

Rapport de TP : JAVA

Sommaire :

1. Introduction	1
2. Conception de la solution	2
3. Réalisation	3
Quizz Java :	3
Exercice de Programmation :	5
1. Identifiants de classes	5
2. Identifiants de méthodes	6
3. Classe Java ParoleChanson.java	7
Debug :	13
Debug 1)	13
Debug 2)	15
Debug 3)	18
Debug 4)	20
Voici les problèmes que j'ai trouvé :	20
4. Conclusion	22

1. Introduction

Objectif :

L'objectif de ce TP est de découvrir les bases du langage Java, en corrigeant des programmes contenant volontairement des erreurs de syntaxe, en répondant à un questionnaire de compréhension théorique et en développant de petits programmes d'affichage et de manipulation d'objets.

Mise en situation :

En tant qu'étudiant en BTS SIO, je dois apprendre à analyser un code, identifier les fautes fréquentes (syntaxe, majuscules/minuscules, ponctuation, signatures de méthodes) et les corriger pour obtenir un programme exécutable. Je dois aussi renforcer mes connaissances théoriques sur les concepts fondamentaux de Java : langage machine, compilateur, variables, méthodes, classes et objets.

Résolution :

J'ai travaillé depuis l'IDE Eclipse (avec le JDK installé), ce qui m'a permis de compiler, exécuter et déboguer les fichiers fournis. J'ai corrigé les

programmes nommés Debug1, Debug2, Debug3 et Debug4, puis j'ai développé des classes supplémentaires (ParoleChanson, TableEtChaises, Triangle, Yummy...) en respectant les consignes données.

2. Conception de la solution

La résolution de ce TP, consacré à mes premiers pas en Java, peut être décomposée en plusieurs sous-problèmes :

Correction de programmes existants :

- Identifier les erreurs présentes dans les fichiers Debug (mauvaise casse, fautes de frappe, erreurs de syntaxe, signature de `main` incorrecte, chaînes de caractères non fermées).
- Corriger le code et le renommer en FixBug1.java, FixBug2.java, etc. pour obtenir des versions fonctionnelles.

Révision théorique :

- Répondre à un QCM portant sur le langage machine, les compilateurs, les variables, la syntaxe et les objets.
- Faire la distinction entre langage bas niveau et haut niveau, comprendre les rôles du compilateur et des instructions Java.

Développement de programmes simples :

- Créer une classe affichant les paroles d'une chanson (ParoleChanson.java).
- Réaliser des modèles graphiques avec `System.out.println` (TableEtChaises.java, Triangle.java).
- Utiliser des boîtes de dialogue (`JOptionPane`) pour afficher des messages ou générer un nombre aléatoire.

Cas pratique Yummy :

- Développer un package **Yummy** contenant plusieurs classes avec des slogans, parfois encadrés par des symboles (* ou S).
- Structurer le code pour respecter les conventions de nommage et d'organisation d'un projet Java.

3. Réalisation

Quizz Java :

1. Le langage machine le plus basique niveau circuit est :

a) Le langage machine

C'est le seul langage directement compris par le processeur. Il est constitué uniquement de 0 et 1 (binaire). Java, C++ ou autres sont traduits en langage machine avant exécution.

2. Les langages qui permettent d'utiliser un vocabulaire qui utilise les termes : read, write ou add sont :

a) Haut niveau

→ Ces langages utilisent un vocabulaire proche du langage humain

3. Les règles du langage de programmation constituent :

a) la syntaxe

La syntaxe définit la manière correcte d'écrire des instructions (ordre des mots, symboles obligatoires). Par exemple, oublier un ; en Java est une erreur de syntaxe.

4. Un _____ traduit les instructions de langage de haut niveau en code machine :

c) un compilateur

Le compilateur analyse le code source (Java, C, etc.) et le traduit en code machine ou en bytecode.

5. Les emplacements de mémoire nommés de l'ordinateur sont appelés :

b) variables

Une variable est un nom qui correspond à une case mémoire où l'on stocke une valeur (nombre, texte, etc.).

6. Les opérations individuelles utilisées dans un programme informatique sont souvent regroupées en unités logiques appelées :

a) procédures

Une procédure (ou fonction/méthode) regroupe plusieurs instructions pour les réutiliser facilement et structurer le programme.

7. Une instance de classe est :

c) un objet

Une classe est un modèle, et un objet est une réalisation concrète de ce modèle (ex. Classe : *Voiture*, Objet : *maVoiture*).

8. Java a une architecture :

a) neutre

Le code Java est compilé en bytecode indépendant du matériel. C'est la JVM qui l'exécute, ce qui rend Java portable entre différents systèmes.

9. Vous devez compiler les classes écrites en Java dans :

a) un bytecode

Le compilateur Java (**javac**) transforme le code source en **bytecode** (**.class**), interprété/exécuté par la JVM.

10. Toutes les instructions de programmation Java doivent se terminer par :

c) un point-virgule

Le `;` marque la fin d'une instruction (comme en C/C++). Sans lui, le compilateur renvoie une erreur de syntaxe.

Exercice de Programmation :

1. Identifiants de classes

Règles Java :

- Doivent commencer par une lettre ou `_` ou `$` (pas un chiffre).
- Pas d'espaces ni de caractères spéciaux (`#`, `!`, etc.).
- Ne pas utiliser de mot réservé (`void`, `class`, etc.).
- Convention : une **classe** commence par une **majuscule**, pas d'accents, pas d'espace (CamelCase).

Identifiant	Statut	Explication
maClasse	✓ Légal mais pas conventionnel	Valide, mais convention → commencer par une majuscule (<code>MaClasse</code>).
void	✗ Illégal	Mot réservé en Java.
Golden Retriever	✗ Illégal	Les espaces sont interdits.
invoice#	✗ Illégal	Le caractère <code>#</code> n'est pas autorisé.
36535CodePostal	✗ Illégal	Ne peut pas commencer par un chiffre.
Appartement	✓ Légal et conventionnel	Commence par majuscule, pas de caractère interdit.
Fruit	✓ Légal et conventionnel	Même remarque que ci-dessus.
8888	✗ Illégal	Ne peut pas commencer par un chiffre.
EcranTotal()	✗ Illégal	Les parenthèses <code>()</code> sont interdites dans un identifiant.
Acompte_recevable	✓ Légal mais pas conventionnel	L'underscore est permis, mais pas recommandé. Convention → CamelCase : <code>AcompteRecevable</code> .

2. Identifiants de méthodes

Règles Java (méthodes) :

- Même règles qu'une classe (pas de chiffres en début, pas d'espaces, pas de #, pas de mots réservés).
- Convention : commencent par une **minuscule**, puis CamelCase (`getValue()`).

Identifiant	Statut	Explication
<code>associationRoles()</code>	✓ Légal et conventionnel	Commence par minuscule, CamelCase.
<code>void()</code>	✗ Illégal	<code>void</code> est un mot réservé.
<code>Golden Retriever()</code>	✗ Illégal	Espace interdit.
<code>invoice#()</code>	✗ Illégal	<code>#</code> interdit.
<code>24500CodePostal()</code>	✗ Illégal	Ne commence pas par une lettre.
<code>PayrollApp()</code>	✓ Légal mais pas conventionnel	Valide, mais convention → commence par minuscule (<code>payrollApp()</code>).
<code>getReady()</code>	✓ Légal et conventionnel	Correspond bien au style Java (getter).
<code>911()</code>	✗ Illégal	Ne commence pas par une lettre.
<code>EcranTotal()</code>	✓ Légal mais pas conventionnel	Valide, mais convention → minuscule (<code>ecranTotal()</code>).
<code>Acompte_Recevable()</code>	✓ Légal mais pas conventionnel	L'underscore est permis, mais convention → <code>acompteRecevable()</code> .

3. Classe Java ParoleChanson.java

```
public class ParoleChanson {  
    public static void main(String[] args) {  
        System.out.println("Alors on danse, alors on danse, alors on danse,");  
        System.out.println("Alors on chante, et puis seulement quand c'est fini,");  
        System.out.println("Alors on pleure, alors on pleure, alors on pleure,");  
        System.out.println("Alors on rit, et puis seulement quand c'est fini.");  
    }  
}
```

1. Déclaration de la classe

```
public class ParoleChanson {
```

- En Java, tout programme doit être dans une **classe**.
 - Le nom de la classe doit correspondre exactement au nom du fichier : ici → `ParoleChanson.java`.
-

2. La méthode `main`

```
public static void main(String[] args) {
```

- C'est le **point d'entrée** du programme.
 - Quand on exécute le fichier compilé, la JVM commence toujours ici.
 - Sans cette méthode, ton programme ne démarrera pas.
-

3. Les instructions `System.out.println`

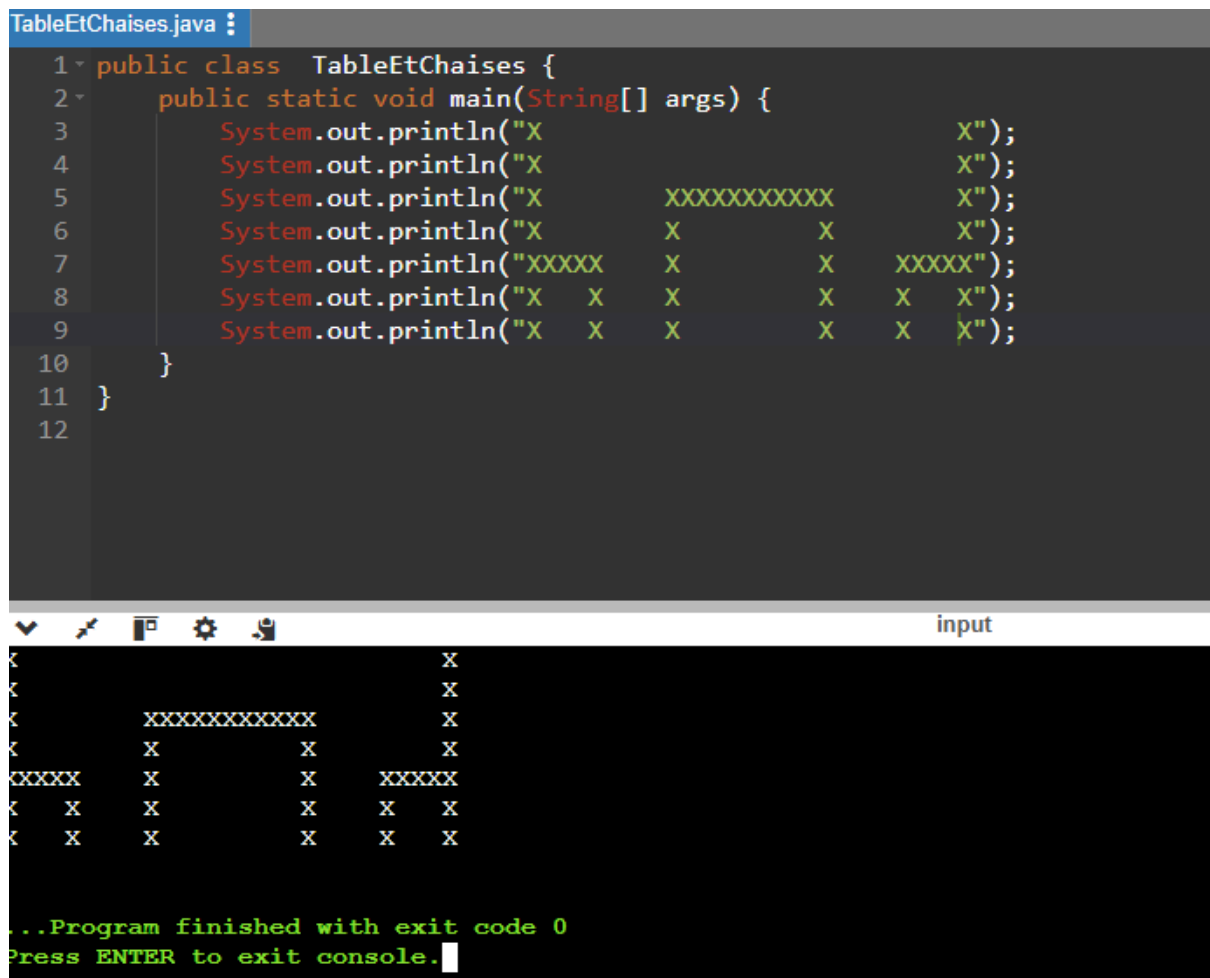
```
System.out.println("Alors on danse, alors on danse,  
alors on danse,");
```

- `System.out.println` → affiche un texte à l'écran **et passe à la ligne suivante**.
- Chaque appel affiche **une ligne de texte** (ici une ligne de la chanson).

- Les guillemets " " indiquent que c'est une chaîne de caractères (string).

4. Résultat à l'exécution

Quand on compile avec :



```
TableEtChaises.java :
1 public class TableEtChaises {
2     public static void main(String[] args) {
3         System.out.println("X");
4         System.out.println("X");
5         System.out.println("XXXXXXXXXXXXX");
6         System.out.println("X      X      X");
7         System.out.println("XXXXX  X      X  XXXXX");
8         System.out.println("X  X  X      X  X  X");
9         System.out.println("X  X  X      X  X  X");
10    }
11 }
12
```

input

```
X
X
X
X      XXXXXXXXXXXXX
X      X      X
XXXXX  X      X  XXXXX
X  X  X      X  X  X
X  X  X      X  X  X

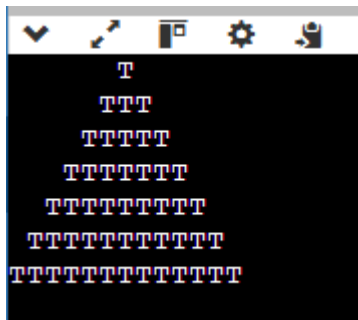
...Program finished with exit code 0
Press ENTER to exit console.
```

J'ai ici renommé le programme "TableEtChaises.java" puis j'ai écrit le code minutieusement en faisant attention à la place des X. Ce programme est un peu répétitif.

5)

```
Triangle.java :
1 public class Triangle {
2     public static void main(String[] args) {
3         System.out.println("    T");
4         System.out.println("   TTT");
5         System.out.println("  TTTTT");
6         System.out.println(" TTTTTTT");
7         System.out.println("TTTTTTTTT");
8         System.out.println("TTTTTTTTTTT");
9         System.out.println("TTTTTTTTTTTTT");
10    }
11 }
12
13
14
```

J'ai ici renommé le programme "Triangle.java" puis j'ai écrit le code minutieusement en faisant attention à la place des T. Ce programme est un peu répétitif comme celui TableEtChaises. En exécutant le code on obtient :



```
T
  TTT
   TTTTT
  TTTTTTT
 TTTTTTTTT
TTTTTTTTT
TTTTTTTTTTT
TTTTTTTTTTTTT
TTTTTTTTTTTTTTT
```

Donc le programme est correct.

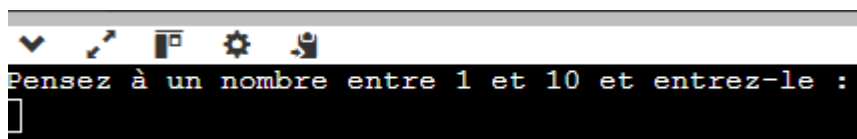
Cas Pratique :

1) Jeu basique avec nombre aléatoire

Voici le code que je propose :

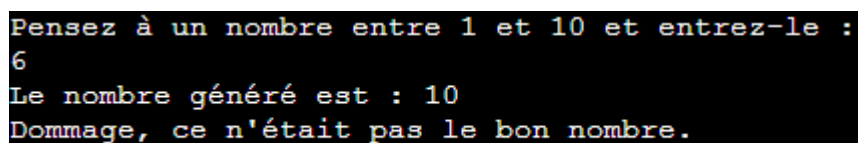
```
GuessNumber.java :
1 import java.util.Scanner;
2
3 public class GuessNumber {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6
7         System.out.println("Pensez à un nombre entre 1 et 10 et entrez-le :");
8         int userNumber = sc.nextInt();
9
10        int randomNumber = 1 + (int)(Math.random() * 10);
11        System.out.println("Le nombre généré est : " + randomNumber);
12
13        if(userNumber == randomNumber) {
14            System.out.println("Bravo, vous avez deviné !");
15        } else {
16            System.out.println("Dommage, ce n'était pas le bon nombre.");
17        }
18
19        sc.close();
20    }
21 }
22
```

Vérifions si le programme marche :



```
Pensez à un nombre entre 1 et 10 et entrez-le :
█
```

Le programme à l'air pour l'instant de fonctionner. Essayons d'entrer un chiffre : 6



```
Pensez à un nombre entre 1 et 10 et entrez-le :
6
Le nombre généré est : 10
Dommage, ce n'était pas le bon nombre.
```

Nous remarquons que le programme fonctionne comme attendu.

2) Devise de l'entreprise Yummy

a) Yummy.java

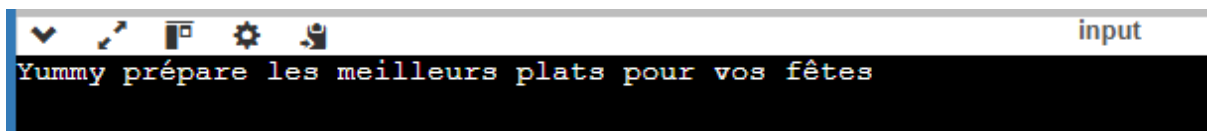
```

Yummy.java :
1 public class Yummy {
2     public static void main(String[] args) {
3         System.out.println("Yummy prépare les meilleurs plats pour vos fêtes");
4     }
5 }
6

```

Voici mon code pour un programme qui affiche la devise du restaurant de la société Yummy

Vérifions si le code fonctionne :



The screenshot shows a Java IDE terminal window. The title bar includes icons for a dropdown menu, a cursor, a window, a gear, and a person. The terminal text is "Yummy prépare les meilleurs plats pour vos fêtes". The word "input" is visible in the top right corner of the terminal area.

Donc, oui le code fonctionne correctement.

b) **Yummy2.java** (avec bordure en astérisques)

Dans cette partie, nous devons ajouter une bordure en astérisque

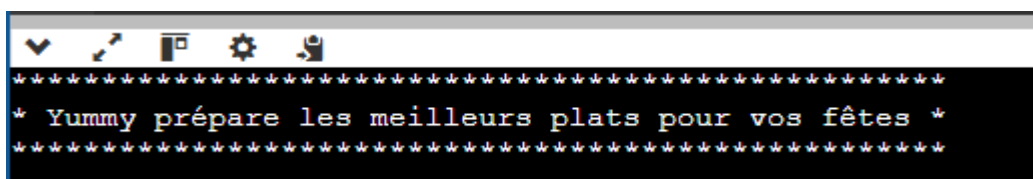
Voici le code (très) simplifié :

```

Yummy2.java :
1 public class Yummy2 {
2     public static void main(String[] args) {
3         System.out.println("*****");
4         System.out.println("* Yummy prépare les meilleurs plats pour vos fêtes *");
5         System.out.println("*****");
6     }
7 }
8

```

Vérifions si le code est correct :



The screenshot shows a Java IDE terminal window. The title bar includes icons for a dropdown menu, a cursor, a window, a gear, and a person. The terminal text is:

 * Yummy prépare les meilleurs plats pour vos fêtes *

 The text is centered and the asterisks form a border around the main line of text.

Donc, oui le code fonctionne.

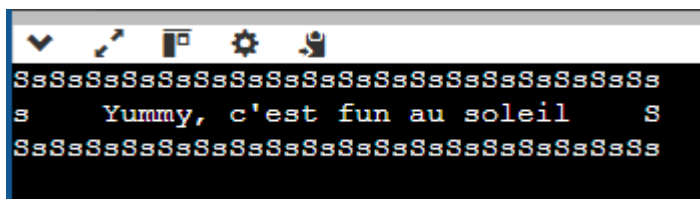
2) Version été avec "fun au soleil"

YummyFun.java (bordure avec 'S')

Voici le programme pour la nouvelle devise "Yummy, c'est fun au soleil" entourée d'une bordure composée de Ss répétés.

```
YummyFun.java :
1 public class YummyFun {
2     public static void main(String[] args) {
3         System.out.println("SsSsSsSsSsSsSsSsSsSsSsSsSsSsSsSsSs");
4         System.out.println("s    Yummy, c'est fun au soleil    S");
5         System.out.println("SsSsSsSsSsSsSsSsSsSsSsSsSsSsSsSsSs");
6     }
7 }
8
```

Vérifions si le programme marche :



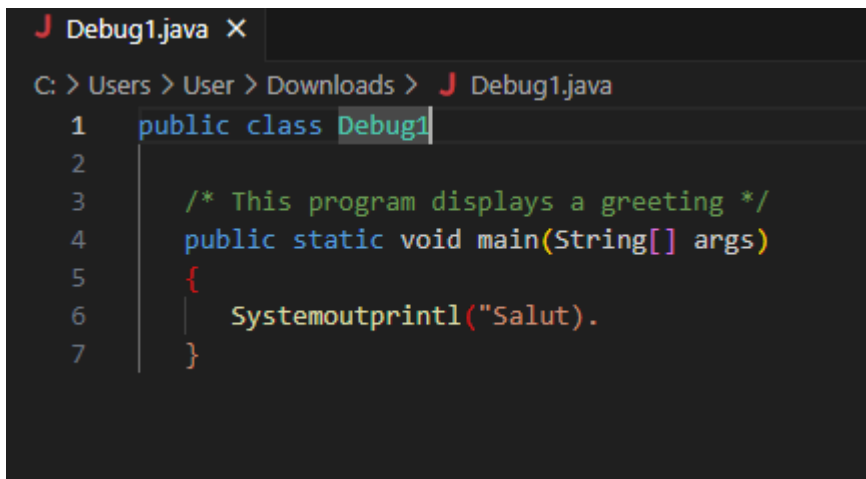
```
SsSsSsSsSsSsSsSsSsSsSsSsSsSsSsSsSs
s    Yummy, c'est fun au soleil    S
SsSsSsSsSsSsSsSsSsSsSsSsSsSsSsSsSs
```

Super, le programme est fonctionnel.

Debug :

Debug 1)

Voici le programme que nous devons résoudre :



```
1 public class Debug1
2
3     /* This program displays a greeting */
4     public static void main(String[] args)
5     {
6         Systemoutprintln("Salut").
7     }
```

Nous pouvons constater plusieurs problèmes :

Accolades manquantes après la déclaration de classe

En Java, après `public class Debug1`, il faut ouvrir une accolade `{` pour délimiter le bloc de la classe.

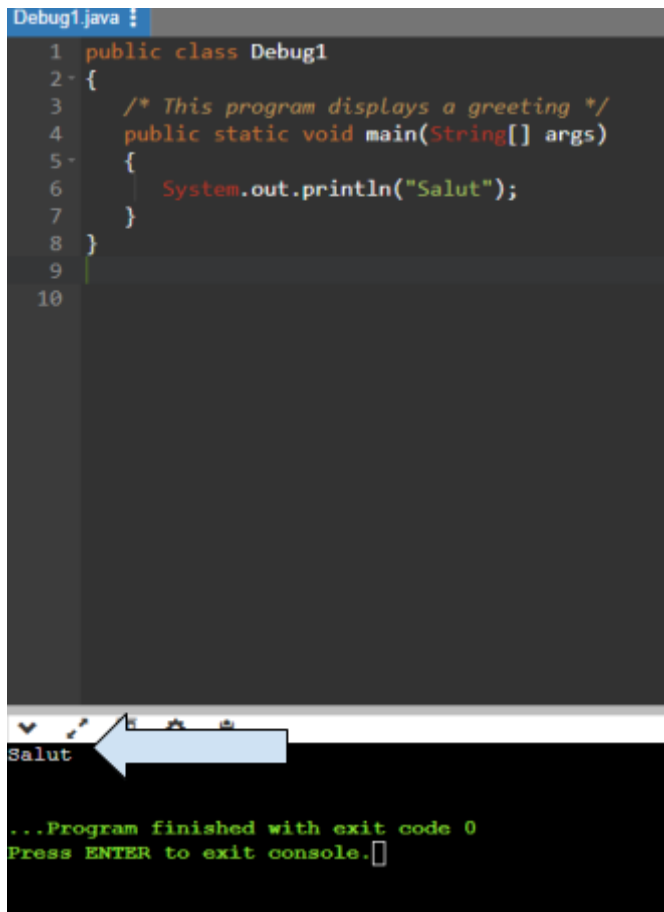
Erreur dans `Systemoutprintln`

L'instruction correcte est `System.out.println` (avec un point entre `System` et `out`, et un `ln` complet à la fin).

Guillemets et parenthèse mal placés dans la chaîne de caractères

Le `("Salut).` → il manque un guillemet fermant avant la parenthèse et le point-virgule est au mauvais endroit.

Code corrigé :

A screenshot of a Java IDE. The top pane shows a file named 'Debug1.java' with the following code:

```
1 public class Debug1
2 {
3     /* This program displays a greeting */
4     public static void main(String[] args)
5     {
6         System.out.println("Salut");
7     }
8 }
9
10
```

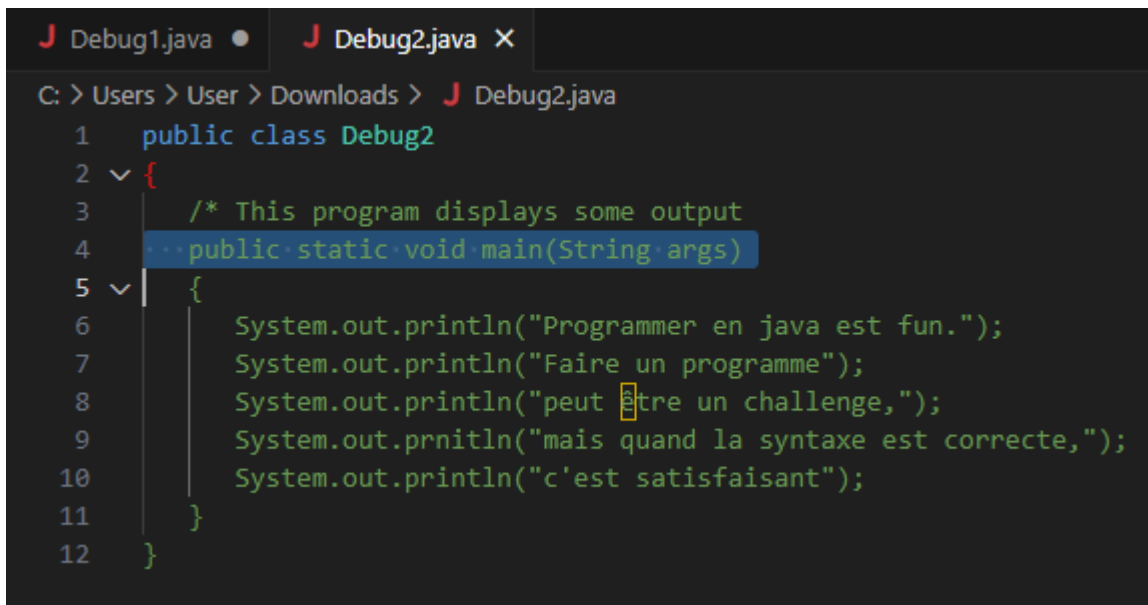
The bottom pane shows the program's output: 'Salut'. A blue arrow points from the 'Salut' output to the 'System.out.println("Salut");' line in the code. Below the output, it says '...Program finished with exit code 0' and 'Press ENTER to exit console.'.

Explication des corrections

- J'ai ajouté une accolade `{` après `public class Debug1` pour ouvrir le corps de la classe.
- J'ai remplacé `Systemoutprintln` par `System.out.println`, qui est la bonne méthode pour afficher du texte dans la console.
- J'ai corrigé la chaîne `"Salut)` en `"Salut"` et remplacé correctement le `;`.
- J'ai fermé la classe avec une accolade `}` à la fin.

Debug 2)

Voici le programme que nous devons résoudre :



```
C: > Users > User > Downloads > Debug2.java
1  public class Debug2
2  {
3      /* This program displays some output
4      ...public static void main(String args)
5      {
6          System.out.println("Programmer en java est fun.");
7          System.out.println("Faire un programme");
8          System.out.println("peut être un challenge,");
9          System.out.prnitln("mais quand la syntaxe est correcte,");
10         System.out.println("c'est satisfaisant");
11     }
12 }
```

Nous pouvons constater plusieurs problèmes :

1. Commentaire non fermé

La ligne `/* This program displays some output` ouvre un commentaire, mais il n'est jamais fermé par `*/`.

→ Résultat : tout le code qui suit est considéré comme un commentaire, donc erreurs à la compilation.

Paramètres du `main` incorrects

En Java, la méthode `main` doit être :

```
public static void main(String[] args)
```

2. Pas `(String args)` (il manque les crochets `[]`).

3. Faute de frappe dans `System.out.prnitln`


→ Ça doit être `System.out.println`.

Code corrigé :

```
Debug2.java :
1 public class Debug2
2 {
3     /* This program displays some output */
4     public static void main(String[] args)
5     {
6         System.out.println("Programmer en java est fun.");
7         System.out.println("Faire un programme");
8         System.out.println("peut être un challenge,");
9         System.out.println("mais quand la syntaxe est correcte,");
10        System.out.println("c'est satisfaisant");
11    }
12 }
13
```

input

Faire un programme
peut être un challenge,
mais quand la syntaxe est correcte,
c'est satisfaisant

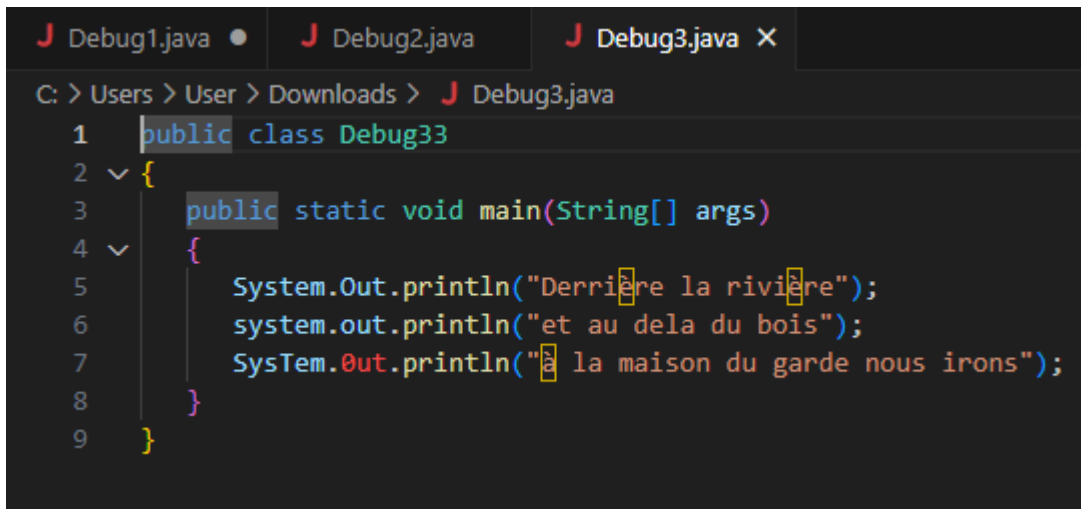


Explication des corrections

- Fermeture du commentaire `/* ... */` pour qu'il n'avale pas tout le code.
- Changement de `public static void main(String args)` → `public static void main(String[] args)` pour respecter la signature standard du point d'entrée d'un programme Java.
- Correction de la faute de frappe `prnitln` → `println`

Debug 3)

Voici le programme que nous devons résoudre :



```
1 public class Debug33
2 {
3     public static void main(String[] args)
4     {
5         System.Out.println("Derrière la rivière");
6         system.out.println("et au dela du bois");
7         SysTem.0ut.println("à la maison du garde nous irons");
8     }
9 }
```

Nous pouvons constater plusieurs problèmes :

1. Majuscules/minuscules incorrectes

- En Java, `System` est sensible à la casse. Il doit toujours être écrit exactement `System` avec un **S majuscule**.
- `Out` → doit être `out` (avec un **o minuscule**).
- `system` → doit être `System` (avec un **S majuscule**).
- `SysTem` → idem, doit être `System`.

2. Faute de frappe dans `0ut`

- Il y a le chiffre **0** au lieu de la lettre **o** → `0ut` devient invalide.

```
Debug3.java :
1 public class Debug3
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Derrière la rivière");
6         System.out.println("et au dela du bois");
7         System.out.println("à la maison du garde nous irons");
8     }
9 }
10
```

Derrière la rivière
et au dela du bois
à la maison du garde nous irons

Explication des corrections

- J'ai uniformisé tous les appels à `System.out.println`.

- J'ai remplacé le **0** par la lettre **o** dans **out**.
- J'ai corrigé les erreurs de casse (**System**, pas **system** ni **SysTem**).

Debug 4)

Voici le programme que nous devons résoudre :

A screenshot of an IDE window titled 'Debug4.java'. The code is as follows:

```
1 import javax.swing.JOptionPane;
2 public class Debug4
3 {
4     public static main(String[] args)
5     {
6         JOptionPane.showMessageDialog(null, 1er GUI program)!
7     }
8 }
```

Voici les problèmes que j'ai trouvé :

Signature de la méthode main incorrecte

Il manque le mot-clé **void** → doit être :

```
public static void main(String[] args)
```

Chaîne de caractères non mise entre guillemets

- **1er GUI program** n'est pas reconnu comme du texte → il faut l'entourer de **"..."**.

Point d'exclamation ! mal placé

En Java, une instruction se termine par `;`, pas par `!`.

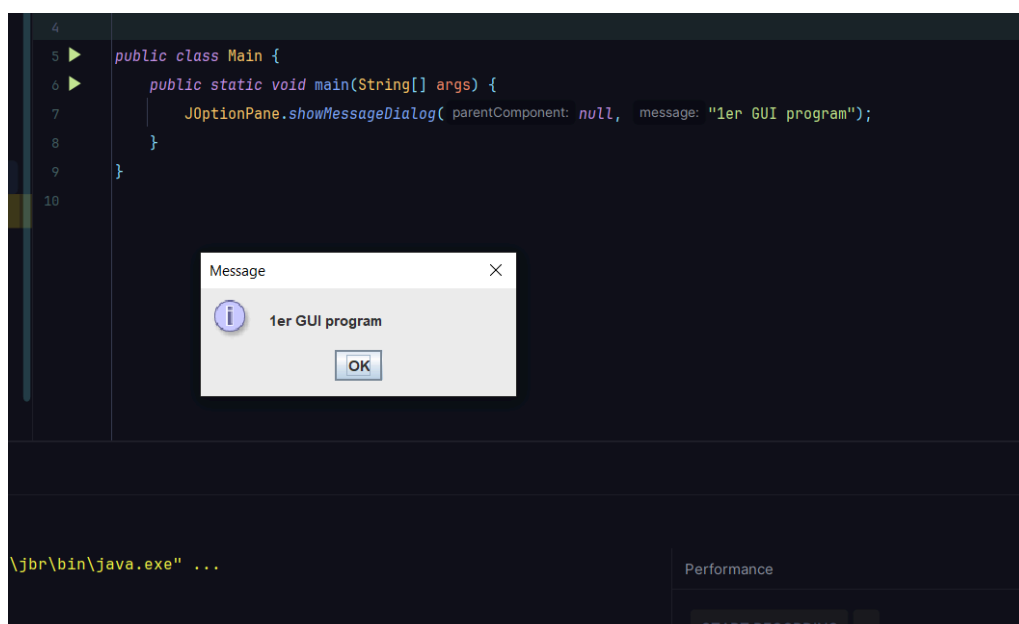
Code corrigé :

```
1 package com.funny;
2 import javax.swing.JOptionPane;
3
4
5 ▶ public class Main {
6 ▶     public static void main(String[] args) {
7         JOptionPane.showMessageDialog( parentComponent: null, message: "1er GUI program");
8     }
9 }
10
```

Explication des corrections

- Ajout de `void` dans la déclaration de `main`.
- Mise en guillemets du texte `"1er GUI program"`.
- Remplacement du `!` par `;` pour terminer correctement l'instruction

Exécution :



4. Conclusion

Ainsi, ce TP m'a permis de progresser dans la compréhension du langage Java en alternant théorie et pratique. J'ai appris à repérer les erreurs les plus fréquentes dans un code Java et à les corriger, ce qui m'a sensibilisé à l'importance de la syntaxe et des conventions. J'ai aussi pu écrire mes premiers programmes personnels.

J'ai apprécié ce travail car il m'a permis de passer de la simple lecture de code à une véritable mise en pratique en corrigeant et en créant mes propres classes. J'y ai consacré environ 3 heures, et je trouve que ce TP constitue une bonne introduction pour préparer les exercices plus avancés de programmation orientée objet en Java.