⇧ SCROLL TO TOP

Marker Interface in Java

In this section, we will discuss about **marker interface in Java**, its **uses, built-in (Serializable, Cloneable, and Remote Interfaces)** and **custom marker interface** with examples.

What is marker interface?

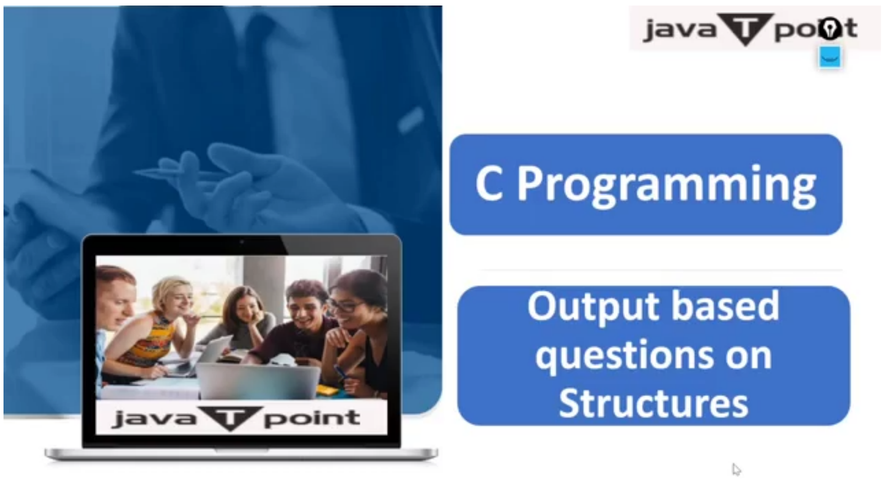
An interface that does not contain methods, fields, and constants is known as **marker interface**. In other words, an empty interface is known as **marker interface** or **tag interface.** It delivers the run-time type information about an object. It is the reason that the JVM and compiler have additional information about an object. The **Serializable** and **Cloneable** interfaces are the example of marker interface. In short, it indicates a signal or command to the JVM.

The declaration of marker interface is the same as interface in Java but the interface must be empty. For example:

```
public interface Serializable
{

}
```

There are the two alternatives of marker interface that produces the same result as the marker interface.



- **Internal Flags:** It can be used in place of marker interface to indicate any specific operation.
- **Annotations:** Since Java 5, **marker interfaces are omitted**. Instead of marker interface, Java 5 provides the **annotations** to achieve the same results. It allows flexible metadata capability. Therefore, by applying annotations to any class, we can perform specific action.

Uses of Marker Interface

Marker interface is used as a tag that inform the Java compiler by a message so that it can add some special behavior to the class implementing it. Java marker interface are useful if we have information about the class and that information never changes, in such cases, we use marker interface represent to represent the same. Implementing an empty interface tells the compiler to do some operations.

It is used to logically divide the code and a good way to categorize code. It is more useful for developing API and in frameworks like Spring.

⇧ SCROLL TO TOP

Built-in Marker Interface

In Java, built-in marker interfaces are the interfaces that are already present in the JDK and ready to use. There are many built-in marker interfaces some of them are:

- Cloneable Interface
- Serializable Interface
- Remote Interface

Let's discuss one by one in detail.

Cloneable Interface

Cleanable interface in Java is also a marker interface that belong to **java.lang** package. It generates replica (copy) of an object with different name. We can implement the interface in the class of which class object to be cloned. It indicates the **clone()** method of the Object class. If we do not implement the Cloneable interface in the class and invokes the clone() method, it throws the **ClassNotSupportedException.**

Note that a class that implements the Cloneable interface must override the clone() method with a public method. Let's see an example.

Product.java

```
import java.util.Scanner;
public class Product implements Cloneable
{
int pid;
String pname;
double pcost;
//Product class constructor
public Product (int pid, String pname, double pcost)
{
this.pid = pid;
this.pname = pname;
this.pcost = pcost;
}
//method that prints the detail on the console
public void showDetail()
{
System.out.println("Product ID: "+pid);
System.out.println("Product Name: "+pname);
System.out.println("Product Cost: "+pcost);
}
public static void main (String args[]) throws CloneNotSupportedException
{
the product from the user
```

⇧ SCROLL TO TOP

```
Scanner sc = new Scanner(System.in);
System.out.print("Enter product ID: ");
int pid = sc.nextInt();
System.out.print("Enter product name: ");
String pname = sc.next();
System.out.print("Enter product Cost: ");
double pcost = sc.nextDouble();
System.out.println("-------Product Detail--------");
Product p1 = new Product(pid, pname, pcost);
//cloning the object of the Product class using the clone() method
Product p2 = (Product) p1.clone();
//invoking the method to print detail
p2.showDetail();
}
}
```

Output:

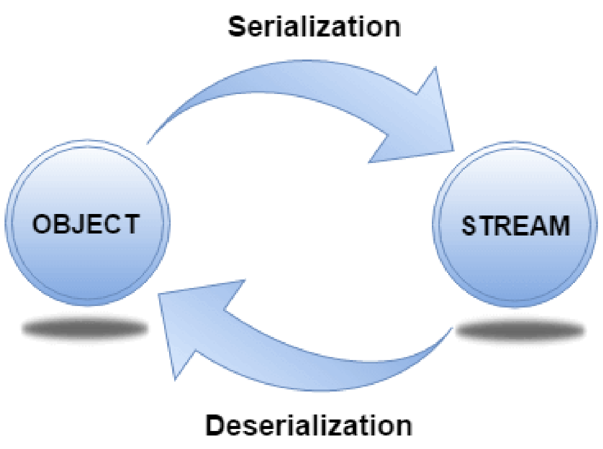
```
Enter product ID: 139872
Enter product name: Printer
Enter product Cost: 3459.67
-------Product Detail--------
Product ID: 139872
Product Name: Printer
Product Cost: 3459.67
```

Serializable Interface

It is a marker interface in Java that is defined in the **java.io** package. If we want to make the class serializable, we must implement the **Serializable** interface. If a class implements the Serializable interface, we can serialize or deserialize the state of an object of that class.

Serialization (converting an object into byte stream) is a mechanism in which **object state is read from the memory and written into a file or database**. Deserialization (converting byte stream into an object) is the [illegible]tion means that **object state reading from a file or database and written back into memory** [illegible]n of object.

⇧ SCROLL TO TOP



Serialization (writing) can be achieved with the **ObjectOutputStream** class and deserialization (reading) can be achieved with the **ObjectInputStream** class.

Let's see example of serialization and deserialization.

Example of Serialization

Employee.java

```
import java.io.Serializable;
public class Employee implements Serializable
{
int empid;
String empname;
public Employee(int empid, String empname)
{
this.empid = empid;
this.empname = empname;
}
}
```

SerializationExample.java

```
import java.io.*;
class SerializationExample
main(String args[])
```

⇧ SCROLL TO TOP

```
{
try
{
//Creating the object
Employee emp =new Employee(1187345,"Andrew");
//Creating stream and writing the object
FileOutputStream fout=new FileOutputStream("employee data.txt");
ObjectOutputStream out=new ObjectOutputStream(fout);
out.writeObject(emp);
out.flush();
//closing the stream
out.close();
System.out.println("Data has been read from the file.");
}
catch(Exception e)
{
e.printStackTrace();
}
}
}
```

Output:

```
Data has been read from the file.
```

Example of Deserialization

Let's deserialize the object state.

DeserializationExample.java

```
import java.io.*;
class DeserializationExample
{
public static void main(String args[])
{
try
{
//Creating stream to read the object
ObjectInputStream in=new ObjectInputStream(new FileInputStream("employee data.txt"));
```

⇧ SCROLL TO TOP

```
nployee)in.readObject();
of the serialized object
```

```
System.out.println(emp.empid+" "+emp.empname);
//closing the stream
in.close();
}
catch(Exception e)
{
e.printStackTrace();
}
}
}
```

Output:

```
1187345 Andrew
```

Remote Interface

Remote interface is a marker interface that belong to **java.rmi** package. It marks an object as remote that can be accessed from another machine (host). We must implement the Remote interface if we want to make an object as remote. It identifies the interfaces whose methods can be invoked from a non-local JVM. Any remote object must implement the interface directly or indirectly.

Let's define a Remote interface and implement it in a Java program.

Defining Remote interface

```
import java.rmi.*;
public interface AddAll extends Remote
{
public int add(int r, int s)throws RemoteException;
}
```

Implement the Remote Interface

There are following two ways to implement the Remote interface:

- By extending the UnicastRemoteObject class
- By using the exportObject() method of the UnicastRemoteObject class

AddAllRemote.java

```
import java.rmi.*;
import java.rmi.server.*;
public class AddAllRemote extends UnicastRemoteObject implements Adder
{
AddAllRemote()throws RemoteException
```

⇧ SCROLL TO TOP

```
super();
}
public int add(int r, int s)
{
return r+s;
}
```

Create and Start the Remote Application

Server.java

```
import java.rmi.*;
import java.rmi.registry.*;
public class Server
{
public static void main(String args[])
{
try
{
AddAll stub=new AddAllRemote();
Naming.rebind("rmi://localhost:5000/sak",stub);
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

Create and Start the Client Application

Client.java

```
import java.rmi.*;
public class Client
{
public static void main(String args[])
{
try
{
AddAll stub=(AddAll)Naming.lookup("rmi://localhost:5000/sak");
System.out.println(stub.add(29,18));
}
catch(Exception e)
```

⇧ SCROLL TO TOP

```
}
}
```

Custom Marker Interface

Apart from built-in marker interface, Java also allows us to create own marker interface. Let's see an example.

CustomMarkerInterfaceExample.java

```
//custom marker interface
interface Car
{

}
//custom marker interface
interface Engine
{

}
//class that implements the Car marker interface
class Vehicle implements Car
{
static void isVehicle()
{
System.out.println("Car is a vehicle.");
}
}
//class that implements the Engine marker interface
class Status implements Engine
{
static void isWorking()
{
System.out.println("Yes, engine is working.");
}
```

⇧ SCROLL TO TOP

```
public class CustomMarkerInterfaceExample
{
public static void main(String args[])
{
//invoking the methods of the class
Vehicle.isVehicle();
Status.isWorking();
}
}
```

Output:

```
Car is a vehicle.
Yes, engine is working.
```

Learn Latest Tutorials

Splunk tutorial
Splunk

SPSS tutorial
SPSS

Swagger tutorial
Swagger

T-SQL tutorial
Transact-SQL

Tumblr tutorial
Tumblr

React tutorial
ReactJS

Regex tutorial
Regex

Reinforcement learning tutorial
Reinforcement Learning

R Programming tutorial
R Programming

RxJS tutorial
RxJS

React Native tutorial
React Native

Python Design Patterns
Python Design Patterns

Python Pillow tutorial
Python Pillow

Python Turtle tutorial
Python Turtle

Keras tutorial
Keras

Preparation

Aptitude
Aptitude

Logical Reasoning
Reasoning

Verbal Ability
Verbal Ability

Interview Questions
Interview Questions

Company Interview Questions
Company Questions

Trending Technologies

Artificial Intelligence Tutorial
Artificial

AWS Tutorial
AWS

Selenium tutorial
Selenium

Cloud Computing tutorial
Cloud Computing

Hadoop tutorial
Hadoop

⇧ SCROLL TO TOP

ReactJS Tutorial
ReactJS

Data Science Tutorial
Data Science

Angular 7 Tutorial
Angular 7

Blockchain Tutorial
Blockchain

Git Tutorial
Git

Machine Learning Tutorial
Machine Learning

DevOps Tutorial
DevOps

B.Tech / MCA

DBMS tutorial
DBMS

Data Structures tutorial
Data Structures

DAA tutorial
DAA

Operating System tutorial
Operating System

Computer Network tutorial
Computer Network

Compiler Design tutorial
Compiler Design

Computer Organization and Architecture
Computer Organization

Discrete Mathematics Tutorial
Discrete Mathematics

Ethical Hacking Tutorial
Ethical Hacking

Computer Graphics Tutorial
Computer Graphics

Software Engineering Tutorial
Software Engineering

html tutorial
Web Technology

Cyber Security tutorial
Cyber Security

Automata Tutorial
Automata

C Language tutorial
C Programming

C++ tutorial
C++

Java tutorial
Java

.Net Framework tutorial
.Net

Python tutorial
Python

List of Programs
Programs

Control Systems tutorial
Control System

Data Mining Tutorial
Data Mining

Data Warehouse Tutorial
Data Warehouse

⇧ SCROLL TO TOP

⇧ SCROLL TO TOP