11.2-Exception handling using try catch

In Java, when we run programs, we expect them to compile and execute smoothly without encountering any problems or exceptions. However, that's not always the case. To ensure our programs handle potential issues, we need a mechanism to manage these situations. This is where **exception handling** comes in.

Types of Statements in a Program

In a Java program, we deal with two types of statements:

1. **Critical Statements**
2. **Normal Statements**

1. Critical Statements (Risky Code)

- These are the blocks or lines of code that may potentially cause exceptions.
- We refer to these statements as ***risky code*** because they are prone to fail under certain conditions (e.g., dividing by zero, accessing a null object).

2. Normal Statements

- These are the parts of the program that execute successfully without any issues.
- These statements compile and run fine without triggering any exceptions.

Real-Life Example

Think of a scenario where you're holding the hand of a child in a park while the child is walking & playing around. This is a **normal situation** because it's a controlled environment with no real danger.
However, when you're walking on a busy road, and the child tries to run towards the traffic, this becomes a **critical situation**. You need to hold the child's hand tightly to prevent any accidents.

In programming terms, handling risky code (similar to walking on the road) is essential to avoid unexpected errors that might cause the program to stop.

Example of Exception in Java

Let’s look at a simple example that demonstrates an exception.

```
public class Demo {
    public static void main(String[] args) {
        int i = 0;  // Changed value of i to 0, which will cause an error
        int j = 18 / i;  // Risky statement, causes an ArithmeticException
        System.out.println(j);  // Unreachable code
    }
}
```

Output:

```
An ArithmeticException occurs with the message: / by 0
```

.

When an exception occurs (in this case, dividing by zero), the program’s execution halts at that point, and the subsequent lines of code become **unreachable**. This is why handling exceptions is crucial for smooth program flow.

Handling Exceptions

To handle exceptions in Java, we use a mechanism that separates risky code into a special block. This allows us to handle potential errors without stopping the program.

1. The Try Block

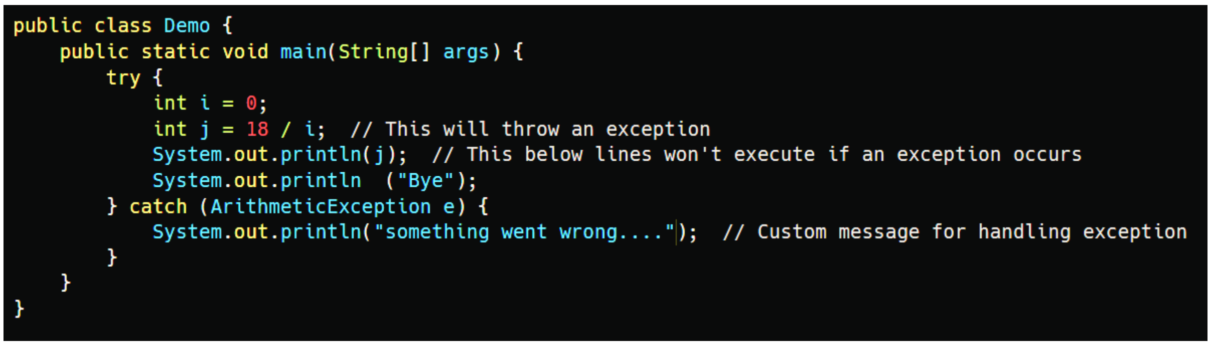
- A **try block** is used to wrap the risky or critical code.
- If an exception occurs within this block, it is caught and handled by the corresponding **catch block**.
- If no exception occurs, the program continues executing normally, skipping the catch block.

2. The Catch Block

- A **catch block** is used to handle the exceptions thrown by the try block.
- The catch block contains code that will execute if an exception occurs.

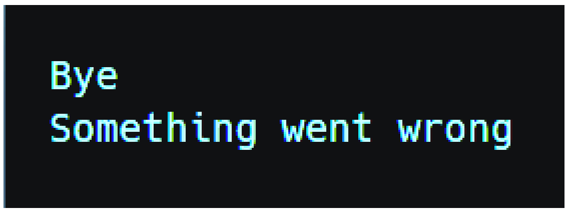
- This block accepts an object of the Exception class, allowing us to customize the message or actions taken when an exception is caught.

Example of Try-Catch Block in Java



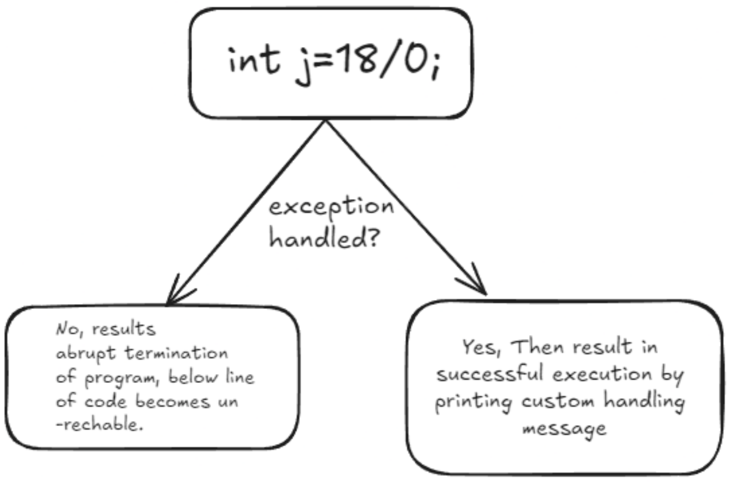
```java
public class Demo {
    public static void main(String[] args) {
        try {
            int i = 0;
            int j = 18 / i;  // This will throw an exception
            System.out.println(j);  // This below lines won't execute if an exception occurs
            System.out.println  ("Bye");
        } catch (ArithmeticException e) {
            System.out.println("something went wrong....");  // Custom message for handling exception
        }
    }
}
```

Output:



```
Bye
Something went wrong
```

How Try-Catch Works:


int j=18/0;
exception handled?
No, results abrupt termination of program, below line of code becomes un -rechable.
Yes, Then result in successful execution by printing custom handling message

- If the code in the try block executes without any issues, the catch block is **skipped**.
- If an exception is encountered, the control jumps to the catch block, which handles the exception, preventing the program from crashing.