1) Public class Singleobject{
Private static Singleobject s=new Singleobject();
Private singleobject(){}
Public static Singleobject getinstance(){
return s;
}
Public void showMessage(){
System.out.println("Helloworld");
}
}
Public class singletonPatternDemo{
Public static void main(String[]args){
Singleobject.getinstance().showmessage();
}
}

2) Public interface item{
Public String name();
Public Packing packing();
Public float price();
}
Public interface Packing{
Public String pack();
}
Public classer Wrapper implements Packing{
Public string pack(){
return "wrapper";
}
}

```
Public class Bottle implements Packing{
    Public String Pack(){
        return "Bottle";
    }
}

Public abstract class Burger implements item{
    Public Packing Packing(){
        return new Wrapper();
    }
    Public abstract float price();
}

Public abstract class Coldriuk implements item{
    Public Packing Packing(){
        return new Bottle();
    }
    Public abstract float Price();
}

Public class VegBurger extends Burger{
    public float price(){
        return 25.0f;
    }
    public String name(){
        return "Veg Burger";
    }
}
```

```
Public class ChickenBurger extends Burger{
  Public float price(){
      return 50.5f;
  }
  Public String name(){
      return "Chicken Burger";
  }
}
Public class Coke extends ColdDrink{
  Public float price(){
      return 30.0f;
  }
  Public String name(){
      return "Coke";
  }
}
Public class Pepsi extends ColdDrink{
  Public float price(){
      return 30.0f;
  }
  Pubic String name(){
      return "Pepsi";
  }
}
```

```
Public class Meal {
  Private List <item> items = new ArrayList <item>();
  public void additem (item item){
      items.add(item);
    }
  Public float getCost(){
      float cost = 10.0f;
      for(item item : items){
        cost += item.Price();
      }
      return cost;
    }
  Public void showitems(){
    for(item item: items){
      System.out.println("item:"+item.name());
      System.out.println("packing"+item.Packing().Pack());
      System.out.println("Price"+item.Price());
    }
  }
}
```

```
ublic 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\n Non-Veg Meal");
    nonVegMeal.showitems();
    System.out.println ("Total Cost "+nonvegMeal.getCost());
}
```

In Lab.

2)
```
Public interface Birthday {
    int getYear();
    Month getMonth();
    int getDay();
    boolean isLaterThan(Birthday other);
    boolean issame(Birthday other);

Pubic enum Month {
  January , Febuary, March, April, May, June, July, August, September,
  October, November, December;
}
}

import . Solution4_1' Date classAdapterFactory;
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

Birthday otherBirthday = factory · getBirthday(2001, Birthday·
                                              MonthDecember2
```

```
Birtuday thirBirtuday = factory.getBirthday(1987, Birtuday.Month.April);
System.out.println("myBirtuday:" + myBirtuday);
System.out.println("otherBirtuday:" + otherBirtuday);
system.out.println("thirdBirtuday:" + thirdBirtuday);
system.out.println("myBirtuday.isLaterThan(otherBirtuday):" +
    myBirtuday.isLaterThan(otherBirtuday));
System.out.println("myBirtuday.isLaterThan(thirdBirtuday):" +
    myBirtuday.isLaterThan(thirdBirthday));
System.out.println("myBirtuday.isSame(otherBirtuday):" +
    myBirtuday.isSame(otherBirtuday));
system.out.println("myBirtuday.isSame(otherBirtuday):" +
    myBirtuday.isSame(thirdBirtuday));
}
Public static final void main(String[] args){
new BirtudayClient(DateobjectAdapterFactory.INSTANCE).run();
System.out.println("-----");
new BirtudayClient(DateclassAdapterFactory.INSTANCE).run();
}
public Interface BirtudayFactory{
  Birtuday getBirtuday(int year, Birtuday.Month month, int day);
}
```

3) 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;
}
}
Public interface Criteria
{ Public List<Person> meetCriteria(List<Person>Person);
}
```

```
public class MaleCriteria implements Criteria
{
 public List<Person> meetCriteria(List<Person>Persons)
 {List<Person> malePerson= new ArrayList<Person>();
  for(Person person: Persons)
  { if(Person.getGender().equalsIgnoreCase("male"))
    { malePersons.add(Person);
    }
  }
   return malePersons;
 }
}

public class FemaleCriteria implements Criteria
 { public List<Person>meetCriteria(List<Person>Persons)
  { List<person> femalePersons=new ArrayList<Person>();
   for(Person person:Persons)
   { if(person.getGender():equalsIgnoreCase("female"))
    {
      femalePerson.add(Person);
    }
   }
   return femalePersons;
 }
}
```

```
Public class MarriedCriteria implements Criteria.
{
  public List<Person> meetCriteria(List<Person> persons)
  {
    List<Person> marriedPerson = new ArrayList<Person>();
    for (Person person: persons)
    {
      if (person.getMaritalStatus().equalsIgnoreCase("Married
      {
        marriedPersons.add(person);
      }
    }
    return marriedPersons;
  }
}

Public class NotMarriedCriteria {
Public List<person> MeetCriteria(List<Person>persons),
{
  List<Person> notMarriedPersons = new ArrayList<Person>();
  for (Person person: persons)
  {
    if (person.getMaritalStatus().equalsIgnoreCase("not Married"
    {
      notMarriedPersons.add(person);
    }
    return notMarriedPersons;
  }
}
```

```
public class CriteriaAndCondition implements Criteria
{ private Criteria criteria;
  private Criteria otherCriteria;
  Public CriteriaAndCondition(Criteria criteria, Criteria otherCriteria).
  { this.criteria = criteria;
    this.otherCriteria = otherCriteria;
  }
Public List<Person> meetCriteria(List<Person> persons)
{ List<Person> firstCriteriaPersons = criteria.meetCriteria(persons);
  return otherCriteria.meetCriteria(firstCriteriaPersons);
  }
}
Public class CriteriaPatternDemo
{ public static void main(String[] args) {
  List<Person> persons = new ArrayList<Person>();
  persons.add(new Person("Robert", "Male", "Not Married"));
  persons.add(new Person("John", "Male", "Married"));
  persons.add(new Person("Mike", "Male", "Not Married"));
  persons.add(new Person("BOBBY", "Male", "Not Married"));
  persons.add(new Person("Laura", "Female", "Married"));
  persons.add(new Person("Diana", "Female", "Not Married"));

  printPersons(persons);

  Criteria male = new MaleCriteria();
  System.out.println("Males:");
```

```
PrintPersons(male.meetCriteria(persons));

Criteria female = new FemaleCriteria();
System.out.println("Female s:");
Criteria NotMarried = new NotMarriedCriteria();
printPersons(notMarried.meetCriteria(persons));

Criteria married = new MarriedCriteria();
System.out.println(married.meetCriteria(persons));

Criteria NotMarriedOfFemale = new CriteriaOr(notMarried, female);

PrintPerson(notMarriedOrFemale.meetCriteria(persons));
}

Public static void printPersons(List<Person> person)
{
for(Person person: person)
{
System.out.println("Persons: [Name:" + person.getName() + "Gender:" +
person.getGender() + "MaritalStatus:" + person.getMaritalStatus
}
}
}
```

Post-Lab

1) 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)
{
```

```
System.out.println("stereo volume at" + "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 StereoWithCDCommand(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());}
    }
}
```