

## SET B Debugging

```
1. public class Counter {  
  
    private int count = 0;  
  
    public void increment() {  
  
        count++;  
  
    }  
  
    public int getCount() {  
  
        return count;  
  
    }  
}  
  
public class Test {  
  
    public static void main(String[] args) {  
  
        Counter counter = new Counter();  
  
  
  
        while (counter.getCount() < 10) {  
  
            counter.increment();  
  
        }  
  
        System.out.println("Counter reached: " + counter.getCount());  
  
    }  
}
```

❑ Issue: Static field not retaining value across instances.

❑ Solution: Check singleton implementation for proper instance handling.



```
class Counter {  
    private static int count = 0;  
    public void increment() {  
        count++;  
    }  
    public int getCount() {  
        return count;  
    }  
}  
  
public class Test {  
    public static void main(String[] args) {  
        Counter counter = new Counter();  
        while (counter.getCount() < 10) {  
            counter.increment();  
        }  
        System.out.println("Counter reached: " + counter.getCount());  
    }  
}
```

```
java -cp /tmp/IQ0p1KZkq/test  
Counter reached: 10  
  
=== Code Execution Successful ===
```

```
2. public class Employee {  
  
    private String name;  
  
    public Employee(String name) {
```

```

this.name = name;
}

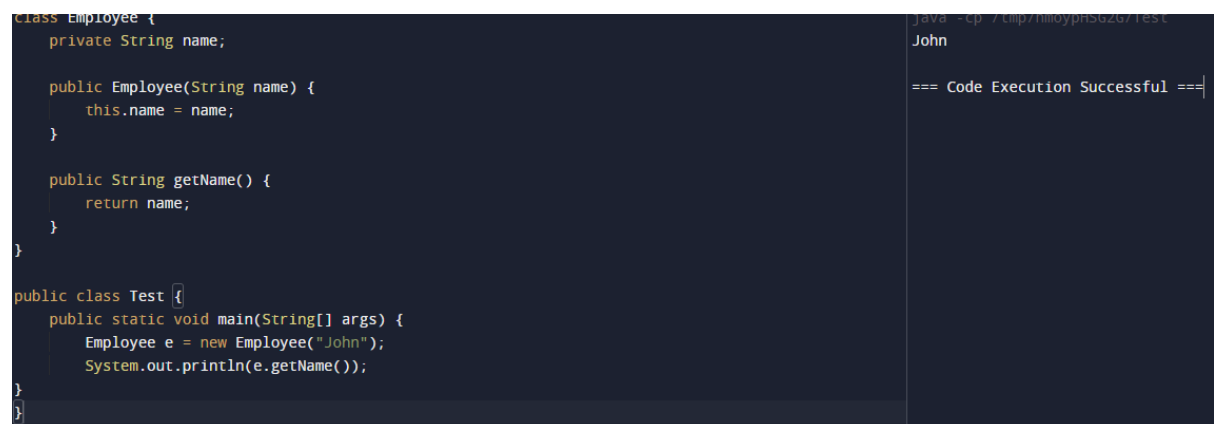
public String getName() {
return name;
}
}

public class Test {
public static void main(String[] args) {
Employee e = new Employee("John");
System.out.println(e.name); // Compilation error
}
}

```

❗ Issue: Direct access to private field name.

❗ Solution: Use getter method getName() to access private fields.



```

class Employee {
    private String name;

    public Employee(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}

public class Test {
    public static void main(String[] args) {
        Employee e = new Employee("John");
        System.out.println(e.getName());
    }
}

```

Terminal Output:

```

java -cp ./tmp/idea/psu20/test
John

=== Code Execution Successful ===

```

3. Question: Why is the FileNotFoundException not being caught when trying to open a file?

❗ Potential Issue: Make sure the FileInputStream or FileReader is enclosed in a try-catch block.

```

public class FileOpener {
public void openFile(String filePath) {
try {
FileReader fileReader = new FileReader(filePath);
BufferedReader br = new BufferedReader(fileReader);
String line;

```

```

while ((line = br.readLine()) != null) {

System.out.println(line);

}

br.close();

} catch (FileNotFoundException e) {

System.out.println("&quot;File not found: &quot; + filePath);

} catch (IOException e) {

e.printStackTrace();

}

}

}

```

```

public class TestFileOpener {

public static void main(String[] args) {

FileOpener opener = new FileOpener();

opener.openFile("&quot;missingfile.txt&quot;);

}

}

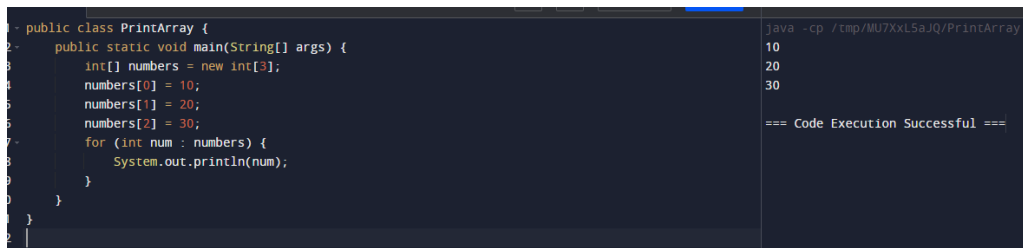
```

<pre> 1- import java.io.*; 2 3- class FileOpener { 4-     public void openFile(String filePath) { 5-         try { 6-             FileReader fileReader = new FileReader(filePath); 7-             BufferedReader br = new BufferedReader(fileReader); 8-             String line; 9-             while ((line = br.readLine()) != null) { 10-                 System.out.println(line); 11-             } 12-             br.close(); 13-         } catch (FileNotFoundException e) { 14-             System.out.println("File not found: " + filePath); 15-         } catch (IOException e) { 16-             e.printStackTrace(); 17-         } 18-     } 19- } 20 21- public class TestFileOpener { 22-     public static void main(String[] args) { 23-         FileOpener opener = new FileOpener(); 24-         opener.openFile("missingfile.txt"); 25-     } 26- } </pre>	<pre> java -cp /tmp/7tFE4V1X1H/TestFileOpener File not found: missingfile.txt  === Code Execution Successful === </pre>
---	---

4. Question: Why is my array not printing the correct values?

🔗 Potential Issue: Ensure the array values are set correctly before printing.

```
public class PrintArray {  
  
    public static void main(String[] args) {  
        int[] numbers = new int[3];  
        numbers[0] = 10;  
        numbers[1] = 20;  
        numbers[2] = 30;  
        for (int num : numbers) {  
            System.out.println(num);  
        }  
    }  
}
```



The screenshot shows a code editor with a dark theme. On the left, the Java code is displayed with line numbers 1 through 12. The code is identical to the one in the first block. On the right, the command prompt shows the command `java -cp /tmp/MU7XxL5aJQ/PrintArray` being executed. The output consists of the numbers 10, 20, and 30, each on a new line. Below the output, a status message reads `=== Code Execution Successful ===`.

```
1 public class PrintArray {  
2     public static void main(String[] args) {  
3         int[] numbers = new int[3];  
4         numbers[0] = 10;  
5         numbers[1] = 20;  
6         numbers[2] = 30;  
7         for (int num : numbers) {  
8             System.out.println(num);  
9         }  
10    }  
11 }  
12 |  
  
java -cp /tmp/MU7XxL5aJQ/PrintArray  
10  
20  
30  
  
=== Code Execution Successful ===
```