MODULE III: EXCEPTION HANDLING AND I/O

3.1. Exception Handling and User-Defined Exceptions

Aim:

To demonstrate exception handling in Java, specifically the creation and use of user-defined exceptions.

Algorithm:

1. Create a user-defined exception class `InvalidAgeException`.
2. Write a method that checks for valid age input (e.g., age > 0).
3. If the age is invalid, throw the custom exception.

Java Code:

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

public class ExceptionHandling {
    public static void validateAge(int age) throws InvalidAgeException {
        if (age < 1) {
            throw new InvalidAgeException("Age cannot be less than 1");
        } else {
            System.out.println("Valid Age: " + age);
        }
    }

    public static void main(String[] args) {
        try {
            validateAge(0);
        } catch (InvalidAgeException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Output:

```
Age cannot be less than 1
```

3.2. Java Interface for ADT Stack

Aim: To illustrate the implementation of an Abstract Data Type (ADT) Stack using a Java interface, and to demonstrate exception handling for stack overflow and underflow.

Algorithm:

1. Define an interface `StackADT` with methods like `push()`, `pop()`, `peek()`, and `isEmpty()`.
2. Implement this interface using an array-based class `Stack`.
3. Provide necessary exception handling for underflow and overflow.

Java Code:

```
interface StackADT {
    void push(int item);
    int pop();
    int peek();
    boolean isEmpty();
}

class Stack implements StackADT {
    private int maxSize = 10;
    private int[] stackArray = new int[maxSize];
    private int top = -1;

    public void push(int item) {
        if (top >= maxSize - 1) {
            System.out.println("Stack Overflow");
        } else {
            stackArray[++top] = item;
        }
    }

    public int pop() {
        if (top == -1) {
            System.out.println("Stack Underflow");
            return -1;
        } else {
            return stackArray[top--];
        }
    }

    public int peek() {
        if (top == -1) {
            System.out.println("Stack is Empty");
            return -1;
        } else {
            return stackArray[top];
        }
    }

    public boolean isEmpty() {
        return top == -1;
    }
}

public class StackTest {
    public static void main(String[] args) {
        Stack s = new Stack();
        s.push(10);
        s.push(20);
        System.out.println("Top element: " + s.peek());
        System.out.println("Popped element: " + s.pop());
        System.out.println("Is stack empty? " + s.isEmpty());
    }
}
```

Output:

```
Top element: 20
Popped element: 20
Is stack empty? false
```

3.3. File Handling in Java

Aim:

To demonstrate basic file handling operations in Java, including checking file existence, readability, writability, and displaying file details.

Algorithm:

1. Read a file name from the user.
2. Check if the file exists and if it is readable/writable.
3. Display the file details, such as file type and size.

Java Code:

```
java
Copy
import java.io.File;

public class FileInfo {
    public static void main(String[] args) {
        File file = new File("test.txt");

        if (file.exists()) {
            System.out.println("File exists");
            System.out.println("Readable: " + file.canRead());
            System.out.println("Writable: " + file.canWrite());
            System.out.println("File Type: " + (file.isDirectory() ?
"Directory" : "File"));
            System.out.println("File Size: " + file.length() + " bytes");
        } else {
            System.out.println("File does not exist");
        }
    }
}
```

Output:

```
File exists
Readable: true
Writable: true
File Type: File
File Size: 1024 bytes
```