OBJECT ORIENTED PROGRAMMING USING

Java

Let's explore technology
together to live in the future

Checkout more on
https://github.com/Sy-hash-collab

Sy-hash-collab

Static Members

Static Members (static methods, variables) are associated with the class rather than with any specific instance of class.

Accessing Static Members within a class:

1) **Direct Access:** Static members can be directly accessed within the class "without the need for any object reference."

```
class MyClass
{ static int staticVar = 10;
  static void staticMethod()
  { System.out.println("Static method
                        called");
  }
  void nonStaticMethod()
  { System.out.println("Non-static method
                        called");
    System.out.println(staticVar); // Direct access to static variable
    staticMethod(); // direct call to static method.
}}
```

2) **Using Class name:** Static members can be accessed using class name, followed by member name.

```
class MyClass
{ static int staticVar = 10;
  static void staticMethod()
  { System.out.println("Static method");
```

```
    void nonStaticMethod()
    { System.out.println("Non-static method");
      System.out.println(MyClass.staticVar);

      MyClass.staticMethod();
    }}
```

Accessing Static Members Outside Class

1) **Using Class Name:** Static members can be accessed using the class name, outside class

```
public class Main{
  public static void main(String[]args)
    { System.out.println(MyClass.staticVar);

      MyClass.staticMethod();
    }}
```

2) **Importing the Class (in a different package):**

If the class containing static members is in a different package, you can import the class and then access its static members directly.

```
import package_name.MyClass;
public class Main{
  public static void main(String[]args)
    { System.out.println(MyClass.staticVar);
      MyClass.staticMethod();
    }}
```

Non-Static Members

Non-static members (instance variables and methods) can be accessed in different ways, within a class and outside of class.

Within a Class:

1) **Direct Access:** Instance variables and methods can be directly accessed within the class without specific qualifiers.

```
class MyClass
{   int instanceVar = 10;

    void instanceMethod()
    { System.out.println("Instance method called");
      System.out.println("Value of
                          instance variable: " +
                            instanceVar);

}}
```

2) **Using 'this' keyword:** The "this" keyword is a reference to current instance of class. It is often used to differentiate instance variables from local variables with same name.

```
class MyClass
{   int instanceVar = 10;

    void instanceMethod()
    {
      int instanceVar = 20;
```

```
        System.out.println ("Value of local
                            instanceVar: "+ instanceVa

        System.out.println ("Value of instanceVar using
                            this: "+ this.Var);
    }}
```

Outside the Class:

1) **Using Object Reference:** To access non-static members outside the class, you need to create an object of the class. Through the object reference, you can access instance variables and call instance metho

```
public class Main {
  public static void main (String [] args)
    {   // creating object of MyClass
        MyClass obj = new MyClass();
      // accessing instance variable
    System.out.println (" Value of instanceVar: "
                        + obj.instanceVar );
      // Calling instance method
    obj.instanceMethod();
  }}
```

2) **Using Constructor:**

```
class MyClass
{   int   instanceVar ;
    public MyClass (int value) // constructor to
      { instanceVar = value; }  initialize
                                instanceVar.
}
 pulic static void main (String [] args)
   { MyClass obj = new Myclass (30);
```

17/10/23

Java Inner Classes :

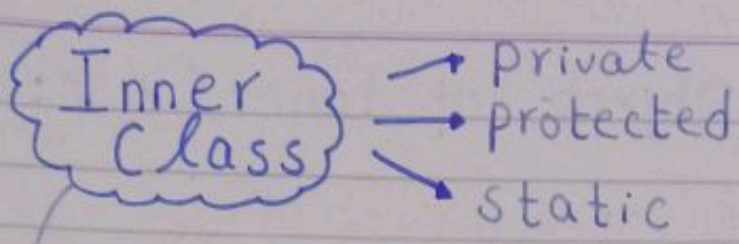
In Java, it is also possible to nest classes (a class within a class). The purpose of nested classes is to group classes that belong together which makes your code more readable and mantainable.

To access the inner class, create an object of the outerclass and then create an object of innerclass.

```
class OuterClass
{ int x = 10;

    class InnerClass
      { int y = 5;
      }
}
public class Main{
public static void main (String [] args)
  { OuterClass o = new OuterClass();

OuterClass.InnerClass i = o.new InnerClass();

System.out.println ("X:" + o.x);

System.out.println ("Y:" + i.y);

}}
```

* OuterClass has two members :
  ① variable - x
  ② class - InnerClass

* we can access x and y bcz both classes are in the same package.



→ bcz it is not an independent class it's a nested class (class within a class). If you don't want outside objects to access the inner class, declare the class as private.

Static Methods :

```
class My
{
  int z;
  public static void main (String [] args)
  {
    My m = new My();
    System.out.println (z);
  }
}
```

// non-static members of a class can't be accessed in static method without objec creatio

Non-Static Methods:

```
void show ()
{
  z ;
  this.z;
}
```

// non-static members can be accessed in non-static methods with 'this'

How to make Object of Inner Class:

```
class Student  → container class
{
   int rollno;

      class CR → inner class (member of
                                Student class)
      {
         void show()
         {
            System.out.println (rollno);
         }
      }
}
class Department
{ public static void main (String [Jargs)
   {  Student s = new Student();

Student.CR n1 = s.new CR();

} }
```

Static Member:

Static member is made only one time for each object in a class.
Made only one time in RAM.
No need for creation of object.

Static Inner Class:

An inner class c
be static, which means you can access
it without creating an object of outerc

```
class OuterClass
{  int x = 10;

    static class InnerClass
    {  int y = 5;
    }
}
public class Main {

public static void main (String [] args)
    {
// OuterClass.InnerClass i = new
OuterClass.InnerClass
System.out.println ("Y:" + i.y);
} }
```

calling static InnerClass creating object.

As innerClass is a member of Outer class so it can access all members of outer class even if they're private.

```
class Student
{ private int rollno;
    class CR
  { void show()
      { System.out.println (rollno);
```