Logo

Formation

introduction java

Logo

Présenté par

hakim toutay

Programme de la Formation

Jour 1:

- Introduction à Java
- Premiers pas en Java

Jour 2:

- Les bases de la syntaxe Java
- Les boucles en Java

Jour 3:

- Les tableaux en Java
- Introduction à la programmation orientée objet (POO)

Jour 4:

- Approfondissement de la POO

- Introduction aux exceptions

Jour 1

Introduction à Java

Présentation de Java et son histoire

Java est un langage de programmation orienté objet, créé par Sun Microsystems en 1995. Il a été conçu pour être portable, c'est-à-dire qu'un programme écrit en Java peut fonctionner sur n'importe quelle plateforme supportant la machine virtuelle Java (JVM). Java est largement utilisé dans le développement d'applications web, mobiles (Android), et d'entreprise.

- Créé par James Gosling et son équipe chez Sun Microsystems.
- Initialement appelé 'Oak', puis renommé 'Java' en 1995.
- Première version publique : Java 1.0 en 1996.
- Oracle a acquis Sun Microsystems en 2010, devenant ainsi le propriétaire de Java.

Caractéristiques et avantages de Java

Java est apprécié pour sa simplicité, sa portabilité, et sa robustesse. Voici quelques-unes de ses caractéristiques principales :

- `Syntaxe similaire à C/C++, mais simplifiée.`
- `Grande communauté et documentation abondante.`
- `Utilisé dans de nombreux domaines, notamment le développement web, mobile, et d'entreprise.`

Caractéristique	Description
Orienté objet	Java suit les principes de la programmation orientée objet (POO), tels que l'encapsulation, l'héritage, et le polymorphisme.
Portable	Grâce à la JVM, un programme Java peut être exécuté sur n'importe quel système d'exploitation.

Sécurisé	Java dispose de fonctionnalités de sécurité intégrées, comme la gestion des exceptions et le sandboxing.
Multi-thread	Java supporte la programmation multi-thread, permettant l'exécution simultanée de plusieurs tâches.
Gestion automatique de la mémoire	Java utilise un garbage collector pour gérer automatiquement la mémoire, réduisant les risques de fuites de mémoire.

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

Pour commencer à développer en Java, vous devez installer le JDK (Java Development Kit) et un IDE (Integrated Development Environment). Voici les étapes pour configurer votre environnement de développement :

- `Téléchargez et installez le JDK depuis le site officiel d'Oracle ou OpenJDK.`
- `Configurez la variable d'environnement JAVA_HOME pour pointer vers le répertoire d'installation du JDK.`
- `Ajoutez le chemin du répertoire 'bin' du JDK à la variable d'environnement PATH pour pouvoir exécuter les commandes Java depuis le terminal.`

- Choisissez un IDE comme IntelliJ IDEA, Eclipse, ou NetBeans pour faciliter le développement.

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

Premiers pas en Java

Structure de base d'un programme Java

Un programme Java est structuré autour de classes et de méthodes. Chaque programme Java commence par une classe principale, qui contient la méthode `main()`, point d'entrée du programme. Une classe est un modèle ou un plan pour créer des objets, et elle encapsule des données (attributs) et des comportements (méthodes).

- Une classe est définie avec le mot-clé `class`.

- Le nom de la classe doit correspondre au nom du fichier Java.
- Les méthodes sont définies à l'intérieur de la classe.
- Le code Java est sensible à la casse.

```
// Exemple de structure de base d'un programme Java
public class MonPremierProgramme {
    // Méthode principale
    public static void main(String[] args) {
        System.out.println("Bonjour, monde !");
    }
}
```

La méthode main()

La méthode `main()` est le point d'entrée de tout programme Java. Elle est obligatoire et doit être déclarée comme `public`, `static`, et `void`. Elle prend un tableau de chaînes de caractères (`String[] args`) comme argument, qui peut être utilisé pour passer des paramètres au programme.

- `public` : La méthode est accessible de partout.
- `static` : La méthode appartient à la classe et non à une instance de la classe.

- `void` : La méthode ne retourne aucune valeur.
- `String[] args` : Permet de passer des arguments en ligne de commande.

```
        // Exemple de la méthode main()
        public class MonPremierProgramme {
            public static void main(String[] args) {
                if (args.length > 0) {
                    System.out.println("Premier argument : " + args[0]);
                } else {
                    System.out.println("Aucun argument passé.");
                }
            }
        }
```

Compilation et exécution d'un programme

Pour exécuter un programme Java, il faut d'abord le compiler en bytecode à l'aide du compilateur `javac`, puis exécuter le bytecode avec la machine virtuelle Java (`java`). Le bytecode est un code intermédiaire qui peut être exécuté sur n'importe quelle plateforme disposant d'une JVM (Java Virtual Machine).

- Compilation : `javac MonPremierProgramme.java`
- Exécution : `java MonPremierProgramme`

- Le fichier source doit avoir l'extension `.java`.
- Le fichier compilé a l'extension `.class`.

```
// Exemple de compilation et exécution
// Compilation : javac MonPremierProgramme.java
// Exécution : java MonPremierProgramme
```

Étape	Commande	Description
Compilation	javac MonPremierProgramme.java	Compile le code source en bytecode.
Exécution	java MonPremierProgramme	Exécute le bytecode sur la JVM.

Jour 2

Les bases de la syntaxe Java

Variables et types de données

En Java, une variable est un conteneur qui stocke des données. Chaque variable a un type de données qui détermine la nature et la taille des données qu'elle peut contenir. Les types de données en Java sont divisés en deux catégories : les types primitifs et les types de référence.

- `Types primitifs : byte, short, int, long, float, double, char, boolean`
- `Types de référence : objets, tableaux, chaînes de caractères`

```
int age = 25; // Déclaration d'une variable de type int
String name = "John"; // Déclaration d'une variable de type String
```

Type	Taille	Valeur par défaut
byte	8 bits	0
short	16 bits	0
int	32 bits	0
long	64 bits	0L
float	32 bits	0.0f
double	64 bits	0.0d
char	16 bits	'\u0000'
boolean	1 bit	false

Opérateurs arithmétiques et logiques

Les opérateurs en Java sont utilisés pour effectuer des opérations sur des variables et des valeurs. Les opérateurs arithmétiques sont utilisés pour les calculs mathématiques, tandis que les opérateurs logiques sont utilisés pour les

comparaisons et les conditions.

- `Opérateurs arithmétiques : +, -, *, /, %`
- `Opérateurs de comparaison : ==, !=, >, <, >=, <=`
- `Opérateurs logiques : &&, ||, !`

```
int a = 10;
int b = 20;
int sum = a + b; // Addition
boolean isEqual = (a == b); // Comparaison
boolean result = (a > 5) && (b < 30); // ET logique
```

Opérateur	Description	Exemple
+	Addition	a + b
-	Soustraction	a - b
*	Multiplication	a * b
/	Division	a / b

%	Modulo	a % b
==	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
&&	ET logique	a && b
||	OU logique	a || b
!	NON logique	!a

Structures de contrôle (if, else, switch)

Les structures de contrôle en Java permettent de contrôler le flux d'exécution du programme en fonction de certaines conditions. Les structures de contrôle les plus couramment utilisées sont 'if', 'else', et 'switch'.

- `'if' : exécute un bloc de code si une condition est vraie`
- `'else' : exécute un bloc de code si la condition 'if' est fausse`
- `'switch' : permet de sélectionner un bloc de code à exécuter parmi plusieurs options`

```
int number = 10;

// Structure if-else
if (number > 0) {
    System.out.println("Le nombre est positif.");
} else {
    System.out.println("Le nombre est négatif.");
}

// Structure switch
switch (number) {
    case 10:
        System.out.println("Le nombre est 10.");
        break;
    case 20:
        System.out.println("Le nombre est 20.");
        break;
    default:
        System.out.println("Le nombre n'est ni 10 ni 20.");
}
```

Les boucles en Java

Boucle for

La boucle 'for' est utilisée pour répéter un bloc de code un nombre spécifique de fois. Elle est composée de trois parties : l'initialisation, la condition de continuation, et l'incrémentation/décrémentation.

- `Initialisation : Déclare et initialise une variable de contrôle.`
- `Condition : Vérifiée avant chaque itération. Si elle est vraie, la boucle continue.`
- `Incrémentation : Exécutée après chaque itération pour modifier la variable de contrôle.`

```
for (int i = 0; i < 5; i++) {
    System.out.println("Itération : " + i);
}
```

Boucle while

La boucle 'while' répète un bloc de code tant qu'une condition spécifiée est vraie. La condition est vérifiée avant chaque itération.

- `La boucle s'exécute tant que la condition est vraie.`

- Si la condition est fausse dès le début, la boucle ne s'exécute pas.

```
int i = 0;
while (i < 5) {
    System.out.println("Itération : " + i);
    i++;
}
```

Boucle do-while

La boucle 'do-while' est similaire à la boucle 'while', mais la condition est vérifiée après chaque itération. Cela garantit que le bloc de code est exécuté au moins une fois.

- Le bloc de code est exécuté au moins une fois, même si la condition est fausse.
- La condition est vérifiée après chaque itération.

```
int i = 0;
do {
    System.out.println("Itération : " + i);
    i++;
} while (i < 5);
```

Utilisation de break et continue

Les mots-clés 'break' et 'continue' sont utilisés pour contrôler l'exécution des boucles. 'break' termine immédiatement la boucle, tandis que 'continue' saute l'itération courante et passe à la suivante.

- `'break' : Interrompt la boucle et sort de celle-ci.`
- `'continue' : Saute le reste du code dans l'itération courante et passe à la suivante.`

```
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        break; // Sort de la boucle lorsque i atteint 5
    }
    if (i % 2 == 0) {
        continue; // Saute les itérations paires
    }
    System.out.println("Itération : " + i);
}
```

Jour 3

Les tableaux 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 dans une seule variable. La déclaration d'un tableau se fait en spécifiant le type des éléments et la taille du tableau. L'initialisation peut se faire soit lors de la déclaration, soit ultérieurement.

- `Déclaration : `type[] nomTableau;``
- `Initialisation : `nomTableau = new type[taille];``
- `Déclaration et initialisation en une ligne : `type[] nomTableau = {valeur1, valeur2, ...};``

Accès et manipulation des éléments

Les éléments d'un tableau sont accessibles via leur indice, qui commence à 0. Il est possible de lire, modifier ou parcourir les éléments d'un tableau en utilisant des boucles ou des méthodes spécifiques.

- Accès à un élément : `nomTableau[indice]`
- Modification d'un élément : `nomTableau[indice] = nouvelleValeur;`
- Parcours d'un tableau : Utilisation de boucles `for` ou `foreach`

Tableaux multidimensionnels

Un tableau multidimensionnel est un tableau de tableaux. En Java, il est possible de créer des tableaux à deux dimensions (matrices) ou plus. Chaque dimension est représentée par un ensemble de crochets supplémentaires.

- Déclaration : `type[][] nomTableau;`
- Initialisation : `nomTableau = new type[taille1][taille2];`
- Accès à un élément : `nomTableau[indice1][indice2]`

Introduction à la programmation orientée objet (POO)

Concepts de base de la POO

La programmation orientée objet (POO) est un paradigme de programmation qui organise le code autour des objets, qui sont des instances de classes. Les principaux concepts de la POO incluent l'encapsulation, l'héritage, le polymorphisme et l'abstraction.

- `Encapsulation : Masquer les détails internes d'un objet et exposer uniquement les interfaces nécessaires.`
- `Héritage : Permet à une classe de dériver des propriétés et des méthodes d'une autre classe.`
- `Polymorphisme : Permet à un objet de prendre plusieurs formes, c'est-à-dire de se comporter différemment selon le contexte.`

- `Abstraction : Simplifier les systèmes complexes en modélisant uniquement les aspects pertinents.`

Concept	Description
Encapsulation	Masquage des détails internes
Héritage	Dérivation de propriétés et méthodes
Polymorphisme	Comportement contextuel
Abstraction	Modélisation des aspects pertinents

Classes et objets

Une classe est un modèle ou un plan pour créer des objets. Un objet est une instance d'une classe. Les classes définissent les attributs (variables) et les méthodes (fonctions) que les objets auront.

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

- Attributs : Variables définies dans une classe.
- Méthodes : Fonctions définies dans une classe.

```
// Exemple de classe en Java
class Voiture {
    // Attributs
    String marque;
    int annee;

    // Méthode
    void demarrer() {
        System.out.println("La voiture démarre.");
    }
}

// Création d'un objet
Voiture maVoiture = new Voiture();
maVoiture.marque = "Toyota";
maVoiture.annee = 2020;
maVoiture.demarrer();
```

Méthodes et constructeurs

Les méthodes sont des fonctions définies dans une classe qui décrivent le comportement des objets. Les constructeurs sont des méthodes spéciales utilisées pour initialiser les objets lors de leur création.

- Méthodes : Fonctions définies dans une classe.
- Constructeurs : Méthodes spéciales pour initialiser les objets.

```
// Exemple de constructeur en Java
class Voiture {
    String marque;
    int annee;

    // Constructeur
    Voiture(String marque, int annee) {
        this.marque = marque;
        this.annee = annee;
    }

    // Méthode
    void demarrer() {
        System.out.println("La voiture démarre.");
    }
}

// Utilisation du constructeur
Voiture maVoiture = new Voiture("Toyota", 2020);
maVoiture.demarrer();
```

Jour 4

Approfondissement de la POO

Encapsulation

L'encapsulation est un principe fondamental de la programmation orientée objet (POO) qui consiste à regrouper les données (attributs) et les méthodes qui les manipulent au sein d'une même classe, tout en contrôlant l'accès à ces données. Cela permet de protéger les données internes d'une classe contre des modifications non autorisées et de garantir l'intégrité des objets.

- `Utilisation de modificateurs d'accès (public, private, protected) pour contrôler la visibilité des attributs et méthodes.`
- `Mise en place de méthodes getters et setters pour accéder et modifier les attributs privés.`

- Amélioration de la maintenance et de la sécurité du code.

```
public class Personne {
    private String nom; // Attribut privé

    // Getter
    public String getNom() {
        return nom;
    }

    // Setter
    public void setNom(String nom) {
        this.nom = nom;
    }
}
```

Héritage et polymorphisme

L'héritage permet à une classe (appelée classe enfant) de hériter des attributs et méthodes d'une autre classe (appelée classe parent). Le polymorphisme permet quant à lui à une méthode de se comporter différemment selon le contexte, notamment en redéfinissant des méthodes dans les classes enfants.

- L'héritage favorise la réutilisation du code et la hiérarchie des classes.

- Le polymorphisme permet une plus grande flexibilité dans la gestion des objets.
- Utilisation du mot-clé 'extends' pour l'héritage et 'override' pour la redéfinition de méthodes.

```
class Animal {
    void faireDuBruit() {
        System.out.println("L'animal fait un bruit.");
    }
}

class Chien extends Animal {
    @Override
    void faireDuBruit() {
        System.out.println("Le chien aboie.");
    }
}
```

Surcharge et redéfinition de méthodes

La surcharge de méthodes permet de définir plusieurs méthodes avec le même nom mais avec des paramètres différents dans une même classe. La redéfinition de méthodes permet à une classe enfant de redéfinir une méthode héritée de sa classe parent.

- La surcharge est basée sur la signature de la méthode (nom et paramètres).
- La redéfinition nécessite que la méthode ait la même signature que celle de la classe parent.
- La redéfinition permet d'adapter le comportement de la méthode dans la classe enfant.

```
class Calculatrice {
    // Surcharge
    int addition(int a, int b) {
        return a + b;
    }

    double addition(double a, double b) {
        return a + b;
    }
}

class CalculatriceAvancee extends Calculatrice {
    // Redéfinition
    @Override
    int addition(int a, int b) {
        return a + b + 10;
    }
}
```

Classes abstraites et interfaces

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

- Les classes abstraites sont utiles pour définir un comportement commun à plusieurs classes.
- Les interfaces permettent de définir des contrats que les classes doivent respecter.
- Une classe peut implémenter plusieurs interfaces mais ne peut hériter que d'une seule classe abstraite.

```
abstract class Forme {
    abstract double calculerAire(); // Méthode abstraite
}

class Cercle extends Forme {
    double rayon;

    Cercle(double rayon) {
        this.rayon = rayon;
    }

    @Override
    double calculerAire() {
        return Math.PI * rayon * rayon;
    }
}

interface Dessinable {
    void dessiner();
}

class Rectangle implements Dessinable {
    @Override
    public void dessiner() {
        System.out.println("Dessiner un rectangle.");
    }
}
```

Introduction aux exceptions

Qu'est-ce qu'une exception ?

Une exception en Java est un événement qui se produit pendant l'exécution d'un programme et qui interrompt le flux normal des instructions. Les exceptions sont utilisées pour gérer les erreurs et autres événements exceptionnels de manière structurée.

- `Les exceptions sont des objets qui représentent des erreurs ou des conditions inattendues.`
- `Elles permettent de séparer la logique métier de la gestion des erreurs.`
- `Java fournit une hiérarchie de classes d'exceptions, avec `Throwable` comme classe de base.`

Type d'exception	Description
Checked Exception	Doit être gérée explicitement (ex: IOException)
Unchecked Exception	Ne nécessite pas de gestion explicite (ex: NullPointerException)

Error	Problème grave qui ne devrait pas être attrapé (ex: OutOfMemoryError)

Gestion des exceptions avec try-catch

La gestion des exceptions en Java se fait principalement à l'aide des blocs `try-catch`. Le bloc `try` contient le code susceptible de générer une exception, tandis que le bloc `catch` permet de capturer et de gérer cette exception.

- Le bloc `try` doit être suivi d'au moins un bloc `catch` ou d'un bloc `finally`.
- Plusieurs blocs `catch` peuvent être utilisés pour gérer différents types d'exceptions.
- L'ordre des blocs `catch` est important : les exceptions les plus spécifiques doivent être attrapées en premier.

```
try {
    int result = 10 / 0; // Division par zéro
} catch (ArithmeticException e) {
    System.out.println("Erreur arithmétique : " + e.getMessage());
}
```

Utilisation de finally

Le bloc `finally` est utilisé pour exécuter du code qui doit toujours être exécuté, qu'une exception soit levée ou non.

Cela est utile pour libérer des ressources ou effectuer des opérations de nettoyage.

- Le bloc `finally` est exécuté après les blocs `try` et `catch`, quel que soit le résultat.
- Il est souvent utilisé pour fermer des fichiers, des connexions réseau, etc.
- Le bloc `finally` est optionnel, mais fortement recommandé dans certains cas.

```
        FileInputStream file = null;
        try {
            file = new FileInputStream("fichier.txt");
            // Lecture du fichier
        } catch (FileNotFoundException e) {
            System.out.println("Fichier non trouvé : " + e.getMessage());
        } finally {
            if (file != null) {
                try {
                    file.close();
                } catch (IOException e) {
                    System.out.println("Erreur lors de la fermeture du fichier : " + e.getMessage());
                }
            }
        }
```