JAVA Assignment Programs

1.Encapsulation + Getter/Setter

WAP in Java

Create a class named Employee with private instance variables empId, empName, and salary. Provide public getters and setters for all variables. Create a method displayDetails() to print employee details. Create an object in the main method and assign values using setters then display them.

Solution:

```
package JavaProgramAssignment;
class Employee {
  private int empId;
  private String empName;
  private double salary;
  public int getEmpId() {
     return empId;
  }
  public void setEmpId(int empId) {
     this.empId = empId;
  }
  public String getEmpName() {
     return empName;
  }
  public void setEmpName(String empName) {
     this.empName = empName;
  }
  public double getSalary() {
     return salary;
  }
  public void setSalary(double salary) {
     this.salary = salary;
  }
  public void displayDetails() {
     System.out.println("Employee Details");
     System.out.println("Emp ID: " + empId);
     System.out.println("Emp Name: " + empName);
     System.out.println("Emp Salary: " + salary);
  }
```

```
}
public class Encapsulation1 {
  public static void main(String[] args) {
     Employee emp = new Employee();
     emp.setEmpId(101);
     emp.setEmpName("Samrat");
     emp.setSalary(75000.00);
     emp.displayDetails();
  }
}
```

Output:
Employee Details
Emp ID: 101
Emp Name: Samrat
Emp Salary: 75000.0

2.Inheritance + Method Overriding

Create a base class Vehicle with a method fuelType() which prints "Runs on fuel". Create a child class ElectricCar and override the fuelType() method to print "Runs on electricity". Create objects of both classes and call their respective methods.

Solution:

```
package JavaProgramAssignment;
class Vehicle
{
   void fuelType()
   {
     System.out.println("Runs on fuel");
  }
}
class ElectricCar extends Vehicle
{

   void fuelType()
   {
     System.out.println("Runs on electricity");
  }
}
```

```
public class Inheritance
{
  public static void main(String[] args)
  {
     Vehicle v = new Vehicle();
     v.fuelType();
     System.out.println();
     ElectricCar ec = new ElectricCar();
     ec.fuelType();
  }
}
```

Output:
Runs on fuel
Runs on electricity

3.Constructor Overloading

Create a class Product having instance variables productId, productName, and price. Implement: A default constructor that prints "Product Created". A parameterized constructor that initializes the product details. Write a method displayProduct() to print product details. Create both types of objects in the main method.

Solution:

```
package JavaProgramAssignment;
class Product
{
  int productId;
  String productName;
  double price;
  Product()
  {
     System.out.println("Product  Created");
  }
  Product(int productId, String productName, double price)
  {
     this.productId = productId;
     this.productName = productName;
     this.price = price;
  }
```

```
  public void displayProduct()
  {
     System.out.println("Product ID: " + productId);
     System.out.println("Name:       " + productName);
     System.out.println("Price:      " + price);
  }
}
public class ConstructorOverloading3
{
  public static void main(String[] args)
  {

     Product p1 = new Product();
     p1.displayProduct();
     System.out.println();
     Product p3 = new Product(102, "Samrat", 499.99);
     p3.displayProduct();
  }
}
```

Output:

```
Product  Created
Product ID: 0
Name:       null
Price:      0.0
Product ID: 102
Name:       Samrat
Price:      499.99
```

4. Method Overloading

Create a class Calculator with overloaded methods add(): add(int a, int b) add(double a, double b) Call both methods inside the main method and print results.

Solution:

```
package JavaProgramAssignment;
class Calculator {

  int add(int a, int b) {
     return a + b;
```

```
  }

  double add(double a, double b) {
    return a + b;
  }
}
public class MehodOverloadingQ4 {

  public static void main(String[] args) {

    Calculator c1 = new Calculator();
    System.out.println(c1.add(5, 5));
    System.out.println(c1.add(5.5, 4.5));
  }
}
```

Output:

10
9.7

5.Abstraction using Abstract Class

Create an abstract class Animal with an abstract method sound(). Create two subclasses Dog and Cat and provide implementation for sound() method. Create objects and call sound() for each.

Solution:

```
package JavaProgramAssignment;
abstract class Animal
{
        abstract void sound();
}
class Dog extends Animal
{
        void sound()
        {
                System.out.println("Barking");
        }
}
class Cat extends Animal
{
        void sound()
```

```
	{
		System.out.println("Meow");
	}
}
public class AbstractionClassQ5 {

	public static void main(String[] args)
	{
		Dog g = new Dog();
		g.sound();
		Cat c = new Cat();
		c.sound();

	}
}
```

Output:

Barking
Meow

6.Interface Implementation

Create an interface Payment with a method makePayment(). Create two classes UPI and CreditCard implementing this interface and define their own payment messages. Call the method through interface reference.

Solution:

```
package JavaProgramAssignment;
interface Payment
{
	void makePayment();
}
class UPI implements Payment
{
	public void makePayment()
	{
		System.out.println("Payment done by UPI");
	}
}
class CreditCard implements Payment
{
```

```
	public void makePayment()
	{
		System.out.println("Payment done by Credit Card");
	}
}
public class InterfaceQ6
{
	public static void main(String[] args)
	{
		Payment p = new UPI();
		p.makePayment();

		Payment p2 = new CreditCard();
		p2.makePayment();
	}
}
```

Output:

Payment done by UPI
Payment done by Credit Card

7.Use of super keyword

Create a class Person with a constructor that prints "Person Created". Create a subclass Student that calls the parent constructor using super() and prints "Student Created". Create an object and observe the order of execution.

Solution:

```
package JavaProgramAssignment;
class Person
{
	Person()
	{
		System.out.println("Person Created");
	}
}
class Student extends Person
{
	Student()
	{
		super();
```

```
            System.out.println("Student Created");
        }
}
public class UseOfSuperKeywordQ7
{
        public static void main(String[] args)
        {
            new Student();
        }
}
```

Output:

Person Created
Student Created

8.Polymorphism (Dynamic Binding)

Create a parent class Shape with a method area(). Create subclasses Rectangle and Circle and override the area() method. Create a reference of Shape and assign objects of both subclasses one by one, calling area() each time.

Solution:

```
package JavaProgramAssignment;
class Shape
{
        double area(double x, double y)
        {
            return 0;
        }
}
class Rectangle extends Shape
{

        double area(double x, double y)
        {
            return x * y;
        }
}
class Circle extends Shape
{
        double pi = 3.14;
```

```
        double area(double x, double y)
        {
                return pi*x*y;
        }
}
public class PolymorphismQ8
{
        public static void main(String[] args)
        {
                Shape s1;
                s1 = new Rectangle();
                System.out.println("Are of Rectangle: "  + s1.area(10.0, 5.0));

                s1 = new Circle();
                System.out.println("Are of Circle: "  + s1.area(5.0, 5.0));
        }
}
```

Output:
Are of Rectangle: 50.0
Are of Circle: 78.5

9.Final Keyword

Create a class Bank with a final variable IFSC and final method showIFSC().
Try creating a subclass HDFCBank and attempt overriding the final method (should show compile-time restriction). Create a main method to demonstrate usage.

Solution:

```
package JavaProgramAssignment;
class Bank
{

  final String IFSC = "BANK0001234";

  final void showIFSC()
  {
     System.out.println("The IFSC Code is: " + IFSC);
  }
}
```

```
class HDFCBank extends Bank
{

  // Attempting to override the final method
  //  If uncomment the lines below, the program will NOT compile.

 //    void showIFSC() {
//        System.out.println("Trying to change implementation...");
//    }


   public void display()
   {
      System.out.println("This is HDFC Bank.");
   }
}
public class FinalKeyword_Q9
{
public static void main(String[] args)
{
    HDFCBank b = new HDFCBank();
     b.showIFSC();
     b.display();
   }
}
```

Output:
The IFSC Code is: BANK0001234
This is HDFC Bank.

10.Static Keyword
Create a class Student having static variable collegeName and instance variables name and rollNo. Write a method to print both static and instance data. Create multiple objects to show static value remains constant.

Solution:

```
package JavaProgramAssignment;
class Students
{
        static String collegeName = "ABC";
```

```
        String name;
        int rollNo;

        void display(String n, int r)
        {
                name = n;
                rollNo = r;

                 System.out.println("\nCollege Name: " + collegeName +
            "\nName of the Student: " + name +
            "\nRoll No: " + rollNo);
        }
}
public class StaticKeyword_Q10 {
        public static void main(String[] args)
        {
                Students s = new Students();
                s.display("Samu", 1011);
                s.display("Ammu", 1012);
                s.display("Tarun", 1013);
                s.display("Virat", 1014);
        }
}
```

Output:

College Name: ABC
Name of the Student: Samu
Roll No: 1011
College Name: ABC
Name of the Student: Ammu
Roll No: 1012
College Name: ABC
Name of the Student: Tarun
Roll No: 1013
College Name: ABC
Name of the Student: Virat
Roll No: 1014

11.Class + Object + Method

Create a class Library with an instance variable libraryName.

Create a default constructor to print "Welcome to the Library!".
Create a method showLocation() which prints "This library is located in Mumbai".
Create an object in main() and call both.

Solution:

```
package JavaProgramAssignment;
class Library
{
	String libraryName = "City Central Library";

	Library()
	{
		System.out.println("Welcome to the Library");
	}

	void showLocation()
	{
		System.out.println("This library is located in Mumbai");
	}
}
public class ClassObjectMethod_Q11 {
	public static void main(String[] args) {
		Library l = new Library();
		l.showLocation();
		System.out.println("The Name of the Library is: " +l.libraryName);
	}
}
```

Output:

Welcome to the Library
This library is located in Mumbai
The Name of the Library is: City Central Library

12.Encapsulation + Validation Logic

Create a class Account with private variables accountHolderName and balance. Provide setters and getters, where: setBalance() should not accept negative values (print a warning). Create an object and update values through setters only.

Solution:

```
package JavaProgramAssignment;
```

```
class Account
{
	private String accountHolderName;
	private double balance;

	public String getaccountHolderName()
	{
		return accountHolderName;
	}

	public void setaccountHolderName(String accountHolderName)
	{
		this.accountHolderName = accountHolderName;
	}

	public double getBalance()
	{
		return balance;
	}

	public void setBalance(double balance)
	{
		if (balance < 0)
		{
			System.out.println("Warning: Balance cannot be negative!
Value rejected.");
		}
		else
		{
			this.balance = balance;
			System.out.println("Name :" +accountHolderName);
			System.out.println("Balance :" +balance);
		}
	}
}
public class EncapsulationValidation_12 {

	public static void main(String[] args)
	{
		Account a1 = new Account();
```

```
                a1.getaccountHolderName();
                a1.setaccountHolderName("Tarun");
                a1.getBalance();
                a1.setBalance(25000);

                //Set -ve value
                Account a2 = new Account();
                a2.getaccountHolderName();
                a2.setaccountHolderName("Manu");
                a2.getBalance();
                a2.setBalance(-5);
        }
}
```

Output:

Name :Tarun
Balance :25000.0

Warning: Balance cannot be negative! Value rejected.

13.Inheritance (Multilevel)

Create three classes:
Device â†’ method start()
Mobile extends Device â†’ method calling()
SmartPhone extends Mobile â†’ method internet()
Create object of SmartPhone and call all methods.

Solution:

```
package JavaProgramAssignment;
class Device
{
        void start()
        {
                System.out.println("This is from Device class");
        }
}
class Mobile extends Device
{
        void calling()
        {
```

```
                System.out.println("This is from Mobile class");
        }
}
class SmartPhone extends Mobile
{
        void internet()
        {
                System.out.println("This is from SmartPhone class");
        }
}
public class MultilevelInheritance_13
{
        public static void main(String[] args)
        {
                SmartPhone sp = new SmartPhone();
                sp.start();
                sp.calling();
                sp.internet();
        }
}
```

Output:

This is from Device class
This is from Mobile class
This is from SmartPhone class

14. Hierarchical Inheritance

Create a class Course with a method courseInfo().
Create subclasses Science, Commerce, and Arts each with their own method.
Create objects of each and call methods to show hierarchy.

Solution:

```
package JavaProgramAssignment;
class Course
{
        void courseInfo()
        {
                System.out.println("All courses are registered under University");
        }
}
class Science extends Course
```

```
{
	void scienceSub()
	{
		System.out.println("Science subjects : Pysics, Maths, Biology
Chemestry");
	}
}
class Arts extends Course
{
	void artsSub()
	{
		System.out.println("Arts subjects : Literature, History, Sociology,
Homescience");
	}
}
class Commerce extends Course
{
	void commerceSub()
	{
		System.out.println("Commerce subjects : Economics, Accounts,
Business Studies");
	}
}
public class HirarchicalInheritence_Q14 {
	public static void main(String[] args)
	{
		Science sc = new Science();
		sc.courseInfo();
		sc.scienceSub();
		System.out.println();

		Commerce cm = new Commerce();
		cm.courseInfo();
		cm.commerceSub();

		System.out.println();
		Arts ar = new Arts();
		ar.courseInfo();
		ar.artsSub();
	}}
```

Output:
All courses are registered under University
Science subjects : Pysics, Maths, Biology Chemestry

All courses are registered under University
Commerce subjects : Economics, Accounts, Business Studies

All courses are registered under University
Arts subjects : Literature, History, Sociology, Homescience

15.Method Overloading (Bank Scenario)
Create a class LoanCalculator with two overloaded methods: calculateLoan(int amount) calculateLoan(int amount, double interestRate) Print loan details accordingly. Call both methods from main.
Solution:

```
package JavaProgramAssignment;
class LoanCalculator
{
	double rate = 10;
	double interest;

	void calculateLoan(int amount)
	{
		interest = rate * amount /100;
		System.out.println("Principle Amount: " +amount);
		System.out.println("Default Interest: " +rate+ "%");
		System.out.println("Calculated Interest: "+interest);
	}

	void calculateLoan(int amount, double interestRate)
	{
		interest = interestRate * amount / 100;
		System.out.println("Principle Amount: "+amount);
		System.out.println("Rate of interest: "+interestRate+ "%");
		System.out.println("Calculated Interest:"+interest);
	}
}
public class MethodOverloading_Q15 {

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

```
	{
		LoanCalculator lc = new LoanCalculator();
		lc.calculateLoan(10000);
		System.out.println();
		lc.calculateLoan(50000, 7.5);
	}
}
```

Output:

Principle Amount: 10000
Default Interest: 10.0%
Calculated Interest: $1000.0

Principle Amount: 50000
Rate of interest: 7.5%
Calculated Interest: $3750.0

16. Method Overriding with super

Create a base class Hospital with a method emergencyService(). Create a subclass CityHospital that overrides the method and calls parent method using super.emergencyService(). Demonstrate overriding in main.

Solution:

```
package JavaProgramAssignment;
class Hospital
{
	void emergencyService()
	{
		System.out.println("Hospital Emergency Services:");
		System.out.println("1. 24x7 Emergency Ward");
		System.out.println("2. Ambulance Service");
	}
}
class CityHospital extends Hospital
{
	void emergencyService()
	{
		super.emergencyService();
		System.out.println("City Hospital Emergency Services:");
		System.out.println("1. ICU & Critical Care");
	   System.out.println("2. Trauma Care Unit");
	   System.out.println("3. Emergency Helpline: 108");
```

```
	}
}
public class MethodOverridingWithSuper_Q16 {

	public static void main(String[] args)
	{
		CityHospital ch = new CityHospital();
		ch.emergencyService();
	}
}
```

Output:

Hospital Emergency Services:
1. 24x7 Emergency Ward
2. Ambulance Service
City Hospital Emergency Services:
1. ICU & Critical Care
2. Trauma Care Unit
3. Emergency Helpline: 108

16.Abstract Class + Real Usage

Create an abstract class Employee with: abstract method: calculateSalary() concrete method: employeeDetails() Subclass FullTimeEmployee and PartTimeEmployee implementing salary calculation logic differently.

Solution:

```
package JavaProgramAssignment;
abstract class EmployeeType //Employee class is already created in previous program
{
	String empName;
	int empID;


	abstract void calculateSalary();

	EmployeeType(String n, int id)
	{
		this.empName = n;
		this.empID = id;
	}
```

```
	void employeeDetails()
	{
		System.out.println("Name :"+empName+ "\nID: "+empID);
	}
}
class FullTimeEmployee extends EmployeeType
{
	double monthSalary;

	FullTimeEmployee(String name, int id, double salary)
	{
		super(name,id);
		this.monthSalary = salary;
	}

	void calculateSalary()
	{
		System.out.println("Employee Type : Full Time");
		System.out.println("Salara : Monthly Fixed Pay");
		System.out.println("Total Salary : $" +monthSalary);
	}
}
class PartTimeEmployee extends EmployeeType
{
	double hourRate;
	double workedHour;
	double total;

	PartTimeEmployee(String name, int id, double hourRate, double
workedHour)
	{
		super(name,id);
		this.hourRate = hourRate;
		this.workedHour = workedHour;
	}
	void calculateSalary()
	{
		total = hourRate * workedHour;
```

```
                System.out.println("Employee Type : Part Time");
                System.out.println("Salary : Hourly base");
                System.out.println("Total Salary :" +total);
        }
}
public class AbstractClassRealUse_16 {

        public static void main(String[] args)
        {
                FullTimeEmployee ft = new FullTimeEmployee("Tarun",101,75000);
                ft.employeeDetails();
                ft.calculateSalary();
                System.out.println();
                PartTimeEmployee pt = new
PartTimeEmployee("Virat",102,2000,250);
                pt.employeeDetails();
                pt.calculateSalary();
        }
}
```

Output:

Name :Tarun
ID: 101
Employee Type : Full Time
Salara : Monthly Fixed Pay
Total Salary : $75000.0

Name :Virat
ID: 102
Employee Type : Part Time
Salary : Hourly base
Total Salary :500000.0

18. Interface with Multiple Implementations

Create an interface Transport with method booking(). Implement it in Bus and Flight classes. Call using interface reference.

Solution:

```
package JavaProgramAssignment;
interface Transport
{
        void booking();
```

```
}
class Bus implements Transport
{
	public void booking()
	{
		System.out.println("Bus Booking Details");
		System.out.println("Seat Reserved");
		System.out.println("Boarding Point: SBI Bank");
		System.out.println("Ticket Type : A/C Sleeper");
	}
}
class Flight implements Transport
{
	public void booking()
	{
		System.out.println("Flight Booking Details");
		System.out.println("Seat Allocated");
		System.out.println("Boarding Gate: E12");
		System.out.println("Class : Economy");
	}
}
public class Interface_Q18 {

	public static void main(String[] args)
	{
		Transport ts;
		ts = new Bus();
		ts.booking();
		System.out.println();
		ts = new Flight();
		ts.booking();
	}
}
```

Output:

Bus Booking Details
Seat Reserved
Boarding Point: SBI Bank
Ticket Type : A/C Sleeper
Flight Booking Details

Seat Allocated
Boarding Gate: E12
Class : Economy

19.Polymorphism (Runtime + Upcasting)

Create a class Camera with a method capture(). Create a subclass DSLCamera that overrides the method. Use parent reference to call child object method (dynamic polymorphism).

Solution:

```
package JavaProgramAssignment;
class Camera
{
	void capture()
	{
		System.out.println("Camera: Capture a basic photos");
	}
}
class DSLCamera extends Camera
{
	void capture()
	{
		System.out.println("DSLCamera : Capture high resolution photos");
	}
}
public class Polymorphism_Q19 {

	public static void main(String[] args)
	{
		Camera c = new DSLCamera(); //Upcasting
		c.capture();
	}
}
```

Output:

DSLCamera : Capture a high resolution photos

20. Aggregation (Has-A Relationship)

Create a class Engine with a method engineInfo(). Create a class Car that has-a Engine object and uses it to show engine details.
Show aggregation in main method.

```
package JavaProgramAssignment;
class Engine
{
	String name;
	int hp;

	Engine(String n, int h)
	{
		this.name = n;
		this.hp = h;
	}

	void engineInfo()
	{
		System.out.println("Car Name: " +name+ "\nHorse Power :" +hp);
	}
}
class Car {
	String model;
	Engine engineRef; // Aggregation Car HAS-A Engine

	Car(String model, Engine engineRef) {
      this.model = model;
      this.engineRef = engineRef;
    }

	void showCarInfo()
	{
		System.out.println("Car Model :" +model);
		engineRef.engineInfo();
	}
}
public class Aggregation_Q20 {

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

```
            Engine e = new Engine("BMW",450);
            Car c = new Car("XYZ",e);
            c.showCarInfo();
        }
}
```

Output:

Car Model :XYZ
Car Name: BMW
Horse Power :450

21. Static Concepts

Create a class University with: static variable country = "India" instance variable universityName Print values using different objects to show static effect.

Solution:

```
package JavaProgramAssignment;
class University
{
        static String country = "India";
        String universityName;

        University(String name)
        {
                this.universityName = name;
        }

        void display()
        {
                System.out.println("University :" +universityName+   "  Country :"
+country);
        }
}
public class StaticConcepts_Q21 {

        public static void main(String[] args)
        {
                University u1 = new University("KLE");
                u1.display();
                University u2 = new University("IIT Bombay");
                u2.display();
```

```
                University u3 = new University("Dharwad University");
                u3.display();
                System.out.println();
                //Change static value
                University.country = "Bharat";
                u1.display();
                u2.display();
                u3.display();
        }
}
```

Output:

University :KLE Country :India
University :IIT Bombay Country :India
University :Dharwad University Country :India

University :KLE Country :Bharat
University :IIT Bombay Country :Bharat
University :Dharwad University Country :Bharat

22. Final Keyword + Constant

Create a class GovernmentRules with a final variable MAX_WORKING_HOURS = 8
Try modifying it inside main and observe compile-time restriction.

Solution:

```
package JavaProgramAssignment;
class GovernmentRules
{
        final int MAX_WORKING_HOURS = 8;

        void display()
        {
                System.out.println("Standard working hours :"
+MAX_WORKING_HOURS);
        }
}
public class FinalKeyword_Q22 {
        public static void main(String[] args)
        {
                GovernmentRules g = new GovernmentRules();
                g.display();
```

```
		//Try to change final value
		//If uncomment below line will get compile-time restriction
		//g.MAX_WORKING_HOURS = 12;
	}
}
```

Output:

Standard working hours :8

23. Constructor Chaining

Create a class Mall with: Default constructor printing "Welcome to the Mall" Parameterized constructor calling default constructor using this() Demonstrate constructor chaining in main.

Solution:

```
package JavaProgramAssignment;
class Mall
{
	Mall()
	{
		System.out.println("Welcpme to the Mall");
	}

	Mall(String name)
	{
		this(); // Calling Default constructor
		System.out.println("This is a" +name+ " Mall Loacated in
Ahemdabad");
	}
}
public class ConstructorChaining_Q23 {

	public static void main(String[] agrs)
	{
		new Mall("Alpha One");
	}
}
```

Output:

Welcome to the Mall
This is aAlpha One Mall Located in Ahmedabad

24 WAP in Java

Create a Class named Shape with length as instance variable , create three methods as square , rectangle , circle and find out their respective areas Create a object in main method and call these different methods with the instance of object.

Solution:

```
package JavaProgramAssignment;
class Shape24
{
	int length = 10;

	void square()
	{
		int area;
		area = this.length * this.length;
		System.out.println("Area of Square :" +area);
	}
	void rectangle(double w)
	{
		double area;
		area = this.length * w;
		System.out.println("Area of Rectangle :" +area);
	}
	void circle()
	{
		double area;
		area = 3.14 * this.length * this.length;
		System.out.println("Area of Circle : " +area);
	}
}
public class WAP_Q24 {

	public static void main(String[] args)
	{
		Shape24 s = new Shape24();
		s.square();
		s.rectangle(5.0);
		s.circle();
	}
}
```

Output:
Area of Square :100
Area of Rectangle :50.0
Area of Circle : 314.0

25. WAP in Java
Create a class named school ,create name as their instance variables
Create a default constructor of this class which will have a print statement to display the name of school. Create a method inside the class which will display a message as "This School is based out of Kolkata" Create a object under main method and call the constructor and the method

Solution:

```
package JavaProgramAssignment;
class School
{
	String name = "Divine Life";

	School()
	{
		System.out.println("Name of the School : " +name);
	}

	void display()
	{
		System.out.println("This School is based out of Kolkata");
	}
}
public class WAP_25 {
	public static void main(String[] args)
	{
		School s = new School();
		s.display();
	}
}
```

Output:
Name of the School : Divine Life
This School is based out of Kolkata

26. WAP in Java to create a class named school

Create name, address,strength as their instance variables. Create two constructor one with two variables and one with all the three variables. Create a method that will display all the three parameters. Create two object of this class and call the respective methods

Solution:

```
package JavaProgramAssignment;
class School26
{
	String name;
	String address;
	int streanth;

	School26(String n, String a)
	{
		this.name = n;
		this.address = a;
		this.streanth = 0;
	}

	School26(String n, String a, int s)
	{
		this.name = n;
		this.address = a;
		this.streanth = s;
	}

	void display()
	{
		System.out.println("Name of School: " +name);
		System.out.println("Address : " +address);
		System.out.println("Streanth : " +streanth);
	}
}
public class WAP_Q26 {
	public static void main(String[] args)
	{
		School26 s1 = new School26("Divine Life","Ahmedabad");
		s1.display();
		System.out.println();
```

```
            School26 s2 = new School26("Gurukulam","Gandhinagar",1500);
            s2.display();
        }
}
```

Output:

```
Name of School: Divine Life
Address : Ahmedabad
Strength : 0

Name of School: Gurukulam
Address : Gandhinagar
Strength : 1500
```