Exception Handling

Premanand S

Assistant Professor
School of Electronics Engineering
Vellore Institute of Technology
Chennai Campus

premanand.s@vit.ac.in

March 24, 2025

Introduction to EH

What is an Exception?

- An exception is an unexpected event that occurs during the execution of a program that disrupts the normal flow of instructions.
- It occurs due to programming errors or unforeseen circumstances, preventing the program from executing successfully.

Difference Between Error and Exception

Feature	Error	Exception
Definition	Represents serious system-level issues beyond user control.	Represents problems that can be handled during program execution.
Recoverable?	No, cannot be recovered.	Yes, can be handled using exception handling.
Examples	`OutOfMemoryError`, `StackOverflowError`	`NullPointerException`, `IOException`
When It Occurs	Occurs at runtime due to system failure.	Occurs at runtime due to logical errors.
Handling	Cannot be handled using `try-catch`.	Handled using `try-catch` and `throws`.

Exception - Analogically

- Imagine withdrawing money from an ATM.
  - Receiving Money
  - Not Received Money
    - Insufficient balance
    - Network failure. Try again later.

Exception - Programmically

- Problem Statement
  - Getting Output
  - Getting Errors
    - ArithmeticException.
    - FileNotFoundException.

Why Exception Handling is Required?

Example (Understanding)

- Preventing Program Crashes
- Ensuring Graceful Program Termination
- Improving Fault Tolerance

```
public class Main {
    public static void main(String[] args) {
        int result = 10 / 0;
        System.out.println("Result: " + result);
    }
}
```

Types of Java Errors

- Compile-Time Errors
  - Syntax errors
  - Data type mismatch
  - Missing semicolons or parentheses
- Runtime Errors
  - ArithmeticException
  - ArrayIndexOutOfBoundsException
  - NullPointerException
- Logical Errors
  - Incorrect implementation of algorithms.
  - Wrong conditions in if-else statements.

Types of Exceptions

Checked vs. Unchecked Exceptions

Type	Definition	Examples
Checked Exception	Exceptions checked at **compile-time**. The compiler ensures that these exceptions are either handled using `try-catch` or declared using `throws`.	`IOException`, `SQLException`, `ClassNotFoundException`
Unchecked Exception	Exceptions not checked at **compile-time** but occur at **runtime**. The program may compile successfully but fail during execution.	`ArithmeticException`, `NullPointerException`, `ArrayIndexOutOfBoundsException`

Differences: Checked vs. Unchecked Exceptions

Aspect	**Checked Exception**	**Unchecked Exception**
Compilation	Checked at **compile-time**.	Not checked at **compile-time**.
Handling	Must be handled using `try-catch` or declared using `throws`.	No mandatory handling required.
Occurrence	Usually caused by external factors (e.g., I/O, DB).	Caused by programming errors or logical flaws.
Examples	`IOException`, `SQLException`	`NullPointerException`, `ArithmeticException`
Subclass of	Subclass of `Exception`	Subclass of `RuntimeException`

When to Use Checked and Unchecked Exceptions?

- Use Checked Exceptions:
  - For recoverable errors (e.g., file not found, invalid input).
  - When the program has alternative ways to handle the error.
- Use Unchecked Exceptions:
  - For programming errors (e.g., accessing null objects, invalid array index).
  - When the errors are due to logical mistakes that can be avoided with proper validation.

Checked Exception Example

Example (Understanding)

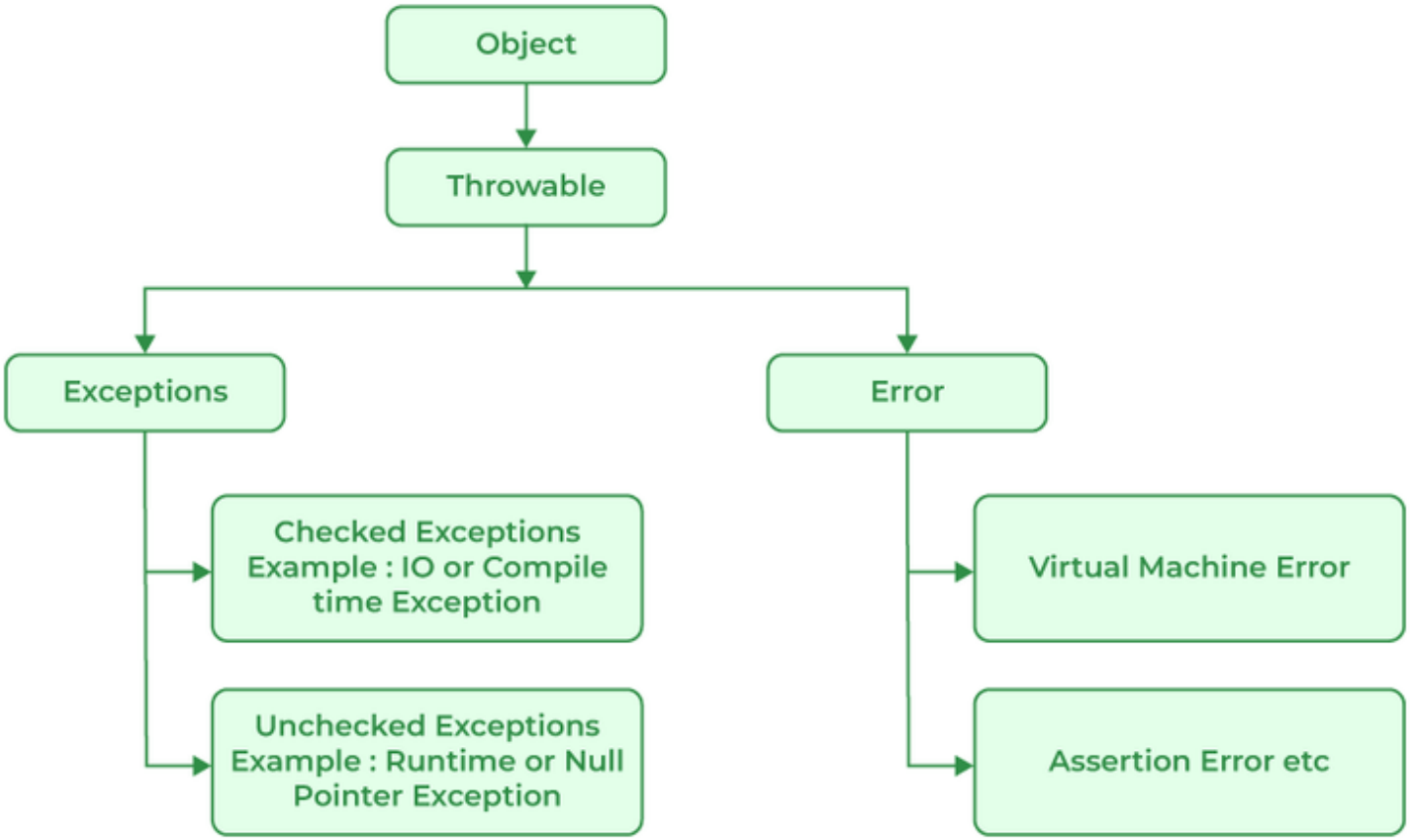
```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        try {
            File file = new File("input.txt");
            Scanner scanner = new Scanner(file);
        } catch (FileNotFoundException e) {
            System.out.println("File not found.
            Please check the file path.");
        }     } }
```

Unchecked Exception Example

Example (Understanding)

```
public class Main {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3};
        System.out.println(arr[5]);
        // ArrayIndexOutOfBoundsException
    }
}
```

Java Exception Hierarchy Diagram



Commonly Used Exceptions

Example (Understanding)

```
# IOException

try {
    FileReader file = new FileReader("test.txt");
} catch (IOException e) {
    e.printStackTrace();
}
```

Commonly Used Exceptions

Example (Understanding)

```
# SQLException

try {
    Connection conn = DriverManager.getConnection("jdbc:mysql:
    //localhost:3306/db", "user", "pass");
} catch (SQLException e) {
    e.printStackTrace();
}
```

Commonly Used Exceptions

Example (Understanding)

```
# ArithmeticException

int result = 10 / 0;  // ArithmeticException
```

Commonly Used Exceptions

Example (Understanding)

```
# NullPointerException

String str = null;
System.out.println(str.length());  // NullPointerException
```

Commonly Used Exceptions

Example (Understanding)

```
# ArrayIndexOutOfBoundsException

int[] arr = {1, 2, 3};
System.out.println(arr[5]);  // ArrayIndexOutOfBoundsExceptio
```

Commonly Used Exceptions

Example (Understanding)

```
# ClassNotFoundException

try {
    Class.forName("com.example.MyClass");
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}
```

Control Flow in EH

Normal Control Flow vs. Exception Control Flow

- Normal Control Flow
  - Program executes line-by-line sequentially.
  - No deviation in control flow if no exception occurs.
  - Final statement executes at the end of the method.
- Exception Control Flow
  - If an exception occurs, the normal flow is disrupted.
  - Control shifts immediately to the nearest matching catch block.
  - If no matching catch block is found, the JVM terminates the program.
  - finally block (if present) executes after try-catch regardless of exception occurrence.

Normal Flow Example

Example (Understanding)

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Start of the program");
        int result = 10 / 2;  // No exception
        System.out.println("Result: " + result);
        System.out.println("End of the program");
    }
}
```

Exception Flow Example

Example (Understanding)

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Start of the program");
        try {
            int result = 10 / 0;  // ArithmeticException
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Exception occurred: "
            + e.getMessage());
        }
        System.out.println("End of the program");
    }
}
```

Incorrect Order Example

Example (Understanding)

```
try {
    int num = 10 / 0;
} catch (Exception e) {  // Parent first (incorrect)
    System.out.println("Exception occurred");
} catch (ArithmeticException e) {  // Child after parent
    System.out.println("Arithmetic Exception");
}
```

Example (No Exception Occurs)

Example (Understanding)

```
try {
    int result = 10 / 2;
    System.out.println("Result: " + result);
} finally {
    System.out.println("Finally block executed");
}
```

Example (Unhandled Exception)

Example (Understanding)

```
try {
    int result = 10 / 0;
} finally {
    System.out.println("Finally block executed");
}
```

What is try-catch?

- The try block contains code that may throw an exception.
- The catch block handles the exception if one occurs.
- If no exception occurs, the catch block is skipped.

Syntax

Example (Understanding)

```
try {
    // Code that may throw an exception
} catch (ExceptionType e) {
    // Handling exception
}
```

Single Catch

Example (Understanding)

```
public class Main {
    public static void main(String[] args) {
        try {
            int result = 10 / 0;  // ArithmeticException
        } catch (ArithmeticException e) {
            System.out.println("Caught Exception:
            " + e.getMessage());
        }
    }
}
```

Multiple Catch Blocks

- A try block can have multiple catch blocks to handle different types of exceptions.
- The order of catch blocks is important.
- Parent classes should always come after child classes to prevent unreachable code.
- Note to consider,
  - Place specific exceptions first, followed by general exceptions.
  - Exception (parent class) should be the last catch block.
  - Catching a parent class before its subclass will result in unreachable code.

Example

Example (Understanding)

```
public class Main {
    public static void main(String[] args) {
        try {
            int arr[] = new int[5];
            arr[10] = 50 / 0;  // Multiple exceptions possible
        } catch (ArithmeticException e) {
            System.out.println("Arithmetic Exception: "
            + e.getMessage());
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Array Index Exception: "
            + e.getMessage());
        } catch (Exception e) {
            System.out.println("Parent Exception: "
            + e.getMessage());
        }    } }
```

Multiple catch Blocks

Example (Understanding)

```
public class Main {
    public static void main(String[] args) {
        try {
            int arr[] = new int[5];
            arr[10] = 30 / 0;  // Multiple exceptions possible
        } catch (ArithmeticException e) {
            System.out.println("Arithmetic Exception:
            " + e.getMessage());
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Array Index Exception:
            " + e.getMessage());
        } catch (Exception e) {
            System.out.println("Parent Exception:
            " + e.getMessage());
        }    } }
```

What is finally?

- The finally block always executes after the try and catch blocks.
- It is typically used for cleanup operations such as closing files, releasing resources, etc.

Syntax

Example (Understanding)

```
try {
    // Code that may throw an exception
} catch (ExceptionType e) {
    // Handle exception
} finally {
    // Code that always executes
}
```

Syntax

Example (Understanding)

```
public class Main {
    public static void main(String[] args) {
        try {
            int result = 10 / 0;
        } catch (ArithmeticException e) {
            System.out.println("Exception caught: " +
            e.getMessage());
        } finally {
            System.out.println("Finally block executed");
        }
    }
}
```

Using finally Without catch

Example (Understanding)

```
try {
    int data = 50 / 5;
    System.out.println(data);
} finally {
    System.out.println("Finally block executed");
}
```

throw Keyword

- The throw keyword is used to explicitly throw an exception.
- You can throw both built-in exceptions and user-defined exceptions.

Syntax

Example (Understanding)

```
throw new ExceptionType("Error Message");
```

throw

Example (Understanding)

```
public class Main {
    public static void main(String[] args) {
        int age = 15;
        if (age < 18) {
            throw new ArithmeticException("Age
            must be 18 or above.");
        } else {
            System.out.println("Welcome to
            the voting system.");
        }
    }
}
```

Throwing Custom Exceptions

Example (Understanding)

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

public class Main {
    public static void checkAge(int age)
    throws InvalidAgeException {
        if (age < 18) {
            throw new InvalidAgeException("Access denied
            - Age below 18.");
        } else {
            System.out.println("Access granted.");
        }     }
```

Throwing Custom Exceptions

Example (Understanding)

```
    public static void main(String[] args) {
        try {
            checkAge(16);
        } catch (InvalidAgeException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

throws Keyword

- throws is used to declare exceptions in the method signature.
- It tells the caller that the method may throw an exception.
- Multiple exceptions can be declared using a comma.

Syntax

Example (Understanding)

```
returnType methodName() throws ExceptionType1, ExceptionType2
    // Method code
}
```

Syntax

Example (Understanding)

```
import java.io.*;

public class Main {
    public static void readFile() throws IOException {
        FileReader file = new FileReader("nonexistent.txt");
        BufferedReader br = new BufferedReader(file);
        String line;
        while ((line = br.readLine()) != null) {
            System.out.println(line);
        }
    }
```

Syntax

Example (Understanding)

```
    public static void main(String[] args) {
        try {
            readFile();
        } catch (IOException e) {
            System.out.println("File not found: "
            + e.getMessage());
        }
    }
}
```

throw vs. throws in Java

Aspect	**throw**	**throws**
Purpose	Used to explicitly throw an exception.	Declares exceptions that may be thrown by the method.
Placement	Inside the method body.	In the method signature.
Type	Can throw only one exception at a time.	Can declare multiple exceptions.
Syntax	`throw new Exception()`	`throws IOException, SQLException`
Example	`throw new ArithmeticException("Error")`	`void method() throws IOException`