Experiment 9a Date: 31/10/2023

Roll number: 22z215

Interfaces

1.Teacher and Student are two classes each of them having their own unique components. You want to define an interface, say, Resume which will include all the components both in Teacher and Student.

Aim:

To write a java program to create an interface and implement it in two classes.

Code:

```
package Exp9a;

interface Resume {
    void displayInfo();
}

class Teacher implements Resume {
    @Override
    public void displayInfo() {
        System.out.println("Teacher's information in printed.");
        // Additional components specific to the Teacher class
    }
}

class Student implements Resume {
    @Override
    public void displayInfo() {
        System.out.println("Student's information is printed.");
        // Additional components specific to the Student class
    }
}

public class Q1 {
    public static void main(String[] args) {
        // Creating objects of Teacher and Student
        Teacher teacher = new Teacher();
        Student student = new Student();

        // Calling displayInfo for Teacher
        System.out.println("Teacher's information:");
        teacher.displayInfo();

        // Calling displayInfo for Student
```

```
        System.out.println("\nStudent's information:");
        student.displayInfo();
    }
}
```

Output:

```
Teacher's information:
Teacher's information in printed.

Student's information:
Student's information is printed.
```

---

2. Interface, in many way, similar to a class; however, no object can be instantiated from an interface. Demonstrate it.

Aim:
To write a java program to create an interface and try to instantiate it.

Code:

```
package Exp9a;

interface MyInterface {
    void myMethod();
}

public class Q2 {
    public static void main(String[] args) {
        // This will throw an error
        MyInterface interfaceObject = new MyInterface();
    }
}
```

Output:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        Cannot instantiate the type MyInterface

        at Exp9a.Q2.main(Q2.java:10)
```

---

3. Like classes, interfaces also have a chain of inheritance i.e. an interface can be derived from other interface. Implement single inheritance with interface .

Aim:

To write a java program to create an interface and implement it in two classes.

Code:

```
package Exp9a;

// Interface I1
interface I1 {
    void methodI1();
}

// Interface I2 extending I1
interface I2 extends I1 {
    void methodI2();
}

// Class implementing both I1 and I2
class Example implements I2 {
    public void methodI1() {
        System.out.println("Method from I1");
    }

    public void methodI2() {
        System.out.println("Method from I2");
    }
}

public class Q3 {
    public static void main(String[] args) {

        Example example = new Example();

        example.methodI1();
        example.methodI2();
    }
}
```

Output:

```
Method from I1
Method from I2
```

4. An interface is a significant feature in Java in the sense that it enables the multiple inheritance. Implement the same. Illustrate class A which "implements" two interfaces (I1 and I2)

Aim:

To write a java program to create an interface and implement it in two classes.

Code:

```
package Exp9a;

interface I1{
    public void methodI1();
}

interface I2{
    public void methodI2();
}

public class Q4_MultipleInheritance implements I1, I2{
    public void methodI1() {
        System.out.println("Method from I1");
    }

    public void methodI2() {
        System.out.println("Method from I2");
    }

    public static void main(String[] args) {
        Q4_MultipleInheritance myObj = new Q4_MultipleInheritance();
        myObj.methodI1();
        myObj.methodI2();
    }
}
```

Output:

```
Method from I1
Method from I2
```

---

5. Use two interfaces and implement the drive() method in Hybrid_Car class.

Aim:

To write a java program to create an interface and implement it in two classes.

Code:

```java
package Exp9a;

interface CNG_Car{
    public void cng_kit();
    public void drive();
}

interface Petrol_Car{
    public void petrol_kit();
    public void drive();
}

public class Q5_hybridCar {

    public void cng_kit(){
        System.out.println("CNG Kit Installed");
    }

    public void petrol_kit(){
        System.out.println("Petrol Kit Installed");
    }

    public void drive(){
        System.out.println("Driving");
    }

    public static void main(String[] args) {
        Q5_hybridCar h = new Q5_hybridCar();
        h.cng_kit();
        h.petrol_kit();
        h.drive();
    }

}
```

Output:

```
CNG Kit Installed
Petrol Kit Installed
Driving
```

---

6. Design an interface named Stack with the following methods

a. Push and Pop elements from the stack
b. Check whether the stack is empty or not. Implement the stack with the help of arrays and if the size of the array becomes too small to hold the elements, create a new one. Test this interface by inheriting it in its subclass StackTest.java.

Aim:

To write a java program to create an interface and implement it in two classes.

Code:

```
package Exp9a;

// Stack interface
interface Stack {
    void push(int element);
    int pop();
    boolean isEmpty();
}

// Implementation of Stack interface
class StackTest implements Stack {
    private int[] stack;
    private int top;
    private static final int DEFAULT_CAPACITY = 10;

    public StackTest() {
        stack = new int[DEFAULT_CAPACITY];
        top = -1;
    }

    public void push(int element) {
        if (top == stack.length - 1) {
            // If the size is too small, create a new array
            int[] newStack = new int[stack.length * 2];
            System.arraycopy(stack, 0, newStack, 0, stack.length);
            stack = newStack;
        }
        stack[++top] = element;
    }

    public int pop() {
        if (isEmpty()) {
            throw new RuntimeException("Stack is empty");
        }
        return stack[top--];
    }

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

    public static void main(String[] args) {
        StackTest myStack = new StackTest();

        // Pushing elements onto the stack
        myStack.push(10);
```

```
        myStack.push(20);
        myStack.push(30);

        // Popping elements from the stack
        System.out.println("Popped element: " + myStack.pop());
        System.out.println("Popped element: " + myStack.pop());

        // Checking if the stack is empty
        System.out.println("Is the stack empty? " + myStack.isEmpty());
    }
}
```

Output:

```
Popped element: 30
Popped element: 20
Is the stack empty? false
```

Result:

All the programs are executed and the output are verified.
