Chapitre 1 : Introduction à Java

1.1 Qu'est-ce que Java ?

Java est un langage de programmation populaire, créé par James Gosling et Mike Sheridan chez Sun Microsystems (aujourd'hui acquis par Oracle Corporation) en 1991. C'est un langage orienté objet, ce qui signifie que tout en Java est un objet. Il est conçu pour être portable, sécurisé et fiable, ce qui en fait un choix idéal pour le développement d'applications variées, des applications de bureau aux applications mobiles et aux applications web.

1.2 Installation de Java

Avant de commencer à programmer en Java, vous devez installer le **JDK (Java Development Kit)**, qui contient les outils nécessaires pour développer des applications Java. Voici les étapes pour l'installer :

1. **Téléchargez le JDK** : Rendez-vous sur le site officiel d'Oracle (https://www.oracle.com/java/technologies/javase-downloads.html) et téléchargez la dernière version du JDK compatible avec votre système d'exploitation.
2. **Installez le JDK** : Suivez les instructions du programme d'installation pour installer le JDK sur votre ordinateur.
3. **Vérifiez l'installation** : Ouvrez une invite de commande (ou un terminal) et exécutez la commande `java -version`. Vous devriez voir la version du JDK installée.

1.3 Votre premier programme Java

Commençons par écrire un programme Java simple qui affiche "Hello, World!" à l'écran. Suivez ces étapes :

1. **Ouvrez un éditeur de texte** : Utilisez un éditeur de texte tel que Notepad (sur Windows), TextEdit (sur macOS) ou gedit (sur Linux).
2. **Écrivez le code** :

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
```

```
}
```

3. **Sauvegardez le fichier** : Enregistrez le fichier avec l'extension `.java`, par exemple `HelloWorld.java`.
4. **Compilez le programme** : Ouvrez une invite de commande, accédez au répertoire où vous avez enregistré le fichier `HelloWorld.java` et exécutez la commande `javac HelloWorld.java`. Cela créera un fichier `HelloWorld.class` qui contient le bytecode exécutable.
5. **Exécutez le programme** : Tapez `java HelloWorld` dans l'invite de commande pour exécuter votre programme. Vous devriez voir "Hello, World!" affiché à l'écran.

Félicitations ! Vous avez écrit et exécuté votre premier programme en Java.

1.4 Concepts clés en Java

- **Classes et objets** : Java est un langage orienté objet, ce qui signifie que tout est basé sur des classes et des objets.
- **Méthodes** : Les méthodes sont des blocs de code qui effectuent des actions spécifiques. La méthode `main` est le point d'entrée de toute application Java.
- **Variables** : Les variables sont utilisées pour stocker des données temporaires dans un programme.
- **Types de données** : Java a différents types de données tels que int, double, String, etc., qui définissent le type et la taille des données qu'une variable peut contenir.
- **Contrôle de flux** : Les structures de contrôle, telles que les boucles (for, while) et les instructions conditionnelles (if, else), sont utilisées pour contrôler le flux d'exécution du programme.

Chapitre 2 : Concepts Fondamentaux de Java

Dans ce chapitre, nous approfondirons nos connaissances sur les concepts fondamentaux de Java tels que les variables, les types de données, les opérateurs, les structures de contrôle et les boucles.

2.1 Variables et Types de Données

2.1.1 Déclaration de Variables

En Java, vous devez déclarer une variable avant de l'utiliser. La déclaration consiste à spécifier le type de données et le nom de la variable.

Exemple :

```
int age; // Déclaration d'une variable de type entier nommée "age"
```

2.1.2 Types de Données

Java a plusieurs types de données, tels que :

- **int** : pour les nombres entiers.
- **double** : pour les nombres à virgule flottante.
- **boolean** : pour les valeurs booléennes (true ou false).
- **char** : pour les caractères.
- **String** : pour les chaînes de caractères, etc.

Exemples :

```
int score = 100;        // Un entier
double pi = 3.14;       // Un nombre à virgule flottante
boolean estVrai = true;  // Une valeur booléenne
char grade = 'A';       // Un caractère
String nom = "Alice";    // Une chaîne de caractères
```

2.2 Opérateurs

Les opérateurs sont des symboles spéciaux utilisés pour effectuer des opérations sur les variables et les valeurs. Il existe plusieurs types d'opérateurs en Java, tels que les opérateurs arithmétiques (+, -, *, /), les opérateurs de comparaison (==, !=, <, >), et les opérateurs logiques (&&, ||, !).

Exemple :

```
int a = 10;
int b = 5;
int somme = a + b;     // Addition
int produit = a * b;   // Multiplication
boolean estPlusGrand = a > b;  // Opérateur de comparaison
boolean etLogique = (a > 0) && (b > 0);  // Opérateur logique
```

2.3 Structures de Contrôle

Les structures de contrôle permettent de contrôler le flux d'exécution du programme en utilisant des conditions et des boucles.

2.3.1 Instructions Conditionnelles (if, else)

Les instructions conditionnelles permettent d'exécuter des blocs de code en fonction d'une condition.

Exemple :

```
int age = 18;


if (age >= 18) {
    System.out.println("Vous êtes majeur.");
} else {
    System.out.println("Vous êtes mineur.");
}
```

2.3.2 Boucles (for, while)

Les boucles permettent de répéter un bloc de code plusieurs fois.

Boucle "for" :

```
for (int i = 1; i <= 5; i++) {
    System.out.println("Répétition " + i);
}
int i = 1;
while (i <= 5) {
    System.out.println("Répétition " + i);
    i++;
}
```

2.4 Méthodes

Les méthodes sont des blocs de code réutilisables. Elles peuvent accepter des paramètres et renvoyer une valeur.

Exemple :

```
public int additionner(int a, int b) {
    return a + b;
}
```

Chapitre 3 : Tableaux, Chaînes de Caractères et Fonctions Avancées

Dans ce chapitre, nous explorerons les tableaux, les chaînes de caractères, les fonctions avancées et la modularité du code en Java.

3.1 Tableaux

Un tableau en Java est une structure de données permettant de stocker une collection d'éléments du même type sous un même nom.

3.1.1 Déclaration et Initialisation

Exemple de déclaration et d'initialisation d'un tableau :

```
int[] tableauEntiers = {10, 20, 30, 40, 50};
String[] tableauMots = new String[3]; // Crée un tableau de 3 chaînes de caractères
tableauMots[0] = "Java";
tableauMots[1] = "est";
tableauMots[2] = "génial";
```

3.1.2 Accès aux Éléments

Vous pouvez accéder aux éléments d'un tableau en utilisant l'indice de l'élément (qui commence à 0 pour le premier élément).

Exemple :

```
int premierElement = tableauEntiers[0]; // Accès au premier élément du tableau
String mot = tableauMots[1]; // Accès au deuxième élément du tableau
```

3.2 Chaînes de Caractères (String)

Les chaînes de caractères en Java sont des séquences de caractères utilisées pour représenter du texte.

3.2.1 Déclaration et Utilisation

Exemple de déclaration et d'utilisation d'une chaîne de caractères :

```
String message = "Bonjour, Java!";

System.out.println(message);
```

3.2.2 Manipulation de Chaînes de Caractères

Java offre de nombreuses méthodes pour manipuler les chaînes de caractères, telles que `length()`, `charAt()`, `substring()`, `toUpperCase()`, `toLowerCase()`, `indexOf()`, etc.

Exemple :

```
String str = "Exemple de chaîne";

int longueur = str.length(); // Obtient la longueur de la chaîne

char premierCaractere = str.charAt(0); // Obtient le premier caractère

String sousChaine = str.substring(8, 12); // Obtient la sous-chaîne à partir de l'indice 8 à 12 (non inclus)
```

3.3 Fonctions Avancées

3.3.1 Fonctions Récursives

Une fonction récursive est une fonction qui s'appelle elle-même pour résoudre un problème plus petit.

Exemple d'une fonction factorielle récursive :

```
public int factorielle(int n) {
    if (n == 0 || n == 1)
        return 1;
    else
        return n * factorielle(n - 1);
}
```

3.3.2 Fonctions Variadiques (varargs)

Les fonctions variadiques permettent de passer un nombre variable d'arguments à une fonction.

Exemple :

```
public int addition(int... nombres) {
    int somme = 0;
    for (int nombre : nombres) {
        somme += nombre;
    }
    return somme;
}
```

3.4 Modularité et Réutilisabilité

Il est important d'écrire du code modulaire et réutilisable en Java. Vous pouvez répartir votre code en différentes classes et packages pour le rendre plus organisé et facile à gérer.

Exemple de création d'une classe et d'appel de ses méthodes :

```
// Définition d'une classe
class Calculatrice {
    public int addition(int a, int b) {
        return a + b;
    }
}


// Utilisation de la classe
public class Main {
    public static void main(String[] args) {
        Calculatrice calc = new Calculatrice();
        int resultat = calc.addition(10, 20);
        System.out.println("Résultat de l'addition : " + resultat);
```

```
    }
}
```

Chapitre 4 : Classes, Interfaces et Gestion des Exceptions

Dans ce chapitre, nous explorerons les concepts avancés en programmation Java tels que les classes, les interfaces et la gestion des exceptions.

4.1 Classes et Objets

4.1.1 Définition d'une Classe

En Java, une classe est un modèle permettant de créer des objets. Elle définit les propriétés et les méthodes que les objets de cette classe auront.

Exemple de définition d'une classe `Personne` :

```
public class Personne {
    String nom;
    int age;

    public void afficherDetails() {
        System.out.println("Nom : " + nom);
        System.out.println("Âge : " + age);
    }
}
```

4.1.2 Création d'Objets

Pour utiliser une classe, vous devez créer des objets de cette classe à l'aide de l'opérateur `new`.

Exemple de création d'un objet de la classe `Personne` :

```
Personne personne1 = new Personne();
personne1.nom = "Alice";
```

```
personne1.age = 30;
personne1.afficherDetails();  // Appel d'une méthode de l'objet
```

4.2 Interfaces

Une interface en Java est une collection de méthodes abstraites et de constantes. Elle définit un contrat que les classes doivent suivre.

Exemple de définition d'une interface `Forme` :

```
public interface Forme {
    double calculerAire();
    double calculerPerimetre();
}
```

Les classes qui implémentent cette interface doivent fournir une implémentation pour les méthodes `calculerAire()` et `calculerPerimetre()`.

4.3 Héritage

L'héritage est un mécanisme qui permet à une classe d'hériter des propriétés et des méthodes d'une autre classe. Cela favorise la réutilisabilité et la structuration du code.

Exemple de définition d'une classe héritée `Etudiant` :

```
public class Etudiant extends Personne {
    int numeroEtudiant;

    public void afficherNumeroEtudiant() {
        System.out.println("Numéro étudiant : " + numeroEtudiant);
    }
}
```

Dans cet exemple, la classe `Etudiant` hérite des propriétés et méthodes de la classe `Personne`.

4.4 Gestion des Exceptions

Les exceptions en Java permettent de gérer les erreurs et les situations exceptionnelles. Une exception peut être levée ou attrapée.

Exemple de levée d'une exception :

```
public void diviser(int a, int b) {
    if (b == 0) {
        throw new ArithmeticException("Division par zéro");
    }
    int resultat = a / b;
    System.out.println("Résultat : " + resultat);
}
```

Exemple de gestion d'une exception :

```
try {
    diviser(10, 0);
} catch (ArithmeticException e) {
    System.out.println("Une exception s'est produite : " + e.getMessage());
}
```

4.5 Modificateurs d'Accès

Les modificateurs d'accès en Java (public, private, protected, default) permettent de contrôler l'accès aux classes, méthodes et variables.

- **public** : accessible partout.
- **private** : accessible uniquement dans la classe.
- **protected** : accessible dans le même package et dans les classes dérivées.
- **default** : accessible uniquement dans le même package.

4.6 Final et Abstract

- **final** : utilisé pour définir des classes, des méthodes ou des variables constantes et immuables.
- **abstract** : utilisé pour définir des classes ou des méthodes abstraites qui doivent être implémentées par les classes dérivées.

Chapitre 5 : Collections, Fichiers et Threads en Java

Dans ce chapitre, nous explorerons les collections, la manipulation de fichiers et la gestion des threads en Java.

5.1 Collections

Les collections en Java permettent de stocker, manipuler et transmettre des groupes d'objets. Il existe plusieurs types de collections, dont les listes, les ensembles, les cartes, etc.

5.1.1 Listes

Les listes permettent de stocker une séquence d'éléments.

Exemple d'utilisation d'une liste (ArrayList) :

```
import java.util.ArrayList;
import java.util.List;

List<String> liste = new ArrayList<>();
liste.add("Java");
liste.add("Python");
liste.add("C++");

for (String langage : liste) {
    System.out.println(langage);
}
```

5.1.2 Ensembles

Les ensembles ne permettent pas d'avoir des éléments en double.

Exemple d'utilisation d'un ensemble (HashSet) :

```
import java.util.HashSet;
import java.util.Set;
```

```
Set<Integer> ensemble = new HashSet<>();
ensemble.add(10);
ensemble.add(20);
ensemble.add(10);  // Ne sera pas ajouté car 10 est déjà présent

for (int nombre : ensemble) {
    System.out.println(nombre);
}
```

5.1.3 Cartes (Map)

Les cartes permettent de stocker des paires clé-valeur.

Exemple d'utilisation d'une carte (HashMap) :

```
import java.util.HashMap;
import java.util.Map;

Map<String, Integer> carte = new HashMap<>();
carte.put("Alice", 30);
carte.put("Bob", 35);
carte.put("Charlie", 25);

int ageBob = carte.get("Bob");  // Obtient la valeur associée à la clé "Bob"
System.out.println("Âge de Bob : " + ageBob);
```

5.2 Lecture et Écriture de Fichiers

Java propose des classes pour lire et écrire des fichiers texte ou binaires.

5.2.1 Lecture de Fichiers

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

```
try (BufferedReader reader = new BufferedReader(new FileReader("fichier.txt"))) {
    String ligne;
    while ((ligne = reader.readLine()) != null) {
        System.out.println(ligne);
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

5.2.2 Écriture dans un Fichier

java

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

try (BufferedWriter writer = new BufferedWriter(new FileWriter("fichier.txt"))) {
    writer.write("Écriture dans un fichier.");
} catch (IOException e) {
    e.printStackTrace();
}
```

5.3 Threads

Les threads en Java permettent d'exécuter plusieurs tâches simultanément.

Exemple de création et d'exécution d'un thread :

```
class MonThread extends Thread {
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println("Thread " + i);
            try {
                Thread.sleep(1000); // Pause d'une seconde
            } catch (InterruptedException e) {
                e.printStackTrace();
```

```
        }
      }
    }
}

public class Main {
    public static void main(String[] args) {
        MonThread thread = new MonThread();
        thread.start();  // Démarrage du thread
    }
}
```

5.4 Synchronisation de Threads

La synchronisation en Java permet de contrôler l'accès concurrentiel aux ressources partagées entre les threads.

Exemple d'utilisation de la synchronisation :

```
class Compteur {
    private int valeur = 0;

    public synchronized void incrementer() {
        valeur++;
    }

    public synchronized int getValeur() {
        return valeur;
    }
}
```

Utilisation de "this" dans le contexte des Classes

Dans Java, "this" est un mot-clé qui fait référence à l'instance actuelle de la classe dans laquelle il est utilisé. Voici quelques utilisations courantes de "this" dans le contexte des classes :

1. Accès aux Variables d'Instance

"this" peut être utilisé pour faire référence aux variables d'instance de la classe. Cela est souvent utilisé lorsque les noms des paramètres de méthode ont le même nom que les variables d'instance.

Exemple :

```
public class Personne {
    private String nom;

    public void setNom(String nom) {
        this.nom = nom;  // Utilisation de "this" pour distinguer entre le paramètre et la variable d'instance
    }
}
```

2. Appel de Méthodes de la Classe Actuelle

"this" peut être utilisé pour appeler d'autres méthodes de la même classe à l'intérieur d'une méthode.

Exemple :

```
public class Exemple {
    public void methode1() {
        // ...
    }

    public void methode2() {
        this.methode1();  // Appel de methode1() de cette classe
    }
}
```

3. Retour de l'Instance Actuelle

"this" peut être utilisé pour retourner l'instance actuelle de la classe à partir d'une méthode, permettant ainsi des appels chaînés (appelés "fluent interface").

Exemple :

```
public class Calculatrice {
    private int resultat;

    public Calculatrice additionner(int a) {
        this.resultat += a;
        return this;  // Retourne l'instance actuelle
    }

    public int getResultat() {
        return resultat;
    }
}

// Utilisation de la classe
Calculatrice calc = new Calculatrice();
int total = calc.additionner(10).additionner(5).getResultat();
System.out.println("Total : " + total);  // Affiche : Total : 15
```