Experiment No. : 06

Title: Design java interfaces for ADT stack and Queue. Develop a class that implements these interfaces for achieving multiple inheritance

Objectives:

1) To learn how to create an interface.
2) To learn how to achieve multiple inheritance.

Theory:

Interfaces-

There are a number of situations in software engineering when it is important for disparate groups of programmers to agree to a "contract" that spells out how their software interacts. Each group should be able to write their code without any knowledge of how the other group's code is written. Generally speaking, interfaces are such contracts.

For example, imagine a futuristic society where computer-controlled robotic cars transport passengers through city streets without a human operator. Automobile manufacturers write software (Java, of course) that operates the automobile—stop, start, accelerate, turn left, and so forth. Another industrial group, electronic guidance instrument manufacturers, makes computer systems that receive GPS (Global Positioning Satellite) position data and wireless transmission of traffic conditions and use that information to drive the car.

The auto manufacturers must publish an industry-standard interface that spells out in detail what methods can be invoked to make the car move (any car, from any manufacturer). The guidance manufacturers can then write software that invokes the methods described in the interface to command the car. Neither industrial group needs to know how the other group's software is implemented. In fact, each group considers its software highly proprietary and reserves the right to modify it at any time, as long as it continues to adhere to the published interface.

Interfaces in Java-

In the Java programming language, an interface is a reference type, similar to a class that can contain only constants, method signatures, and nested types. There are no method bodies. Interfaces cannot be instantiated—they can only be implemented by classes or extended by other interfaces. Extension is discussed later in this lesson.

Defining an interface is similar to creating a new class:

```
public interface OperateCar
{
        // constant declarations, if any
        // method signatures
}
```

Note that the method signatures have no braces and are terminated with a semicolon.

To use an interface, you write a class that implements the interface. When an instantiable class implements an interface, it provides a method body for each of the methods declared in the interface. For example,

```
public class OperateBMW760i implements OperateCar
{
        // the OperateCar method signatures, with implementation --
}
// other members, as needed -- for example, helper classes
// not visible to clients of the interface
}
```

In the robotic car example above, it is the automobile manufacturers who will implement the interface. Chevrolet's implementation will be substantially different from that of Toyota, of course, but both manufacturers will adhere to the same interface. The guidance manufacturers, who are the clients of the interface, will build systems that use GPS data on a car's location, digital street maps, and traffic data to drive the car. In so doing, the guidance systems will invoke the interface methods: turn, change lanes, brake, accelerate, and so forth.

Interfaces as APIs

The robotic car example shows an interface being used as an industry standard Application Programming Interface (API). APIs are also common in commercial software products. Typically, a company sells a software package that contains complex methods that another company wants to use in its own software product. An example would be a package of digital image processing methods that are sold to companies making end-user graphics programs. The image processing company writes its classes to implement an interface, which it makes public to its customers. The graphics company then invokes the image processing methods using the signatures and return types defined in the interface. While the image processing company's API is made public (to its customers), its implementation of the API is kept as a closely guarded secret—in fact, it may revise the implementation at a later date as long as it continues to implement the original interface that its customers have relied on.

Interfaces and Multiple Inheritance

Interfaces have another very important role in the Java programming language. Interfaces are not part of the class hierarchy, although they work in combination with classes. The Java programming language does not permit multiple inheritance, but interfaces provide an alternative.

In Java, a class can inherit from only one class but it can implement more than one interface. Therefore, objects can have multiple types: the type of their own class and the types of all the interfaces that they implement. This means that if a variable is declared to be the type of an interface, its value can reference any object that is instantiated from any class that implements the interface. This is discussed later in this lesson, in the section titled "Using an Interface as a Type."

Defining an Interface

An interface declaration consists of modifiers, the keyword interface, the interface name, a comma-separated list of parent interfaces (if any), and the interface body. For example:

```
public interface GroupedInterface extends Interface1,Interface2, Interface3 {
// constant declarations
double E = 2.718282; // base of natural logarithms
// method signatures
void doSomething (int i, double x);
int doSomethingElse(String s);
}
```

The public access specifier indicates that the interface can be used by any class in any package. If you do not specify that the interface is public, your interface will be accessible only to classes defined in the same package as the interface.

An interface can extend other interfaces, just as a class can extend or subclass another class. However, whereas a class can extend only one other class, an interface can extend any number of

interfaces. The interface declaration includes a comma-separated list of all the interfaces that it extends.

The Interface Body

The interface body contains method declarations for all the methods included in the interface. A method declaration within an interface is followed by a semicolon, but no braces, because an interface does not provide implementations for the methods declared within it. All methods declared in an interface are implicitly public, so the public modifier can be omitted.

An interface can contain constant declarations in addition to method declarations. All constant values defined in an interface are implicitly public, static, and final. Once again, these modifiers can be omitted.

Implementing an Interface

To declare a class that implements an interface, you include an implements clause in the class declaration. Your class can implement more than one interface, so the implements keyword is followed by a comma-separated list of the interfaces implemented by the class. By convention, the implements clause follows the extends clause, if there is one.

Using an Interface as a Type

When you define a new interface, you are defining a new reference data type. You can use interface names anywhere you can use any other data type name. If you define a reference variable whose type is an interface, any object you assign to it must be an instance of a class that implements the interface.

Keywords: class, static, javac, public, command line argument

Algorithm:

1. Define interface StackInterface and declare the methods push, pop, peek, display inside the interface.
2. Define interface QueueInterface and declare the methods enqueue, dequeue, display inside the interface.
3. Define a class StackQueue that implements both these interfaces and define the methods declared in these interfaces.
4. Use the class StackQueue to perform the stack and queue operations using array implementation.
5. Define class StackQueueTest and write main method in this class. Create object of StackQueue class and call the methods using switch.
6. Display the result.

Note: Please follow the naming conventions while writing the program.