OOPs

1. Abstraction

```java
abstract class Animal{
    void eat(){
        System.out.println("Animal Eats.");
    }
    abstract void walk();
}
class Horse extends Animal{
    void walk(){
        System.out.println("Walks on 4 Legs.");
    }
}
class Chicken extends Animal{
    void walk(){
        System.out.println("Walk on 2 Legs.");
    }
}
public class Abstraction {
    public static void main(String[] args) {
        Horse h = new Horse();
        h.eat();
        h.walk();
        Chicken c = new Chicken();
        c.eat();
        c.walk();
    }
}
```

2. Access Modifiers

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

public class AccessModifiers {
    public static void main(String[] args) {
        BankAccount myAcc = new BankAccount();
        System.out.println(myAcc.username = "Anubhav Singh");

        // myAcc.password = "abcde";   //Error becoz of private
        myAcc.setPassword("abcdefgh");
        // Unfortunately, you cannot directly print the password as it is private.
        // You would need a public method in the BankAccount class to access it.
    }
}
```

3. Complex Number

```
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) {
            System.out.println(imag + "i");
        } else if(imag == 0 && real != 0){
            System.out.println(real);
        } else {
            System.out.println(real + " + " + imag + "i");
        }
    }
}
public class CalculateComplexNums {
    public static void main(String[] args) {
        Complex c = new Complex(4, 5);
        Complex d = new Complex(9, 4);
        Complex e = Complex.add(c, d);
        Complex f = Complex.diff(c, d);
        Complex g = Complex.product(c, d);
        e.printComplex();
        f.printComplex();
        g.printComplex();
    }
}
```

4. Class – Object

```
class Pen{
    String color;
    int tip;
    void setColor(String newColor){
        color = newColor;
    }
    void setTip(int newTip){
        tip = newTip;
    }
}
public class ClassObject {
    public static void main(String[] args) {
        Pen p1 = new Pen(); //creating 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);
    }
}
```

5. Constructor

```
class Student{
    String name;
    int roll;
    Student(){
        System.out.println("Constructor is Called...");
    }
}
public class Constructor {
    // Constructor is a special method which is invoked automatically at time of object creation.
    public static void main(String[] args) {
        Student s1 = new Student();
    }
}
```

6. Constructor Chaining Ist

```
// within same class using this() keyword

class Temp{
    Temp(){
        this(5);
        System.out.println("The Default Constructor");
    }
    Temp(int x){
        this(5, 15);
        System.out.println(x);
    }
    Temp(int x, int y){
        System.out.println(x * y);
    }
}
public class ConstructorChaining {
    public static void main(String[] args) {
        new Temp();
    }
}
```

7. Constructor Chaining IInd

```
class Base{
    String name;
    Base(){
        this(" ");
        System.out.println("No Argument Constructor of Base Class.");
    }
```

```
    Base(String name){
        this.name = name;
        System.out.println("Calling Parameterised Constructor of Base.");
    }
}
class Derived extends Base{
    Derived(){
        System.out.println("No Argument Constructor of Derived.");
    }
    Derived(String name){
        System.out.println(name);
        System.out.println("Calling Parameterised Constructor of Derived.");
    }
}
public class ConstructorChainingSec {
    public static void main(String[] args) {
        Derived obj = new Derived("test");
    }
}
```

8. Copy Constructor

```
class Fruit{
    private double fprice;
    private String fname;
    Fruit(double fprice, String fname){
        this.fprice = fprice;
        this.fname = fname;
    }
    Fruit(Fruit fruit){
        System.out.println("\n After invoking the Copy Constructor.");
        fprice = fruit.fprice;
        fname = fruit.fname;
    }
    double showPrice(){
        return fprice;
    }
    String showName(){
        return fname;
    }
}
public class CopyConstructor {
    public static void main(String[] args) {
        Fruit f1 = new Fruit(399, "Anubhav");
        System.out.println("Name of the First Fruit: " + f1.showName());
        System.out.println("Price of the First Fruit: " + f1.showPrice());
        // Copy Constructor
        Fruit f2 = new Fruit(f1);
        System.out.println("Name of the Second Fruit: " + f2.showName());
        System.out.println("Price of the Second Fruit: " + f2.showPrice());
    }
}
```

9. Deep Copy

```java
class Address {
    String city;

    // Constructor to initialize city
    public Address(String city) {
        this.city = city;
    }

    // Copy constructor for deep copy
    public Address(Address address) {
        this.city = address.city;
    }
}

class Person {
    String name;
    Address address;

    // Constructor to initialize name and address
    public Person(String name, Address address) {
        this.name = name;
        this.address = new Address(address); // Deep copy of Address
    }

    // Copy constructor for deep copy
    public Person(Person person) {
        if (person != null) {
            this.name = person.name;
            this.address = new Address(person.address); // Deep copy of address object
        }
    }
}

public class DeepCopy {
    public static void main(String[] args) {
        Address address1 = new Address("New York");
        Person person1 = new Person("John", address1);

        // Creating a deep copy of person1
        Person person2 = new Person(person1);

        // Modifying the address of person2
        person2.address.city = "Los Angeles";

        // Printing the addresses to verify deep copy
        System.out.println("Person 1 Address: " + person1.address.city); // Should print "New
York"
        System.out.println("Person 2 Address: " + person2.address.city); // Should print "Los
Angeles"
    }
}
```

10. Getters and Setters

```
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 GettersSetters {
    public static void main(String[] args) {
        Pen p1 = new Pen();
        p1.setColor("Blue");
        System.out.println(p1.getColor());
        p1.setTip(5);
        System.out.println(p1.getTip());
    }
}
```

11. Hierarchical Inheritance

```
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...");
    }
}
public class HierarchicalInheritance {
    public static void main(String[] args) {
        Cat c = new Cat();
        c.meow();
        c.eat();
    }
}
```

12. Hybrid Inheritance

```
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 HybridInheritance {
    public static void main(String[] args) {
        Dog PitBull = new Dog();
        PitBull.breathe();
        PitBull.eat();
        PitBull.play();
    }
}
```

13. Inheritance

```
//Base Class
class Animal{
    String color;
    void eat(){
        System.out.println("Eats");
    }
    void breathe(){
        System.out.println("Breathes");
    }
}
//Derived Class or Sub-class
class Fish extends Animal{
    int fins;
    void swim(){
        System.out.println("Swims in Water.");
    }
}
public class Inheritance {
    public static void main(String[] args) {
        Fish shark = new Fish();
        shark.eat();
        shark.breathe();
    }
}
```

14. Interfaces

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

15. Multi-Level Inheritance

```
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 class MultiLevelInheritance {
    public static void main(String[] args) {
        Z obj = new Z();
        obj.methodX();
        obj.methodY();
        obj.methodZ();
    }
}
```

16. Polymorphism

```java
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;
    }
}
public class Polymorphism {
    public static void main(String[] args) {
        Calculator cal = new Calculator();
        System.out.println(cal.sum(5, 8));
        System.out.println(cal.sum(55, 95));
        System.out.println(cal.sum(5, 5, 5));
    }
}
```

17. Quiz Output

```java
abstract class Car{
    static{
        System.out.println("1");
    }
    public Car(String name){
        super();
        System.out.println("2");
    }
    {
        System.out.println("3");
    }
}
class Bluecar extends Car{
    {
        System.out.println("4");
    }
    public Bluecar(){
        super("blue");
        System.out.println("5");
    }
}
public class QuizOutput {
    public static void main(String[] args) {
        new Bluecar();
    }
}
```

18. Run Time Polymorphism

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

19. Shallow Copy

```java
class Person {
    String name;
    String address;
    // Shallow copy constructor
    public Person(Person person) {
        if (person != null) {
            this.name = person.name;
            this.address = person.address;
            // Shallow copy of address object
        }
    }
}
public class ShallowCopy {
    public static void main(String[] args) {
        Person person1 = new Person(null);
        person1.name = "John";
        person1.address = "123 Main St";

        // Creating a shallow copy of person1
        Person person2 = new Person(person1);

        // Displaying the details of person1 and person2
        System.out.println("Person 1 Name: " + person1.name);
        System.out.println("Person 1 Address: " + person1.address);
        System.out.println("Person 2 Name: " + person2.name);
        System.out.println("Person 2 Address: " + person2.address);

        // Modifying the address of person2
        person2.address = "456 Elm St";

        // Displaying the details again to show the effect of shallow copy
        System.out.println("After modifying person2's address:");
        System.out.println("Person 1 Address: " + person1.address);
        System.out.println("Person 2 Address: " + person2.address);
    }
}
```

20. Single Inheritance

```
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 class SingleLevelInheritance {
    public static void main(String[] args) {
        B obj = new B();
        obj.methodA();
        obj.methodB();
    }
}
```

21. Static Keyword

```
class Student{
    String name;
    int roll;
    static String schoolName;
    void setName(String name){
        this.name = name;
    }
    String getName(){
        return this.name;
    }
}
public class StaticKeyword {
    public static void main(String[] args) {
        Student s1 = new Student();
        s1.schoolName = "RSMT";
        Student s2 = new Student();
        System.out.println(s2.schoolName);
        Student s3 = new Student();
        s3.schoolName = "GDES";
    }
}
```

22. Super Keyword

```
class Animal{
    String color;
    Animal(){
        System.out.println("Animal Constructor is Called...");
    }
}
class Horse extends Animal{
    Horse(){
        super.color = "Brown";
        System.out.println("Horse Constructor is Called...");
```

```
    }
}
public class SuperKeyword {
    public static void main(String[] args) {
        Horse h = new Horse();
        System.out.println(h.color);
    }
}
```

23. Types of Constructors

```
class Student{
    String name;
    int roll;
    Student() {     //Non-parameterized
        System.out.println("Constructor is Called...");
    }
    Student(String name){   //Parameterized
        this.name = name;
    }
    Student(int roll){      //Parameterized
        this.roll = roll;
    }
}
public class TypesOfConstructor {
    public static void main(String[] args) {
        Student s1 = new Student();
        Student s2 = new Student("Anubhav");
        System.out.println("Student name: " + s2.name);
        Student s3 = new Student(210801);
        System.out.println("Student roll: " + s3.roll);
    }
}
```