Unit 4: Exception Handling and Multithreading

4.1 The Exception Hierarchy

Definition

An exception is an **abnormal condition** that occurs during program execution and **interrupts the normal flow**.

Java follows a **class hierarchy** for exceptions.

Hierarchy (Main Levels)

- `Object`
- `Throwable`
- `Exception` (checked exceptions)
- `Error` (serious system errors)

Features

- All exceptions are objects
- `Exception` can be handled
- `Error` usually cannot be handled

Example Program

```
class ExceptionDemo {
    public static void main(String[] args) {
        int a = 10 / 0;
        System.out.println(a);
    }
}
```

Output

```
Exception in thread "main" java.lang.ArithmeticException
```

4.2 Exception Handling Fundamentals

Definition

Exception handling is a mechanism to **handle runtime errors** so that the program continues execution.

Advantages

- Prevents abnormal termination
- Maintains normal program flow
- Separates error-handling code

Basic Syntax

```
try {
    // risky code
} catch (Exception e) {
    // handling code
}
```

Program

```
class HandleDemo {
    public static void main(String[] args) {
        try {
            int a = 10 / 0;
        } catch (ArithmeticException e) {
            System.out.println("Error handled");
        }
        System.out.println("Program continues");
    }
}
```

Output

```
Error handled
Program continues
```

4.3 Throwing, Re-throwing and Catching Exceptions

Definition

Throwing: Manually creating an exception using `throw`

Catching: Handling exception using `catch`

Re-throwing: Throwing the same exception again

Features

Used for custom error conditions

Improves program reliability

Program

```
class ThrowDemo {
    static void check(int age) {
        if (age < 18) {
            throw new ArithmeticException("Not eligible");
        } else {
            System.out.println("Eligible");
        }
    }

    public static void main(String[] args) {
        try {
            check(15);
        } catch (ArithmeticException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Output

```
Not eligible
```

4.4 try, catch, throw, throws, and finally Keywords

Definitions

`try` → contains risky code

`catch` → handles exception

`throw` → explicitly throws exception

`throws` → declares exception

`finally` → always executes

Syntax

```
try {
} catch (Exception e) {
} finally {
}
```

Program

```
class FinallyDemo {
    static void divide() throws ArithmeticException {
        int a = 10 / 0;
    }

    public static void main(String[] args) {
        try {
            divide();
        } catch (Exception e) {
            System.out.println("Exception caught");
        } finally {
            System.out.println("Finally block executed");
        }
    }
}
```

Output

```
Exception caught
Finally block executed
```

4.5 Multithreading Fundamentals

Definition

Multithreading is the ability of a program to **run multiple threads simultaneously**.

Features

- Improves CPU utilization
- Faster execution
- Each thread runs independently

Thread Life Cycle

- New
- Runnable
- Running
- Blocked
- Terminated

Example Program

```
class MyThread extends Thread {
    public void run() {
        System.out.println("Thread is running");
    }

    public static void main(String[] args) {
        MyThread t = new MyThread();
        t.start();
    }
}
```

Output

```
Thread is running
```

4.6 Thread Class and Runnable Interface

Definition

Java provides **two ways to create threads**:

- Extending `Thread` class
- Implementing `Runnable` interface

Comparison

- `Thread` → cannot extend another class
- `Runnable` → supports multiple inheritance

Using Thread Class

```
class ThreadDemo extends Thread {
    public void run() {
        System.out.println("Thread using Thread class");
```

```
    }

    public static void main(String[] args) {
        ThreadDemo t = new ThreadDemo();
        t.start();
    }
}
```

Output

```
Thread using Thread class
```

Using Runnable Interface

```
class RunnableDemo implements Runnable {
    public void run() {
        System.out.println("Thread using Runnable");
    }

    public static void main(String[] args) {
        RunnableDemo r = new RunnableDemo();
        Thread t = new Thread(r);
        t.start();
    }
}
```

Output

```
Thread using Runnable
```