Experiment No. 4

Implementation of Stack Operations using Interface

Instructions:

This manual consists of three parts: **A) Theory and Concepts, B) Problems for Implementation, and C) Write-up Questions.**

1. Students must understand the **theory and concepts** provided before implementing the problem statement(s) for **Experiment 4**.
2. They should **practice the given code snippets** within the theory section.
3. Later, they need to **implement the problems provided**.
4. **Write-up:** Students are required to **write answers** to the questions on journal pages, **maintain a file**, and get it checked regularly. The file should include index, write-up, and implementation code with results.
5. Referencing: Include proper sources or references for the content used.
6. Use of Generative AI: Clearly mention if you have used any AI tools (e.g., ChatGPT, Copilot, Bard) to generate text, explanations, or code. Cite the AI-generated content appropriately in the write-up.

Part A. Theory and Concepts:

An **Interface** is a reference type in Java, it acts as a contract for classes to follow. It is **similar to a class** and serves as a **collection of abstract methods**. A class **implements** an interface, thereby inheriting the abstract methods of the interface.
Along with **abstract methods**, an interface may also contain:

- **Constants** (public, static, and final by default)
- **Default methods** (with method bodies)
- **Static methods** (with method bodies)
- **Nested types**

Method bodies exist **only for default methods and static methods**.

Writing an interface is similar to writing a class. However, while a **class describes the attributes and behaviors of an object**, an **interface contains behaviors that a class implements**. Unless

the class that implements the interface is **abstract**, all the methods of the interface must be defined in the class.

Similarities Between an Interface and a Class:

- An interface can contain **any number of methods**.
- An interface is written in a **.java** file, and its name should match the file name.
- The **bytecode of an interface** is stored in a **.class** file.
- Interfaces are placed inside **packages**, and their corresponding bytecode files must be in a directory structure matching the package name.

Differences Between an Interface and a Class:

However, an **interface differs from a class** in several ways, including:

a. An interface cannot be instantiated.
b. An interface does not contain any constructors.
c. All methods in an interface are abstract (except default and static methods).
d. An interface cannot contain instance fields. The only fields allowed in an interface must be declared both static and final.
e. An interface is not extended by a class; it is implemented by a class.

Defining the interface –

```
//Syntax of an Interface:

interface InterfaceName {
    returnType method1(parameters);
    returnType method2(parameters);

    // Constants (By default: public, static, final)
    type FINAL_VARIABLE = value;
}
```

Implementing the interface –

```
// Syntax for Implementing an Interface:

class ClassName implements Interface1, Interface2 {
    // Class body
}
```

Example 1: Basic Interface Implementation

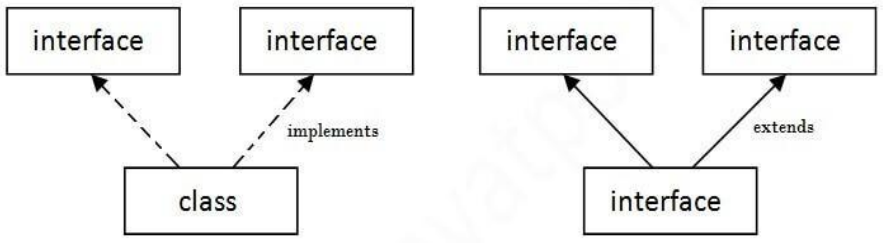
```
// Example 1: Basic Interface Implementation
// Define an interface
interface Vehicle
{
    void start();  // Abstract method
}

// Class implementing the interface
class Car implements Vehicle {
    public void start() {
        System.out.println("Car is starting with a key ignition.");
    }
}

// Test class
class Test {
    public static void main(String args[]) {
        Car myCar = new Car();
        myCar.start();
    }
}
```

Output:

```
Car is starting with a key ignition.
```

Multiple inheritance in Java by interface:

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



Multiple Inheritance in Java

Example 2: Multiple Interface Implementation

```
// Example 2a: Multiple Interface Implementation

interface Iface1 {
    void print();
}

interface Iface2 {
    void show();
}

class interfacedemo implements Iface1, Iface2 {
    public void print() {
        System.out.println("Iface1 Method");
    }

    public void show() {
        System.out.println("Iface2 Method ");
    }

    public static void main(String args[]) {
        interfacedemo obj = new interfacedemo();
        obj.print();
        obj.show();
    }
}
```

```
Output:
Iface1 Method
Iface2 Method
```

```
// Example 2b: Multiple Interface Implementation

// First interface
interface Printer {
    void print();
}

// Second interface
interface Scanner {
    void scan();
}

// Class implementing both interfaces

class AllInOnePrinter implements Printer, Scanner {
    public void print() {
        System.out.println("Printing document...");
    }

    public void scan() {
        System.out.println("Scanning document...");
    }

    public static void main(String args[]) {
        AllInOnePrinter device = new AllInOnePrinter();
```

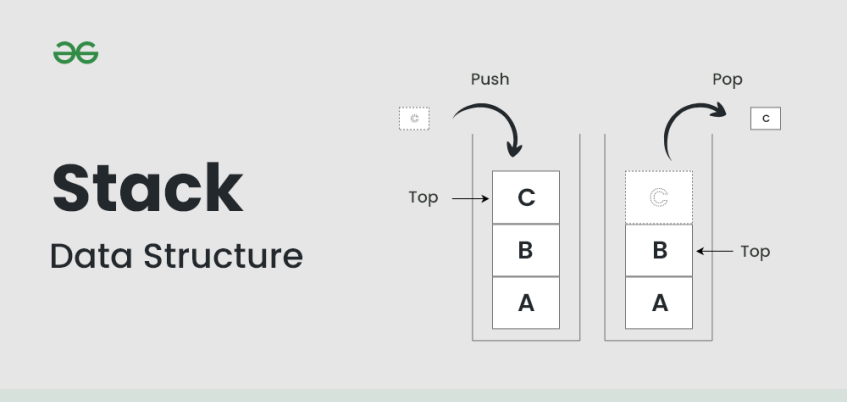
```
        device.print();
        device.scan();
    }
}
```

```
Output:
Printing document...
Scanning document...
```

Stack Data Structure

A **Stack** is a linear data structure that follows a particular order in which the operations are performed. The order may be **LIFO (Last In First Out)** or **FILO (First In Last Out). LIFO** implies that the element that is inserted last, comes out first and **FILO** implies that the element that is inserted first, comes out last. It behaves like a stack of plates, where the last plate added is the first one to be removed.

- Pushing an element onto the stack is like adding a new plate on top.
- Popping an element removes the top plate from the stack.



(*https://www.geeksforgeeks.org/stack-data-structure/*)

Part B. Problems for Implementation:

Aim: Implement stack operations using an interface.

1. **Stack Operations using Interface:** Create an interface Stack with a variable size and abstract methods push(), pop(), display(), overflow(), and underflow(). Implement a subclass IntegerStack that implements the Stack interface. Create a test class to check the working of all methods in the IntegerStack class.

2. **Shape Interface with Rectangle and Triangle:** Implement the following:

    a. Create an interface Shape with an abstract method area().
    b. Create two classes, Rectangle and Triangle, that implement the Shape interface.
    c. Calculate and display the area of both Rectangle and Triangle.

3. **Student Exam Results Using Inheritance and Interface in:** Implement the following hierarchy:
    a. Create a class Student with a variable rollNo and methods getRollNo() and setRollNo().
    b. Create a class Test that inherits Student and has variables sub1 and sub2 with methods getMarks() and setMarks().
    c. Create an interface Sports with a variable sMarks and a method set().
    d. Create a class Result that inherits Test and implements the Sports interface. It should display the marks.
    e. Demonstrate the functionality of these classes in a test application.

Part C. Write-up Questions:

1. What is an **interface** in Java? Explain its features.
2. Explain an interface with **default** and **static** methods in Java.
3. How does Java support **multiple inheritance using interfaces**? Explain with an example.
4. What is the **difference** between an **abstract class** and an **interface** in Java?

Conclusion:

Students should be able to implement the interface concept.