**BY SHANTANU RANA**

**What is OOPS?**

Solving a problem by creating objects is one of the most popular approaches in programming. This is called OOPS.

OOPS tries to map code instructions with real world making the code short and easier to understand.

**Why do we need OOPS ?**

* Our solutions are complex and relates to real life projects.
* It is easier to write , maintain and reuse.
* It is flexible and suits many use cases.
* Relatively easily modelled.



**Class** **: -** A class is a blueprint for creating objects.

**JEE APPLICATION FORM**

**APPLICATION FOR THAT STUDENT**

* Filled by a Student 🡪

**OBJECT**

**CLASS**

* Object Installation 🡪

Contains information to

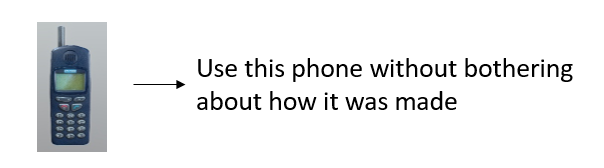
create valid object

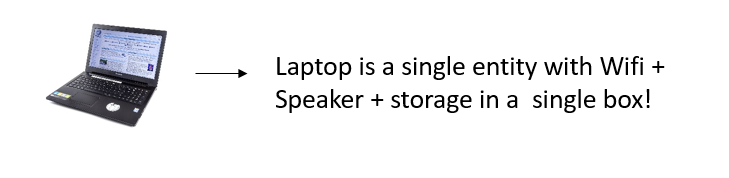
**OBJECT :-** Object is an instantiation of a class. When a Class is defined , a template (information) is defined. Memory is allocated after Object Installation.

#### FFFour pillars of Object-Oriented-Programming Language :

1. **Abstraction** :
   * Let's suppose you want to turn on the bulb in your room. What do you do to switch on the bulb. You simply press the button and the light bulb turns on. Right? Notice that here you're only concerned with your final result, i.e., turning on the light bulb. You do not care about the circuit of the bulb or how current flows through the bulb. The point here is that you press the switch, the bulb turns on! You don't know how the bulb turned on/how the circuit is made because all these details are hidden from you. This phenomenon is known as abstraction.
   * More formally, data abstraction is the way through which only the essential info is shown to the user, and all the internal details remain hidden from the user.

Example :



1. **Polymorphism** :
   * One entity many forms.
   * The word polymorphism comprises two words, poly which means many, and morph, which means forms.
   * In OOPs, polymorphism is the property that helps to perform a  single task in different ways.
   * Let us consider a real-life example of polymorphism. A woman at the same time can be a mother, wife, sister, daughter, etc. Here, a woman is an entity having different forms.
   * Let's take another example, a smartphone can work like a camera as well as like a calculator. So, you can see the a smartphone is an entity having different forms. Also :  
       
     
2. **Encapsulation :**
   * The act of putting various components together (in a capsule).
   * In java, the variables and methods are the components that are wrapped inside a single unit named class.
   * All the methods and variables of a class remain hidden from any other class.
   * A automatic cold drink vending machine is an example of encapsulation.
   * Cold drinks inside the machine are data that is wrapped inside a single unit cold drink vending machine.
3. **Inheritance :**
   * The act of deriving new things from existing things.
   * In Java, one class can acquire all the properties and behaviours of other some other class
   * The class which inherits some other class is known as child class or sub class.
   * The class which is inherited is known as parent class or super class.
   * Inheritance helps us to write more efficient code because it increases the reusablity of the code.
   * Example :
   * Rickshaw      →        E-Rickshaw
   * Phone           →        Smart Phone

**PROGRAMMING IN OOPS**

1.Creating class and printing hello world

**Public class first**  // here public class is access modifier and we have created a class name as First

**{**

**Public static void main(string arg[ ]) {** /\* **Public**- it is access specifier from anywhere we can access it.

**Static**- it is access modifier we can call the methods directly by class name without creating its objects

**Void**- it is the return type

**Main**- it is a method name

**String[]args**- in java we accept only the string type of argument and store it \*/

**System.out.println(“Hello World”);**

**}**

**}**

2. Creating Instance / Object

**Explanation :-**

SO we will create two classes first and second.

**Public class Second**

**Public void SecondMethod( )**

**{**

**System.out.println(“Second Method”);**

**}**

Side by side We will create another class First

**Public class first**

**{**

**Public static void main(string arg[ ]) {**

\\* Creating object or instance

Syntax is : **ClassName reference\_variable\_name=new ClassName(** **)<- constructor**

\*\

**Int i;**

**i = 10;**

**Second s;** \\ declaring s (reference variable) of type Second this second is user defined datatype but we don’t call it a datatype. We can consider that it’s behaviour is like a datatype here.

**Second s = new Second( );** \\ here new is memory allocation operator

\\ important : - Objects are having no name in java

\\ There are only refrences that can refer to object. Eg **s** in this case

}

}

EXAMPLE :-

We will create two classes naming second and first :-

**public** **class** first{

**public** **static** **void** main(String arg[]) {

Second s=**new** Second();

s.a=40;

Second t=s;

System.***out***.println(t.a);

}

}

We will also create a second class in another file :-

**public** **class** Second {

**int** a;

**public** **void** secondMethod(**int** b) {

a=b;

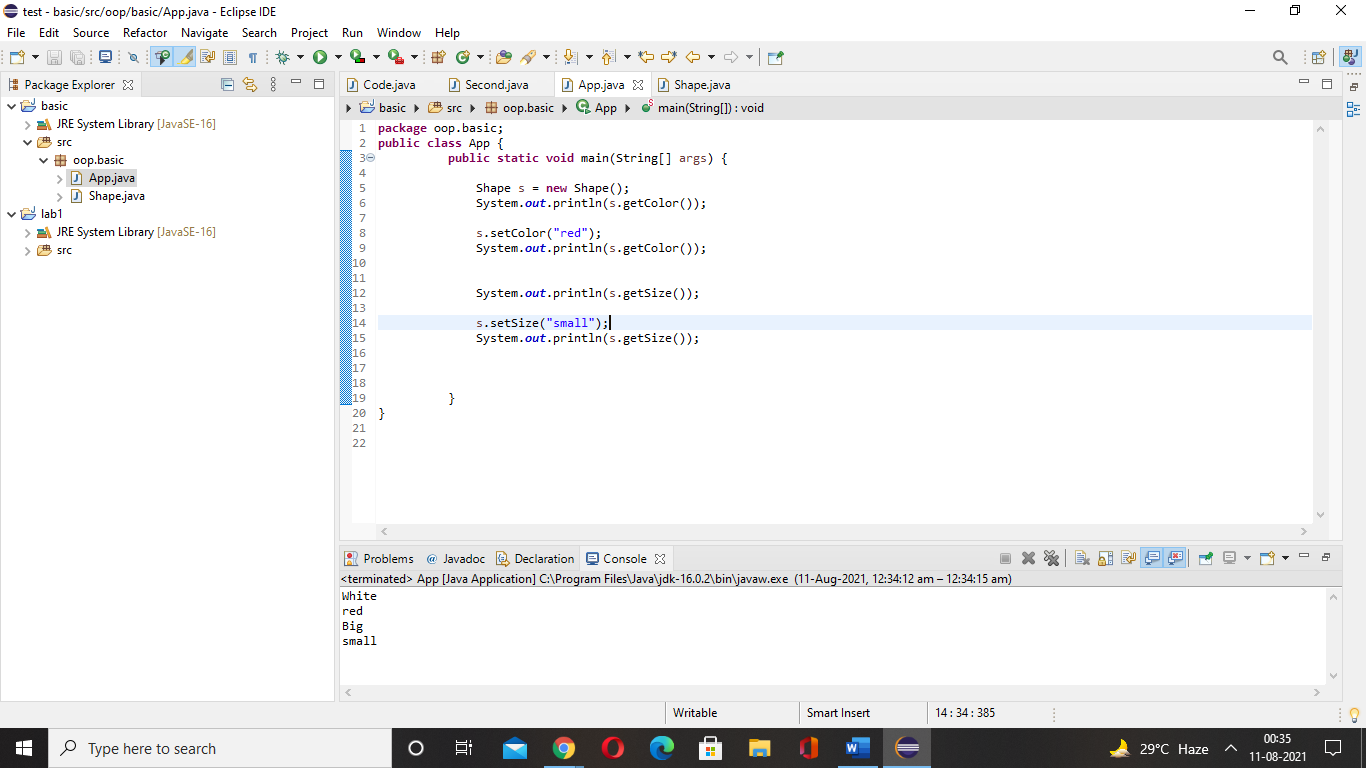
System.***out***.println("Second Method "+a);

}

}

**Output : - 40**

**EXAMPLE 2:-**



**package** oop.basic;

**public** **class** App {

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

Shape s = **new** Shape();

System.***out***.println(s.getColor());

s.setColor("red");

System.***out***.println(s.getColor());

System.***out***.println(s.getSize());

s.setSize("small");

System.***out***.println(s.getSize());

}

}

On Shape file :-

**package** oop.basic;

**public** **class** Shape {

**private** String color= "White";

**private** String size="Big";

**public** String getColor(){

**return** color;

}

**public** **void** setColor(String c){

color =c;

}

**public** String getSize(){

**return** size;

}

**public** **void** setSize(String s){

size =s;

}

}

**OUTPUT :**

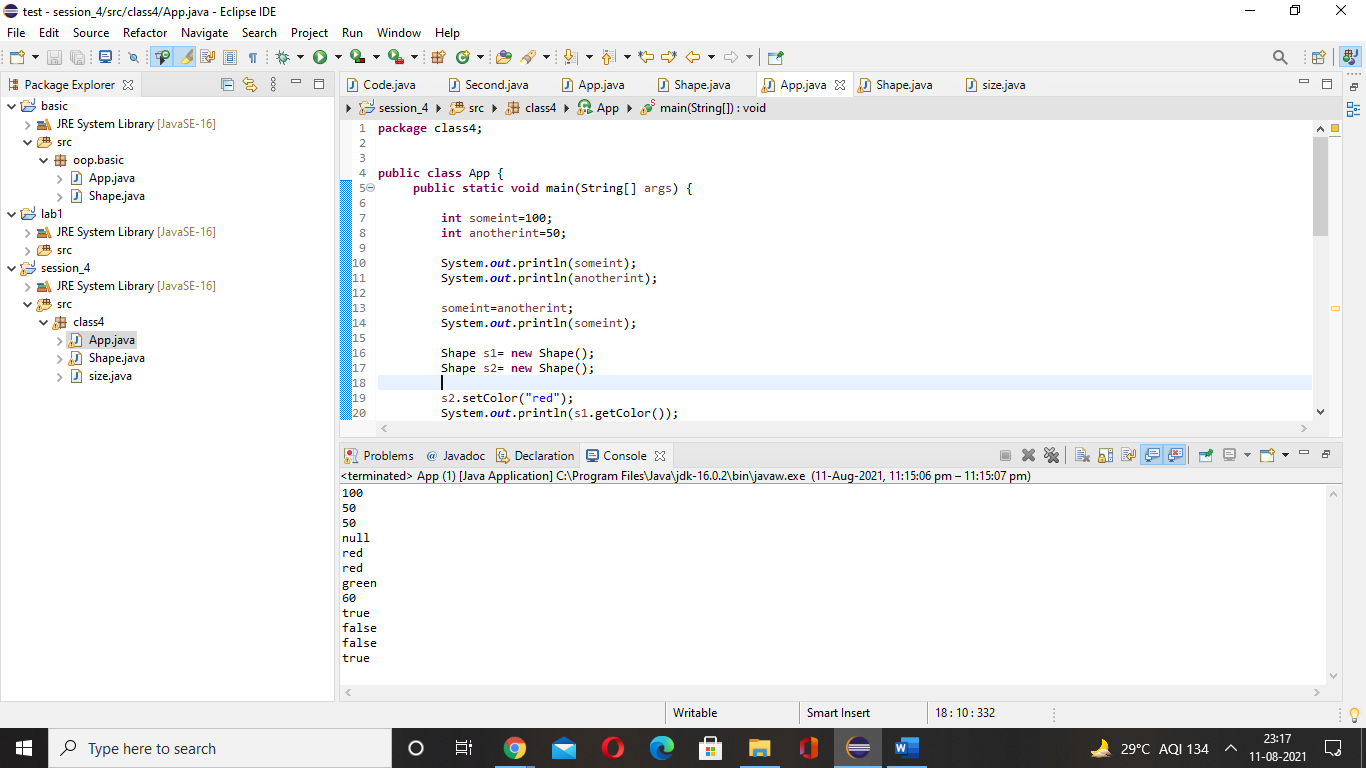
**White**

**red**

**Big**

**Small**

**EXAMPLE 3:-**

****

**package** class4;

**public** **class** App {

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

**int** someint=100;

**int** anotherint=50;

System.***out***.println(someint);

System.***out***.println(anotherint);

someint=anotherint;

System.***out***.println(someint);

Shape s1= **new** Shape();

Shape s2= **new** Shape();

s2.setColor("red");

System.***out***.println(s1.getColor());

System.***out***.println(s2.getColor());

s1=s2;

System.***out***.println(s1.getColor());

s1.setColor("green");

s2.getColor( );

System.***out***.println(s1.getColor());

System.***out***.println(someint + 10);

// boolean data type

Shape s= **new** Shape();

**boolean** b= **true**;

System.***out***.println(b);

b=**false**;

System.***out***.println(b);

**boolean** b1= **true**;

**boolean** b2= **false**;

System.***out***.println(b1 && b2);

System.***out***.println(b1 || b2);

}

}

**package** class4;

**public** **class** Shape {

**private** String color;

**private** String size="Big";

**private** **int** i=100;

**public** String getColor(){

**return** color;

}

**public** **void** setColor(String c){

**if**(c.equals("black") ||c.equals("white") ) {

}

**else** {

color =c;

}

}

**public** String getSize(){

**return** size;

}

**public** **void** setSize(String s){

size =s;

}

}

**package** class4;

**public** **class** size {

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

Shape s = **new** Shape();

s.setColor("black");

System.***out***.println(s.getColor());

}

}

**OUTPUT :**

**On compiling App.java :-**

100

50

50

null

red

red

green

60

true

false

false

true

**On compiling Size.java :-**

null

**CONSTRUCTOR**

A constructor defines an initialization code, to be executed whenever an object is created.

Looks like a method which :

* Has the same name as the class
* Has no return type specified(not even void)
* May receive parameters
* May be overloaded

**This keyword**

The this keyword refers to the current object in a method or constructor.

The most common use of the this keyword is to eliminate the confusion between class attributes and parameters with the same name (because a class attribute is shadowed by a method or constructor parameter).

EXAMPLE :-

Here we are creating two class App and Shape

**package** session6;

**public** **class** App {

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

Shape s= **new** Shape(10);

System.***out***.println(s.getWidth());

}

}

**package** session6;

**public** **class** Shape {

**private** **int** width;

// This is constructor --🡪**public** Shape(**int** width) {

**this**.width=width;

}

**public** **int** getWidth() {

**return** width;

}

**public** **void** setWidth(**int** width) {

**this**.width=width;

}

}

**OUTPUT :-**

10

**EXAMPLE :-**

**package** constructor2;

**public** **class** App {

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

Shape s1= **new** Shape();

Shape s2= **new** Shape(30);

Shape s3= **new** Shape(40,60);

System.***out***.println(s1.getWidth());

System.***out***.println(s2.getWidth());

System.***out***.println(s3.getWidth());

System.***out***.println(s3.getheight());

}

}

**package** constructor2;

**public** **class** Shape {

**private** **int** width;

**private** **int** height;

**public** Shape() {

width=0;

}

**public** Shape(**int** width) {

**this**.width=width;

}

**public** Shape(**int** width, **int** height) {

**this**.width=width;

**this**.height=height;

}

**public** **int** getWidth() {

**return** width;

}

**public** **void** setWidth(**int** width) {

**this**.width=width;

}

**public** **void** setheight(**int** height) {

**this**.height=height;

}

**public** **int** getheight() {

**return** height;

}

}

**OUTPUT :-**

0

30

40

60