Problem 1: Method Overriding in Polymorphism

Problem:
Create a superclass `Shape` with a method `area()`. Create subclasses `Circle` and `Rectangle`, each with its own implementation of `area()`. Use polymorphism to calculate the area of both shapes.

Solution:

```
java
Copy code
class Shape {
    public void area() {
        System.out.println("Calculating area...");
    }
}

class Circle extends Shape {
    double radius;

    Circle(double radius) {
        this.radius = radius;
    }

    @Override
    public void area() {
        System.out.println("Area of Circle: " + Math.PI * radius * radius);
    }
}

class Rectangle extends Shape {
    double length, breadth;

    Rectangle(double length, double breadth) {
        this.length = length;
        this.breadth = breadth;
    }

    @Override
    public void area() {
        System.out.println("Area of Rectangle: " + length * breadth);
    }
}

public class Main {
    public static void main(String[] args) {
        Shape s1 = new Circle(5);   // Circle object
        Shape s2 = new Rectangle(4, 5);  // Rectangle object

        s1.area();  // Calls Circle's area() method
        s2.area();  // Calls Rectangle's area() method
    }
}
```

Output:

```
mathematica
Copy code
Area of Circle: 78.53981633974483
Area of Rectangle: 20.0
```

Problem 2: Method Overloading in Polymorphism

Problem:
Create a class `MathOperations` with a method `add()` that is overloaded to add two integers and three integers. Demonstrate compile-time polymorphism by calling both overloaded methods.

Solution:

```
java
Copy code
class MathOperations {
    // Method to add two numbers
    public int add(int a, int b) {
        return a + b;
    }

    // Overloaded method to add three numbers
    public int add(int a, int b, int c) {
        return a + b + c;
    }
}

public class Main {
    public static void main(String[] args) {
        MathOperations math = new MathOperations();

        System.out.println("Sum of 2 and 3: " + math.add(2, 3));         // 2
parameters
        System.out.println("Sum of 1, 2, and 3: " + math.add(1, 2, 3));  // 3
parameters
    }
}
```

Output:

```
mathematica
Copy code
Sum of 2 and 3: 5
Sum of 1, 2, and 3: 6
```

Problem 3: Polymorphism in Arrays

Problem:
Create a superclass `Animal` with a method `speak()`. Create subclasses `Dog` and `Cat`, each

overriding `speak()`. Use an array of `Animal` objects to store instances of `Dog` and `Cat`, and invoke the `speak()` method for each object.

Solution:

```
java
Copy code
class Animal {
    public void speak() {
        System.out.println("Animal speaks.");
    }
}

class Dog extends Animal {
    @Override
    public void speak() {
        System.out.println("Dog barks.");
    }
}

class Cat extends Animal {
    @Override
    public void speak() {
        System.out.println("Cat meows.");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal[] animals = { new Dog(), new Cat() };

        // Loop through the array and call speak() for each
        for (Animal animal : animals) {
            animal.speak();  // Polymorphic call
        }
    }
}
```

Output:

```
Copy code
Dog barks.
Cat meows.
```

Problem 4: Runtime Polymorphism with Abstract Class

Problem:
Create an abstract class `Vehicle` with an abstract method `move()`. Create two subclasses `Car` and `Bicycle`, each with its own implementation of the `move()` method. Demonstrate runtime polymorphism by calling the `move()` method on both `Car` and `Bicycle` objects through a `Vehicle` reference.

Solution:

```
java
Copy code
abstract class Vehicle {
    public abstract void move();
}

class Car extends Vehicle {
    @Override
    public void move() {
        System.out.println("Car is moving.");
    }
}

class Bicycle extends Vehicle {
    @Override
    public void move() {
        System.out.println("Bicycle is moving.");
    }
}

public class Main {
    public static void main(String[] args) {
        Vehicle v1 = new Car();      // Car object as Vehicle
        Vehicle v2 = new Bicycle();  // Bicycle object as Vehicle

        v1.move();  // Calls Car's move() method
        v2.move();  // Calls Bicycle's move() method
    }
}
```

Output:

```
csharp
Copy code
Car is moving.
Bicycle is moving.
```