Week1 (Design Patterns and Principles)

Exercise1 – Implementing the Singleton Pattern

class Singleton {

    private Singleton() {

        System.out.println("Singleton instance created.");

    }

    private static class SingletonHelper {

        private static final Singleton INSTANCE = new Singleton();

    }

    public static Singleton getInstance() {

        return SingletonHelper.INSTANCE;

    }

    public void showMessage() {

        System.out.println("Hello from Singleton!");

    }

}

public class SingletonExample {

    public static void main(String[] args) {

        Singleton object1 = Singleton.getInstance();

        Singleton object2 = Singleton.getInstance();

        object1.showMessage();

        if (object1 == object2) {

            System.out.println("Both references point to the same Singleton instance.");

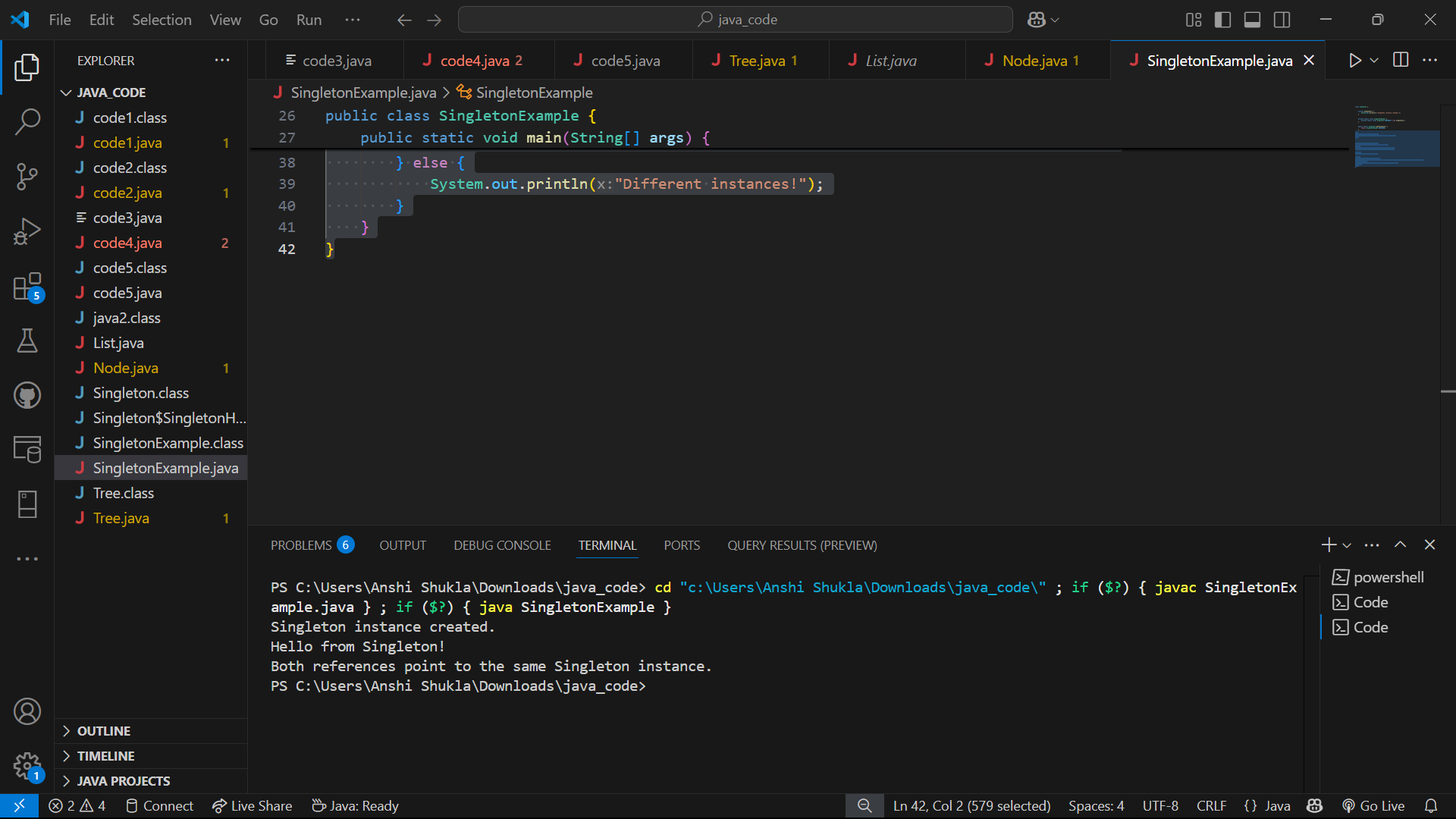
        } else {

            System.out.println("Different instances!");

        }

    }

}



Exercise 2 – Implementing the factory method pattern

interface Vehicle {

    void design();

}

class Car implements Vehicle {

    public void design() {

        System.out.println("Designing a Car.");

    }

}

class Bike implements Vehicle {

    public void design() {

        System.out.println("Designing a Bike.");

    }

}

abstract class VehicleFactory {

    public abstract Vehicle createVehicle();

}

class CarFactory extends VehicleFactory {

    public Vehicle createVehicle() {

        return new Car();

    }

}

class BikeFactory extends VehicleFactory {

    public Vehicle createVehicle() {

        return new Bike();

    }

}

public class main {

    public static void main(String[] args) {

        VehicleFactory carFactory = new CarFactory();

        Vehicle car = carFactory.createVehicle();

        car.design();

        VehicleFactory bikeFactory = new BikeFactory();

        Vehicle bike = bikeFactory.createVehicle();

        bike.design();

    }

}

