ID.NO:2100030045

JAVA LAB 01-SOLUTIONS

1. 1. Write a Java Program to print “Hello ! world” using Singleton pattern SOL:

```
public class SingleObject {

   private static SingleObject instance = new SingleObject();

   private SingleObject(){}  public static
SingleObject getInstance(){       return
instance;
   }
 public void showMessage(){
      System.out.println("Hello World!");
   }
}
public class SingletonPatternDemo {
public static void main(String[] args) {
 SingleObject object = SingleObject.getInstance();
object.showMessage();
   }
}
```

2. Consider a business case of fast-food restaurant where a typical meal could be a burger and a cold drink. Burger could be either a Veg Burger or Chicken Burger and will be packed by a wrapper. Cold drink could be either a coke or pepsi and will be packed in a bottle.
   Draw a Builder pattern UML diagram for this case after listing the objects, classes , interfaces and methods needed for this use case.

SOL.

```
public interface Item {
public String name();
public Packing packing();
public float price();
}
public interface Packing {
public String pack();
}
public class Wrapper implements Packing {

  @Override    public
String pack() {
return "Wrapper";
  }
}
public class Bottle implements Packing {

  @Override    public
String pack() {
return "Bottle";
  }
}
public abstract class Burger implements Item {

  @Override    public
Packing packing() {
return new Wrapper();
  }
```

```
   @Override
   public abstract float price();
}

public abstract class ColdDrink implements Item {

        @Override
        public Packing packing() {
      return new Bottle();
        }

        @Override
        public abstract float price();
}

public class VegBurger extends Burger {

   @Override
   public float price() {
return 25.0f;
   }

   @Override    public
String name() {
return "Veg Burger";
   }
}

public class ChickenBurger extends Burger {
```

```
    @Override
    public float price() {
        return 50.5f;
    }

    @Override    public String
name() {        return
"Chicken Burger";
    }
}

public class Coke extends ColdDrink {

    @Override
    public float price() {
return 30.0f;
    }

    @Override    public
String name() {
return "Coke";
    }
}

public class Pepsi extends ColdDrink {

    @Override
    public float price() {
return 35.0f;
    }
```

```
   @Override
   public String name() {
return "Pepsi";
   }
   import java.util.ArrayList; import
java.util.List;

public class Meal {    private List<Item> items =
new ArrayList<Item>();

   public void addItem(Item item){
items.add(item);
   }

   public float getCost(){
float cost = 0.0f;

      for (Item item : items) {
cost += item.price();
      }
      return cost;
   }

   public void showItems(){

      for (Item item : items) {
         System.out.print("Item : " + item.name());
         System.out.print(", Packing : " + item.packing().pack());
         System.out.println(", Price : " + item.price());
      }
```

```
}
}

public class MealBuilder {

   public Meal prepareVegMeal (){
Meal meal = new Meal();
meal.addItem(new VegBurger());
meal.addItem(new Coke());       return
meal;
   }

   public Meal prepareNonVegMeal (){
Meal meal = new Meal();
meal.addItem(new ChickenBurger());
meal.addItem(new Pepsi());       return
meal;
   }
}

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

      MealBuilder mealBuilder = new MealBuilder();

      Meal vegMeal = mealBuilder.prepareVegMeal();
System.out.println("Veg Meal");
vegMeal.showItems();
```

```
    System.out.println("Total Cost: " + vegMeal.getCost());        Meal nonVegMeal =
mealBuilder.prepareNonVegMeal();          System.out.println("\n\nNon-Veg  Meal");
nonVegMeal.showItems();
    System.out.println("Total Cost: " + nonVegMeal.getCost());
  }
}
```

In-Lab:

1. Implement the following scenario by **Bridge pattern** in Java :

   Add the student names into the class register like “Ajay”, “Bala”, “Cathey”,”Chella”,”Dolly”, ”Ellan”, Francis”, ”Stella”.

   Do the following operations on the class register

   i. Display all student names
   ii. Delete “chella” from the register
   iii. Display previous and next names from the register.
   iv. Add “Zara” into the register
   v. Display all student names after addition of “Zara”

SOL:

2. Birthday : Interface that decribes methods to access the year, month, and day fields of a birthday, as well as a few methods to compare two Birthday instances.

   BirthdayFactory : Abstract factory interface for the creation of Birthday instances.

   BirthdayClient : Client class that gets uses a BirthdayFactory object to create and compare a few Birthday instances.

   Your task is to make the GregorianCalendar Java library class usable as an implementation of the Birthday interface. Because GregorianCalendar cannot be modified, you will need to write an adapter class.

   (a) Implement an object **adapter** DateObjectAdapter that adapts the given class GregorianCalendar to the new Birthday interface. Write a concrete factory class DateObjectAdapterFactory that implements BirthdayFactory.

   (b) Similarly, implement a class adapter DateClassAdapter and a factory class DateClassAdapterFactory.

   (c) Run the client class BirthdayClient.

   SOL:

```
 package
exercise4_1;
 public interface Birthday
{     int getYear();
Month getMonth();
```

```
    int getDay();
         boolean isLaterThan(Birthday other); // refers to year, month,
and day     boolean isSame(Birthday other); // refers to month and day
only

    public enum Month {
    January, February, March, April, May, June, July, August,
    September, October, November, December;
    }
     } package
exercise4_1;
 import solution4_1.DateClassAdapterFactory;
import solution4_1.DateObjectAdapterFactory;
 public class BirthdayClient
{
     private final BirthdayFactory
factory;

    BirthdayClient(BirthdayFactory factory) {
this.factory = factory;
    }
    public void run() {
    Birthday myBirthday = factory.getBirthday(1965, Birthday.Month.December,
28);
    Birthday otherBirthday = factory.getBirthday(2001,
Birthday.Month.December, 28);
    Birthday thirdBirthday = factory.getBirthday(1987, Birthday.Month.April,
1);

    System.out.println("myBirthday: " + myBirthday);
    System.out.println("otherBirthday: " + otherBirthday);
    System.out.println("thirdBirthday: " + thirdBirthday);

    System.out.println("myBirthday.isLaterThan(otherBirthday): " +
myBirthday.isLaterThan(otherBirthday));
    System.out.println("myBirthday.isLaterThan(thirdBirthday): " +
myBirthday.isLaterThan(thirdBirthday));
    System.out.println("otherBirthday.isLaterThan(thirdBirthday): " +
otherBirthday.isLaterThan(thirdBirthday));
    System.out.println("myBirthday.isSame(otherBirthday): " +
myBirthday.isSame(otherBirthday));
    System.out.println("myBirthday.isSame(thirdBirthday): " +
myBirthday.isSame(thirdBirthday));
    System.out.println("otherBirthday.isSame(thirdBirthday): " +
otherBirthday.isSame(thirdBirthday));

    }           public static final void
main(String[] args) {
```

```
    new BirthdayClient(DateObjectAdapterFactory.INSTANCE).run();
System.out.println("---------------------------------");     new
BirthdayClient(DateClassAdapterFactory.INSTANCE).run();
    }


}  package
exercise4_1;


public interface BirthdayFactory {
    Birthday getBirthday(int year, Birthday.Month month, int day);
}
```

3.Design a Criteria Design pattern for the following output:

Person : [Name : Robert, Gender : Male, Marital Status : NotMarried]
Person : [Name : John, Gender : Male, Marital Status : Married]
Person : [Name : Mike, Gender : Male, Marital Status : NotMarried]
Person : [Name : Bobby, Gender : Male, Marital Status : NotMarried]
Person : [Name : Laura, Gender : Female, Marital Status : Married]
Person : [Name : Diana, Gender : Female, Marital Status : NotMarried]

Males:
Person : [Name : Robert, Gender : Male, Marital Status : NotMarried]
Person : [Name : John, Gender : Male, Marital Status : Married]
Person : [Name : Mike, Gender : Male, Marital Status : NotMarried]
Person : [Name : Bobby, Gender : Male, Marital Status : NotMarried]

Females:
Person : [Name : Laura, Gender : Female, Marital Status : Married]
Person : [Name : Diana, Gender : Female, Marital Status : NotMarried]

NotMarried:
Person : [Name : Robert, Gender : Male, Marital Status : NotMarried]
Person : [Name : Mike, Gender : Male, Marital Status : NotMarried]
Person : [Name : Bobby, Gender : Male, Marital Status : NotMarried]
Person : [Name : Diana, Gender : Female, Marital Status : NotMarried]

Married:
Person : [Name : John, Gender : Male, Marital Status : Married]
Person : [Name : Laura, Gender : Female, Marital Status : Married]

public class Person

```
{

 private String name;

private String gender;

private String 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;
```

```
}
}
```

Criteria.java

```
import java.util.List;

public interface Criteria
{  public List<Person> meetCriteria(List<Person>
persons);
}
```

MaleCriteria.java

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

public class MaleCriteria implements Criteria
{
// Return the List of Persons who are male
@Override public List<Person> meetCriteria(
List<Person> persons ) {

List<Person> malePersons = new ArrayList<Person>(); for(
Person person : persons )
```

```
{  if(
person.getGender().equalsIgnoreCase("male") )
 {
 malePersons.add(person);
 }
 }  return
malePersons;
 }
}
```

FemaleCriteria.java

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

public class FemaleCriteria implements Criteria
{

// Return the List of Persons who are female
@Override public List<Person> meetCriteria(
List<Person> persons )
{
List<Person> femalePersons = new ArrayList<Person>(); for(
Person person : persons )
```

```
{ if( person.getGender().equalsIgnoreCase("female")
)
 {
 femalePersons.add(person);
 }
 }  return
femalePersons;
 }
}
```

MarriedCriteria.java

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

public class MarriedCriteria implements Criteria
{
 // Return the List of Persons who are Married
 @Override public List<Person> meetCriteria(
 List<Person> persons )
{
List<Person> marriedPersons = new ArrayList<Person>(); for(
Person person : persons )

{ if( person.getMaritalStatus().equalsIgnoreCase("Married")
)
```

```
{ marriedPersons.add(person);
 }
 }  return
marriedPersons;
 }
}
```

NotMarriedCriteria.java

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

public class NotMarriedCriteria implements Criteria
{

 // Return the List of Persons who are not Married
 @Override  public List<Person> meetCriteria(
List<Person> persons )
{
List<Person> notMarriedPersons = new ArrayList<Person>(); for(
Person person : persons )
{
if( person.getMaritalStatus().equalsIgnoreCase("notMarried") )
{ notMarriedPersons.add(person);
```

```
 } } return
notMarriedPersons;
 }
}
```

CriteriaOrCondition.java

```
import java.util.List;

public class CriteriaOrCondition implements Criteria
{  private Criteria criteria; /* male or female or married or notMarried Criteria  */
private Criteria otherCriteria; /* male or female or married or notMarried Criteria  */

 public CriteriaOrCondition( Criteria criteria, Criteria otherCriteria )
 {  this.criteria = criteria;
this.otherCriteria = otherCriteria;
}

// This will perform OR operation of two Criteria results
@Override public List<Person> meetCriteria(
List<Person> persons )
{
List<Person> firstCriteriaItems = criteria.meetCriteria(persons);
 List<Person> otherCriteriaItems = otherCriteria.meetCriteria(persons);
```

```
 for( Person person : otherCriteriaItems )
 {  if(
!firstCriteriaItems.contains(person) )
 {
firstCriteriaItems.add(person);
 }  }  return
firstCriteriaItems;
 }
}
```

CriteriaAndCondition.java

```
import java.util.List;

public class CriteriaAndCondition implements Criteria
{

private Criteria criteria;   /* male or female or married or notMarried Criteria  */ private Criteria
otherCriteria; /* male or female or married or notMarried Criteria  */

public CriteriaAndCondition( Criteria criteria, Criteria otherCriteria )
 {  this.criteria = criteria;
this.otherCriteria = otherCriteria;
 }

  // This will perform AND operation of two Criteria results
```

```
 @Override  public List<Person> meetCriteria(
List<Person> persons )
 {
 List<Person> firstCriteriaPersons = criteria.meetCriteria(persons);
return otherCriteria.meetCriteria(firstCriteriaPersons);
 }
}
```

CriteriaPatternDemo.java

```
public class CriteriaPatternDemo
{  public static void main(String[]
args)
 {
List<Person> persons = new ArrayList<Person>();
 persons.add(new Person("Robert", "Male", "NotMarried"));
persons.add(new Person("John", "Male", "Married")); persons.add(new
Person("Mike", "Male", "NotMarried")); persons.add(new
Person("Bobby", "Male", "NotMarried"));  persons.add(new
Person("Laura", "Female", "Married"));  persons.add(new
Person("Diana", "Female", "NotMarried"));

 printPersons(persons);

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

```
Criteria male = new MaleCriteria();

System.out.println("Males: ");
printPersons(male.meetCriteria(persons));

Criteria female = new FemaleCriteria();

System.out.println("\nFemales: ");
printPersons(female.meetCriteria(persons));

Criteria notMarried = new NotMarriedCriteria();

System.out.println("\nNotMarried: "); printPersons(notMarried.meetCriteria(persons));
Criteria married = new MarriedCriteria();

System.out.println("\nMarried: "); printPersons(married.meetCriteria(persons));

Criteria marriedMale = new CriteriaAndCondition(married, male);

System.out.println("\nMarried and Male: ");
printPersons(marriedMale.meetCriteria(persons));

Criteria notMarriedOrFemale = new CriteriaOrCondition(notMarried, female);
```

```
System.out.println("\nNotMarried Or Female");

printPersons(notMarriedOrFemale.meetCriteria(persons));

}


public static void printPersons(List<Person> persons)

{  for (Person person :

persons)

{

System.out.println("Person : [ Name : " + person.getName() + ", Gender : " +
person.getGender() + ", Marital Status : " + person.getMaritalStatus() + " ]");

}

}

}
```

Post-Lab:

1. Build a home automation system where a programmable remote can be used to turn on and off various items in your home like lights, stereo, etc. Keep in mind that turning on some devices like stereo comprises of many steps like setting cd, volume etc. Implement this system by **command pattern**. **Output:**

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

SOL: 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 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 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);
    }
}

class SimpleRemoteControl
{
   Command slot;

   public SimpleRemoteControl()
   {
   }

   public void setCommand(Command command)
   {
      slot = command;
   }
```

```
    public void buttonWasPressed()
    {
slot.execute();
    }
}

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

        Stereo stereo = new Stereo();

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