**PRACTICAL # 04**

**OBJECT:**

**Interfaces and Inheritance in Java**

**Extension of the Stack class**

**THEORY:**

An interface is a group of abstract methods with empty bodies. Classes implement interface, thereby inheriting the abstract methods of the interface.

Class and interface are two different concepts. A class describes the attributes and behaviors of an object. An interface contains unimplemented behaviors that a class implements.

A non abstract class that implements the interface, has to define all the methods of the interface.

**Interface properties:**

* An interface can contain any number of methods.
* An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file.
* The bytecode of an interface appears in a **.class** file.
* Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.
* An interface is implicitly abstract. The **abstract** keyword is not used when declaring an interface.
* Each method in an interface is also implicitly abstract, so the abstract keyword is not used.
* Interface Methods are implicitly public.

Here are some interface properties that differ with class:

* An interface cannot instantiated.
* An interface does not contain any constructors.
* All of the methods in an interface are abstract.
* An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
* An interface is not extended by a class; it is implemented by a class.
* Instances of interface cannot be by itself. There must be some class that implements the interface, and reference that ckass instance as an instance of the interface.

**Interface Syntax:**

public interface InterfaceName

{

//final, static fields

//abstract method declarations\

}

**Interfaces Implementation:**

When a class implements an interface, it has to perform the specific behaviors of the interface. If a class does not perform all the behaviors of the interface, the class must declare itself as abstract.

A class uses the implements keyword to implement an interface. The implements keyword appears in the class declaration following the extends part of the declaration.

//Vehicle.java

public interface Vehicle {

public void accelerate();

public void brake();

}

//Car.java

public class Car implements Vehicle {

public void accelerate(){

//code for vehicle acceleration

}

public void brake(){

//code for braking system

}

}

**Interface Implementation Rules:**

* The signature of the interface method should be same when overriding the methods.
* A class can extend only one class, but implement many interfaces.
* An interface can extend another interface, as a class can extend another class.

**code:**

interface Bicycle {

void changeGear(int newValue);

void speedUp(int increment);

void applyBrakes(int decrement);

}

To implement this interface, define a class such as ACMBicycle), and use the implements keyword in the class declaration:

public class ACMBicycle implements Bicycle {

int speed = 0;

int gear = 1;

// The compiler will now require that methods

// changeGear, speedUp, and applyBrakes all be implemented. Compilation will fail if those

// methods are missing from this class.

void changeGear(int newValue) {

gear = newValue;

}

void speedUp(int increment) {

speed = speed + increment;

}

void applyBrakes(int decrement) {

speed = speed - decrement;

}

void printStates() {

System.out.println(" speed:" + speed + " gear:" + gear);

}

}

Implementing an interface allows a class to implement the behavior of the interface. Interfaces form a contract between the class and the outside world, and this contract is enforced at build time by the compiler. If a class implements an interface, all methods defined by that interface must appear in its source code before the class will successfully compile.

**Extending Interfaces:**

An interface can extend another interface. The extends keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.

The following Sports interface is extended by Hockey and Football interfaces.

//Sports.java

public interface Sports

{

public void setHomeTeam(String name);

public void setVisitingTeam(String name);

}

//Football.java

public interface Football extends Sports

{

public void homeTeamScored(int points);

public void visitingTeamScored(int points);

public void endOfQuarter(int quarter);

}

//Hockey.java

public interface Hockey extends Sports

{

public void homeGoalScored();

public void visitingGoalScored();

public void endOfPeriod(int period);

public void overtimePeriod(int ot);

}

The Football interface has three methods, and it inherits two from Sports; thus, a class that implements Football needs to implement all the five methods, three methods from Football and the two methods from Sports.

**Extending Multiple Interfaces:**

A class in Java can only extend one parent class. Multiple classes inheritance is not allowed. However an interface can extend more than one parent interface.

The extends keyword is used once, and the parent interfaces are declared in a comma-separated list.

For example, if the Football interface extended both Sports and Event, it would be declared as:

public interface Football extends Sports, Event{

}

## Default Methods

If the developer of the API adds one more method to some interface, then the classes using the interface of the API will be broken when that project upgrades to the new version of the API.

To alleviate this Java interface evolution problem, Java 8 has added the concept of *interface default methods* to Java interfaces. An interface default method can contain a default implementation of that method. Classes that implement the interface but which contain no implementation for the default interface will then automatically get the default method implementation.

Public interface Vehicle{

public void accelerate();

default public void setClimateControl(float temp){

}

}

Let say the setClimateControl method is added in a newer version of the API. So the classes implementing the interface and using old version of the API will not break if the project API is upgraded. A class can override the implementation of a default method simply by implementing that method explicitly, as is done normally when implementing a Java interface.

**Inheritance**

Inheritance allows reusablity of the existing code. Its one of the core concepts of the object-oriented programming. This feature allows to design a better code.

Inheritance preserves the existing feature of the original class and adds more functionality.

The example of the stack class done in the previous lab is a good candidate for inheritance. We can add more useful functions like how much is stack full now and clear all the items in the stack.

public class Stack1 extends Stack{

public int getTotalStackElements(){

//stackTop index at any point will be the total number of elements in the stack

return stackTop;

}

public void emptyStack(){

stackTop = 0; //this resets the stack

}

}

**ACTIVITIES**

**Activity 1**

Write a similar program that extends the class Queue and adds

**Activity 2**

Write a similar program that extends the class Queue and adds the functionality of getting queue size and clearing the queue.

**REVIEW QUESTIONS**

1. Where are the remainder and the quotient found in case of 16 bit number division?
2. What will happen in division operation if dividend is 0 and what will happen if divisor is 0?
3. What will be the result of using uninitialized variable in program without properly initializing its value?
4. Why does the multiplication of two 16-bit numbers require for their product two registers instead of one? Give some suitable example.