Lab Manual

**OOP**

**Laboratory 08:**

**Statement Purpose:**

At the end of this lab, the students should be able to:

* Create Constructors
* Implement Method Overloading
* Implement Argument Passing (Pass by value and Pass by reference)
* Use Access Modifiers (Public, Private)

**Constructor:**

A constructor initializes an object immediately upon creation. It has the same name as the class in which it resides and has no return type, not even void.

**Using Constructor to initialize an object:**

**Note**: Save this program as DemoBox.java. The newly added code is shown as bold.

//Here Box uses a constructor to initialize the dimensions of a box.

**public** **class** Box {

**double** width;

**double** height;

**double** depth;

//This is constructor for Box.

**Box(){**

**System.*out*.println("Constructing Box");**

**width=10;**

**height=10;**

**depth=10;**

**}**

**//compute and return volume**

**double volume()**

**{**

**return width\*height\*depth;**

**}**

**}**

**class** DemoBox{

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

{

// Declare, allocate and initialize Box objects

**Box mybox1=new Box();**

**Box mybox2=new Box();**

**double** vol;

//get volume of first box

vol=mybox1.volume();

System.*out*.println("Volume is " + vol);

//get volume of second box

vol=mybox2.volume();

System.*out*.println("Volume is " + vol);

}

}

**Output:**

Constructing Box

Constructing Box

Volume is 1000.0

Volume is 1000.0

**Using Parameterized Constructor to initialize Box objects:**

In previous example, all boxes have the same dimensions. Box objects of various dimensions can be created by adding parameters to the constructor.

//Box uses a parameterized constructor to initialize the dimensions of a box.

**public** **class** Box {

**double** width;

**double** height;

**double** depth;

//This is parameterized constructor for Box.

Box(**double** w, **double** h, **double** d){

width=w;

height=h;

depth=d;

}

//compute and return volume

**double** volume()

{

**return** width\*height\*depth;

}

}

**public** **class** BoxDemo6{

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

{

// Declare, allocate and initialize Box objects

Box mybox1=**new** Box(10,20,15);

Box mybox2=**new** Box(3,6,9);

**double** vol;

//get volume of first box

vol=mybox1.volume();

System.*out*.println("Volume is " + vol);

//get volume of second box

vol=mybox2.volume();

System.*out*.println("Volume is " + vol);

}

}

**Output:**

Volume is 3000.0

Volume is 162.0

**Argument Passing:**

There are two ways that a computer language can pass an argument to a method.

**Call-by-value:** This method copies the value of an argument into the formal parameter of the method. Therefore changes made to the parameter of the method have no effect on the argument.

**Call-by-reference:** In this method, a reference to an argument (not the value of the argument) is passed to the parameter. Inside the method, this reference is used to access the actual argument specified in the call. This means that changes made to the parameter will affect the argument used to call the method.

**Aim: Develop programs based on variation in methods i.e. passing by value, passing by reference.**

**Passing By Values**

//swapping example

**public** **class** c1 {

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

**int** a = 30;

**int** b = 45;

System.*out*.println("Before swapping, a = " +

a + " and b = " + b);

// Invoke the swap method

*swapFunction*(a, b);

System.*out*.println("\n\*\*Now, Before and After swapping values will be same here\*\*:");

System.*out*.println("After swapping, a = " +

a + " and b is " + b);

}

**public** **static** **void** swapFunction(**int** a, **int** b) {

System.*out*.println("Before swapping(Inside), a = " + a

+ " b = " + b);

// Swap n1 with n2

**int** c = a;

a = b;

b = c;

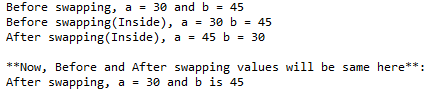
System.*out*.println("After swapping(Inside), a = " + a

+ " b = " + b);

}

}

**Output:**



**Passing By Reference:**

//PassbyReference Example

**public** **class** c2 {

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

Car car = **new** Car("BMW");

System.*out*.println("Brand of Car Inside main() before: "+ car.brand);

*printBrand*(car);

System.*out*.println("Brand of Car Inside main()after: "+ car.brand);

}

**public** **static** **void** printBrand(Car car)

{

car.brand = "Maruti";

System.*out*.println("Brand of Car Inside printBrand(): "+car.brand);

}

**private** **static** **class** Car

{

**private** String brand;

**public** Car(String brand){

**this**.brand = brand;

}

}

}

**Output:**

Brand of Car Inside main() before: BMW

Brand of Car Inside printBrand(): Maruti

Brand of Car Inside main()after: Maruti

**Method Overloading:**

* In Java, it is possible to define two or more methods within the same class that share the same name, as long as their parameter declarations are different. These methods are said to be **overloaded**.
* When an overloaded method is called, Java uses the type and/or number of arguments to determine which version of the overloaded method to actually call. Thus, overloaded methods must differ in the type and/or number of their parameters. They may have different return types, since return types do not play a role in overload resolution.

**Aim: Develop programs based on overloading methods.**

**Example 1:**

//calculation(method overloading example)

**public** **class** c3 {

**void** sum(**int** a,**int** b)

{

System.*out*.println(a+b);

}

**void** sum(**int** a,**int** b,**int** c)

{

System.*out*.println(a+b+c);

}

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

{

c3 obj=**new** c3();

obj.sum(10,10,10);

obj.sum(20,20);

}

}

**Output:**

30

40

**Example 2:**

//Method Overloading Example

**public** **class** c5 {

**int** addition(**int** i, **int** j)

{

**return** i + j ;

}

String addition(String s1, String s2)

{

**return** s1 + s2;

}

**double** addition(**double** d1, **double** d2)

{

**return** d1 + d2;

}

}

**class** AddOperation2 {

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

{

c5 sObj = **new** c5();

System.*out*.println(sObj.addition(1,2));

System.*out*.println(sObj.addition("Hello ","World"));

System.*out*.println(sObj.addition(1.5,2));

}

}

**Output:**

3

Hello World

3.5

**Access Control:**

Access Control is a way to limit the access others have to your code.

**Same package** - can access each other’s variables and methods, except for private members.

**Outside package** –

1. Can access public classes.
2. Can access members that are public.
3. Can access protected members if the class is a subclass of that class.

Same package - use package keyword in first line of source file, or no package keyword and in same directory.

**Keywords -**

1. public - outside of package access.

2. [no keyword] - same package access only.

3. protected - same package access. Access if class is a subclass of, even if in another package.

4. private - same class access only.

**Aim: Develop program to demonstrate private and public access modifiers.**

**Example:**

//This program demonstrate the difference between public and private access modifiers

**public** **class** c8 {

**int** a;

**public** **int** b;

**private** **int** c;

**void** setc(**int** i)

{

c=i;

}

**int** getc()

{

**return** c;

}

}

**class** Accessc8{

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

{

c8 c=**new** c8();

c.a=10;

c.b=20;

c.c=100;

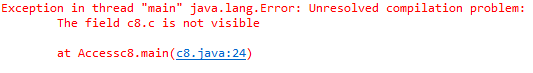
c.setc(100);

System.*out*.println("a, b, and c:" + c.a + " " + c.b + " " + c.getc());

}

}

**Output:**



If this statement c.c=100; is commented, then the following output will be shown.

**Output:**

a, b, and c:10 20 100

**Lab Tasks:**

**Task 1: Marks: 5**

**Make a class named Student. Declare a field named name of type String in Student class. Make it Private field. Define two methods getName() and setName() in class Student. The return\_type of getName is String and Access modifier is public. The return\_type of setName is Void and Access modifier is public. In main class make an object of Student class. Using method setName(), set any name in name field, and using getName method print the name. Then using object (don’t use setName() method), set the name field with any name, and print the name using getName method. Show output. If you get an error then tell its reason explicitly.**

**Task 2: Marks: 5**

**Make a class Program1 with four fields x=5, y=5, z=0 and result=0 of type double. In this class defines a public method named calculate which calculates**

1. **Sum //No parameter method (sum of x and y only)**
2. **Division of two numbers //two parameters of type double**
3. **Multiplication of three numbers //three parameters of type int**
4. **Subtraction of two numbers //two parameters of type int**
5. **Square of a number //one parameter of type double**

**All methods have return type void except one which calculates square. It should return square of a number so it’s return type would be double. All methods have public accessibility.**

**In main class make an object and call all methods with different arguments to show method overloading.**