

11.1-What is an Exception

What is an Exception in Java?

An **exception** in Java is an event that disrupts the normal flow of a program. Exceptions occur during the execution of a program (runtime) and can cause the program to terminate unexpectedly unless properly handled. Exception handling allows a program to handle errors gracefully and continue its execution.

Errors in Java

In Java, an **Error** signifies a serious issue, indicating that a normal Java application should not attempt to handle it. Errors are often associated with abnormal conditions, such as hardware failures or critical system-level issues, which are beyond the control of the program.

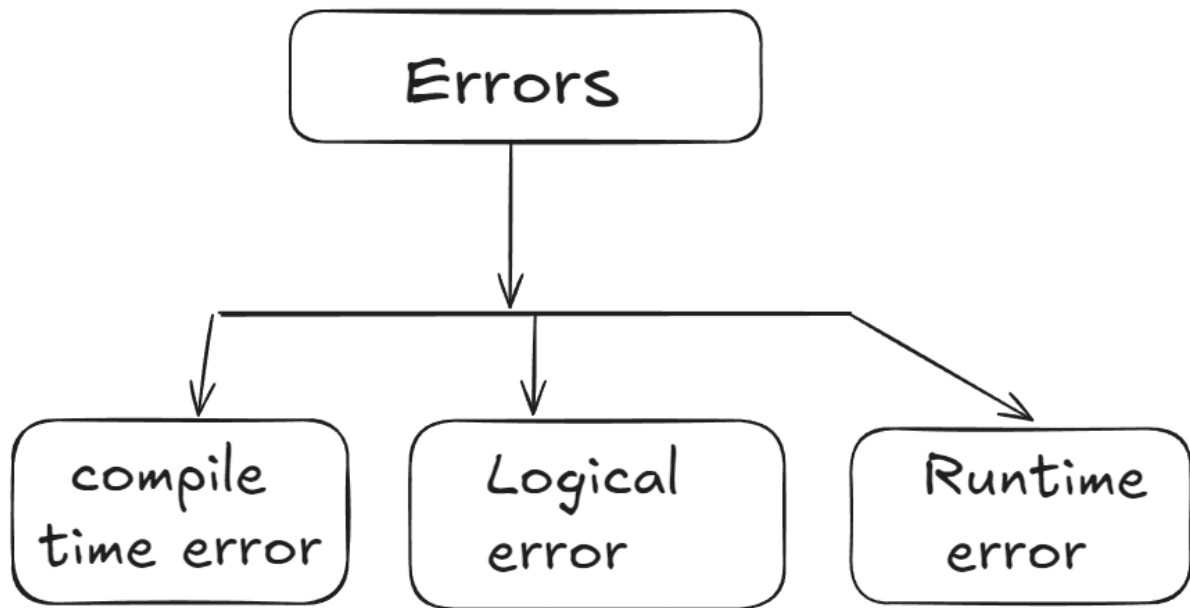
Unlike exceptions, errors cannot be resolved or handled through normal programming techniques. When an error occurs, it typically causes the program to terminate because errors represent conditions that the application should not try to recover from.

Types of Errors

In programming, there are three major types of errors that developers encounter:

1. **Compile-time errors**
2. **Logical errors**

3. Runtime errors (exceptions)



1. Compile-time Errors

Compile-time errors are errors that are detected by the compiler when a program is being compiled. These errors occur due to incorrect syntax, missing files, or other issues that prevent the program from being compiled.

Example:

```
public class Demo {  
    public static void main(String[] args)  
    {  
        System.out.println("Hello World");  
    }  
}
```

Error: In this code, there is a spelling mistake in `System.out.println`, where `println` should be `println`.

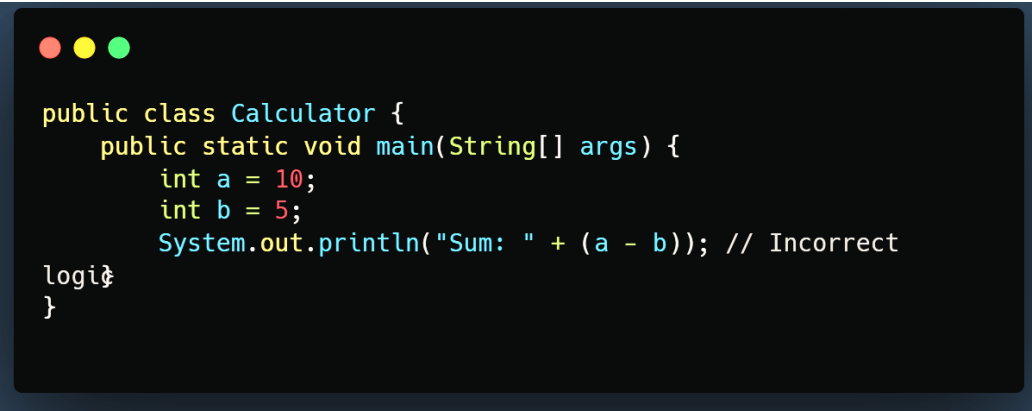
Explanation:

- The error is detected during compilation because of incorrect syntax (misspelled method name).
 - After fixing the error, the program will compile and run successfully.
-

2. Logical Errors

Logical errors are errors that occur when the program runs successfully, but the output is not what was expected. These errors do not cause compilation or runtime issues, but they cause incorrect results.

Example:



```
public class Calculator {  
    public static void main(String[] args) {  
        int a = 10;  
        int b = 5;  
        System.out.println("Sum: " + (a - b)); // Incorrect  
    }  
}
```

Error: Here, the intention is to add two numbers ($a + b$), but instead, the code performs a subtraction ($a - b$), which results in incorrect output.


Explanation:

- Logical errors occur due to a flaw in the algorithm or logic of the program.
 - These errors are harder to detect because they don't stop the execution but produce incorrect results.
 - To fix these errors, testing and debugging are required to identify the problem in the logic.
-

3. Runtime Errors (Exceptions)

Runtime errors occur while the program is running. These errors typically happen when something unexpected occurs, such as trying to divide by zero, accessing an invalid index in an array, or working with unavailable resources (like files). These errors are also called **exceptions**.

Example: Suppose you write a program to open and read a file from your system:



```
import java.io.*;
public class FileReaderDemo {
    public static void main(String[] args) throws IOException
    {
        FileReader fr = new FileReader("nonexistentfile.txt");
        BufferedReader br = new BufferedReader(fr);
        System.out.println(br.readLine());
    }
}
```

Error:

- If the file nonexistentfile.txt does not exist, the program will throw a **FileNotFoundException** at runtime, causing the program to terminate abruptly.

Explanation:

- Runtime errors (exceptions) occur after the program has compiled successfully but face an issue while running.
- Such errors could be critical if they happen in a real-world application (e.g., trying to access an important or confidential file).
- **Exception Handling** is the mechanism that allows the program to handle these errors gracefully without crashing.

Handling Runtime Errors (Exception Handling)

In real-world applications, it's crucial that runtime errors are managed appropriately. For instance, if a file is missing, the program should not crash but instead notify the user and continue running.

This process of anticipating, detecting, and managing errors is known as **Exception Handling**. By using **try-catch** blocks, we can manage exceptions in a structured way, ensuring that the program doesn't abruptly terminate.

Example of Handling an Exception:

```
import java.io.*;

public class FileReaderDemo {
    public static void main(String[] args) {
        try {
            FileReader fr = new FileReader("nonexistentfile.txt");
            BufferedReader br = new BufferedReader(fr);
            System.out.println(br.readLine());
        } catch (FileNotFoundException e) {
            System.out.println("File not found. Please check the file path.");
        } catch (IOException e) {
            System.out.println("An I/O error occurred.");
        }
    }
}
```

Explanation:

- In this code, we handle the potential `FileNotFoundException` using a `try-catch` block.
- Instead of the program terminating abruptly, the error message is displayed, and the program continues.

Here's a refined version of your notes on **Errors in Java** with improvements in structure, grammar, and clarity. I've also included suggestions on where to add images and provided feedback at the end.

Errors in Java

In Java, an **Error** is a subclass of **Throwable** that signifies a serious issue, indicating that a normal Java application should not attempt to handle it. Errors are often associated with abnormal conditions, such as hardware failures or critical system-level issues, which are beyond the control of the program.

Unlike exceptions, errors cannot be resolved or handled through normal programming techniques. When an error occurs, it typically causes the program

to terminate because errors represent conditions that the application should not try to recover from.

Key Characteristics of Errors

- **Non-Recoverable Issues:** Errors are usually related to the environment in which the application is running, such as system crashes, memory leaks, or hardware failures.
 - **Program Termination:** When an error occurs, it usually results in the termination of the program. Unlike exceptions, errors cannot be caught or resolved using try-catch blocks.
 - **Cannot Be Handled:** Since errors indicate serious problems, Java does not expect developers to handle them. They are beyond the scope of normal recovery strategies.
-

Summary: Key Points on Exceptions

- **Compile-time errors** are detected by the compiler and usually result from syntax issues.
- **Logical errors** occur when the program runs but produces incorrect results due to flawed logic.
- **Runtime errors (exceptions)** are unexpected issues that arise when the program is running, such as accessing unavailable resources.

By using **Exception Handling**, runtime errors can be managed, ensuring that the program continues to function smoothly and avoids abrupt termination.