

**1. 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().**

**weather.java**

```
import java.util.Observable;
import java.util.Observer;
```

```
class CurrentConditionsDisplay implements Observer, DisplayElement{
    Observable observable;
    private float temperature;
    private float humidity;
```

```
    public CurrentConditionsDisplay(Observable observable){
        this.observable=observable;
        observable.addObserver(this);
    }
    public void update(Observable obs, Object arg){
        if(obs instanceof WeatherData){
            WeatherData weatherData=(WeatherData)obs;
            this.temperature=weatherData.getTemperature();
            this.humidity=weatherData.getHumidity();
            display();
        }
    }
}
```

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

```
interface DisplayElement{
    public void display();
}
```

```
class ForecastDisplay implements Observer, DisplayElement{
    private float currentPressure=29.92f;
    private float lastPressure;
    public ForecastDisplay(Observable observable){
        observable.addObserver(this);
```

```
}
```

```
public void update(Observable observable, Object arg){  
    if(observable instanceof WeatherData){  
        WeatherData weatherData=(WeatherData)observable;  
        lastPressure=currentPressure;  
        currentPressure=weatherData.getPressure();  
        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 HeatIndexDisplay implements Observer, DisplayElement{  
    float heatIndex=0.0f;
```

```
    public HeatIndexDisplay(Observable observable){  
        observable.addObserver(this);  
    }
```

```
    public void update(Observable observable, Object arg){  
        if(observable instanceof WeatherData){  
            WeatherData weatherData=(WeatherData)observable;  
            float t=weatherData.getTemperature();  
            float rh=weatherData.getHumidity();  
            heatIndex=(float)(0.5*(t+61.0+(t-68)*1.2+rh*0.094));  
            display();  
        }  
    }
```

```
    public void display(){  
        System.out.println("Heat Index is" +heatIndex);
```

```
}  
}
```

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

```
public StatisticsDisplay(Observable observable){  
observable.addObserver(this);  
}
```

```
public void update(Observable observable, Object arg){  
if(observable instanceof WeatherData){  
WeatherData weatherData=(WeatherData)observable;  
float temp=weatherData.getTemperature();  
tempSum+=temp;  
numReadings++;  
if(temp > maxTemp){  
minTemp=temp;  
}  
display();  
}  
}
```

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

```
class WeatherData extends Observable{  
private float temperature;  
private float humidity;  
private float pressure;  
public WeatherData(){  
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;
}
}

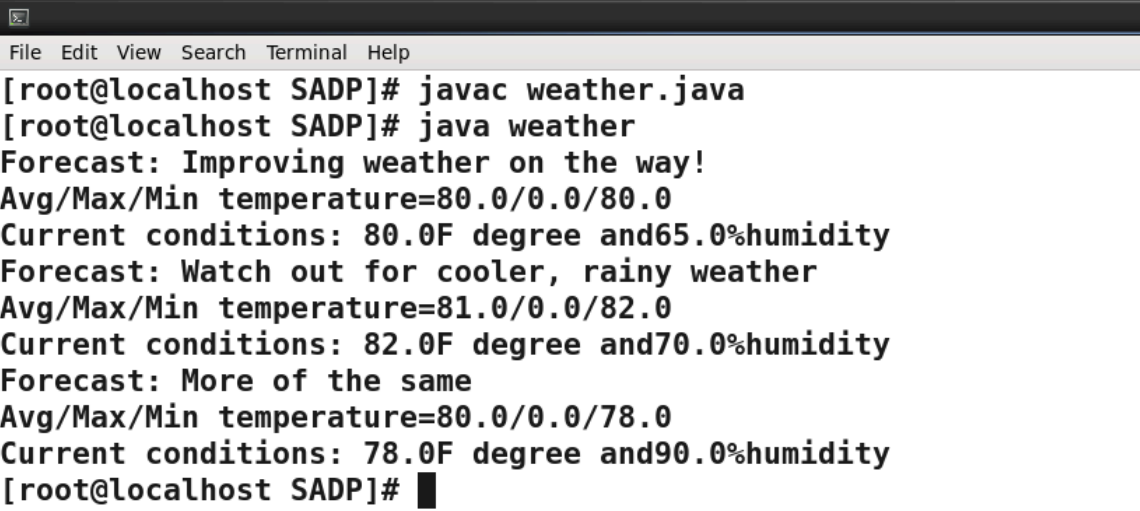
```

```

public class weather{
public static void main(String[] args){
WeatherData weatherData=new WeatherData();
CurrentConditionsDisplay currentConditions=new
CurrentConditionsDisplay(weatherData);
StatisticsDisplay statisticsDisplay=new StatisticsDisplay(weatherData);
ForecastDisplay forecastDisplay=new ForecastDisplay(weatherData);
weatherData.setMeasurements(80, 65, 30.4f);
weatherData.setMeasurements(82, 70, 29.2f);
weatherData.setMeasurements(78, 90, 29.2f);
}
}

```

## Output:



```

File Edit View Search Terminal Help
[root@localhost SADP]# javac weather.java
[root@localhost SADP]# java weather
Forecast: Improving weather on the way!
Avg/Max/Min temperature=80.0/0.0/80.0
Current conditions: 80.0F degree and65.0%humidity
Forecast: Watch out for cooler, rainy weather
Avg/Max/Min temperature=81.0/0.0/82.0
Current conditions: 82.0F degree and70.0%humidity
Forecast: More of the same
Avg/Max/Min temperature=80.0/0.0/78.0
Current conditions: 78.0F degree and90.0%humidity
[root@localhost SADP]# █

```

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

### lower.java

```
import java.io.FilterInputStream;
import java.io.IOException; import
java.io.InputStream;
class LowerCaseInputStream extends FilterInputStream {
protected 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;
}
}

public class lower {
public static void main(String[] args) {
try {

InputStream in = new LowerCaseInputStream(System.in);
int c;
System.out.println("Enter text, it will be converted to lowercase (Press Ctrl+D to
exit):");

while ((c = in.read()) >= 0) {
System.out.print((char) c);
}
} catch (IOException e) {
e.printStackTrace();
}
}
```

## Output:

```
root@localhost:~/SADP
File Edit View Search Terminal Help
[root@localhost SADP]# javac lower.java
[root@localhost SADP]# java lower
Enter text, it will be converted to lowercase (Press Ctrl+D to exit):
ANKITA KHAKARE
ankita khakare
[root@localhost SADP]#
```

**3. 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.**

**pizzaMain.java**

```
import java.util.ArrayList;
abstract class Pizza{
    String name;
    String dough;
    String sauce;
    ArrayList<String> 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(String topping : toppings){
            System.out.println(" "+topping);
        }
    }
    void bake(){
        System.out.println("bake for 25 minutes at 350");
    }
    void cut(){
        System.out.println("cutting the pizza into diagonal style");
    }
    void box(){
        System.out.println("place pizza in official pizzastore box");
    }
    public String getName(){
        return name;}
    public String toString(){
        StringBuilder display=new StringBuilder();
        display.append("---- ").append(name).append("----\n");
        display.append(dough).append("\n");
        display.append(sauce).append("\n");
        for(String topping: toppings){
            display.append(topping).append("\n");
        }
        return display.toString();
    }
}
```

```

abstract class PizzaStore{
abstract Pizza createPizza(String item);
public Pizza orderPizza(String type){
Pizza pizza = createPizza(type);
System.out.println("---making a "+pizza.getName()+"---");
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 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 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 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 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 clam from long island sound");
}
}

class NYStylePepperoniPizza extends Pizza{
public NYStylePepperoniPizza()
{

```

```

name="NY style veggie 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 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 pizza into square slices");
}
}
class ChicagoStyleVeggiePizza extends Pizza{
public ChicagoStyleVeggiePizza(){
    name="Chicago style deep dish veggie pizza";
    dough=" extra thick crust dough";
    sauce="plum tomato sauce";
    toppings.add("shredded mozzarella cheese");
    toppings.add("black olive");
    toppings.add("spinach");
    toppings.add("eggplant");
}
void cut(){
    System.out.println("cutting 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 clam from chesapeake bay");
}
}

```

```

void cut(){
System.out.println("cutting 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 olive");
    toppings.add("spinach");
    toppings.add("eggplant");
    toppings.add("sliced pepperoni");
}
void cut(){
System.out.println("cutting pizza into square slices");
}
}
public class pizzaMain{
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");

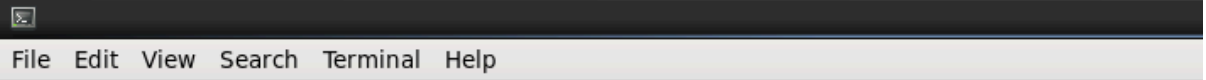
pizza = nyStore.orderPizza("pepperoni");
System.out.println("ethan ordered a "+pizza.getName()+"\n");
pizza=chicagoStore.orderPizza("pepperoni");
System.out.println("joel ordered a "+pizza.getName()+"\n");

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");
}
}

```

}

Output:



```
[root@localhost SADP]# javac pizzaMain.java
[root@localhost SADP]# java pizzaMain
---making a NY style sauce and cheese pizza---
preparingNY style sauce and cheese pizza
Tossing dough...
adding sauce...
adding toppings:
    grated reggiano cheese
bake for 25 minutes at 350
cutting the pizza into diagonal style
place pizza in official pizzastore box
ethan ordered a NY style sauce and cheese pizza

---making a Chicago style deep dish cheese pizza---
preparingChicago style deep dish cheese pizza
Tossing dough...
adding sauce...
adding toppings:
    shredded mozzarella cheese
bake for 25 minutes at 350
cutting pizza into square slices
place pizza in official pizzastore box
joel ordered a Chicago style deep dish cheese pizza

---making a NY style clam pizza---
preparingNY style clam pizza
Tossing dough...
adding sauce...
adding toppings:
    grated reggiano cheese
    fresh clam from long island sound
bake for 25 minutes at 350
```



File Edit View Search Terminal Help

```
---making a Chicago style clam pizza---  
preparingChicago style clam pizza  
Tossing dough...  
adding sauce...  
adding toppings:  
  shredded mozzarella cheese  
  frozen clam from chesapeake bay  
bake for 25 minutes at 350  
cutting pizza into square slices  
place pizza in official pizzastore box  
joel ordered a Chicago style clam pizza
```

```
---making a NY style veggie pizza---  
preparingNY style veggie pizza  
Tossing dough...  
adding sauce...  
adding toppings:  
  grated reggiano cheese  
  sliced pepperoni  
  garlic  
  onion  
  mushrooms  
  red pepper  
bake for 25 minutes at 350  
cutting the pizza into diagonal style  
place pizza in official pizzastore box  
ethan ordered a NY style veggie pizza
```

```
---making a Chicago style Pepperoni pizza---  
preparingChicago style Pepperoni pizza  
Tossing dough...
```

```
root@localhost:~/SADP
File Edit View Search Terminal Help
joel ordered a Chicago style Pepperoni pizza

---making a NY style veggie pizza---
preparingNY style veggie pizza
Tossing dough...
adding sauce...
adding toppings:
  grated reggiano cheese
  garlic
  onion
  mushrooms
  red pepper
bake for 25 minutes at 350
cutting the pizza into diagonal style
place pizza in official pizzastore box
ethan ordered a NY style veggie pizza

---making a Chicago style deep dish veggie pizza---
preparingChicago style deep dish veggie pizza
Tossing dough...
adding sauce...
adding toppings:
  shredded mozzarella cheese
  black olive
  spinach
  eggplant
bake for 25 minutes at 350
cutting pizza into square slices
place pizza in official pizzastore box
joel ordered a Chicago style deep dish veggie pizza

[root@localhost SADP]# █
```

#### 4. Write a Java Program to implement Singleton pattern for multithreading.

##### **singleton.java**

```
public class singleton
{
    private static volatile singleton instance;
    private singleton(){}
    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 instance: "+instance);
    }
}
```

##### **singletonRunnable.java**

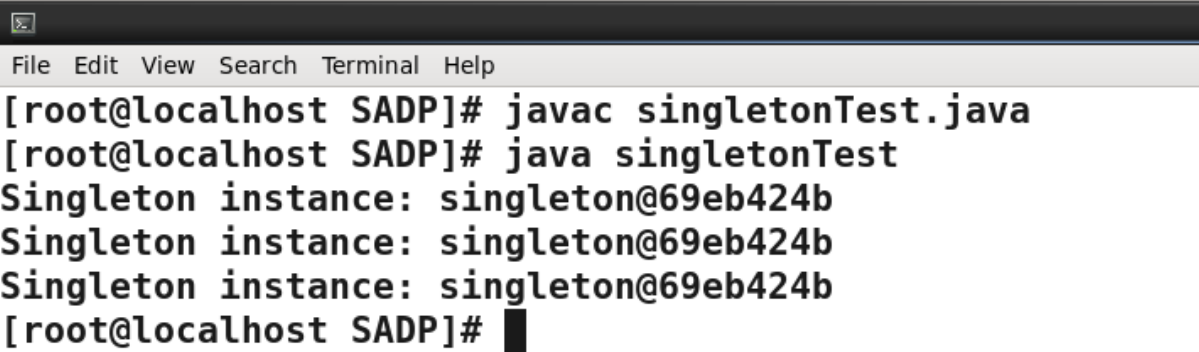
```
class singletonRunnable implements Runnable
{
    public void run(){
        singleton Singleton = singleton.getInstance();
        Singleton.showMessage();
    }
}
```

##### **singletonTest.java**

```
public class singletonTest{
    public static void main(String args[]){
        Thread thread1=new Thread (new singletonRunnable(),"Thread-1");
        Thread thread2=new Thread (new singletonRunnable(),"Thread-2");
        Thread thread3=new Thread (new singletonRunnable(),"Thread-3");
```

```
thread1.start();
thread2.start();
thread3.start();
}
}
```

### Output:

A terminal window with a dark title bar and a light menu bar containing 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal content shows the execution of a Java program. It starts with a compilation command, followed by a run command. The output consists of three identical lines, each displaying 'Singleton instance: singleton@69eb424b'. The prompt '[root@localhost SADP]#' is visible at the end of each line.

```
[root@localhost SADP]# javac singletonTest.java
[root@localhost SADP]# java singletonTest
Singleton instance: singleton@69eb424b
Singleton instance: singleton@69eb424b
Singleton instance: singleton@69eb424b
[root@localhost SADP]#
```



## 5. Write a Java Program to implement command pattern to test Remote Control. Book

### remoteMain.java

```
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 LightOnCommand implements Command{
    Light l1;
    public LightOnCommand(Light a){
        this.l1=a;
    }
    public void execute(){
        l1.on();
    }
}
class LightOffCommand implements Command{
    Light l1;
    public LightOffCommand(Light a){
        this.l1=a;
    }
    public void execute(){
        l1.off();
    }
}
class SimpleRemoteControl{
    Command slot;
    public SimpleRemoteControl(){
    }
    public void setCommand(Command command){
        slot=command;
    }
    public void buttonWasPressed(){
        slot.execute();
    }
}
```

```
}  
public class remoteMain{  
    public static void main(String[] args)  
    {  
        SimpleRemoteControl r1=new SimpleRemoteControl();  
        Light l1=new Light();  
        LightOnCommand lo=new LightOnCommand(l1);  
        r1.setCommand(lo);  
        r1.buttonWasPressed();  
        LightOffCommand IO=new LightOffCommand(l1);  
        r1.setCommand(IO);  
        r1.buttonWasPressed();  
    }  
}
```

### Output:



A terminal window with a dark title bar and a light menu bar containing 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal output shows the compilation and execution of the Java program, resulting in 'Light is on' and 'Light is off' messages.

```
[root@localhost SADP]# javac remoteMain.java  
[root@localhost SADP]# java remoteMain  
Light is on  
Light is off  
[root@localhost SADP]#
```

## 6. Write a Java Program to implement undo command to test Ceilingfan.

**fan.java**

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

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

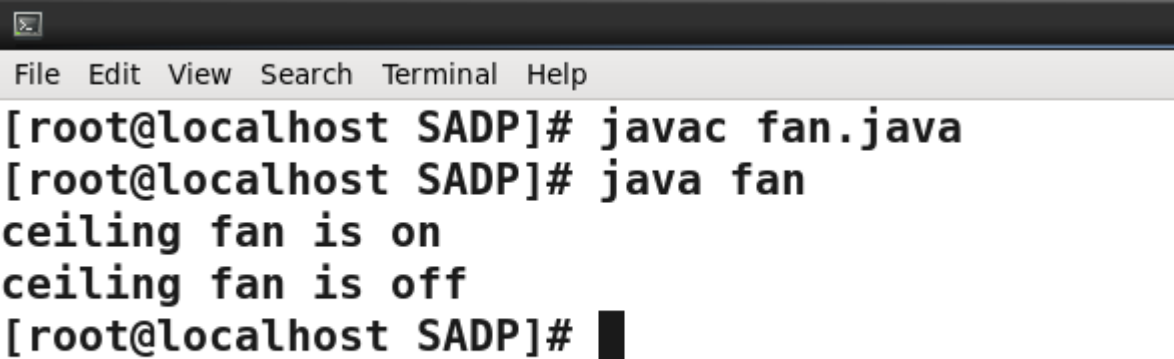
class CeilingFanOnCommand implements Command {
    CeilingFan c;
    public CeilingFanOnCommand(CeilingFan l) {
        this.c=l;
    }
    public void execute() {
        c.on();
    }
}

class CeilingFanOffCommand implements Command { CeilingFan c;
    public CeilingFanOffCommand(CeilingFan l) {
        this.c=l;
    }
    public void execute() {
        c.off();
    }
}

class SimpleRemoteControl {
    Command slot;
    public SimpleRemoteControl() {}
    public void setCommand(Command command) {
        slot=command;
    }
    public void buttonWasPressed() {
        slot.execute();
    }
}
```

```
}  
public class fan {  
    public static void main(String[] args) {  
        SimpleRemoteControl remote=new SimpleRemoteControl();  
        CeilingFan ceilingFan=new CeilingFan();  
        CeilingFanOnCommand ceilingFanOn=new CeilingFanOnCommand(ceilingFan);  
        remote.setCommand(ceilingFanOn);  
        remote.buttonWasPressed();  
        CeilingFanOffCommand ceilingFanOff=new CeilingFanOffCommand(ceilingFan);  
        remote.setCommand(ceilingFanOff);  
        remote.buttonWasPressed();  
    }  
}
```

### Output:



```
File Edit View Search Terminal Help  
[root@localhost SADP]# javac fan.java  
[root@localhost SADP]# java fan  
ceiling fan is on  
ceiling fan is off  
[root@localhost SADP]#
```

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

### MenuItem.java

```
public class MenuItem{
    private String name;
    private String description;
    private boolean vegetarian;
    private double price;
    public MenuItem(String name, String description, boolean vegetarian, double price){
        this.name=name;
        this.description=description;
        this.vegetarian=vegetarian;
        this.price=price;
    }
    public String getName(){
        return name;
    }
    public String getDescription(){
        return description;
    }
    public boolean isVegetarian(){
        return vegetarian;
    }
    public double getPrice(){
        return price;
    }
}
```

### Menu.java

```
import java.util.Iterator;
public interface Menu{
    Iterator<MenuItem> createIterator();
}
```

### BreakfastMenu.java

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class BreakfastMenu implements Menu{
    private List<MenuItem> menuItems;
```

```

public BreakfastMenu(){
    menuItems= new ArrayList<>();
    addItem("Pancakes","Pancakes with syrup", true, 2.99);
    addItem("waffles","waffles with blueberries", true, 3.49);
}
public void addItem(String name, String description, boolean vegetarian, double price){
    MenuItem menuItem=new MenuItem(name, description, vegetarian, price);
    menuItems.add(menuItem);
}
public Iterator<MenuItem> createIterator(){
    return menuItems.iterator();
}
}

```

### **LunchMenu.java**

```

import java.util.Iterator;

```

```

public class LunchMenu implements Menu{
    private MenuItem[] menuItems;
    private int numberOfItems=0;

```

```

    public LunchMenu(){
        menuItems=new MenuItem[2];
        addItem("Burger","Beef burger with fries", false, 5.99);
        addItem("Salad","Caesar salad", true, 4.99);
    }

```

```

    public void addItem(String name, String description, boolean vegetarian, double price){
        MenuItem menuItem=new MenuItem(name, description, vegetarian , price);
        if(numberOfItems >= menuItems.length){
            System.err.println("Menu is full! can't add to menu");
        }
        else
        {
            menuItems[numberOfItems]=menuItem;
            numberOfItems++;
        }
    }
    public Iterator<MenuItem> createIterator(){
        return new LunchMenuIterator(menuItems);
    }
}

```

```

}
class LunchMenuIterator implements Iterator <MenuItem>{
private MenuItem[] items;
private int position=0;
public LunchMenuIterator(MenuItem[] items)
{
this.items=items;
}
public boolean hasNext(){
return position < items.length && items[position]!=null;
}
public MenuItem next(){
MenuItem menuItem=items[position];
position++;
return menuItem;
}
}

```

### **DinnerMenu.java**

```

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

public class DinnerMenu implements Menu{
private List<MenuItem> menuItems;

public DinnerMenu(){
menuItems=new ArrayList<>();
addItem("steak","grilled steak with vegetables", false, 9.99);
addItem("pasta", "pasta with marinara sauce", true, 7.99);
}
public void addItem(String name, String description, boolean vegetarian, double price){
MenuItem menuItem=new MenuItem(name,description,vegetarian,price);
menuItems.add(menuItem);
}
public Iterator<MenuItem> createIterator(){
return menuItems.iterator();
}
}

```

### **MenuTest.java**

```

import java.util.Iterator;
public class MenuTest

```

```

{
public static void main(String[] args)
{
Menu breakfastMenu=new BreakfastMenu();
Menu lunchMenu=new LunchMenu();
Menu dinnerMenu=new DinnerMenu();

PrintMenu(breakfastMenu.createIterator());
PrintMenu(lunchMenu.createIterator());
PrintMenu(dinnerMenu.createIterator());
}
public static void PrintMenu(Iterator<MenuItem>iterator)
{
while(iterator.hasNext())
{
MenuItem menuItem=iterator.next();
System.out.println(menuItem.getName()+" , ");
System.out.println(menuItem.getPrice()+" -- ");
System.out.println(menuItem.getDescription());
}
}
}

```

### Output:

```

Pancakes ,
2.99 --
Pancake With Syrup
Waffles ,
3.49 --
Waffles with blueberries
Burger ,
5.99 --
Beef buger with fries
Salad ,
4.99 -- Caesar
salad
Steak ,
9.99 --
Grilled With vegetable
Pasta ,
7.99 --
Pasta with Marinara sauce

```



**8. Write a java program to implements Abstract Factory pattern for shape interface**

**shapeMain.java**

```
interface Shape
{
    void draw();
}
class RoundedRectangle implements Shape
{
    public void draw()
    {
        System.out.println("Inside RR");
    }
}
class RoundedSquare implements Shape
{
    public void draw()
    {
        System.out.println("Inside RS");
    }
}
class Rectangle implements Shape
{
    public void draw()
    {
        System.out.println("Inside Simple R");
    }
}
class Square implements Shape
{
    public void draw()
    {
        System.out.println("Inside Simple sq");
    }
}

abstract class AbstractFactory
{
    abstract Shape getShape(String st);
}
class ShapeFactory extends AbstractFactory
```

```

{
public Shape getShape(String st)
{
if(st.equalsIgnoreCase("Rectangle"))
{
return new Rectangle();
}
else if(st.equalsIgnoreCase("Square"))
{
return new Square();
}
return null;
}
}

class RoundedShapeFactory extends AbstractFactory
{
public Shape getShape(String st)
{
if(st.equalsIgnoreCase("Rectangle"))
{
return new RoundedRectangle();
}
else if(st.equalsIgnoreCase("Square"))
{
return new RoundedSquare();
}
return null;
}
}

class FactoryProducer
{
public static AbstractFactory getFactory(boolean rounded)
{
if(rounded)
{
return new RoundedShapeFactory();
}
else
{
return new ShapeFactory();
}
}
}

```

```
public class shapeMain
{
    public static void main(String[] args)
    {
        AbstractFactory shapeFactory=FactoryProducer.getFactory(false);
        Shape shape1=shapeFactory.getShape("Rectangle");
        shape1.draw();
        Shape shape2=shapeFactory.getShape("Square");
        shape1.draw();
        AbstractFactory shapeFactory1=FactoryProducer.getFactory(true);
        Shape shape3=shapeFactory1.getShape("Rectangle");
        shape3.draw();
        Shape shape4=shapeFactory1.getShape("square");
        shape4.draw();
    }
}
```

### Output:



```
[root@localhost SADP]# javac shapeMain.java
[root@localhost SADP]# java shapeMain
Inside Simple R
Inside Simple R
Inside RR
Inside RS
[root@localhost SADP]#
```

**9. 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, StereoOnWithCDCommand.**

**commandMain.java**

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

class Stereo{
    public void On(){
        System.out.println("Stereo is on");
    }
}

class GarageDoor{
    public void Up(){
        System.out.println("Garage Door is up");
    }
}

class GarageDoorUpCommand implements Command{
    GarageDoor c;
    public GarageDoorUpCommand(GarageDoor l){
        this.c=l;
    }
    public void execute(){
        c.Up();
    }
}

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

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

```

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

class StereoOn implements Command{
    Stereo s;
    public StereoOn(Stereo l){
        this.s=l;
    }
    public void execute(){
        s.On();
    }
}

class SimpleRemoteControl{
    Command slot;
    public SimpleRemoteControl() {}
    public void setCommand(Command command){
        slot=command;
    }
    public void buttonWasPressed(){
        slot.execute();
    }
}

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

        LightOnCommand lightOn=new LightOnCommand(light);
        remote.setCommand(lightOn); remote.buttonWasPressed();

        LightOffCommand lightOff=new LightOffCommand(light);
        remote.setCommand(lightOff); remote.buttonWasPressed();
    }
}

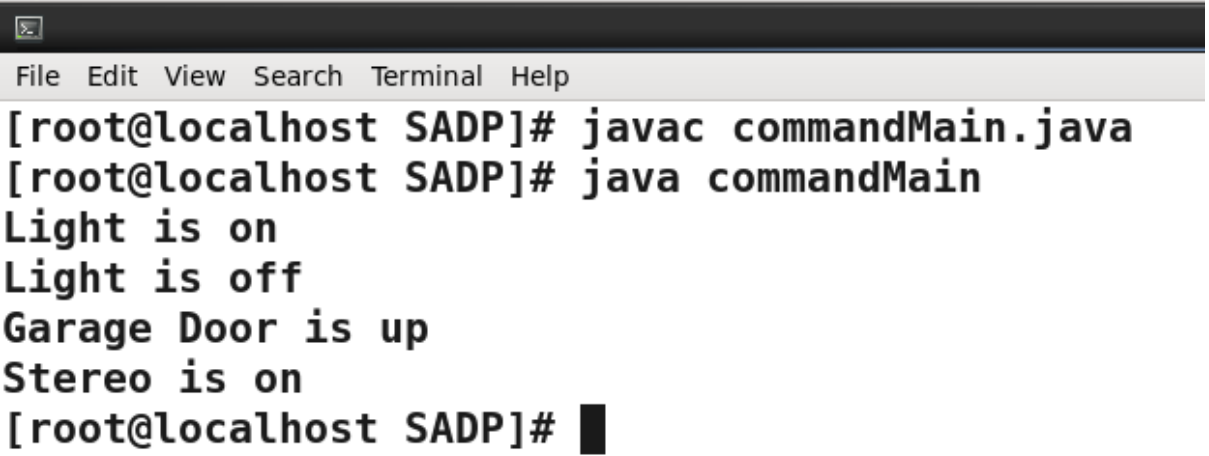
```

```
GarageDoor garageDoor=new GarageDoor();
GarageDoorUpCommand garageDoorUp=new
GarageDoorUpCommand(garageDoor);
remote.setCommand(garageDoorUp); remote.buttonWasPressed();
```

```
Stereo s1=new Stereo(); StereoOn
s2=new StereoOn(s1);
remote.setCommand(s2);
remote.buttonWasPressed();
```

```
}
}
```

### Output:



```
File Edit View Search Terminal Help
[root@localhost SADP]# javac commandMain.java
[root@localhost SADP]# java commandMain
Light is on
Light is off
Garage Door is up
Stereo is on
[root@localhost SADP]#
```

**10 . Write a java program to implements state pattern for Gumball Machine. Create Instance variable that holds current state from there, we just need to handle all actions, behaviour and state transition that can happen**

### **GumballMachineTest.java**

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

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 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 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 -= 1;
        }
    }
    int getCount() {
        return count;
    }
    void refill(int count) {
        this.count += count;
    }
}

```

```

System.out.println("The gumball machine was just refilled; its 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() {
StringBuilder result = new StringBuilder();

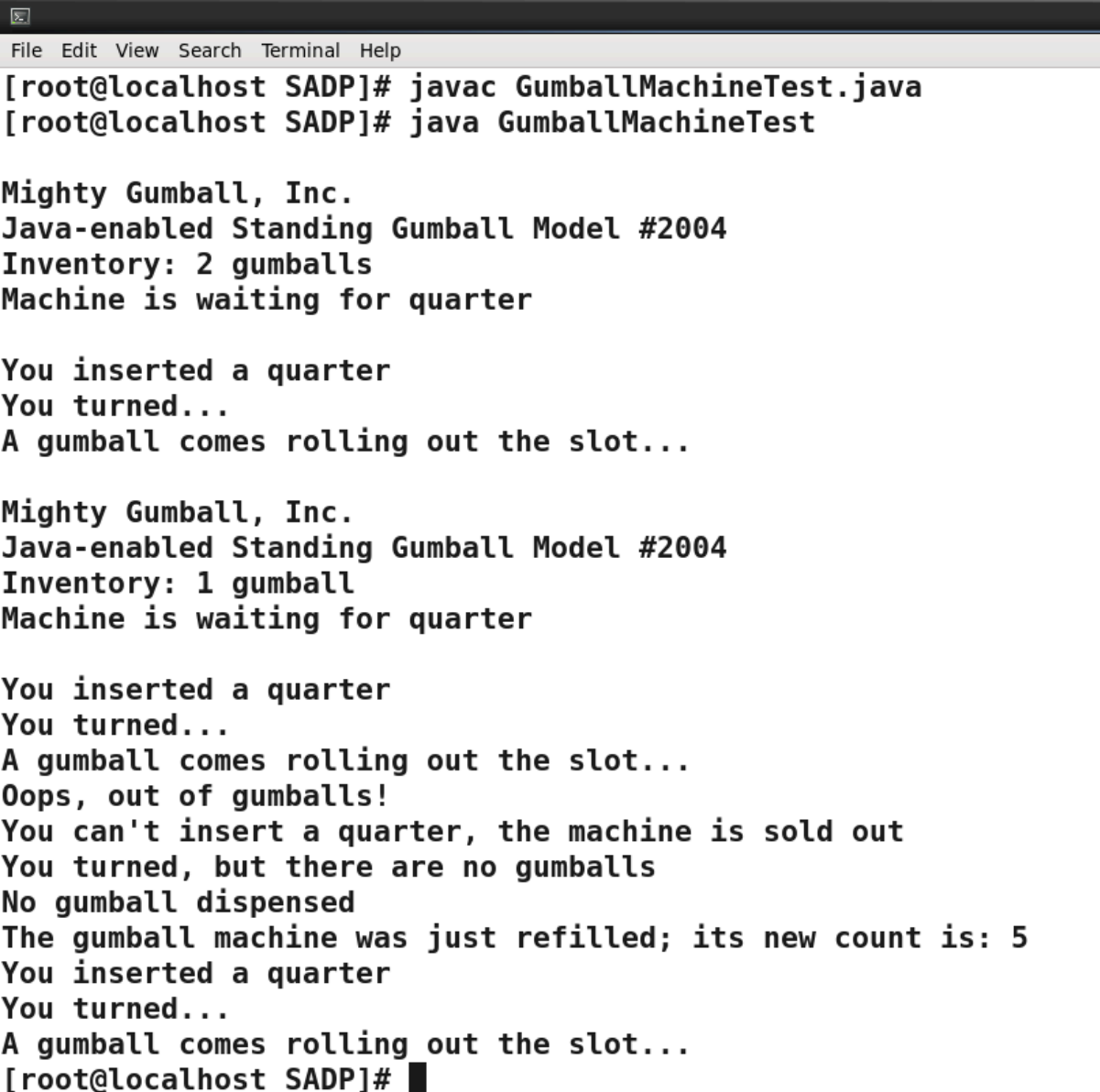
result.append("\nMighty Gumball, Inc.");
result.append("\nJava-enabled Standing Gumball Model #2004");
result.append("\nInventory: " + count + " gumball");
if (count !=1) {
result.append("s");
}
result.append("\n");
result.append("Machine is " + state + "\n");
return result.toString();
}
}

public class GumballMachineTest{
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();
}
}
```

### Output:



```
File Edit View Search Terminal Help
[root@localhost SADP]# javac GumballMachineTest.java
[root@localhost SADP]# java GumballMachineTest

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 2 gumballs
Machine is waiting for quarter

You inserted a quarter
You turned...
A gumball comes rolling out the slot...

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 1 gumball
Machine is waiting for quarter

You inserted a quarter
You turned...
A gumball comes rolling out the slot...
Oops, out of gumballs!
You can't insert a quarter, the machine is sold out
You turned, but there are no gumballs
No gumball dispensed
The gumball machine was just refilled; its new count is: 5
You inserted a quarter
You turned...
A gumball comes rolling out the slot...
[root@localhost SADP]#
```

## 11. Write a java program to implement Adapter Pattern to design Heat Model.

```
import javax.sound.midi.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;

// Define BeatObserver and BPMObserver interfaces
interface BeatObserver {
    void updateBeat();
}

interface BPMObserver {
    void updateBPM();
}

// Define the BeatModelInterface interface
BeatModelInterface {
    void initialize();
    void on();
    void off();
    void setBPM(int bpm);
    int getBPM();
    void registerObserver(BeatObserver o);
    void removeObserver(BeatObserver o);
    void registerObserver(BPMObserver o);
    void removeObserver(BPMObserver o);
}

// Implement the BeatModel class
class BeatModel implements BeatModelInterface, MetaEventListener {

    Sequencer sequencer;
    ArrayList<BeatObserver> beatObservers = new ArrayList<>();
    ArrayList<BPMObserver> bpmObservers = new ArrayList<>();
    int bpm = 90;

    public void initialize() {
        setUpMidi();
        buildTrackAndStart();
    }
}
```

```
public void on() {  
    sequencer.start();  
    setBPM(90);  
}
```

```
public void off() {  
    setBPM(0);  
    sequencer.stop();  
}
```

```
public void setBPM(int bpm) {  
    this.bpm = bpm;  
    sequencer.setTempoInBPM(getBPM());  
    notifyBPMObservers();  
}
```

```
public int getBPM() {  
    return bpm;  
}
```

```
public void registerObserver(BeatObserver o) {  
    beatObservers.add(o);  
}
```

```
public void removeObserver(BeatObserver o) {  
    beatObservers.remove(o);  
}
```

```
public void registerObserver(BPMObserver o) {  
    bpmObservers.add(o);  
}
```

```
public void removeObserver(BPMObserver o) {  
    bpmObservers.remove(o);  
}
```

```
void beatEvent() {  
    notifyBeatObservers();  
}
```

```
private void notifyBeatObservers() {  
    for(BeatObserver observer : beatObservers) {  
        observer.updateBeat();  
    }  
}
```

```
}
```

```
private void notifyBPMObservers() {
```

```
for (BPMObserver observer : bpmObservers) {
```

```
observer.updateBPM();
```

```
}
```

```
}
```

```
private void setUpMidi() {
```

```
try {
```

```
sequencer = MidiSystem.getSequencer();
```

```
sequencer.open();
```

```
sequencer.addMetaEventListener(this);
```

```
}
```

```
catch (Exception e) {
```

```
e.printStackTrace();
```

```
}
```

```
}
```

```
private void buildTrackAndStart() {
```

```
try {
```

```
Sequence sequence = new Sequence(Sequence.PPQ, 4);
```

```
Track track = sequence.createTrack(); for (int i = 0; i < 60; i += 2) {
```

```
track.add(new MidiEvent(new ShortMessage(ShortMessage.NOTE_ON, 9, 60, 100),  
i));
```

```
track.add(new MidiEvent(new ShortMessage(ShortMessage.NOTE_OFF, 9, 60, 100),  
i + 1));
```

```
}
```

```
sequencer.setSequence(sequence);
```

```
sequencer.setTempoInBPM(bpm);
```

```
}
```

```
catch (Exception e) {
```

```
e.printStackTrace();
```

```
}
```

```
}
```

```
public void meta(MetaMessage meta) {
```

```
if (meta.getType() == 47) {
```

```
beatEvent();
```

```
}
```

```
}
```

```
}
```

```

// Implement DJView class
class DJView implements ActionListener, BeatObserver, BPMObserver {
    BeatModelInterface model;
    ControllerInterface controller;
    JFrame viewFrame;
    JPanel viewPanel;
    BeatBar beatBar;
    JLabel bpmOutputLabel;
    JButton startButton, stopButton, increaseBPMButton, decreaseBPMButton;

    public DJView(ControllerInterface controller, BeatModelInterface model) {
        this.controller = controller; this.model = model;
        model.registerObserver((BeatObserver) this); // Explicitly cast to BeatObserver
        model.registerObserver((BPMObserver) this); // Explicitly cast to BPMObserver
        createView(); createControls();
    }

    public void createView() {
        viewFrame = new JFrame("DJ View");
        viewPanel = new JPanel();
        beatBar = new BeatBar();
        bpmOutputLabel = new JLabel("Current BPM: 90");
        viewPanel.add(bpmOutputLabel);
        viewPanel.add(beatBar);
        viewFrame.getContentPane().add(viewPanel);
        viewFrame.setSize(300, 200);
        viewFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        viewFrame.setVisible(true);
    }

    public void createControls() {
        startButton = new JButton("Start");
        stopButton = new JButton("Stop");
        increaseBPMButton = new JButton("Increase BPM");
        decreaseBPMButton = new JButton("Decrease BPM");
        startButton.addActionListener(this);
        stopButton.addActionListener(this);
        increaseBPMButton.addActionListener(this);
        decreaseBPMButton.addActionListener(this);
        viewPanel.add(startButton);
        viewPanel.add(stopButton);
        viewPanel.add(increaseBPMButton);
        viewPanel.add(decreaseBPMButton);
        disableStopMenuItem(); enableStartMenuItem();
    }

```



```

}

public void updateBPM() {
int bpm= model.getBPM();
if (bpm == 0) {
bpmOutputLabel.setText("Offline");
}
else {
bpmOutputLabel.setText("Current BPM: " + model.getBPM());
}
}

public void updateBeat() {
beatBar.setValue(100);
}

public void actionPerformed(ActionEvent e) {
if (e.getSource() == startButton) {
controller.start();
}
else if (e.getSource() == stopButton) {
controller.stop();
}
else if (e.getSource() == increaseBPMBUTTON) {
controller.increaseBPM();
}
else if (e.getSource() == decreaseBPMBUTTON) {
controller.decreaseBPM();
}
}

public void enableStartMenuItem() {
startButton.setEnabled(true);
}

public void disableStartMenuItem() {
startButton.setEnabled(false);
}

public void enableStopMenuItem() {
stopButton.setEnabled(true);
}

public void disableStopMenuItem() {
stopButton.setEnabled(false);
}
}

// Implement ControllerInterface
interface ControllerInterface {
void start();

```

```

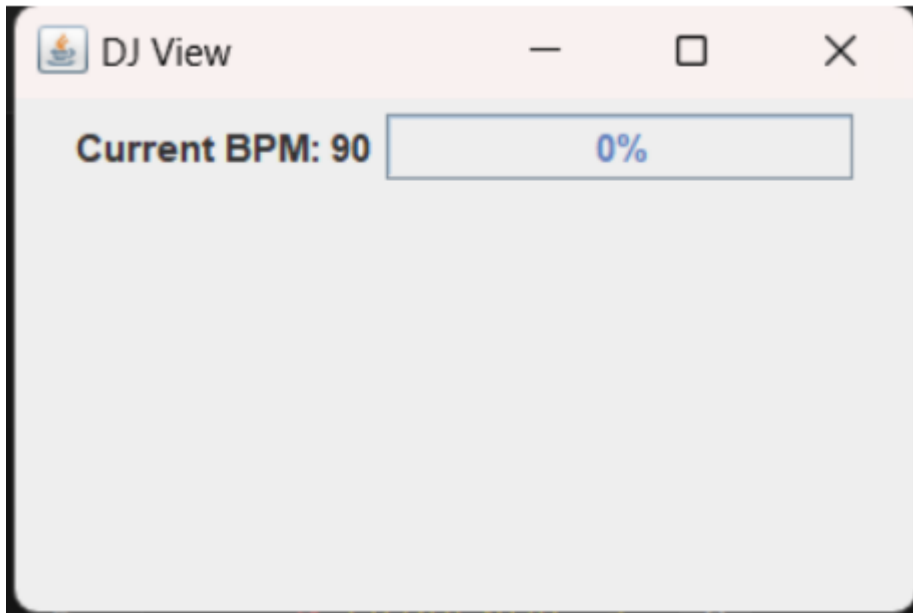
void stop();
void increaseBPM();
void decreaseBPM();
void setBPM(int bpm);
}
// Implement BeatController class class
BeatController implements ControllerInterface {
    BeatModelInterface model;
    DJView view;
    public BeatController(BeatModelInterface model) {
        this.model = model;
        view = new DJView(this,model);
        model.initialize();
    }
    public void start() {
        model.on();
        view.disableStartMenuItem();
        view.enableStopMenuItem();
    }
    public void stop() {
        model.off();
        view.enableStartMenuItem();
        view.disableStopMenuItem();
    }
    public void increaseBPM() {
        int bpm = model.getBPM();
        model.setBPM(bpm + 1);
    }
    public void decreaseBPM() {
        int bpm = model.getBPM();
        model.setBPM(bpm - 1);
    }
    public void setBPM(int bpm) {
        model.setBPM(bpm);
    }
}
// Dummy BeatBar class class
BeatBar extends JProgressBar {
    public BeatBar() {
        setValue(0);
        setStringPainted(true);
    }
}

```

```
// Main class
public class beatModel
{
    public static void main(String[] args)
    {
        BeatModelInterface model = new BeatModel();

        ControllerInterface controller = new BeatController(model);
    }
}
```

**Output:**



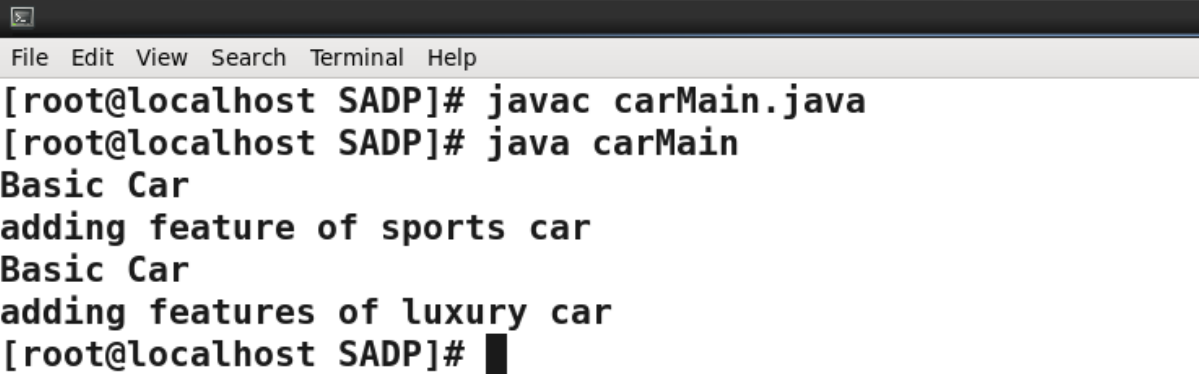
**12. Write a java program to implement decorator pattern for interface car assemble() method and then decorator it to sports car and luxury car.**

**carMain.java**

```
interface Car{
    public void assemble();
}
class BasicCar implements Car{
    public void assemble(){
        System.out.println("Basic Car");
    }
}
class CarDecorator implements Car{
    protected Car car;
    public CarDecorator(Car c){
        this.car=c;
    }
    public void assemble(){
        this.car.assemble();
    }
}
class SportsCar extends CarDecorator{
    public SportsCar(Car c){
        super(c);
    }
    public void assemble(){
        car.assemble();
        System.out.println("adding feature of sports car");
    }
}
class LuxuryCar extends CarDecorator{
    public LuxuryCar(Car c){
        super(c);
    }
    public void assemble(){
        car.assemble();
        System.out.println("adding features of luxury car");
    }
}
public class carMain{
    public static void main(String[] args){
        Car s1 = new SportsCar(new BasicCar());
```

```
s1.assemble();  
Car s2= new LuxuryCar(new BasicCar());  
s2.assemble();  
}  
}
```

### Output:



```
File Edit View Search Terminal Help  
[root@localhost SADP]# javac carMain.java  
[root@localhost SADP]# java carMain  
Basic Car  
adding feature of sports car  
Basic Car  
adding features of luxury car  
[root@localhost SADP]#
```

**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 and default 120 volts. Implements adapter pattern using class adapter.**

**voltMain.java**

```
class Volt{
private int volts;
public Volt(int v){
this.volts=v;
}
public int getVolts(){
return volts;
}
public void setVolts(int volts){
this.volts= volts;
}
}
class Socket{
public Volt getVolt(){
return new Volt(120);
}
}
interface SocketAdapter{
public Volt get120Volt();
public Volt get12Volt();
public Volt get3Volt();
}
class SocketClassAdapterImp1 extends Socket implements SocketAdapter{
public Volt get120Volt(){
return getVolt();
}
public Volt get12Volt(){
Volt v=getVolt();
return convertVolt(v,10);
}
public Volt get3Volt(){
Volt v=getVolt();
return convertVolt(v,40);
}
private Volt convertVolt(Volt v, int i){
return new Volt(v.getVolts()/i);
}
```

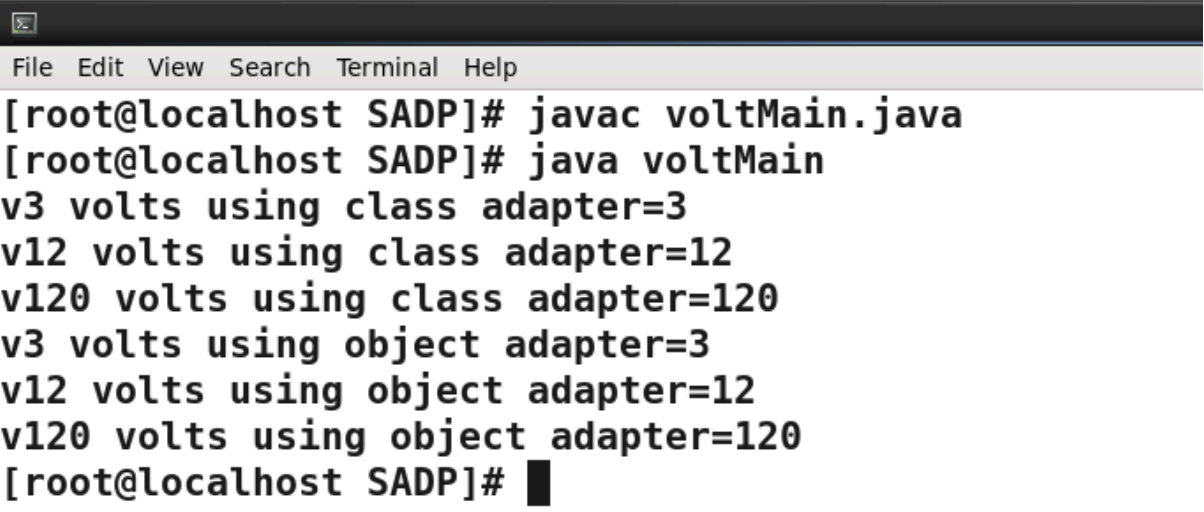
```

}
}
class SocketObjectAdapterImp1 implements SocketAdapter{
private Socket sock=new Socket();
public Volt get120Volt(){
return sock.getVolt();
}
public Volt get12Volt(){
Volt v= sock.getVolt();
return convertVolt(v, 10);
}
public Volt get3Volt(){
Volt v=sock.getVolt();
return convertVolt(v,40);
}
private Volt convertVolt(Volt v, int i){
return new Volt(v.getVolts()/i);
}
}
public class voltMain{
public static void main(String[] args){
testClassAdapter();
testObjectAdapter();
}
private static void testObjectAdapter(){
SocketAdapter socketAdapter= new SocketObjectAdapterImp1();
Volt v3=getVolt(socketAdapter,3);
Volt v12= getVolt(socketAdapter,12);
Volt v120=getVolt(socketAdapter, 120);
System.out.println("v3 volts using object adapter="+v3.getVolts());
System.out.println("v12 volts using object adapter="+v12.getVolts());
System.out.println("v120 volts using object adapter="+v120.getVolts());
}
private static void testClassAdapter (){
SocketAdapter socketAdapter = new SocketClassAdapterImp1();
Volt v3 = getVolt(socketAdapter, 3);
Volt v12= getVolt(socketAdapter, 12);
Volt v120=getVolt(socketAdapter, 120);
System.out.println("v3 volts using class adapter="+v3.getVolts());
System.out.println("v12 volts using class adapter="+v12.getVolts());
System.out.println("v120 volts using class adapter="+v120.getVolts());
}
private static Volt getVolt(SocketAdapter socketAdapter, int i){
switch(i){

```

```
case 3: return socketAdapter.get3Volt();
case 12: return socketAdapter.get12Volt();
case 120: return socketAdapter.get120Volt();
default: return socketAdapter.get120Volt();
}
}
}
```

### Output:

A screenshot of a terminal window with a dark background and a light-colored title bar. The title bar contains the text "File Edit View Search Terminal Help". The terminal content shows the compilation and execution of a Java program. The prompt is [root@localhost SADP]#. The commands executed are javac voltMain.java and java voltMain. The output shows three lines of results for the class adapter, followed by three lines of results for the object adapter. The final prompt is [root@localhost SADP]# followed by a cursor.

```
[root@localhost SADP]# javac voltMain.java
[root@localhost SADP]# java voltMain
v3 volts using class adapter=3
v12 volts using class adapter=12
v120 volts using class adapter=120
v3 volts using object adapter=3
v12 volts using object adapter=12
v120 volts using object adapter=120
[root@localhost SADP]#
```



#### 14. Write a java program to implement a facade design pattern for hometheater.

##### **Amplifier.java**

```
package headfirst.facade.hometheater;

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 volume to"+tuner);
        this.tuner=tuner;
    }
    public void setDvd(DvdPlayer dvd){
        System.out.println(description+"setting volume to"+dvd);
        this.dvd=dvd;
    }
    public void setCd(CdPlayer cd){
        System.out.println(description+"setting volume to"+cd);
        this.cd=cd;
    }
    public String toString(){
        return description;
    }
}
```

### **Tuner.java**

```
package headfirst.facade.hometheater;
class Tuner{
    String description;
    Amplifier amplifier;
    double frequency;
    public Tuner(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 setFrequency(double frequency){
        this.frequency = frequency;
        System.out.println(description + " setting frequency to " + frequency);
    }
    public String toString(){return description;
    }
}
```

### **CdPlayer.java**

```
package headfirst.facade.hometheater;
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" + track+ ", 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;
}
}

```

### **DvdPlayer.java**

```

package headfirst.facade.hometheater;
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(){
title=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(title==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;
}
}

```

### **Projector.java**

```

package headfirst.facade.hometheater;
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;
}
}

```

### **TheaterLights.java**

```

package headfirst.facade.hometheater;
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;
}
}

```

### **Screen.java**

```
package headfirst.facade.hometheater;
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;
    }
}
```

### **PopcornPopper.java**

```
package headfirst.facade.hometheater;
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;
    }
}
```

### **TheaterMain.java**

```
package headfirst.facade.hometheater;
```

```

public class TheaterMain{
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-OLine 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();
}
}

```

### **Output:**

```

Get ready to watch a movie...
Popcorn Popper is on
Popcorn Popper popping popcorn!
Theater Ceiling Lights dimmed to 10%
Theater Screen is down
Top-O-Line Projector is on
Top-O-Line Projector in widescreen mode
Top-O-Line Amplifier is on
Top-O-Line Amplifier volume set to 5
Top-O-Line DVD Player is on Top-O-Line
DVD Player playing "Inception" Enjoy your movie!
Shutting down the movie theater...
Popcorn Popper is off
Theater Ceiling Lights are on
Theater Screen is up
Top-O-Line Projector is off
Top-O-Line Amplifier is off
Top-O-Line DVD Player is off
Getting ready to listen to the radio
Top-O-Line Tuner is on
Top-O-Line Tuner set to FM 101 MHz
Top-O-Line Amplifier is on

```

