# Object oriented programming

**Lecture notes**
**In JAVA language.**

## JAVA BASIC STRUCTURE AND PRINTING STATEMENT

```
Class Demo
{
 static void main(String args[])
{
   System.out.println("Hallow world");
}


}
```

**OR** public class test

**OR** public static void main(String[] args)

Method

Class

**Main keywords**

public - anywhere or the access modifiers.

static- access level

main() - main method

void means - no return type

- ***there is only one main method.***

## PRINTING VAREABLE

```
Class Demo
{
 static void main(String args[])
{
     String name = "Raveen";
     int age = 10;
     System.out.println(name+""+age);
}
}
```

## ADDING NUMBERS

```
public class test{
   public static void main(String[] args)
   {
      int a=10,b=20,tot=0;
      tot=a+b;
      System.out.println(tot);
   }
}
```

## METHODS IN JAVA

### CREATING AND CALLING A FUNTION (METHOD TYPE 1)

```
public class test{
   public static void main(String[] args) -
   {
      printMessage();
```

Passing an arguments or data insert to the method as parameters to the main function for the process in the main function. The reason is, the codes inside of the main take to the array and passes to the main function for the execution. “Args” means the array name and [] use to denote the array

Function created

```
    }
    public static void printMessage() {
        System.out.println("Hallow to funtions!");
    }
}
```

* When crating methods always use simple letters and avoid spaces use _ or Camal cases.

There are 2 types of methods

* void methods – to execute all of the lines

* return type methods – to execute the returned value or selected line only

**CREATING AND CALLING A FUNTION (METHOD TYPE 2)**

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

        int result=0;
        result=getProduct();
        System.out.println(result);
    }
    public static int getProduct() {
        return 10*2;
    }
}
```

Using and calling of function

Function created

Returning of mathematical operation.

**USING OF PARAMETERS IN METHODS**

```
public class test{
    public static void main(String[] args)
    {
        getData("Raveen", 17);

    }
    public static void getData(String name,int age) {
        System.out.println(name+"\n"+age);
    }

}
```

Calling the function inserting arguments

Arguments

Declaring parameter variables

**CREATION OF OBJECTS**

```
class Student
{
    String sname;
    int age;
    double gpa;

    public static void main(String args[])
    {
        Student s1=new Student();
        Student s2=new Student();
        s1.sname="sumith";
        s1.age=22;
```

Creation of class for the object

Creating of the object

Or declaring object

Inserting data to the created object

Constructor of the class (present or absent)

Student s2=new Student();

Class name or custom data type

Object name or variable that we assign

```
        s1.gpa=3.5;
        System.out.println("The name - "+s1.sname+"\nThe age - "+s1.age+"\nThe GPA - "+s1.gpa);
        s2.sname="Pavan";
        s2.age=80;
        s2.gpa=3.4;
        System.out.println("The name - "+s2.sname+"\nThe age - "+s2.age+"\nThe GPA - "+s2.gpa);
    }
}
```

**<u>OBJECTS AND FUNTIONS (FUNTIONS EXECUTABLE WITH ONLY OBJECT)</u>**

```
class Student
{
    String sname;
    int age;
    double gpa;

    public static void main(String args[])
    {
        Student s1=new Student();
        s1.printMessage("Raveen",17,3.4);
        System.out.println("The name - "+s1.sname+"\nThe age - "+s1.age+"\nThe GPA - "+s1.gpa);

    }
    public void printMessage(String aname,int aAge,double Agpa)
    {
        System.out.println("Hallow!!");
        sname=aname;
        age=aAge;
        gpa=Agpa;

    }
}
```

**<u>COMMON STRUCTURE</u>**

```
S1.printMessage();
```

***Accessing to the function like accessing to a property of the object. – functions that works and calls with only objects.***

**USE OF THE FUNTION**

```
  Student s1=new Student();
        s1.printMessage("Raveen",17,3.4);
        System.out.println("The name - "+s1.sname+"\nThe age - "+s1.age+"\nThe GPA - "+s1.gpa);

    }
    public void printMessage(String aname,int aAge,double Agpa)
  {
     System.out.println("Hallow!!");
     sname=aname;
     age=aAge;
     gpa=Agpa;

  }
```

**USING OF 2 CLASSES**

```
package   lessons;

/**
 * lesson
 */
 public class raveen{
   int lengh,width;

   void calcArea(int lengh,int width){
   int area = lengh*width;
   System.out.println(area);

   }
}
/**
 * area
 */

class Main{

   public static void main(String[] args) {
      raveen a1= new raveen();
      raveen a2= new raveen ();
      a1.calcArea(5, 10);
      a2.calcArea(5, 20);

   }
}
```

Class 1

Class 2 that includes the main function

**Constructors**

```
public class constructors {

    String sname;//instance vareable

    int age;
```

Special functions that is capable of call itself when an object is created relevant to the class that the constructor is created.

```
    double gpa;
    constructors()
    {
    sname="Kaveen";
    age=17;
    gpa=3.3;
    }
```

Created constructor

```
public static void main(String[] args)
 {
    constructors s1=new constructors();
    System.out.println(s1.sname);
 }

}
```

Created object

class

Constructor that calls automatically within the object

Using the object like normal

**INSERING DATA USING CONSTRUCTORS**

```
public class constructors2
{
    int length;
    int width;
    int height;

    constructors2(int Alength,int Awidth,int Aheight)
    {
        length=Alength;
        width=Awidth;
        height=Aheight;
    }
    void getVolume()
```

Memory parameters to insert or access the class properties with using memory parameters.

```
    {
       int volume=length*height*width;
       System.out.println(volume);

    }
    public static void main(String[] args) {
        constructors2 c1=new constructors2(10, 20, 30);
        c1.getVolume();

    }

}
```

Inserting data according to the order that the parameters are defined while calling the constructor.

**<u>"THIS" OPERATOR IN CONSTRUCTORS</u>**

- Use – to separate the parameters with variables.
- This. pname – equals to the class variable not the parameter.

```
public class constructors2
{
    int length;
    int width;
    int height;
    constructors2(int length,int width,int height)
    {
        this.length=length;
        this.width=width;
        this.height=height;
    }
    void getVolume()
    {
       int volume=length*height*width;
       System.out.println(volume);

    }
    public static void main(String[] args) {
        constructors2 c1=new constructors2(10, 20, 30);
        c1.getVolume();

    }
 }
```

Base class

Equals

Separate variables

Constructor

Object functions

Main method

## ACCESS MODIFIERS

| Access modifiers | information | Non-access modifiers | Information |
|---|---|---|---|
| Public | Information is public | Final | Attributes that can’t overridden |
| Private | Access within the class only | Static | Attributes in a class without creating object |
| Protected | Ability to access only when inherited | Abstract | Use to hide the data |
| Default | Visible within the package | Transient | Modifier used in serialization |
| | | Synchronized | ensure only one thread can access a shared resource at a time |
| | | Volatile | |

1) Public

- A class, method or variable can be declared as public and it means that it is accessible from any class.
- A public class, method, or variable can be accessed from any other class at any time.

```
public class acess_modifiers {
  public String name="Raveen";
  public String lname="Jayasanka";
  public String email="raveenjayasanka4@gmail.com";
  public int age=20;
}
```

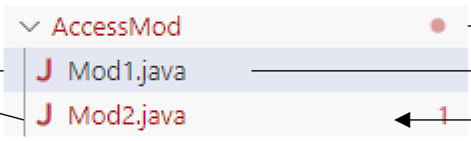
Properties with public

```
public class acess_modifiers2 {
  public static void main(String[] args) {
    acess_modifiers student = new acess_modifiers();
    System.out.println(student.name);
    System.out.println(student.age);
    System.out.println(student.email);
  }

}
```

Ability to access the data in the “Public” class properties in the same file

And also, we have the access like below,



Folder

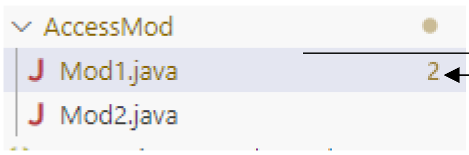
Java files

Accessible to the class properties in “Mod1.java” file to “Mod2.java” file

2) Private

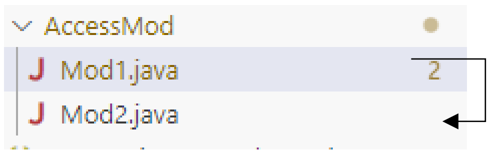
- Allows a variable or method to only be accessed in the class in which it was created.

```
package AccessMod;
public class Mod1 {
  private String name;
  private int age;
```



Accessible is limited within this file only. But we can access the properties in the **file only**

```
        private double gpa;
        public static void main(String[] args) {
            Mod1 student1 = new Mod1();
            student1.name="Jhonn";
            student1.age=20;
            student1.gpa=3.3;
            System.out.println(student1.name);
        }
    }
```

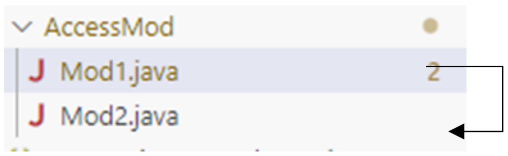


But we can't access the properties within other files.

3) Protected

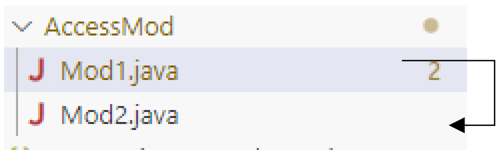
- Unlike "Private", we have the access to the programs in the same folder. But this condition available only for the inherited classes from one class to another class in different file.

When using "Private" modifier



We can't access

When using "Public" modifier



Has got the access for the properties in the "Mod1.java" file from "Mod2.java" file, **Only** there is an inheritance from "Mod1.java" to "Mod2,java"

**The protected keyword is an access modifier used for attributes, methods and constructors making them accessible in the same package and subclasses.**

| private | Protected |
|---|---|
| It can only be invoked from within the implementation of a class or its subclass | This can be invoked on any instance of the class, when that class is inherited to the accessing class. |

## USING OF "SUPPER" KEYWORD IN CLASSES

- The supper keyword refers to superclass (Parent) properties in another class.
- This is use to call superclass methods and properties of the parent class.

- The main benefit of this keyword - To avoid confusion between superclass and the subclasses that have the methods in the same name.

**Example 1 – for accessing the functions**

```
package AccessMod;

public class Mod1 {
   public void printing1()
   {
      System.out.println("Hallow to codes!!");
   }
}
class Mod2 extends Mod1 {
   public void printing2()
   {
      System.out.println("Hallow again!!");
      super.printing1();
   }
   public static void main(String[] args) {
      Mod2 m1 = new Mod2();
      m1.printing2();
   }

}
```

Inherits from Mod1 to Mod2 to get access the properties

Reference to " Mod2" class

Reference to "Mod1" class. So, use "super" keyword to get the access the function in the "Mod1" class which inherits to "Mod2" class. (For calling)

All need to call the "printing2" function in "mod2" Class.

**Example 2 – to access the properties**

```
public class day5 {
   String colour="White";
}
class dog extends day5 {

   String colour="black";
   void printColour()
   {
      System.out.println(colour);
      System.out.println(super.colour);
   }
}
class TestSupper {
   public static void main(String args[])
 {
   dog d=new dog();
   d.printColour();
 }
}
```

Accessing the properties of the day5 class and also It must be inherited the class that we going to access.

**INHERITANCE**

```
package AccessMod;
public class Mod1 {
  public void printing1()
  {
    System.out.println("Hallow to codes!!");
  }
}
class Mod2 extends Mod1 {
  public void printing2()
  {
    System.out.println("Hallow again!!");
    super.printing1();
  }
  public static void main(String[] args) {
    Mod2 m1 = new Mod2();
    m1.printing2();
  }
}
```

Inherits form "Mod1" to "Mod2"

- Inheritance in java is the method to create a hierarchy between classes by inheriting from other classes.
- The inheritance concept is two categories,
  1) Subclass (child) – the class inherits from another class (parent class)
  2) Supper class (parent) – the class being inherited from
- We use "extends" keyword to inherit classes.
- Also due to security reasons, we can use access modifiers to control the access between the classes.
- Main benefit of using inheritance – For code reusability (reuse the attributes and methods of an existing class when creating a new class).

**THE "Final" KEYWORD**

- If you make final - we can't inherit the "final" class to another class (to atop the inheriting process of a class)
- Main benefit – to enhance the security.

```
package AccessMod;
final class Mod1 {
  public void printing1()
  {
    System.out.println("Hallow to codes!!");
  }
}
```

```
  class Mod2 extends Mod1 {
     public void printing2()
     {
        System.out.println("Hallow again!!");
        super.printing1();
     }
     public static void main(String[] args) {
        Mod2 m1 = new Mod2();
        m1.printing2();
     }
  }
```

We can't access or inherit the "Mod1" class. Because the "Mod1" class is final.

**STATIC VAREABLES IN JAVA**

Whenever a variable is declared as static, this means there is only one copy of it for the entire class, rather than each instance having its own copy. A static method means it can be called without creating an instance (object) of the class.

We don't need to create an object according to the class, that the variable is created in the class to access the variable or to modify the variable.

**Example**

```
package AccessMod;
 class Mod1 {
   public static int num =10;
}
 class Mod2 extends Mod1 {
   public static void main(String[] args) {
     num++;
    System.out.println(num);
   }
}
```

We don't need to create an object to access the variable. But we must have to create an object to access the variable when we don't use "static".

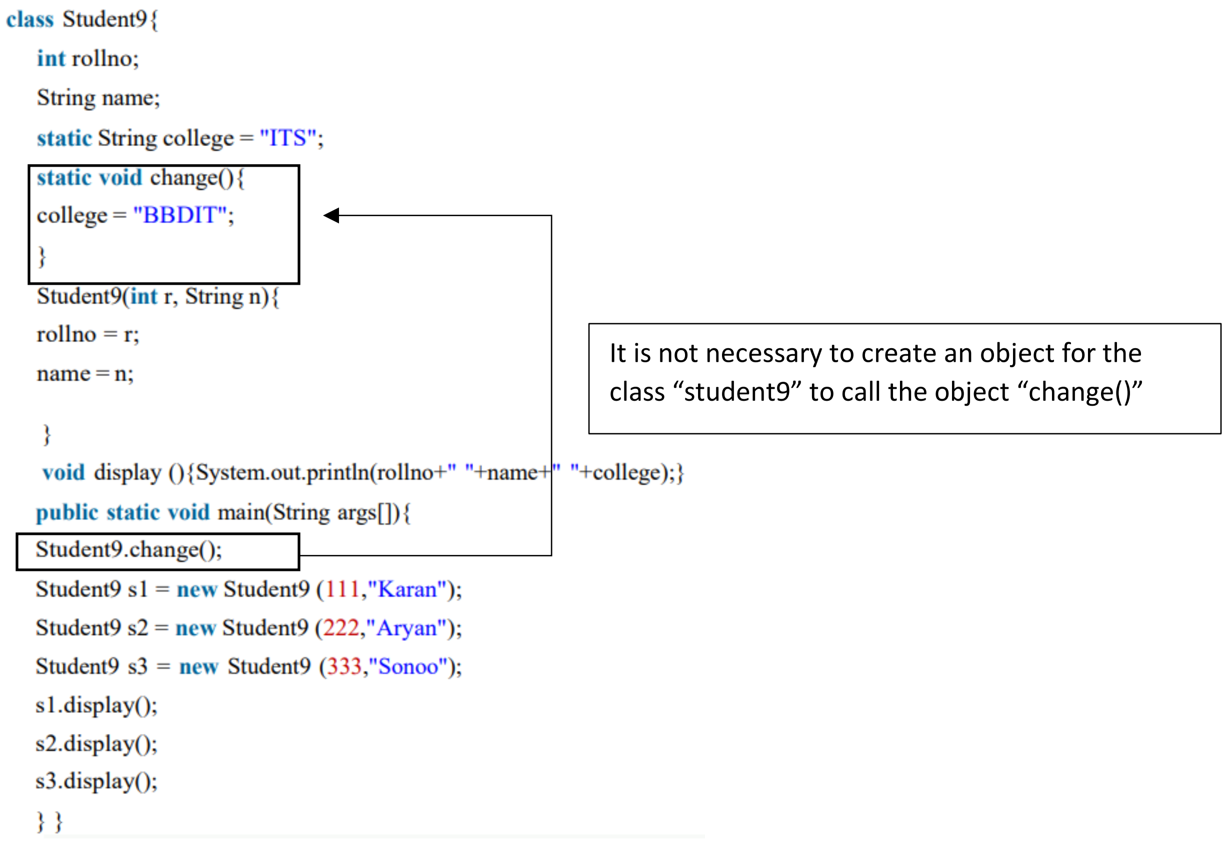
```
package AccessMod;
 class Mod1 {
    int num =10;
}
 class Mod2 extends Mod1 {
    public static void main(String[] args) {
       //num++;
      System.out.println(num);
    }
}
```

We can't access the variable according to this way.

## STATIC METHODS IN JAVA

A static method can be invoked without the need for creating an instance (object) of a class. static method can access static data member and can change the value of it without creating an object.

Static methods are methods that belong to the class rather than to any specific instance of the class. Static methods can be called directly on the class itself without needing to create an instance of the class.



```
class Student9{
    int rollno;
    String name;
    static String college = "ITS";
    static void change(){
    college = "BBDIT";
    }
    Student9(int r, String n){
    rollno = r;
    name = n;

     }
     void display (){System.out.println(rollno+" "+name+" "+college);}
    public static void main(String args[]){
    Student9.change();
    Student9 s1 = new Student9 (111,"Karan");
    Student9 s2 = new Student9 (222,"Aryan");
    Student9 s3 = new Student9 (333,"Sonoo");
    s1.display();
    s2.display();
    s3.display();
    } }
```

It is not necessary to create an object for the class “student9” to call the object “change()”

## SUMMERY OF ACCESS MODIFIERS

| Access Modifier | within class | within package | outside package by subclass only | outside package |
|---|---|---|---|---|
| Private | Y | N | N | N |
| Default | Y | Y | N | N |
| Protected | Y | Y | Y | N |
| Public | Y | Y | Y | Y |

## DEFAULT ACCESS MODIFIER

- In this modifier, we don't use any special keyword.
- This is also called as, package private.
- This means that, all the members are visible within the same package.
- But this isn't accessible from  other packages.

**Example**

```
package AccessMod;
 class Mod1 {
    int num;
}
 class Mod2 {
   public static void main(String[] args) {
      Mod1 m1=new Mod1();
      m1.num=30;
   }
}
```

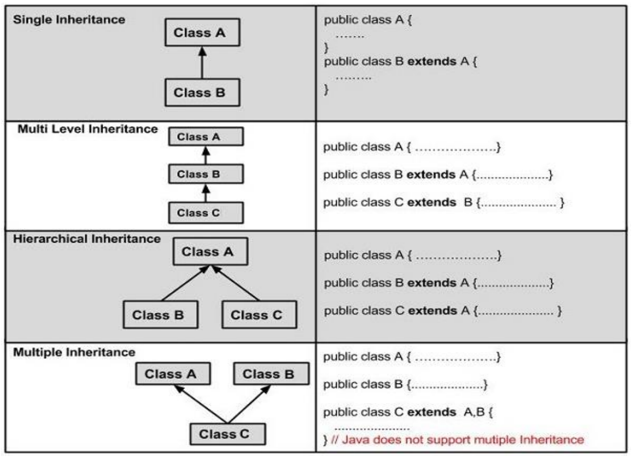
## PUSRPOSE OF INHERITENCE

**Code Reusability:** The code written in the Superclass is common to all subclasses. Child classes can directly use the parent class code.

**Method Overriding:** Method Overriding is achievable only through Inheritance. It is one of the ways by which Java achieves Run Time Polymorphism.

**Abstraction:** The concept of abstract where we do not have to provide all details is achieved through inheritance. Abstraction only shows the functionality to the user.

## TYPES OF JAVA INHERITENCE



```
public class A {
   .......
}
public class B extends A {
   .........
}
```

```
public class A { ...................}
public class B extends A {....................}
public class C extends  B {..................... }
```

```
public class A { ...................}
public class B extends A {....................}
public class C extends A {..................... }
```

```
public class A { ...................}
public class B {....................}
public class C extends  A,B {
   .....................
} // Java does not support mutiple Inheritance
```

**POLYMORPHISM IN JAVA**

- Polymorphism allows us to perform a single action in different ways.
- In other words, polymorphism allows you to define one interface and have multiple implementations.
- Polymorphism is two types,
    - Compile-time polymorphism
    - Runtime polymorphism

Compile time polymorphism / overloading

- It is also known as **static polymorphism.** This type of polymorphism is achieved by **function overloading** or operator overloading.
- **When there are multiple functions with the same name but different parameters then these functions are said to be overloaded.** Functions can be overloaded by changes in the number of arguments or/and a change in the type of arguments.

  **Example**

```
package AccessMod;
public class poly {
  static int printing(int num1,int num2)
  {
     int tot=num1+num2; //why we can enter same vareables
     return tot;
  }
  static double printing(double num1,double num2)
  {
     double tot= num1+num2;
     return tot;
  }
  public static void main(String[] args) {
     int result=printing(10, 20);
     double result2=printing(23.56, 22.45);
     System.out.println(result2);
     System.out.println(result);
  }
}
```

Same function name, same variables and same memory parameters but different parameter data types

Calling in the main method

- With **method overloading**, multiple methods can have the same name with different parameters.

Runtime compilation or polymorphism / Overriding functions

- It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at **Runtime.**
- This type of polymorphism is achieved by **Method Overriding**. Method overriding, on the other hand, **occurs when a derived class has a definition for one of the member functions of the base class.** That base function is said to be overridden.
- Functions with the same **name but different tasks or processes**. OR **we can change the process of a function in a class with a different class while inheriting the classes.**
- Main condition – must inherit to a different class to change the functions with the same name.

**Example**

```
package AccessMod;
public class poly{
  void printing()
  {
    System.out.println("Hallow world!!");
  }
}
 class printing2 extends poly {
  void printing()
  {
    System.out.println("Hallow to codes");
  }
  public static void main(String[] args) {
    printing2 p1=new printing2();
    p1.printing();
    poly p2 = new poly();
    p2.printing();
  }
}
```

Inherits from "poly" to "printing2" so we can make different processes but with same function name. but for the access, all we have to do is, create the separate objects but call those functions with the same name to obtain different results.

Must create an object with "printing2" class to obtain the different process

Must create an object with "poly" class to obtain the different process

Same name

## ADVANTAGES OF POLYMORPHISM IN JAVA

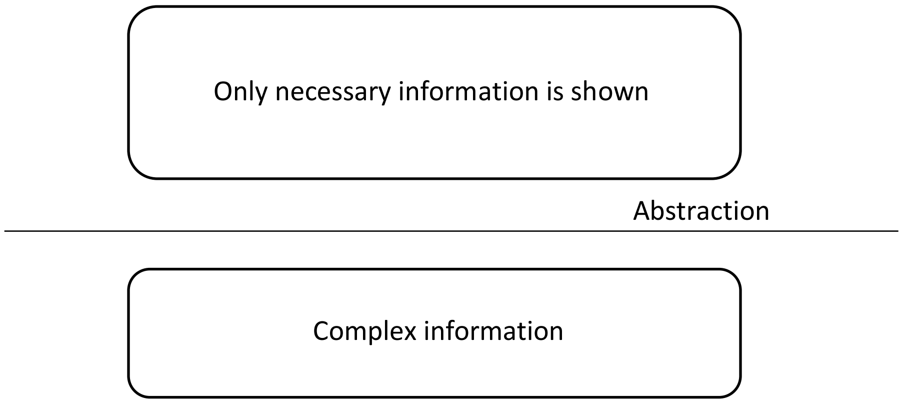
- Increases code reusability by allowing objects of different classes to be treated as objects of a common class.
- Improves readability and maintainability of code by reducing the amount of code that needs to be written and maintained.
- Supports dynamic binding, enabling the correct method to be called at runtime, based on the actual class of the object.
- Enables objects to be treated as a single type, making it easier to write generic code that can handle objects of different types.

## DIFFERENCE BETWEEN OVERLOADING AND OVERIDING

| Overloading | Overriding |
|---|---|
| • Use to increase readability of the program.<br>• Performed within class<br>• Parameter must be different | • Used to provide the specific implementation of the method that is already provided by its supper class.<br>• Occurs in 2 classes that have a relationship.<br>• Parameter must be same. |

**DATA ABSTRACTION**

- Data abstraction is the process of hiding certain details and showing only essential information to the user.
- Abstraction can be achieved with either abstract classes or interfaces.



- In Java, abstraction is achieved by interfaces and abstract classes. We can achieve 100% abstraction using interfaces.
- Abstract class: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- **Abstract methods don't have a body.**
- In general, **if a class contains at least one incomplete method (in programming terms, one abstract method**), the class itself is an **abstract class.** The term **abstract method tells you that the method has a declaration (or signature) but no implementation.**

COMMEN STRUCTURE

```
abstract class Bike {
  abstract void run();
}
```

This class contains an incomplete method or not implemented so, this is an abstract class

- In other words, you can think of abstract members as virtual members without a default implementation.
- A class that contains at least one abstract method must be marked as an abstract class.
- The subclass (the class that inherits the abstract class) must finish the incomplete task; that is, a subclass needs to provide the complete method body for the abstract method, but **if it fails to provide that, the subclass itself will be marked as another abstract class.**

EXAMPLE

```
package Abstraction;
abstract class Bike {
   abstract void run();
}
```

```
class honda extends Bike{
   void run(){
      System.out.println("running safely..");
   }
```

Inherit the class from “bike” to “honda” to complete the abstract method or to create the body of the “run” method

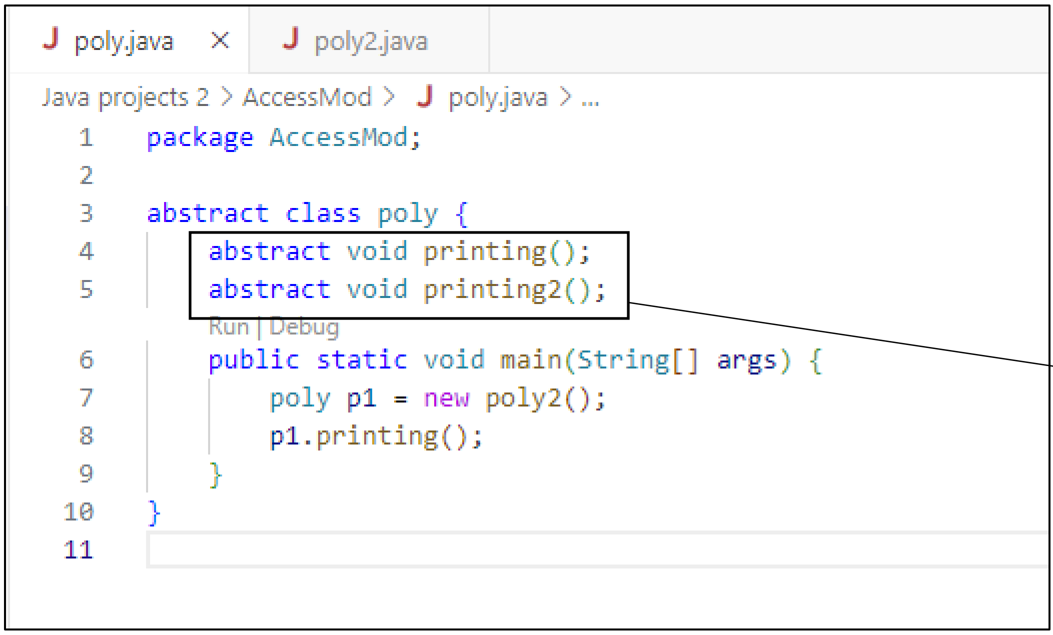
EXAMPLE 2 – formation 2

```
package AccessMod;

abstract class poly {
   abstract void printing();
   abstract void printing2();
   public static void main(String[] args) {
      poly p1 = new poly2();
      p1.printing();
   }
}

  class poly2 extends poly {
   void printing()
   {
      System.out.println("Hallow world!!");
   }
   void printing2()
   {
      System.out.println("Hallow to codes!!");
   }
   }
 }
```

PRACTICAL USAGE

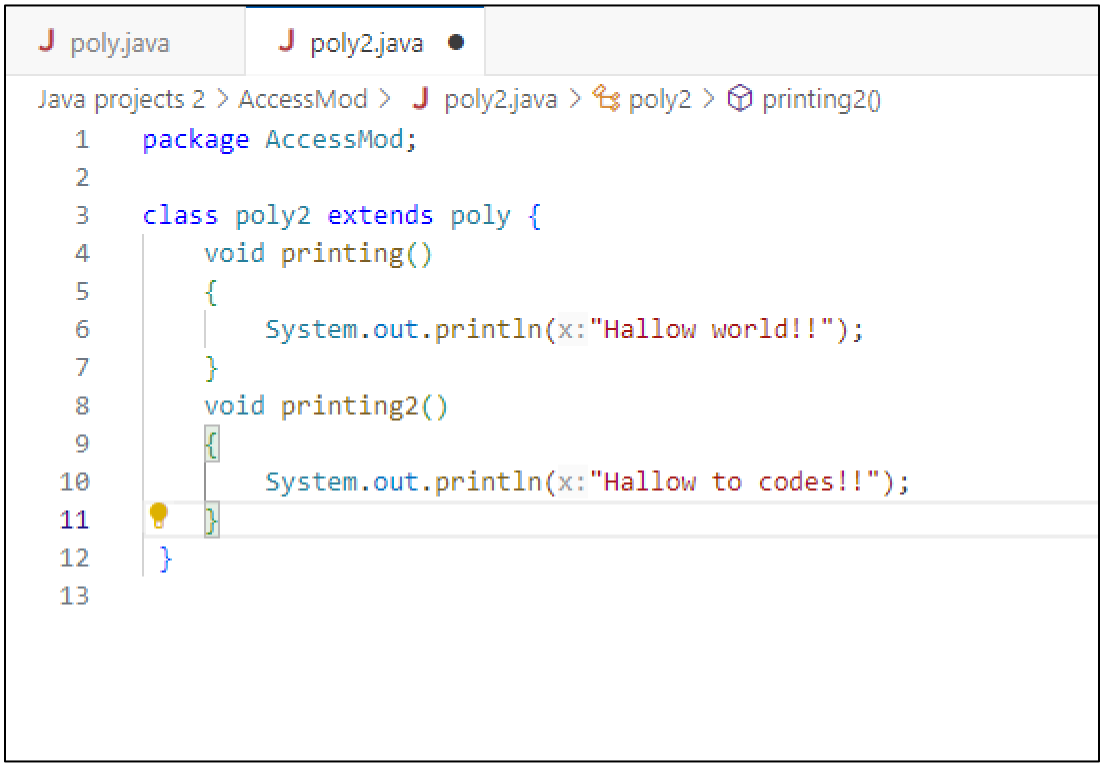


```
package AccessMod;

abstract class poly {
    abstract void printing();
    abstract void printing2();
    public static void main(String[] args) {
        poly p1 = new poly2();
        p1.printing();
    }
}
```

Main methods that contain visible part to the user

Function parts that are hided information from the user.



poly.java
poly2.java
Java projects 2 > AccessMod > poly2.java > poly2 > printing2()

```
package AccessMod;

class poly2 extends poly {
    void printing()
    {
        System.out.println(x:"Hallow world!!");
    }
    void printing2()
    {
        System.out.println(x:"Hallow to codes!!");
    }
}
```

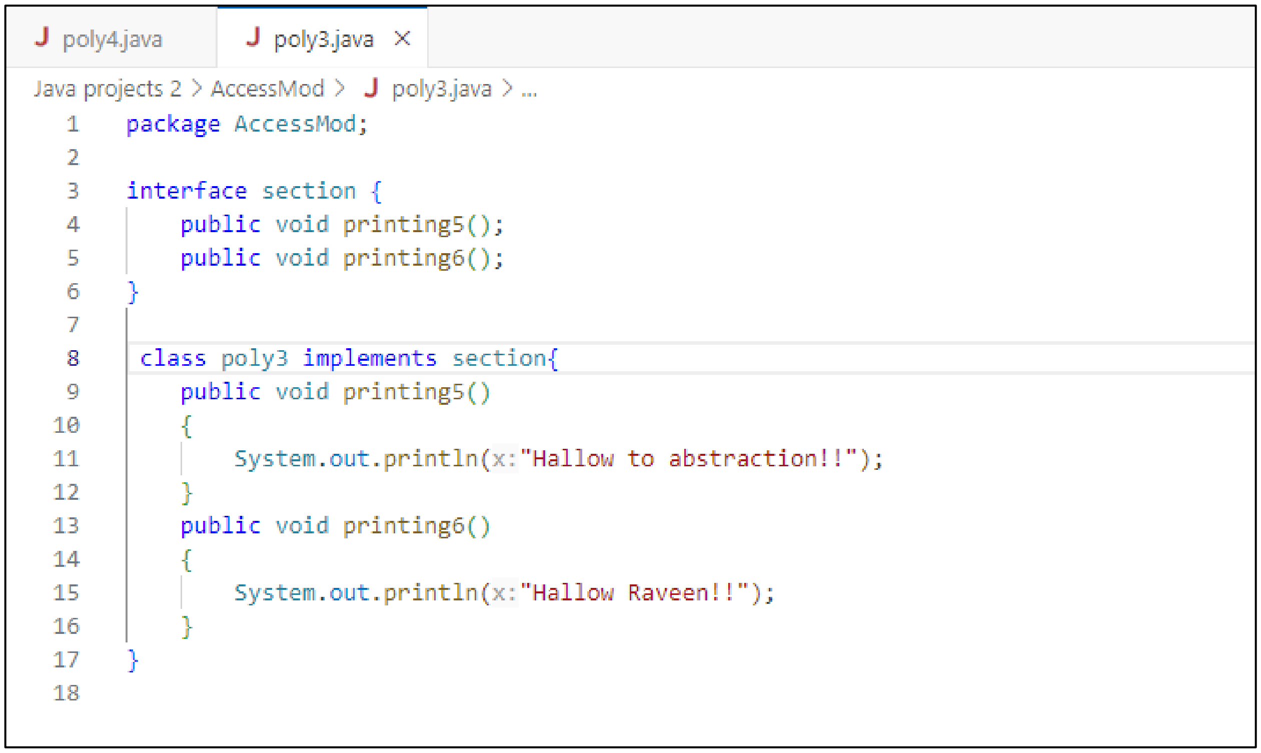
Data and the processes that hide from the user and for the abstract bodies. The part that the user cannot see

## INTERFACES IN JAVA

- The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not the method body.

| Abstraction | Interfaces |
|---|---|
| To access the abstracted part, we have to create a sub class in each time when you are going to access the abstract function | But in interfaces, hide the functions that I need to hide but allow access or calling the function in implement part without creating a subclass for each function that I need to access. |

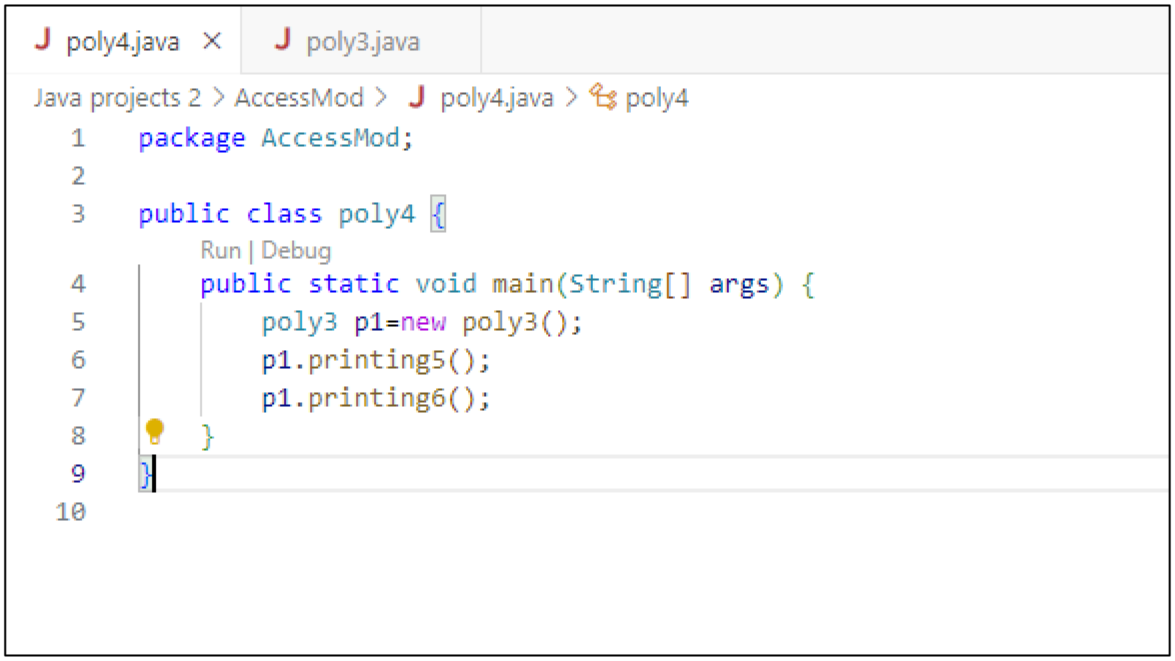
EXAMPLE



```java
package AccessMod;

interface section {
    public void printing5();
    public void printing6();
}

 class poly3 implements section{
    public void printing5()
    {
        System.out.println(x:"Hallow to abstraction!!");
    }
    public void printing6()
    {
        System.out.println(x:"Hallow Raveen!!");
    }
}
```



```
package AccessMod;

public class poly4 {
    public static void main(String[] args) {
        poly3 p1=new poly3();
        p1.printing5();
        p1.printing6();
    }
}
```

**CONSTRUCTOR TYPES IN OOP**

- There are 2 types of constructors in OOP they are,
    - Default constructors - The constructor without the memory parameters.
    - Parametric constructors - The constructor with memory parameters.
- Examples – For Default constructor

  constructorName(){

  }
- Example – For Parametric constructor

  constructorName(datatype parameter1, datatype parameter2……..){

  }

**USING OF DEFAULT AND PARAMETRIC CONSTRUCTORS AT THE SAME TIME**

- Main use – to use 2 different constructors of the same class to create 2 different processes.

  **EXAMPLE**

```
class student2 {
  String name,DOB,address;
  int age;
  student2(String name,String DOB,String address,int age){
   this.name=name;
   this.DOB=DOB;
   this.address=address;
   this.age=age;
```

$\Big\}$ Constructor 1 with parameters to enter data.

```
  }
  student2(){                                   Constructor type 2 without memory
   System.out.println("No data added!!");       parameters
  }
}
public class student {
   public static void main(String[] args) {
      student2 s1=new student2("Jhonn", "2005-02-15", "California", 20);
      student2 s2=new student2();                                          2 different outputs
   }
}
```

**OVERLOADING OF CONSTRCUTORS**

- In this case, there are different parameters of the 2 different constructors.

**EXAMPLE**

```
class student2 {
  String name,DOB,address;
  int age,phone;
  student2(String name,String DOB,String address,int age){
   this.name=name;
   this.DOB=DOB;                                           Different parameters
   this.address=address;
   this.age=age;
  }
  student2(int phone){
   this.phone=phone;
  }
}
public class student {
   public static void main(String[] args) {
      student2 s1=new student2("Jhonn", "2005-02-15", "California", 20);
      student2 s2=new student2(778733391);
   }
}
```

**ENCAPSULATION IN OOP**

In object-oriented programming, you do not allow your data to flow freely inside the system. Instead, you wrap the data and functions into a single unit (i.e., in a class). The purpose of encapsulation is at least one of the following:

• inserting restrictions in place so that the components of an object cannot be accessed directly
• Binding the data with methods that will act on that data (i.e., forming a capsule).

**<u>GETTERS AND SETTERS</u>**

- Main use – to access the “private” attributes.
- Main benefits of using getters and setters
  • You can provide controlled access to your data.
  • You can make your class variable either read-only or write-only. When you provide the getter method only, you can only get the value of the private variable, so it becomes read-only. Similarly, when you provide only the setter method, you make the variable write-only.

**EXAMPLE**

```
public class lesson1 {
   private String name;
   public String getage()          ┐
   {                               ├ Getter – to return the value only at this stage.
      return name;                 │
   }                               ┘
   public void setAge(String name)   ┐
   {                                 ├ Setter – to access the data or
      this.name=name;                │ inserting data.
   }                                 ┘
}
 class section {
   public static void main(String[] args) {
      lesson1 l1=new lesson1();
      l1.setAge("Sanuja");
      System.out.println(l1.getage());
   }
}
```

**<u>PACKAGES IN JAVA</u>**

- A set of classes and interfaces grouped together are known as Packages.
- Every class is a part of a certain package.
- When we want to use an existing class, we have to add that package into the java program.
- Benefits,
  - The packages organize the group of classes into a single API unit.
  - Ability to control naming conflicts.
  - Ability to Reuse the packages.
- There are 2 types of packages They are, - User defined Packages
  In built packages

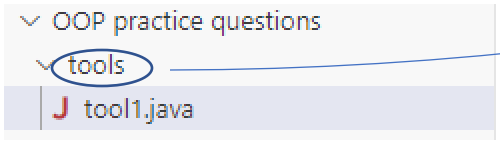
EXAMPLE

Name of the package(folder)

```
package tools;

public class tool1 {
    public void process1(){
        System.out.println("Test passed!!");
        process3();
    }
    public static void process3(){
        System.out.println("Test3 passed!!");
    }
}
```

Processes or classes that are included inside the package.



IMPROTING PACKAGES

```
import tools.tool1;
public class test{
    public static void main(String[] args) {
        tool1 t1=new tool1();
        t1.process1();

    }
}
```