JAVA Programing

OOP : OOP is abbreviated as object oriented programming in which programs are considered as a collection of object . Each object is nothing but an instance of a class.

1 . Abstraction
2 . Encapsulation
3 . Inheritance
4 . Polymorphism

JAVA : Java is a high-level object-oriented programming language.

1.. high level
2.. portable
3.. object oriented
4.. platform independent
5.. secure
6 . dynamic
7 . multithreaded
8 . interpreted
9 . distributed
10 . robust

JVM : (Java Virtual Machine)

- it provides Runtime environment in which Java bytecode can be executed.
- the bytecode which is the intermediate language between source code and machine code .
- this bytecode is not platform specific and can be execute on any computer. Java compiler convert bytecode to Java programs (Java write once and run anywhere).

JRE : (Java Runtime environment)

- it is set of software tools and physically exist and constant a set of libraries others files that JVM users.

JDK : (Java development kit)

- JDK contain JRE and development tools

Object oriented programming	Object based programming
Follow all concept of oop	Does not Follow all concept of oop
don't have in inbuilt object	have in inbuilt object
Java , C#	Javascript , C

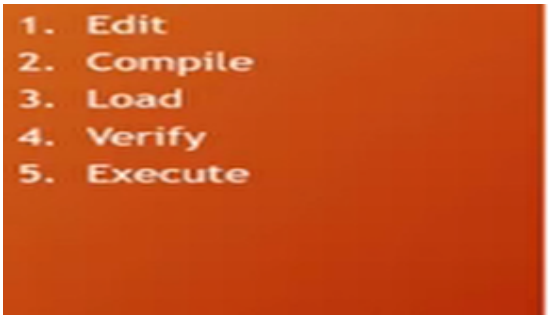
Access specifier : (class , method , variable)

- Public : can be accessed by any class or method.
- Protected : can be accessed in same package.
- Default : can be accessed package only.
- Private : can be accessed this class only.

Java shortly-

- a object oriented programming language.
- Work by (bytecode , JVM , JIT) + (JRE , JDK)
- API (JSE , JEE , JME , Java FX)
- IDE (eclipse , NetBeans , Intellij IDEA)

[object like variable which datatype is class] + [method is like function] + [class is collection]



@#1

1 . A program (hello world print) :

```
package hello;

public class Hello {
    public static void main( String[] args){
        System.out.println("Hello World");
    }
}
```

2 . Input and Output :

```
import java.util.Scanner;
        Scanner inp = new Scanner(System.in);
        int a = inp.nextInt();
```

```
package input;
import java.util.Scanner;
public class Input {
    public static void main(String[] args) {
        Scanner inp = new Scanner(System.in);
        int a;
        a = inp.nextInt();
        System.out.println("input is " + a);
    }
}
```

3 . Package and Class declaration :

@ package name start with small letter (hello)
@ class name start with big letter (Hello)
@ all class is stay on a single package (foldering)

Advantage of package :

@ in Java package avoid the name clashes
@ it is easier to locate the related class
@ it provide easy access control
@ can make hidden class (which is not visible) by using package

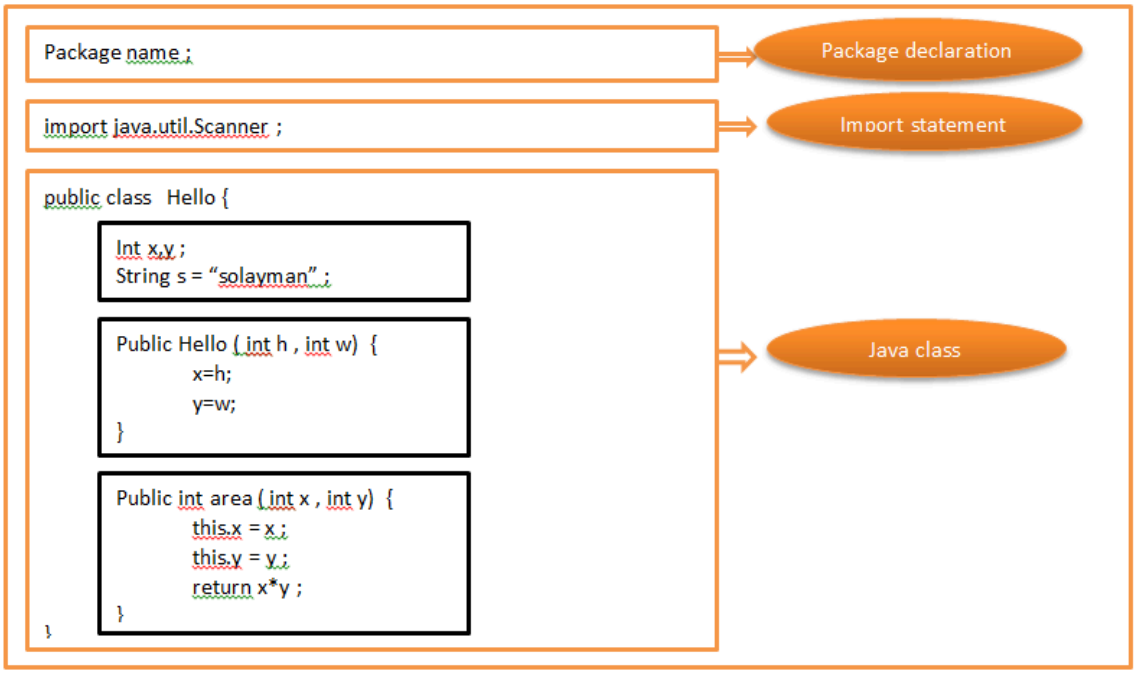
4 . Key Word :

Keywords in Java				
abstract	default	if	private	this
assert	do	implements	protected	throw
boolean	double	import	public	throws
break	else	instanceof	return	transient
byte	enum	int	short	try
case	extends	interface	static	void
catch	final	long	strictfp	volatile
char	finally	native	super	while
class	float	new	switch	
continue	for	package	synchronized	

@#2

1 . Class :

- class is the blueprint / template that Describe the details of an object .
- class is group of object which have common properties .



2 . Object :

- Object is an instance of a class it has its own state , behavior and identity
- the object of a class can be created by using the **new** keyword

$ object instantiation : (object creat) *(in java object type is [class , interface , enum])*

- Declaration
- Instantiation (object)
- initialization (normal variable)

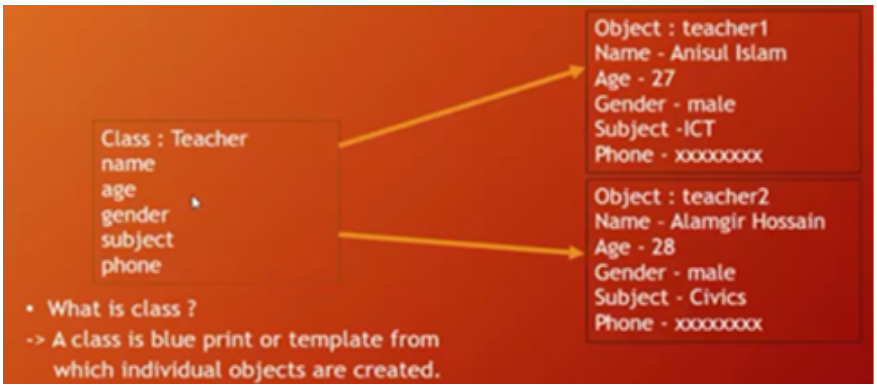
Main.java

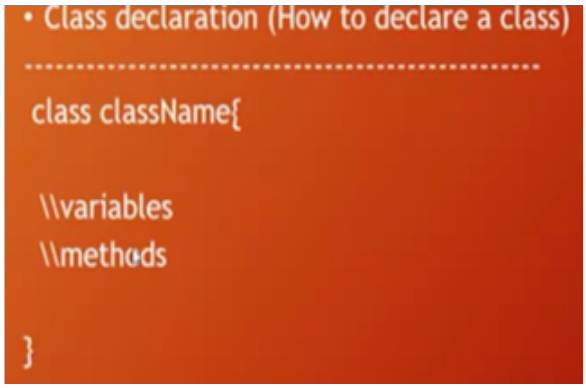
```java
package example;

public class Main {
    public static void main(String[] args) {
        Person p1= new Person();
        Person p2= new Person();
        p1.prin(20 , 's');
        p2.prin(18 , 't');
    }
}
```

Person.java

```java
package example;

public class Person {
    int age ;
    char name ;
    void prin( int age , char n){
        this.age=age;
        name = n;
        System.out.println("Name is : " + name );
        System.out.println("Age is : " + age );
    }
}
```

UNDERSTANDING Class & Object :





Teacher.java

```java
package oop;

public class Teacher {

    String name,gender;
    int phone;

}
```

Test.java

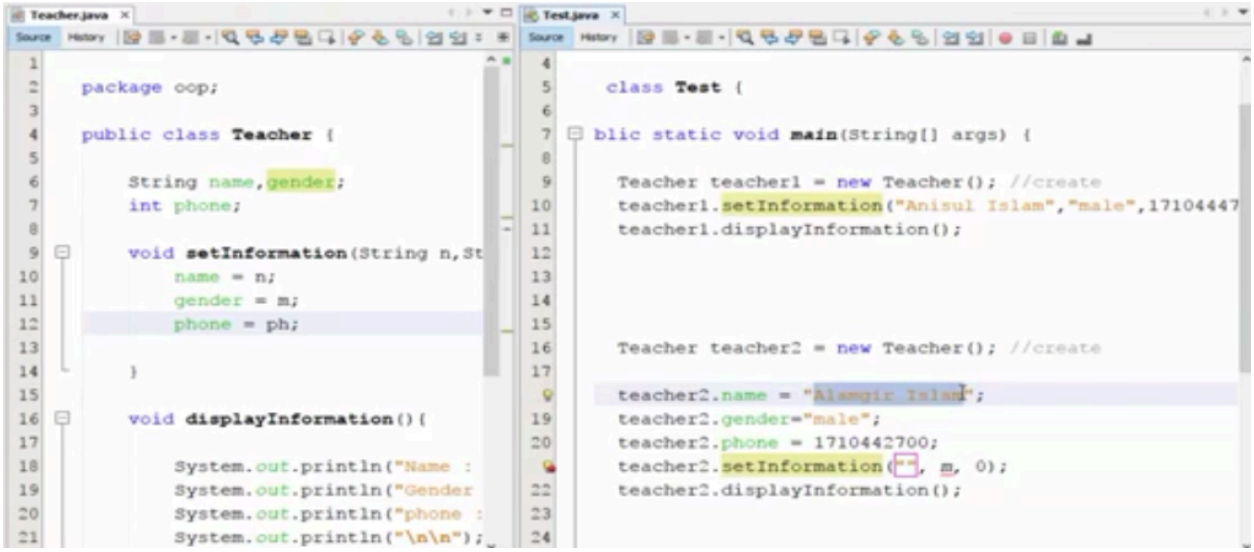
```java
} oop;


class Test {

lic static void main(String[] args) {

  Teacher teacher1 = new Teacher(); //create

  teacher1.name = "Anisul Islam";
  teacher1.gender="male";
  teacher1.phone = 01710444700;


  System.out.println("Name : "+teacher1.name);
  System.out.println("Gender : "+teacher1.gender);
  System.out.println("phone : "+teacher1.phone);
```

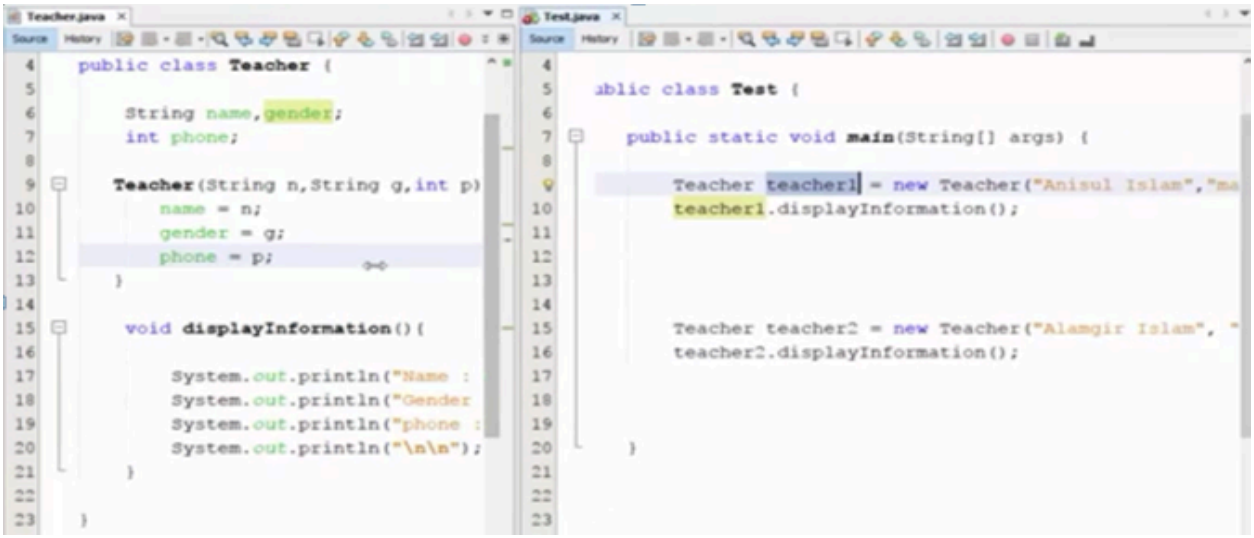
Teacher.java

```java
package oop;

public class Teacher {

    String name,gender;
    int phone;

    void displayInformation(){

        System.out.println("Name : "+name);
        System.out.println("Gender : "+gender);
        System.out.println("phone : "+phone);

    }

}
```

Test.java

```java
        teacher1.name = "Anisul Islam";
        teacher1.gender="male";
        teacher1.phone = 1710444700;
        teacher1.displayInformation();



        Teacher teacher2 = new Teacher(); /

        teacher2.name = "Alamgir Islam";
        teacher2.gender="male";
        teacher2.phone = 1710442700;
        teacher2.displayInformation();


    }
```





@#3

1 . Method :

 # method is to expose the behaviour of an object

 # method is like a function which stay inside the class (ক্লাসে এক বা একাধিক থাকতে মেথড পারে).

 # The method completing a specific one or more tasks. (মেথড মূলত কোন নির্দিষ্ট এক বা একাধিক কাজ সম্পন্ন করে)

 # The program must have Entrypoint to run the instructions.(প্রোগ্রামে সব ইনস্ট্রাকশন রান করার জন্য একটি এন্ট্রিপয়েন্ট থাকতে হয়)

 # Main method is the Entrypoint from which the code starts to run. মেইন মেথড সেই এন্ট্রিপয়েন্ট যেখান হতে কোড রান করতে শুরু করে

 # We can execute a program without main method using static block .

 # method or variable defined as static are shared among all the objects of the class .

 # there is no need to create the object to call the static method

 # abstract method static is not allowed. We can not override static method .

 # We can overload main method. And main method is static because the object is not required to call the static method. if we make main method not static JVM will have create its object first and then call main method which will lead extra memory allocation

Main Method :

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

}
```

Any Method :

```
void prin( int age , char n){
    this.age=age;
    name = n;
    System.out.println("Name is : " + name );
    System.out.println("Age is : " + age );
}
```

static or class method	Non-static or instance method
A method that is declared as static is known as the static method.	A method that is not declared as static is known as the instance method.
We don't need to create the objects to call the static methods.	The object is required to call the instance methods.
Non-static (instance) members cannot be accessed in the static context (static method, static block, and static nested class) directly.	Static and non-static variables both can be accessed in instance methods
For example: public static int cube(int n){ return n*n*n;}	For example: public void msg(){...}

```
public class Overload {

    void add(double a,double b){
        System.out.println(a+b);
    }

    void add(int a,int b,int c){
        System.out.println(a+b+c);
    }

    void add(){
        System.out.println("Nothing to add");
    }
}
```

Method overriding

যে মেথড গুলোতে value assing and value return ছাড়া অন্য কোন কাজ নাই সেই মেথড গুলোকে getter (accessor) / setter (mutator) মেথড বলে | data প্রাইভেট করতে ব্যবহার করা হয় (এনক্যাপসুলেশন)

```
public static getname(){
    return a;
}

public void setname(int a){
    this.a = a ;
}
```

Method Overloading	Method Overriding
1. Parameter must be different.	1. Parameter must be different.
2. It occurs within the same class.	2. It occurs between two classes - sub class and a super class.
3. Inheritance is not involved.	3. Inheritance is involved.
4. Return type may or may not be same.	4. Return type must be same.
5. One method does not hide another.	5. child method hides parent another.

2 . Constructor :

- constructor is a special type of method to initialise the state of an object
- constructor is like a method but it has not return type but return value
- the constructor is not inherited and can not be final
- constructor implicitly returns the current instance of the class value. it can not static(will shown compile error)
- if you not made any constructor then will make a default constructor
- the name of constructor must be similar to the class name
- constructor can overloading

Type

1. **By defult (not accept any value)**

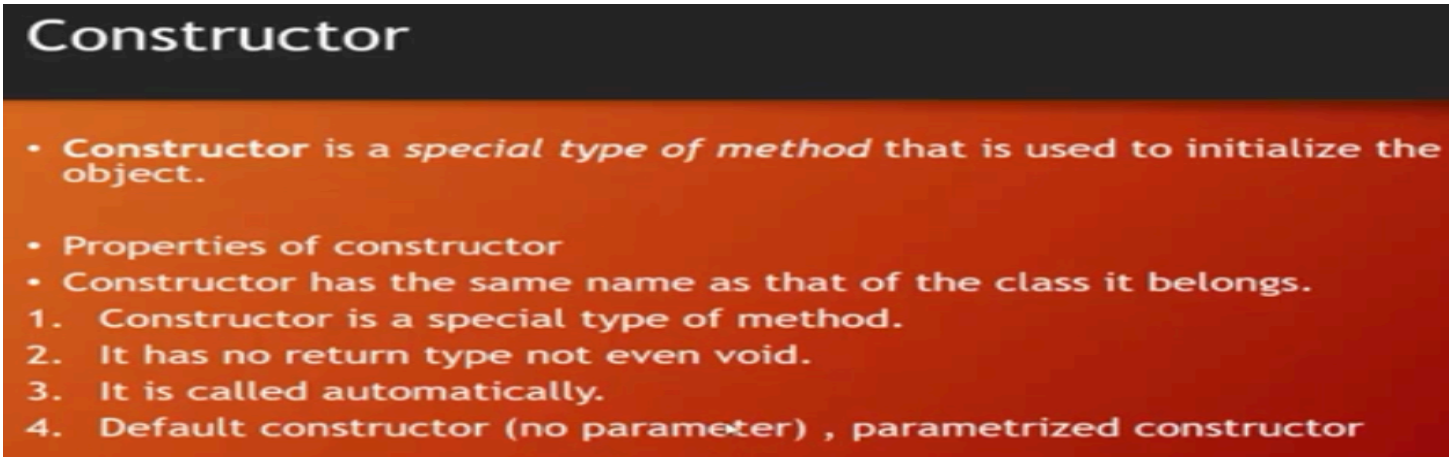
   Public Person(){

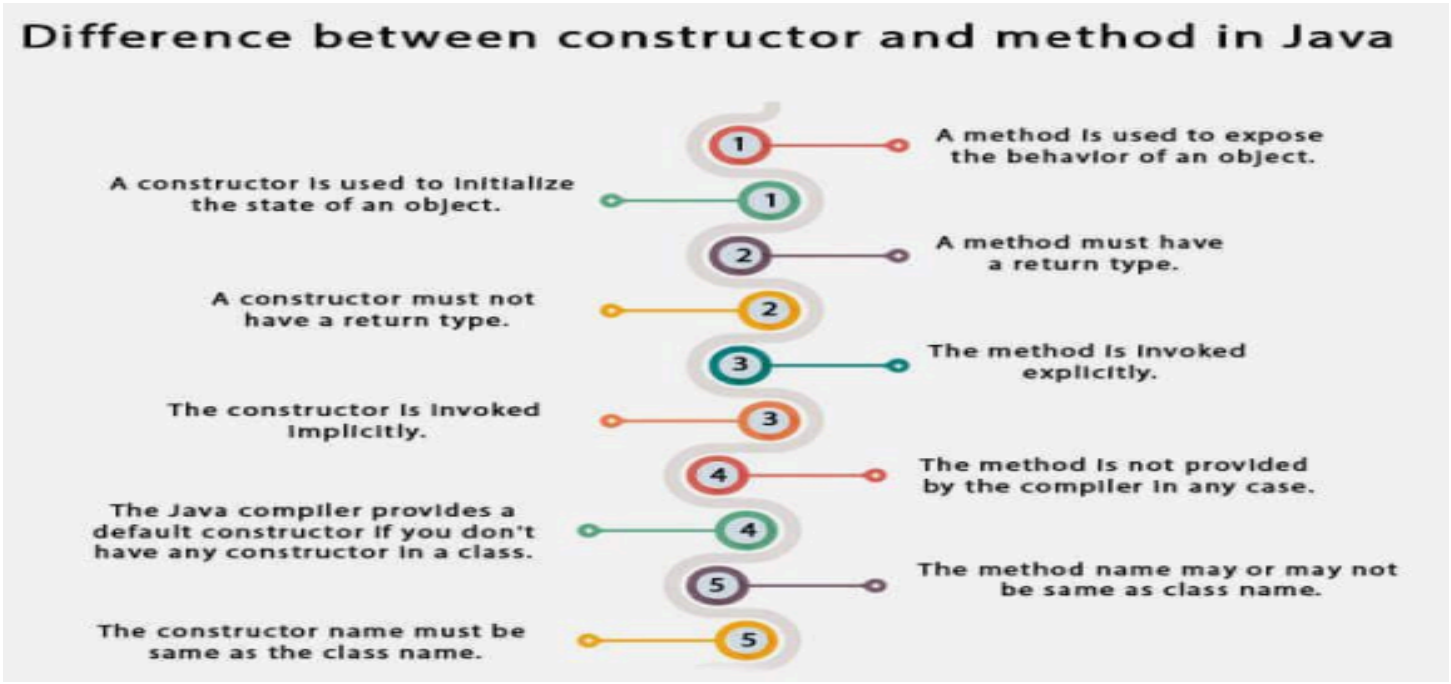
   }

2. **Creat /parameterize (accept any argument)**

   Public Person(int id){

```
        This.id = id ;
    }
```





@#4

1 . Datatype : (Wrapper Class)

- Wrapper classes provide a way to use primitive data types (`int`, `boolean`, etc) as objects.
- Referance type (Primative and non-primative) object is creat by using new Keyword.
- But Wrapper class creat object by direct assigne value
- Autoboxing(Primative to Wrapper class) Unboxing(Wrapper class to Primative)
- Sometimes you must use wrapper classes, for example when working with Collection objects, such as `ArrayList`, where primitive types cannot be used (the list can only store objects):

```
ArrayList<int> myNumbers = new ArrayList<int>(); // Invalid

ArrayList<Integer> myNumbers = new ArrayList<Integer>(); // Valid
```

- Literal (learn)
- Escape sequence (\n \t \r \")
- Comment (same as c++)

Primitive Data Type	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

```
class Autoboxing
{
   public integer sum(Integer a,Integer b){
        System.out.println(a+b);
   }
   public static void main(String[] args) {
       Autoboxing e = new Autoboxing();
       int a=5 , b=10 ;
       e.sum(a,b);
   }
}
```

```
class Unboxing
{
   public sum(int a , int b){
        System.out.println(a+b);
   }
   public static void main(String[] args) {

        Integer inum = new Integer(10 , 20);
        sum(inum);
    }
}
```

2 . Variable :

1. Local variable (Declared in method body and only work there)
2. Instance Variable (Object creat kore)*
3. Parameter variable (passing when method is call and onli access into method)
4. Class variable (stasic int a = 2 ;)

3 . OPerator :

```
1. Arithmetic : +   -   *   /   %
2. Relational : <   >   <=   >=   ==   !=
3.Logical     :  &&   ||   !
4.Assingment:  =   -=   +=   *=   /=
5.Increment  : ++
5.Decrement :  - -
6.Ternary     : exp1 ? true : false
7.Bitwise     : &   |   ^   <<   >>
8.size of     : m=sizeof(sum)
```

@#5

1 . condition (Decision making) : [if else & switch case]

2 . Looping : [for , while , do-while ,for-each]

3 . Branching : [break . continue , return]

@#6

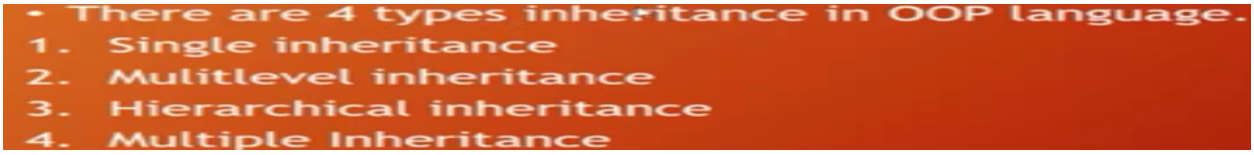
1 . Array :

2 . String :

1. Inheritance :

1. What is inheritance ?
   Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another.

2. Why do we need inheritance ?
   I. For code Reusability
   II. For method overriding
   III. To implement parent-child relationship.




```
package inheritance;

public class Person {
    String name;
    int age;

    void displayInformation1(){
        System.out.println("Name : "+name);
        System.out.println("Age : "+age);
    }
}
```

```
package inheritance;

public class Teacher {
    String name;
    int age;
    String qualification;

    void displayInformation2(){
        System.out.println("Name : "+name);
        System.out.println("Age : "+age);
        System.out.println("Qualification  : "+qualification);
    }
}
```

```
package inheritance;

public class Person {
    String name;
    int age;

    void displayInformation1(){
        System.out.println("Name : "+name);
        System.out.println("Age : "+age);
    }
}
```

```
package inheritance;

public class Teacher extends Person {

    String qualification;

    void displayInformation2(){
        displayInformation1();
        System.out.println("Qualification  : "+qualification);
    }
}
```

```
package method_oevrriding;

public class Person {
    String name;
    int age;

    void displayInformation(){
        System.out.println("Name : "+name);
        System.out.println("Age : "+age);
    }
}
```

```
package method_oevrriding;

public class Teacher extends Person{

    //name,age,displayInformation()
    String qualification;

    @Override
    void displayInformation(){
        System.out.println("Name : "+name);
        System.out.println("Age : "+age);
        System.out.println("Qualification : "+
    }
```

2 . Encpsulation :

Encapsulation is a process of

1. Packaging Variables and methods into a single unit.
2. Protecting data by declaring them as private.

```
package oop;

public class Person {
    private String name;
    private int age;

    void eat(){

    }

    void talk(){

    }
}
```

```
package oop;

public class EncapTest {
    public static void main(String[] args) {
        Person p1 = new Person();
        p1.name = "anis";
        p1.age = 27;
        p1.talk();
    }
}
```

Person.java

```
4   public class Person {
5
6       private String name;
7       private int age;
8
9       public void setName(String name){
10          this.name = name;
11      }
12
13      public String getName(){
14          return name;
15      }
```

EncapTest.java

```
1
2   package encapsulation;
3
4   public class EncapTest {
5       public static void main(String[] args)
6
7           Person p1 = new Person();
8           p1.setName("Anis");
9           System.out.println();p1.getName()
10
11
12      }
```

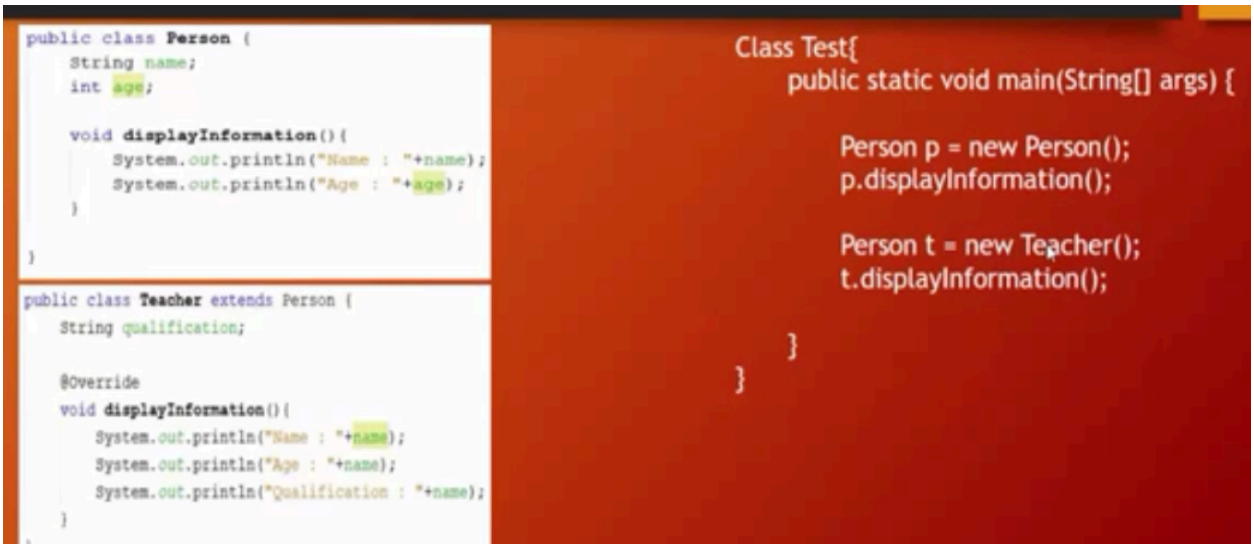
Benefits of Encapsulation

1. Provides data hiding
2. Reusability
3. Code can be modified without breaking the code.
4. Maintainability : Hiding implementation details reduces complexity.

- How to do encapsulation ?

1. Declare the variables as private.
2. Provide public setter and getter method to modify and get the variables value.

3 . Polymorphisom :



4 . Abstruction :

1 . Exception Handling :

2 . Generics :