

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 ⇒ `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 ⇒ `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 ?

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

```
try {  
    traitement pouvant générer une erreur de type <type d'exception>  
} catch (<type d'exception> ex) {  
    comportement à adopter 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