Exception Handling in Java

Exception

An exception is an unwanted or unexpected event that occurs during the execution of a program, which disrupts the normal flow of instructions.

Exceptions can happen due to various reasons like invalid input, file not found, or trying to divide by zero.

Exception Handling

Exception handling is a mechanism in Java to manage runtime exceptions and maintain the normal flow of the application.

Java provides a way to catch and handle exceptions so that the program doesn't crash.

The most common approach is using try, catch, and finally blocks.

Types of Exceptions in Java

Java exceptions are broadly classified into two categories:

- Checked Exceptions:
  - Occur at compile-time.
  - Must handle them using a try-catch block or by declaring them using throws.
  - Example: IOException, FileNotFoundException

- Unchecked Exceptions:
  - Occur at runtime.
  - Can be handled using try-catch.
  - Example: NullPointerException, ArithmeticException, ArrayIndexOutOfBoundsException

Handling Unchecked Exceptions Using Try-Catch Block

Example:

```
public class UncheckedExceptionDemo {
    public static void main(String[] args) {
        // ArithmeticException
        try {
            int result = 10 / 0;
        } catch (ArithmeticException e) {
            System.out.println("Cannot divide by zero!");
        }

        // NullPointerException
        try {
            String str = null;
            System.out.println(str.length());
        } catch (NullPointerException e) {
            System.out.println("Null pointer exception occurred!");
        }

        // NumberFormatException
        try {
            String num = "abc";
            int value = Integer.parseInt(num);
        } catch (NumberFormatException e) {
            System.out.println("Invalid number format!");
        }

        // ArrayIndexOutOfBoundsException
        try {
            int[] arr = new int[3];
            System.out.println(arr[5]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Array index is out of bounds!");
        }
    }
}
```

Exception Handling in Java

Multiple Catch Blocks

You can have multiple catch blocks to handle different exceptions separately.

Example:

```
public class MultipleCatchDemo {
    public static void main(String[] args) {
        try {
            int[] arr = new int[3];
            System.out.println(arr[5]);
            int result = 10 / 0;
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Array index error!");
        } catch (ArithmeticException e) {
            System.out.println("Arithmetic error!");
        }
    }
}
```

Finally Block

The finally block is always executed, whether an exception occurs or not. It is typically used for cleaning up resources like closing files or releasing memory.

Example:

```
public class FinallyBlockDemo {
    public static void main(String[] args) {
        try {
            int result = 10 / 0;
        } catch (ArithmeticException e) {
            System.out.println("Exception caught!");
        } finally {
            System.out.println("This will always execute!");
        }
    }
}
```

Handling Checked Exceptions Using Try-Catch and Throws

Example:

- Try-Catch:

```
import java.io.*;
public class CheckedExceptionDemo {
    public static void main(String[] args) {
        try {
            FileReader file = new FileReader("nonexistent.txt");
        } catch (FileNotFoundException e) {
            System.out.println("File not found exception!");
        }
    }
}
```

- Throws:

```
import java.io.*;
public class ThrowsExample {
    public static void main(String[] args) throws IOException {
        FileReader file = new FileReader("nonexistent.txt");
    }
}
```

Throw Keyword

The throw keyword is used to manually throw an exception.

Example:

Exception Handling in Java

```
public class ThrowKeyword {
    void findSquare(int num) {
        if (num < 1) {
            throw new ArithmeticException("Number is negative, cannot calculate square.");
        } else {
            System.out.println(num * num);
        }
    }

    public static void main(String[] args) {
        ThrowKeyword tk = new ThrowKeyword();
        try {
            tk.findSquare(-1);
        } catch (ArithmeticException e) {
            System.out.println("Data is not valid.");
        }
    }
}
```

Difference Between Throws and Throw

- Throw:
  - Used to explicitly throw an exception.
  - Example: throw new ArithmeticException("error message");

- Throws:
  - Used in method declarations to specify that a method might throw an exception.
  - Example: public void method() throws IOException { }

Java Exception Keywords Summary

Keyword	Description

Exception Handling in Java

try	Used to define a block of code that might throw an exception. It is followed by either catch, finally, or both.
catch	Used to catch exceptions thrown by the try block. It defines the block of code to be executed if an exception occurs.
finally	A block that always executes after the try block, whether an exception is handled or not. Commonly used for cleanup code.
throw	Used to explicitly throw an exception from a method or block of code. It is followed by an instance of Throwable.
throws	Used in the method signature to declare exceptions that a method might throw. It is followed by the exception type(s).
Throwable	The superclass of all exceptions and errors. All exception classes derive from this class.