CS4097D Object Oriented Systems Laboratory

CSED NIT Calicut

09/08/2021

Contents

1 Objects and Classes

Object

> It is a basic unit of Object-Oriented Programming and represents the real life entities.

Anything we can see, touch, feel is an object. **Examples:- Laptop, Mobile Phone, Dog, Square etc.**

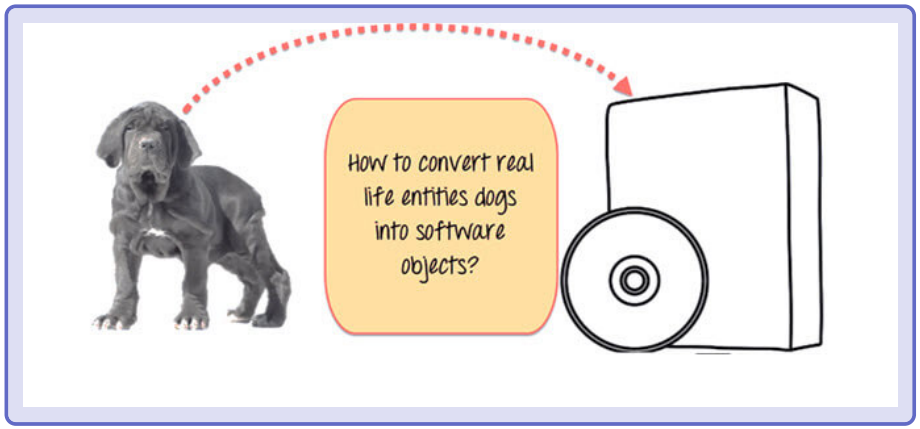
Class

> A class is a user defined blueprint or prototype from which objects properties and behaviours are decided

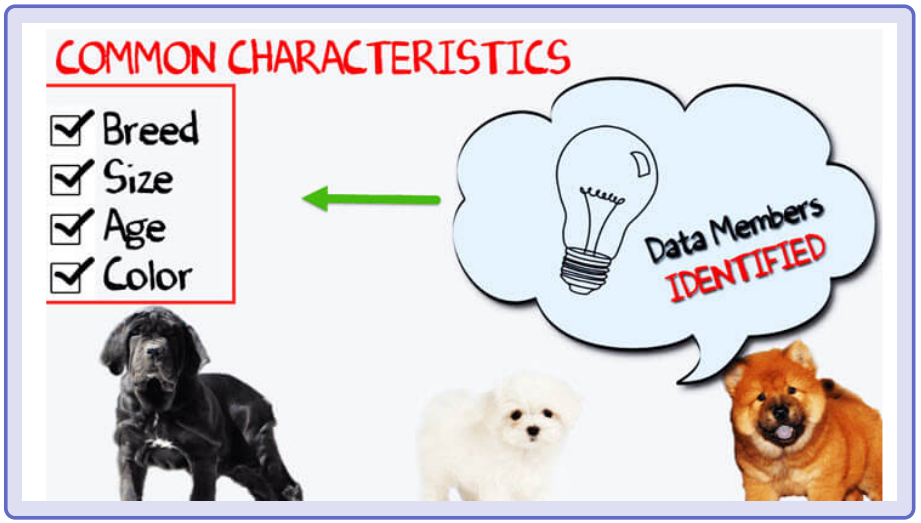
A drawing of an object which explains how it will looks like and how it's represented. Objects will have properties and behaviours, they are written in the class to describe the object.

Let us understand the concept of Java Classes and Objects with an example.

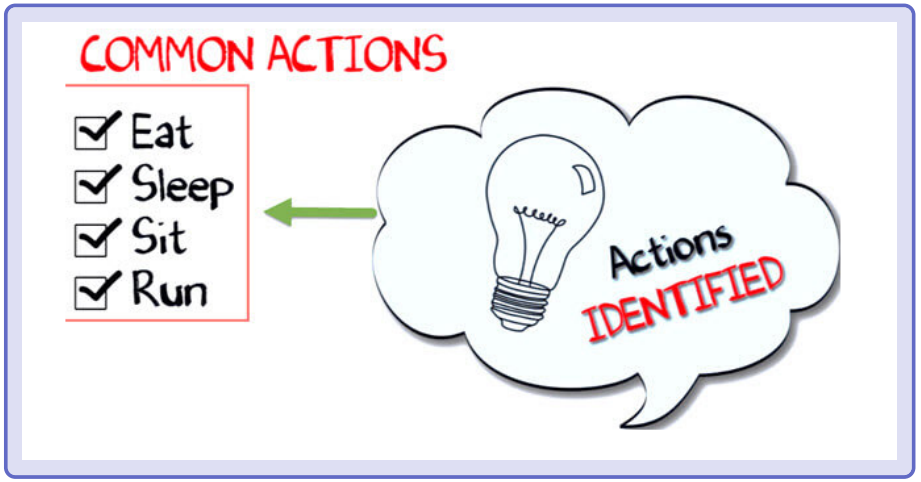
Modelling real-life beings into software entities.

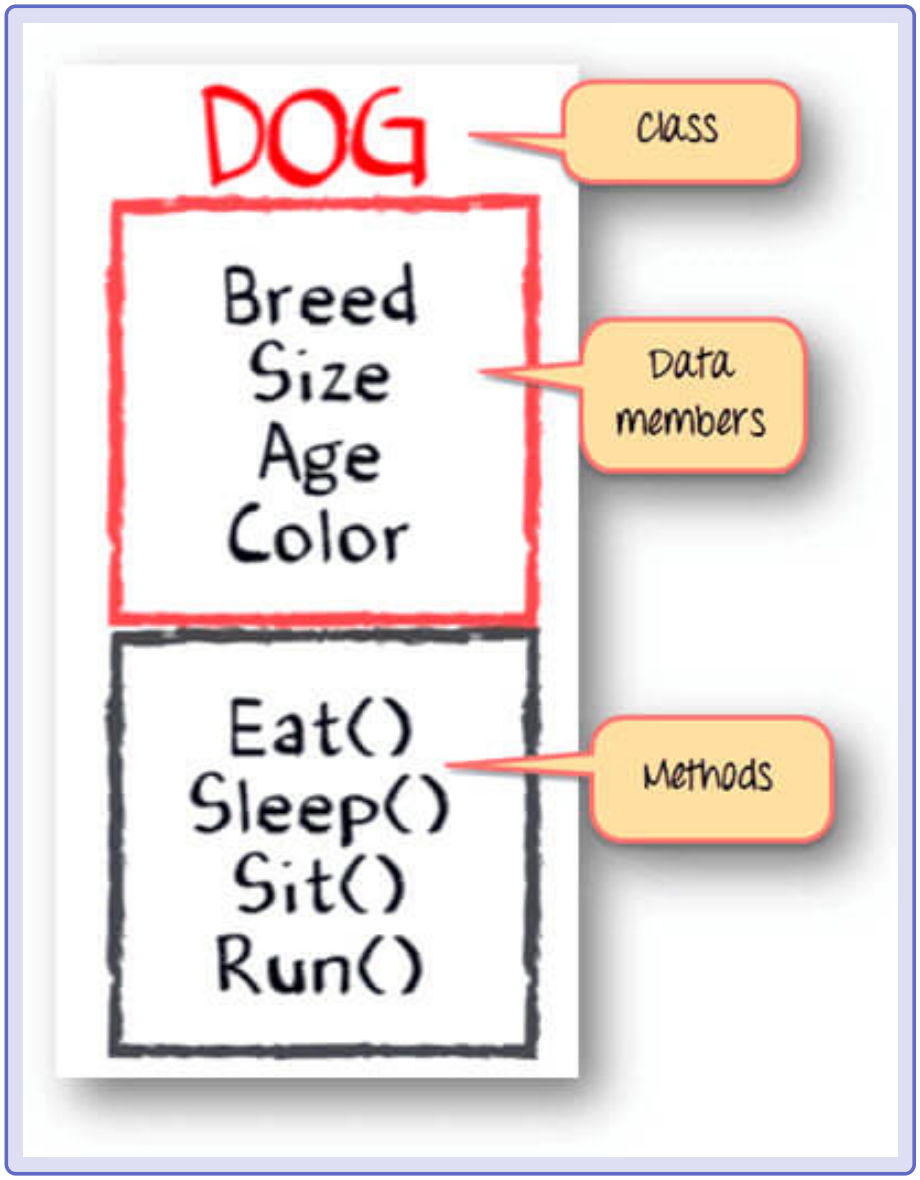


Characteristics or differences (breed, age, size, color) form a data members for the object.

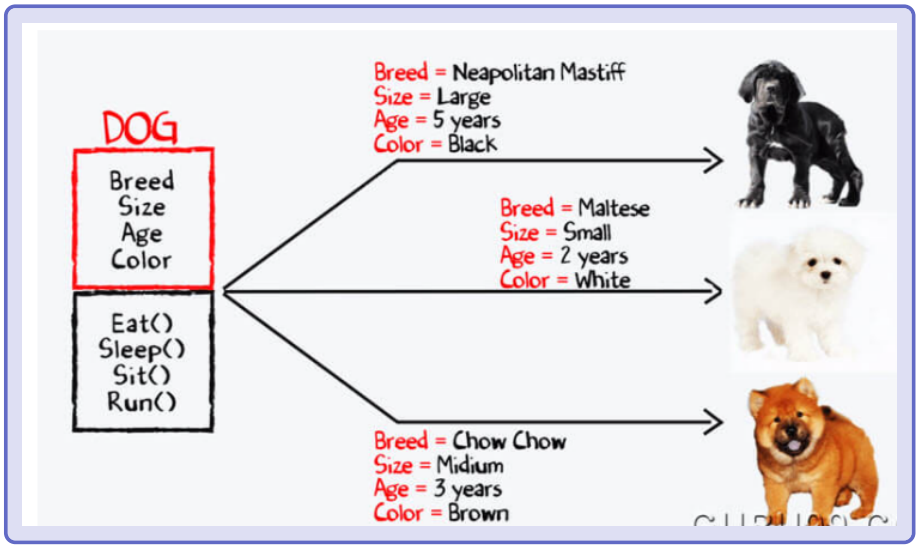


Common behaviours of these dogs like sleep, sit, eat, etc. will be the actions for software objects.





For different values of data members (breed size, age, and color) in Java class, you will get different dog objects.



2 Collections

collection

> A collection is a group of objects (represented as a single unit). In Java, these objects are called elements of the collection.

> Technically, a collection is an object or container which stores a group of other objects as a single unit or single entity. Therefore, it is also known as container object or collection object in java.

Examples:- A classroom is a collection and students are objects.

The world is a collection and humans, animals and different things are different objects.

- A collection object has a class that is known as collection class or container class.
- All collection classes are present in java.util package.

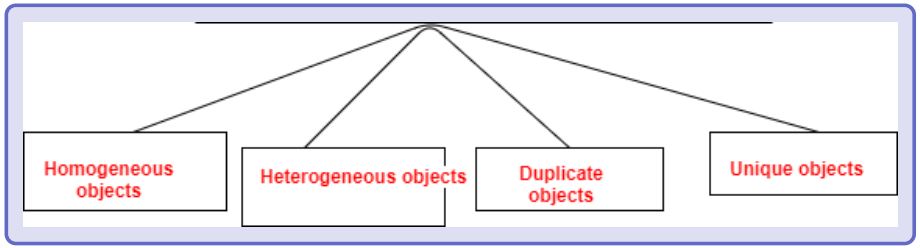
- A group of collection classes is called collections framework in java.

A simple example of a collection

```
// Create a container list of cities (objects or elements).
   List<String> cities = new ArrayList<String>();

// Adding names of cities.
    cities.add("New York");
    cities.add("Dhanbad");
    cities.add("Mumbai");
```

2.1 Types of Objects Stored in Collection



- Homogeneous objects are a group of multiple objects that belong to the same class.

  Example:- Creating three objects Student s1, Student s2, and Student s3 of the same class Student.

- Heterogeneous objects are a group of different objects that belong to different classes.

  Example:- creating two different objects of different classes such as one object Student s1, and another one object Employee e1. Here, student and employee objects together are called a collection of heterogeneous objects.

- The multiple objects of a class that contains the same data are called duplicate objects.

```
Person p1 = new Person( "abc");
Person p2 = new Person("abc");
```

- The multiple objects of a class that contains different data are called unique objects.

```
Person p1 = new Person("abcd");
Person p2 = new Person("abcde");
```

2.2 Need for Collections

we cannot store different class objects into the same array.

```
Employee[ ] emp = new Employee[5000]; // It will hold only employee type objects.

For example:
    emp[0] = new Employee(); // valid.
    emp[1] = new Customer(); // invalid because here, we are providing the customer
type object.
```

- An array is static in nature. It is fixed in length and size. We cannot change (increase/decrease) the size of the array based on our requirements once they created.
- We can add elements at the end of an array easily. But, adding and deleting elements or objects in the middle of array is difficult.
- We cannot insert elements in some sorting order using array concept because array does not support any method. We will have to write the sorting code for this but in the case of collection, method support is available for sorting using Tree set.
- We cannot search a particular element using an array, whether the particular element is present or not in the array index. For this, we will have to write the searching code using array but in the case of collection, method called contains() is available.

Due to all these above limitations of array, programmers need a better mechanism to store a group of objects. So, the alternative option is a collection object or container object in java.

> A collection in java is a container object that is used for storing multiple homogeneous and heterogeneous, duplicate, and unique elements without any size limitation.

3 Strings

Strings, which are widely used in Java programming, are a sequence of characters. In the Java programming language, strings are objects. The Java platform provides the **String** class to create and manipulate strings.

3.1 Create String

The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class. Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

Create A String

The most direct way to create a string is to write:
String str = "abc";

In this case, "abc" is a string literal—a series of characters in your code that is enclosed in double quotes. Whenever it encounters a string literal in your code, the compiler creates a String object with its value—in this case, abc is equivalent to: .

Create A String

The above code is equivalent to
char data[] = {'a', 'b', 'c'};
String str = new String(data);

3.2 String Length

A String in Java is actually an object, which contain methods that can perform certain operations on strings. For example, the length of a string can be found with the *length()* method:

```
public class Main {
  public static void main(String[] args) {
    String txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    System.out.println("The length of the txt string is: " + txt.length());
  }
}
```

```
The length of the txt string is: 26
```

3.3 Finding a Character in a String

The *indexOf()* method returns the index (the position) of the first occurrence of a specified text in a string (including whitespace):

```
public class Main {
  public static void main(String[] args) {
    String txt = "Please locate where 'locate' occurs!";
    System.out.println(txt.indexOf("locate"));
  }
}
```

```
7
```

Note

Java counts positions from zero.
0 is the first position in a string, 1 is the second, 2 is the third ...

3.4 String Concatenation

The **+** operator can be used between strings to combine them. This is called concatenation:

```
public class Main {
  public static void main(String args[]) {
    String firstName = "John";
    String lastName = "Doe";
    System.out.println(firstName + " " + lastName);
  }
}
```

```
John Doe
```

Note

Note that we have added an empty text (" ") to create a space between firstName and lastName on print.
You can also use the concat() method to concatenate two strings:

```
public class Main {
  public static void main(String args[]) {
    String firstName = "John";
    String lastName = "Doe";
    System.out.println(firstName + " " + lastName);
  }
}
```

```
John Doe
```

4 Constructors

Constructors are used to initialize the object's state. Like methods, a constructor also contains a collection of statements (i.e. instructions) that are executed at the time of Object creation.

4.1 Need of Constructor

1. Think of a Box. If we talk about a box class then it will have some class variables (say length, breadth, and height). But when it comes to creating its object(i.e Box will now exist in the computer's memory), then can a box be there with no value defined for its dimensions. The answer is no.

2. So constructors are used to assign values to the class variables at the time of object creation, either explicitly done by the program-

mer or by Java itself (default constructor).

4.2 When is a Constructor called?

1. Each time an object is created using a new() keyword, at least one constructor (it could be the default constructor) is invoked to assign initial values to the data members of the same class.

2. A constructor is invoked at the time of object or instance creation. For Example:

Constructor invoke

```
class JavaLab
{
  .......

  // A Constructor
  new JavaLab() {}

  .......
}

// We can create an object of the above class
// using the below statement. This statement
// calls above constructor.
JavaLab obj = new JavaLab();
```

4.3 Rules for writing Constructor

1. Constructor(s) of a class must have the same name as the class name in which it resides.

2. A constructor in Java can not be abstract, final, static and Synchronized.

3. Access modifiers can be used in constructor declaration to control its access i.e which other class can call the constructor.

4.4 Types of constructor

There are two types of constructor in Java:

1. **No-argument constructor** A constructor that has no parameter is known as the default constructor. If we don't define a constructor in a class, then the compiler creates default constructor(with no arguments) for the class. And if we write a constructor with arguments or no-arguments then the compiler does not create a default constructor.
   Default constructor provides the default values to the object like 0, null, etc. depending on the type.

No-argument constructor

```
// Java Program to illustrate calling a no-argument constructor
import java.io.*;
class JavaLab
{       int num;
        String name;

        // this would be invoked while an object of that class is created
        JavaLab()
        {
                System.out.println("Constructor called");
        }
}
class s5Lab
{
     public static void main (String[] args)
        {
                // this would invoke default constructor.
                JavaLab intro1 = new JavaLab();

// Default constructor provides the default values to the object like 0,
// null
                System.out.println(intro1.name);
                System.out.println(intro1.num);
        }
}
```

Output

Constructor Called
null
0

Parameterized Constructor 1 of 2

```
// Java Program to illustrate calling of parameterized constructor.
import java.io.*;
class JavaLab
{
        // data members of the class.
        String name;
        int id;

        // constructor would initialize data members with the values of passed
       //arguments while object of that class created.
        JavaLab(String name, int id)
        {
                this.name = name;
                this.id = id;
        }
}
```

2. **Parameterized Constructor** A constructor that has parameters is **parameterized constructor**. If we want to initialize fields of the class with your own values,then use a parameterized constructor.

Parameterized Constructor 2 of 2

```
//Parameterized constructor (continued...)

//Main Class

class s5Lab

{

        public static void main (String[] args)

        {

                // this would invoke the parameterized constructor.

                JavaLab intro1 = new JavaLab("Aswin", 1);

                System.out.println("CR Name :" + intro1.name +

                                                " and CR Id :" + intro1.id);

        }

}
```

Try predicting the output, by running this program

Constructor Overloading 1 of 2

```
// Java Program to illustrate constructor overloading
// using same task (addition operation ) for different
// types of arguments.
import java.io.*;

class JavaLab
{
    // constructor with one argument
    JavaLab(String name)
    {
        System.out.println("Constructor with one " +
                        "argument - String : " + name);
    }

    // constructor with two arguments
    JavaLab(String name, int age)
    {

        System.out.println("Constructor with two arguments : " +
                " String and Integer : " + name + " "+ age);

    }

    // Constructor with one argument but with different
    // type than previous..
    JavaLab(long id)
    {
        System.out.println("Constructor with one argument : " +
                                                "Long : " + id);
    }
}//close class JavaLab
```

4.5 Does constructor return any value?

There are no “return value” statements in the constructor, but the constructor returns the current class instance. We can write ‘return’ inside a constructor.

4.6 Constructor Overloading

Like methods, we can overload constructors for creating objects in different ways. Compiler differentiates constructors on the basis of numbers of parameters, types of the parameters and order of the parameters.

Constructor Overloading 2 of 2

```
//constructor overloading (continued…)
//Main class

class s5Lab
{
    public static void main(String[] args)
    {
        // Creating the objects of the class named 'JavaLab'
        // by passing different arguments

        // Invoke the constructor with one argument of
        // type 'String'.
        JavaLab intro2 = new JavaLab("OOS Lab");

        // Invoke the constructor with two arguments
        JavaLab intro3 = new JavaLab("Aswin", 1);

        // Invoke the constructor with one argument of
        // type 'Long'.
        JavaLab intro4 = new JavaLab(325614546);
    }
}
```

- **Try predicting the output, by running this program**

4.7 How constructors are different from methods in Java?

1. Constructor(s) must have the same name as the class within which it defined while it is not necessary for the method in java.
2. Constructor(s) do not return any type while method(s) have the return type or void if does not return any value.
3. Constructor is called only once at the time of Object creation while method(s) can be called any numbers of time.

5 Access Modifiers

The purpose to set accessibility of classes, methods, and other members are called the **Access Modifiers**. In Java language they are in use to restricting or set the limits to the accessibility of classes, interfaces, variables, methods, constructors, data members, and the setter methods.

5.1 WHY ACCESS MODIFIERS?

1. Access modifiers are really important features use in the Java language to make the code clean and efficient.
2. They have a major role in dealing with the visibility of the class members.
3. It has a greater dominance in deciding the control whether other classes can see or change methods or variables of the class. however, it has a greater role relating to implementation and defining the properties of the important part of Object-Orient Programming refer
4. Encapsulation. It is a feature in which the linkage among data with the code takes place and it also can manipulate the data. With controlled access, we can prevent misuse.

5.2 Types of Modifiers

Modifiers

There are two types of *Modifiers* in Java

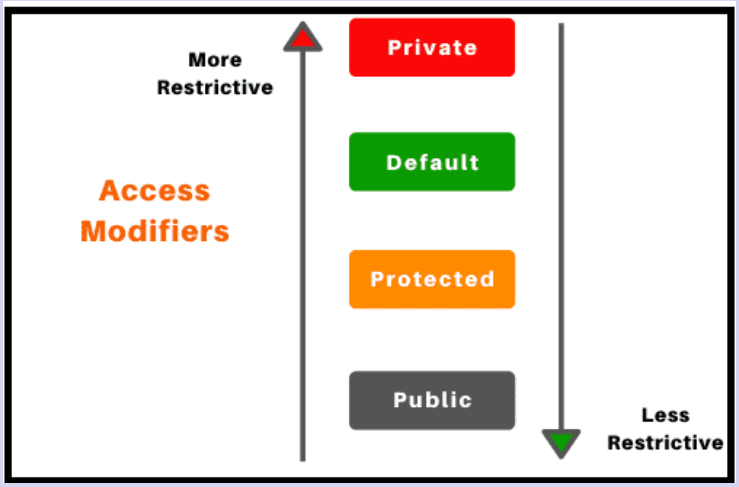
- **Access Modifiers**:*Default,Public,Protected,Private*
- **Access Modifiers**: *static, abstract, synchronized, native, volatile, transient, etc*

Main Focus on Access Modifiers

Access Modifiers

There are four types of *Access Modifiers* in Java

Types



1. **Default:**The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
2. **Public:**The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.
3. **Protected:**The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Private:**The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

Scope of Access Modifiers in Java

		public	private	protected	default
Same Package	Class	YES	YES	YES	YES
	Sub class	YES	NO	YES	YES
	Non sub class	YES	NO	YES	YES
Different Package	Sub class	YES	NO	YES	NO
	Non sub class	YES	NO	NO	NO

5.2.1 Default Access Modifier

- In the default access modifier, if no access is specifies for any class, method, or data member hence, then it is taken as default.
- The data members, class, or methods are accessible only within the same package that is not declare using any access modifiers.
- Classes from other packages cannot be used along with default modifiers. If we use then it will have a compile time error.

When no access specifier is used, it is taken as the default specifier

Default Access Modifier

```
package p;
class  PianalytiX
{
    void display()
       {

System.out.println("PianalytiX");
       }
}
```

Classes from other packages cannot be used along with default modifiers. If we use then it will have a compile-time error.

Default Access Modifier

```
package q;
import p.*;
class PianalytiXs
{
    public static void main(String
args[])
       {
           PianalytiX obj = new
PianalytiX();
           obj.display();
       }
}
Output:
Compile time error
```

5.2.2 Private Access Modifier

The keyword private is use to specify the private access modifier. Data members and methods are only available in that class where it is declared. So, the members cannot be access by the equal package of which they were create, Among top-level classes or interfaces.
anyone out of them cannot be called private. Also, classes cannot be declare as private access modifiers. So, a Class with a private constructor also cannot create the object of that class from outside the class.

Program to show error while using a class from a different package with a private modifier.

Private Access Modifier

```
package p1;
class A
{
   private void show( )
    {
        System.out.println("PianalytiX");

    }
}
class B
{
   public static void main(String args[])

      {
          A obj = new A( );
          obj.show( );
      }
}

Output:
error: show( ) has private access in A
          obj.show( );
```

5.2.3 Protected Access Modifier

The keyword protected is used to specify the protected access modifier. The protected access modifier has an inclusive use where data members or methods declare as protect are accessible within the same subclasses or packages in non-identical packages.

The use of a protected access modifier

Protected Access Modifier

```
package p1;
public class A
{
   protected void display( )
    {
 System.out.println("PianalytiX");
    }
}
```

A second package where we import the first where the protected access modifier is in use.

Protected Access Modifier

```
package p2;
import p1.*;
class B extends A
{
   public static void main(String args[])

   {
       B obj = new B();
       obj.display();
   }

}

Output:
PianalytiX
```

5.2.4 Public Access Modifier

- The keyword public is use to specify the public access modifier.
- It has the most spacious range in comparison to all other access modifiers.
- It helps any classes, methods, or data members to get access from anywhere in the program.
- It ensures there is no restriction on the scope of a public data member.

Public Access Modifier

```
package p1;
public class A
{
   public void show()
      {

 System.out.println("PianalytiX");
      }
}
package p2;
import p1.*;
class B
{
    public static void main(String
args[])
      {
          A obj = new A;
          obj.show();
      }
}
```

Usage of the public access modifier