CSE 1203

Object Oriented Programming [C++]

Chapter 5:
Java Programming -02
[OOP's Features in JAVA]

OOP's Features

OOP: Features

- Class
- Objects
- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

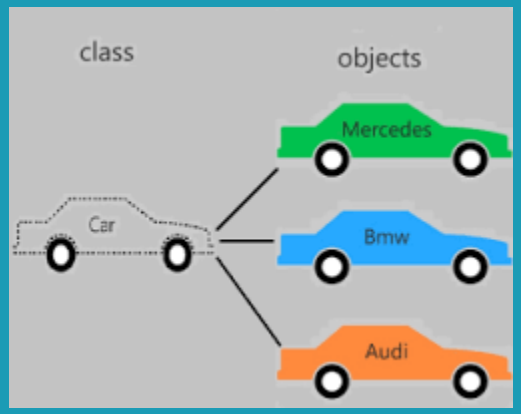
Code Reusability Features

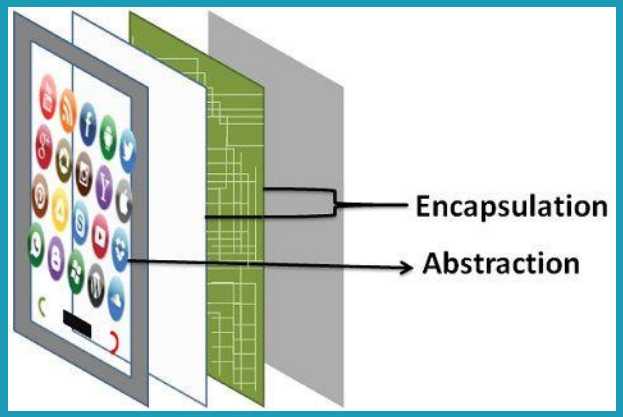
- Inheritance
- Polymorphism

Security Features

- Encapsulation
- Abstraction

Abstraction	Encapsulation
Abstraction is the method of hiding the unwanted information.	Whereas encapsulation is a method to hide the data in a single entity or unit along with a method to protect information from outside.
We can implement abstraction using abstract class and interfaces.	Whereas encapsulation can be implemented using by access modifier i.e. private, protected and public.
In abstraction, implementation complexities are hidden using abstract classes and interfaces.	While in encapsulation, the data is hidden using methods of getters and setters.





Java: inheritance

Java: Inheritance

```
class Calculator{
   int s;
   public int add(int x,int y) {
     s=x+y;
   return s;
   }
   public int sub(int x,int y) {
     s=x-y;
   return s;
   }
}
public class First extends Calculator {
  public static void main(String[] args) {
   First cal=new First();
   System.out.println(cal.add(3, 5));
}
}
```

Note: while defining multiple classes in a single Java file you need to make sure that only one class among them is public. If you have more than one public classes a single file a compile-time error will be generated.
Use **extends** keyword after class definition for inheritance

Java: uses of super keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

```
class Person{
String name="Person";
public void Display() {
  System.out.println(name);
}
}
class Employee extends Person{
String name="Employee";

public void Display() {
  super.Display();
  System.out.println(name);
}
}
public class First {
public static void main(String[] args) {
Employee emp=new Employee();
emp.Display();
}
```

Note: here **super**, calls the parent class Display() method

Java: super Example

Java: calling constructor

Constructor: Default constructor is automatically called when child class object is created but super() keyword is used to call parameterized constructor

Note: Generally parent and then child class constructor is called when child class object is created. If we want to call parameterized constructor then super(values) is used as the first statement inside the child constructor.

Java: calling constructor

```
class A{
int x=10;
public A() {
System.out.println("in A constructor");
}
public A(int x) {
System.out.println("in A parameter constructor x="+x);
}
public  void Display() {
System.out.println("X="+x);
}
}
class B extends A{
int x=30;
public B() {
super(10);
System.out.println("in B constructor");
}
public  void Display() {
super.Display();
System.out.println("X="+x);
}
}
public class First{
 public static void main(String[] args) {
  B a=new B();
  a.Display();
}
}
```

OOP's Features: abstract class

Java: OOP Features

Java implements OOP features in 2 ways

1. Abstract class (0-100% abstraction)
2. Interface (100% abstraction)

Points: Abstract class

- Any class that contains one or more abstract methods must also be declared abstract.
- If a class contains an abstract method it needs to be abstract and vice versa is not true.
- If a non-abstract class extends an abstract class, then the class must implement all the abstract methods of the abstract class else the concrete class has to be declared as abstract as well.

Example: Abstract class

In the following class car and bike has same data member and method. Modifications is as follows

```
class car{
  int no_tyres=4;
  public void Start(){
   Sop("start by key");
  }
}
```

```
class bike{
  int no_tyres=2;
  public void Start(){
   Sop("start by kick");
  }
}
```

Syntax: Abstract class

```
abstract class class_name {
//abstract or non-abstract methods
}
```

Example: Abstract class

The above code can be modified using abstract class. If a class contains a abstract method then the class must be abstract but an abstract class may or may not have abstract methods.. Here implementation is hidden. Abstract method has no body.

```
abstract class vehicle{
   int no_tyre;
   abstract void Start();
 }
```

```
class car extends vechicle{
  public void Start(){
  no_tyres=4;
   Sop("start by key");
  }
}
```

```
class bike extends vechicle{
  public void Start(){
   no_tyres=2;
   Sop("start by kick");
  }
}
```

OOP Features: abstract class

Java: abstract class example

```
abstract class Animal {
    public abstract void animalSound();
    public void sleep()
    {
    System.out.println("Zzz");
    }
}
```

- Like **abstract classes**, interfaces **cannot** be used to create objects
- Interface methods do not have a body - the body is provided by the "implement" class
- On implementation of an interface, you must override all of its methods
- Interface methods are by default `abstract` and `public`
- Interface attributes are by default `public`, `static` and `final`
- An interface cannot contain a constructor (as it cannot be used to create objects)

Java: abstract class

```
// Abstract class
abstract class Animal { // Abstract method (does
not have a body)
public abstract void animalSound();
// Regular method
public void sleep() { System.out.println("Zzz");
 }
}
 // Subclass (inherit from Animal)
 class Pig extends Animal {
 public void animalSound() {
// The body of animalSound() is provided here
System.out.println("The pig says: wee wee");
 }
}
class Main {
public static void main(String[] args)
{
  Pig myPig = new Pig();
/ / Create a Pig object
  myPig.animalSound();
  MyPig.sleep();
}
}
```

OOP's Features: interface

Java: OOP Features

Java implements OOP features in 2 ways

1. Abstract class (0-100% abstraction)
2. Interface (100% abstraction)

What is: Interface

It is an abstract class with all the methods are abstract

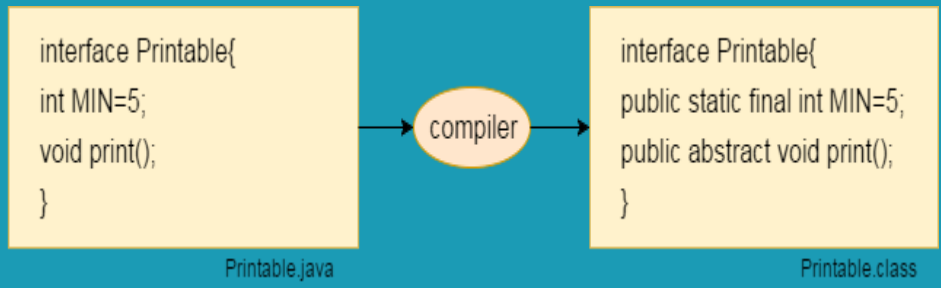
Syntax: interface

```
interface <interface_name>{
    // declare constant fields
    // declare methods that abstract
    // by default.
}
```

Java 8 Interface Improvement:
Since Java 8, interface can have default and static methods .

Internal Addition by the Compiler

The Java compiler adds public and abstract keywords before the interface method. Moreover, it adds public, static and final keywords before data members.



Example: interface

```
interface vehicle{
  int no_tyre=4;
 void Start();
 }
```

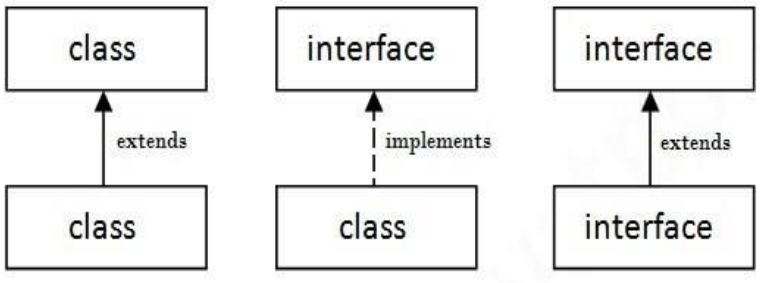
```
class car extends vechicle{
public void Start(){
   Sop("start by key");
  }
}
```

```
class bike extends vechicle{
public void Start(){
   Sop("start by kick");
  }
}
```

OOP's Features: interface

Java: The relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface, but a **class implements an interface**.



Example: interface

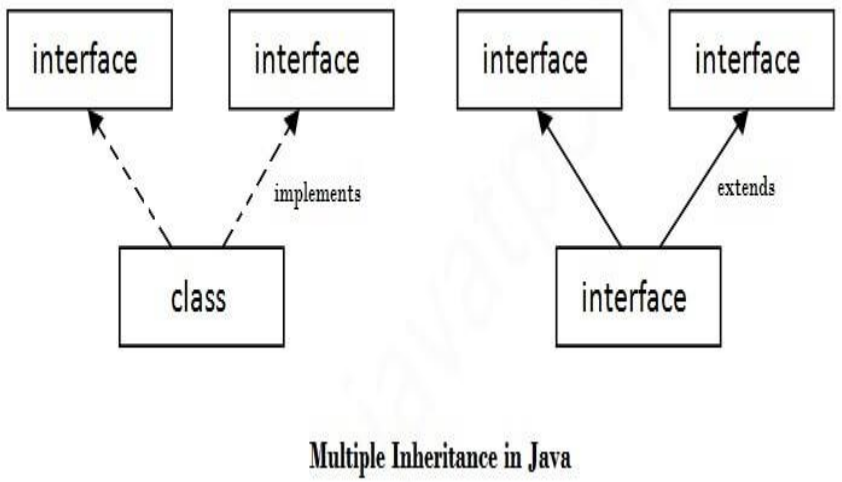
```
/Interface declaration: by first user
interface Drawable{
void draw();
}
//Implementation: by second user
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}

}
class Circle implements Drawable{
public void draw(){System.out.println("drawing circle");}
}
//Using interface: by third user
class TestInterface1{
public static void main(String args[]){
Drawable d=new Circle();//In real scenario, object is provide
d by method e.g. getDrawable()
d.draw();
}}
```

OOP's Features: interface

Java: 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: multiple interface

```
/interface Printable{
void print();
}
interface Showable{
void show();
}
class A7 implements Printable,Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
 }
}
```

Multiple inheritance is not supported through class in java, but it is possible by an interface,

OOP's Features: interface

Java: Interface inheritance

A class implements an interface, but one interface extends another interface.

Example:

```
interface Printable{
void print();
}
interface Showable extends Printable{
void show();
}
class TestInterface4 implements Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
TestInterface4 obj = new TestInterface4();
obj.print();
obj.show();
 }
}
```

Java 8: Default Method in Interface

Since Java 8, we can have method body in interface. But we need to make it default method.

```
interface Drawable{
void draw();
default void msg(){System.out.println("default method");}
}
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}

}
class TestInterfaceDefault{
public static void main(String args[]){
Drawable d=new Rectangle();
d.draw();
d.msg();
}}
```

OOP's Features: interface

Java 8: Static Method in Interface

Since Java 8, we can have static method in interface.

```
interface Drawable{
void draw();
static int cube(int x){return x*x*x;}
}
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}

}

class TestInterfaceStatic{
public static void main(String args[]){
Drawable d=new Rectangle();
d.draw();
System.out.println(Drawable.cube(3));
}}
```

Key Points: Interface

- Like **abstract classes**, interfaces **cannot** be used to create objects
- Interface methods do not have a body - the body is provided by the "implement" class
- On implementation of an interface, you must override all of its methods
- Interface methods are by default `abstract` and `public`
- Interface attributes are by default `public`, `static` and `final`
- An interface cannot contain a constructor (as it cannot be used to create objects)

Java: abstact class vs interface

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

Java: inheritance

Java: abstract class

Notes on prev program: In Shape class an abstract method is declared but its implementation is done in Circle class. Here Shape class must be abstract as an abstract method in it. However an abstract class may or may not have abstract method. As Shape is an abstract class no object is created of that class.

Java: interface

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.
The interface in Java is *a mechanism to achieve abstraction*.
There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.
In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.
Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Java: multiple nterface

```
package CSE1203;
interface FI{
  public void myMethod(); // interface method
 }
interface SI {
  public void myOtherMethod(); // interface method
 }
class DemoClass implements FI, SI {
public void myMethod() {
 System.out.println("From FI");
}
public void myOtherMethod() {
System.out.println("From SI");
}
}
public class First {
public static void main(String[] args) {
DemoClass myObj = new DemoClass();
    myObj.myMethod();
    myObj.myOtherMethod();
}
}
```

Notes Two interfaces FI and SI are implemented in DemoClass and there achieving multiple inheritance. Multiple inheritance is only achieved by the interface not by the class

THANK YOU