Sri SAI RAM ENGINEERING COLLEGE
INSTITUTE OF TECHNOLOGY
West Tambaram, Chennai - 44

SAIRAM DIGITAL RESOURCES

YEAR II SEM III

CS8392

OBJECT ORIENTED PROGRAMMING
(Common to CSE, EEE, EIE, ICE, IT)

COMPUTER SCIENCE & ENGINEERING

UNIT NO 1

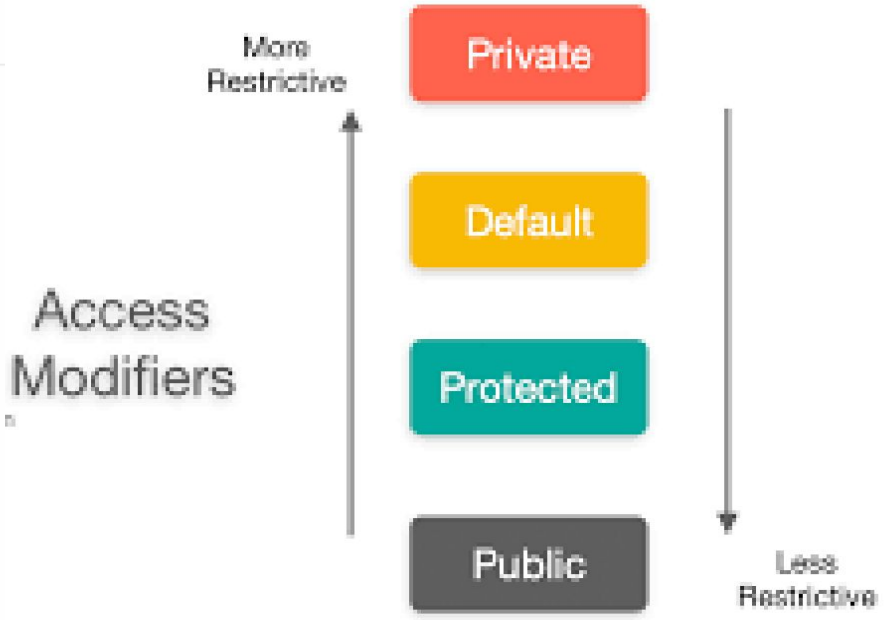
INTRODUCTION TO OOP AND JAVA FUNDAMENTALS

1.6. Access Specifiers, Static Members

Comments

Access Specifiers

- The access modifiers/specifiers in Java specifies the accessibility or scope of a field, method, constructor, or class.
- The access level of fields, constructors, methods and class can be changed by applying the access modifier on it.
- Access specifiers are applied before data members or methods of a class. They provide features accessing and controlling mechanism among the classes and interfaces.
- Types of Access specifiers
    1. Private
    2. Public
    3. Protected
    4. Default



1. Private Access Specifier

- The access level of a private modifier is only within the class. It cannot be accessed from outside the class. They are also called as class level access modifiers.
- The private access modifier is specified using the keyword private.
- If the class constructor is specified as private, the instance of that class (Object) cannot be created from outside the class.

Example

```
class Hello {
        private int a=20;
        private void show()      {
                System.out.println("Hello World");
        }
}
public class Demo {
         public static void main(String args[])      {
                    Hello obj=new Hello();
                            System.out.println(obj.a); // Generates Compile Time Error,can't access private data
                    obj.show();   //Generates Compile Time Error, can't access private methods
        }
}
```

2. Public Access Specifier

- 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. They are also called as universal access modifiers.
- The public access modifier is specified using the keyword public.

Example

```
class Hello  {

        public int a=20;
        public void show()       {
                System.out.println(“Welcome to Java");
        }
}
public class Demo {
         public static void main(String args[])      {
                Hello obj=new Hello();
                System.out.println(obj.a);      // 20
                obj.show();                  //Welcome to Java
        }
}
```

3. Protected Access Specifier

- The access level of a protected modifier is within the package and outside the package through child class. It is also accessible in inherited class of another package. They are also called as derived level access modifiers.
- The protected access modifier is specified using the keyword protected.

Example

```
package pack1;
public class A {
        protected void show() {
                System.out.println("Hello World");
        }
}

package pack2;
import pack1.*;
class B extends A  {
        public static void main(String args[]) {
                B obj = new B();
                obj.show();  //  Hello World
         }
}
```

4. Default Access Specifier

- The access level of a default modifier is only within the package. It cannot be accessed from outside the package. They are also called as package level access modifiers.
- When no access modifier is specified for a class , method or data member – It is said to be having the default access modifier by default.
- The data members, class or methods having default access modifier are accessible only within the same package.

Example

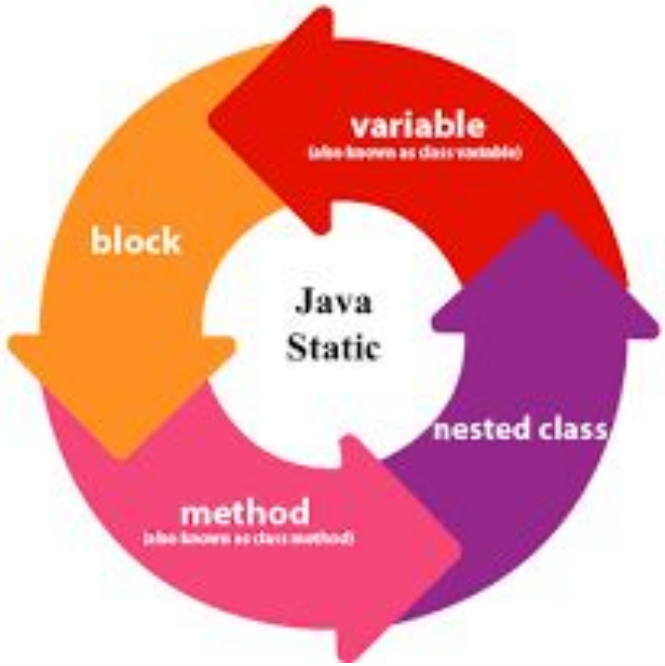
```
package pack1;
class A  {
        void show()  {                System.out.println("Hello World");        }
}

package pack2;
import pack1.*;
class B  {
        public static void main(String args[])  {
                A obj = new A(); //Compile Time Error, can't access outside the package
                obj.show();   //Compile Time Error, can't access outside the package
         }
}
```

Scope of Access Specifiers

	Within same class	Within same package	Outside the package(sub-class)	Outside the package(Global)
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes(only to derived class)	No
Default	Yes	Yes	No	No
Private	Yes	No	No	No

Static Keyword

- Static keyword in Java is mainly used for memory management. It can be used with variables, methods, blocks and nested classes.
- Static keyword is a non-access modifier and can be used for the following
  - Static Variable
  - Static Block
  - Static Method
  - Static Class



Static Variable

- When a variable is declared as static, then a single copy of the variable is created and divided among all objects at the class level. They are also known as class variables.
- Static variables can be created at class-level only. They are basically global variables.
- All the instances of the class share the same static variable.
- Memory allocation for such variables only happens once when the class is loaded in the memory.
- Can be accessed directly in Static method
- Are shared among all the instances of class
- Advantage:
  - Makes program memory efficient

Rules of Static Variable

- The static variable maintains single copy for a whole class.
- The static variables cannot be declared within a method or block.
- If the value of a static variable changed by an object then it gets reflected into all the objects of the class.
- The static variables get created at the time of class loading and destroy at the time of class unloading.
- A static variable can be accessed using a reference variable or class name.
- A static variable can be used to hold a value which is common for all the objects of a class.
- The JVM assigns default values to the static variables.

Syntax

```
static datatype variableName;
```

Static Variable

Example

```
class Sample {
        static int x = 10;// static variable
        public static void main(String[] args)  {
                // can access static variables from static method directly
                System.out.println(x);// 10
                Sample s1 = new Sample();
                s1.m1();
        }
        public void m1()  {
                //access static variable x from instance method m1() directly
                System.out.println(x); // 10
        }
}
```

Static Block

- Static block is used for initializing the static variables. This block gets executed when the class is loaded in the memory.
- The static block gets executed very first before static and non static methods even before public static void main(String[] args).
- A class can have multiple Static blocks, which will execute in the same sequence in which they have been written into the program.

Rules of Static Block

- Static block always get executed before static method.
- Static block cannot return a value.
- Static block cannot be called explicitly.
- Static block cannot throws an exception.
- The this and super keywords cannot be used inside the static block.

Syntax

```
static {      //body      }
```

Example – Single Static Block

```
class JavaExample {
        static int num;
       static String mystr;
        static{
                num = 97;
                 mystr = “Single Static Block in Java";
        }
       public static void main(String args[])  {
                System.out.println("Value of num: "+num);  //97
                System.out.println("Value of mystr: "+mystr); // Single Static Block in Java
        }
}
```

Example – Multiple Static Block

```
class JavaExample2{
    static String mystr;
    //First Static block
    static{
        System.out.println("Static Block 1");
        mystr = "Block1";
    }
   //Second static block
   static{
        System.out.println("Static Block 2");
        mystr = "Block2";
    }
    public static void main(String args[])  {
        System.out.println("Value of String: "+mystr);
    }
}
```

Output:

```
Static Block 1
Static Block 2
Value of String: Block 2
```

Static Methods

- When a method is declared with the static keyword, it is known as a static method.
- Static Methods can access class variables(static variables) without using object(instance) of the class, however non-static methods and non-static variables can only be accessed using objects.
- Static methods can be accessed directly in static and non-static methods.

Rules of Static Methods

- A static method belongs to the class rather than object of a class.
- A static method can be invoked without(object) the need for creating an instance of a class.
- The static method can access static data members and can call static methods directly.
- The static method cannot access non static data member or cannot call non-static method directly. An object is required to access non static members from a static method.
- The this and super keywords cannot be used in static context.
- The static methods can be called on an instance variable or on the class name. Recommended is to call on class name.

Restrictions for static method

- They can directly call other static methods only.
- They can access static data directly.

Syntax

```
static returntype methodName() {
        //body
}
```

Example – static method main accessing static variables without object

```
class StaticMethodExample1 {
    static int i = 10;
    static String s = " Static Method";
   //This is a static method
  public static void main(String args[])   {
        System.out.println("i:"+i);  //10
        System.out.println("s:"+s); // Static Method
    }
}
```

Example – Static method accessed directly in static and non-static method

```
class StaticMethodExample2{
    static String s = " Static Method";
    static void display()  {
        System.out.println("String:"+s);
    }
    void funcn()  {
        display();
 }
public static void main(String args[])  {
   StaticMethodExample2 obj = new StaticMethodExample2();
    obj.funcn();
    display();
  }
}
```

Output:

String: Static Method

String: Static Method

Static Class

- A class can be made static only if it is a nested class.
- Nested static class doesn't need reference of Outer class
- A static class cannot access non-static members of the Outer class

Example

```
public class Nested  {
    private static String str= "Static Class";
     //Static class
     static class MyClass  {
         //non-static method
         public void disp(){
              System.out.println(str);
     }
}
```

```
public static void main(String args[]) {
        Nested.MyClass obj = new Nested.MyClass();
        obj.disp();
 }
```

Output:

Static Class

Java Comments

- Statements that are not executed by the compiler and interpreter.
- Provides information or explanation about the variable, method, class or any statement.
- Used to hide program code.
- Types of Comments
  - Single Line Comment
  - Multi Line Comment
  - Documentation Comment

Single Line Comment

- The single line comment is used to comment only one line.
- These are mostly used for describing the code functionality. It is the most easiest typed comments

Syntax:

```
// This is a single Line Comment
```

Example:

```
class SingleLineComment {
        public static void main(String args[])     {
            // Single line comment here
            System.out.println("Single line Comment");   //Single line Comment
       }
}
```

Multi Line Comment

- The multi line comment is used to comment multiples lines of code.

Syntax:

```
/*  This
    is a
    Multi Line Comment */
```

Example:

```
class MultiLineComment {
    public static void main(String args[])    {
        /* Multi Line
          comment Example */
        System.out.println(“Multi line Comment");   //Multi line Comment
    }
}
```

Documentation Comment

- The documentation comment is used to create documentation API.
- To create documentation API, javadoc tool has to be used.
- Generally used when writing code for a project/software package, since it helps to generate a documentation page for reference, which can be used for getting information about methods present, its parameters, etc.

Syntax:

```
/**
This
is
documentation
comment
*/
```

Example:

```
/** The Calculator class provides methods to get addition and subtraction of given 2 numbers.*/
public class Calculator {
        /** The add() method returns addition of given numbers.*/
       public static int add(int a, int b)    {   return a+b   ;}
        /** The sub() method returns subtraction of given numbers.*/
         public static int sub(int a, int b)   {   return a-b;   }
}
```

Compile it by javac tool:

```
javac Calculator.java
```

Create Documentation API by javadoc tool:

```
javadoc Calculator.java
```

- A HTML files will be created for the Calculator class in the current directory. It will contain the explanation of Calculator class provided through documentation comment.

Video Link

https://www.youtube.com/watch?v=WZRTwkCOYBQ