**1. Creating a Class and Object**

**Program: Define a simple class and create an object**

class Car {

String model;

int year;

void display() {

System.out.println("Model: " + model);

System.out.println("Year: " + year);

}

}

public class Main {

public static void main(String[] args) {

Car myCar = new Car();

myCar.model = "Toyota";

myCar.year = 2020;

myCar.display();

}

}

**2. No-arg Constructor**

**Program: Define a class with a no-arg constructor**

class Dog {

String breed;

int age;

// No-arg constructor

Dog() {

breed = "Labrador";

age = 3;

}

void display() {

System.out.println("Breed: " + breed);

System.out.println("Age: " + age);

}

}

public class Main {

public static void main(String[] args) {

Dog myDog = new Dog();

myDog.display();

}

}

**3. Parameterized Constructor**

**Program: Define a class with a parameterized constructor**

class Book {

String title;

String author;

// Parameterized constructor

Book(String title, String author) {

this.title = title;

this.author = author;

}

void display() {

System.out.println("Title: " + title);

System.out.println("Author: " + author);

}

}

public class Main {

public static void main(String[] args) {

Book myBook = new Book("1984", "George Orwell");

myBook.display();

}

}

**4. Default Constructor**

**Program: Define a class with a default constructor**

class Student {

String name;

int rollNumber;

// Default constructor provided by Java

void display() {

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

System.out.println("Roll Number: " + rollNumber);

}

}

public class Main {

public static void main(String[] args) {

Student student = new Student();

student.name = "John Doe";

student.rollNumber = 101;

student.display();

}

}

**5. Constructor Overloading**

**Program: Define a class with overloaded constructors**

class Rectangle {

int length;

int breadth;

// No-arg constructor

Rectangle() {

length = 5;

breadth = 10;

}

// Parameterized constructor

Rectangle(int length, int breadth) {

this.length = length;

this.breadth = breadth;

}

void display() {

System.out.println("Length: " + length);

System.out.println("Breadth: " + breadth);

}

}

public class Main {

public static void main(String[] args) {

Rectangle rect1 = new Rectangle();

Rectangle rect2 = new Rectangle(7, 14);

rect1.display();

rect2.display();

}

}

**6. Using Constructor in Real-World Scenario**

**Program: Define a class for a real-world scenario**

class Employee {

String name;

int id;

double salary;

// Parameterized constructor

Employee(String name, int id, double salary) {

this.name = name;

this.id = id;

this.salary = salary;

}

void display() {

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

System.out.println("ID: " + id);

System.out.println("Salary: " + salary);

}

}

public class Main {

public static void main(String[] args) {

Employee emp = new Employee("Alice", 1001, 75000);

emp.display();

}

}

**7. Constructor Overloading in Bank Account**

**Program: Define a class with overloaded constructors for a bank account**

class BankAccount {

String accountNumber;

double balance;

// No-arg constructor

BankAccount() {

accountNumber = "000000";

balance = 0.0;

}

// Parameterized constructor

BankAccount(String accountNumber, double balance) {

this.accountNumber = accountNumber;

this.balance = balance;

}

void display() {

System.out.println("Account Number: " + accountNumber);

System.out.println("Balance: " + balance);

}

}

public class Main {

public static void main(String[] args) {

BankAccount acc1 = new BankAccount();

BankAccount acc2 = new BankAccount("123456", 1000.0);

acc1.display();

acc2.display();

}

}

**8. Multiple Objects Creation**

**Program: Create multiple objects of a class**

class Pen {

String color;

String type;

Pen(String color, String type) {

this.color = color;

this.type = type;

}

void display() {

System.out.println("Color: " + color);

System.out.println("Type: " + type);

}

}

public class Main {

public static void main(String[] args) {

Pen pen1 = new Pen("Blue", "Ballpoint");

Pen pen2 = new Pen("Black", "Gel");

pen1.display();

pen2.display();

}

}

**9. Object Initialization Using Constructor**

**Program: Initialize objects using different constructors**

class Laptop {

String brand;

int ram;

// No-arg constructor

Laptop() {

brand = "HP";

ram = 8;

}

// Parameterized constructor

Laptop(String brand, int ram) {

this.brand = brand;

this.ram = ram;

}

void display() {

System.out.println("Brand: " + brand);

System.out.println("RAM: " + ram + "GB");

}

}

public class Main {

public static void main(String[] args) {

Laptop laptop1 = new Laptop();

Laptop laptop2 = new Laptop("Dell", 16);

laptop1.display();

laptop2.display();

}

}

**10. Constructor Overloading in Student Class**

**Program: Define a class with overloaded constructors for a student**

class Student {

String name;

int age;

// No-arg constructor

Student() {

name = "Unknown";

age = 0;

}

// Parameterized constructor

Student(String name, int age) {

this.name = name;

this.age = age;

}

void display() {

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

System.out.println("Age: " + age);

}

}

public class Main {

public static void main(String[] args) {

Student student1 = new Student();

Student student2 = new Student("John", 20);

student1.display();

student2.display();

}

}

**11. Defining Multiple Constructors**

**Program: Define a class with multiple constructors and fields**

class Mobile {

String brand;

String model;

double price;

// No-arg constructor

Mobile() {

brand = "Samsung";

model = "Galaxy";

price = 0.0;

}

// Parameterized constructor

Mobile(String brand, String model, double price) {

this.brand = brand;

this.model = model;

this.price = price;

}

void display() {

System.out.println("Brand: " + brand);

System.out.println("Model: " + model);

System.out.println("Price: $" + price);

}

}

public class Main {

public static void main(String[] args) {

Mobile mobile1 = new Mobile();

Mobile mobile2 = new Mobile("Apple", "iPhone", 999.99);

mobile1.display();

mobile2.display();

}

}

**Method Overloading**

1. **Question:** Create a class Calculator with an overloaded add method that adds two integers and two doubles.

**Answer:**

java

Copy code

class Calculator {

int add(int a, int b) {

return a + b;

}

double add(double a, double b) {

return a + b;

}

public static void main(String[] args) {

Calculator calc = new Calculator();

System.out.println(calc.add(2, 3)); // Output: 5

System.out.println(calc.add(2.5, 3.5)); // Output: 6.0

}

}

1. **Question:** Create a class Printer with an overloaded print method that prints an integer, a double, and a string.

**Answer:**

java

Copy code

class Printer {

void print(int i) {

System.out.println(i);

}

void print(double d) {

System.out.println(d);

}

void print(String s) {

System.out.println(s);

}

public static void main(String[] args) {

Printer printer = new Printer();

printer.print(5); // Output: 5

printer.print(5.5); // Output: 5.5

printer.print("Hello"); // Output: Hello

}

}

1. **Question:** Write a class Volume with an overloaded method calculateVolume to calculate the volume of a cube, a rectangular box, and a cylinder.

**Answer:**

java

Copy code

class Volume {

double calculateVolume(double side) {

return side \* side \* side;

}

double calculateVolume(double length, double width, double height) {

return length \* width \* height;

}

double calculateVolume(double radius, double height) {

return Math.PI \* radius \* radius \* height;

}

public static void main(String[] args) {

Volume volume = new Volume();

System.out.println(volume.calculateVolume(3)); // Output: 27.0 (cube)

System.out.println(volume.calculateVolume(2, 3, 4)); // Output: 24.0 (rectangular box)

System.out.println(volume.calculateVolume(2, 5)); // Output: 62.83185307179586 (cylinder)

}

}

**Static Keyword**

1. **Question:** Create a class Counter with a static variable count that increments every time an object is created.

**Answer:**

java

Copy code

class Counter {

static int count = 0;

Counter() {

count++;

System.out.println("Count: " + count);

}

public static void main(String[] args) {

Counter c1 = new Counter(); // Output: Count: 1

Counter c2 = new Counter(); // Output: Count: 2

Counter c3 = new Counter(); // Output: Count: 3

}

}

1. **Question:** Write a class MathOperations with a static method square that returns the square of a number.

**Answer:**

java

Copy code

class MathOperations {

static int square(int x) {

return x \* x;

}

public static void main(String[] args) {

System.out.println(MathOperations.square(4)); // Output: 16

System.out.println(MathOperations.square(7)); // Output: 49

}

}

1. **Question:** Create a class Utility with a static block that prints "Static block executed" and a static method printMessage.

**Answer:**

java

Copy code

class Utility {

static {

System.out.println("Static block executed");

}

static void printMessage() {

System.out.println("Hello from static method");

}

public static void main(String[] args) {

Utility.printMessage(); // Output: Static block executed

// Hello from static method

}

}

**This Keyword**

1. **Question:** Create a class Person with instance variables name and age. Use the this keyword to differentiate between instance variables and method parameters.

**Answer:**

java

Copy code

class Person {

String name;

int age;

Person(String name, int age) {

this.name = name;

this.age = age;

}

void display() {

System.out.println("Name: " + this.name + ", Age: " + this.age);

}

public static void main(String[] args) {

Person p = new Person("John", 25);

p.display(); // Output: Name: John, Age: 25

}

}

1. **Question:** Write a class Rectangle with instance variables length and width. Use this keyword to invoke the constructor with the same parameter names.

**Answer:**

java

Copy code

class Rectangle {

int length;

int width;

Rectangle(int length, int width) {

this.length = length;

this.width = width;

}

int area() {

return this.length \* this.width;

}

public static void main(String[] args) {

Rectangle rect = new Rectangle(5, 3);

System.out.println("Area: " + rect.area()); // Output: Area: 15

}

}

1. **Question:** Create a class Employee with instance variables id and name. Use the this keyword to pass the current object as an argument in a method call.

**Answer:**

java

Copy code

class Employee {

int id;

String name;

Employee(int id, String name) {

this.id = id;

this.name = name;

}

void display(Employee e) {

System.out.println("ID: " + e.id + ", Name: " + e.name);

}

void show() {

display(this);

}

public static void main(String[] args) {

Employee emp = new Employee(101, "Alice");

emp.show(); // Output: ID: 101, Name: Alice

}

}

1. **Question:** Write a class Car with an instance variable model. Use this keyword to return the current class instance from a method.

**Answer:**

java

Copy code

class Car {

String model;

Car(String model) {

this.model = model;

}

Car getCar() {

return this;

}

void display() {

System.out.println("Model: " + this.model);

}

public static void main(String[] args) {

Car car = new Car("Tesla");

car.getCar().display(); // Output: Model: Tesla

}

}