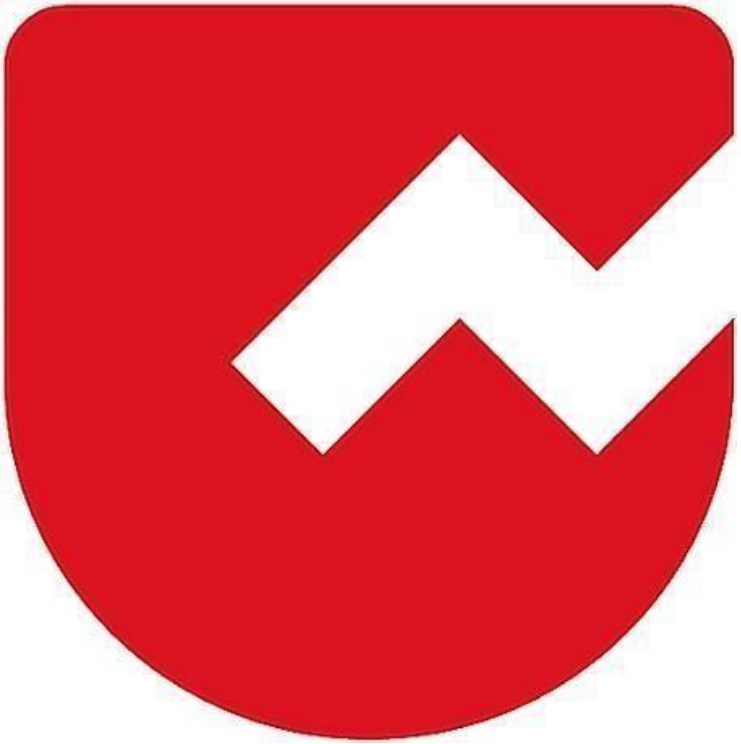
Zoho Schools for Graduate Studies



Notes

DAY – 9 (29-07-2025)

Session -1

Exception Handling

Definition:

Exception Handling in Java is a mechanism to handle runtime errors, so the normal flow of the application can be maintained.

- **The type of the error:** The class name of the exception object itself (e.g., ArithmeticException).
- **A descriptive message:** A string detailing the specific cause of the error (e.g., "/ by zero").
- **The program state:** The execution stack trace, which is a snapshot of the sequence of method calls that led to the point where the error occurred.

Why Do We Need Exception Handling?

Real-Life Analogy:

Imagine you're withdrawing money from an ATM: - Normal flow: You insert your card, enter PIN, and withdraw money. - Unexpected issue: The machine says "Insufficient Balance".

This is like an exception — something went wrong, but instead of the machine crashing, it shows a message and continues working.

Primary Advantages:

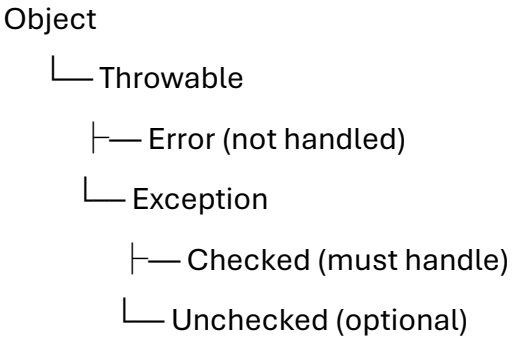
Maintain flow of execution.

Separating Error-Handling Code from "Regular" Code

Types of Errors

Type	Example	Handled By
Compile-time	Syntax errors, missing semicolons	Compiler
Runtime Exception	Divide by zero, null access	Try-Catch
Logical Error	Incorrect formula for calculation	Manually

Java Exception Hierarchy



Common Exception Types

Exception	Description
ArithmeticException	Divide by zero
NullPointerException	Using null object
ArrayIndexOutOfBounds	Invalid array index
IOException	Input/Output failure
FileNotFoundException	File not found on disk

Syntax of Try-Catch Block

```
try {
    // risky code
} catch (ExceptionType name) {
    // handling code
}
```

Optional Blocks:

- finally { } — always executes.
- throw — manually throw exception.
- throws — declare exception in method.

Step-by-Step with Real-Life Examples

Step 1: Basic Try-Catch (ATM Example)

```
public class ATM {
    public static void main(String[] args) {
        int balance = 1000;
        int withdraw = 1200;
        try {
            if (withdraw > balance) {
                throw new ArithmeticException("Insufficient Balance");
            } else {
                balance -= withdraw;
                System.out.println("Withdrawn: " + withdraw);
            }
        } catch (ArithmeticException e) {
            System.out.println("Exception: " + e.getMessage());
        }
        System.out.println("Transaction ends.");
    }
}
```

Step 2: Multiple Catch Blocks (Parcel Delivery)

```
public class Delivery
 {
        public static void main(String[] args) {
          String address = null;
          int[] parcels = new int[2];

          try {
            System.out.println(address.length());
            parcels[5] = 10;
          } catch (NullPointerException e) {
            System.out.println("Missing address: " + e);
          } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Parcel limit exceeded: " + e);
          }
        }
      }
```

Step 3: Finally Block (Train Reservation)

```
public class TrainReservation {
      public static void main(String[] args) {
        try {
          int tickets = 5 / 0;
        } catch (ArithmeticException e) {
          System.out.println("Booking failed: " + e);
        } finally {
          System.out.println("Connection closed.");
        }
      }
    }
```

Step 4: Throw and Throws (Bank Loan Check)

```
class Loan {
    static void checkEligibility(int age) throws Exception {
      if (age < 18) {
        throw new Exception("Not eligible for loan");
      } else {
        System.out.println("Eligible for loan");
      }
    }

    public static void main(String[] args) {
      try {
        checkEligibility(16);
      } catch (Exception e) {
        System.out.println("Exception: " + e.getMessage());
      }
    }
  }
```

Step 5: Custom Exception (Library Fine)

```
class FineException extends Exception {
    public FineException(String message) {
      super(message);
    }
  }

  public class Library {
    static void returnBook(int daysLate) throws FineException {
      if (daysLate > 10) {
        throw new FineException("Fine exceeds limit!");
```

```
        } else {
            System.out.println("Book returned successfully.");
        }
    }

    public static void main(String[] args) {
        try {
            returnBook(15);
        } catch (FineException e) {
            System.out.println("Custom Exception: " + e.getMessage());
        }
    }
}
```

Best Practices

- Use specific exceptions in catch blocks.
- Always clean up resources in finally.
- Avoid empty catch blocks.
- Use custom exceptions for domain-specific errors.

Summary

Concept	Used When
try-catch	To catch and handle exceptions
finally	To clean up resources
throw	To manually throw an exception
throws	To declare an exception in method
Custom Exception	To create user-defined exceptions