TP N°1

Java de base

Préparée par : RGUIBI Farah

Objectif :

Ce travail pratique vise à initier les étudiants aux fondamentaux de la programmation Java en couvrant les aspects essentiels du développement : la mise en place de l'environnement de développement, la compréhension du cycle de compilation et d'exécution, ainsi que la maîtrise des structures de base du langage.

À travers une série d'exercices progressifs, nous développons les compétences nécessaires pour créer des programmes Java fonctionnels en manipulant les variables, les structures de contrôle (conditions et boucles), les méthodes statiques, et les types de données fondamentaux tels que les entiers et les chaînes de caractères.

Ce TP permet également de découvrir l'interaction avec l'utilisateur via la classe `Scanner`, de comprendre l'importance de la validation des données, et d'appliquer des algorithmes classiques de programmation. L'objectif final est d'acquérir une base solide en programmation Java qui servira de fondation pour des concepts plus avancés de la programmation orientée objet.

FIGURE 1 – Java

1 Mise en place de l'environnement

1.1 Installation et configuration

- Télécharger et installer Eclipse
- Télécharger et installer Java
- Vérifier l'installation en affichant la version de Java dans un cmd :

```
java --version

```

- En cas d'échec, télécharger le JDK et l'intégrer dans la variable d'environnement

2 Les exercices

2.1 Exercice 0 : Premier programme Java (en ligne de commande)

2.1.1 Idée de l'exercice :

Comprendre le processus de compilation et d'exécution d'un programme Java en utilisant uniquement la ligne de commande, sans IDE. Cet exercice permet de saisir les bases du cycle de développement Java : écriture du code source, compilation en bytecode, et exécution par la JVM.

2.1.2 Étapes réalisées :

1. Création d'un fichier texte nommé `BonjourENDIAD.java` avec Notepad
2. Écriture d'un programme simple qui affiche le message : *'Bonjour ENIAD !'*
3. Compilation du fichier en utilisant la commande `javac BonjourENDIAD.java` dans le cmd
4. Exécution du fichier binaire avec la commande `java BonjourENDIAD` et observation du résultat

2.1.3 Résultat :

Le programme a affiché avec succès le message "Bonjour ENIAD !" dans la console, validant ainsi la bonne installation de Java et la compréhension du processus de compilation/exécution.

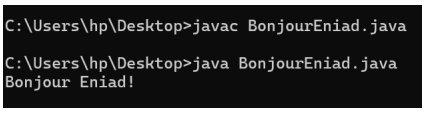

FIGURE 2 – Premier programme Java (en ligne de commande)

2.2 Exercice 1 : Échange de valeurs

2.2.1 Idée de l'exercice :

Comprendre le mécanisme d'échange de valeurs entre deux variables en utilisant une variable temporaire. C'est un algorithme fondamental qui illustre la manipulation de variables et l'importance de l'ordre des opérations en programmation.

2.2.2 Approche adoptée :

J'ai développé une méthode statique `echange()` qui prend deux entiers en paramètres. La méthode utilise une variable temporaire pour sauvegarder la première valeur avant de procéder à l'échange. Dans le `main`, j'ai utilisé la classe `Scanner` pour permettre à l'utilisateur de saisir les deux valeurs, puis j'ai affiché les valeurs avant et après l'échange.

2.2.3 Points clés de l'implémentation :

- Utilisation d'une variable temporaire pour stocker la valeur de `n`
- Affectation de la valeur de `m` à `n`
- Affectation de la valeur temporaire à `m`
- Saisie interactive des valeurs via `Scanner`
- Affichage des valeurs avant et après l'échange pour vérification

```
Entrer le premiere valeur
5
Entrer la deuxieme valeur
9
les valeurs avant l'echange sont n =5 et m = 9
les valeurs apres l'echange sont n =9et m = 5
```

FIGURE 3 – Échange de valeurs

2.3 Exercice 2 : Maximum de trois nombres

2.3.1 Idée de l'exercice :

Développer une méthode statique qui compare trois nombres entiers et détermine le plus grand. Cet exercice illustre l'utilisation des structures conditionnelles, la création de méthodes réutilisables, et la logique de comparaison multiple.

2.3.2 Approche adoptée :

J'ai créé une méthode `max()` qui prend trois entiers en paramètres. La méthode initialise une variable `max` avec la première valeur, puis effectue deux comparaisons successives avec les deuxième et troisième valeurs. Si une valeur est supérieure à `max`, elle devient la nouvelle valeur maximale. Dans le `main`, l'utilisateur saisit les trois valeurs via `Scanner`.

2.3.3 Points clés de l'implémentation :

- Initialisation de la variable `max` avec la première valeur (`x`)
- Comparaison avec la deuxième valeur : si `y > max`, alors `max = y`
- Comparaison avec la troisième valeur : si `z > max`, alors `max = z`
- Utilisation de deux structures `if` indépendantes pour les comparaisons
- Affichage du résultat final

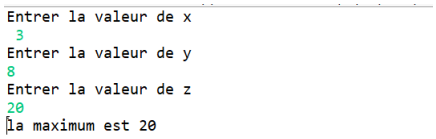

FIGURE 4 – Maximum de trois nombres

2.4 Exercice 3 : Calcul du factoriel

2.4.1 Idée de l'exercice :

Implémenter le calcul mathématique du factoriel (n ! = n × (n-1) × ... × 2 × 1) en utilisant une boucle itérative. Cet exercice permet de maîtriser les boucles, la manipulation des variables, et la gestion des cas particuliers.

2.4.2 Approche adoptée :

J'ai développé une méthode `fact()` qui calcule le factoriel d'un nombre entier. La méthode gère d'abord les cas particuliers (0 ! = 1 et 1 ! = 1), puis utilise une boucle `for` pour calculer le factoriel des autres nombres. Dans le `main`, l'utilisateur entre un nombre et le programme affiche son factoriel.

2.4.3 Points clés de l'implémentation :

- Gestion des cas de base : retour de 1 si `a = 0` ou `a = 1`
- Initialisation d'une variable `b` à 1 pour stocker le résultat
- Utilisation d'une boucle `for` de 1 à `a` pour multiplier successivement
- Multiplication cumulative : `b = b * i` à chaque itération
- Retour du résultat final

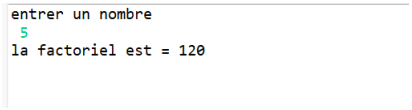

FIGURE 5 – Calcul du factoriel

2.5 Exercice 4 : Caractère à un index spécifique

2.5.1 Idée de l'exercice :

Apprendre à manipuler les chaînes de caractères en Java, particulièrement l'accès à un caractère spécifique via son index. Cet exercice introduit la méthode `charAt()` et la validation des indices pour éviter les erreurs d'exécution.

2.5.2 Approche adoptée :

J'ai créé une méthode `remplacer()` (qui en réalité extrait un caractère) prenant une chaîne et un index en paramètres. La méthode vérifie d'abord si l'index est valide (entre 0 et la longueur de la chaîne moins 1), puis utilise `charAt()` pour récupérer le caractère. Dans le `main`, l'utilisateur saisit la chaîne et l'index.

2.5.3 Points clés de l'implémentation :

- Validation de l'index avec une condition : `index >= 0 && index < chaine.length()`
- Utilisation de la méthode `charAt(index)` pour extraire le caractère
- Affichage du caractère trouvé avec son index
- Gestion des erreurs : affichage de "Invalide" si l'index est hors limites
- Utilisation de `Scanner` pour la saisie de la chaîne et de l'index

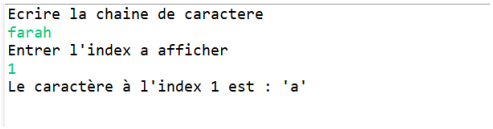

FIGURE 6 – Caractère à un index spécifique

2.6 Exercice 5 : Remplacement d'un caractère

2.6.1 Idée de l'exercice :

Développer une méthode qui modifie une chaîne de caractères en remplaçant un caractère à une position donnée. Comme les chaînes sont immuables en Java, cet exercice enseigne l'utilisation de `StringBuilder` pour effectuer des modifications.

2.6.2 Approche adoptée :

J'ai créé une méthode `remplacerchaine()` qui prend trois paramètres : la chaîne, l'index, et le nouveau caractère. La méthode utilise `StringBuilder` pour permettre la modification de la chaîne. Après validation de l'index, elle utilise `setCharAt()` pour remplacer le caractère, puis retourne la nouvelle chaîne.

2.6.3 Points clés de l'implémentation :

- Validation de l'index avant toute modification
- Utilisation de `StringBuilder` pour contourner l'immuabilité des chaînes
- Utilisation de la méthode `setCharAt(index, ch)` pour remplacer le caractère
- Conversion du `StringBuilder` en `String` avec `toString()`
- Retour de la chaîne originale en cas d'index invalide

- Saisie du caractère avec `sc.next().charAt(0)`

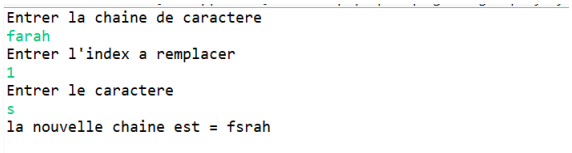

FIGURE 7 – Remplacement d'un caractère

2.7 Exercice 6 : Inversion d'une chaîne

2.7.1 Idée de l'exercice :

Implémenter un algorithme d'inversion de chaîne, qui est un exercice classique illustrant la manipulation de caractères et les algorithmes de traitement de chaînes. L'objectif est de parcourir la chaîne de la fin vers le début pour construire la chaîne inversée.

2.7.2 Approche adoptée :

J'ai développé une méthode `reverse()` qui prend une chaîne en paramètre et retourne son inverse. La méthode utilise une boucle `for` qui parcourt la chaîne de la fin vers le début (de `length()-1` à 0), et concatène chaque caractère à une nouvelle chaîne vide. Dans le `main`, l'utilisateur saisit une chaîne et le programme affiche son inverse.

2.7.3 Points clés de l'implémentation :

- Initialisation d'une chaîne vide `reverse` pour stocker le résultat
- Utilisation d'une boucle `for` décroissante : de `chaine.length()-1` à 0
- Extraction de chaque caractère avec `charAt(i)`
- Concaténation successive des caractères dans l'ordre inverse
- Retour de la chaîne inversée complète
- Affichage de la chaîne originale et de son inverse pour comparaison

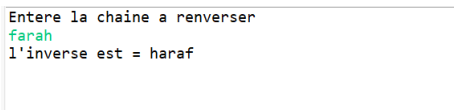

FIGURE 8 – Inversion d'une chaîne

3 Synthèse

Ces exercices m'ont permis de maîtriser les concepts fondamentaux de la programmation Java :

- **Compilation et exécution :** Compréhension du processus de compilation avec `javac` et d'exécution avec `java`
- **Méthodes statiques :** Création et utilisation de méthodes pour structurer le code de manière modulaire
- **Structures de contrôle :** Maîtrise des conditions (`if`) et des boucles (`for`)
- **Manipulation de chaînes :** Utilisation des classes `String` et `StringBuilder` avec leurs méthodes
- **Saisie utilisateur :** Utilisation efficace de la classe `Scanner` pour l'interaction
- **Validation des données :** Vérification des indices et gestion des cas limites

Ces compétences constituent une base solide pour progresser vers des concepts plus avancés de la programmation orientée objet en Java.