SNS COLLEGE OF TECHNOLOGY

Kurumbapalayam (Po), Coimbatore – 641 107

AN AUTONOMOUS INSTITUTION

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND TECHNOLOGY

23CSB201-OBJECT ORIENTED PROGRAMMING

UNIT IV - EXCEPTION AND MULTITHREADING

TOPIC 1 - Exception handling

1. What is an Exception?

- An **exception** is an abnormal condition that occurs during the execution of a program, disrupting the normal flow of instructions.

Example:

- Dividing a number by zero
- Accessing an invalid array index
- Opening a non-existing file

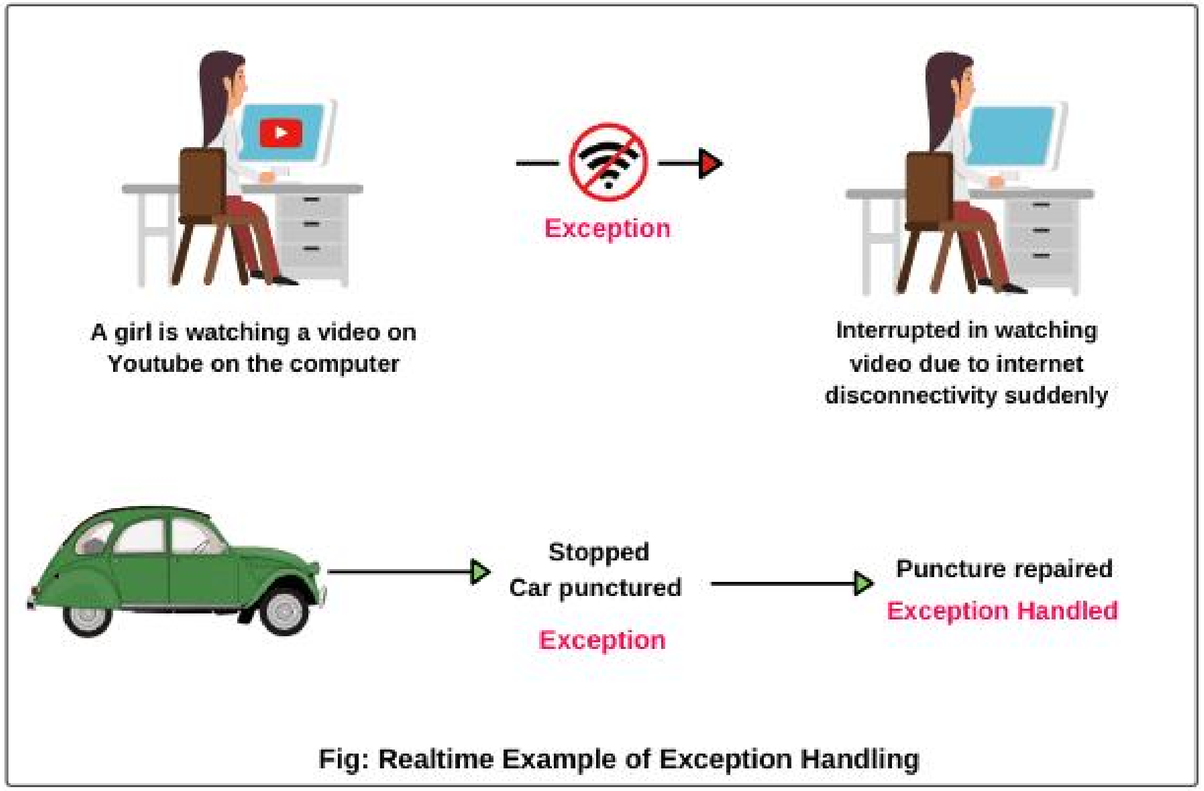
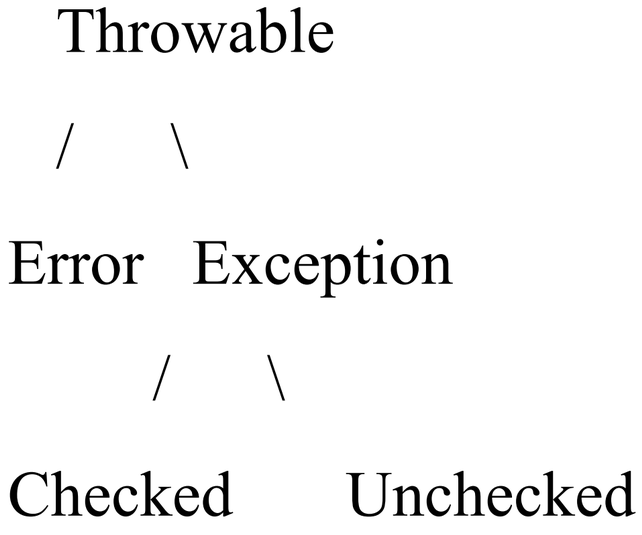


Fig: Realtime Example of Exception Handling

2. What is Exception Handling?

- **Exception handling** is a mechanism in Java used to **handle runtime errors** so that the program can continue execution normally without terminating abruptly.

3. Exception Hierarchy in Java:

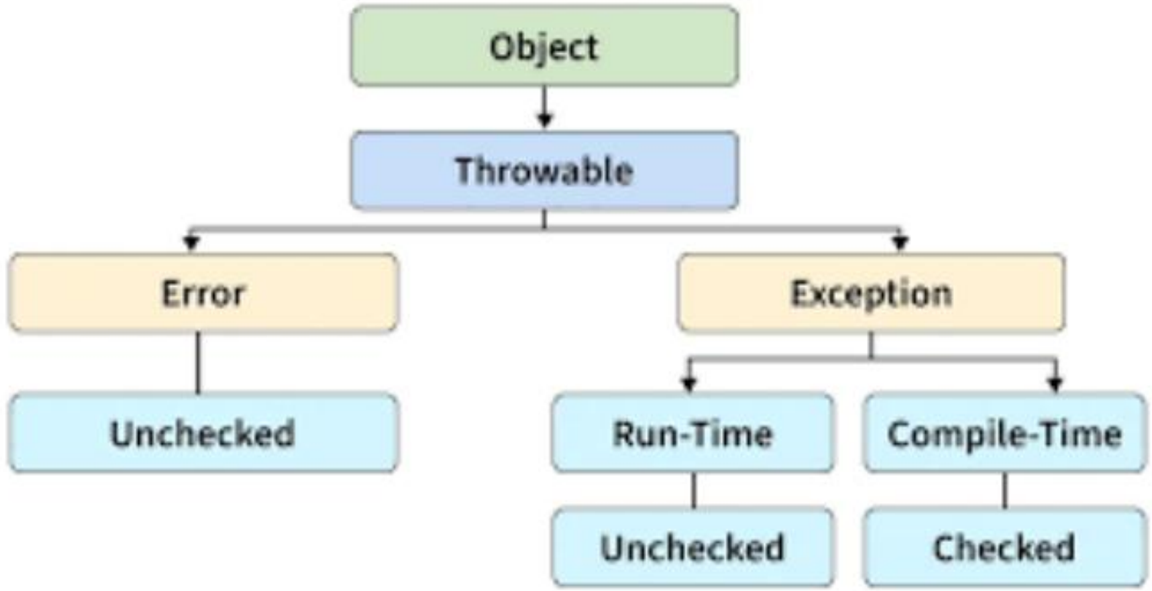


Error:

- Serious issues (not handled by programmer)
- Example: OutOfMemoryError

Exception:

- Can be handled using code



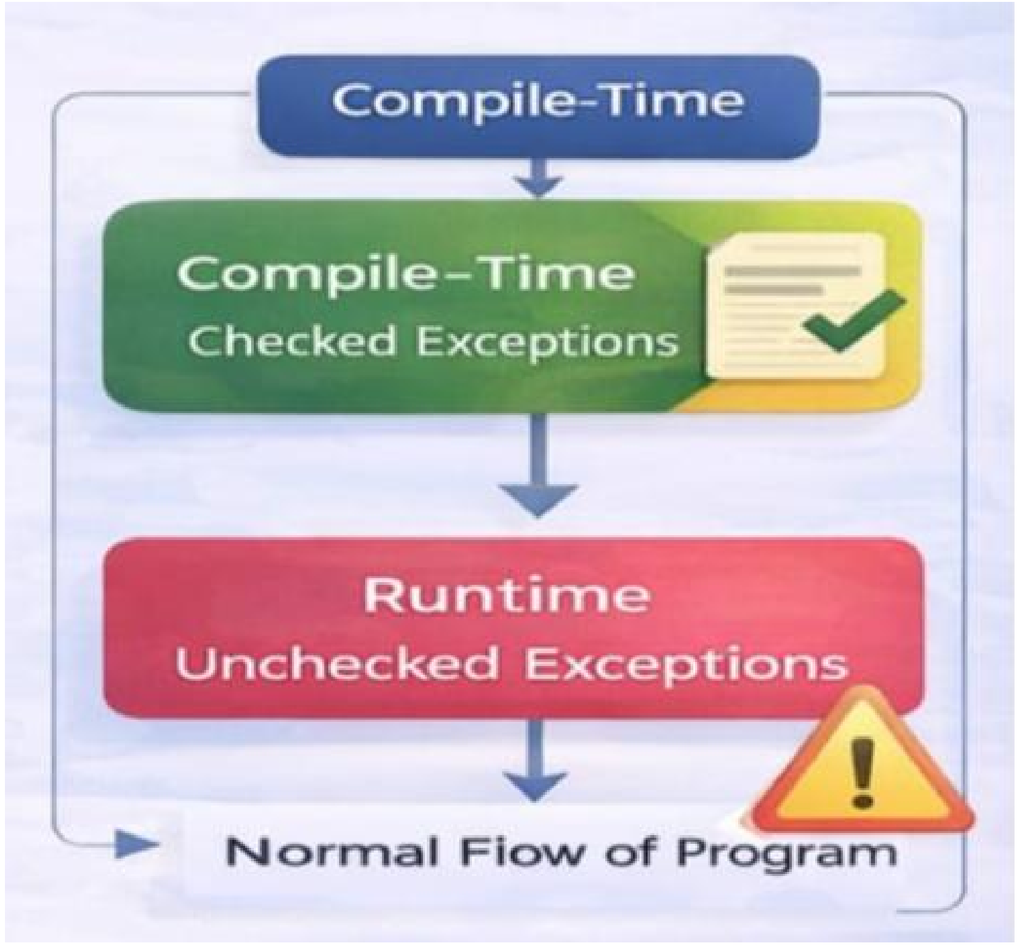
4. Types of Exceptions:

✓ (A) Checked Exceptions (Compile-Time)

- Checked by compiler
- Must be handled

Examples:

- IOException
- SQLException
- FileNotFoundException



Example:

```
import java.io.*;

class Test {
    public static void main(String[] args) {
        try {
            FileReader f = new FileReader("file.txt");
        } catch (IOException e) {
            System.out.println("File not found");
        }
    }
}
```

(B) Unchecked Exceptions (Runtime)

- Occur during execution
- Not checked at compile time

Examples:

- ArithmeticException
- NullPointerException
- ArrayIndexOutOfBoundsException

Example:

```
class Test {
    public static void main(String[] args) {
        int a = 10 / 0; // ArithmeticException
    }
}
```

5. Keywords Used in Exception Handling:

1. Try:

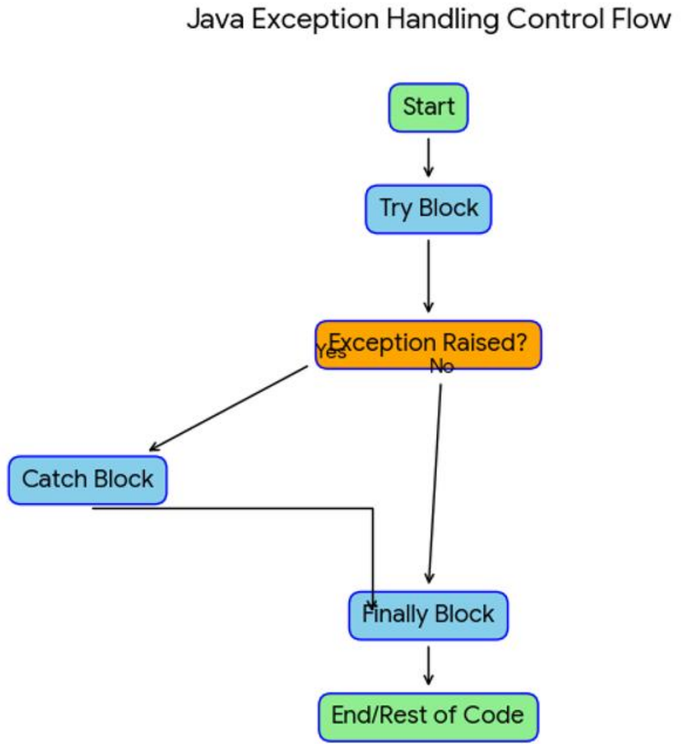
- Contains risky code

2. Catch:

- Handles exception

3. Finally:

- Always executes



4. Throw;

Used to manually throw exception

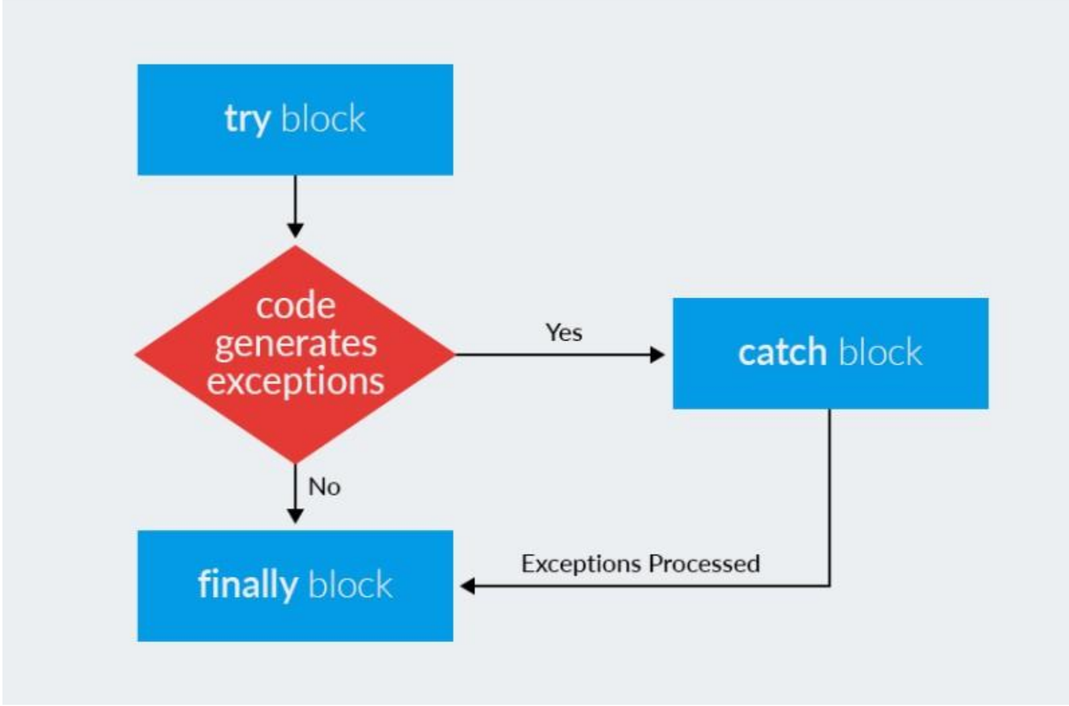
5. Throws:

Declares exception

6. try-catch Block:

Syntax:

```
try {
    // code that may cause exception
} catch (ExceptionType e) {
    // handling code
}
```



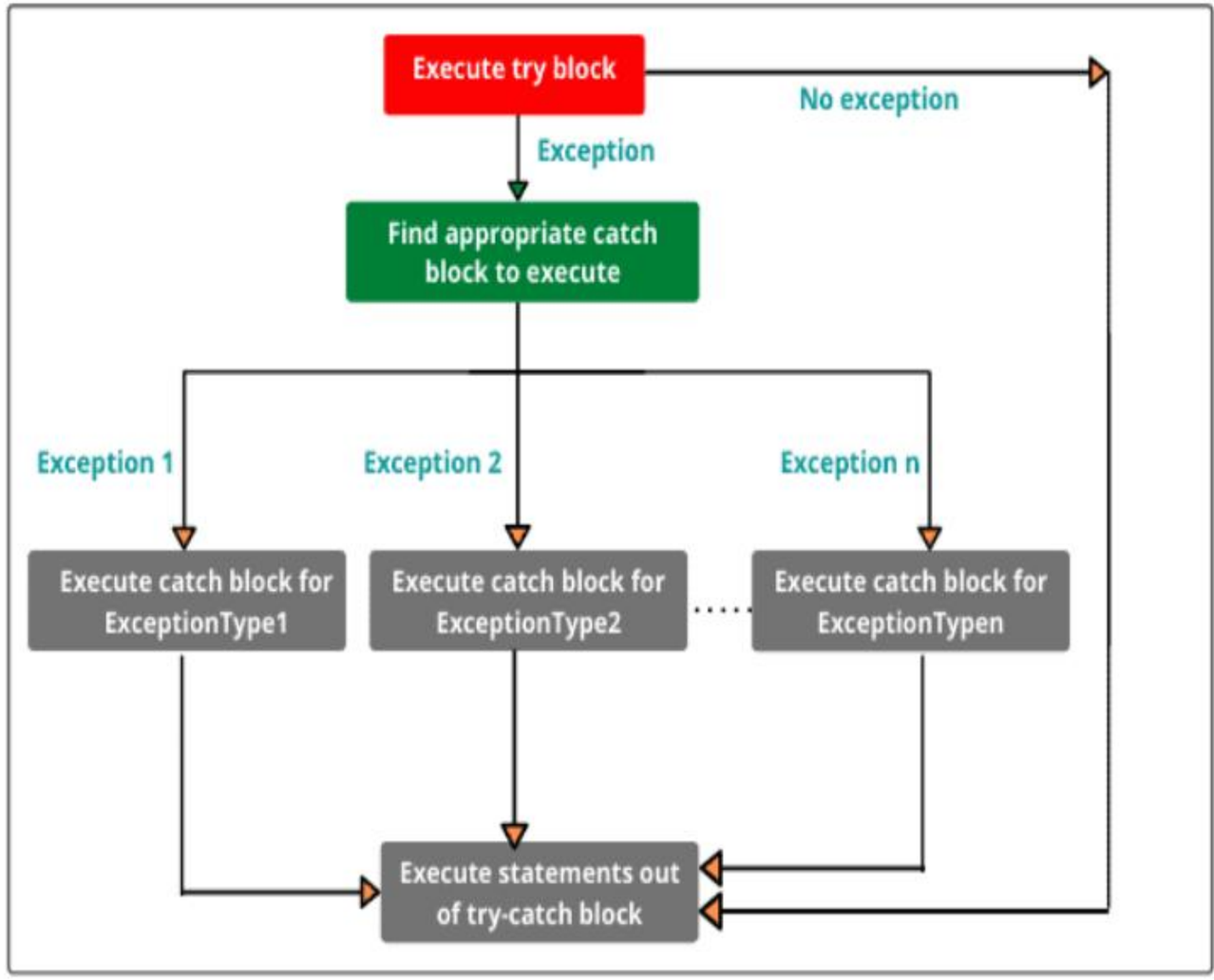
Example:

```
class Demo {
    public static void main(String[] args) {
        try {
            int x = 10 / 0;
        } catch (ArithmeticException e) {
            System.out.println("Division by zero not allowed");
        }
        System.out.println("Program continues...");
    }
}
```

7. Multiple Catch Blocks:

- Used to handle different exceptions separately

```
class Test {
    public static void main(String[] args) {
        try {
            int arr[] = new int[5];
            arr[10] = 100;
        } catch (ArithmeticException e) {
            System.out.println("Arithmetic Error");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Array Index Error");
        }
    }
}
```



8. finally Block:

- Executes whether exception occurs or not

```
class Test {
    public static void main(String[] args) {
        try {
            int x = 5 / 1;
        } catch (Exception e) {
            System.out.println("Error");
        } finally {
            System.out.println("Always executed");
        }
    }
}
```

9. throw Keyword:

- Used to explicitly throw an exception

```
class Test {
    static void check(int age) {
        if (age < 18) {
            throw new ArithmeticException("Not eligible");
        } else {
            System.out.println("Eligible");
        }
    }
    public static void main(String[] args) {
        check(16);
    }
}
```

10. throws Keyword:

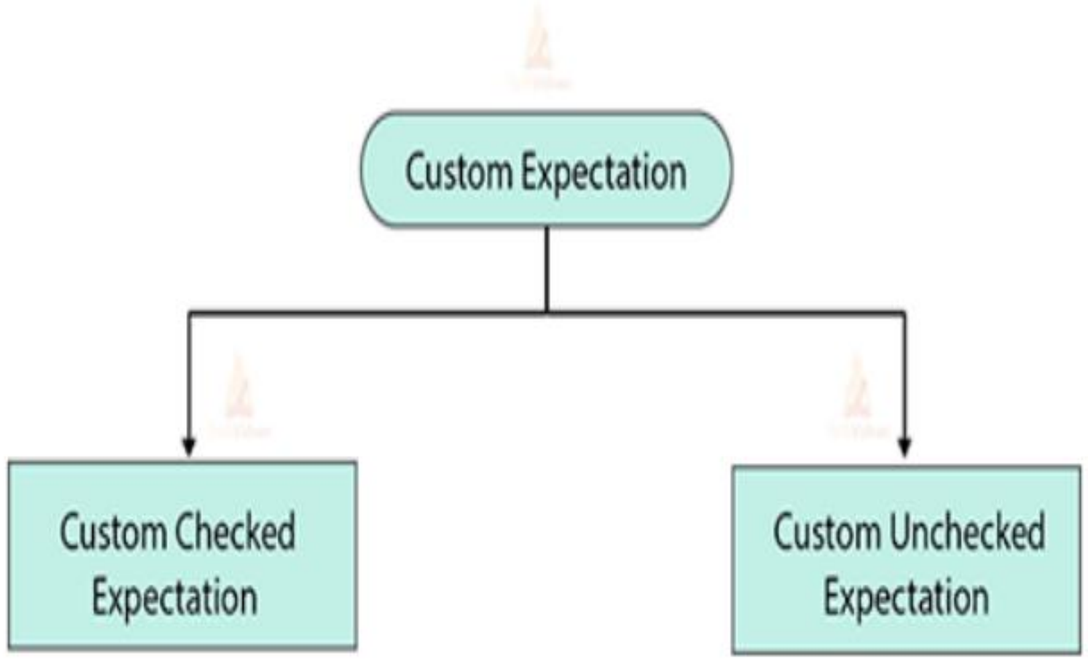
- Used in method declaration

```
import java.io.*;
class Test {
    static void readFile() throws IOException {
        FileReader f = new FileReader("abc.txt");
    }
    public static void main(String[] args) {
        try {
            readFile();
        } catch (IOException e) {
            System.out.println("Handled");
        }
    }
}
```

11. Custom Exception:

- User-defined exception class

```
class MyException extends Exception {
    MyException(String msg) {
        super(msg);
    }
}
class Test {
    public static void main(String[] args) {
        try {
            throw new MyException("Custom Error");
        } catch (MyException e) {
            System.out.println(e.getMessage());
        }
    }
}
```



12. Important Methods in Exception Class:

- getMessage() → returns message
- printStackTrace() → prints error details
- toString() → returns exception info

13. Advantages of Exception Handling:

✓ Prevents program crash

✓ Maintains normal flow

✓ Improves readability

✓ Helps debugging

✓ Separates error-handling code

14. Important Rules:

- Catch blocks must be in order (specific → general)
- Only one finally block per try
- finally always executes (except system crash)
- A try block must be followed by at least one catch or finally

15. Real-Time Example:

```
class Bank {
    public static void main(String[] args) {
        int balance = 5000;
        try {
            int withdraw = 6000;
            if (withdraw > balance) {
                throw new ArithmeticException("Insufficient Balance");
            } else {
                balance -= withdraw;
                System.out.println("Transaction successful");
            }
        } catch (ArithmeticException e) {
            System.out.println(e.getMessage());
        }
        System.out.println("Remaining Balance: " + balance);
    }
}
```


Exception Handling
Types of Exceptions
Checked Exceptions
Unchecked Exceptions
Handling Mechanisms
try-catch
finally block
throws Statement
Keywords
try
catch
finally
throw
Advantages
Error Handling
Program Stability
Improved Debugging
Real-World Applications
File I/O
Database Access

Assessment

1. What is the parent class of all Exceptions and Errors in Java?
2. Differentiate between Checked and Unchecked Exceptions
3. Explain the difference between throw and throws
4. Which of the following keywords is NOT used in Java exception handling?

    a) try

    b) catch

    c) thrown

    d) finally

THANK YOU