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

```
package adapterPattern;

class WildTurkey implements Turkey {
    public void gobble() {
        System.out.println("Gobble gobble");
    }

    public void fly() {
        System.out.println("I'm flying a short distance");
    }
}


class TurkeyAdapter implements Duck {
    Turkey turkey;

    public TurkeyAdapter(Turkey turkey) {
        this.turkey = turkey;
    }

    public void quack() {
        turkey.gobble();
    }

    public void fly() {
        for(int i=0; i < 5; i++) {
            turkey.fly();
        }
    }
}


class MallardDuck implements Duck {
    public void quack() {
        System.out.println("Quack");
    }

    public void fly() {
        System.out.println("I'm flying");
    }
}


interface Turkey {  //Turkeys don't quack
    public void gobble();
    public void fly();
```

```
}

interface Duck {
    public void quack();
    public void fly();
}


public class TurkeyTestDrive {
    public static void main(String[] args) {
        MallardDuck duck = new MallardDuck();
        WildTurkey turkey = new WildTurkey();
        Duck turkeyAdapter = new TurkeyAdapter(turkey);

        System.out.println("\nThe Turkey says ...");
        turkey.gobble();
        turkey.fly();

        System.out.println("\nThe Duck says ...");
        testDuck(duck);

        System.out.println("\nThe TurkeyAdapter says ...");
        testDuck(turkeyAdapter);
    }

    static void testDuck(Duck duck) {
        duck.quack();
        duck.fly();

    }
}
```

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

```
package car;

class BasicCar implements Car {
    @Override
    public void assemble() {
        System.out.println("Basic car assembled");
    }
}

interface Car {
    void assemble();

}
```

```
abstract class CarDecorator implements Car {
    protected Car car;

    public CarDecorator(Car car) {
        this.car = car;
    }

    public void assemble() {
        car.assemble();
    }
}

class LuxuryCar extends CarDecorator {
    public LuxuryCar(Car car) {
        super(car);
    }

    @Override
    public void assemble() {
        super.assemble();
        System.out.println("Added features for a luxury car");
    }
}

class SportsCar extends CarDecorator {
    public SportsCar(Car car) {
        super(car);
    }

    @Override
    public void assemble() {
        super.assemble();
        System.out.println("Added features for a sports car");
    }
}

public class DecoratorPatternExample {
    public static void main(String[] args) {
        // Creating a basic car
        Car basicCar = new BasicCar();
        basicCar.assemble();
        System.out.println("---");

        // Decorating basic car with SportsCar features
        Car sportsCar = new SportsCar(basicCar);
        sportsCar.assemble();
        System.out.println("---");

        // Decorating basic car with LuxuryCar features
        Car luxuryCar = new LuxuryCar(basicCar);
        luxuryCar.assemble();
```

```
        System.out.println("---");

        // Decorating basic car with SportsCar and then LuxuryCar features
        Car sportsLuxuryCar = new LuxuryCar(new SportsCar(basicCar));
        sportsLuxuryCar.assemble();
    }
}
```

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

```
package ceilingFan;

class CeilingFan {
    private String state;

    public CeilingFan() {
        state = "OFF";
    }

    public void turnOn() {
        state = "ON";
        System.out.println("Ceiling Fan is turned ON.");
    }

    public void turnOff() {
        state = "OFF";
        System.out.println("Ceiling Fan is turned OFF.");
    }

    public String getState() {
        return state;
    }
}

class CeilingFanOffCommand implements Command {
    private CeilingFan fan;

    public CeilingFanOffCommand(CeilingFan fan) {
        this.fan = fan;
    }

    @Override
    public void execute() {
        fan.turnOff();
    }

    @Override
    public void undo() {
        fan.turnOn();
    }
```

```
}

class CeilingFanOnCommand implements Command {
    private CeilingFan fan;

    public CeilingFanOnCommand(CeilingFan fan) {
        this.fan = fan;
    }

    @Override
    public void execute() {
        fan.turnOn();
    }

    @Override
    public void undo() {
        fan.turnOff();
    }
}

interface Command {
    void execute();
    void undo();
}

class RemoteControl {
    private Command command;

    public void setCommand(Command command) {
        this.command = command;
    }
   public void pressButton() {
        command.execute();
    }
    public void pressUndo() {
        command.undo();
    }
}

public class CeilingFanTest {

    public static void main(String[] args) {
        CeilingFan fan = new CeilingFan();

        Command fanOnCommand = new CeilingFanOnCommand(fan);
        Command fanOffCommand = new CeilingFanOffCommand(fan);

        RemoteControl remoteControl = new RemoteControl();
        remoteControl.setCommand(fanOnCommand);
        remoteControl.pressButton(); // Turns the fan on
        remoteControl.pressUndo();   // Undoes the previous command (turns the fan
```

```
off)

        remoteControl.setCommand(fanOffCommand);
        remoteControl.pressButton(); // Turns the fan off
        remoteControl.pressUndo();   // Undoes the previous command (turns the fan
on)
    }
}
```

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

```
package changeCase;
import java.util.Scanner;
public class ChangeCase {
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter Any String :");
        String str1=sc.nextLine();
        StringBuffer newStr=new StringBuffer(str1);
        System.out.println("\nString before case conversion : ");
        System.out.println(newStr);

        for(int i = 0; i < str1.length(); i++)
        {
            /*
             * //Checks for lower case character
if(Character.isLowerCase(str1.charAt(i))) {
             * //Convert it into upper case using toUpperCase() function
newStr.setCharAt(i,
             * Character.toUpperCase(str1.charAt(i))); } //Checks for upper case
character
             * else
             */
            if(Character.isUpperCase(str1.charAt(i)))
            {
                //Convert it into upper case using toLowerCase() function
                newStr.setCharAt(i, Character.toLowerCase(str1.charAt(i)));
            }
        }
        System.out.println("\n String after case conversion : ");
        System.out.println(newStr);
    }
}
```

Write a Java Program to implement Iterator Pattern for Designing Menu like
Breakfast,
Lunch or Dinner Menu

```
package dinnerMenu;

import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Iterator;

class CoffeeMenu implements Menu {
    Hashtable menuItems = new Hashtable();

    public CoffeeMenu() {
        addItem("Mocha","Han Meimei order on couple of Mocha",false,3.01);
    }

    private void addItem(String s, String s1, boolean b, double v) {
        MenuItem menuItem = new MenuItem(s,s1,b,v);
        menuItems.put(menuItem.getName(),menuItem);
    }

    @Override
    public Iterator createIterator() {
        return menuItems.values().iterator();
    }
}
class DinerMenuIterator implements Iterator {
    MenuItem[] list;
    int position = 0;

    public DinerMenuIterator(MenuItem[] list) {
        this.list = list;
    }

    @Override
    public boolean hasNext() {
        if(position>=list.length || list[position] == null){
            return false;
        }else{
            return true;
        }
    }

    @Override
    public Object next() {
        MenuItem menuItem = list[position];
        position = position + 1;
        return menuItem;
    }

    @Override
    public void remove() {
        if(position <=0){
```

```
            throw new IllegalStateException("now  you can not remove an item");
        }
        if(list[position] != null){
            for(int i=position-1;i<(list.length-1);i++){
                list[i] = list[i+1];
            }
            list[list.length-1]=null;
        }
    }
}

class DinnerMenu implements Menu {
    private static final int MAX_SIZE = 6;
    int numOfItems = 0;
    MenuItem[] menuItems;

    public DinnerMenu() {
        menuItems = new MenuItem[MAX_SIZE];
        addItem("Vegetarian BLT","Bacon with tomato",true,2.99);
        addItem("Hot dog","With onions and cheese",false,3.05);
    }

    private void addItem(String s, String s1, boolean b, double v) {
        MenuItem menuItem = new MenuItem(s,s1,b,v);
        if(numOfItems >= MAX_SIZE){
            System.err.println("sorry,menu is full!");
        }else{
            menuItems[numOfItems]=menuItem;
            numOfItems = numOfItems + 1;
        }
    }

    @Override
    public Iterator createIterator() {
        return new DinerMenuIterator(menuItems);
    }
}

interface Menu {
    public Iterator createIterator();
}

class MenuItem {
    private String name;
    private String desc;
    private boolean vegetarian;
    private double price;

    public MenuItem(String name, String desc, boolean vegetarian, double price) {
        this.name = name;
        this.desc = desc;
```

```
        this.vegetarian = vegetarian;
        this.price = price;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDesc() {
        return desc;
    }

    public void setDesc(String desc) {
        this.desc = desc;
    }

    public boolean isVegetarian() {
        return vegetarian;
    }

    public void setVegetarian(boolean vegetarian) {
        this.vegetarian = vegetarian;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }
}

class PancakeHouseMenu implements Menu {
    ArrayList menuItems;
    public PancakeHouseMenu() {
        menuItems = new ArrayList();
        addItem("kobe's pancake breakfast","pancakes with eggs",false,2.99);
        addItem("lilei's pancake breakfast", "pancakes with toast", false, 3.59);
    }
    public void addItem(String s, String s1, boolean b, double v) {
        MenuItem menuItem = new MenuItem(s,s1,b,v);
        menuItems.add(menuItem);
    }
    public Iterator createIterator(){
        return menuItems.iterator();
    }
```

```
}
class Waitress {
    Menu pancakeHouseMenu;
    Menu dinnerMenu;
    Menu coffeeMenu;

    public Waitress(Menu pancakeHouseMenu, Menu dinnerMenu, Menu coffeeMenu) {
        this.pancakeHouseMenu = pancakeHouseMenu;
        this.dinnerMenu = dinnerMenu;
        this.coffeeMenu = coffeeMenu;
    }

    public void printMenu(){
        Iterator pancakeHouseIterator = pancakeHouseMenu.createIterator();
        Iterator dinnerIterator = dinnerMenu.createIterator();
        Iterator coffeeIterator = coffeeMenu.createIterator();
        System.out.println("Menu\n=====Breakfast==start======");
        printMenu(pancakeHouseIterator);
        System.out.println("Menu\n=====Breakfast===end=====");
        System.out.println("Menu\n=====Lunch==start======");
        printMenu(dinnerIterator);
        System.out.println("Menu\n=====Lunch===end=====");
        System.out.println("Menu\n=====Coffee==start======");
        printMenu(coffeeIterator);
        System.out.println("Menu\n=====Coffee===end=====");
    }

    private void printMenu(Iterator iterator){
        while (iterator.hasNext()){
            MenuItem menuItem = (MenuItem)iterator.next();
            System.out.println(menuItem.getName()+", ");
            System.out.println(menuItem.getPrice()+", ");
            System.out.println(menuItem.getDesc());
        }
    }
}

public class MenuTest {
    public static void main(String[] args) {
        PancakeHouseMenu pancakeHouseMenu = new PancakeHouseMenu();
        DinnerMenu dinnerMenu = new DinnerMenu();
        CoffeeMenu coffeeMenu = new CoffeeMenu();

        Waitress waitress = new Waitress(pancakeHouseMenu,dinnerMenu,coffeeMenu);
        waitress.printMenu();
    }
}
```

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

```
package gumBall;

interface State {
    public void insertQuarter();
    public void ejectQuarter();
    public void turnCrank();
    public void dispense();
    public void refill();
}

class SoldState implements State {

    GumballMachine gumballMachine;

    public SoldState(GumballMachine gumballMachine) {
        this.gumballMachine = gumballMachine;
    }

    public void insertQuarter() {
        System.out.println("Please wait, we're already giving you a gumball");
    }

    public void ejectQuarter() {
        System.out.println("Sorry, you already turned the crank");
    }

    public void turnCrank() {
        System.out.println("Turning twice doesn't get you another gumball!");
    }
    public void dispense() {
        gumballMachine.releaseBall();
        if (gumballMachine.getCount() > 0) {
            gumballMachine.setState(gumballMachine.getNoQuarterState());
        } else {
            System.out.println("Oops, out of gumballs!");
            gumballMachine.setState(gumballMachine.getSoldOutState());
        }
    }

    public void refill() { }

    public String toString() {
        return "dispensing a gumball";
    }
}

class SoldOutState implements State {
    GumballMachine gumballMachine;
```

```
    public SoldOutState(GumballMachine gumballMachine) {
        this.gumballMachine = gumballMachine;
    }

    public void insertQuarter() {
        System.out.println("You can't insert a quarter, the machine is sold out");
    }

    public void ejectQuarter() {
        System.out.println("You can't eject, you haven't inserted a quarter yet");
    }

    public void turnCrank() {
        System.out.println("You turned, but there are no gumballs");
    }

    public void dispense() {
        System.out.println("No gumball dispensed");
    }

    public void refill() {
        gumballMachine.setState(gumballMachine.getNoQuarterState());
    }

    public String toString() {
        return "sold out";
    }
}

class NoQuarterState implements State {
    GumballMachine gumballMachine;

    public NoQuarterState(GumballMachine gumballMachine) {
        this.gumballMachine = gumballMachine;
    }

    public void insertQuarter() {
        System.out.println("You inserted a quarter");
        gumballMachine.setState(gumballMachine.getHasQuarterState());
    }

    public void ejectQuarter() {
        System.out.println("You haven't inserted a quarter");
    }

    public void turnCrank() {
        System.out.println("You turned, but there's no quarter");
     }

    public void dispense() {
```

```
        System.out.println("You need to pay first");
    }

    public void refill() { }

    public String toString() {
        return "waiting for quarter";
    }
}

class HasQuarterState implements State {
    GumballMachine gumballMachine;

    public HasQuarterState(GumballMachine gumballMachine) {
        this.gumballMachine = gumballMachine;
    }

    public void insertQuarter() {
        System.out.println("You can't insert another quarter");
    }

    public void ejectQuarter() {
        System.out.println("Quarter returned");
        gumballMachine.setState(gumballMachine.getNoQuarterState());
    }

    public void turnCrank() {
        System.out.println("You turned...");
        gumballMachine.setState(gumballMachine.getSoldState());
    }

    public void dispense() {
        System.out.println("No gumball dispensed");
    }

    public void refill() { }

    public String toString() {
        return "waiting for turn of crank";
    }
}

class GumballMachine {
    State soldOutState;
    State noQuarterState;
    State hasQuarterState;
    State soldState;
    State state;
    int count = 0;
    public GumballMachine(int numberGumballs) {
        soldOutState = new SoldOutState(this);
```

```
        noQuarterState = new NoQuarterState(this);
        hasQuarterState = new HasQuarterState(this);
        soldState = new SoldState(this);

        this.count = numberGumballs;
        if (numberGumballs > 0) {
            state = noQuarterState;
        } else {
            state = soldOutState;
        }
    }

    public void insertQuarter() {
        state.insertQuarter();
    }

    public void ejectQuarter() {
        state.ejectQuarter();
    }

    public void turnCrank() {
        state.turnCrank();
        state.dispense();
    }

    void releaseBall() {
        System.out.println("A gumball comes rolling out the slot...");
        if (count != 0) {
            count = count - 1;
        }
    }

    int getCount() {
        return count;
    }

    void refill(int count) {
        this.count += count;
        System.out.println("The gumball machine was just refilled; it's new count
is: " + this.count);
        state.refill();
    }

    void setState(State state) {
        this.state = state;
    }
    public State getState() {
        return state;
    }

    public State getSoldOutState() {
```

```
        return soldOutState;
    }

    public State getNoQuarterState() {
        return noQuarterState;
    }

    public State getHasQuarterState() {
        return hasQuarterState;
    }

    public State getSoldState() {
        return soldState;
    }

    public String toString() {
        StringBuffer result = new StringBuffer();
        result.append("\nMighty Gumball, Inc.");
        result.append("\nJava-enabled Standing Gumball Model");
        result.append("\nInventory: " + count + " gumball");
        if (count != 1) {
            result.append("s");
        }
        result.append("\n");
        result.append("Machine is " + state + "\n");
        return result.toString();
    }
}

public class GumballMachineTestDrive {

    public static void main(String[] args) {
        GumballMachine gumballMachine = new GumballMachine(2);

        System.out.println(gumballMachine);

        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();

        System.out.println(gumballMachine);

        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();
        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();

        gumballMachine.refill(5);
        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();

        System.out.println(gumballMachine);
```

```
    }
}
```

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

```
package heartBeat;

class Heart implements HeartModel {
    private int heartRate;

    public Heart(int heartRate) {
        this.heartRate = heartRate;
    }

    public int getHeartRate() {
        return heartRate;
    }
}

interface HeartModel {
    int getHeartRate();
}

interface BeatModel {
    void playBeat();

}
class HeartToBeatAdapter implements BeatModel {
    private HeartModel heartModel;

    public HeartToBeatAdapter(HeartModel heartModel) {
        this.heartModel = heartModel;
    }

    public void playBeat() {
        int heartRate = heartModel.getHeartRate();
        System.out.println("Playing beat with heart rate: " + heartRate);
    }
}

public class AdapterPatternExample {
    public static void main(String[] args) {
        // Create a Heart object
        Heart heart = new Heart(80);

        // Create an adapter to convert Heart Model to Beat Model
        BeatModel beatModel = new HeartToBeatAdapter(heart);

        // Use the adapter to play the beat
        beatModel.playBeat();
```

```
    }
}
```

Write a Java Program to implement Facade Design Pattern for HomeTheater

```
package homeTheator;

class Amplifier {
     String description;
     Tuner tuner;
     DvdPlayer dvd;
     CdPlayer cd;

     public Amplifier(String description) {
      this.description = description;
     }
     public void on() {
      System.out.println(description + " on");
     }
     public void off() {
      System.out.println(description + " off");
     }
     public void setStereoSound() {
      System.out.println(description + " stereo mode on");
     }
     public void setSurroundSound() {
      System.out.println(description + " surround sound on (5 speakers, 1
subwoofer)");
     }
     public void setVolume(int level) {
      System.out.println(description + " setting volume to " + level);
     }
     public void setTuner(Tuner tuner) {
      System.out.println(description + " setting tuner to " + dvd);
      this.tuner = tuner;
     }
     public void setDvd(DvdPlayer dvd) {
      System.out.println(description + " setting DVD player to " + dvd);
      this.dvd = dvd;
     }
     public void setCd(CdPlayer cd) {
      System.out.println(description + " setting CD player to " + cd);
      this.cd = cd;
     }
    public String toString() {
      return description;
     }
    }
```

```
class CdPlayer {
     String description;
     int currentTrack;
     Amplifier amplifier;
     String title;

     public CdPlayer(String description, Amplifier amplifier) {
      this.description = description;
      this.amplifier = amplifier;
     }
     public void on() {
      System.out.println(description + " on");
     }
     public void off() {
      System.out.println(description + " off");
     }
     public void eject() {
      title = null;
      System.out.println(description + " eject");
     }
     public void play(String title) {
      this.title = title;
      currentTrack = 0;
      System.out.println(description + " playing \"" + title + "\"");
     }
     public void play(int track) {
      if (title == null) {
       System.out.println(description + " can't play track " + currentTrack +
         ", no cd inserted");
      } else {
       currentTrack = track;
       System.out.println(description + " playing track " + currentTrack);
      }
     }

     public void stop() {
      currentTrack = 0;
      System.out.println(description + " stopped");
     }

     public void pause() {
      System.out.println(description + " paused \"" + title + "\"");
     }

     public String toString() {
      return description;
     }
    }
```

```
class DvdPlayer {
     String description;
     int currentTrack;
     Amplifier amplifier;
     String movie;

     public DvdPlayer(String description, Amplifier amplifier) {
      this.description = description;
      this.amplifier = amplifier;
     }

     public void on() {
      System.out.println(description + " on");
     }

     public void off() {
      System.out.println(description + " off");
     }

            public void eject() {
      movie = null;
                     System.out.println(description + " eject");
            }

     public void play(String movie) {
      this.movie = movie;
      currentTrack = 0;
      System.out.println(description + " playing \"" + movie + "\"");
     }

     public void play(int track) {
      if (movie == null) {
       System.out.println(description + " can't play track " + track + " no dvd
inserted");
      } else {
       currentTrack = track;
       System.out.println(description + " playing track " + currentTrack + " of
\"" + movie + "\"");
      }
     }

     public void stop() {
      currentTrack = 0;
      System.out.println(description + " stopped \"" + movie + "\"");
     }

     public void pause() {
      System.out.println(description + " paused \"" + movie + "\"");
     }
```

```
    public void setTwoChannelAudio() {
     System.out.println(description + " set two channel audio");
    }

    public void setSurroundAudio() {
     System.out.println(description + " set surround audio");
    }

    public String toString() {
     return description;
    }
   }



class Projector {
    String description;
    DvdPlayer dvdPlayer;

    public Projector(String description, DvdPlayer dvdPlayer) {
     this.description = description;
     this.dvdPlayer = dvdPlayer;
    }

    public void on() {
     System.out.println(description + " on");
    }

    public void off() {
     System.out.println(description + " off");
    }

    public void wideScreenMode() {
     System.out.println(description + " in widescreen mode (16x9 aspect ratio)");
    }

    public void tvMode() {
     System.out.println(description + " in tv mode (4x3 aspect ratio)");
    }

           public String toString() {
                   return description;
           }
   }



class TheaterLights {
    String description;

    public TheaterLights(String description) {
```

```
      this.description = description;
     }

     public void on() {
      System.out.println(description + " on");
     }

     public void off() {
      System.out.println(description + " off");
     }

     public void dim(int level) {
      System.out.println(description + " dimming to " + level  + "%");
     }

            public String toString() {
                    return description;
            }
    }



class Screen {
     String description;

     public Screen(String description) {
      this.description = description;
     }

     public void up() {
      System.out.println(description + " going up");
     }

     public void down() {
      System.out.println(description + " going down");
     }
            public String toString() {
                    return description;
            }
    }


class PopcornPopper {
     String description;

     public PopcornPopper(String description) {
      this.description = description;
     }

     public void on() {
      System.out.println(description + " on");
```

```
    }

    public void off() {
     System.out.println(description + " off");
    }

    public void pop() {
     System.out.println(description + " popping popcorn!");
    }
           public String toString() {
                   return description;
           }
   }



class HomeTheaterFacade {
    Amplifier amp;
    Tuner tuner;
    DvdPlayer dvd;
    CdPlayer cd;
    Projector projector;
    TheaterLights lights;
    Screen screen;
    PopcornPopper popper;

    public HomeTheaterFacade(Amplifier amp,
        Tuner tuner,
        DvdPlayer dvd,
        CdPlayer cd,
        Projector projector,
        Screen screen,
        TheaterLights lights,
        PopcornPopper popper) {

     this.amp = amp;
     this.tuner = tuner;
     this.dvd = dvd;
     this.cd = cd;
     this.projector = projector;
     this.screen = screen;
     this.lights = lights;
     this.popper = popper;
    }

    public void watchMovie(String movie) {
     System.out.println("Get ready to watch a movie...");
     popper.on();
     popper.pop();
     lights.dim(10);
     screen.down();
```

```
 projector.on();
 projector.wideScreenMode();
 amp.on();
 amp.setDvd(dvd);
 amp.setSurroundSound();
 amp.setVolume(5);
 dvd.on();
 dvd.play(movie);
}


public void endMovie() {
 System.out.println("Shutting movie theater down...");
 popper.off();
 lights.on();
 screen.up();
 projector.off();
 amp.off();
 dvd.stop();
 dvd.eject();
 dvd.off();
}

public void listenToCd(String cdTitle) {
 System.out.println("Get ready for an audiopile experence...");
 lights.on();
 amp.on();
 amp.setVolume(5);
 amp.setCd(cd);
 amp.setStereoSound();
 cd.on();
 cd.play(cdTitle);
}
public void endCd() {
 System.out.println("Shutting down CD...");
 amp.off();
 amp.setCd(cd);
 cd.eject();
 cd.off();
}
public void listenToRadio(double frequency) {
 System.out.println("Tuning in the airwaves...");
 tuner.on();
 tuner.setFrequency(frequency);
 amp.on();
 amp.setVolume(5);
 amp.setTuner(tuner);
}
public void endRadio() {
 System.out.println("Shutting down the tuner...");
 tuner.off();
```

```
      amp.off();
     }
    }


public class HomeTheaterTestDrive {
     public static void main(String[] args) {
      Amplifier amp = new Amplifier("Top-O-Line Amplifier");
      Tuner tuner = new Tuner("Top-O-Line AM/FM Tuner", amp);
      DvdPlayer dvd = new DvdPlayer("Top-O-Line DVD Player", amp);
      CdPlayer cd = new CdPlayer("Top-O-Line CD Player", amp);
      Projector projector = new Projector("Top-O-Line Projector", dvd);
      TheaterLights lights = new TheaterLights("Theater Ceiling Lights");
      Screen screen = new Screen("Theater Screen");
      PopcornPopper popper = new PopcornPopper("Popcorn Popper");

      HomeTheaterFacade homeTheater =
        new HomeTheaterFacade(amp, tuner, dvd, cd,
          projector, screen, lights, popper);

      homeTheater.watchMovie("Raiders of the Lost Ark");
      homeTheater.endMovie();
     }
    }


class Tuner {
        String description;
        Amplifier amplifier;
        double frequency;
        public Tuner(String description, Amplifier amplifier) {
            this.description = description;
        }
        public void on() {
            System.out.println(description + " on");
        }
        public void off() {
            System.out.println(description + " off");
        }
        public void setFrequency(double frequency) {
            System.out.println(description + " setting frequency to " +
frequency);
            this.frequency = frequency;
        }
        public void setAm() {
            System.out.println(description + " setting AM mode");
        }

        public void setFm() {
            System.out.println(description + " setting FM mode");
        }
```

```
        public String toString() {
            return description;
        }
    }
```

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

```
package mobileCharger;


class Voltage
{
    private int voltage;
    public Voltage(int v)
    {
        this.voltage = v;
    }
    public int getVolts()
    {
        return voltage;
    }
    public void setVolts(int voltage)
    {
        this.voltage = voltage;
    }
}

interface SocketAdapter
{
     public Voltage get120Voltage();
     public Voltage get12Voltage();
     public Voltage get3VVoltage();
}
////////////////////////////////////////////
class Socket
{
    public Voltage getVoltage()
    {
        return new Voltage(120); //In India 240 is the default voltage
    }
}

////////////////////////////////////////////////////
class SocketAdapterImpl extends Socket implements SocketAdapter
{

    //Using Composition for adapter pattern
```

```
    private Socket sock = new Socket();
    private Voltage convertVolt(Voltage v, int i)
    {
        return new Voltage(v.getVolts() / i);
    }
    @Override
    public Voltage get120Voltage()
    {
        return sock.getVoltage();
    }
    @Override
    public Voltage get12Voltage()
    {
        Voltage v = sock.getVoltage();
        return convertVolt(v, 20);
    }
    @Override
    public Voltage get3VVoltage()
    {
        Voltage v = sock.getVoltage();
        return convertVolt(v, 80);
    }
}
////////////////////////////////////////////////////

class AdapterEx {

    public static void main(String[] args)
    {
        SocketAdapter socketAdapter = new SocketAdapterImpl();
        Voltage voltage12 = socketAdapter.get12Voltage();
        System.out.println("Socket : "+voltage12.getVolts());

        Voltage voltage3 = socketAdapter.get3VVoltage();
        System.out.println(voltage3.getVolts());

        Voltage voltage120 = socketAdapter.get120Voltage();
        System.out.println(voltage120.getVolts());
    }
}
```

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

```
package numberConversion;

import java.util.ArrayList;
import java.util.List;

class Subject {
```

```
   private List<Observer> observers = new ArrayList<Observer>();
   private int state;

   public int getState() {
      return state;
   }

   public void setState(int state) {
      this.state = state;
      notifyAllObservers();
   }

   public void attach(Observer observer){
      observers.add(observer);
   }

   public void notifyAllObservers(){
      for (Observer observer : observers) {
         observer.update();
      }
   }
}
//////////////////////////////////////
abstract class Observer {
   protected Subject subject;
   public abstract void update();
}
//////////////////////////////////////////
class BinaryObserver extends Observer{

   public BinaryObserver(Subject subject){
      this.subject = subject;
      this.subject.attach(this);
   }

   @Override
   public void update() {
      System.out.println( "Binary String: " + Integer.toBinaryString(
subject.getState() ) );
   }
}
/////////////////////////////////////////////////////
class OctalObserver extends Observer{

   public OctalObserver(Subject subject){
      this.subject = subject;
      this.subject.attach(this);
   }

   @Override
```

```
   public void update() {
     System.out.println( "Octal String: " + Integer.toOctalString(
subject.getState() ) );
   }
}
////////////////////////////////////////////////////////
class HexaObserver extends Observer{

   public HexaObserver(Subject subject){
      this.subject = subject;
      this.subject.attach(this);
   }

   @Override
   public void update() {
      System.out.println( "Hex String: " + Integer.toHexString( subject.getState()
).toUpperCase() );
   }
}
////////////////////////////////////////////////////////////////
public class ObserverPatternDemo {
   public static void main(String[] args) {
      Subject subject = new Subject();

      new HexaObserver(subject);
      new OctalObserver(subject);
      new BinaryObserver(subject);

      System.out.println("First state change: 15");
      subject.setState(15);
      System.out.println("Second state change: 10");
      subject.setState(10);
   }
}
```

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

```
package pizza;

import java.util.ArrayList;

class ChicagoStyleCheesePizza extends Pizza
{
    public ChicagoStyleCheesePizza()
    {
        name = "Chicago Style Deep Dish Cheese Pizza";
        dough = "Extra Thick Crust Dough";
```

```
        sauce = "Plum Tomato Sauce";
        toppings.add("Shredded Mozzarella Cheese");
    }

    void cut()
    {
        System.out.println("Cutting the pizza into square slices");
    }
}

class ChicagoStyleClamPizza extends Pizza
{
    public ChicagoStyleClamPizza()
    {
        name = "Chicago Style Clam Pizza";
        dough = "Extra Thick Crust Dough";
        sauce = "Plum Tomato Sauce";
        toppings.add("Shredded Mozzarella Cheese");
        toppings.add("Frozen Clams from Chesapeake Bay");
    }

    void cut()
    {
        System.out.println("Cutting the pizza into square slices");
    }
}

class ChicagoStylePepperoniPizza extends Pizza
{
    public ChicagoStylePepperoniPizza()
    {
        name = "Chicago Style Pepperoni Pizza";
        dough = "Extra Thick Crust Dough";
        sauce = "Plum Tomato Sauce";
        toppings.add("Shredded Mozzarella Cheese");
        toppings.add("Black Olives");
        toppings.add("Spinach");
        toppings.add("Eggplant");
        toppings.add("Sliced Pepperoni");
    }

    void cut()
    {
        System.out.println("Cutting the pizza into square slices");
    }
}

class ChicagoStyleVeggiePizza extends Pizza
{
    public ChicagoStyleVeggiePizza()
    {
```

```
        name = "Chicago Deep Dish Veggie Pizza";
        dough = "Extra Thick Crust Dough";
        sauce = "Plum Tomato Sauce";
        toppings.add("Shredded Mozzarella Cheese");
        toppings.add("Black Olives");
        toppings.add("Spinach");
        toppings.add("Eggplant");
    }

    void cut()
    {
    System.out.println("Cutting the pizza into square slices");
    }
}

class ChicagoPizzaStore extends PizzaStore
{
    Pizza createPizza(String item)
    {
        if (item.equals("cheese"))
       {
            return new ChicagoStyleCheesePizza();
       }

        else if (item.equals("veggie"))
       {
            return new ChicagoStyleVeggiePizza();
        }

        else if (item.equals("clam"))
       {
            return new ChicagoStyleClamPizza();
       }

        else if (item.equals("pepperoni"))
       {
            return new ChicagoStylePepperoniPizza();
       }
        else return null;
    }
}

class DependentPizzaStore
{
    public Pizza createPizza(String style, String type)
   {
      Pizza pizza = null;
        if (style.equals("NY"))
            {
                if (type.equals("cheese"))
                {
```

```
                    pizza = new NYStyleCheesePizza();
                }
            else if (type.equals("veggie"))
                    {
                        pizza = new NYStyleVeggiePizza();
                    }
                else if (type.equals("clam"))
                    {
                        pizza = new NYStyleClamPizza();
                    }
                else if (type.equals("pepperoni"))
                {
                    pizza = new NYStylePepperoniPizza();
                }
          }
        else if (style.equals("Chicago"))
           {
             if (type.equals("cheese"))
                {
                    pizza = new ChicagoStyleCheesePizza();
                }
             else if (type.equals("veggie"))
                {
                    pizza = new ChicagoStyleVeggiePizza();
                }
             else if (type.equals("clam"))
                {
                    pizza = new ChicagoStyleClamPizza();
                }
             else if (type.equals("pepperoni"))
                {
                    pizza = new ChicagoStylePepperoniPizza();
                }
          }
         else
             {
             System.out.println("Error: invalid type of pizza"); return null;
             }
             pizza.prepare();
             pizza.bake();
             pizza.cut();
             pizza.box();
             return pizza;
    }
}

class NYPizzaStore extends PizzaStore
{
        Pizza createPizza(String item)
             {
                 if (item.equals("cheese"))
```

```
                    {
                        return new NYStyleCheesePizza();
                    }
                else if (item.equals("veggie"))
                    {
                    return new NYStyleVeggiePizza();
                    }
                else if (item.equals("clam"))
                    {
                return new NYStyleClamPizza();
                    }
                else if (item.equals("pepperoni"))
                    {
                return new NYStylePepperoniPizza();
                    }
                else return null;
            }
    }

class NYStyleCheesePizza extends Pizza
{
    public NYStyleCheesePizza()
    {
        name = "NY Style Sauce and Cheese Pizza";
        dough = "Thin Crust Dough";
        sauce = "Marinara Sauce";
        toppings.add("Grated Reggiano Cheese");
    }
}

class NYStyleClamPizza extends Pizza
{
    public NYStyleClamPizza()
    {
        name = "NY Style Clam Pizza";
        dough = "Thin Crust Dough";
        sauce = "Marinara Sauce";
        toppings.add("Grated Reggiano Cheese");
        toppings.add("Fresh Clams from Long Island Sound");
    }
}

class NYStylePepperoniPizza extends Pizza
{
    public NYStylePepperoniPizza()
    {
        name = "NY Style Pepperoni Pizza";
        dough = "Thin Crust Dough";
        sauce = "Marinara Sauce";
        toppings.add("Grated Reggiano Cheese");
        toppings.add("Sliced Pepperoni");
```

```
        toppings.add("Garlic");
        toppings.add("Onion");
        toppings.add("Mushrooms");
        toppings.add("Red Pepper");
    }
}

class NYStyleVeggiePizza extends Pizza
{
    public NYStyleVeggiePizza()
    {
        name =  "NY Style Veggie Pizza";
        dough = "Thin Crust Dough";
        sauce = "Marinara Sauce";
        toppings.add("Grated Reggiano Cheese");
        toppings.add("Garlic");
        toppings.add("Onion");
        toppings.add("Mushrooms");
        toppings.add("Red Pepper");
    }
}

class Pizza
{
    String name;
    String dough;
    String sauce;
    ArrayList toppings = new ArrayList();
    void prepare()
    {
        System.out.println("Preparing " + name);
        System.out.println("Tossing dough...");
        System.out.println("Adding sauce...");
        System.out.println("Adding toppings: ");
        for (int i = 0; i < toppings.size(); i++)
            {
                System.out.println("   " + toppings.get(i));
            }
    }
    void bake()
    {
        System.out.println("Bake for 25 minutes at 350");
    }
    void cut()
    {
        System.out.println("Cutting the pizza into diagonal slices");
    }
    void box()
    {
        System.out.println("Place pizza in official PizzaStore box");
    }
```

```
    public String getName()
    {
        return name;
    }
    public String toString()
    {
        StringBuffer display = new StringBuffer();
        display.append("---- " + name + " ----\n");
        display.append(dough + "\n");
        display.append(sauce + "\n");
        for (int i = 0; i < toppings.size(); i++)
        {
            display.append((String )toppings.get(i) + "\n");
        }
        return display.toString();
    }
}

abstract class PizzaStore
{
    abstract Pizza createPizza(String item);
    public Pizza orderPizza(String type)
    {
        Pizza pizza = createPizza(type);
        System.out.println("\n### Making a " + pizza.getName() + " ### \n");
        pizza.prepare();
        pizza.bake();
        pizza.cut();
        pizza.box();
        return pizza;
    }
}

public class Main {
    public static void main(String[] args)
    {
        PizzaStore nyStore = new NYPizzaStore();
        PizzaStore chicagoStore = new ChicagoPizzaStore();

        Pizza pizza = nyStore.orderPizza("cheese");
        System.out.println("[] Ethan ordered a " + pizza.getName() + "\n");

        pizza = chicagoStore.orderPizza("cheese");
        System.out.println("[] Joel ordered a " + pizza.getName() + "\n");

        pizza = nyStore.orderPizza("clam");
        System.out.println("[] Ethan ordered a " + pizza.getName() + "\n");

        pizza = chicagoStore.orderPizza("clam");
        System.out.println("[] Joel ordered a " + pizza.getName() + "\n");
```

```
        pizza = nyStore.orderPizza("pepperoni");
        System.out.println("[] Ethan ordered a " + pizza.getName() + "\n");

        pizza = chicagoStore.orderPizza("pepperoni");
        System.out.println("[] Joel ordered a " + pizza.getName() + "\n");

        pizza = nyStore.orderPizza("veggie");
        System.out.println("[] Ethan ordered a " + pizza.getName() + "\n");

        pizza = chicagoStore.orderPizza("veggie");
        System.out.println("[] Joel ordered a " + pizza.getName() + "\n");
    }

}
```

Write a Java Program to implement command pattern to test Remote Control

```
package remoteControll;
import java.util.*;

interface Command {
    public void execute();
}

class Light {
    public void on()
    {
        System.out.println("Light is on");
    }
    public void off()
    {
        System.out.println("Light is off");
    }
}

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

class LightOnCommand implements Command
{
    Light light;
```

```
    // The constructor is passed the light it
    // is going to control.
    public LightOnCommand(Light light)
    {
    this.light = light;
    }
    public void execute()
    {
    light.on();
    }
}

class SimpleRemoteControl
{
    Command slot; // only one button

    public SimpleRemoteControl()
    {
    }
    public void setCommand(Command command)
    {
        // set the command the remote will
        // execute
        slot = command;
    }
    public void buttonWasPressed()
    {
        slot.execute();
    }
}

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("Stereo is set " +
                        "for CD input");
    }
    public void setDVD()
    {
        System.out.println("Stereo is set"+
                        " for DVD input");
    }
```

```
    public void setRadio()
    {
        System.out.println("Stereo is set" +
                        " for Radio");
    }
    public void setVolume(int volume)
    {
    // code to set the volume
    System.out.println("Stereo volume set"
                        + " to " + volume);
    }
}

class StereoOffCommand implements Command
{
    Stereo stereo;
    public StereoOffCommand(Stereo stereo)
    {
        this.stereo = stereo;
    }
    public void execute()
    {
    stereo.off();
    }
}

class StereoOnWithCDCommand implements Command
{
    Stereo stereo;
    public StereoOnWithCDCommand(Stereo stereo)
    {
        this.stereo = stereo;
    }
    public void execute()
    {
        stereo.on();
        stereo.setCD();
        stereo.setVolume(11);
    }
}

public class RemoteControlTest
{
    public static void main(String[] args)
    {
        SimpleRemoteControl remote =
                new SimpleRemoteControl();
        Light light = new Light();
        Stereo stereo = new Stereo();

        // we can change command dynamically
```

```
        remote.setCommand(new
                    LightOnCommand(light));
        remote.buttonWasPressed();
        remote.setCommand(new
                StereoOnWithCDCommand(stereo));
        remote.buttonWasPressed();
        remote.setCommand(new
                StereoOffCommand(stereo));
        remote.buttonWasPressed();
    }
}
```

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

```
package shape;

interface Shape {
       void draw();
    }

class RoundedRectangle implements Shape {
       @Override
       public void draw() {
          System.out.println("Inside RoundedRectangle::draw() method.");
       }
    }

class RoundedSquare implements Shape {
       @Override
       public void draw() {
          System.out.println("Inside RoundedSquare::draw() method.");
       }
    }


class Rectangle implements Shape {
       @Override
       public void draw() {
          System.out.println("Inside Rectangle::draw() method.");
       }
    }
    //Step 3
    //Create an Abstract class to get factories for Normal and Rounded Shape
Objects.


abstract class AbstractFactory {
       abstract Shape getShape(String shapeType) ;
    }
    //Step 4
    //Create Factory classes extending AbstractFactory to generate object of
```

concrete class based on given information.

```
class ShapeFactory extends AbstractFactory {
       @Override
       public Shape getShape(String shapeType){
          if(shapeType.equalsIgnoreCase("RECTANGLE")){
             return new Rectangle();
          }else if(shapeType.equalsIgnoreCase("SQUARE")){
             return new RoundedSquare();
          }
          return null;
       }
    }

class RoundedShapeFactory extends AbstractFactory {
       @Override
       public Shape getShape(String shapeType){
          if(shapeType.equalsIgnoreCase("RECTANGLE")){
             return new RoundedRectangle();
          }else if(shapeType.equalsIgnoreCase("SQUARE")){
             return new RoundedSquare();
          }
          return null;
       }
    }
    //Step 5
    //Create a Factory generator/producer class to get factories by passing an
information such as Shape


class FactoryProducer {
       public static AbstractFactory getFactory(boolean rounded){
          if(rounded){
             return new RoundedShapeFactory();
          }else{
             return new ShapeFactory();
          }
       }
    }
    //Step 6
    //Use the FactoryProducer to get AbstractFactory in order to get factories of
concrete classes by passing an information such as type.



public class AbstractFactoryPatternDemo {
       public static void main(String[] args) {
          //get shape factory
          AbstractFactory shapeFactory = FactoryProducer.getFactory(false);
```

```
          //get an object of Shape Rectangle
          Shape shape1 = shapeFactory.getShape("RECTANGLE");
          //call draw method of Shape Rectangle
          shape1.draw();
          //get an object of Shape Square
          Shape shape2 = shapeFactory.getShape("SQUARE");
          //call draw method of Shape Square
          shape2.draw();
          //get shape factory
          AbstractFactory shapeFactory1 = FactoryProducer.getFactory(true);
          //get an object of Shape Rectangle
          Shape shape3 = shapeFactory1.getShape("RECTANGLE");
          //call draw method of Shape Rectangle
          shape3.draw();
          //get an object of Shape Square
          Shape shape4 = shapeFactory1.getShape("SQUARE");
          //call draw method of Shape Square
          shape4.draw();

       }
    }
```

Write a Java Program to implement Singleton pattern for multithreading

```
package singletonPattern;

final class Test1 implements Runnable {

    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println(Thread.currentThread().getName() + " : " +
Single.getInstance().hashCode());
        }
    }

}

class Single {
    private final static Single sing = new Single();
    private Single() {
    }
    public static Single getInstance() {
        return sing;
    }
}

public class Test {

    public static void main(String ar[]) {
        Test1 t = new Test1();
```

```
        Test1 t2 = new Test1();
        Test1 t3 = new Test1();
        Thread tt = new Thread(t);
        Thread tt2 = new Thread(t2);
        Thread tt3 = new Thread(t3);
        Thread tt4 = new Thread(t);
        Thread tt5 = new Thread(t);
        tt.start();
        tt2.start();
        tt3.start();
        tt4.start();
        tt5.start();

    }
}
```

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()

```
package weather;

import java.util.ArrayList;

interface DisplayElement
{
    public void display();
}

interface Observer
{
    public void update(float temp, float humidity, float pressure);
}

interface Subject
{
    public void registerObserver(Observer o);
    public void removeObserver(Observer o);
    public void notifyObservers();
}

class CurrentConditionsDisplay implements Observer, DisplayElement
{
    private float temperature;
    private float humidity;
    private Subject weatherData;

    public CurrentConditionsDisplay(Subject weatherData)
```

```
    {
        this.weatherData = weatherData;
        weatherData.registerObserver(this);
    }

    public void update(float temperature, float humidity, float pressure)
    {
        this.temperature = temperature;
        this.humidity = humidity;
        display();
    }

    public void display()
    {
        System.out.println("Current conditions: " + temperature + "F degrees and "
+ humidity + "% humidity");
    }
}

class ForecastDisplay implements Observer, DisplayElement
{
    private float currentPressure = 29.92f;
    private float lastPressure;
    private WeatherData weatherData;

    public ForecastDisplay(WeatherData weatherData)
    {
        this.weatherData = weatherData;
        weatherData.registerObserver(this);
    }

    public void update(float temp, float humidity, float pressure)
    {
        lastPressure = currentPressure;
        currentPressure = pressure;

        display();
    }

    public void display()
    {
        System.out.print("Forecast: ");
        if (currentPressure > lastPressure)
        {
            System.out.println("Improving weather on the way!");
        }
        else if (currentPressure == lastPressure)
        {
            System.out.println("More of the same");
        }
        else if (currentPressure < lastPressure)
```

```
        {
            System.out.println("Watch out for cooler, rainy weather");
        }
    }
}

class StatisticsDisplay implements Observer, DisplayElement
{
    private float maxTemp = 0.0f;
    private float minTemp = 200;
    private float tempSum= 0.0f;
    private int numReadings;
    private WeatherData weatherData;

    public StatisticsDisplay(WeatherData weatherData)
    {
        this.weatherData = weatherData;
        weatherData.registerObserver(this);
    }

    public void update(float temp, float humidity, float pressure)
    {
        tempSum += temp;
        numReadings++;

        if (temp > maxTemp)
        {
            maxTemp = temp;
        }

        if (temp < minTemp)
        {
            minTemp = temp;
        }

        display();
    }

    public void display()
    {
        System.out.println("Avg/Max/Min temperature = " + (tempSum / numReadings)+
"/" + maxTemp + "/" + minTemp);
    }
}

class WeatherData implements Subject
{
    private ArrayList<Observer> observers;
    private float temperature;
    private float humidity;
    private float pressure;
```

```
public WeatherData()
{
    observers = new ArrayList<>();
}
public void registerObserver(Observer o)
{
    observers.add(o);
}




public void removeObserver(Observer o)
{
    int i = observers.indexOf(o);
    if (i >= 0)
    {
        observers.remove(i);
    }
}
public void notifyObservers()
{
    for (int i = 0; i < observers.size(); i++)
    {
        Observer observer = (Observer)observers.get(i);
        observer.update(temperature, humidity, pressure);
    }
}



public void measurementsChanged()
{
    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;
    }
}

public class WeatherStation {
    public static void main(String[] args) {
        // Create an instance of WeatherData
        WeatherData weatherData = new WeatherData();

        // Create display elements
        CurrentConditionsDisplay currentConditionsDisplay = new
CurrentConditionsDisplay(weatherData);
        StatisticsDisplay statisticsDisplay = new StatisticsDisplay(weatherData);
        ForecastDisplay forecastDisplay = new ForecastDisplay(weatherData);

        // Simulate weather changes (replace with actual data)
        weatherData.setMeasurements(80, 65, 30.4f);
        weatherData.setMeasurements(82, 70, 29.2f);
        weatherData.setMeasurements(78, 90, 29.2f);
    }
}
```