API REST avec Spring Boot

```
@Controller
@RequestMapping("/livres")
public class LivreController {
        // GET http://adresse.serveur/livres
        @GetMapping
        public List<Livre> listerLesLivres() {
                ...
        }
        // GET http://adresse.serveur/livres/{id}
        @GetMapping("/{id}")
        public Livre getInformationLivre(@PathVariable String id) {
                ...
        }
}
```

L'INCONTOURNABLE SPRING BOOT

Exposer une API REST avec Spring Boot

```
@Controller
@RequestMapping("/livres")
public class LivreController {
	// GET http://adresse.serveur/livres/{id}/extrait
	@GetMapping("/{id}/extrait")
	public String getExtraitLivre(@PathVariable String id) {
		...
	}
	// POST http://adresse.serveur/livres {"auteur":"Dan Bill", "titre":"Le repère", "extrait":"Lorem ipsum"}
	@PostMapping
	public void ajouterLivre(@RequestBody Livre Livre) {
		...
	}
}
```

L'INCONTOURNABLE SPRING BOOT

Communications avec la base de données

- Spring JPA : facilite la connexion avec les bases de données
- @Entity : faire correspondre une table avec une classe
- @Id : définit l'attribut de la classe qui correspond à l'id
- @GeneratedValue : définit la stratégie d'obtention d'un nouvel id
- Le nom de la classe est mappé sur le nom de la classe
- Le type et le nom des champs sont mappés aux types et noms des attributs de la classe
- L'interface JpaRepository permet d'obtenir avec un minimum de code un moyen d'interagir avec la base de données
    - L'implémentation de l'interface est générée par Spring

L'INCONTOURNABLE SPRING BOOT

Communications avec la base de données

- Attention : tout objet qui utilise un autre objet géré par Spring doit aussi être géré par Spring

```
@Service
public class Bibliothecaire {
        // LivreRepository est un objet géré par Spring, il faut donc que Bibliothecaire le soit aussi avec l'annotation @Service
        // il existe d'autres annotations pour indiquer qu'une classe est gérée par Spring
        private final LivreRepository livreRepository;
        public Bibliothecaire(LivreRepository livreRepository) {
                this.livreRepository = livreRepository;
        }
        public Livre ajouterLivre(Livre livre) {
             return livreRepository.save(livre);
        }
}
```

L'INCONTOURNABLE SPRING BOOT

Communications avec la base de données

```java
@Entity
public class Livre {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String auteur;
    private String titre;
}
```

```sql
CREATE TABLE Livre (
    id BIGINT AUTO_INCREMENT NOT NULL,
    auteur VARCHAR(255),
    titre VARCHAR(255),
    PRIMARY KEY (id)
)
```

L'INCONTOURNABLE SPRING BOOT

Communications avec la base de données

```
@Repository
public interface LivreRepository extends JpaRepository<Livre, Long> {
   // Méthode personnalisée pour trouver les livres par auteur
   List<Livre> findByAuteur(String auteur);
}
// Il n'est pas nécessaire de coder l'implémentation de l'interface LivreRepository
// Spring génère et gère automatiquement l'implémentation
```

FIL ROUGE

L’incontournable Spring Boot

- Branche git 04-01-spring-boot-api
  - base de code avec la configuration Spring boot pour créer une API REST
  - Exemple de controller fourni
  - Jeter un œil au fichier Readme.md
- Branche git 04-02-spring-boot-bdd
  - Base de code avec la configuration Spring Boot pour communiquer avec une BDD en mémoire
  - Jeter un œil au fichier Readme.md

FIL ROUGE

Projet soumis à évaluation

- Imaginez un scénario à réaliser à travers l'application DoctoCrest
- Développer l'API qui permettra de dérouler ce scénario
- Déposer le projet sous Teams avec un fichier décrivant le scénario imaginé et les différents choix d'implémentation
- Critères d'évaluation
  - Lisibilité du code : prêter attention au nommage
  - Modularité : penser à bien répartir les responsabilités dans les différentes classes
  - Structures de données adaptées : choisir une structure de données pertinente pour stocker les données à traiter
  - Persistance : connecter une base de données à l'application

POUR ALLER PLUS LOIN

- Sécurisation d'une API REST
  - https://blog.octo.com/securiser-une-api-rest-tout-ce-quil-faut-savoir
- Les patrons de conceptions (Design pattern)
  - https://refactoring.guru/fr/design-patterns/java
- Quelques bonnes pratiques de base : clean code
  - https://www.ionos.fr/digitalguide/sites-internet/developpement-web/clean-code-avantages-principes-et-exemples/