Logo

Formation

introduction java

Logo

Présenté par

hakim toutay

Programme de la Formation

Jour 1:

- Introduction à Java
- Structure d'un programme Java
- Introduction à Java

Jour 2:

- Opérateurs et expressions
- Structure de contrôle en Java

Jour 3:

- Tableau en Java
- Méthodes en Java

Jour 4:

- Programmation orientée objet
- Constructeur et Destructeur en Java

Jour 5:

- Héritage et polymorphisme
- Gestion des exceptions

Jour 1

Introduction à Java

Présentation de Java et son histoire

Java est un langage de programmation orienté objet développé par Sun Microsystems en 1995. Il a été conçu pour être portable, ce qui signifie que les programmes écrits en Java peuvent fonctionner sur n'importe quelle plateforme disposant d'une machine virtuelle Java (JVM). Java a été initialement conçu pour les appareils embarqués, mais il est rapidement devenu populaire pour le développement d'applications web et d'entreprise.

- `Créé par James Gosling et son équipe chez Sun Microsystems`
- `Première version publique en 1995`
- `Acquis par Oracle Corporation en 2010`

Caractéristiques et avantages de Java

Java est connu pour ses nombreuses caractéristiques qui en font un choix populaire pour les développeurs. Voici quelques-unes des principales caractéristiques et avantages de Java :

- `Syntaxe simple et facile à apprendre`
- `Grande communauté de développeurs`
- `Bibliothèque standard riche et complète`

Caractéristique	Description
Portabilité	Les programmes Java peuvent fonctionner sur n'importe quelle plateforme grâce à la JVM.
Sécurité	Java dispose de fonctionnalités de sécurité intégrées pour protéger contre les menaces.
Performance	La compilation Just-In-Time (JIT) améliore les performances des applications Java.

Gestion automatique de la mémoire	Le ramasse-miettes (Garbage Collector) gère automatiquement la mémoire, réduisant les fuites de mémoire.

Installation de l'environnement de développement (JDK, IDE)

Pour commencer à développer en Java, vous devez installer le Java Development Kit (JDK) et un environnement de développement intégré (IDE). Le JDK contient les outils nécessaires pour compiler et exécuter des programmes Java, tandis qu'un IDE facilite le développement en fournissant des fonctionnalités telles que la complétion de code et le débogage.

- `Téléchargez le JDK depuis le site officiel d'Oracle ou OpenJDK.`
- `Installez le JDK en suivant les instructions pour votre système d'exploitation.`
- `Configurez les variables d'environnement pour inclure le chemin du JDK.`
- `Choisissez un IDE comme IntelliJ IDEA, Eclipse ou NetBeans pour le développement.`

```
Exemple de configuration de variable d'environnement sur Windows :
1. Ouvrez les propriétés du système.
2. Cliquez sur 'Variables d'environnement'.
3. Ajoutez le chemin du JDK à la variable 'Path'.
```

Structure d'un programme Java

Syntaxe de base

La syntaxe de Java est basée sur des règles strictes qui définissent comment le code doit être écrit. Java est sensible à la casse, ce qui signifie que les mots-clés, les identifiants et les noms de classes doivent être écrits exactement comme définis. Les instructions Java se terminent par un point-virgule (;). Les blocs de code sont délimités par des accolades {}.

Exemple: // Exemple de syntaxe de base public class Main { public static void main(String[] args) { System.out.println("Hello, World!"); } }

- Java est sensible à la casse.
- Les instructions se terminent par un point-virgule.
- Les blocs de code sont délimités par des accolades.

Structure d'une classe

Une classe en Java est un modèle ou un plan pour créer des objets. Une classe contient des variables (attributs) et des méthodes (fonctions) qui définissent le comportement des objets. La structure de base d'une classe commence par le mot-clé 'class' suivi du nom de la classe. Le corps de la classe est ensuite défini entre des accolades {}.

Exemple: // Exemple de structure d'une classe public class Car { // Attributs String brand; int year; // Méthode void start() { System.out.println("The car is starting."); } }

- Une classe est un modèle pour créer des objets.
- Contient des variables et des méthodes.
- Définie par le mot-clé 'class'.

Méthode principale (méthode main)

La méthode principale, ou méthode 'main', est le point d'entrée de tout programme Java. Elle est définie comme 'public static void main(String[] args)'. Cette méthode est appelée par la JVM (Java Virtual Machine) pour exécuter le programme. La méthode 'main' doit être incluse dans une classe et est généralement utilisée pour démarrer l'exécution du programme.

Exemple: // Exemple de méthode main public class Main { public static void main(String[] args) { System.out.println("This is the main method."); } }

- Point d'entrée du programme Java.
- Définie comme 'public static void main(String[] args)'.
- Appelée par la JVM pour exécuter le programme.

Jour 2

Opérateurs et expressions

Opérateurs arithmétiques

Les opérateurs arithmétiques sont utilisés pour effectuer des opérations mathématiques de base sur des valeurs numériques. Ils incluent l'addition, la soustraction, la multiplication, la division, et le modulo.

```
int a = 10;
int b = 3;
int somme = a + b; // somme = 13
int difference = a - b; // difference = 7
int produit = a * b; // produit = 30
int quotient = a / b; // quotient = 3
int reste = a % b; // reste = 1
```

Opérateur	Description	Exemple
+	Addition	a + b

-	Soustraction	a - b
*	Multiplication	a * b
/	Division	a / b
%	Modulo (reste de la division)	a % b

Opérateurs relationnels

Les opérateurs relationnels sont utilisés pour comparer deux valeurs. Ils retournent un booléen (true ou false) en fonction de la relation entre les deux valeurs.

```
int a = 10;
int b = 20;
boolean estEgal = (a == b); // estEgal = false
boolean estDifferent = (a != b); // estDifferent = true
boolean estSuperieur = (a > b); // estSuperieur = false
boolean estInferieur = (a < b); // estInferieur = true
```

Opérateur	Description	Exemple
==	Égal à	a == b

!=	Différent de	a != b
>	Supérieur à	a > b
<	Inférieur à	a < b
>=	Supérieur ou égal à	a >= b
<=	Inférieur ou égal à	a <= b

Opérateurs logiques

Les opérateurs logiques sont utilisés pour combiner ou inverser des expressions booléennes. Ils incluent ET logique, OU logique, et NON logique.

- `L'opérateur ET logique (&&) retourne true uniquement si les deux opérandes sont true.`
- `L'opérateur OU logique (||) retourne true si au moins un des opérandes est true.`
- `L'opérateur NON logique (!) inverse la valeur booléenne de l'opérande.`

```
        boolean a = true;
        boolean b = false;
        boolean etLogique = a && b; // etLogique = false
        boolean ouLogique = a || b; // ouLogique = true
        boolean nonLogique = !a; // nonLogique = false
```

Opérateur	Description	Exemple
&&	ET logique	a && b
||	OU logique	a || b
!	NON logique	!a

Structure de contrôle en Java

Instruction conditionnelle if-else

L'instruction if-else permet de prendre des décisions en fonction de la véracité d'une condition. Si la condition est vraie, le bloc de code associé à 'if' est exécuté. Sinon, le bloc de code associé à 'else' est exécuté.

- `Utilisation de conditions booléennes`
- `Possibilité de chaîner plusieurs conditions avec 'else if'`

Instruction switch-case

L'instruction switch-case permet de sélectionner un bloc de code à exécuter parmi plusieurs options en fonction de la valeur d'une variable. Elle est souvent utilisée pour remplacer une série de conditions if-else.

- `Utilisation de la valeur d'une variable pour le choix`
- `Possibilité d'utiliser 'default' pour un cas par défaut`

Boucles (for, while, do-while)

Les boucles permettent de répéter un bloc de code plusieurs fois. Java propose trois types de boucles : for, while et do-while.

- La boucle 'for' est utilisée pour un nombre d'itérations connu
- La boucle 'while' répète un bloc de code tant qu'une condition est vraie
- La boucle 'do-while' exécute le bloc de code au moins une fois, puis répète tant que la condition est vraie

Jour 3

Tableau en Java

Déclaration et initialisation de tableaux

En Java, un tableau est une structure de données qui permet de stocker plusieurs éléments de même type. La déclaration d'un tableau se fait en spécifiant le type des éléments et en utilisant des crochets []. L'initialisation peut se faire soit lors de la déclaration, soit après, en utilisant l'opérateur new.

- `Syntaxe de déclaration : `type[] nomTableau;``
- `Syntaxe d'initialisation : `nomTableau = new type[taille];``
- `Initialisation directe : `type[] nomTableau = {valeur1, valeur2, ...};``

Accès aux éléments d'un tableau

Les éléments d'un tableau sont accessibles via leur indice, qui commence à 0. L'accès se fait en utilisant la syntaxe `nomTableau[indice]`. Il est important de vérifier que l'indice est valide pour éviter une exception ArrayIndexOutOfBoundsException.

- `Accès à un élément : `nomTableau[indice]``
- `Modification d'un élément : `nomTableau[indice] = nouvelleValeur;``

Manipulation de tableaux (tri, recherche)

Java fournit des méthodes pour manipuler les tableaux, comme le tri et la recherche. La classe `Arrays` offre des méthodes statiques pour trier un tableau avec `Arrays.sort()` et pour rechercher un élément avec `Arrays.binarySearch()`. Le tri est effectué en ordre croissant par défaut.

- `Tri d'un tableau : `Arrays.sort(nomTableau);``
- `Recherche binaire : `Arrays.binarySearch(nomTableau, valeur);``
- `La recherche binaire nécessite un tableau trié.`

Méthode	Description
Arrays.sort()	Trie un tableau en ordre croissant
Arrays.binarySearch()	Recherche un élément dans un tableau trié

Méthodes en Java

Définition et appel de méthode

Une méthode en Java est un bloc de code qui effectue une tâche spécifique. Elle est définie avec un nom, des paramètres (optionnels) et un type de retour (ou void si aucun retour n'est nécessaire). Une méthode est appelée en utilisant son nom suivi de parenthèses contenant les arguments (si nécessaire).

Exemple: public class Calculatrice { public int addition(int a, int b) { return a + b; } } // Appel de la méthode Calculatrice calc = new Calculatrice(); int resultat = calc.addition(5, 3); // resultat = 8

- Une méthode doit être définie à l'intérieur d'une classe.
- Le nom de la méthode doit suivre les conventions de nommage Java.
- Les paramètres sont des variables locales à la méthode.

Paramètres et valeur de retour

Les paramètres sont des valeurs passées à une méthode pour qu'elle puisse effectuer une tâche. Une méthode peut retourner une valeur en utilisant le mot-clé 'return'. Le type de retour doit correspondre au type déclaré dans la signature de la méthode. Si la méthode ne retourne rien, le type de retour est 'void'.

Exemple: public class Calculatrice { public double division(double a, double b) { if (b == 0) { throw new IllegalArgumentException("Division par zéro"); } return a / b; } } // Appel de la méthode Calculatrice calc = new Calculatrice(); double resultat = calc.division(10.0, 2.0); // resultat = 5.0

- Les paramètres sont passés par valeur en Java.
- Une méthode peut avoir plusieurs paramètres.
- Le type de retour peut être un type primitif, un objet ou void.

Portée des variables

La portée d'une variable définit où elle peut être utilisée dans le code. En Java, les variables ont une portée locale ou globale. Une variable locale est définie à l'intérieur d'une méthode ou d'un bloc et ne peut être utilisée que dans ce bloc. Une variable globale (ou de classe) est définie à l'extérieur de toutes les méthodes et peut être utilisée dans toute la classe.

Exemple: public class ExemplePortee { int variableGlobale = 10; // Variable de classe public void methode() { int variableLocale = 20; // Variable locale System.out.println("Variable globale: " + variableGlobale); System.out.println("Variable locale: " + variableLocale); } } // Appel de la méthode ExemplePortee exemple = new ExemplePortee(); exemple.methode(); // Affiche: Variable globale: 10, Variable locale: 20

- Les variables locales sont détruites après l'exécution du bloc où elles sont définies.
- Les variables de classe sont accessibles à toutes les méthodes de la classe.
- Les variables de classe peuvent être statiques ou d'instance.

Jour 4

Programmation orientée objet

Introduction à la POO

La Programmation Orientée Objet (POO) est un paradigme de programmation qui organise le code autour des objets plutôt que des actions et des données plutôt que de la logique. Elle repose sur quatre principes fondamentaux : l'encapsulation, l'héritage, le polymorphisme et l'abstraction.

- `Encapsulation : Cacher les détails internes d'un objet et exposer uniquement ce qui est nécessaire.`
- `Héritage : Permet à une classe d'hériter des propriétés et méthodes d'une autre classe.`
- `Polymorphisme : Capacité d'un objet à prendre plusieurs formes.`

- Abstraction : Simplifier la complexité en ne montrant que les détails essentiels.

Classe et objet

Une classe est un modèle ou un plan pour créer des objets. Elle définit les propriétés (attributs) et les comportements (méthodes) que les objets de cette classe auront. Un objet est une instance d'une classe.

- Classe : Modèle ou plan pour créer des objets.
- Objet : Instance d'une classe.

```
// Exemple de classe en Java
public class Voiture {
    // Attributs
    String marque;
    String couleur;
    int vitesse;

    // Méthodes
    void accelerer() {
        vitesse += 10;
    }

    void freiner() {
        vitesse -= 10;
    }
}

// Création d'un objet
Voiture maVoiture = new Voiture();
maVoiture.marque = "Toyota";
maVoiture.couleur = "Rouge";
maVoiture.accelerer();
```

Encapsulation

L'encapsulation est un mécanisme qui consiste à cacher les détails internes d'un objet et à exposer uniquement ce qui est nécessaire. Cela se fait généralement en utilisant des modificateurs d'accès comme `private`, `protected`, et `public`.

- Private : Accessible uniquement à l'intérieur de la classe.
- Protected : Accessible à l'intérieur de la classe et de ses sous-classes.
- Public : Accessible de partout.

```
// Exemple d'encapsulation en Java
public class CompteBancaire {
    // Attribut privé
    private double solde;

    // Méthode publique pour accéder au solde
    public double getSolde() {
        return solde;
    }

    // Méthode publique pour modifier le solde
    public void deposer(double montant) {
        if (montant > 0) {
            solde += montant;
        }
    }
}

// Utilisation de la classe
CompteBancaire monCompte = new CompteBancaire();
monCompte.deposer(1000);
System.out.println("Solde: " + monCompte.getSolde());
```

Constructeur et Destructeur en Java

Définition et Utilisation des Constructeurs

Un constructeur en Java est une méthode spéciale utilisée pour initialiser un objet. Il est appelé automatiquement lors de la création d'une instance de classe. Le constructeur a le même nom que la classe et ne retourne aucune valeur, pas même void.

Exemple: public class Voiture { String marque; // Constructeur public Voiture(String m) { marque = m; } } // Utilisation Voiture maVoiture = new Voiture("Toyota");

- Le constructeur est appelé lors de l'instanciation de l'objet.
- Il peut être utilisé pour initialiser les variables d'instance.
- Si aucun constructeur n'est défini, Java fournit un constructeur par défaut.

Surcharge de Constructeurs

La surcharge de constructeurs permet de définir plusieurs constructeurs dans une classe, chacun ayant une signature différente. Cela permet de créer des objets de différentes manières.

Exemple: public class Voiture { String marque; int annee; // Constructeur 1 public Voiture(String m) { marque = m; } // Constructeur 2 public Voiture(String m, int a) { marque = m; annee = a; } } // Utilisation Voiture voiture1 = new Voiture("Toyota"); Voiture voiture2 = new Voiture("Honda", 2020);

- `Chaque constructeur doit avoir une liste de paramètres unique.`
- `La surcharge permet une plus grande flexibilité dans l'initialisation des objets.`

Gestion de la Mémoire

En Java, la gestion de la mémoire est principalement gérée par le Garbage Collector (GC). Le GC est un processus qui libère automatiquement la mémoire occupée par les objets qui ne sont plus référencés. Java ne fournit pas de destructeur explicite comme en C++.

Exemple: public class Test { @Override protected void finalize() throws Throwable { System.out.println("L'objet est en cours de destruction"); super.finalize(); } } // Utilisation Test t = new Test(); t = null; // L'objet est éligible pour le Garbage Collector

- Le Garbage Collector fonctionne en arrière-plan pour libérer la mémoire.
- Les objets non référencés sont éligibles pour la collecte de déchets.
- La méthode `finalize()` peut être utilisée pour exécuter des actions avant que l'objet ne soit collecté, mais elle est rarement utilisée.

Jour 5

Héritage et polymorphisme

Principe de l'héritage

L'héritage est un mécanisme fondamental en Java qui permet à une classe (appelée classe enfant ou sous-classe) d'hériter des attributs et méthodes d'une autre classe (appelée classe parent ou superclasse). Cela favorise la réutilisation du code et la hiérarchie entre les classes.

Exemple: class Animal { void faireDuBruit() { System.out.println("L'animal fait un bruit."); } } class Chien extends Animal { void faireDuBruit() { System.out.println("Le chien aboie."); } }

- `La classe enfant hérite de tous les attributs et méthodes de la classe parent.`
- `La classe enfant peut ajouter de nouveaux attributs et méthodes ou modifier ceux hérités.`

- L'héritage est représenté par le mot-clé 'extends' en Java.

Surcharge et redéfinition de méthode

La surcharge (overloading) permet de définir plusieurs méthodes avec le même nom mais avec des paramètres différents dans la même classe. La redéfinition (overriding) permet à une sous-classe de fournir une implémentation spécifique d'une méthode déjà définie dans sa superclasse.

Exemple: class Calculatrice { int addition(int a, int b) { return a + b; } double addition(double a, double b) { return a + b; } } class CalculatriceAvancee extends Calculatrice { @Override int addition(int a, int b) { return a + b + 10; } }

- La surcharge est basée sur le nombre ou le type des paramètres.
- La redéfinition nécessite que la méthode ait la même signature (nom et paramètres) que celle de la superclasse.
- La redéfinition utilise l'annotation '@Override' pour indiquer explicitement l'intention.

Classes abstraites et interfaces

Une classe abstraite est une classe qui ne peut pas être instanciée et peut contenir des méthodes abstraites (sans implémentation) et des méthodes concrètes. Une interface est une collection de méthodes abstraites qui définissent un contrat que les classes implémentant l'interface doivent respecter.

Exemple: abstract class Animal { abstract void faireDuBruit(); } interface Voler { void voler(); } class Oiseau extends Animal implements Voler { void faireDuBruit() { System.out.println("L'oiseau chante."); } public void voler() { System.out.println("L'oiseau vole."); } }

- `Une classe abstraite est déclarée avec le mot-clé 'abstract'.`
- `Une interface est déclarée avec le mot-clé 'interface'.`
- `Une classe peut implémenter plusieurs interfaces mais ne peut hériter que d'une seule classe.`

Classe Abstraite	Interface

Peut contenir des méthodes concrètes et abstraites	Ne contient que des méthodes abstraites (avant Java 8)
Utilisée pour partager du code entre classes	Utilisée pour définir un contrat
Une classe peut hériter d'une seule classe abstraite	Une classe peut implémenter plusieurs interfaces

Gestion des exceptions

Type d'exceptions

En Java, les exceptions sont des événements qui se produisent pendant l'exécution d'un programme et qui perturbent le flux normal des instructions. Il existe deux types d'exceptions : les exceptions vérifiées (checked) et les exceptions non vérifiées (unchecked). Les exceptions vérifiées sont vérifiées au moment de la compilation, tandis que les exceptions non vérifiées sont vérifiées au moment de l'exécution.

- `Exceptions vérifiées : IOException, SQLException`
- `Exceptions non vérifiées : NullPointerException, ArrayIndexOutOfBoundsException`

Type d'exception	Exemples
Vérifiée	IOException, SQLException
Non vérifiée	NullPointerException, ArrayIndexOutOfBoundsException

Bloc try-catch-finally

Le bloc try-catch-finally est utilisé pour gérer les exceptions en Java. Le bloc try contient le code susceptible de générer une exception. Le bloc catch est utilisé pour capturer et gérer l'exception. Le bloc finally est exécuté après le bloc try et le bloc catch, qu'une exception ait été levée ou non. Il est généralement utilisé pour libérer des ressources.

- `try : Contient le code susceptible de générer une exception`
- `catch : Capture et gère l'exception`
- `finally : Exécuté après try et catch, qu'une exception ait été levée ou non`

```
        try {
            int[] numbers = {1, 2, 3};
            System.out.println(numbers[3]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Index hors limites");
        } finally {
            System.out.println("Bloc finally exécuté");
        }
```

Propagation des exceptions

La propagation des exceptions se produit lorsqu'une exception est levée dans une méthode et n'est pas capturée dans cette méthode. L'exception est alors propagée à la méthode appelante. Si l'exception n'est pas capturée dans la méthode appelante, elle continue à se propager jusqu'à ce qu'elle soit capturée ou jusqu'à ce qu'elle atteigne la méthode main, ce qui entraîne l'arrêt du programme.

- `Une exception est propagée si elle n'est pas capturée dans la méthode où elle est levée`
- `L'exception est propagée à la méthode appelante`
- `Si l'exception n'est pas capturée, elle peut entraîner l'arrêt du programme`

```
        public void method1() {
            method2();
        }

        public void method2() {
            throw new NullPointerException("Exception levée dans method2");
        }

        public static void main(String[] args) {
            try {
                new Main().method1();
            } catch (NullPointerException e) {
                System.out.println("Exception capturée dans main");
            }
        }
```