Object-Oriented-Programming #LEC-9

▲ **Packages:** Packages are container for classes. So basically packages are folders

Ex: Suppose you have something like:

com.Kunal

→ That means there is a 'Com' folder and inside that there is 'Kunal' folder only

■ **Use case:**

Suppose you want create two files with the same name, let's just say Greeting.java but you can not do it as Java wouldn't allow you to do it inside the same folder or package.

So, you can create two file with same name inside different package, but those different package can be inside same parent directory or folder.

- It is use for visibility control and naming

■ **Import statement:**

- How does the run time of Java determine where to look for this packages we are creating?

→ By default, the Java runtime will look in the System, where it uses

the current working directory as it's starting point, and in the sub directory of the current directory it will be found.

- How does the runtime of Java know that we are looking for a particular file in a particular package.

→ So, when you specify the path, what happens is that only those item within the package, that are declare as 'public' only they will be available to the the files that are outside of that file package

* In simple manner: *

when you import a particular file path all those items we can take from that file that are declared as public in that file

* If any item is being declared as private in that file, we can not access it *

Another eg is:

when we use 'ArrayList' in Java

Java import - 'java.util.ArrayList'

it means - ArrayList is a class which is in the util folder, which is present inside the Java folder.

▶ **static in Java**: properties which are not related to object or object independent those are known as static variables or static methods properties not related objects but are common to the all the objects

eg: Suppose you have a Human class with bunch of properties & construction

```
public class Human{
    int age;
    String name;
    int salary;
    boolean married;
public Human(int age, String name, int salary, boolean married){
    this.age = age;
    this.name = name;
    this.salary = salary;
    this.married = married;}
                }
```

Now, suppose you have created two different objects with different name and properties using Human class

```
Human Kunal = new Human (22, "Kunal Kushwaha", 10000, false);
Human rahul = new Human (34, "Rahul Rana ", 10000, false);
```

So, Now, what if there is some sort of property which is not dependend on Kunal or rahul it's independent. The property which will be same for both Kunal and rahul. it's not related to object, it's object independent.

→ eg: like, Population

Suppose, you have put population property inside Human class like ↓

```
public class Human{
    int age;
    String name;
    int salary;
    boolean married;
static long population;

public Human(int age, String name, int salary, boolean married){
        this.age = age;
        this.name = name;
        this.salary = salary;
        this.married = married;

    this.population += 1;  ✗ // we can not use this over here, because this represents the object reference

    Human.population += 1;  ✓ // Instead, inorder to access the static variable use the class name, as this population property is common to all Human beings
    }
}
```

Now if you print like this

```
sout (Kunal.population);
sout (rahul.population);
```

we get, Output : 2
2

Note: whenever you are accessing static variable, modifying static variable, declaring static variable, do it using class Name instead of object name.

It will work but still don't do it

sout(Kunal.population); ✗ // instead of this

sout(Human.population); ✓ // do it like this

* Without reference to any object, without object being created, you can still use static variable, like: accessing or modifying *

- Why is main declare as static ?

→ Main is declare as static method (if you think about what static is) because you can use this Main function or main method without creating an object of that class, & Main is actually the very first thing that actually run whenever you run a Java program. If main is not there you will be not able to run it.

So, if main is ~~inside~~ not a static or as main is inside a class

```
public class Main{
  public static void main(String[] args)
```

obviously inorder to run anything that is inside a class you have to create an object of that class

But, how can you run the program to create an object of main function, when the main function is the very first thing that is running

Hence, You should be able to run main

without creating any object of the class in which main function is.

And that is why it's static

So, with in Main class, before any object or anything is being created, main function should run first that's why it's static.

* static method, static variable actually belong to the class, do not belong to the object. *

➤ You can not use non-static method inside a static method, As a static method can only access a static data, can not access a non static-data.

Ex:

```
public class Main{
public static void main (String[] args){
 greeting();
 }
 void greeting(){
 sout("Hello. world");}
                }
```

The program will not run if you don't put 'static' before void greeting, like this ↓

```
static void greeting();
```

* Something which is non-static is belongs to object *

```
public class Main{
public static void main(String[] args){
 greeting(); }

void greeting(){
sout("Helloworld");
}
}
```

This is not going to have an instance variable as it's static

This is going to have an instance variable

- **Instance variable:** An instance variable is a variable that is specific to a certain object. It is declared with in the curly braces of the class but outside of any method. The value of an instance variable can be changed by any method in the class, but it is not accessible from outside the class.

- Static is not dependent on objects, but non-static is dependent on objects.

Another example:

```
Static void fun(){ //
  greeting(); }

void greeting(){
  sout("Hello world");
}
```

This is not dependent on objects & does not belongs to an instance

This is dependent on object & belongs to an instance

- This will not work, as without specifying which instance the greeting method belong to, we can not use it inside static void fun method.

As, method fun() doesn't depend on instances and to use greeting() method it requires an instance

So, As fun() method doesn't depend on instances, So you can not use something ~~that~~ inside it that depend on instances. like, greeting() method.

* But you can have something which is static inside non-static one *

ex:

```
static void fun(){
  // some code
        }

void greeting(){
  fun();      }
```

So, You cannot have a non-static member inside static without explicitly mentioning an object reference to it. You cannot access non static stuff without referencing their instances in a static context. like ↓

ex:

```
static void fun(){
Main obj = new Main();   // here we are explicitly
obj.greeting();}         // referencing to an object
                         // Now it will work

void greeting(){
sout("Hello World")
        }
```

- You can not use 'this' keyword inside static stuff

eg:

```
public class Human{
int age;
String name;
int salary;
boolean married;
static long population;

static void message(){ ------> //this is part of the class not part of the object
Sout("Helloworld");
Sout(this.age); //this will not work
}

public Human(int age, String name, int salary, boolean married){
this.age = age;
this.name = name;
this.salary = salary;
this.married = married;
Human.population += 1;}
}
```

this.age will not work inside static void message() method as 'this' belong to an object and message() method is independent of object.

So, thing which belong to object will not work inside which is independent of object.

- How to initialize static variables

→ we can initialize static variables using a static block.

eg:

```
public class StaticBlock {
 static int a = 4;
 static int b;
 static { // This is a static block
 sout ("I am in static block");
 b = a * 5;
      }
public static void main(String[] args){
 StaticBlock obj = new StaticBlock();
 sout (StaticBlock.a + "" + StaticBlock.b);
 StaticBlock.b += 3;
 StaticBlock obj2 = new StaticBlock();
 System.out.println(StaticBlock.a+""+StaticBlock.b);
    }}
```

- Static Block will only run once, when the first obj is create i.e when the class is loaded for the first time.

If it is run the second time we will have another static block and in output we will have "I am in static block" two times. which we suppose to have one time.

So, the static Block only run once when the first object is created.

▶ **Inner classes:** we can have class inside a class

eg:

```
public class Innerclasses {
    class Test {
                }
}
```

- outside class can not be static

```
static public class Innerclasses{ → // This will not work
    class Test{
                }
}
```

- only inner class can be static

```
public class Innerclasses{ → // This will work
    static class Test{
                }
}
```

* outside can not be static, because it is not itself dependent on any other class But inner class can be static, because, it dependent on outside class *

Now, let's take an example

Eg:

```
public class InnerClasses{
  class Test {
    String name;
  public Test (String name){
    this.name = name;    }
              }
public static void main (String[] args){
 Test a = new Test ("Kunal");
 Test b = new Test ("Rahul");      }
                    }
```

- The above code will not work & will show an error, as main function is static so it can not have a reference from inner class which is not static
- Also, the Test class dependent on the outter class. But if you put the Test class out side InnerClasses then it will not dependent on any other class
- The program will also run if we put static before the Test class

```
static class Test{
    // Some code
              }
public static void main (String[] args){
  Test a = new Test ("Kunal");  ------> // Now this will not show an error
  Test b = new Test ("Rahul"); }
```

when, we make Test class as static, it is not dependent object of InnerClasses.

So, after we make Test class as static & there after, to run the Test class an object of the InnerClasses doesn't need to be created. Hence, we can able to use Test class directly inside main function

Now, If we run the previous code after we are modifying Test class into static the, Output: will be ↓

Kunal
Rahul

- Although Test class mentioned inside the InnerClasses but after you mentioned it as static it doesn't depend on the object of InnerClasses class. but, main function and Test class they can have instances of each other.

* Since objects created at run time and static - methods, and variables do not have anything to do with objects. Hence static stuff, like; methods, class, variables they are resolve during compile time *

```
System.out.println("code to be print");
```

System is a class

out is a variable

println is a method

This is something that Java people have provided for us

System → if you wanna work with System, like system input, system output or file reading, we have written code for that already.

out → If you ctrl + click on out you will see out is basically

```
public static final PrintStream out = null;
```

out is just a reference variable of a class of type print stream

public → means it can be use from anywhere

static → means an object of PrintStream class doesn't need to be created in order to use 'out' variable

null → means, the ~~display output~~ standard output stream is, typically it's the display output on the console

out → It's a reference variable of class of type Print Stream.

. → 'dot' binds the instance variable, instance methods with the reference variable

Now, as out is type print stream so printstream must having a println method

- If you hold ctrl + click on out ——then—— you will get,

public static final PrintStream out = null;

Then, again if you hold ctrl + click on PrintStream ——then——> you will get,

PrintStream having a println method and it's just calling that. & it's just checking a String and printing that

println → It does two thing, it always calculate the value of it

valueOf → which in return calls the toString method

- Now,

Suppose you are doing a test by printing a value or something random any object of Test class

```
System.out.println(a);
```

It will give, Output: some random value

when you pass a object, Java is going to call value of, that is going to call toString method, actually internally it's calling a.toString, hence if 'a' which class

it belongs to, hence which is Test class it does not contain to string, hence 'a' is going to use it's own toString by default, And that is just :

class name @ hash value

- But what if we have toString method inside our test class

If, a.toString() doesn't have its toString method, it's gonna check does Test class have toString() or not. if not then 'a' gonna use the default to.String() method provided by Java

→ If we have toString method inside Test class

```
public class InnerClasses{
static class Test{
    String name;
    public Test(String name){
     this.name = name;   }
    public String toString(){
      return name;      }
}
public static void main(String[] args){
  Test a = new Test("Kunal");
  Test b = new Test("Rahul");
   sout(a);                    }
                    }
```

As, Test class contain it's own to string method no if run the program and try to print a gonna look insid if Test class contain toString

method or not, This time yes it does, so in output we gonna get

Output : Kunal

because Test class own toString() method, returns the name, so this time it's print A name, not any random garbage value.

▶ Singleton class : Sometime you may want only one instance of class to be created, only one object is allowed.
~~It~~'s singleton class is class which can create one object only.

- If you only want to create one object of a class you should not allow it to call the construction of this class by anyone
→ ~~eg~~. As, whenever you call a construction new object will be ~~ing~~ created and you can not allow anyone to create new object, hence we shouldn't allow anyone to use construction
for that we should make our construction private

```
public class Singleton{
  private Singleton(){
  }
}
```

private → the Singleton method can only be used on that particular Java file only, in that class only

```
public class Singleton{
  private Singleton(){
        }
private static Singleton instance;
public static Singleton getInstance(){
// check whether only 1 obj is created or not
 if(instance == null){
  instance = new Singleton();
                }
  return instance;                    }
                }
 public class Main{
 public static void main(String[]args){
 Singleton obj1 = Singleton.getInstance();
 Singleton obj2 = Singleton.getInstance();
 Singleton obj3 = Singleton.getInstance();
                                    }
                }
```

From the above code we can see

(i) As our goal is to create only one instance and for that we can create an Instance using the method getInstance

(ii) we have to put 'static' keyword before Singleton getInstance() method inorder to use it inside static method public static void main. as you can not use non-static stuff inside a static method.

(iii) Since Singleton instance is not going to create an object of the class Singleton, hence we can put static before it as well, as we have done.

(iv) So, in the main method whenever multiple objects with so many reference variable like: obj1, obj2, obj3 going to ask for instances you will give them the same instance. As, only one object is created in the memory & you will give reference variables to that object only which is a Singleton instance; we will always give. this ——→

(v) whenever someone call getInstance() method check weather an object is already created or not. So, if (instance == null) → in that case create object. Otherwise, return instance if a object is already being created.

- As, this is a Singletone class we are only allow to create one object only, after that no object will be created, second time when we ask for an object to getInstance() method, it will just return already created object
- So, obj1, obj2, obj3 are just pointing to the same one object

(vi) So, from the code we can see no way we are allowed to call a construcion and if we are not allowed to call a construcion we cannot create new objects.

It's useful when you want create one object of a particular thing