

INTRODUCTION AU DÉVELOPPEMENT EN JAVA

PAR DIDIER ERIN – APPRENTI
ARTISAN DÉVELOPPEUR



version 202405

DÉROULEMENT DE L'INITIATION

Plan

INTRODUCTION

LES CONCEPTS DE BASE

UN PEU DE MODÉLISATION

LA ROUE EXISTE DÉJÀ

L'INCONTOURNABLE SPRING BOOT

ATTENTION AUX HACKERS

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 bytecode
 - 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 bytecode
 - Runtime Data Areas : gestion des zones de mémoire



INTRODUCTION

Un peu d'histoire

- Initié par James Gosling at 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 Gosling at 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

- 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 \Rightarrow `System.in` ou `Scanner` / `System.out`
 - Système de fichiers \Rightarrow `FileReader` / `FileWriter`
 - Flux de bytes (fichiers binaires, images, ...) \Rightarrow `InputStream` / `OutputStream`
 - Réseau \Rightarrow `Socket` (client réseau) / `ServerSocket` (serveur réseau)
- Améliorer les performances en utilisant la mémoire
 - Mémoire \Rightarrow `ByteArrayInputStream` / `ByteArrayOutputStream`
 - Buffer \Rightarrow `BufferedReader` / `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 appelle 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<>();`



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 superflus



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

- 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



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

```
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

- 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

- 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
 - Le mot clé **this** permet d'accéder aux attributs
 - Facultatif

```
public class MaClasse {  
    String propriete1  
    Integer propriete2  
  
    public MaClasse(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 MaClass 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 visibilitées `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 **concurrency** : 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.println` 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 Formatable<Personne, String> {  
    String formater(Personne personne);  
}  
  
Formatable personneFormatable = (personne) ->  
{  
    return "%s %s".formatted(personne.prenom,  
        personne.nom);  
}  
  
Ou  
  
Formatable personneFormatable = 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
- Exposer une API Rest
- Communications avec la base de données



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
- Exemple de Spring Boot + Spring MVC



ATTENTION AUX HACKERS



L'INCONTOURNABLE SPRING BOOT

- Quelques principes fondamentaux de la sécurité
- Authentification, autorisations, confidentialité
- Sécuriser l'application avec Spring Security

