★ OOPs

Four pillars of OOP:
- Encapsulation
- Inheritance
- Abstraction
- Polymorphism

- Object Oriented Programming.
- It is a methodology or paradigm to design a problem using classes & objects.

* Classes & Object

Object → entities in the real world.
Classes → group of these entities.

- Classes → attributes (properties) + functions (behaviours)

- By convention, we use classes first letter in Capital i.e., Uppercase.
- We can make multiple classes in a single file.
- Object are stored in Heap memory.

CODE 1:

```
public class OOPS {
    public static void main (String args[]) {
        Pen p1 = new Pen(); //created a pen object p1
        p1.setColor ("Blue");
        System.out.println (p1.color);
        p1.setTip (5);
        System.out.println (p1.tip);
        p1.color = "Yellow";
                //or
    // p1.setColor ("Yellow");
        System.out.println (p1.color);
    }
}
class Pen {
```

```
    String Color;
    int tip;
    void setColor (string newColor) {
        color = newColor;
    }
    void setTip (int newTip) {
        tip = newTip;
    }
}
```

- If we have to excess any object property or function then we use Dot (.) Operator.

* Access Modifiers or Specifiers
- They are used aspect of data hiding.
- They are used to assign the accessibility to the class members, i.e., they set some restrictions on the class members so that they can't be directly accessed by the outside functions.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Yes	No	No	No
Default	Yes	Yes	No	No
Protected	Yes	Yes	Yes	No
Public	Yes	Yes	Yes	Yes

CODE 2:
```
psvm {
    BankAccount MyAcc = new BankAccount();
    myAcc.username = "Anubhav Singh";
//  myAcc.password = "abcde"; // Error
    myAcc.setPassword("abcdefgh");
}
```

```
class BankAccount {
    public String username;
    private String password;
    public void setPassword (String pwd) {
        password = pwd;
    }
}
```

* Getters & Setters

- Get : to return the value.
- Set : to modify the value.

- this : it is a keyword used to the current object. if there are ~~different~~ similar name of variable then 'this' helps to know which is one & another.

CODE ⇒ 3

```
class Pen {
private String color;
    private int tip;
    String getColor () {
        return this.color;
    }
    int getTip () {
        return this.tip;
    }
    void setColor (String newColor) {
        this.color = newColor;
    }
    void setTip (int tip) {
        this.tip = tip;
    }
}
```

```
public class getAndSet {
psvm {
    Pen p1 = new Pen();
    p1.setColor ("Blue");
    System.out.println (p1.getColor());
    p1.setTip (5);
    System.out.println (p1.getTip());
    p1.setColor ("Yellow");
    System.out.println (p1.getColor());
  }
}
```

* Encapsulation

"Encapsulation is defined as the wrapping up of data & methods under a single unit. It also implements data hiding."

data (properities or attributes)	Methods (functions)

- (Useless / sensitive data → private, default, protected) Data Hiding.

* Constructors

"Constructor is a special method which is invoked automatically at the time of object creation."

- Constructors have the same name as class or structure.
- Constructors don't have a return type (not even void).
- Constructors are only called once, at object creation.
- Memory allocation happens when constructor is called.

CODE 4:
```
public static void main (String args[]) {
    Student s1 = new Student();
}
class Student {
    String name;
    int roll;
    Student () {
        System.out.println ("Constructor is called...");
    }
}
```

* Types of Constructors
  1. Non-Parameterized Constructor
  2. Parameterized Constructor
  3. Copy Constructor

(1) A constructor that has no parameters is known as default or non-parametrized type of constructor.

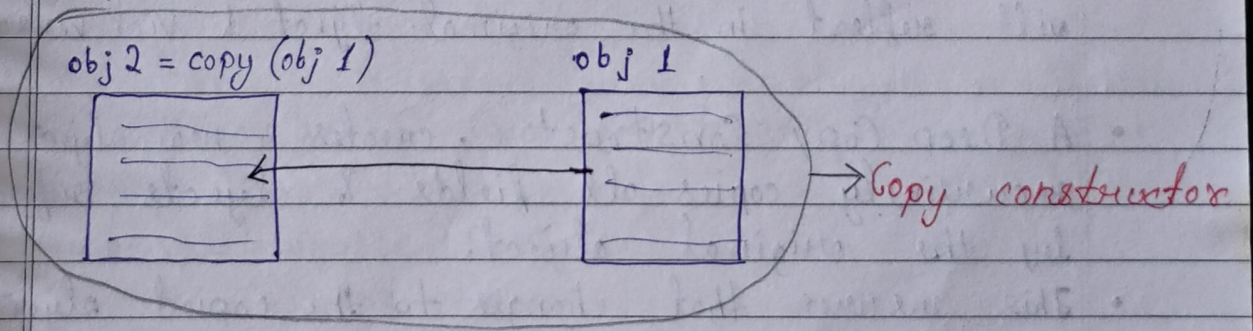
(2) A constructor that has parameters is known as Parametcrized constructor. It is used, when we want to initialize fields of the class with our own values.

(3) Copy Constructor is passed with another object which copies the data available from the passed object to the newly created object.

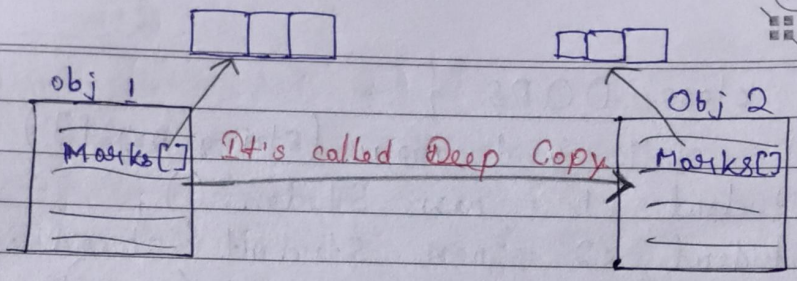
In Java, there is NO inbuilt copy constructor available like in C++.

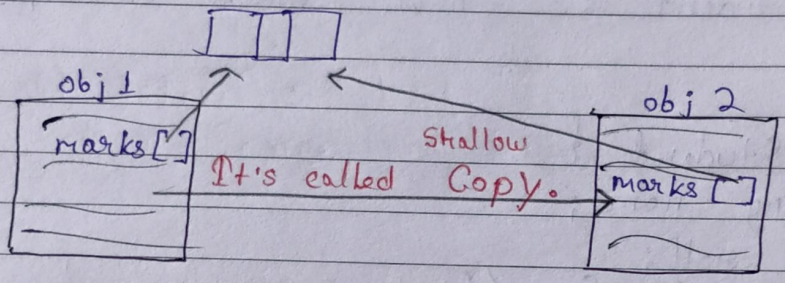
CODE 5:
```
public class OOPS {
    public static void main (String args[]) {
        Student s1 = new Student();
        Student s2 = new Student("shradha");
        Student s3 = new Student(123);
    }
}
class Student {
    String name;
    int roll;
    Student() {  // non-parameterized
        System.out.println("Constructor is called...");
    }
    Student(String name) {
        this.name = name;
    }
    Student(int roll) {
        this.roll = roll;
    }
}
```
// Parameterized

* Copy Constructor (CODE 7) → Pg no. 116



- Copying properities of obj 1 in obj 2.





* Shallow & Deep Copy

- In Shallow copy, changes reflect.
- In Deep copy, changes don't reflect.

- A Shallow copy constructor creates a new object & copies all of the fields of the existing object into the new object. However, if the field itself is an object, it only copies the reference of that object, not actual object.
- This means changes to the copied object's fields will reflect in the original object & vice versa.

- A Deep Copy Constructor, creates a new object & recursively copies all fields & objects referenced by the original object.
- This ensures that changes to the copied object do not affect the original object & vice versa.

CODE 6:
```
class Person {
    String name;
    Address address;
    //shallow copy constructor
    public Person (Person person) {
        this.name = person.name;
        this.address = person.address;
    //shallow copy of address object
    }

    //Deep Copy Constructor
    public Person (Person person) {
        this.name = person.name;
        this.address = new Address (person.address);
    //deep copy of address object.
    }
}
class Address {
    String city;

    public Address (Address address) {
        this.city = address.city;
    }
}
```

CODE 7: Example of Copy Constructor
```
public class Fruit {
private double fprice;
private String fname;
Fruit (double fPrice, String fName) {
    fprice = fPrice;
    fname = fName;
}
```

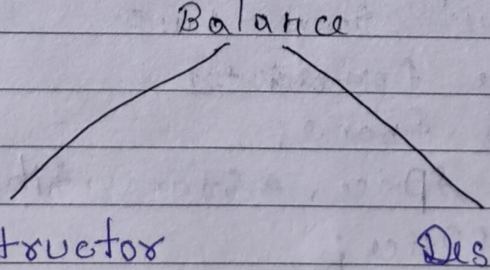
```
Fruit (Fruit fruit) {
    Syso ("In After invoking the Copy Constructor: \n");
    fprice = fruit.fprice;
    fname = fruit.fname;
}

double showPrice() {
    return fprice;
}

String showName() {
    return fname;
}
public static void main (String args[]) {
    Fruit f1 = new Fruit (399, "Anubhav");
    Syso ("Name of the first fruit: " + f1.showName());
    Syso ("Price of the first fruit : " + f1.showPrice());

    Fruit f2 = new Fruit (f1);
    Syso ("Name of second fruit : " + f2.showName());
    Syso ("Price of second fruit: " + f2.showPrice());
}
```

There should be the balance in the world.
— Thanos



- In Java, we can use Destructor but we won't because in java there is used Garbage Collector which automatically used in Java.

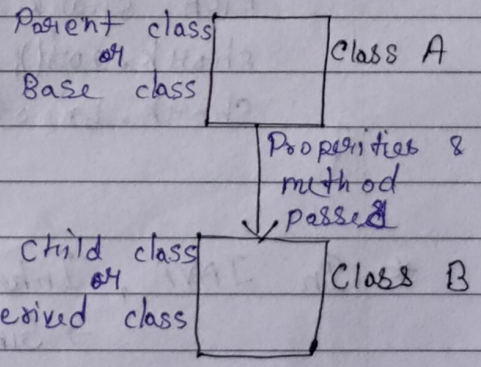
* ~~Gabe~~ Garbage Collector,

In Java, there are no destructors like those found in C++ instead, Java provides automatic garbage collection, which means that the system automatically deallocates memory when an object is no longer referenced.

- Garbage Collector automatically manages memory by reclaiming memory occupied by objects that are no longer in use.

* Inheritance

Inheritance is when properties & methods of base class are passed on to a derived class.



Ex→ Animal {
eat()
breathe()
skincolor(String)
}

Fish {
fins(int)
swim()
}

Animal class properties

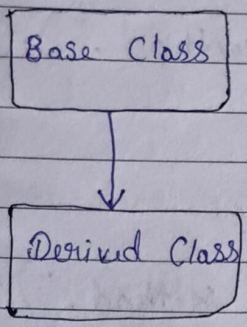
We use extends Keyword for inheritance.

CODE 8:

```
//Base Class
class Animal {
    String color;
    void eat() {
        Syso ("eats");
    }
    void breathe() {
        Syso ("breathes");
    }
}
//Derived class //Sub class
class Fish extends Animal {
    int fins;
    void swim() {
        Syso ("swims in water");
    }
}
public static void main (String args[]) {
    Fish shark = new Fish();
    shark.eat();
    shark.breathe(); // Inheritance
}
```

* In JAVA, Inheritance are of 4 types:
  - → Single Level Inheritance
  - → Multi Level Inheritance
  - → Hierarchial Inheritance
  - → Hybrid Inheritance
  - → Multiple Inheritance.
    - ⇓
    - It doesn't support in Java.
    - We do this indirectly.
    - We will study Interfaces.

1. Single Level Inheritance

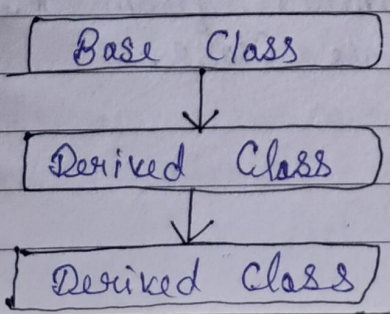


* A class inherits the properties / method from a single class.
* Class which inherits → Derived class
* Class from which inherits → Base class

CODE:

```
class A {
    public void methodA() {
        System.out.println("Base Class Called");
    }
}
class B extends A {
    public void methodB() {
        System.out.println("Child Class Called");
    }
    public static void main(String args[]) {
        B obj = new B();
        obj.methodA();
        obj.methodB();
    }
}
```

2. Multi-level Inheritance

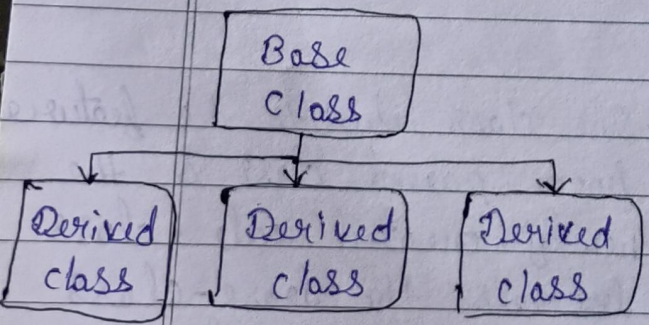


* One class inherits the features from parent class & the newly created sub-class becomes the base-class for another new class.
* Chain of Inheritance.

CODE 10:

```
class X {
    public void methodX() {
        System.out.println("Class X method.");
    }
}
class Y extends X {
    public void methodY() {
        System.out.println("Class Y Method.");
    }
}
class Z extends Y {
    public void methodZ() {
        System.out.println("Class Z Method.");
    }
    public static void main(String args[]){
        Z obj = new Z();
        obj.methodX();
        obj.methodY();
        obj.methodZ();
    }
}
```

3. Hierarchical Inheritance



* The type of inheritance where many subclasses inherit from one single base class.

CODE 11:

```
class Animal {
    void eat() {
        System.out.println("Eating...");
    }
}
class Dog extends Animal {
    void bark() {
        System.out.println("Barking...");
    }
}
class Cat extends Animal {
    void meow() {
        System.out.println("Meowing...");
    }
}
class Heirarchical {
    public static void main(String args[]) {
        Cat c = new Cat();
        c.meow();
        c.eat();
    }
}
```

4. Hybrid Inheritance

It is a combination of more than two types of inheritances single, multiple & heirarchical inheritance.

* In Java, pure multiple inheritance is not supported due to "diamond problem", where conflicts arise when a class inherits from multiple classes with same method or field names.

* To ~~achi~~ achieve hybrid inheritance in Java, we use a combination of multiple inheritance through interfaces & multilevel inheritance through class hierarchies.

CODE 12:

```
interface LivingBeing {
    void breathe();
}
interface Animal extends LivingBeing {
    void eat();
}
interface Pet {
    void play();
}
class Dog implements Animal, Pet {
    public void breathe() {
        System.out.println("Dog is Breathing");
    }
    public void eat() {
        System.out.println("Dog is eating.");
    }
    public void play() {
        System.out.println("Dog is Playing.");
    }
}
public class Hybrid {
    public static void main(String args[]) {
        Dog Pitbull = new Dog();
        Pitbull.breathe();
        Pitbull.eat();
        Pitbull.play();
    }
}
```

many forms

* Polymorphism - When we try to achieve same things or similar things in multiple forms.

- Java supports two types of polymorphism are :
1. Compile-time Polymorphism : Also known as static polymorphism. This polymorphism is facilitated through method overloading, where the decision on which method to invoke is made during compile time rather than at runtime.
- Method Overloading - multiple functions with the same name but different type of parameters or different no. of parameters.

Operator Overloading isn't supported in JAVA.

CODE 13:
```
public static void main (String args[]) {
    Calculator calc = Calculator ();
    System.out.println (calc.sum (1,2));
    System.out.println (calc.sum ((float) 1.5, (float) 2.5));
    System.out.println (calc.sum (1,2,3));
}
class Calculator {
    int sum (int a, int b) {
        return a + b;
    }
    float sum (float a, float b) {
        return a + b;
    }
    int sum (int a, int b, int c) {
        return a + b + c;
    }
}
```

2. Run-time polymorphism : It is also referred as Dynamic polymorphism, involves the resolution of overridden methods at runtime rather than at compile time.

- Method Overriding - Parent & child classes both contain the same function (name of class, no. & type of arguments must be same) with a different function definition.

CODE 14:
```
public static void main (String args[]) {
    Deer d = new Deer();
    d.eat();
}
class Animal {
  void eat() {
      System.out.println("Eats Anything");
  }
}
class Deer extends Animal {
  void eat() {
      System.out.println("Eats Grass");
  }
}
```

* Packages in Java

Package is group of similar types of classes, interfaces & sub-packages.

- Real life Ex → Amazon.com ; Ratings, Add to cart.
- There are two types of Packages are :
  i) in-Built packages
  ii) user-Defined packages

i) Built-in Packages – these packages consist of a large no. of classes which are a part of Java API.
Ex→ java.lang, java.io, java.util, java.applet, java.net.

ii) User-defined Packages – These are the packages that are defined by the user.
- First we create a directory "myPackage".
- Then create the "MyClass" inside directory with the first statement being the package names.

CODE 15:
```
// Name of package must be same as the directory.
// under which this file is saved package myPackage.
package myPackage;
public class MyClass {
    public void getNames (String s) {
        System.out.println (s);
    }
}
```

CODE 16:
```
/*import 'MyClass' from 'myPackage'*/
import myPackage.MyClass;
public class PrintName {
    public static void main (String args[]) {
        String name = "AnubhavSingh";
        MyClass obj = new MyClass();
        obj.getNames (name);
    }
}
```

खयाली पुलाव { Idea ✓ Implementation ✗

* Abstraction

Hiding all the unnecessary details & showing only the important parts to the user.

Abstract Class

Interfaces

1. Abstract Class

To make any class into abstract class we simply add 'abstract' keyword before class name.

- Cannot creat an instance (object) of abstract class.
- Can have abstraction / non-abstract method.
- Can have constructors.

CODE 17:

```
abstract class Animal {
    void eat() {
        Syso("Animal Eats");
    }
    abstract void walk();
}
class Horse extends Animal {
    void walk() {
        Syso("Walks on 4 Legs");
    }
}
class Chicken extends Animal {
    void walk() {
        Syso("Walks on 2 Legs");
    }
}
```

```
public static void main (String args[]) {
    Horse h = new Horse ();
    h.eat();
    h.walk();
    Chicken c = new Chicken ();
    c.eat();
    c.walk();
}
```

2. Interfaces

Blueprint (interface) → Blueprint (class) → M800 (object)

Interface is a blueprint of a class.

Ex: Car [wheels, speed, engine] → interface

↓

Maruti 800 → class

↓

car1	car2	car3	carn
M800	M800	M800	M800

→ object

- Multiple Inheritance is not supported in JAVA, so for implementing this we use Interfaces.
- In Interfaces, Abstraction is 100%, therefore for total Abstraction we use Interfaces.
- We define interface with Interface Keyword.
- To ~~interface~~ inherit the interface we use implements.
- All methods are public, abstract & without implementation.
- Variables in the interface are final, public & static.

CODE 17:

```
interface ChessPlayer {
    void moves();
}
class Queen implements ChessPlayer {
    public void moves() {
        Syso("up, down, left, right, diagonal (in all 4 dirns)");
    }
}
class Rook implements ChessPlayer {
    public void moves() {
        Syso("up, down, left, right);
    }
}
class King implements ChessPlayer {
    public void moves() {
        Syso("up, down, left, right, diagonal (by 1 step)");
    }
}
public static void main(String args[]) {
    Queen q = new Queen();
    q.moves();
}
```

* Static Keyword

In Java, it is used to share the same variable or method of a given class.

- एक Program या class में इन चीजों को हम static बना सकते हैं :-
  - Properties (int, Float, char, etc.)
  - Functions
  - Blocks of Codes
  - Nested Classes (class के अंदर class)

Static object memory के अंदर सिर्फ एक बार ही बनती है। If you a change one object then all other obj will also changed.

CODE 18:

```
public class OOPs {
    public static void main (String args[]) {
        Student s1 = new Student();
        s1.schoolName = "RSMT";
        Student s2 = new Student();
        Syso (s2.schoolName);
        Student s3 = new Student();
        s3.schoolName = "ABCD";
    }
}
class Student {
    String name;
    int roll;
    static String schoolName;
    void SetName (String name) {
        this.name = name;
    }
    String getName() {
        return this.name;
    }
}
```

* Super Keyword

Super keyword is used to refer immediate parent class object.

- to access parent's properties.
- to access parent's functions.
- to access parent's constructor.

CODE 19:
```
public class OOPs {
    public static void main (String args[]) {
        Horse h = new Horse();
        Syso (h.color);
    }
}
class Animal {
    String color;
    Animal () {
        Syso ("Animal constructor is called");
    }
}
class Horse extends Animal {
    Horse () {
        super.color = "brown";
        Syso ("Horse constructor is called");
    }
}
```

* Constructor Chaining

It is the process of calling one constructor from another constructor with respect to current object.

- It is used to avoid duplicate codes ~~with~~ while having multiple constructor & make code relatable.
- Constructor chaining can be done in two ways:

  i) within same class ⟶ this()
  ii) from base class ⟶ super()

//within same class using this() Keyword

CODE20:
```
class Temp {
  Temp() {
    this(5);
    Syso("The Default Constructor");
  }
  Temp(int x) {
    this(5, 15);
    Syso(x);
  }
  Temp(int x, int y) {
    Syso(x * y);
  }
  public static void main(String args[]) {
    new Temp();
  }
}
```

Order in Constructor chaining is not important

// other class using super() keyword

CODE21:
```
class Base {
  String name;
  Base() {
    this("");
    Syso("No-argument constructor of" +
        "base class");
  }
  Base(String name) {
    this.name = name;
    Syso("Calling parameterised constructor" + "of Base");
  }
}
```

```
class Derived extends Base {
  Derived() {
    Syso ("No-argument constructor" + "of derived");
  }
  Derived (String name) {
     super (name);
     Syso ("Calling parameterized " + "constructor
           of derived");
  }
  public static void main (String args[]) {
     Derived obj = new Derived ("test");
  }
}
```

Super() should be the first line of the constructor as super class's constructor are invoked before the sub class's constructor.

PRACTICE QUESTIONS

Ques 1. Find out correct option to assign name to object. (s)

a) s -> = "Anubhav";
b) student.name = "Anubhav";
c) s.name = "Anubhav"; ⇒ objname = "value"
↳ dot operator

Ques 2. Which variable can the class Person access in code?

a) name
b) weight
c) rollNumber
d) schoolName

Person
↓
Student

```
class Person {
    String name;
    int weight;
}
class Student extends Person{
    int rollNumber;
    String schoolName;
}
```

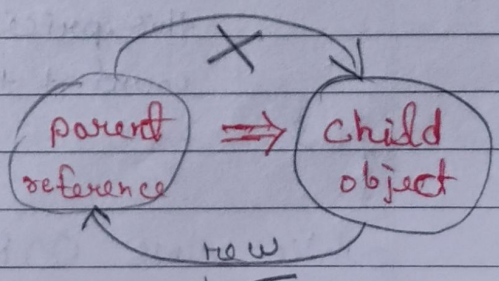
Ques 3. Which of the following modifiers are not allowed in front of class?
a) private
b) protected
c) public
d) default

⇒ किसी भी class को private बनाने के बाद वो class Unusable हो जाएगी।

Ques 4. Which of the following is a correct statement? (both classes in same package).

→ class Vehicle()
class Car extends Vehicle()

a) Car c = new Car();
b) Vehicle v = new Vehicle();
c) Vehicle v = new Car();
d) Car c = new Vehicle();



Ques 5. What will be the output of this code?

```
public class Inheritance {
    public static void main (String args[]) {
        Vehicle obj1 = new Car();
        obj1.print();
        Vehicle obj2 = new Vehicle();
        obj2.print();
    }
}
class Vehicle {
    void print() {
        System.out.println("Base class (Vehicle)");
    }
}
class Car extends Vehicle {
    void print() {
        System.out.println("Derived class (Car)");
    }
}
```

Output : Derived class (Car)
Base class (Vehicle)

Ques 6. What will be the output of this code?

```
class Book {
    int price;          → initialize 0
    static int count;
    public Book (int price) {
        this.price = price;
        count++;
    }
}
public class OOPs {
```

```
public static void main (String [] args) {
    Syso (Book.count + " ");  ⇒ 0
    Book b1 = new Book (150);  → count = 1
    Book b2 = new Book (250);  → count = 2
    Syso (Book.count);  ⇒ count = 2
  }
}
```

Output: 0 2.

Ques 7. Which line has error?

```
class Test {
  static int marks;
  void set_marks (int marks) {
    this.marks = marks;        // Line 1
  }
}
public class OOPs {
  public static void main (String[] args) {
    Test t = new Test();
    t.set_marks (98);          // Line 2
    System.out.print (Test.marks);   // Line 3
  }
}
```

NO ERROR 98

Output:

Ques 8. What would be the output of the following code?

```
class Test {
  static int a = 10;
  static int b;
  static void changeB() {
    b = a * 3;
  }
}
```

```
public class OOPS {
    public static void main (String [] args) {
        Test t = new Test();
        t.changeB();
        Syso (Test.a + Test.b);
    }
}
```

a = 10
b = a * 3 = 10 * 3 = 30
Syso (10 + 30)

Output: 40

Ques 9. Look at following & choose right option.

```
public class Shape {
    protected void display () {
        Syso ("Display - base");
    }
}
public class Circle extends Shape { <
    < access - modifier > void display () {
        Syso ("Display - derived");
    }
}
```

a) Only protected can used.
~~b) public & protected " "~~
c) public, protected & private can used.
d) only public can be used.

* We can provide only a less restrictive or same-access modifier when overriding a method.

Ques 10. Print sum, difference & product of two complex numbers by creating a class named 'Complex' with separate methods for each operation whose real & imaginary parts are entered by user.


```
import java.util.*;
class Complex {
    int real;
    int imag;
    public Complex (int r, int i) {
        real = r;
        imag = i;
    }
    public static Complex add (Complex a, Complex b) {
        return new Complex ((a.real + b.real), (a.imag + b.imag));
    }
    public static Complex diff (Complex a, Complex b) {
        return new Complex ((a.real - b.real), (a.imag - b.imag));
    }
    public static Complex product (Complex a, Complex b) {
        return new Complex (((a.real * b.real) - (a.imag * b.imag)),
                            ((a.real * b.imag) + (a.imag * b.real)));
    }
    public void printComplex () {
        if (real == 0 && imag != 0) {
            Syso (imag + "i");
        }
        else if (imag == 0 && real != 0) {
            Syso (real);
        }
        else {
            Syso (real + "+" + imag + "i");
        }
    }
}
```

```
class Solution {
  public static void main (String[] args) {
    Scanner sc = new Scanner (System.in);
    Complex c = new Complex (4,5);
    Complex d = new Complex (3,4);
    Complex e = ~~new~~ Complex.add (c,d);
    Complex f = Complex.diff (c,d);
    Complex g = Complex.product (c,d);
    e.printComplex();
    f.printComplex();
    g.printComplex();
  }
}
```

Ques 11. What is the output of the program?

```
class Automobile {
  private String drive() {
    return "Driving vehicle";
  }
}
class Car extends Automobile {
  protected String drive() {
    return "Driving car";
  }
}
public class ElectricCar extends Car {
  @Override
  public final String drive() {
    return "Driving electric car";
  }
  psvm (String[] wheels) {
```

```
        final Car car = new ElectricCar();
        Syso (car.drive());
    }
}
```

Output: Driving electric Car.

Ques 12. What is Output of the program?

```
abstract class Car {
    static {
        Syso ("1");
    }
    public Car (String name) {
        super();
        Syso ("2");
    }
    {
        Syso ("3");
    }
}

public class BlueCar extends Car {
    {
        Syso ("4");
    }
    public BlueCar () {
        super ("blue");
        Syso ("5");
    }
    public static void main (String[] ~~args~~ gears) {
        new BlueCar();
    }
}
```

Output: 13245

* Exception Handling

- Exception is an abnormal condition.
- Exception is an event that disrupts the normal flow of the diagram.
- It is an object which is thrown at runtime.
- Exception Handling is a mechanism to handle runtime errors.

Explain → Suppose there are 10 statements in a program & an exception occurs at statement 5; the rest of the code will not be executed i.e., statements 6 to 10 will not be executed. However, when we perform exception handling, the rest of the statements will be executed.

- There are mainly two types of exceptions:
  - i) Checked
  - ii) Unchecked
- An Error is considered as the unchecked.
- According to Oracle, there are three types:

i) Checked Exception – The classes that directly inherit the Throwable class except Runtime Exception & Error.

ii) Unchecked Exception – The classes that inherit the Runtime Exception.

iii) Error – Error is irrecoverable.

* try & catch & finally

→ try statement allows you to define a block of code to be tested for errors while it is being executed.

→ catch statement allows you to define a block of code to be executed, if an errors occur in try block

→ finally statement lets you execute code, after try-catch regardless of result.

CODE:
```
public static void main (String args[]) {
    try {
        int [] myNumbers = {1, 2, 3};
        Syso (myNumbers [10]);
    } catch (Exception e) {
        Syso ("Something went Wrong.");
    } finally {
        Syso ("The "try-catch" is finished.");
    }
}
```

* throw - allows us to create a custom error.

CODE:
```
public class throw {
    static void checkAge (int age) {
        if (age < 18) {
            throw new ArithmeticException ("Access denied - You must
            be least 18 year old.");
        } else {
            Syso ("Access denied - You are old enough");
        }
    }
    psvm {
        checkAge (15);
    }
}
```