INTRODUCTION TO PROGRAMMING LANGUAGE II(JAVA)

Fariha Zahin

Motivation: Why Exception Handling Matters?

- Programs often face abnormal situations during execution
- Common causes include:
  - Invalid user input
  - Missing files
  - Network failures
  - Division by zero

Motivation: Why Exception Handling Matters?

- Without handling, programs may terminate abruptly
- Exception handling improves:
  - Program robustness
  - Maintainability
  - Code readability
  - Separation of normal logic and error-handling logic

What is Exception Handling?

•Exception: An event that disrupts the normal flow of the program.

•Handling: Mechanism to detect and handle runtime problems.

Examples:

- Division by zero
- Accessing invalid array index
- Reading a non-existent file

****Java provides a robust exception handling model using try, catch, throw, throws, and finally.**

Types of Exceptions

1.Checked Exception

1. Checked at compile time.
2. Must be handled using try-catch or declared using throws
3. Program will not compile if ignored
4. Examples: IOException, SQLException

Types of Exceptions

2. Unchecked Exception
    1. Occur at runtime.
    2. Not checked at compile time
    3. Usually caused by programming errors
    4. Examples: NullPointerException, ArithmeticException

Error:

- Serious system-level problems
- Not meant to be handled by applications
- Example: OutOfMemoryError

Main.java

```java
import java.util.List;

Rename usages
public class Main {
    public static void main(String[] args) {
        List<byte[]> list = new ArrayList<>();
        while (true) {
            // Keep adding 1MB arrays to the list until memory is exhausted
            list.add(new byte[1024 * 1024]);
        }
    }
}
```

```
C:\Users\Paran\.jdks\openjdk-25.0.1\bin\java.exe "-javaagent:C:\Program Files\J
Exception in thread "main" java.lang.OutOfMemoryError Create breakpoint : Java heap
    at Main.main(Main.java:9)

Process finished with exit code 1
```

ArithmeticException
Division by zero

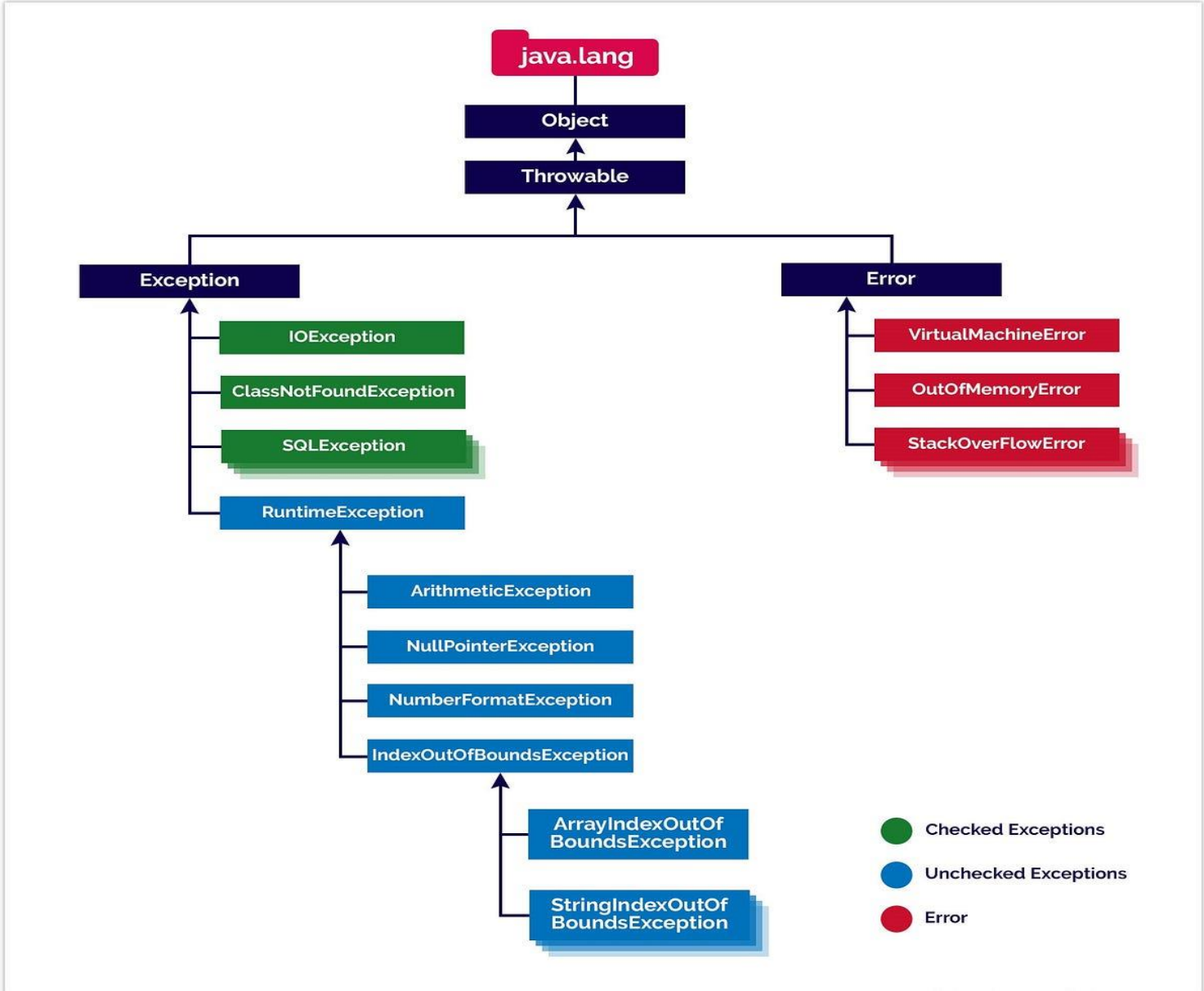
Main.java

```java
public class Main {
    public static void main(String[] args)
    {
        int a = 10;
        int b = 0;
        int c = a/b;
    }
}

```

```
C:\Users\Paran\.jdks\openjdk-25.0.1\bin\java.exe "-javaagent:C:\Program Fi
Exception in thread "main" java.lang.ArithmeticException Create breakpoint : /
    at Main.main(Main.java:6)

Process finished with exit code 1
```


java.lang
Object
Throwable
Exception
Error
IOException
ClassNotFoundException
SQLException
RuntimeException
VirtualMachineError
OutOfMemoryError
StackOverFlowError
ArithmeticException
NullPointerException
NumberFormatException
IndexOutOfBoundsException
ArrayIndexOutOf BoundsException
StringIndexOutOf BoundsException
Checked Exceptions
Unchecked Exceptions
Error

Difference Between Error and Exception

Feature	Error	Exception
Origin	Outside the control of program	Within program's scope
Recovery	Not usually recoverable	Can be handled using code
Examples	OutOfMemoryError, StackOverflowError	IOException, NumberFormatException
Type	Unchecked	Checked or Unchecked
Use of try-catch	Not advised	Strongly recommended

5 Keywords in Exception Handling

1.try – Block to monitor for exceptions
2.catch – Handles the exception
3.throw – Used to explicitly throw an exception
4.throws – Declares exceptions
5.finally – Always executes after try-catch

try block

Used for: Wrapping code that might throw an exception.

Rules: Must be followed by either a catch block, a finally block, or both.

```
try {
    int result = 10 / 0; // This will throw ArithmeticException
    System.out.println("Result: " + result);
}
```

catch block
Used for: Handling exceptions that are thrown inside the try block.
Rules: You can use multiple catch blocks for different exception types.

```
try {
    // Risky code
} catch (ExceptionType e) {
    // Handling code
}
```

Only the first exception that occurs is handled. In this case, the array index error happens first, so the division is never reached.

```
public class MultipleCatchExample {
    public static void main(String[] args) {
        try {
            int[] numbers = new int[3];
            numbers[5] = 10;  // ArrayIndexOutOfBoundsException
            int result = 10 / 0;  // ArithmeticException
        } 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("General Exception: " + e.getMessage());
        }
    }
}
```

Why Child Exception First, Parent Last

•Java matches exceptions from **top to bottom**.

•When the parent class (Exception) comes **before** the child classes, the child catch blocks become **unreachable**, and this results in a **compile-time error**.

•**Correct Order:** Write the more **specific exceptions first**, and the **general one (Exception)** at the end.

Wrong Order (Compile-Time Error)

```
try {
    int a = 10 / 0;
} catch (Exception e) {
    System.out.println("Generic Exception");
} catch (ArithmeticException e) {
    System.out.println("Arithmetic Exception");
}
```

Correct Order (Child First, Parent Last)

```
try {
    int a = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Arithmetic Exception: " + e.getMessage());
} catch (Exception e) {
    System.out.println("Generic Exception: " + e.getMessage());
}
```

finally block

Used for: Code that should always run, whether an exception occurs or not.

Rules: Used for cleanup like closing files or releasing resources.

```
public class Test {
    public static void main(String[] args) {
        try {
            int data = 10 / 0;
        } catch (ArithmeticException e) {
            System.out.println("Caught exception");
        } finally {
            System.out.println("finally block executed");
        }
    }
}
```

throw keyword
Used for: Manually throwing an exception (built-in or custom).
Rules: Only one exception can be thrown at a time using throw.

```
public class Test {
    public static void main(String[] args) {
        int age = 16;
        if (age < 18) {
            throw new ArithmeticException("Age must be 18 or above");
        }
        System.out.println("You are eligible");
    }
}
```

throws keyword
Used for: Declaring exceptions a method might throw.
Rules: Helps inform the caller to handle the exception.

```
public class Test {
    static void divide() throws ArithmeticException {
        int result = 10 / 0;
        System.out.println(result);
    }

    public static void main(String[] args) {
        try {
            divide();
        } catch (ArithmeticException e) {
            System.out.println("Exception handled in main");
        }
    }
}
```

Custom User-Defined Exception in Java

Java allows you to create your own exception classes by extending the built-in Exception class (or any of its subclasses). This helps you represent application-specific errors clearly.

- Custom exceptions usually extend Exception (checked exceptions) or RuntimeException (unchecked exceptions).
- You can add constructors and custom messages.
- When your code detects a specific error condition, you throw your custom exception.
- The caller handles it like any other exception using try-catch.

```java
// Custom exception class
class MyException extends Exception {
    public MyException(String message) {
        super(message);
    }
}
```

```java
public class TestCustomException {
    static void checkNumber(int num) throws MyException {
        if (num < 0) {
            throw new MyException("Number cannot be negative");
        }
        System.out.println("Number is " + num);
    }
```

```
    public static void main(String[] args) {
        try {
            checkNumber(-5);
        } catch (MyException e) {
            System.out.println("Caught Exception: " + e.getMessage());
        }
    }
}
```

This code defines a custom exception class named MyException without any constructor. The method check throws this exception when the input number is less than zero. In the main method, check is called inside a try block. If the exception occurs, it is caught in the catch block, and a simple message is printed. Since the custom exception has no constructor with a message, no detailed error message is available. This shows that defining a constructor is optional but useful for passing error details.