*** SADP Practicals***

1.Write a Java Program to implement I/O Decorator for converting uppercase letters to lower case letters.

Main.java

```
package sadp_practicals;
import java.io.*;
public class Main {
   public static void main(String[] args) throws IOException {
      int c;
      try {
         InputStream in = new LowerCaseInputStream(
               new BufferedInputStream(
                     new FileInputStream("test.txt")));
         while((c = in.read()) >= 0) {
            System.out.print((char)c);
         }
         in.close();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

LowerCaseInputStream.java

```
package sadp_practicals;
import java.io.*;
class LowerCaseInputStream extends FilterInputStream {
   public LowerCaseInputStream(InputStream in) {
      super(in);
   }

   public int read() throws IOException {
      int c = super.read();
      return (c == -1 ? c : Character.toLowerCase((char)c));
```

```
    }

    public int read(byte[] b, int offset, int len) throws IOException {
        int result = super.read(b, offset, len);
        for (int i = offset; i < offset + result; i++) {
            b[i] = (byte)Character.toLowerCase((char)b[i]);
        }
        return result;
    }
}
```

test.txt

```
INDIA IS MY COUNTRY
```

2. Write a Java Program to implement Singleton pattern for multithreading

Main.java

```
package sadp2;
public class Main {
    public static void main(String[] args) {
        Thread thread1 = new Thread(() -> {
            Singleton instance1 = Singleton.getInstance();
            instance1.showMessage();
        });
        Thread thread2 = new Thread(() -> {
            Singleton instance2 = Singleton.getInstance();
            instance2.showMessage();
        });
        thread1.start();
             thread2.start();
    }
}
```

Singleton.java

```
package sadp2;
public class Singleton {
    private static volatile Singleton instance = null;
    private Singleton() {
        System.out.println("Singleton instance created");
    }
    public static Singleton getInstance() {
        if (instance == null) {
            synchronized (Singleton.class) {
                if (instance == null) {
                    instance = new Singleton();
                }
            }
        }
        return instance;
    }
    public void showMessage() {
        System.out.println("Singleton method accessed by: " +
Thread.currentThread().getName());
    }
}
```

3. Write a JAVA Program to implement built-in support (java.util.Observable) Weather station with members temperature, humidity, pressure and methods mesurmentsChanged(), setMesurment(), getTemperature(), getHumidity(),

getPressure()

Main.java

```
package sadp3;
public class Main {
    public static void main(String[] args) {
        WeatherStation weatherStation = new WeatherStation();
        WeatherDisplay display = new WeatherDisplay();
        weatherStation.addObserver(display);
        weatherStation.setMeasurement(25.5f, 65.0f, 1013.1f);
        weatherStation.setMeasurement(26.3f, 70.0f, 1012.5f);
        weatherStation.setMeasurement(27.8f, 72.0f, 1011.9f);
    }
}
```

WeatherDisplay.java

```
package sadp3;
public class WeatherDisplay implements WeatherObserver {
 private float temperature;
 private float humidity;
 private float pressure;
 @Override
 public void update(float temperature, float humidity, float pressure) {
     this.temperature = temperature;
     this.humidity = humidity;
     this.pressure = pressure;
     display();
 }
 public void display() {
     System.out.println("Current conditions:");
     System.out.println("Temperature: " + temperature + "°C");
     System.out.println("Humidity: " + humidity + "%");
     System.out.println("Pressure: " + pressure + " hPa");
 }
}
```

WeatherObserver.java

```
package sadp3;
public interface WeatherObserver {
 void update(float temperature, float humidity, float pressure);
}
```

WeatherStation.java

```
package sadp3;
import java.util.ArrayList;
import java.util.List;
public class WeatherStation {
   private List<WeatherObserver> observers;
   private float temperature;
   private float humidity;
   private float pressure;
   public WeatherStation() {
      observers = new ArrayList<>();
   }
   public void addObserver(WeatherObserver observer) {
      observers.add(observer);
   }
   public void removeObserver(WeatherObserver observer) {
      observers.remove(observer);
   }
   private void notifyObservers() {
      for (WeatherObserver observer : observers) {
         observer.update(temperature, humidity, pressure);
      }
   }
   public void setMeasurement(float temperature, float humidity, float pressure) {
      this.temperature = temperature;
      this.humidity = humidity;
      this.pressure = pressure;
      notifyObservers();
   }
   public float getTemperature() {
      return temperature;
   }
   public float getHumidity() {
        return humidity;
    }
    public float getPressure() {
        return pressure;
    }
}
```

4. Write a Java Program to implement Factory method for Pizza Store with createPizza(),orderPizza(), prepare(), Bake(), cut(), box(). Use this to create variety of pizza's like NyStyleCheesePizza, ChicagoStyleCheesePizza etc.

ChicagoPizzaStore.java

```
package sadp4;
public class ChicagoPizzaStore extends PizzaStore {
    @Override
    protected Pizza createPizza(String type) {
        if (type.equals("cheese")) {
            return new ChicagoStyleCheesePizza();
        }
        return null;
    }
}
```

ChicagoStyleCheesePizza.java

```
package sadp4;
public class ChicagoStyleCheesePizza extends Pizza {
    public ChicagoStyleCheesePizza() {
        name = "Chicago Style Deep Dish Cheese Pizza";
    }
    @Override
    public void cut() {
        System.out.println("Cutting pizza into square slices");
    }
}
```

NyPizzaStore.java

```
package sadp4;
public class NyPizzaStore extends PizzaStore {
   @Override
   protected Pizza createPizza(String type) {
      if (type.equals("cheese")) {
         return new NyStyleCheesePizza();
      }
      return null;
   }
}
```

NyStyleCheesePizza.java

```
package sadp4;
public class NyStyleCheesePizza extends Pizza {
   public NyStyleCheesePizza() {
      name = "New York Style Cheese Pizza";
   }
}
```

Pizza.java

```
package sadp4;
public abstract class Pizza {
   String name;
   public void prepare() {
      System.out.println("Preparing " + name);
   }
   public void bake() {
      System.out.println("Baking " + name);
   }
   public void cut() {
      System.out.println("Cutting " + name);
   }
   public void box() {
        System.out.println("Boxing " + name);
    }
    public String getName() {
        return name;
    }
}
```

PizzaStore.java

```
package sadp4;
public abstract class PizzaStore {
    public Pizza orderPizza(String type) {
        Pizza pizza = createPizza(type);
        pizza.prepare();
        pizza.bake();
        pizza.cut();
        pizza.box();
        return pizza;
    }
    protected abstract Pizza createPizza(String type);
}
```

Main.java

```
package sadp4;
public class Main {
    public static void main(String[] args) {
        PizzaStore nyStore = new NyPizzaStore();
        PizzaStore chicagoStore = new ChicagoPizzaStore();
        Pizza pizza1 = nyStore.orderPizza("cheese");
        System.out.println("Ordered a " + pizza1.getName() + "\n");

        Pizza pizza2 = chicagoStore.orderPizza("cheese");
        System.out.println("Ordered a " + pizza2.getName() + "\n");
    }
}
```

5. Write a Java Program to implement Adapter pattern for Enumeration iterator

EnumerationIterator.java

```
package sadp5;

import java.util.Enumeration;

import java.util.Iterator;

public class EnumerationIterator<E> implements Iterator<E> {

   private Enumeration<E> enumeration;


   public EnumerationIterator(Enumeration<E> enumeration) {

      this.enumeration = enumeration;

   }

   public boolean hasNext() {

      return enumeration.hasMoreElements();

   }

   public E next() {

      return enumeration.nextElement();

   }

   public void remove() {

      throw new UnsupportedOperationException();

   }

}
```

Main.java

```
package sadp5;

import java.util.*;

public class Main {

   public static void main(String[] args) {

      Vector<String> vector = new Vector<>();

      vector.add("Item 1");

      vector.add("Item 2");

      vector.add("Item 3");
```

```
        Enumeration<String> enumeration = vector.elements();
        Iterator<String> iterator = new EnumerationIterator<>(enumeration);
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }
}
```

6.Write a java Program to implement command pattern to test Remote Control

Command.java

```
package sadp6;
public interface Command {
    void execute();
}
```

Light.java

```
package sadp6;
public class Light {
    public void on() {
        System.out.println("Light is ON");
    }
    public void off() {
        System.out.println("Light is OFF");
    }
}
```

LightOffCommand.java

```
package sadp6;
public class LightOffCommand implements Command {
    private Light light;
    public LightOffCommand(Light light) {
        this.light = light;
    }
    public void execute() {
        light.off();
    }
}
```

LightOnCommand.java

```
package sadp6;
public class LightOnCommand implements Command {
    private Light light;
    public LightOnCommand(Light light) {
        this.light = light;
    }
    public void execute() {
        light.on();
    }
}
```

RemoteControl.java

```
package sadp6;
public class RemoteControl {
    private Command command;
    public void setCommand(Command command) {
        this.command = command;
    }
    public void pressButton() {
        command.execute();
    }
}
```

Main.java

```
package sadp6;
public class Main {
    public static void main(String[] args) {
        Light light = new Light();
        Command lightOn = new LightOnCommand(light);
        Command lightOff = new LightOffCommand(light);
        RemoteControl remote = new RemoteControl();
        remote.setCommand(lightOn);
        remote.pressButton();
        remote.setCommand(lightOff);
        remote.pressButton();
    }
}
```

7. Write a Java Program to implement undo command to test Ceiling fan.

CeilingFan.java

```
package sadp7;
public class CeilingFan {
    private String speed;
    public void high() {
        speed = "High";
        System.out.println("Ceiling fan is on High");
    }
    public void medium() {
        speed = "Medium";
        System.out.println("Ceiling fan is on Medium");
    }
    public void off() {
        speed = "Off";
        System.out.println("Ceiling fan is Off");
    }
    public String getSpeed() {
        return speed;
    }
}
```

CeilingFanHighCommand.java

```
package sadp7;
public class CeilingFanHighCommand implements Command {
    private CeilingFan fan;
    private String prevSpeed;
    public CeilingFanHighCommand(CeilingFan fan) {
        this.fan = fan;
    }
    public void execute() {
        prevSpeed = fan.getSpeed();
        fan.high();
    }
    public void undo() {
        if (prevSpeed != null) {
            switch (prevSpeed) {
                case "Medium":
                    fan.medium();
                    break;
                case "High":
                    fan.high();
                    break;
                default:
                    fan.off();
                    break;
            }
        } else {
            fan.off(); // If prevSpeed is null, we assume it was off.
        }
    }
}
```

Command.java

```
package sadp7;
public interface Command {
    void execute();
    void undo();
}
```

RemoteControl.java

```
package sadp7;
public class RemoteControl {
    private Command command;
    public void setCommand(Command command) {
        this.command = command;
    }
    public void pressButton() {
        command.execute();
    }
    public void pressUndo() {
        command.undo();
    }
}
```

Main.java

```
package sadp7;
public class Main {
    public static void main(String[] args) {
        CeilingFan fan = new CeilingFan();
        Command fanHigh = new CeilingFanHighCommand(fan);

        RemoteControl remote = new RemoteControl();
        remote.setCommand(fanHigh);

        remote.pressButton();
        remote.pressUndo();
    }
}
```

8. Write a Java Program to implement State Pattern for Gumball Machine.

Create instance variable that holds current state from there, we just need to handle all

actions, behaviors and state transition that can happen

9. Design simple HR Application using Spring Framework

10. Write a Java Program to implement Strategy Pattern for Duck Behavior. Create instance variable that holds current state of Duck from there, we just need to handle all Flying Behaviors and Quack Behavior

Duck.java

```
package sadp10;
public abstract class Duck {
   FlyBehavior flyBehavior;
   QuackBehavior quackBehavior;
   public Duck() {}
   public void performFly() {
      flyBehavior.fly();
   }
   public void performQuack() {
      quackBehavior.quack();
   }
   public void swim() {
      System.out.println("All ducks float!");
   }
   public void setFlyBehavior(FlyBehavior fb) {
      flyBehavior = fb;
   }
   public void setQuackBehavior(QuackBehavior qb) {
      quackBehavior = qb;
   }
}
```

FlyBehavior.java

```
package sadp10;
public interface FlyBehavior {
   void fly();
}
```

FlyNoWay.java

```
package sadp10;
```

```
public class FlyNoWay implements FlyBehavior {
    public void fly() {
        System.out.println("I can't fly.");
    }
}
```

FlyWithWings.java

```
package sadp10;
public class FlyWithWings implements FlyBehavior {
    public void fly() {
        System.out.println("I'm flying with wings!");
    }
}
```

MallardDuck.java

```
package sadp10;
public class MallardDuck extends Duck {
    public MallardDuck() {
        flyBehavior = new FlyWithWings();
        quackBehavior = new Quack();
    }
}
```

Quack.java

```
package sadp10;
public class Quack implements QuackBehavior {
    public void quack() {
        System.out.println("Quack!");
    }
}
```

QuackBehavior.java

```
package sadp10;
public interface QuackBehavior {
    void quack();
}
```

RubberDuck.java

```
package sadp10;
public class RubberDuck extends Duck {
    public RubberDuck() {
        flyBehavior = new FlyNoWay();
        quackBehavior = new Squeak();
    }
}
```

Squeak.java

```
package sadp10;
public class Squeak implements QuackBehavior {
    public void quack() {
        System.out.println("Squeak!");
    }
}
```

Main.java

```
package sadp10;
public class Main {
    public static void main(String[] args) {
        Duck mallard = new MallardDuck();
        mallard.performFly();
        mallard.performQuack();

        Duck rubberDuck = new RubberDuck();
        rubberDuck.performFly();
        rubberDuck.performQuack();

        rubberDuck.setFlyBehavior(new FlyWithWings());
        rubberDuck.performFly();
    }
}
```

11. Write a java program to implement Adapter pattern to design Heart Model to Beat Model

BeatModel.java

```
package sadp11;
public interface BeatModel {
    void start();
    void stop();
    void setBPM(int bpm);
    int getBPM();
}
```

Heart.java

```
package sadp11;
public class Heart {
    private int bpm;

    public void beat() {
        System.out.println("Heart is beating...");
    }
    public void setBPM(int bpm) {
        this.bpm = bpm;
        System.out.println("Heart BPM set to: " + bpm);
    }
    public int getBPM() {
        return bpm;
    }
}
```

HeartAdapter.java

```
package sadp11;
public class HeartAdapter implements BeatModel {
    private Heart heart;
    public HeartAdapter(Heart heart) {
        this.heart = heart;
    }
    public void start() {
        heart.beat();
    }
    public void stop() {
        System.out.println("Heart stopped beating.");
    }
    public void setBPM(int bpm) {
        heart.setBPM(bpm);
    }
    public int getBPM() {
        return heart.getBPM();
    }
}
```

Main.java

```
package sadp11;
public class Main {
    public static void main(String[] args) {
        Heart heart = new Heart();
        BeatModel heartModel = new HeartAdapter(heart);

        heartModel.setBPM(75);
        heartModel.start();
        heartModel.stop();
    }
}
```

12. Write a Java Program to implement Decorator Pattern for interface Car to define the assemble() method and then decorate it to Sports car and Luxury Car

BasicCar.java

```
package sadp12;
public class BasicCar implements Car {
    public String assemble() {
        return "Basic Car";
    }
}
```

Car.java

```
package sadp12;
public interface Car {
    String assemble();
}
```

CarDecorator.java

```
package sadp12;
public abstract class CarDecorator implements Car {
    protected Car car;
    public CarDecorator(Car car) {
        this.car = car;
    }
    public String assemble() {
        return car.assemble();
    }
}
```

LuxuryCar.java

```
package sadp12;
public class LuxuryCar extends CarDecorator {
    public LuxuryCar(Car car) {
        super(car);
    }
    public String assemble() {
        return super.assemble() + ", Adding features of Luxury Car";
    }
}
```

SportsCar.java

```
package sadp12;
```

```
public class SportsCar extends CarDecorator {
    public SportsCar(Car car) {
        super(car);
    }
    public String assemble() {
        return super.assemble() + ", Adding features of Sports Car";
    }
}
```

Main.java

```
package sadp12;
public class Main {
    public static void main(String[] args) {
        Car basicCar = new BasicCar();
        System.out.println(basicCar.assemble());
        Car sportsCar = new SportsCar(basicCar);
        System.out.println(sportsCar.assemble());
        Car luxuryCar = new LuxuryCar(basicCar);
        System.out.println(luxuryCar.assemble());
        Car sportsLuxuryCar = new SportsCar(new LuxuryCar(basicCar));
        System.out.println(sportsLuxuryCar.assemble());
    }
}
```

13. Write a Java Program to implement an Adapter design pattern in mobile charger. Define two classes – Volt (to measure volts) and Socket (producing constant volts of 120V). Build an adapter that can produce 3 volts, 12 volts and default 120 volts.

Implements Adapter pattern using Class Adapter

ChargerAdapter.java

```
package sadp13;
public class ChargerAdapter extends Socket {
    public Volt get3Volt() {
        return convertVolt(getVolt(), 40);
    }
    public Volt get12Volt() {
        return convertVolt(getVolt(), 10);
    }
    private Volt convertVolt(Volt v, int divisor) {
        return new Volt(v.getVolts() / divisor);
    }
}
```

Socket.java

```
package sadp13;
public class Socket {
    public Volt getVolt() {
        return new Volt(120);
    }
}
```

Volt.java

```
package sadp13;
public class Volt {
    private int volts;
    public Volt(int volts) {
        this.volts = volts;
    }
    public int getVolts() {
        return volts;
    }
}
```

Main.java

```
package sadp13;
public class Main {
    public static void main(String[] args) {
        ChargerAdapter chargerAdapter = new ChargerAdapter();
        Volt v3 = chargerAdapter.get3Volt();
        System.out.println("3 Volts: " + v3.getVolts() + "V");

        Volt v12 = chargerAdapter.get12Volt();
        System.out.println("12 Volts: " + v12.getVolts() + "V");

        Volt v120 = chargerAdapter.getVolt();
        System.out.println("120 Volts: " + v120.getVolts() + "V");
    }
}
```

14. Write a Java Program to implement Command Design Pattern for Command Interface with execute() . Use this to create variety of commands for LightOnCommand,LightOffCommand, GarageDoorUpCommand, StereoOnWithCDComman.

Command.java

```
package sadp14;
public interface Command {
    void execute();
}
```

GarageDoor.java

```
package sadp14;
public class GarageDoor {
    public void up() {
        System.out.println("Garage Door is Open");
    }
    public void down() {
        System.out.println("Garage Door is Closed");
    }
}
```

GarageDoorUpCommand.java

```
package sadp14;
public class GarageDoorUpCommand implements Command {
    private GarageDoor garageDoor;
    public GarageDoorUpCommand(GarageDoor garageDoor) {
        this.garageDoor = garageDoor;
    }
    public void execute() {
        garageDoor.up();
    }
}
```

Light.java

```
package sadp14;
public class Light {
    public void on() {
        System.out.println("Light is ON");
    }
    public void off() {
        System.out.println("Light is OFF");
    }
}
```

LightOffCommand.java

```
package sadp14;
public class LightOffCommand implements Command {
    private Light light;

    public LightOffCommand(Light light) {
        this.light = light;
    }
    public void execute() {
        light.off();
    }
}
```

LightOnCommand.java

```
package sadp14;
```

```
public class LightOnCommand implements Command {
    private Light light;
    public LightOnCommand(Light light) {
        this.light = light;
    }
    public void execute() {
        light.on();
    }
}
```

RemoteControl.java

```
package sadp14;
public class RemoteControl {
    private Command command;
    public void setCommand(Command command) {
        this.command = command;
    }
    public void pressButton() {
        command.execute();
    }
}
```

Stereo.java

```
package sadp14;
public class Stereo {
    public void on() {
        System.out.println("Stereo is ON");
    }
    public void off() {
        System.out.println("Stereo is OFF");
    }
    public void setCD() {
        System.out.println("CD is set");
    }
    public void play() {
        System.out.println("Stereo is playing");
    }
}
```

StereoOnWithCDCommand.java

```
package sadp14;
public class StereoOnWithCDCommand implements Command {
    private Stereo stereo;
    public StereoOnWithCDCommand(Stereo stereo) {
        this.stereo = stereo;
    }
    public void execute() {
        stereo.on();
        stereo.setCD();
        stereo.play();
    }
}
```

Main.java

```
package sadp14;
public class Main {
    public static void main(String[] args) {
        Light light = new Light();
        GarageDoor garageDoor = new GarageDoor();
        Stereo stereo = new Stereo();
        Command lightOn = new LightOnCommand(light);
        Command lightOff = new LightOffCommand(light);
        Command garageDoorUp = new GarageDoorUpCommand(garageDoor);
        Command stereoOnWithCD = new StereoOnWithCDCommand(stereo);
        RemoteControl remote = new RemoteControl();
        remote.setCommand(lightOn);
        remote.pressButton();
```

```java
        remote.setCommand(lightOff);
        remote.pressButton();
        remote.setCommand(garageDoorUp);
        remote.pressButton();
        remote.setCommand(stereoOnWithCD);
        remote.pressButton();
    }
}
```

15. Write a Java Program to implement Facade Design Pattern for Home Theater

Amplifier.java

```java
package sadp15;
public class Amplifier {
    public void on() {
        System.out.println("Amplifier is ON");
    }
    public void setVolume(int level) {
        System.out.println("Volume set to " + level);
    }
    public void off() {
        System.out.println("Amplifier is OFF");
    }
}
```

HomeTheaterFacade.java

```java
package sadp15;
public class HomeTheaterFacade {
    private Amplifier amplifier;
    private Projector projector;
    private Screen screen;
    public HomeTheaterFacade(Amplifier amplifier, Projector projector, Screen screen) {
        this.amplifier = amplifier;
        this.projector = projector;
        this.screen = screen;
    }
    public void watchMovie() {
        System.out.println("Get ready to watch a movie...");
        screen.down();
        projector.on();
        projector.setInput("DVD");
        amplifier.on();
        amplifier.setVolume(10);
        System.out.println("Movie is starting!");
    }
    public void endMovie() {
        System.out.println("Shutting down the theater...");
        screen.up();
        projector.off();
        amplifier.off();
        System.out.println("Theater is closed.");
    }
}
```

16. Write a Java Program to implement Observer Design Pattern for number conversion. Accept a number in Decimal form and represent it in Hexadecimal, Octal and Binary. Change the Number and it reflects in other forms also

Main.java

```
package sadp16;
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Subject subject = new Subject();
        NumberConverter converter = new NumberConverter(subject);
        Scanner scanner = new Scanner(System.in);
        while (true) {
            System.out.print("Enter a decimal number (or -1 to exit): ");
            int decimal = scanner.nextInt();
            if (decimal == -1) {
                break;
            }
            subject.setDecimal(decimal);
        }
        scanner.close();
    }
}
```

NumberConverter.java

```
package sadp16;
public class NumberConverter implements Observer {
    private Subject subject;
    public NumberConverter(Subject subject) {
        this.subject = subject;
        subject.addObserver(this);
    }
    public void update(int decimal) {
        System.out.println("Decimal: " + decimal);
        System.out.println("Binary: " + Integer.toBinaryString(decimal));
        System.out.println("Octal: " + Integer.toOctalString(decimal));
        System.out.println("Hexadecimal: " + Integer.toHexString(decimal).toUpperCase());
        System.out.println();
    }}
```

Observer.java

```
package sadp16;
public interface Observer {
    void update(int decimal);
}
```

Subject.java

```
package sadp16;
import java.util.ArrayList;
import java.util.List;
public class Subject {
    private List<Observer> observers = new ArrayList<>();
    private int decimal;
    public void addObserver(Observer observer) {
        observers.add(observer);
    }
    public void removeObserver(Observer observer) {
        observers.remove(observer);
    }
    public void setDecimal(int decimal) {
        this.decimal = decimal;
        notifyObservers();
    }
    public int getDecimal() {
        return decimal;
    }
    private void notifyObservers() {
        for (Observer observer : observers) {
            observer.update(decimal);
        }
    }
}
```

17. Write a Java Program to implement Abstract Factory Pattern for Shape interface.

Circle.java

```
package sadp17;
public class Circle implements Shape {
```

```java
    public void draw() {
        System.out.println("Drawing a Circle");
    }
}
```

ColorShapeFactory.java

```java
package sadp17;
public class ColorShapeFactory extends ShapeFactory {
    public Shape createShape(String shapeType) {
        if (shapeType.equalsIgnoreCase("CIRCLE")) {
            return new Circle();
        } else if (shapeType.equalsIgnoreCase("SQUARE")) {
            return new Square();
        }
        return null;
    }
}
```

Shape.java

```java
package sadp17;
public interface Shape {
    void draw();
}
```

ShapeFactory.java

```java
package sadp17;
public abstract class ShapeFactory {
    public abstract Shape createShape(String shapeType);
}
```

Square.java

```java
package sadp17;
public class Square implements Shape {
    public void draw() {
        System.out.println("Drawing a Square");
    }
}
```

Main.java

```
package sadp17;
public class Main {
    public static void main(String[] args) {
        ShapeFactory shapeFactory = new ColorShapeFactory();
        Shape circle = shapeFactory.createShape("CIRCLE");
        circle.draw();
        Shape square = shapeFactory.createShape("SQUARE");
        square.draw();
    }}
```

18. Write a JAVA Program to implement built-in support (java.util.Observable) Weather station with members temperature, humidity, pressure and methods mesurmentsChanged(), setMesurment(), getTemperature(), getHumidity(),

getPressure()

WeatherData.java

```
package sadp18;
import java.util.Observable;
public class WeatherData extends Observable {
    private float temperature;
    private float humidity;
    private float pressure;
    public void measurementsChanged() {
        setChanged();
        notifyObservers();
    }
    public void setMeasurements(float temperature, float humidity, float pressure) {
        this.temperature = temperature;
        this.humidity = humidity;
        this.pressure = pressure;
        measurementsChanged();
    }
    public float getTemperature() {
        return temperature;
    }
    public float getHumidity() {
        return humidity;
    }
    public float getPressure() {
        return pressure;
    }
}
```

WeatherDisplay.java

```
package sadp18;
import java.util.Observable;
import java.util.Observer;
public class WeatherDisplay implements Observer {
    public void update(Observable obs, Object arg) {
        if (obs instanceof WeatherData) {
            WeatherData weatherData = (WeatherData) obs;
            System.out.println("Current conditions: " +
                weatherData.getTemperature() + "°C, " +
                weatherData.getHumidity() + "% humidity, " +
                weatherData.getPressure() + " hPa");
        }
    }
}
```

19. Write a Java Program to implement Factory method for Pizza Store with createPizza(),orderPizza(), prepare(), Bake(), cut(), box(). Use this to create variety of pizza's like NyStyleCheesePizza, ChicagoStyleCheesePizza etc.

ChicagoPizzaStore.java

```
package sadp4;
public class ChicagoPizzaStore extends PizzaStore {
    @Override
    protected Pizza createPizza(String type) {
        if (type.equals("cheese")) {
            return new ChicagoStyleCheesePizza();
        }
        return null;
    }
}
```

ChicagoStyleCheesePizza.java

```
package sadp4;
public class ChicagoStyleCheesePizza extends Pizza {
    public ChicagoStyleCheesePizza() {
        name = "Chicago Style Deep Dish Cheese Pizza";
    }
    @Override
    public void cut() {
        System.out.println("Cutting pizza into square slices");
    }
}
```

NyPizzaStore.java

```
package sadp4;
public class NyPizzaStore extends PizzaStore {
    @Override
    protected Pizza createPizza(String type) {
        if (type.equals("cheese")) {
            return new NyStyleCheesePizza();
        }
        return null;
    }
}
```

NyStyleCheesePizza.java

```
package sadp4;
```

```
public class NyStyleCheesePizza extends Pizza {
   public NyStyleCheesePizza() {
      name = "New York Style Cheese Pizza";
   }
}
```

Pizza.java

```
package sadp4;
public abstract class Pizza {
   String name;
   public void prepare() {
      System.out.println("Preparing " + name);
   }
   public void bake() {
      System.out.println("Baking " + name);
   }
   public void cut() {
      System.out.println("Cutting " + name);
   }
   public void box() {
      System.out.println("Boxing " + name);
   }
   public String getName() {
      return name;
   }
}
```

PizzaStore.java

```
package sadp4;
public abstract class PizzaStore {
   public Pizza orderPizza(String type) {
      Pizza pizza = createPizza(type);
      pizza.prepare();
      pizza.bake();
```

```
        pizza.cut();
        pizza.box();
        return pizza;
    }
    protected abstract Pizza createPizza(String type);
}
```

Main.java

```
package sadp4;
public class Main {
    public static void main(String[] args) {
        PizzaStore nyStore = new NyPizzaStore();
        PizzaStore chicagoStore = new ChicagoPizzaStore();
        Pizza pizza1 = nyStore.orderPizza("cheese");
        System.out.println("Ordered a " + pizza1.getName() + "\n");

        Pizza pizza2 = chicagoStore.orderPizza("cheese");
        System.out.println("Ordered a " + pizza2.getName() + "\n");
    }
}
```

20. Write a Java Program to implement I/O Decorator for converting uppercase letters to lower case letters.

Main.java

```
package sadp_practicals;
import java.io.*;
public class Main {
    public static void main(String[] args) throws IOException {
        int c;
        try {
            InputStream in = new LowerCaseInputStream(
                    new BufferedInputStream(
                            new FileInputStream("test.txt")));
            while((c = in.read()) >= 0) {
              System.out.print((char)c);
            }
            in.close();
          } catch (IOException e) {
            e.printStackTrace();
          }
        }
      }
```

LowerCaseInputStream.java

```
package sadp_practicals;
import java.io.*;
class LowerCaseInputStream extends FilterInputStream {
  public LowerCaseInputStream(InputStream in) {
    super(in);
  }

  public int read() throws IOException {
    int c = super.read();
    return (c == -1 ? c : Character.toLowerCase((char)c));
  }

  public int read(byte[] b, int offset, int len) throws IOException {
    int result = super.read(b, offset, len);
    for (int i = offset; i < offset + result; i++) {
      b[i] = (byte)Character.toLowerCase((char)b[i]);
    }
    return result;
  }
}
```

test.txt

```
INDIA IS MY COUNTRY
```