private : [Within the same class]

---

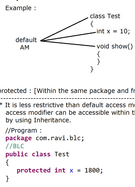
* private is the **most restrictive** access modifier because the member declared with private access modifier will be accessible from the **same class only.**
* In order to achieve data hiding concept, We should declare our non static variables with private modifier.
* An outer class we cannot declare with private and protected access modifier. An outer class we can declare with **public, abstract, final, sealed and non-sealed modifiers.**

default : [Within the same package]

---

* It is less restrictive access modifier than private because the members which are having default access modifier will be accessible within the same package only.
* If a programmer does not provide any kind of access modifier before name of the class and before the member of the class then by default the access modifier is default. It is also known as private-package access modifier.
* As far as its accessibility is concerned, default members are accessible **within the same package only.**

Example :



protected : [Within the same package and from another package also by using Inheritance]:

---

* It is less restrictive than default access modifier because the member declared with protected access modifier can be accessible within the same package, even though from another package by using Inheritance.

```
//Program :
package com.ravi.blc;
//BLC
public class Test
{
    protected int x = 1800;
}

 package com.ravi.elc;
 import com.ravi.blc.Test;
 public class ELC extends Test //Inheritance
 {
    public static void main(String[] args)
    {
       ELC e = new ELC();
       System.out.println(e.x);
    }
 }
```

public [No restriction, Can be accessible from every where]

---

* It is an access modifier which does not provide any kind of restriction because the members which are declared with public access modifiers can be accessible from everywhere.
* According to Object Oriented rule, We should always declare non static variable with private access modifier where as we should declare the classes and methods with public access modifier.
* We can also declare methods with **private access modifier** if that method we are using for internal validation which are known as **Helper Method.**

Access Modifier	Within the same class	Within the same package	Outside of the package
private	Yes	No	NO
default	Yes	Yes	NO
Protected	Yes	Yes	Yes [But using Inheritance]
public	Yes	Yes	Yes

How to print the non static field by using overridden toString() method :

If we want to print our object properties (Instance Variables) then we should generate(override) toString() method in our class from Object class.

Now with the help of toString() method **we need not to write** any display kind of method to print the object properties i.e instance variable.

In order to generate the toString() method we need to follow the steps
Right click on the program -> source -> generate toString()

In order to call this toString() method, we need to **print** the corresponding object reference by using System.out.println() statement.

Manager **scott** = new Manager();
System.out.println(**scott**); //Calling **toString()** method of **Manager class**

Employee e = new Employee();
System.out.println(e); //Calling toString() method of Employee class.

```
//Program
package com.ravi.to_string;
//BLC
public class Customer
{
   private int customerId;
   private String customerName;
   private double customerBill;

   public void setCustomerData(int id, String name, double bill) {
      customerId = id;
      customerName = name;
      customerBill = bill;
   }

   //Right click -> source -> generate toString() method
   @Override
   public String toString()
   {
      return "Customer [customerId=" + customerId + ", customerName=" + customerName
+ ", customerBill="
            + customerBill + "]";
   }
}

package com.ravi.to_string;

public class CustomerDemo
{
   public static void main(String[] args)
   {
      Customer scott = new Customer();
      scott.setCustomerData(111, "Scott", 12890.90);
       System.out.println(scott);//calling toString() method of Customer class

       Customer alen = new Customer();
       alen.setCustomerData(222, "Alen", 18890.90);
       System.out.println(alen);//calling toString() method of Customer class

   }
}

package com.ravi.to_string;

public class CustomerDemo
{
   public static void main(String[] args)
   {
      Customer scott = new Customer();
      scott.setCustomerData(111, "Scott", 12890.90);
       System.out.println(scott);//calling toString() method of Customer class

       Customer alen = new Customer();
       alen.setCustomerData(222, "Alen", 18890.90);
       System.out.println(alen);//calling toString() method of Customer class

   }
}
```

What is Variable Shadow in java ?

If class level variables and method level variables are having exactly same name then method level variables will hide class level variable inside the method OR constructor OR block body, This concept is known as Variable Shadow.

```
package com.ravi;

class Student
{
    private int roll = 101;
    private String name = "Scott";
    static String address = "Ameerpet Hostel";


    public void accept(int roll, String name)
    {
        String address = "NIT campus";
        System.out.println("Roll is :"+roll); //201
        System.out.println("Name is :"+name); //Alen
        System.out.println("Address is :"+address); //NIT campus
    }
}

public class VaribleShadow
{
    public static void main(String[] args)
    {
        Student s1 = new Student();
        s1.accept(201, "Alen");

    }
}
```