Q1. Create a base class Shape with a method draw(). Derive two classes Circle and Square from Shape. Implement the draw() method in each derived class to display a message specific to the shape. Demonstrate polymorphism by creating an array of Shape objects and calling the draw() method for each object.

**package** Inheritance;

**class** Shape {

**void** draw() {

System.***out***.println("Drawing a generic shape");

}

}

**class** Circle **extends** Shape {

**void** draw() {

System.***out***.println("Drawing a circle");

}

}

**class** Square **extends** Shape {

**void** draw() {

System.***out***.println("Drawing a square");

}

}

**public** **class** ShapeDemo {

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

Shape[] shapes = **new** Shape[3];

shapes[0] = **new** Circle();

shapes[1] = **new** Square();

shapes[2] = **new** Shape();

**for** (Shape shape : shapes) {

shape.draw();

}

}

}

Q2. Explain the difference between compile-time polymorphism and runtime polymorphism in Java. Provide examples for each.

Compile time :

**package** Inheritance;

**class** CompileTimePloymorphism {

**void** display() {

System.***out***.println("In Display without parameter");

}

**void** display(String value) {

System.***out***.println("In Display with parameter" + value);

}

}

**public** **class** Compiletime {

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

CompileTimePloymorphism obj = **new** CompileTimePloymorphism();

obj.display();

obj.display("Polymorphism");

}

}

Runtime :

**package** Inheritance;

**class** RunTimePolymorphism {

**public** **void** display() {

System.***out***.println("Overridden Method");

}

}

**class** Runtimesub **extends** RunTimePolymorphism {

**public** **void** display() {

System.***out***.println("Overriding Method");

}

}

**public** **class** Runtime {

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

RunTimePolymorphism obj = **new** RunTimePolymorphism();

obj.display();

}

}

Q3. How can you achieve method overloading and method overriding in Java? Provide examples for both.

Method Overloading :

**package** Inheritance;

**class** Adder{

**static** **int** add(**int** a , **int** b) {

**return** a+b;

}

**static** **double** add(**double** a , **double** b) {

**return** a+b;

}

}

**public** **class** MethodOverloading {

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

Adder a = **new** Adder();

System.***out***.println(Adder.*add*(2,3));

System.***out***.println(Adder.*add*(2.5,3.5));

}

}

Method Overriding:

**package** Inheritance;

**class** Vehicle1{

**void** run(){System.***out***.println("Vehicle is running");}

}

**public** **class** Bike {

**void** run(){System.***out***.println("Bike is running safely");

}

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

Bike obj = **new** Bike();

obj.run();

}

}

Q4. Create a base class Vehicle with properties like speed and fuelCapacity. Derive two classes Car and Motorcycle from Vehicle. Include additional properties like numDoors for Car and hasSideCar for Motorcycle. Demonstrate the use of inheritance by creating objects of both derived classes.

**package** Inheritance;

**class** Vehicle {

**private** **int** speed;

**private** **int** fuelCapacity;

**public** Vehicle(**int** speed, **int** fuelCapacity) {

**this**.speed = speed;

**this**.fuelCapacity = fuelCapacity;

}

**public** **int** getSpeed() {

**return** speed;

}

**public** **int** getFuelCapacity() {

**return** fuelCapacity;

}

}

**class** Car **extends** Vehicle {

**private** **int** numDoors;

**public** Car(**int** speed, **int** fuelCapacity, **int** numDoors) {

**super**(speed, fuelCapacity);

**this**.numDoors = numDoors;

}

**public** **int** getNumDoors() {

**return** numDoors;

}

}

**class** Motorcycle **extends** Vehicle {

**private** **boolean** hasSideCar;

**public** Motorcycle(**int** speed, **int** fuelCapacity, **boolean** hasSideCar) {

**super**(speed, fuelCapacity);

**this**.hasSideCar = hasSideCar;

}

**public** **boolean** hasSideCar() {

**return** hasSideCar;

}

}

**public** **class** VehicleDemo {

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

Car myCar = **new** Car(60, 50, 4);

Motorcycle myMotorcycle = **new** Motorcycle(80, 20, **true**);

System.***out***.println("Car Speed: " + myCar.getSpeed());

System.***out***.println("Car Fuel Capacity: " + myCar.getFuelCapacity());

System.***out***.println("Car Number of Doors: " + myCar.getNumDoors());

System.***out***.println("\nMotorcycle Speed: " + myMotorcycle.getSpeed());

System.***out***.println("Motorcycle Fuel Capacity: " + myMotorcycle.getFuelCapacity());

System.***out***.println("Motorcycle Has Sidecar: " + myMotorcycle.hasSideCar());

}

}

Q5. Create a class Calculator with multiple overloaded methods named add that can accept different types of parameters (int, double, and String). Demonstrate how method overloading can be used for polymorphism.

**package** Inheritance;

**public** **class** Calculator {

**public** **int** add(**int** a, **int** b) {

**return** a + b;

}

**public** **double** add(**double** a, **double** b) {

**return** a + b;

}

**public** String add(String a, String b) {

**return** a + b;

}

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

Calculator calculator = **new** Calculator();

System.***out***.println("Sum of integers: " + calculator.add(5, 10));

System.***out***.println("Sum of doubles: " + calculator.add(5.5, 10.5));

System.***out***.println("Concatenated strings: " +calculator.add("Hello", " World"));

}

}

Q6. Define a base class Animal with a method makeSound(). Derive two classes, Dog and Cat, from Animal. Override the makeSound() method in each derived class to make a specific sound. Create an array of Animal objects and call the makeSound() method for each object. Show how this illustrates runtime polymorphism.

**package** Inheritance;

**class** Animal {

**void** makeSound() {

System.***out***.println("Generic Animal Sound");

}

}

**class** Dog **extends** Animal {

**void** makeSound() {

System.***out***.println("Bark");

}

}

**class** Cat **extends** Animal {

**void** makeSound() {

System.***out***.println("Meow");

}

}

**public** **class AnimalDemo** {

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

Animal[] animals=**new** Animal[3];

animals[0] = **new** Dog();

animals[1] = **new** Cat();

animals[2]=**new** Animal();

**for**(Animal animal : animals) {

animal.makeSound();

}

}

}

Q7. Create a base class Vehicle with a method startEngine() that returns a Vehicle. Derive two classes, Car and Motorcycle, and override the startEngine() method to return an instance of the derived class. Demonstrate how covariant return types can be used for polymorphism.

**package** Relationship;

**class** Vehicle {

Vehicle startEngine() {

System.***out***.println("Generic Vehicle Engine Started");

**return** **this**;

}

}

**class** Car **extends** Vehicle {

Car startEngine() {

System.***out***.println("Car Engine Started");

**return** **this**;

}

}

**class** Motorcycle **extends** Vehicle {

Motorcycle startEngine() {

System.***out***.println("Motorcycle Engine Started");

**return** **this**;

}

}

**public** **class** Returntype {

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

Vehicle vehicle1 = **new** Car();

Vehicle vehicle2 = **new** Motorcycle();

vehicle1.startEngine();

vehicle2.startEngine();

}

}