Access modifiers in java :

* An access modifier describes the **accessibility level of the class OR the accessibility level of the member of the class.**

* In order to define the **accessibility level** we have total 4 access modifiers :

1) private access modifier (**Within the same class**)
2) default OR private - package access modifier (**Within the same package**)
3) protected access modifier (**Within the same package and from another package using inheritance**)
4) public access modifier (**No restriction**)

private :

It is the most restrictive access modifier because the member declared as private can't be accessible from outside of the class.

In Java we can't declare an outer class as a private or protected Or static. Generally we should declare the non static fields with private access modifier (Data hiding).

In java outer classes can be declared as public, abstract, final, sealed and non-sealed modifier only.

default OR private-package :

```
                    class Test
                    {
default  ───►       int x = 10;
                    void show()
                    {
                    }
                    }
```

It is an access modifier which is less restrictive than private. It is such kind of access modifier whose physical existence is not available that means when we don't specify any kind of access modifier before the class name, field name or method name then by default it would be default.

As far as its accessibility is concerned, default members are accessible within the same folder(package) only.
It is also known as private-package modifier.

```
package com.ravi.d1c;                    package com.ravi.d1c;

public class Test                        import com.ravi.d1c.Test;
{
    int x = 100;                         public class Main {
}
                                            public static void main(String[] args)
                                            {
                                                Test t1 = new Test();
                                                SD.println(t1.x);
                                            }
                                         }
                                         x field is not accessible from another package
                                         because the access
                                         modifier is default
```

protected :

It is an access modifier which is less restrictive than default because the member declared as protected can be accessible from the outside of the package (folder) also but only by using inheritance concept.

```
package com.ravi.d1c;                    package com.ravi.d1c;

public class Test                        import com.ravi.d1c.Test;
{
    protected int x = 100;               public class Main extends Test  //Inheritance
}                                        {
                                            public static void main(String[] args)
                                            {
                                                Main obj = new Main();
                                                SD.println(obj.x);
                                            }
                                         }
                                         x is accessible here in the ELC class because
                                         x is protected so
                                         we all are are using Inheritance
```

public :

It is an access modifier which does not contain any kind of restriction that is the reason the member declared as public can be accessible from everywhere without any restriction.

According to Object Oriented rules we should declare the classes and methods as public where as fields must be declared as private or protected according to the requirement.

Note : If a method is used for internal purpose one (like validation) then we can declare that method as private method (it is called Helper method)

```
public void makePayment(String name, String creditCard, double amount)
{
    if(isValidCard(creditCard))
    {
    }
}

public void makePayment(String debitCard, double amount)
{
    if(isValidCard(debitCard))
    {
    }
}

private boolean isValid(String card)  //Helper method (Javadoc)
{
    if(card.length() ==16)
    {
    }
    return true;
}
```

private	yes	no	no	no
default	yes	yes	no	no
protected	yes	yes	yes	no
public	yes	yes	yes	yes

How to print Object properties (Non static fields) by using toString() :

If we want to print our object properties (Instance Variables OR Non static fields) 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 SD.println() statement.

Manager m = new Manager();
SD.println(m); //Calling toString() method of Manager class

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

```
package com.ravi.d1c;

public class Manager
{
    private int managerId;
    private String managerName;
    private double managerSalary;

    public void setManagerDetails(int id, String name, double salary)
    {
        managerId = id;
        managerName = name;
        managerSalary = salary;
    }

    public String toString() {
        return "Manager [managerId=" + managerId + ", managerName=" + managerName + ",
        managerSalary=" + managerSalary
            + "]";
    }
}
```

```
package com.ravi.d1c;

import com.ravi.d1c.Manager;

public class ManagerDemo {

    public static void main(String[] args)
    {
        Manager m1t1 = new Manager();
        m1t1.setManagerDetails(30, "Mr Scott", 80000);
        SD.println(m1t1);  //calling toString() internally

        SD.println("..........");

        Manager m1t1 = new Manager();
        m1t1.setManagerDetails(30, "Miss Sumit", 90000);
        SD.println(m1t1);  //calling toString() internally
    }
}
```