//Abstraction in Java is a fundamental concept that allows you to represent the essential features of an object without including the implementation details. It helps to simplify complex systems by focusing on what an object does rather than how it does it.//

//Abstract Classes and interfaces//

//An abstract class is a class that cannot be instantiated directly and may contain abstract methods (methods without a body).//

//An interface is a reference type in Java that is similar to a class but contains only constants, method signatures, default methods, static methods, and nested types.//

abstract class Shape {

public abstract double calculateArea();

}

class Circle extends Shape {

private double radius;

public Circle(double radius) {

this.radius = radius;

}

@Override

public double calculateArea() {

return Math.PI \* radius \* radius;

}

}

interface Drawable {

void draw();

}

class Rectangle implements Drawable {

private double length;

private double width;

public Rectangle(double length, double width) {

this.length = length;

this.width = width;

}

@Override

public void draw() {

System.out.println("Drawing a rectangle");

}

}

public class AbstractionExample {

public static void main(String[] args) {

Circle circle = new Circle(5);

Rectangle rectangle = new Rectangle(4, 6);

System.out.println("Area of circle: " + circle.calculateArea());

rectangle.draw();

}

}

Sterilization.example

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.ObjectOutputStream;

import java.io.Serializable;

class Customer implements Serializable {

private int id;

private String name;

private String contactNo;

private String address;

public Customer(int id, String name, String contactNo, String address) {

this.id = id;

this.name = name;

this.contactNo = contactNo;

this.address = address;

}

@Override

public String toString() {

return "Customer{" +

"id=" + id +

", name='" + name + '\'' +

", contactNo='" + contactNo + '\'' +

", address='" + address + '\'' +

'}';

}

}

public class SerializationExample {

public static void main(String[] args) {

Customer customer = new Customer(1, "John Doe", "1234567890", "123 Main Street");

try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("JavaObject.txt"))) {

oos.writeObject(customer);

System.out.println("Customer object serialized successfully!");

} catch (IOException e) {

e.printStackTrace();

}

}

}

Synchronisation.example

class Counter {

private int count = 0;

public synchronized void increment() {

count++;

}

public synchronized void decrement() {

count--;

}

public int getCount() {

return count;

}

}

public class SynchronizationExample {

public static void main(String[] args) throws InterruptedException {

Counter counter = new Counter();

Thread thread1 = new Thread(() -> {

for (int i = 0; i < 1000; i++) {

counter.increment();

}

});

Thread thread2 = new Thread(() -> {

for (int i = 0; i < 1000; i++) {

counter.increment();

}

});

thread1.start();

thread2.start();

thread1.join();

thread2.join();

System.out.println("Final count: " + counter.getCount());

}

}