Fondamentaux du développeur

Séance 5 - TP 2 - Débogage avancé (logique & contexte)

Philippe Roussille

3iL INGENIEURS
</les sciences informatiques>

1 Objectif du TP

Dans ce TP, vous allez :

- Détecter des erreurs de logique.
- Repérer des erreurs de contexte (fichiers, entrées utilisateur, ressources externes).
- Utiliser le débogueur pour analyser les états internes.
- Identifier les causes profondes (pas seulement les symptômes).
- Rédiger un mini-rapport de débogage.

Le projet contient plusieurs zones marquées `// BUG ICI ?`.
Certaines correspondent à de vrais problèmes, d'autres non.

2 Préparation du projet

1. Créez un dossier :

```
mkdir tp-debug-2
cd tp-debug-2
```

2. Initialisez un dépôt Git :

```
git init
```

3. Créez la structure :

```
src/
 ├── Main.java
 ├── Statistics.java
 ├── FileLoader.java
 └── UserInput.java
```

4. Ouvrez VSCode :

```
code .
```

3 Code fourni

Copiez les fichiers ci-dessous dans `src/`.

3.1 Main.java

```java
import java.io.IOException;

public class Main {
    public static void main(String[] args) {
        System.out.println("=== TP Débogage - Partie 2 ===");

        int[] values = {5, 15, 25, 35, 45};

        Statistics stats = new Statistics();
        FileLoader loader = new FileLoader();
        UserInput ui = new UserInput();

        System.out.println("Moyenne : " + stats.average(values)); // BUG
        ↪ ICI ?
        System.out.println("Maximum : " + stats.max(values));     // BUG
        ↪ ICI ?
        System.out.println("Minimum : " + stats.min(values));     // BUG
        ↪ ICI ?

        System.out.println("--- Lecture de fichier ---");

        try {
            String data = loader.load("data.txt"); // BUG ICI ?
            System.out.println(data);
        } catch (IOException e) {
            System.out.println("Erreur lors de la lecture : " +
            ↪ e.getMessage());
        }

        System.out.println("--- Entrée utilisateur ---");
        String name = ui.askName(); // BUG ICI ?
        System.out.println("Bonjour " + name + " !");

        System.out.println("Programme terminé.");
    }
}
```

3.2 Statistics.java

```java
public class Statistics {
```

```java
    public double average(int[] nums) {
        int sum = 0;
        for (int i = 0; i <= nums.length; i++) { // BUG ICI ?
            sum += nums[i];
        }
        return sum / nums.length; // BUG ICI ?
    }

    public int max(int[] nums) {
        int m = nums[0];
        for (int n : nums) {
            if (n > m) {
                m = n; // BUG ICI ?
            }
        }
        return m;
    }

    public int min(int[] nums) {
        int m = nums[0];
        for (int n : nums) {
            if (n < m) {
                m = n; // BUG ICI ?
            }
        }
        return m;
    }
}
```

3.3 FileLoader.java

```java
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;

public class FileLoader {

    public String load(String path) throws IOException {
        File f = new File(path); // BUG ICI ?

        if (!f.exists()) {
            throw new IOException("Fichier manquant");
        }

        byte[] bytes = Files.readAllBytes(f.toPath()); // BUG ICI ?
        return new String(bytes);
    }
}
```

3.4 UserInput.java

```java
import java.util.Scanner;

public class UserInput {

    public String askName() {
        Scanner sc = new Scanner(System.in); // BUG ICI ?

        System.out.println("Entrez votre nom :");
        String name = sc.nextLine(); // BUG ICI ?

        // BUG ICI ?
        return name;
    }
}
```

4 Travaux demandés

4.1 Étape 1 : Exécution initiale

- Lancez le programme.
- Notez les symptômes :
  - exceptions ?
  - sorties incohérentes ?
  - comportements étranges ?

4.2 Étape 2 : Recherche des erreurs de logique

- Placez des breakpoints dans `Statistics.java`.
- Exécutez pas-à-pas pour observer :
  - valeurs intermédiaires,
  - indices,
  - divisions,
  - bornes de boucles.

4.3 Étape 3 : Recherche des erreurs de contexte

- Testez le programme **avec** et **sans** fichier `data.txt`.
- Placez un breakpoint dans `FileLoader.load`.
- Analysez :
  - l'état de `f`,
  - son chemin absolu,
  - le contenu du fichier,
  - les exceptions.

4.4 Étape 4 : Entrées utilisateur

- Placez un breakpoint dans `askName`.

— Observez :

  — la gestion du scanner,
  — l'état du buffer,
  — la valeur réellement reçue.

4.5 Étape 5 : (Mini) Rapport à rendre

Pour chaque fichier :

1. Décrire les comportements observés.
2. Identifier les zones problématiques.
3. Expliquer comment le débogueur a permis l'analyse.
4. Proposer des corrections (sans les appliquer dans vos commits).