AOOP LAB

LAB SESSION 01: STRUCTURAL PATTERNS

Pre-Lab:

1. Code — Singleton Pattern

```
package preLab_1;
class Prob1 {
    private static Prob1 myObject = new Prob1();
    static Prob1 getInstance(){
        return myObject;
    }
    private Prob1(){
    }
    void print() {
        System.out.println("Hello ! World");
    }
}
```

```
package preLab_1;
class Prob1Demo {
    public static void main(String [] args){
        Prob1 a= Prob1.getInstance();
        a.print();
    }
}
```

OUTPUT: Hello ! World

2. Code Builder Pattern

```
package preLab_2;

public interface Item{
        public String name();
        public Packaging packaging();
        public float price();
}

package prelab_2;

public abstract class Burgers implements Item{
       public Packaging packaging(){
              return new Wrapper();
       }
}
package preLab_2;
public class VegBurger extends Burgers{
       public float price(){
              return 40.0f;
       }
       public String name(){
              return "Veg Burger";
       }
}
package prelab_2;
public class ChickenBurger extends Burgers{
       public float price(){
              return 60.0f;
       }
```

```
        public String name(){
            return "Chicken Burger";
        }
 }
package preLab_2;
public abstract class ColdDrink implements Item{
      public Packaging packaging(){
             return new Bottle();
      }
}
package preLab_2;
public class Pepsi extends ColdDrink{
      public float Price(){
            return 25.0f;
      }
      public String name(){
            return "Pepsi";
      }
}
package preLab_2;
public class Coke implements ColdDrink{
      public float price(){
           return 20.0f;
      }
      public String name(){
           return "Coke";
      }
}
package preLab_2;
public interface Packaging{
      public String pack();
}
```

```
package preLab_2;
public class Bottle implements Packaging {
    public String pack(){
        return "Bottle";
    }
}
package preLab-2;
public class Wrapper implements Packaging {
    public String Pack(){
        return "Wrapper";
    }
}
package preLab_2;
import java.util.*;
public class Meal {
    private List<Item> i=new ArrayList<Item>();
    public void addItem (Item item){
        i.add (item);
    }
    public float get Cost () {
        float cost =0;
        for (Item i: i) {
            Cost += i.price ();
        }
        return cost;
    }
    public void display () {
        for (Item i:i) {
            System.out.println (" Item: "+ i.name ());
            System.out.println (", Packaging: "+i.Packaging ().pack());
            System.out.println (" Price: "+ i.price());
} } }
```

```
package prelab_2;
public class Meal Builder {
      public Meal Bill1(){
            Meal m= new Meal();
            m. addItem (new VegBurger());
            m. addItem (new Pepsi());
      }     return m;

      public Meal Bill2(){
          Meal m= new Meal();
          m.add Item (new Chicken Burger ());
          m. add Item (new Coke());
          return m;
      }
}

package pre_Lab2;
public class Demo {
      public static void main (String [] args){
          MealBuilder mealBuilder = new MealBuilder ();
          Meal vegMeal= mealBuilder. Bill1();
          System.out.println (" Veg Meal");
          VegMeal.display();
          System.out.println (" Total Cost: "+ vegMeal.getCost());

          Meal nonVegMeal = mealBuilder. Bill2();
          System.out.println ("\n\n Non-Veg Meal");
          nonVegMeal.display();
          System.out.println (" Total Cost: "+nonVegMeal.getCost()
      }
}
```

OUTPUT:

```
Veg Meal
Item : Veg Burger, Packaging : Wrapper Price: 40.0
Item : Pepsi, Packaging : Bottle Price: 25.0
Total Cost: 65.0

Non-Veg Meal
Item : Chicken Burger, Packaging : Wrapper Price: 60.0
Item : Coke, Packaging : Bottle Price : 20.0
Total Cost: 80.0
```

IN LAB

1. Bridge Pattern

```
package inLab_1;
public interface Student {
        public void addStudent (String name);
        public void deleteStudent (String name);
        public void display();
}

package inLab_1;
import java.util.*;
public class Students implements Student {
        List <String> list = new ArrayList <String>();
        public void addStudent (String name) {
                list.add(name);
                System.out.println (" Added "+ name);
        }
```

```
    public void deleteStudent(String name) {
        int i;
        for (i=0; i<list.size(); i++) {
            ~~if (i=0; i<list.size(); i++){~~
                ~~list.remove(i~~
            if (list.get(i).equals(name)) {
                list.remove(i);
                System.out.println("Deleted "+name);
                break;
            }
        }
        if (i== list.size())
            System.out.println(".Name not found");
    }
    public void display() {
        System.out.println("All Student Names: ");
        for (String i: list)
            System.out.println(i);
    }
}
package inLab_1;
public class StudentBridge {
    Private Student s= new Students()
    public void addStudent (String name) {
        s.addStudent(name);
    }
    public void deleteStudent (String name) {
        s.deleteStudent(name);
    }
    public void display() {
        s.display();
    }
}
```

```
package inLab_1;

public class BridgeFormat extends StudentBridge {
    public void display () {
        System.out.println ("--------------------\n");
        Super.display ();
        System.out.println ("--------------\n");
    }
}

package inLab_1;

public class Demo {

    public static void main (String [] args) {
        BridgeFormat bgf = new BridgeFormat ();
        bgf.addStudent (" Ajay");
        bgf.addStudent (" Bala");
        bgf.addStudent (" Cathey");
        bgf.addStudent (" Chella");
        bgf.addStudent ("Dolly");
        bgf.addStudent ("Ellan");
        bgf.addStudent ("Francis");
        bgf.addStudent (" Stella");
        bgf.display ();
        bgf.deleteStudent (" Chella");
        bgf.display ();
        bgf.addStudent (" Zara");
        bgf.display ();
    }
}
```

OUTPUT

```
All Student Names:
Ajay
```

```
Bala
Cathey
chella
Dolly
Ellan
Francis
Stella
- - - - - - - - - -
Deleted Chella
Added Zara
- - - - - - - - -
All Student Names:
Ajay
Bala
Cathey
Dolly
Ellan
Francis
Stella
zara
- - - - - - - - -
```

3. Criteria Design Pattern

```
package inLab_3;
public class Person {
    private String name, gender, maritalStatus;
    public Person (String name, String gender, String
                    maritalStatus){
        this.name = name;
        this.gender = gender;
        this.maritalStatus = maritalStatus;
    }
    public String getName(){
        return name;
    }
    public String getGender (){
        return gender;
    }
    public String getMaritalStatus (){
        return maritalStatus;
    }
    public String toString() {
        return "Person [Name="+ name+", Gender="+ gender
            +", MaritalStatus=" + maritalStatus +" ] \n";
    }
}
package inLab_3;
import java.util.ArrayList ;
public interface Criteria {
    public ArrayList <Person> getCriteria (ArrayList <Person>
                                    list);
}
package inLab_3;
import java.util.ArrayList;
public class Male implements Criteria {
    public ArrayList <Person> getCriteria (ArrayList
                <Person> list){
        ArrayList<Person> male=new ArrayList <Person>();
        for(Person i: list) {
            if (i.getGender().equalsIgnoreCase("male")) {
                male.add(i);
        }   }
    } return male;
}
```

```
Package inLab_3;
import java.util.ArrayList;
public class Female implements Criteria {
    public ArrayList <Person> getCriteria
        (ArrayList <Person> list) {
      ArrayList <Person> female=new ArrayList
                               <Person>();
      for (Person i:list) {
        if (i.getGender().equalsIgnoreCase("female")) {
            female.add(i);
        }
      }
      return female;
    }
}
package inLab_3;
import java.util.ArrayList;
public class Married implements Criteria {
    public ArrayList <Person> getCriteria (ArrayList
                <Person> list) {
      ArrayList <Person> married=new ArrayList
                               <Person>();
      for (Person i:list) {
        if (i.getMaritalStatus().equalsIgnoreCase
                             ("married")) {
            married.add(i);
      }   }
```

```
            return married;
        }
  }
package inLab_3;
import java.util.ArrayList;
public class Married implements Criteria{
     public ArrayList<Person> getCriteria(ArrayList
                    <Person> list) {
          ArrayList<Person> married = new ArrayList
                                   <Person>();
          for (Person i:list) {
               if (i.getMaritalStatus().equalsIgnoreCase
                    ("married")) {
                        married.add(i);
          }    }
          return married;
     }
}
package inLab_3;
Import java.util.ArrayList;
public class NotMarried implements Criteria {
     public ArrayList<Person> get Criteria (ArrayList
                    <Person> list) {
      ArrayList<Person> notMarried = new ArrayList<Person>();
      for (Person i:list) {
           if (i.getMaritalStatus().equalsIgnoreCase
                           ("not married")) {
```

```
                notMarried.add(i);
              }
           }
           return notMarried;
     }
}
package inLab_3;
import java.util.ArrayList;
public class Demo{
     public static void main(String[] args){
           ArrayList<Person> list=new ArrayList<Person>
                                                       ();
           list.add(new Person("Robert", "Male", "Not Married"));
           list.add(new Person("John", "Male", "Married"));
           list.add(new Person("Mike", "Male", "Not Married"));
           list.add(new Person("Bobby", "Male", "Not Married"));
           list.add(new Person("Laura", "Female", "Married"));
           list.add(new Person["Diana", "Female", "Not Married"];
           Criteria maleCriteria=new Male();
           System.out.println("Males:\n"+maleCriteria.getCriteria
                                                      (list));
           Criteria femaleCriteria=new Female();
           System.out.println("Females:\n"+femaleCriteria.
                                       getCriteria(list));
           Criteria marriedCriteria=new Married();
           System.out.println("Married:\n"+marriedCriteria.
                                  getCriteria(list));
           Criteria nonMarriedCriteria=new NotMarried();
           System.out.println("Not Married:\n"+nonMarriedCriteria
}    }                 .getCriteria(list));
```

OUTPUT

```
Males:
[Person [Name= Robert, Gender=Male, Marital Status=NotMarri
                                                          ed]
, Person [Name= John, Gender=Male, Marital Status = Married]
, Person [Name= Mike, Gender = Male, Marital Status= Not Married]
, Person [Name= Bobby, Gender = Male, Marital Status = NotMarried]
]
Females:
[ Person [Name = Laura, Gender = Female, Marital Status= Married],
  Person [Name= Diana, Gender= Female, Marital Status= Not Married]
]
Married:
[Person [Name= John, Gender = Male, Marital Status = Married]
, Person [Name= Laura, Gender = Female, Marital Status = Married]
]

Not Married:
[Person [Name=Robert, Gender= Male, Marital Status= Not Married]
, Person [Name= Mike, Gender=Male, Marital Status= Not Married]
, Person [Name= Bobby, Gender= Male, Marital Status= Not Married]
, Person [Name= Diana, Gender= Female, Marital Status= NotMarried]
]
```

POST LAB

1. Command Pattern

```
package postLab_1;
public interface Command {
    public void execute();
}
package postLab_1;
public class Light {
    public void on() {
        System.out.println("Light is on");
    }
    public void off() {
        System.out.println("Light is off");
    }
}
package postLab_1;
public class LightOff implements Command {
        Light light;
        public LightOff (Light light) {
            this.light = light;
        }
        public void execute() {
            light.off ();
        }
}
package postLab_1;
public class LightOn implements Command {
        Light light;
        public LightOn (Light light){
            this.light = light;
        }
        public void execute(){
            light.On();
        }
}
package postLab_1;
public class Stereo{
        public void on(){
            System.out.println("Stereo is on");
        }
        public void off(){
            System.out.println("Stereo is off");
        }
        public void setCD(){
            System.out.println("Stereo is set for CD input");
        }
        public void setVolume(int volume){
            System.out.println("Stereo volume set to"+ volume);
        }
}
package postLab_1;
public class StereoOff implements Command{
        Stereo stereo;
        public StereoOff(Stereo stereo){
            this.stereo= stereo;
        }
        public void execute(){
            stereo.off();
        }
}
package postLab_1;
public class StereoOn implements Command{
        Stereo stereo;
        public StereoOn(Stereo stereo){
                this.stereo= stereo;
        }
        public void execute(){
                Stereo.on();
                stereo.set(DC();
                stereo.setVolume(11);
        }
}
package postLab_1;
public class RemoteControl{
    Command click;
    public void setCommand(Command command){
            click = command;
    }
    public void button(){
            click.execute();
    }
}
package postLab_1;
public class Demo{
    public Static void main(String[] args){
            RemoteControl remote = new RemoteControl();
            Light light = new Light();
            Stereo stereo = new Stereo();
            remote.setCommand(new LightOn(light));
            remote.button();
            remote.setCommand(new StereoOn(stereo));
            remote.button();
            remote.setCommand(new StereoOff(stereo));
            remote.button():
    }
}
```

OUTPUT

```
Light is on
Stereo is on
Stereo is set for CD input
Stereo volume set to 11
Stereo is off
```