Slip-1

Q1 Write a Java Program to implement I/O Decorator for converting uppercase letters to lower case letters.

Program Name:- vi uplo.java

```
import java.io.*;
import java.util.*;

class LowerCaseInputStream extends FilterInputStream
{
	public 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;
	}
}
class uplo
{
	public static void main(String[] args) throws IOException
	{
	int c;
	try
	{

		InputStream in =new LowerCaseInputStream(new BufferedInputStream(new
		FileInputStream("a.txt")));
		while((c = in.read()) >= 0)
		{
			System.out.print((char)c);
		}

		in.close();
	}
	catch(IOException e)
	{
		e.printStackTrace();
	}
```

```
        }
}
```
OUTPUT:-
```
vi a.txt
Harshada
HARSHADA
TEJU
Tejaswi
Tejaswini
```

Run:-
```
student@localhost ~]$ javac uplo.java
[student@localhost ~]$ java uplo
harshada
harshada
teju
tejaswi
tejaswini
```

Q2 Write a Python program to prepare Scatter Plot for Iris Dataset.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
iris = pd.read_csv("Iris.csv")   # Reading the dataset “Iris.csv”.
print (iris.head(10))   # head() will display the top rows of the dataset, the default value of this function is 5,
#that is it will show top 5 rows when no argument is given to it.
 plt.plot(iris.Id, iris["SepalLengthCm"],"r--")
plt.show    #plt.show () will display the current figure that you are working on
iris.plot(kind ="scatter", x ='SepalLengthCm', y ='PetalLengthCm')
plt.grid()    # grid () function to add grid lines to the plot
```

Q3 Create an HTML form that contain the Student Registration details and write a JavaScript to validate Student first and last name as it should not contain other than alphabets and age should be between 18 to 50.

```
<html>
<head>
   <script type="text/javascript">
   function validate()
   {
      var regName=/^[a-zA-z]+[a-zA-Z]+$/;
      var fname=document.getElementById("txtfname").value;
      var lname=document.getElementById("txtlname").value;
      var age=document.getElementById("txtage").value;
      var mobno=document.getElementById("txtmobno").value;

      if(age<18||age>50)
         alert("student age must be 18 to 50");
```

```
        if(!regName.test(fname))
            alert("invalid name is given");
        else
            alert("valid name is given");
    }
    </script>
    </head>
    <body>
    <form>
        enter student first name
        <input type="text" name="txtfname" id="txtfname"><br>
        enter student last name
        <input type="text" name="txtlname" id="txtlname"><br>
        enter student age
        <input type="text" name="txtage" id="txtage"><br>
        enter mobile no
        <input type="text" name="txtmobno" id="txtmobno"><br>
        <input type="button" value="validate" onclick="validate()">
    </form>
    </body>
    </html>
```

Slip-2
Q1

```
import java.io.*;
import java.util.*;

class slip2Q1
{
  private static slip2Q1 instance = new slip2Q1();
  private slip2Q1(){}
  public static slip2Q1 getInstance()
  {
    return instance;
  }

  public void showMessage()
  {
    System.out.println("Hello Everyone!!!!");
  }
}

class slip21
{
  public static void main(String[] args)
  {
    slip2Q1 object = slip2Q1.getInstance();
    object.showMessage();
  }
```

}

Q2 Write a python program to find all null values in a given dataset and remove them.

```
import pandas as pd
import numpy as np
dict={'first score':[100,90,np.nan,95],
     'second score':[30,45,56,np.nan],
     'third score':[np.nan,40,80,98]}
df=pd.DataFrame(dict)
df.head()
df.isnull()
df.notnull()
#df=pd.DataFrame(dict)
df.fillna(0)
df.fillna(method='pad')
df.fillna(method='bfill')
df.replace(to_replace=np.nan,value=-99)
df.dropna()
df.dropna(axis=1)
new_data=df.dropna(axis=0)
new_data
```

3) Create an HTML form that contain the Employee Registration details and write a JavaScript to validate DOB, Joining Date, and Salary.

```
<html>
   <head>
      <script type="text/javascript">
      function validate()
      {

      var regName=/^[a-zA-z]+[a-zA-Z]+$/;
      var dateformatdob = /^(0?[1-9]|[12][0-9]|3[01])[\/\-](0?[1-9]|1[012])[\/\-]\d{4}$/;
      var dateformatjdate = /^(0?[1-9]|[12][0-9]|3[01])[\/\-](0?[1-9]|1[012])[\/\-]\d{4}$/;


     //Max six digits, a dot, max two digits after dot

    var salaryformat=/^\d{1,6}(?:\.\d{0,2})?$/

      var name=document.getElementById("txtname").value;
      var dob=document.getElementById("txtdob").value;
      var jdate=document.getElementById("txtjdate").value;
      var salary=document.getElementById("txtsalary").value;

      if(!regName.test(name))
         alert("invalid name is given");
      else
```

```
        alert("valid name is given");

      if(!dateformatjdate.test(jdate))
        alert("invalid joining date is given");
      else
        alert("valid joining date is given");
      if(!dateformatdob.test(dob))
        alert("invalid date of birth is given");
      else
        alert("valid date of birth is given is given");
      if(!salaryformat.test(salary))
        alert("invalid salary");
      else
        alert("salary is valid");
      }
      </script>

  </head>
<body>
<form>
  <h1>Employee Rsgistration Details</h1>
    enter employee first name
    <input type="text" name="txtfname" id="txtname"><br>

    enter date of birth
    <input type="text" name="txtdob" id="txtdob"><br>
    enter joining date
    <input type="text" name="txtjdate" id="txtjdate"><br>
    enter salary
    <input type="text" name="txtsalary" id="txtsalary"><br>
    <input type="button" value="validate" onclick="validate()">
  </form>
  </body>
  </html>
```

Slip-3

Q1 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()

```
import java.util.*;

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

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

```
}

interface Subject {
   public void registerObserver(Observer o);

   public void removeObserver(Observer o);

   public void notifyObservers();
}

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

   public WeatherData() {
      observers = new ArrayList<>();
   }

   public void registerObserver(Observer o) {
      observers.add(o);
   }

   public void removeObserver(Observer o) {
      int i = observers.indexOf(o);
      if (i >= 0) {
         observers.remove(i);
      }
   }

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

   public void measurementsChanged() {
      notifyObservers();
   }

   public void setMeasurements(float temperature, float humidity, float pressure) {
      this.temperature = temperature;
      this.humidity = humidity;
      this.pressure = pressure;
      measurementsChanged();
   }

   public float getTemperature() {
```

```
        return temperature;
    }

    public float getHumidity() {
        return humidity;
    }

    public float getPressure() {
        return pressure;
    }
}

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

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

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

        display();
    }

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

class HeatIndexDisplay implements Observer, DisplayElement {
    float heatIndex = 0.0f;
    private WeatherData weatherData;

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

```
    public void update(float t, float rh, float pressure) {
        heatIndex = computeHeatIndex(t, rh);
        display();
    }

    private float computeHeatIndex(float t, float rh) {
        float index = (float) ((16.923 + (0.185212 * t) + (5.37941 * rh) - (0.100254 * t * rh)
                + (0.00941695 * (t * t)) + (0.00728898 * (rh * rh))
                + (0.000345372 * (t * t * rh)) - (0.000814971 * (t * rh * rh)) +
                (0.0000102102 * (t * t * rh * rh)) - (0.000038646 * (t * t * t)) + (0.0000291583 *
                        (rh * rh * rh))
                + (0.00000142721 * (t * t * t * rh)) +
                (0.000000197483 * (t * rh * rh * rh)) - (0.0000000218429 * (t * t * t * rh * rh)) +
                0.000000000843296 * (t * t * rh * rh * rh)) -
                (0.0000000000481975 * (t * t * t * rh * rh * rh)));
        return index;
    }

    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;
    private WeatherData weatherData;

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

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

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

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

        display();
    }
```

```
    public void display() {
        System.out.println("Avgerage/Maximum/Minimum temperature = " + (tempSum /
numReadings)
                + "/" + maxTemp + "/" + minTemp);
    }
}

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

    public CurrentConditionsDisplay(Subject weatherData) {
        this.weatherData = weatherData;
        weatherData.registerObserver(this);
    }

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

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

class weather {

    public static void main(String[] args) {
        WeatherData weatherData = new WeatherData();

        CurrentConditionsDisplay currentDisplay = new
CurrentConditionsDisplay(weatherData);
        StatisticsDisplay statisticsDisplay = new StatisticsDisplay(weatherData);
        ForecastDisplay forecastDisplay = new ForecastDisplay(weatherData);

        weatherData.setMeasurements(70, 55, 40.4f);
        weatherData.setMeasurements(72, 60, 39.2f);
        weatherData.setMeasurements(68, 80, 39.2f);
    }
}
```

2) Write a python program to make Categorical values in numeric format for a given dataset.

```
import pandas as pd
```

```
#load Iris data set
iris = pd.read_csv('Iris.csv')
iris.head()
iris['code']=pd.factorize(iris.Species)[0]
iris.tail()
iris.code.value_counts()
```

3) Create an HTML form for Login and write a JavaScript to validate email ID using Regular Expression.

```
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
   </style>
</head>
<body>
<script>
function validateform(){
var email = document.getElementById("email").value;
var password = document.getElementById("psw").value;
if (/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/.test(email))
  {
    alert("Valid Email Id..")
    return (true)
  }
   else{ alert("You have entered an invalid email address!")
    return (false)
   }
}
</script>
<form name="myform" onsubmit="return validateform()">
  <div class="container">

    <p>Please fill in this form to Login.</p>
    <hr>

    <label for="email"><b>Email</b></label>
    <input type="text" autocomplete="off" placeholder="Enter Email" name="email"
id="email" required>

    <label for="psw"><b>Password</b></label>
    <input type="password" autocomplete="off" placeholder="Enter Password" name="psw"
id="psw" required>

    <hr>
    <button type="submit" class="registerbtn">Register</button>
  </div>

</form>
```

```
</body>
</html>
```

Slip-4

Q1 Write a Java Program to implement Factory method for Pizza Store with createPizza(), orederPizza(), prepare(), Bake(), cut(), box(). Use this to create variety of pizza's like NyStyleCheesePizza, ChicagoStyleCheesePizza etc.

```
// Abstract Pizza class
abstract class Pizza {
    String name;

    public void prepare() {
        System.out.println("Preparing " + name);
    }

    public void bake() {
        System.out.println("Baking " + name);
    }

    public void cut() {
        System.out.println("Cutting " + name);
    }

    public void box() {
        System.out.println("Boxing " + name);
    }
}

// Concrete Pizza classes
class NyStyleCheesePizza extends Pizza {
    public NyStyleCheesePizza() {
        name = "NY Style Cheese Pizza";
```

```
    }
}

class ChicagoStyleCheesePizza extends Pizza {
    public ChicagoStyleCheesePizza() {
        name = "Chicago Style Cheese Pizza";
    }
}

// Abstract PizzaStore class with Factory Method
abstract class PizzaStore {
    public Pizza orderPizza(String type) {
        Pizza pizza = createPizza(type);

        pizza.prepare();
        pizza.bake();
        pizza.cut();
        pizza.box();

        return pizza;
    }

    protected abstract Pizza createPizza(String type);
}

// Concrete PizzaStore classes
class NYPizzaStore extends PizzaStore {
    @Override
    protected Pizza createPizza(String type) {
        if (type.equals("cheese")) {
            return new NyStyleCheesePizza();
        }
        // Add more pizza types here as needed
```

```
        return null;
    }
}

class ChicagoPizzaStore extends PizzaStore {
    @Override
    protected Pizza createPizza(String type) {
        if (type.equals("cheese")) {
            return new ChicagoStyleCheesePizza();
        }
        // Add more pizza types here as needed
        return null;
    }
}

public class PizzaTestDrive {
    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.name);

        pizza = chicagoStore.orderPizza("cheese");
        System.out.println("Joel ordered a " + pizza.name);
    }
}
```

OR

```
import java.util.ArrayList;
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 ChicagoStyleCheesePizza extends Pizza
{public ChicagoStyleCheesePizza()
{
name = "Chicago Style Deep Dish Cheese Pizza";
dough = "Extra Thick Crust Dough";
sauce = "Plum Tomato Sauce";
toppings.add("Shredded Mozzarella Cheese");
}

void cut()
{System.out.println("Cutting the pizza into square slices");
}}
class ChicagoStyleClamPizza extends Pizza
{public ChicagoStyleClamPizza()
{
name = "Chicago Style Clam Pizza";
dough = "Extra Thick Crust Dough";
sauce = "Plum Tomato Sauce";
toppings.add("Shredded Mozzarella Cheese");
toppings.add("Frozen Clams from Chesapeake Bay");
```

```
}
void cut()
{System.out.println("Cutting the pizza into square slices");
}}
class ChicagoStylePepperoniPizza extends Pizza
{public ChicagoStylePepperoniPizza()
{
name = "Chicago Style Pepperoni Pizza";
dough = "Extra Thick Crust Dough";
sauce = "Plum Tomato Sauce";
toppings.add("Shredded Mozzarella Cheese");
toppings.add("Black Olives");
toppings.add("Spinach");
toppings.add("Eggplant");
toppings.add("Sliced Pepperoni");
}
void cut()
{System.out.println("Cutting the pizza into square slices");
}}
class ChicagoStyleVeggiePizza extends Pizza
{public ChicagoStyleVeggiePizza()
{name = "Chicago Deep Dish Veggie Pizza";
dough = "Extra Thick Crust Dough";
sauce = "Plum Tomato Sauce";
toppings.add("Shredded Mozzarella Cheese");
toppings.add("Black Olives");
toppings.add("Spinach");
toppings.add("Eggplant");
}
void cut()
{System.out.println("Cutting the pizza into square slices");
}}
class DependentPizzaStore
```

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

```
pizza.bake();
pizza.cut();
pizza.box();
return pizza;
}}
class NYPizzaStore extends PizzaStore
{Pizza createPizza(String item)
{if (item.equals("cheese"))
{return new NYStyleCheesePizza();
}
else if (item.equals("veggie"))
{return new NYStyleVeggiePizza();
}
else if (item.equals("clam"))
{return new NYStyleClamPizza();
}
else if (item.equals("pepperoni"))
{return new NYStylePepperoniPizza();
}
else return null;
}}
class NYStyleCheesePizza extends Pizza
{public NYStyleCheesePizza()
{
name = "NY Style Sauce and Cheese Pizza";
dough = "Thin Crust Dough";
sauce = "Marinara Sauce";
toppings.add("Grated Reggiano Cheese");
}}
class NYStyleClamPizza extends Pizza
{public NYStyleClamPizza()
{
name = "NY Style Clam Pizza";
```

```
dough = "Thin Crust Dough";
sauce = "Marinara Sauce";
toppings.add("Grated Reggiano Cheese");
toppings.add("Fresh Clams from Long Island Sound");
}}
class NYStylePepperoniPizza extends Pizza
{public NYStylePepperoniPizza()
{
name = "NY Style Pepperoni Pizza";
dough = "Thin Crust Dough";
sauce = "Marinara Sauce";
toppings.add("Grated Reggiano Cheese");
toppings.add("Sliced Pepperoni");
toppings.add("Garlic");
toppings.add("Onion");
toppings.add("Mushrooms");
toppings.add("Red Pepper");
}}
class NYStyleVeggiePizza extends Pizza
{public NYStyleVeggiePizza()
{
name = "NY Style Veggie Pizza";
dough = "Thin Crust Dough";
sauce = "Marinara Sauce";
toppings.add("Grated Reggiano Cheese");
toppings.add("Garlic");
toppings.add("Onion");
toppings.add("Mushrooms");
toppings.add("Red Pepper");
}}
abstract class Pizza
{
String name;
```

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

```
}}
abstract class PizzaStore
{abstract Pizza createPizza(String item);
public Pizza orderPizza(String type)
{Pizza pizza = createPizza(type);
System.out.println("--- Making a " + pizza.getName() + " ---");
pizza.prepare();
pizza.bake();
pizza.cut();
pizza.box();
return pizza;
}}
public class Main
{public static void main(String[] args)
{
PizzaStore nyStore = new NYPizzaStore();
PizzaStore chicagoStore = new ChicagoPizzaStore();
Pizza pizza = nyStore.orderPizza("cheese");
System.out.println("Ethan ordered a " + pizza.getName() + "\n");
pizza = chicagoStore.orderPizza("cheese");
System.out.println("Joel ordered a " + pizza.getName() + "\n");
pizza = nyStore.orderPizza("clam");
System.out.println("Ethan ordered a " + pizza.getName() + "\n");
pizza = chicagoStore.orderPizza("clam");
System.out.println("Joel ordered a " + pizza.getName() + "\n");
pizza = nyStore.orderPizza("pepperoni");
System.out.println("Ethan ordered a " + pizza.getName() + "\n");
pizza = chicagoStore.orderPizza("pepperoni");
System.out.println("Joel ordered a " + pizza.getName() + "\n");
pizza = nyStore.orderPizza("veggie");
System.out.println("Ethan ordered a " + pizza.getName() + "\n");
pizza = chicagoStore.orderPizza("veggie");
System.out.println("Joel ordered a " + pizza.getName() + "\n");
```

}}

Q2 Write a python program to Implement Simple Linear Regression for predicting house price.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_predict
data = pd.read_csv(r'kc_house_data.csv')
data.head(5)
print(data.shape)
# Make a list of importatnt feature which is needed to be incuding in training data
f = ['price', 'bedrooms', 'bathrooms', 'sqft_living', 'floors', 'condition', 'sqft_above', 'sqft_basement', 'yr_built',
    'yr_renovated']
data = data[f]
print(data.shape)
# Drop the missing values
data = data.dropna()
print(data.shape)
# Get the statictial information of the dataset
data.describe()
# Now,Divide the dataset into two parts:independent variable and dependent variable
X = data[f[1:]]
y = data['price']
# Split the dataset into training data and testing data
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=42)
print(X_train.shape)
```

```
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
# Fit the regression model
lr = LinearRegression() # Create object of linear regression class
lr.fit(X_train,y_train) #fit training data
print(lr.coef_)
# Create the Prediction
y_test_predict = lr.predict(X_test)
print(y_test_predict.shape)
# Plot the error
g=plt.plot((y_test - y_test_predict),marker='o',linestyle='')
# # Fit the regression model without b(w0)
lr = LinearRegression(fit_intercept=False)
lr.fit(X_train,y_train)
y_test_predict = lr.predict(X_test)
g=plt.plot((y_test - y_test_predict),marker='o',linestyle='')
```

3) Create a Node.js file that will convert the output "Hello World!" into upper-case letters.

```
var http = require('http');  // includes the http module
var uc = require('upper-case'); // include the upper-case module
http.createServer(function (req, res) {
   res.writeHead(200, {'Content-Type': 'text/html'});
   res.write(("hello world!").toUpperCase()); // assign the upper-case module
   res.end();
}).listen(8080);
```

Slip-5

Q1 Write a Java Program to implement Adapter pattern for Enumeration iterator

```
import java.util.Enumeration;
```

```
import java.util.Iterator;

// Create an adapter class that implements the Iterator interface and adapts an Enumeration
class EnumerationAdapter<T> implements Iterator<T> {
    private Enumeration<T> enumeration;

    public EnumerationAdapter(Enumeration<T> enumeration) {
        this.enumeration = enumeration;
    }

    @Override
    public boolean hasNext() {
        return enumeration.hasMoreElements();
    }

    @Override
    public T next() {
        return enumeration.nextElement();
    }

    @Override
    public void remove() {
        throw new UnsupportedOperationException("Remove operation is not supported");
    }
}

public class AdapterPatternDemo {
    public static void main(String[] args) {
        // Create an Enumeration of some elements
        Enumeration<String> enumeration = new Enumeration<String>() {
            private String[] elements = {"Mango", "Orange", "Apple"};
            private int index = 0;

            @Override
            public boolean hasMoreElements() {
                return index < elements.length;
            }

            @Override
            public String nextElement() {
                if (hasMoreElements()) {
                    return elements[index++];
                } else {
                    return null;
                }
            }
        };

        // Create an Iterator by adapting the Enumeration using the EnumerationAdapter
        Iterator<String> iterator = new EnumerationAdapter<>(enumeration);
```

```
        // Use the Iterator to traverse the elements
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }
}
```

OR

```
import java.util.*;
class EnumerationIterator implements Iterator {
        Enumeration enumeration;

        public EnumerationIterator(Enumeration enumeration) {
                this.enumeration = enumeration;
        }

        public boolean hasNext() {
                return enumeration.hasMoreElements();
        }

        public Object next() {
                return enumeration.nextElement();
        }

        public void remove() {
                throw new UnsupportedOperationException();
        }
}

class EnumerationIteratorTestDrive1 {
        public static void main (String args[]) {
                Vector v = new Vector(Arrays.asList("apple","mango","grapes"));
                Iterator iterator = new EnumerationIterator(v.elements());
                while (iterator.hasNext()) {
                        System.out.println(iterator.next());
                }
        }
}
```

2) Write a python program to implement Multiple Linear Regression for given dataset.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('50_Startups.csv')
X = dataset.iloc[:, :-1].values
```

```
y = dataset.iloc[:, -1].values
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [3])],
remainder='passthrough')
X = np.array(ct.fit_transform(X))
print(X)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
df = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})
df
```

3) Using nodejs create a web page to read two file names from user and append contents of first file into second file.

```
const fs = require('fs');

console.log("\nFile Contents of file before append:",
a=fs.readFileSync("file1.txt", "utf8"));

fs.appendFile("file2.txt", a, (err) => {
if (err) {
   console.log(err);
}
else {

   console.log("\nFile Contents of file after append:",
   fs.readFileSync("file2.txt", "utf8"));
}
});
```

Slip-6

Q1 Write a Java Program to implement command pattern to test Remote Control.

```
import java.util.*;

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

```
// Receiver class - Light
class Light {
    public void turnOn() {
        System.out.println("Light is ON");
    }

    public void turnOff() {
        System.out.println("Light is OFF");
    }
}

// Concrete Command classes
class LightOnCommand implements Command {
    private Light light;

    public LightOnCommand(Light light) {
        this.light = light;
    }

    public void execute() {
        light.turnOn();
    }
}

class LightOffCommand implements Command {
    private Light light;

    public LightOffCommand(Light light) {
        this.light = light;
    }

    public void execute() {
        light.turnOff();
    }
}
```

```
// Remote Control with multiple slots
class RemoteControl {
    private List<Command> commands = new ArrayList<>();

    public void setCommand(Command command) {
        commands.add(command);
    }

    public void pressButton(int slot) {
        if (slot >= 0 && slot < commands.size()) {
            commands.get(slot).execute();
        } else {
            System.out.println("Invalid slot number");
        }
    }
}

public class CommandPatternDemo {
    public static void main(String[] args) {
        // Create the receiver
        Light livingRoomLight = new Light();

        // Create the concrete commands
        Command livingRoomLightOn = new
LightOnCommand(livingRoomLight);
        Command livingRoomLightOff = new
LightOffCommand(livingRoomLight);

        // Create the remote control
        RemoteControl remote = new RemoteControl();

        // Set the commands in different slots
        remote.setCommand(livingRoomLightOn); // Slot 0
        remote.setCommand(livingRoomLightOff); // Slot 1

        // Press the buttons to execute the commands
        remote.pressButton(0); // Turn on the light
```

```
        remote.pressButton(1); // Turn off the light
        remote.pressButton(2); // Invalid slot

        // Create additional receivers and commands, and set them in the remote
control
        // to extend the functionality of the remote control.
    }
}
```

OR

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

public class CeilingFanOffCommand implements Command {
        CeilingFan ceilingFan;
        int prevSpeed;

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

        public void execute() {
                prevSpeed = ceilingFan.getSpeed();
                ceilingFan.off();
        }

        public void undo() {
                if (prevSpeed == CeilingFan.HIGH) {
                        ceilingFan.high();
                } else if (prevSpeed == CeilingFan.MEDIUM) {
                        ceilingFan.medium();
                } else if (prevSpeed == CeilingFan.LOW) {
                        ceilingFan.low();
                } else if (prevSpeed == CeilingFan.OFF) {
                        ceilingFan.off();
                }
        }
}

        public class CeilingFanMediumCommand implements Command {
        CeilingFan ceilingFan;
        int prevSpeed;
```

```
	public CeilingFanMediumCommand(CeilingFan ceilingFan) {
		this.ceilingFan = ceilingFan;
	}

	public void execute() {
		prevSpeed = ceilingFan.getSpeed();
		ceilingFan.medium();
	}

	public void undo() {
		if (prevSpeed == CeilingFan.HIGH) {
			ceilingFan.high();
		} else if (prevSpeed == CeilingFan.MEDIUM) {
			ceilingFan.medium();
		} else if (prevSpeed == CeilingFan.LOW) {
			ceilingFan.low();
		} else if (prevSpeed == CeilingFan.OFF) {
			ceilingFan.off();
		}
	}
}

public class CeilingFanHighCommand implements Command {
	CeilingFan ceilingFan;
	int prevSpeed;

	public CeilingFanHighCommand(CeilingFan ceilingFan) {
		this.ceilingFan = ceilingFan;
	}

	public void execute() {
		prevSpeed = ceilingFan.getSpeed();
		ceilingFan.high();
	}

	public void undo() {
		if (prevSpeed == CeilingFan.HIGH) {
			ceilingFan.high();
		} else if (prevSpeed == CeilingFan.MEDIUM) {
			ceilingFan.medium();
		} else if (prevSpeed == CeilingFan.LOW) {
			ceilingFan.low();
		} else if (prevSpeed == CeilingFan.OFF) {
			ceilingFan.off();
		}
	}
}

public class CeilingFanLowCommand implements Command {
	CeilingFan ceilingFan;
```

```
	int prevSpeed;

	public CeilingFanLowCommand(CeilingFan ceilingFan) {
		this.ceilingFan = ceilingFan;
	}

	public void execute() {
		prevSpeed = ceilingFan.getSpeed();
		ceilingFan.low();
	}

	public void undo() {
		if (prevSpeed == CeilingFan.HIGH) {
			ceilingFan.high();
		} else if (prevSpeed == CeilingFan.MEDIUM) {
			ceilingFan.medium();
		} else if (prevSpeed == CeilingFan.LOW) {
			ceilingFan.low();
		} else if (prevSpeed == CeilingFan.OFF) {
			ceilingFan.off();
		}
	}
}
public class CeilingFan {
	public static final int HIGH = 3;
	public static final int MEDIUM = 2;
	public static final int LOW = 1;
	public static final int OFF = 0;
	String location;
	int speed;

	public CeilingFan(String location) {
		this.location = location;
		speed = OFF;
	}

	public void high() {
		speed = HIGH;
		System.out.println(location + " ceiling fan is on high");
	}

	public void medium() {
		speed = MEDIUM;
		System.out.println(location + " ceiling fan is on medium");
	}

	public void low() {
		speed = LOW;
		System.out.println(location + " ceiling fan is on low");
	}
```

```
	public void off() {
		speed = OFF;
		System.out.println(location + " ceiling fan is off");
	}

	public int getSpeed() {
		return speed;
	}
}
```

2) Write a python program to implement Polynomial Linear Regression for given dataset

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('Position_Salaries.csv')
X = dataset.iloc[:, 1:-1].values
y = dataset.iloc[:, -1].values
dataset.head(5)
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
poly_reg = PolynomialFeatures(degree = 4)
X_poly = poly_reg.fit_transform(X)
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y)
y_pred = lin_reg.predict(X_poly)
df = pd.DataFrame({'Real Values':y, 'Predicted Values':y_pred})
df
X_grid = np.arange(min(X), max(X), 0.1)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X, y, color = 'red')
plt.scatter(X, y_pred, color = 'green')
plt.plot(X_grid,
lin_reg.predict(poly_reg.fit_transform(X_grid)), color =
'black')
plt.title('Polynomial Regression')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```

3) Create a Node.js file that opens the requested file and returns the content to the client. If anything goes wrong, throw a 404 error.

```
var fs = require('fs');
//var http = require('http');
//http.createServer(function (req, res) {

// Use fs.readFile() method to read the file
```

```
fs.readFile('file1.txt', 'utf8', function(err, data){

   /*if (err){

      res.writeHead(404, {'Content-Type': 'text/html'});


      return res.end("404 Not Found");

    }*/
   // Display the file content
   if(err) return console.error(err);


   console.log(data);

});

console.log('readFile called');
```

Slip-7

Q1 Write a Java Program to implement undo command to test Ceiling fan.

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

public class CeilingFanOffCommand implements Command {
        CeilingFan ceilingFan;
        int prevSpeed;

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

        public void execute() {
                prevSpeed = ceilingFan.getSpeed();
                ceilingFan.off();
        }

        public void undo() {
                if (prevSpeed == CeilingFan.HIGH) {
                        ceilingFan.high();
                } else if (prevSpeed == CeilingFan.MEDIUM) {
                        ceilingFan.medium();
                } else if (prevSpeed == CeilingFan.LOW) {
                        ceilingFan.low();
                } else if (prevSpeed == CeilingFan.OFF) {
                        ceilingFan.off();
                }
```

```
        }
}

        public class CeilingFanMediumCommand implements Command {
        CeilingFan ceilingFan;
        int prevSpeed;

        public CeilingFanMediumCommand(CeilingFan ceilingFan) {
                this.ceilingFan = ceilingFan;
        }

        public void execute() {
                prevSpeed = ceilingFan.getSpeed();
                ceilingFan.medium();
        }

        public void undo() {
                if (prevSpeed == CeilingFan.HIGH) {
                        ceilingFan.high();
                } else if (prevSpeed == CeilingFan.MEDIUM) {
                        ceilingFan.medium();
                } else if (prevSpeed == CeilingFan.LOW) {
                        ceilingFan.low();
                } else if (prevSpeed == CeilingFan.OFF) {
                        ceilingFan.off();
                }
        }
}

public class CeilingFanHighCommand implements Command {
        CeilingFan ceilingFan;
        int prevSpeed;

        public CeilingFanHighCommand(CeilingFan ceilingFan) {
                this.ceilingFan = ceilingFan;
        }

        public void execute() {
                prevSpeed = ceilingFan.getSpeed();
                ceilingFan.high();
        }

        public void undo() {
                if (prevSpeed == CeilingFan.HIGH) {
                        ceilingFan.high();
                } else if (prevSpeed == CeilingFan.MEDIUM) {
                        ceilingFan.medium();
                } else if (prevSpeed == CeilingFan.LOW) {
                        ceilingFan.low();
                } else if (prevSpeed == CeilingFan.OFF) {
```

```
                ceilingFan.off();
            }
        }
}

public class CeilingFanLowCommand implements Command {
        CeilingFan ceilingFan;
        int prevSpeed;

        public CeilingFanLowCommand(CeilingFan ceilingFan) {
            this.ceilingFan = ceilingFan;
        }

        public void execute() {
            prevSpeed = ceilingFan.getSpeed();
            ceilingFan.low();
        }

        public void undo() {
            if (prevSpeed == CeilingFan.HIGH) {
                ceilingFan.high();
            } else if (prevSpeed == CeilingFan.MEDIUM) {
                ceilingFan.medium();
            } else if (prevSpeed == CeilingFan.LOW) {
                ceilingFan.low();
            } else if (prevSpeed == CeilingFan.OFF) {
                ceilingFan.off();
            }
        }
}
public class CeilingFan {
        public static final int HIGH = 3;
        public static final int MEDIUM = 2;
        public static final int LOW = 1;
        public static final int OFF = 0;
        String location;
        int speed;

        public CeilingFan(String location) {
            this.location = location;
            speed = OFF;
        }

        public void high() {
            speed = HIGH;
            System.out.println(location + " ceiling fan is on high");
        }

        public void medium() {
            speed = MEDIUM;
```

```
            System.out.println(location + " ceiling fan is on medium");
        }

        public void low() {
            speed = LOW;
            System.out.println(location + " ceiling fan is on low");
        }

        public void off() {
            speed = OFF;
            System.out.println(location + " ceiling fan is off");
        }

        public int getSpeed() {
            return speed;
        }
}
```

2) Write a python program to implement Naive Bayes.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
data=pd.read_csv('pima-indians-diabetes.csv')
data.shape
data.isnull().sum()
data.isnull().values.any()
data.dtypes
#visualisation
data.hist()
columns=list(data)[0:-1]
data[columns].hist()
#identifty the correlation
data.corr()
sns.heatmap(data.corr(),annot=True)
sns.pairplot(data)
#calculate diabetes ratio of true or false target varible
n_true=len(data.loc[data['class']==True])
n_false=len(data.loc[data['class']==False])
print("No.of true cases:{0} {1}%".format(n_true,(n_true/(n_true+n_false))*100))
print("No.of false cases:{0} {1}%".format(n_false,(n_false/(n_true+n_false))*100))
#split the data
from sklearn.model_selection import train_test_split
x=data.drop('class',axis=1)
y=data['class']
X_train,X_test,Y_train,Y_test=train_test_split(x,y,test_size=0.30,random_state=1)
from sklearn.impute import SimpleImputer
rep_0=SimpleImputer(missing_values=0,strategy='mean')
cols=X_train.columns
```

```
X_train=pd.DataFrame(rep_0.fit_transform(X_train))
X_test=pd.DataFrame(rep_0.fit_transform(X_test))
X_train.columns=cols
X_test.columns=cols
X_train.head()
from sklearn.naive_bayes import GaussianNB
diab_mode=GaussianNB()
diab_mode.fit(X_train,Y_train)
diab_train_predict=diab_mode.predict(X_train)
from sklearn import metrics
print("Model Accuracy:{0}".format(metrics.accuracy_score(Y_train,diab_train_predict)))
diab_train_predict=diab_mode.predict(X_test)
from sklearn import metrics
print("Model Accuracy:{0}".format(metrics.accuracy_score(Y_test,diab_train_predict)))
cm1=metrics.confusion_matrix(Y_test,diab_train_predict,labels=[1,0])
df_cm1=pd.DataFrame(cm1,index=[i for i in['1','0']],columns=[i for i in['predict 1','predict
o']] )
df_cm1
```

3) Create a Node.js file that writes an HTML form, with an upload field.

```
var http = require('http');
var formidable = require('formidable');

http.createServer(function (req, res) {
  if (req.url == '/fileupload') {
    var form = new formidable.IncomingForm();
    form.parse(req, function (err, fields, files) {
      res.write('File uploaded');
      res.end();
    });
  } else {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write('<form action="fileupload" method="post" enctype="multipart/form-data">');
    res.write('<input type="file" name="filetoupload"><br>');
    res.write('<input type="submit">');
    res.write('</form>');
    return res.end();
  }
}).listen(8080);
```

Slip-8

Q1 Write a Java Program to implement State Pattern for Gumball Machine. Create instance variable that holds current state from there, we just need to handle all actions, behaviors and state transition that can happen.

State.java

```
public interface State {
```

```
        public void insertQuarter();
        public void ejectQuarter();
        public void turnCrank();
        public void dispense();

        public void refill();
}
```

SoldStat.java

```
public 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";
        }
}
```

SoldOutState.java

```
public 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";
        }
}
NoQuarterState.java
public 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";
        }
}
```

HasQuarterState.java

```
public 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";
        }

}
```

GumballMachine.java

```
public class GumballMachine {

        State soldOutState;
        State noQuarterState;
        State hasQuarterState;
        State soldState;

        State state;
        int count = 0;
```

```
	public GumballMachine(int numberGumballs) {
		soldOutState = new SoldOutState(this);
		noQuarterState = new NoQuarterState(this);
		hasQuarterState = new HasQuarterState(this);
		soldState = new SoldState(this);

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

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

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

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

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

	int getCount() {
		return count;
	}

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

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

```
        return state;
    }

    public State getSoldOutState() {
        return soldOutState;
    }

    public State getNoQuarterState() {
        return noQuarterState;
    }

    public State getHasQuarterState() {
        return hasQuarterState;
    }

    public State getSoldState() {
        return soldState;
    }

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

GumballMachineTestDrive.java

```
public class GumballMachineTestDrive{

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

                System.out.println(gumballMachine);

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

                System.out.println(gumballMachine);

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

```
                gumballMachine.refill(3);
                gumballMachine.insertQuarter();
                gumballMachine.turnCrank();

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

2) Write a python program to implement Decision Tree whether or not to play Tennis.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from scipy.stats import zscore
import seaborn as sns
data=pd.read_csv('play_tennis.csv')
data.isnull().sum()
data.dtypes
data.head()
data.outlook.value_counts()
from sklearn.preprocessing import LabelEncoder
l=LabelEncoder()
for i in data.columns:
   if data[i].dtypes=='object' or data[i].dtypes=='bool':
      data[i]=pd.Categorical(data[i])

for i in data.columns:
      data[i]=l.fit_transform(data[i])
data.dtypes
data.head()
x=data.drop(['play'],axis=1)
y=data['play']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=1)
dtree=DecisionTreeClassifier(criterion='gini',random_state=1)
dtree.fit(x_train,y_train)
print(dtree.score(x_train,y_train)) #data is over fitted so we use max_depth =5 means
prunning technique
print(dtree.score(x_test,y_test))
dtree1=DecisionTreeClassifier(criterion='gini',max_depth=5,random_state=1)
dtree1.fit(x_train,y_train)
print(dtree1.score(x_train,y_train)) #data is over fitted so we use max_depth =5 means
prunning technique
print(dtree1.score(x_test,y_test))
y_predict=dtree.predict(x_test)
from sklearn import metrics
cm=metrics.confusion_matrix(y_test,y_predict,labels=[1,0])
df_cm=pd.DataFrame(cm,index=[i for i in['1','0']],columns=[i for i in['predicted 1','predicted
0']] )
```

```
df_cm
sns.heatmap(df_cm,annot=True)
from sklearn.metrics import classification_report
m=classification_report(y_test,y_predict)
print(m)
```

3) Create a Node.js file that demonstrates create database and table in MySQL.

```
var mysql = require('mysql');
var con = mysql.createConnection({
host: "localhost",
user: "root",
password: "Root@123",
database: "node"
});
con.connect(function(err) {
if (err) throw err;
console.log("Connected!");
con.query("CREATE DATABASE Noded", function (err, result) {
if (err) throw err;
console.log("Database created");
});
});

var sql = "CREATE TABLE customers (name VARCHAR(25), address VARCHAR(25))";
con.query(sql, function (err, result) {
  if (err) throw err;
  console.log("Table created");
});
```

Slip-9

Q1 Design simple HR Application using Spring Framework

2) Write a python program to implement Linear SVM.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn import svm
data=pd.read_csv('letterdata.csv')
data.head()
def getaccuracy(testset,prediction):
    correct=0
    for x in range(len(testset)):
        if(testset[x]==prediction[x]):
            correct=correct+1
    return (correct/float(len(testset)))*100.0
```

```
data.isnull().values.any()
X,Y=np.array(data)[:,1:16],np.array(data.letter)[:]
X_train=X[:16000,:]
X_test=X[16001:,:]
Y_train=Y[:16000]
Y_test=Y[16001:]
clf=svm.SVC(gamma=0.025,C=3)
clf.fit(X_train,Y_train)
Y_predict=clf.predict(X_test)
getaccuracy(Y_test,Y_predict)
y_g=(np.column_stack([Y_test,Y_predict]))
#column stack used for matching supose x=1,2,3,4,5 and y= 10 20 30 40 match x axis on 1
check on y axis on 10 ...
print(y_g)
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2] # we only take the first two features. We could
 # avoid this ugly slicing by using a two-dim dataset
y = iris.target
# we create an instance of SVM and fit out data. We do not scale our
# data since we want to plot the support vectors
C = 1.0 # SVM regularization parameter
svc = svm.SVC(kernel='linear', C=1,gamma=10).fit(X, y)
# create a mesh to plot in
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
h = (x_max / x_min)/100
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
 np.arange(y_min, y_max, h))
plt.subplot(1, 1, 1)
Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('SVC with linear kernel')
plt.show()
svc = svm.SVC(kernel='rbf', C=1,gamma=10).fit(X, y)
plt.subplot(1, 1, 1)
Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
svc = svm.SVC(kernel='poly', C=1,gamma=100).fit(X, y)
plt.subplot(1, 1, 1)
Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
```

```
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
```

3) Create a node.js file that Select all records from the "customers" table, and display the result object on console.

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: 'localhost',
  user: "root",
  password: "Root@123",
  database:'node'
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
});

con.query('SELECT * FROM customers', (err,rows) => {
    if(err) throw err;

    console.log('Data received from Db:');
    console.log(rows);
  });
```

Slip-10

Q1 Write a Java Program to implement Strategy Pattern for Duck Behavior. Create instance variable that holds current state of Duck from there, we just need to handle all Flying Behaviors and Quack Behavior

Duck.Java

```
public abstract class Duck {
  FlyBehaviour flyBehaviour;
  QuackBehaviour quackBehaviour;
  public Duck() {
  }

  public abstract void display();

  public void performFly() {
    flyBehaviour.fly();
  }

  public void performQuack() {
    quackBehaviour.quack();
  }
```

```
  public void swim() {
    System.out.println("All ducks float, even decoys!);
  }
  public void setFlyBehaviour(FlyBehaviour fb) {
    flyBehaviour = fb;
  }
  public void setQuackBehaviour(QuackBehaviour qb) {
    quackBehaviour qb;
  }
}
```

MallardDuck.Java

```
public class MallardDuck extends Duck {

  public MallardDuck() {
    quackBehaviour = new Quack();
    flyBehaviour = new FlyWithWings();
  }

  public void display() {
    System.out.println("I'm a real Mallard duck");
  }
}
```

ModelDuck.Java

```
public class ModolDuck extends Duck {
  public ModelDuck() {
    flyBehaviour = new FlyNoWay();
    quackBehaviour = new Quack();
  }

  public void display() {
    System.out.println("I'm a model duck");
  }
}
```

FlyBehaviour.Java

```
public interface FlyBehaviour {
  public void fly():
}
```

FlyWithWings.Java

```
public class FlyWithWings implements FlyBehaviour {
  public void fly() {
    System.out.println("I'm flying!!");
  }
}
```

FlyNoWay.Java

```
public class FlyNoWay implements FlyBehaviour {
```

```
  public void fly() {
    System.out.println("I can't fly");
  }
}
```

FlyRocketPowered.Java

```
public class FlyRocketPowered implements FlyBehaviour {
  public void fly() {
    System.out.println("I'm flying with a rocket!");
  }
}
```

QuackBehaviour.Java

```
public interface QuackBehaviour {
  public void quack() {
    System.out.println("Quack");
  }
}
```

Quack.Java

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

MuteQuack.Java

```
public class MuteQuack implements QuackBehaviour {
  public void quack() {
    System.out.println("<< Silence >>");
  }
}
```

Squeak.Java

```
public class Squeak implements QuackBehaviour {
  public void quack() {
    System.out.println("Squeak");
  }
}
```

MiniDuckSimulator.Java

```
public class MiniDuckSimulator {
  public static void main(String[] args) {
    Duck mallard = new MallardDuck();
    mallard.performQuack();
    mallard.performFly();
  }
}
```

MiniDuckSimulator.Java - Model Duck Edition

```
public class MiniDuckSimulator {
  public static void main(String[] args) {
    Duck mallard = new MallardDuck();
    mallard.performQuack();
    mallard.performFly();
    Duck model = new ModelDuck();
    model.performFly();
    model.setFlyBehaviour(new FlyRocketPowered());
    model.performFly();
  }
}
```

2) Write a Python program to prepare Scatter Plot for Iris Dataset.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
iris = pd.read_csv("Iris.csv")   # Reading the dataset “Iris.csv”.
print (iris.head(10))   # head() will display the top rows of the dataset, the default value of this function is 5,
#that is it will show top 5 rows when no argument is given to it.

plt.plot(iris.Id, iris["SepalLengthCm"],"r--")
plt.show    #plt.show () will display the current figure that you are working on
iris.plot(kind ="scatter", x ='SepalLengthCm', y ='PetalLengthCm')
plt.grid()    # grid () function to add grid lines to the plot
```

3) Create a node.js file that Insert Multiple Records in "student" table, and display the result object on console.

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "Root@123",
database: "node"
});
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "INSERT INTO student (rollno,namee, percentage) VALUES ?";
var values = [

[1,'abc', 77.6],

[2,'def', 89.6],

[3,'ghi', 91.6]

];
```

```
con.query(sql, [values], function (err, result)
 {

   if (err) throw err;
   console.log("Number of records inserted: " + result.affectedRows);

  });

con.query("SELECT * FROM student", function (err, result, fields) {

   if (err) throw err;
   console.log(result);

  });

});
```

Slip-11

Q1 Write a java program to implement Adapter pattern to design Heart Model to Beat Model

Q2 Write a python program to find all null values in a given dataset and remove them.

```
import pandas as pd
import numpy as np
dict={'first score':[100,90,np.nan,95],
     'second score':[30,45,56,np.nan],
     'third score':[np.nan,40,80,98]}
df=pd.DataFrame(dict)
df.head()
df.isnull()
df.notnull()
#df=pd.DataFrame(dict)
df.fillna(0)
df.fillna(method='pad')
df.fillna(method='bfill')
df.replace(to_replace=np.nan,value=-99)
df.dropna()
df.dropna(axis=1)
new_data=df.dropna(axis=0)
new_data
```

Q3 Create a node.js file that Select all records from the "customers" table, and delete the specified record.

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "Root@123",
  database: "node"
});
con.connect(function(err) {
  if (err) throw err;
  var sql = "DELETE FROM customers WHERE name = 'pooja'";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Number of records deleted: " + result.affectedRows);
  });
});
```

Slip-12

Q1 Write a Java Program to implement Decorator Pattern for interface Car to define the assemble() method and then decorate it to Sports car and Luxury Car .

Car.java

```
package com.journaldev.design.decorator;
public interface Car {
	public void assemble();
}
```

BasicCar.java

```
package com.journaldev.design.decorator;
public class BasicCar implements Car {
	@Override
	public void assemble() {
		System.out.print("Basic Car.");
	}
}
```

CarDecorator.java

```
package com.journaldev.design.decorator;
public class CarDecorator implements Car {
	protected Car car;
	public CarDecorator(Car c){
		this.car=c;
	}
```

```
	@Override
	public void assemble() {
		this.car.assemble();
	}
}
```

SportsCar.java

```
package com.journaldev.design.decorator;
public class SportsCar extends CarDecorator {
	public SportsCar(Car c) {
		super(c);
	}
	@Override
	public void assemble(){
		super.assemble();
		System.out.print(" Adding features of Sports Car.");
	}
}
```

LuxuryCar.java

```
package com.journaldev.design.decorator;
public class LuxuryCar extends CarDecorator {

	public LuxuryCar(Car c) {
		super(c);
	}
	@Override
	public void assemble(){
		super.assemble();
		System.out.print(" Adding features of Luxury Car.");
	}
}
package com.journaldev.design.test;
import com.journaldev.design.decorator.BasicCar;
import com.journaldev.design.decorator.Car;
import com.journaldev.design.decorator.LuxuryCar;
import com.journaldev.design.decorator.SportsCar;
```

DecoratorPatternTest.java

```
public class DecoratorPatternTest {

	public static void main(String[] args) {
		Car sportsCar = new SportsCar(new BasicCar());
		sportsCar.assemble();
		System.out.println("\n*****");

		Car sportsLuxuryCar = new SportsCar(new LuxuryCar(new
BasicCar()));
		sportsLuxuryCar.assemble();
	}

}
```

Q2 Write a python program to make Categorical values in numeric format for a given dataset

```
import pandas as pd
#load Iris data set
iris = pd.read_csv('Iris.csv')
iris.head()
iris['code']=pd.factorize(iris.Species)[0]
iris.tail()
iris.code.value_counts()
```

Q3 Create a Simple Web Server using node js.

```
var http = require('http'); // 1 - Import Node.js core module
var server = http.createServer(function (req, res) {   // 2 - creating server
   //handle incomming requests here..
});
server.listen(5000); //3 - listen for any incoming requests
console.log('Node.js web server at port 5000 is running..')
```

Slip-13

Q1 Write a Java Program to implement an Adapter design pattern in mobile charger. Define two classes – Volt (to measure volts) and Socket (producing constant volts of 120V). Build an adapter that can produce 3 volts, 12 volts and default 120 volts.

Implements Adapter pattern using Class Adapter

Voltage.java

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

SocketAdapter.java

```
public interface SocketAdapter
{
    public Voltage get120Voltage();
    public Voltage get12Voltage();
    public Voltage get3VVoltage();
}
```

Socket.java

```
public class Socket
{
   public Voltage getVoltage()
   {
      return new Voltage(120); //In India 240 is the default voltage
   }
}
```

SocketAdapterImpl.java

```
public class SocketAdapterImpl extends Socket implements SocketAdapter
{

   //Using Composition for adapter pattern
   private Socket sock = new Socket();
```

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

AdapterEx.java

```
public class AdapterEx

{

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

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

        Voltage voltage120 = socketAdapter.get120Voltage();
```

```
        System.out.println(voltage120.getVolts());
    }
}
```

2) Write a Python program to prepare Scatter Plot for Iris Dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
iris = pd.read_csv("Iris.csv")   # Reading the dataset “Iris.csv”.
print (iris.head(10))   # head() will display the top rows of the dataset, the
default value of this function is 5,
#that is it will show top 5 rows when no argument is given to it.
plt.plot(iris.Id, iris["SepalLengthCm"],"r--")
plt.show    #plt.show () will display the current figure that you are working on
iris.plot(kind ="scatter", x ='SepalLengthCm', y ='PetalLengthCm')
plt.grid()    # grid () function to add grid lines to the plot
```

Q.3 Using node js create a User Login System.

Slip-14

Q1 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, StereoOnWithCDComman.

Command.java

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

LightOnCommand.java

```
public class LightOnCommand implements Command {
        Light light;

        public LightOnCommand(Light light) {
                this.light = light;
        }

        public void execute() {
                light.on();
        }
```

```
}
```
LightOffCommand.java
```
public class LightOffCommand implements Command {
	Light light;

	public LightOffCommand(Light light) {
		this.light = light;
	}

	public void execute() {
		light.off();
	}
}
```
Light.java
```
public class Light {
	String location = "";

	public Light(String location) {
		this.location = location;
	}
	public void on() {
		System.out.println(location + " light is on");
	}
	public void off() {
		System.out.println(location + " light is off");
	}
}
```
StereoOnWithCDCommand.java
```
public class StereoOnWithCDCommand implements Command {
	Stereo stereo;

	public StereoOnWithCDCommand(Stereo stereo) {
		this.stereo = stereo;
	}
	public void execute() {
		stereo.on();
		stereo.setCD();
```

```
            stereo.setVolume(11);
        }
}
```

StereoOffCommand.java

```
public class StereoOffCommand implements Command {
        Stereo stereo;

        public StereoOffCommand(Stereo stereo) {
            this.stereo = stereo;
        }
        public void execute() {
            stereo.off();
        }
}
```

RemoteControlWithUndo.java

```
public class RemoteControlWithUndo {
        Command[] onCommands;
        Command[] offCommands;
        Command undoCommand;

        public RemoteControlWithUndo() {
            onCommands = new Command[7];
            offCommands = new Command[7];

            Command noCommand = new NoCommand();
            for(int i=0;i<7;i++) {
                onCommands[i] = noCommand;
                offCommands[i] = noCommand;
            }
            undoCommand = noCommand;
        }
        public void setCommand(int slot, Command onCommand, Command
offCommand) {
            onCommands[slot] = onCommand;
            offCommands[slot] = offCommand;
        }
```

```
	public void onButtonWasPushed(int slot) {
		onCommands[slot].execute();
		undoCommand = onCommands[slot];
	}
	public void offButtonWasPushed(int slot) {
		offCommands[slot].execute();
		undoCommand = offCommands[slot];
	}
	public void undoButtonWasPushed() {
		undoCommand.undo();
	}

	public String toString() {
		StringBuffer stringBuff = new StringBuffer();
		stringBuff.append("\n------ Remote Control -------\n");
		for (int i = 0; i < onCommands.length; i++) {
			stringBuff.append("[slot " + i + "] " +
onCommands[i].getClass().getName()
					+ "    " + offCommands[i].getClass().getName() +
"\n");
		}
		stringBuff.append("[undo] " +
undoCommand.getClass().getName() + "\n");
		return stringBuff.toString();
	}
}
```

NoCommand.java

```
public class NoCommand implements Command {
	public void execute() { }
	public void undo() { }
}
```

RemoteLoader.java

```
public class RemoteLoader {

	public static void main(String[] args) {
		RemoteControlWithUndo remoteControl = new
RemoteControlWithUndo();
```

```
        Light livingRoomLight = new Light("Living Room");
        Light kitchenLight = new Light("Kitchen");
        CeilingFan ceilingFan= new CeilingFan("Living Room");
        GarageDoor garageDoor = new GarageDoor("");
        Stereo stereo = new Stereo("Living Room");

        LightOnCommand livingRoomLightOn =
                new LightOnCommand(livingRoomLight);
        LightOffCommand livingRoomLightOff =
                new LightOffCommand(livingRoomLight);
        LightOnCommand kitchenLightOn =
                new LightOnCommand(kitchenLight);
        LightOffCommand kitchenLightOff =
                new LightOffCommand(kitchenLight);

        CeilingFanOnCommand ceilingFanOn =
                new CeilingFanOnCommand(ceilingFan);
        CeilingFanOffCommand ceilingFanOff =
                new CeilingFanOffCommand(ceilingFan);

        GarageDoorUpCommand garageDoorUp =
                new GarageDoorUpCommand(garageDoor);
        GarageDoorDownCommand garageDoorDown =
                new GarageDoorDownCommand(garageDoor);

        StereoOnWithCDCommand stereoOnWithCD =
                new StereoOnWithCDCommand(stereo);
        StereoOffCommand  stereoOff =
                new StereoOffCommand(stereo);

        remoteControl.setCommand(0, livingRoomLightOn,
livingRoomLightOff);
        remoteControl.setCommand(1, kitchenLightOn, kitchenLightOff);
        remoteControl.setCommand(2, ceilingFanOn, ceilingFanOff);
        remoteControl.setCommand(3, stereoOnWithCD, stereoOff);
```

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

        remoteControl.onButtonWasPushed(0);
        remoteControl.offButtonWasPushed(0);
        remoteControl.onButtonWasPushed(1);
        remoteControl.offButtonWasPushed(1);
        remoteControl.onButtonWasPushed(2);
        remoteControl.offButtonWasPushed(2);
        remoteControl.onButtonWasPushed(3);
        remoteControl.offButtonWasPushed(3);
    }
}
```

2) Write a python program to find all null values in a given dataset and remove them.

```
import pandas as pd
import numpy as np
dict={'first score':[100,90,np.nan,95],
    'second score':[30,45,56,np.nan],
    'third score':[np.nan,40,80,98]}
df=pd.DataFrame(dict)
df.head()
df.isnull()
df.notnull()
#df=pd.DataFrame(dict)
df.fillna(0)
df.fillna(method='pad')
df.fillna(method='bfill')
df.replace(to_replace=np.nan,value=-99)
df.dropna()
df.dropna(axis=1)
new_data=df.dropna(axis=0)
new_data
```

Q3 Write node js script to interact with the filesystem, and serve a web page from a file .

Slip-15

Q1 Write a Java Program to implement Facade Design Pattern for HomeTheater.

Amplifier.java

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

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

```
  System.out.println(description + " setting CD player to " + cd);
  this.cd = cd;
 }
public String toString() {
  return description;
 }
}
```

CdPlayer.java

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

 public CdPlayer(String description, Amplifier amplifier) {
  this.description = description;
  this.amplifier = amplifier;
 }
 public void on() {
  System.out.println(description + " on");
 }
 public void off() {
  System.out.println(description + " off");
 }
 public void eject() {
  title = null;
  System.out.println(description + " eject");
 }
 public void play(String title) {
  this.title = title;
  currentTrack = 0;
  System.out.println(description + " playing \"" + title + "\"");
 }
 public void play(int track) {
  if (title == null) {
```

```
   System.out.println(description + " can't play track " + currentTrack +
     ", no cd inserted");
  } else {
   currentTrack = track;
   System.out.println(description + " playing track " + currentTrack);
  }
 }

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

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

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

DvdPlayer.java

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

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

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

```
 }

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

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

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

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

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

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

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

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

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

Projector.java

```
public 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

```
public 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

```
public 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

```
public 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;
        }
}
```

HomeTheaterFacade.java

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

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

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

 public void watchMovie(String movie) {
```

```
 System.out.println("Get ready to watch a movie...");
 popper.on();
 popper.pop();
 lights.dim(10);
 screen.down();
 projector.on();
 projector.wideScreenMode();
 amp.on();
 amp.setDvd(dvd);
 amp.setSurroundSound();
 amp.setVolume(5);
 dvd.on();
 dvd.play(movie);
}


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

public void listenToCd(String cdTitle) {
 System.out.println("Get ready for an audiopile experence...");
 lights.on();
 amp.on();
 amp.setVolume(5);
 amp.setCd(cd);
 amp.setStereoSound();
 cd.on();
 cd.play(cdTitle);
```

```
 }
 public void endCd() {
  System.out.println("Shutting down CD...");
  amp.off();
  amp.setCd(cd);
  cd.eject();
  cd.off();
 }
 public void listenToRadio(double frequency) {
  System.out.println("Tuning in the airwaves...");
  tuner.on();
  tuner.setFrequency(frequency);
  amp.on();
  amp.setVolume(5);
  amp.setTuner(tuner);
 }
 public void endRadio() {
  System.out.println("Shutting down the tuner...");
  tuner.off();
  amp.off();
 }
}
```

HomeTheaterTestDrive.java

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

  HomeTheaterFacade homeTheater =
    new HomeTheaterFacade(amp, tuner, dvd, cd,
```

```
    projector, screen, lights, popper);

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

Tuner.java

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

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

2) Write a python program to make Categorical values in numeric format for a given dataset

```
import pandas as pd
#load Iris data set
iris = pd.read_csv('Iris.csv')
iris.head()
iris['code']=pd.factorize(iris.Species)[0]
iris.tail()
iris.code.value_counts()
```

3) Write node js script to build Your Own Node.js Module. Use require ('http') module is a built-in Node module that invokes the functionality of the HTTP library to create a local server. Also use the export statement to make functions in your module available externally. Create a new text file to contain the functions in your module called, "modules.js" and add this function to return today's date and time.

```
var http = require('http');
var dt = require('./myfirstmodule17');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write("The date and time are currently: " + dt.myDateTime());
  res.end();
}).listen(8080);
Myfirstmodule:
exports.myDateTime = function () {
    return Date();
  };
```

Slip-16

Q1 Write a Java Program to implement Observer Design Pattern for number conversion. Accept a number in Decimal form and represent it in Hexadecimal, Octal and Binary. Change the Number and it reflects in other forms also

Subject.java

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

```
public class Subject {

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

  public int getState() {
    return state;
  }

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

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

  public void notifyAllObservers(){
    for (Observer observer : observers) {
      observer.update();
    }
  }
}
```

Observer.java

```
public abstract class Observer {
  protected Subject subject;
  public abstract void update();
}
```

BinaryObserver.java

```
public class BinaryObserver extends Observer{

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

```
  @Override
  public void update() {
    System.out.println( "Binary String: " + Integer.toBinaryString(
subject.getState() ) );
  }
}
```

OctalObserver.java

```
public class OctalObserver extends Observer{

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

  @Override
  public void update() {
   System.out.println( "Octal String: " + Integer.toOctalString(
subject.getState() ) );
  }
}
```

HexaObserver.java

```
public class HexaObserver extends Observer{

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

  @Override
  public void update() {
    System.out.println( "Hex String: " + Integer.toHexString( subject.getState()
).toUpperCase() );
  }
}
```

ObserverPatternDemo.java

```
public class ObserverPatternDemo {
```

```
  public static void main(String[] args) {
    Subject subject = new Subject();

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

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

2) Write a python program to Implement Simple Linear Regression for predicting house price.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_predict
data = pd.read_csv(r'kc_house_data.csv')
data.head(5)
print(data.shape)
# Make a list of importatnt feature which is needed to be incuding in training data
f = ['price', 'bedrooms', 'bathrooms', 'sqft_living', 'floors', 'condition',
'sqft_above', 'sqft_basement', 'yr_built',
    'yr_renovated']
data = data[f]
print(data.shape)
# Drop the missing values
data = data.dropna()
print(data.shape)
# Get the statictial information of the dataset
data.describe()
```

```
# Now,Divide the dataset into two parts:independent variable and dependent variable
X = data[f[1:]]
y = data['price']
# Split the dataset into training data and testing data
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=42)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
# Fit the regression model
lr = LinearRegression() # Create object of linear regression class
lr.fit(X_train,y_train) #fit training data
print(lr.coef_)
# Create the Prediction
y_test_predict = lr.predict(X_test)
print(y_test_predict.shape)
# Plot the error
g=plt.plot((y_test - y_test_predict),marker='o',linestyle='')
# # Fit the regression model without b(w0)
lr = LinearRegression(fit_intercept=False)
lr.fit(X_train,y_train)
y_test_predict = lr.predict(X_test)
g=plt.plot((y_test - y_test_predict),marker='o',linestyle='')
```

3) Create a js file named main.js for event-driven application. There should be a main loop that listens for events, and then triggers a callback function when one of those events is detected.

```
// Import events module
var events = require('events');

// Create an eventEmitter object
var eventEmitter = new events.EventEmitter();
// Create an event handler as follows
var connectHandler = function connected() {
```

```
  console.log('connection succesful.');

  // Fire the data_received event
  eventEmitter.emit('data_received');
}
// Bind the connection event with the handler
eventEmitter.on('connection', connectHandler);
// Bind the data_received event with the anonymous function
eventEmitter.on('data_received', function() {
  console.log('data received succesfully.');
});
// Fire the connection event
eventEmitter.emit('connection');
console.log("Program Ended.");
```

Slip-17

Q1 Write a Java Program to implement Abstract Factory Pattern for Shape interface.

Shape.java

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

RoundedSquare.java

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

Rectangle.java

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

Step 3

Create an Abstract class to get factories for Normal and Rounded Shape Objects.

AbstractFactory.java

```
public abstract class AbstractFactory {
   abstract Shape getShape(String shapeType) ;
}
```

Step 4

Create Factory classes extending AbstractFactory to generate object of concrete class based on given information.

ShapeFactory.java

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

RoundedShapeFactory.java

```
public class RoundedShapeFactory extends AbstractFactory {
   @Override
```

```
  public Shape getShape(String shapeType){
    if(shapeType.equalsIgnoreCase("RECTANGLE")){
      return new RoundedRectangle();
    }else if(shapeType.equalsIgnoreCase("SQUARE")){
      return new RoundedSquare();
    }
    return null;
  }
}
```

Step 5

Create a Factory generator/producer class to get factories by passing an information such as Shape

FactoryProducer.java

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

Step 6

Use the FactoryProducer to get AbstractFactory in order to get factories of concrete classes by passing an information such as type.

AbstractFactoryPatternDemo.java

```
public class AbstractFactoryPatternDemo {
  public static void main(String[] args) {
    //get shape factory
    AbstractFactory shapeFactory = FactoryProducer.getFactory(false);
    //get an object of Shape Rectangle
    Shape shape1 = shapeFactory.getShape("RECTANGLE");
    //call draw method of Shape Rectangle
```

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

   }
}
```

2) Write a python program to implement Multiple Linear Regression for a given dataset.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('50_Startups.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [3])],
remainder='passthrough')
X = np.array(ct.fit_transform(X))
print(X)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
from sklearn.linear_model import LinearRegression
```

```
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
df = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})
df
```

3) Write node js application that transfer a file as an attachment on web and enables browser to prompt the user to download file using express js.

```
var express = require('express');
var app = express();
var PORT = 3000;
app.get('/', function(req, res){
   res.download('hello.txt');
});
app.listen(PORT, function(err){
   if (err) console.log(err);
   console.log("Server listening on PORT", PORT);
});
```

Slip-18

Q1 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().

```
import java.util.*;
interface Observer {
   public void update(float temp, float humidity, float pressure);
}
interface DisplayElement {
   public void display();
}

interface Subject {
   public void registerObserver(Observer o);

   public void removeObserver(Observer o);
```

```
    public void notifyObservers();
}

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

    public WeatherData() {
        observers = new ArrayList<>();
    }

    public void registerObserver(Observer o) {
        observers.add(o);
    }

    public void removeObserver(Observer o) {
        int i = observers.indexOf(o);
        if (i >= 0) {
            observers.remove(i);
        }
    }

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

    public void measurementsChanged() {
        notifyObservers();
    }

    public void setMeasurements(float temperature, float humidity, float
pressure) {
```

```
        this.temperature = temperature;
        this.humidity = humidity;
        this.pressure = pressure;
        measurementsChanged();
    }

    public float getTemperature() {
        return temperature;
    }

    public float getHumidity() {
        return humidity;
    }

    public float getPressure() {
        return pressure;
    }
}

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

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

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

        display();
    }

    public void display() {
```

```
        System.out.print("Forecast: ");
        if (currentPressure > lastPressure) {
            System.out.println("Improving weather on the way!");
        } else if (currentPressure == lastPressure) {
            System.out.println("More of the same");
        } else if (currentPressure < lastPressure) {
            System.out.println("Watch out for cooler weather, rainy weather");
        }
    }
}

class HeatIndexDisplay implements Observer, DisplayElement {
    float heatIndex = 0.0f;
    private WeatherData weatherData;

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

    public void update(float t, float rh, float pressure) {
        heatIndex = computeHeatIndex(t, rh);
        display();
    }

    private float computeHeatIndex(float t, float rh) {
        float index = (float) ((16.923 + (0.185212 * t) + (5.37941 * rh) - (0.100254
* t * rh)
                + (0.00941695 * (t * t)) + (0.00728898 * (rh * rh))
                + (0.000345372 * (t * t * rh)) - (0.000814971 * (t * rh * rh)) +
                (0.0000102102 * (t * t * rh * rh)) - (0.000038646 * (t * t * t)) +
(0.0000291583 *
                        (rh * rh * rh))
                + (0.00000142721 * (t * t * t * rh)) +
                (0.000000197483 * (t * rh * rh * rh)) - (0.0000000218429 * (t * t * t *
rh * rh)) +
                0.000000000843296 * (t * t * rh * rh * rh)) -
```

```
            (0.0000000000481975 * (t * t * t * rh * rh * rh)));
        return index;
    }

    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;
    private WeatherData weatherData;

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

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

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

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

        display();
    }

    public void display() {
```

```
        System.out.println("Avgerage/Maximum/Minimum temperature = " +
(tempSum / numReadings)
                + "/" + maxTemp + "/" + minTemp);
    }
}

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

    public CurrentConditionsDisplay(Subject weatherData) {
        this.weatherData = weatherData;
        weatherData.registerObserver(this);
    }

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

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

class weather {

    public static void main(String[] args) {
        WeatherData weatherData = new WeatherData();

        CurrentConditionsDisplay currentDisplay = new
CurrentConditionsDisplay(weatherData);
        StatisticsDisplay statisticsDisplay = new StatisticsDisplay(weatherData);
        ForecastDisplay forecastDisplay = new ForecastDisplay(weatherData);
```

```
        weatherData.setMeasurements(70, 55, 40.4f);
        weatherData.setMeasurements(72, 60, 39.2f);
        weatherData.setMeasurements(68, 80, 39.2f);
    }
}
```

2) Write a python program to implement Polynomial Linear Regression for given dataset.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('Position_Salaries.csv')
X = dataset.iloc[:, 1:-1].values
y = dataset.iloc[:, -1].values
dataset.head(5)
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
poly_reg = PolynomialFeatures(degree = 4)
X_poly = poly_reg.fit_transform(X)
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y)
y_pred = lin_reg.predict(X_poly)
df = pd.DataFrame({'Real Values':y, 'Predicted Values':y_pred})
df
X_grid = np.arange(min(X), max(X), 0.1)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X, y, color = 'red')
plt.scatter(X, y_pred, color = 'green')
plt.plot(X_grid,
lin_reg.predict(poly_reg.fit_transform(X_grid)), color =
'black')
plt.title('Polynomial Regression')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```

Q3 Create your Django app in which after running the server, you should see on the browser, the text "Hello! I am learning Django", which you defined in the index view.

```
<html>
<head>
   <script type="text/javascript">
   function validate()
   {
      var regName=/^[a-zA-z]+[a-zA-Z]+$/;
      var fname=document.getElementById("txtfname").value;
      var lname=document.getElementById("txtlname").value;
      var age=document.getElementById("txtage").value;
      var mobno=document.getElementById("txtmobno").value;

      if(age<18||age>50)
         alert("student age must be 18 to 50");
      if(!regName.test(fname))
         alert("invalid name is given");
      else
         alert("valid name is given");
   }
   </script>
   </head>
   <body>
   <form>
      enter student first name
      <input type="text" name="txtfname" id="txtfname"><br>
      enter student last name
      <input type="text" name="txtlname" id="txtlname"><br>
      enter student age
      <input type="text" name="txtage" id="txtage"><br>
      enter mobile no
      <input type="text" name="txtmobno" id="txtmobno"><br>
      <input type="button" value="validate" onclick="validate()">
   </form>
   </body>
```

</html>

Slip-19

Q1 Write a Java Program to implement Factory method for Pizza Store with createPizza(), orederPizza(), prepare(), Bake(), cut(), box(). Use this to create variety of pizza's like NyStyleCheesePizza, ChicagoStyleCheesePizza etc.

```
import java.util.ArrayList;
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 ChicagoStyleCheesePizza extends Pizza
{public ChicagoStyleCheesePizza()
{
name = "Chicago Style Deep Dish Cheese Pizza";
dough = "Extra Thick Crust Dough";
sauce = "Plum Tomato Sauce";
toppings.add("Shredded Mozzarella Cheese");
}

void cut()
{System.out.println("Cutting the pizza into square slices");
}}
class ChicagoStyleClamPizza extends Pizza
{public ChicagoStyleClamPizza()
{
name = "Chicago Style Clam Pizza";
dough = "Extra Thick Crust Dough";
sauce = "Plum Tomato Sauce";
toppings.add("Shredded Mozzarella Cheese");
```

```
toppings.add("Frozen Clams from Chesapeake Bay");
}
void cut()
{System.out.println("Cutting the pizza into square slices");
}}
class ChicagoStylePepperoniPizza extends Pizza
{public ChicagoStylePepperoniPizza()
{
name = "Chicago Style Pepperoni Pizza";
dough = "Extra Thick Crust Dough";
sauce = "Plum Tomato Sauce";
toppings.add("Shredded Mozzarella Cheese");
toppings.add("Black Olives");
toppings.add("Spinach");
toppings.add("Eggplant");
toppings.add("Sliced Pepperoni");
}
void cut()
{System.out.println("Cutting the pizza into square slices");
}}
class ChicagoStyleVeggiePizza extends Pizza
{public ChicagoStyleVeggiePizza()
{name = "Chicago Deep Dish Veggie Pizza";
dough = "Extra Thick Crust Dough";
sauce = "Plum Tomato Sauce";
toppings.add("Shredded Mozzarella Cheese");
toppings.add("Black Olives");
toppings.add("Spinach");
toppings.add("Eggplant");
}
void cut()
{System.out.println("Cutting the pizza into square slices");
}}
class DependentPizzaStore
{public Pizza createPizza(String style, String type)
{ Pizza pizza = null;
if (style.equals("NY"))
{if (type.equals("cheese"))
{pizza = new NYStyleCheesePizza();
}
else if (type.equals("veggie"))
{pizza = new NYStyleVeggiePizza();
}
```

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

```
else return null;
}}
class NYStyleCheesePizza extends Pizza
{public NYStyleCheesePizza()
{
name = "NY Style Sauce and Cheese Pizza";
dough = "Thin Crust Dough";
sauce = "Marinara Sauce";
toppings.add("Grated Reggiano Cheese");
}}
class NYStyleClamPizza extends Pizza
{public NYStyleClamPizza()
{
name = "NY Style Clam Pizza";
dough = "Thin Crust Dough";
sauce = "Marinara Sauce";
toppings.add("Grated Reggiano Cheese");
toppings.add("Fresh Clams from Long Island Sound");
}}
class NYStylePepperoniPizza extends Pizza
{public NYStylePepperoniPizza()
{
name = "NY Style Pepperoni Pizza";
dough = "Thin Crust Dough";
sauce = "Marinara Sauce";
toppings.add("Grated Reggiano Cheese");
toppings.add("Sliced Pepperoni");
toppings.add("Garlic");
toppings.add("Onion");
toppings.add("Mushrooms");
toppings.add("Red Pepper");
}}
class NYStyleVeggiePizza extends Pizza
{public NYStyleVeggiePizza()
{
name = "NY Style Veggie Pizza";
dough = "Thin Crust Dough";
sauce = "Marinara Sauce";
toppings.add("Grated Reggiano Cheese");
toppings.add("Garlic");
toppings.add("Onion");
toppings.add("Mushrooms");
toppings.add("Red Pepper");
```

```
}}
abstract class Pizza
{
String name;
String dough;
String sauce;
ArrayList toppings = new ArrayList();
void prepare()
{
System.out.println("Preparing " + name);
System.out.println("Tossing dough...");
System.out.println("Adding sauce...");
System.out.println("Adding toppings: ");
for (int i = 0; i < toppings.size(); i++)
{System.out.println(" " + toppings.get(i));
}}
void bake()
{System.out.println("Bake for 25 minutes at 350");
}
void cut()
{System.out.println("Cutting the pizza into diagonal slices");
}
void box()
{System.out.println("Place pizza in official PizzaStore box");
}
public String getName()
{return name;
}
public String toString()
{StringBuffer display = new StringBuffer();
display.append("---- " + name + " ----\n");
display.append(dough + "\n");
display.append(sauce + "\n");
for (int i = 0; i < toppings.size(); i++)
{display.append((String )toppings.get(i) + "\n");
}
return display.toString();
}}
abstract class PizzaStore
{abstract Pizza createPizza(String item);
public Pizza orderPizza(String type)
{Pizza pizza = createPizza(type);
System.out.println("--- Making a " + pizza.getName() + " ---");
```

```
pizza.prepare();
pizza.bake();
pizza.cut();
pizza.box();
return pizza;
}}
public class Main
{public static void main(String[] args)
{
PizzaStore nyStore = new NYPizzaStore();
PizzaStore chicagoStore = new ChicagoPizzaStore();
Pizza pizza = nyStore.orderPizza("cheese");
System.out.println("Ethan ordered a " + pizza.getName() + "\n");
pizza = chicagoStore.orderPizza("cheese");
System.out.println("Joel ordered a " + pizza.getName() + "\n");
pizza = nyStore.orderPizza("clam");
System.out.println("Ethan ordered a " + pizza.getName() + "\n");
pizza = chicagoStore.orderPizza("clam");
System.out.println("Joel ordered a " + pizza.getName() + "\n");
pizza = nyStore.orderPizza("pepperoni");
System.out.println("Ethan ordered a " + pizza.getName() + "\n");
pizza = chicagoStore.orderPizza("pepperoni");
System.out.println("Joel ordered a " + pizza.getName() + "\n");
pizza = nyStore.orderPizza("veggie");
System.out.println("Ethan ordered a " + pizza.getName() + "\n");
pizza = chicagoStore.orderPizza("veggie");
System.out.println("Joel ordered a " + pizza.getName() + "\n");
}}
```

2) Write a python program to implement Naive Bayes.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
data=pd.read_csv('pima-indians-diabetes.csv')
data.shape
data.isnull().sum()
data.isnull().values.any()
data.dtypes
#visualisation
data.hist()
columns=list(data)[0:-1]
data[columns].hist()
```

```
#identifty the correlation
data.corr()
sns.heatmap(data.corr(),annot=True)
sns.pairplot(data)
#calculate diabetes ratio of true or false target varible
n_true=len(data.loc[data['class']==True])
n_false=len(data.loc[data['class']==False])
print("No.of true cases:{0} {1}%".format(n_true,(n_true/(n_true+n_false))*100))
print("No.of false cases:{0} {1}%".format(n_false,(n_false/(n_true+n_false))*100))
#split the data
from sklearn.model_selection import train_test_split
x=data.drop('class',axis=1)
y=data['class']
X_train,X_test,Y_train,Y_test=train_test_split(x,y,test_size=0.30,random_state=1)
from sklearn.impute import SimpleImputer
rep_0=SimpleImputer(missing_values=0,strategy='mean')
cols=X_train.columns
X_train=pd.DataFrame(rep_0.fit_transform(X_train))
X_test=pd.DataFrame(rep_0.fit_transform(X_test))
X_train.columns=cols
X_test.columns=cols
X_train.head()
from sklearn.naive_bayes import GaussianNB
diab_mode=GaussianNB()
diab_mode.fit(X_train,Y_train)
diab_train_predict=diab_mode.predict(X_train)
from sklearn import metrics
print("Model Accuracy:{0}".format(metrics.accuracy_score(Y_train,diab_train_predict)))
diab_train_predict=diab_mode.predict(X_test)
from sklearn import metrics
print("Model Accuracy:{0}".format(metrics.accuracy_score(Y_test,diab_train_predict)))
cm1=metrics.confusion_matrix(Y_test,diab_train_predict,labels=[1,0])
df_cm1=pd.DataFrame(cm1,index=[i for i in['1','0']],columns=[i for i in['predict 1','predict
o']] )
df_cm1
```

3) Design a Django application that adds web pages with views and templates.

```
<html>
   <head>
      <script type="text/javascript">
      function validate()
      {
```

```
    var regName=/^[a-zA-z]+[a-zA-Z]+$/;
    var dateformatdob = /^(0?[1-9]|[12][0-9]|3[01])[\/\-](0?[1-9]|1[012])[\/\-]\d{4}$/;
    var dateformatjdate = /^(0?[1-9]|[12][0-9]|3[01])[\/\-](0?[1-9]|1[012])[\/\-]\d{4}$/;


   //Max six digits, a dot, max two digits after dot

  var salaryformat=/^\d{1,6}(?:\.\d{0,2})?$/

    var name=document.getElementById("txtname").value;
    var dob=document.getElementById("txtdob").value;
    var jdate=document.getElementById("txtjdate").value;
    var salary=document.getElementById("txtsalary").value;

    if(!regName.test(name))
       alert("invalid name is given");
    else
       alert("valid name is given");

    if(!dateformatjdate.test(jdate))
       alert("invalid joining date is given");
    else
       alert("valid joining date is given");
    if(!dateformatdob.test(dob))
       alert("invalid date of birth is given");
    else
       alert("valid date of birth is given is given");
    if(!salaryformat.test(salary))
       alert("invalid salary");
    else
       alert("salary is valid");
    }
    </script>
  </head>
<body>
<form>
  <h1>Employee Rsgistration Details</h1>
    enter employee first name
    <input type="text" name="txtfname" id="txtname"><br>
    enter date of birth
    <input type="text" name="txtdob" id="txtdob"><br>
    enter joining date
    <input type="text" name="txtjdate" id="txtjdate"><br>
```

```
    enter salary
    <input type="text" name="txtsalary" id="txtsalary"><br>
    <input type="button" value="validate" onclick="validate()">
  </form>
  </body>
  </html>
```

Slip-20

Q1 Write a Java Program to implement I/O Decorator for converting uppercase letters to lower case letters.

Lower.java

```
import java.io.*;
import java.util.*;
class LowerCaseInputStream extends FilterInputStream
{
public 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;
}
}
class Lower
{
public static void main(String[] args) throws IOException
{
```

```
int c;
try
{
        InputStream in =new LowerCaseInputStream(new
BufferedInputStream(new FileInputStream("data.txt")));
        while((c = in.read()) >= 0)
        {
                System.out.print((char)c);
        }
        in.close();
}
catch(IOException e)
{
        e.printStackTrace();
}
}
}
```

2) Write a python program to implement Decision Tree whether or not to play Tennis.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from scipy.stats import zscore
import seaborn as sns
data=pd.read_csv('play_tennis.csv')
data.isnull().sum()
data.dtypes
data.head()
data.outlook.value_counts()
from sklearn.preprocessing import LabelEncoder
l=LabelEncoder()
for i in data.columns:
   if data[i].dtypes=='object' or data[i].dtypes=='bool':
```

```
    data[i]=pd.Categorical(data[i])

for i in data.columns:
    data[i]=l.fit_transform(data[i])
data.dtypes
data.head()
x=data.drop(['play'],axis=1)
y=data['play']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=1
)
dtree=DecisionTreeClassifier(criterion='gini',random_state=1)
dtree.fit(x_train,y_train)
print(dtree.score(x_train,y_train)) #data is over fitted so we use max_depth =5
means prunning technique
print(dtree.score(x_test,y_test))
dtree1=DecisionTreeClassifier(criterion='gini',max_depth=5,random_state=1)
dtree1.fit(x_train,y_train)
print(dtree1.score(x_train,y_train)) #data is over fitted so we use max_depth =5
means prunning technique
print(dtree1.score(x_test,y_test))
y_predict=dtree.predict(x_test)
from sklearn import metrics
cm=metrics.confusion_matrix(y_test,y_predict,labels=[1,0])
df_cm=pd.DataFrame(cm,index=[i for i in['1','0']],columns=[i for i in['predicted
1','predicted 0']] )
df_cm
sns.heatmap(df_cm,annot=True)
from sklearn.metrics import classification_report
m=classification_report(y_test,y_predict)
print(m)
```

3) Develop a basic poll application (app).It should consist of two parts:

a) A public site in which user can pick their favourite programming language and vote.

b) An admin site that lets you add, change and delete programming languages.

```
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
   </style>
</head>
<body>
<script>
function validateform(){
var email = document.getElementById("email").value;
var password = document.getElementById("psw").value;
if (/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/.test(email))
  {
    alert("Valid Email Id..")
    return (true)
  }
   else{ alert("You have entered an invalid email address!")
    return (false)
   }
}
</script>
<form name="myform" onsubmit="return validateform()">
  <div class="container">
    <p>Please fill in this form to Login.</p>
    <hr>
    <label for="email"><b>Email</b></label>
    <input type="text" autocomplete="off" placeholder="Enter Email"
name="email" id="email" required>
    <label for="psw"><b>Password</b></label>
    <input type="password" autocomplete="off" placeholder="Enter Password"
name="psw" id="psw" required>
    <hr>
    <button type="submit" class="registerbtn">Register</button>
  </div>
</form>
</body>
</html>
```

Slip-21

Q1 Write a Java Program to implement command pattern to test Remote Control.

```
public interface Command {
public void execute();
public void undo();
}
public class CeilingFanOffCommand implements Command {
CeilingFan ceilingFan;
int prevSpeed;

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

public void execute() {
        prevSpeed = ceilingFan.getSpeed();
        ceilingFan.off();
}

public void undo() {
        if (prevSpeed == CeilingFan.HIGH) {
                ceilingFan.high();
        } else if (prevSpeed == CeilingFan.MEDIUM) {
                ceilingFan.medium();
        } else if (prevSpeed == CeilingFan.LOW) {
                ceilingFan.low();
        } else if (prevSpeed == CeilingFan.OFF) {
                ceilingFan.off();
        }
}
}

public class CeilingFanMediumCommand implements Command {
```

```
CeilingFan ceilingFan;
int prevSpeed;

public CeilingFanMediumCommand(CeilingFan ceilingFan) {
        this.ceilingFan = ceilingFan;
}

public void execute() {
        prevSpeed = ceilingFan.getSpeed();
        ceilingFan.medium();
}

public void undo() {
        if (prevSpeed == CeilingFan.HIGH) {
                ceilingFan.high();
        } else if (prevSpeed == CeilingFan.MEDIUM) {
                ceilingFan.medium();
        } else if (prevSpeed == CeilingFan.LOW) {
                ceilingFan.low();
        } else if (prevSpeed == CeilingFan.OFF) {
                ceilingFan.off();
        }
}
}

public class CeilingFanHighCommand implements Command {
CeilingFan ceilingFan;
int prevSpeed;

public CeilingFanHighCommand(CeilingFan ceilingFan) {
        this.ceilingFan = ceilingFan;
}

public void execute() {
        prevSpeed = ceilingFan.getSpeed();
        ceilingFan.high();
}
```

```
public void undo() {
        if (prevSpeed == CeilingFan.HIGH) {
                ceilingFan.high();
        } else if (prevSpeed == CeilingFan.MEDIUM) {
                ceilingFan.medium();
        } else if (prevSpeed == CeilingFan.LOW) {
                ceilingFan.low();
        } else if (prevSpeed == CeilingFan.OFF) {
                ceilingFan.off();
        }
}
}

public class CeilingFanLowCommand implements Command {
CeilingFan ceilingFan;
int prevSpeed;

public CeilingFanLowCommand(CeilingFan ceilingFan) {
        this.ceilingFan = ceilingFan;
}

public void execute() {
        prevSpeed = ceilingFan.getSpeed();
        ceilingFan.low();
}

public void undo() {
        if (prevSpeed == CeilingFan.HIGH) {
                ceilingFan.high();
        } else if (prevSpeed == CeilingFan.MEDIUM) {
                ceilingFan.medium();
        } else if (prevSpeed == CeilingFan.LOW) {
                ceilingFan.low();
        } else if (prevSpeed == CeilingFan.OFF) {
                ceilingFan.off();
        }
```

```
}
}
public class CeilingFan {
public static final int HIGH = 3;
public static final int MEDIUM = 2;
public static final int LOW = 1;
public static final int OFF = 0;
String location;
int speed;

public CeilingFan(String location) {
        this.location = location;
        speed = OFF;
}

public void high() {
        speed = HIGH;
        System.out.println(location + " ceiling fan is on high");
}

public void medium() {
        speed = MEDIUM;
        System.out.println(location + " ceiling fan is on medium");
}

public void low() {
        speed = LOW;
        System.out.println(location + " ceiling fan is on low");
}

public void off() {
        speed = OFF;
        System.out.println(location + " ceiling fan is off");
}

public int getSpeed() {
        return speed;
```

```
}
}
```

2) Write a python program to implement Linear SVM.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn import svm
data=pd.read_csv('letterdata.csv')
data.head()
def getaccuracy(testset,prediction):
    correct=0
    for x in range(len(testset)):
        if(testset[x]==prediction[x]):
            correct=correct+1
    return (correct/float(len(testset)))*100.0
data.isnull().values.any()
X,Y=np.array(data)[:,1:16],np.array(data.letter)[:]
X_train=X[:16000,:]
X_test=X[16001:,:]
Y_train=Y[:16000]
Y_test=Y[16001:]
clf=svm.SVC(gamma=0.025,C=3)
clf.fit(X_train,Y_train)
Y_predict=clf.predict(X_test)
getaccuracy(Y_test,Y_predict)
y_g=(np.column_stack([Y_test,Y_predict]))
#column stack used for matching supose x=1,2,3,4,5 and y= 10 20 30 40 match
x axis on 1 check on y axis on 10 ...
print(y_g)
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
# import some data to play with
```

```
iris = datasets.load_iris()
X = iris.data[:, :2] # we only take the first two features. We could
 # avoid this ugly slicing by using a two-dim dataset
y = iris.target
# we create an instance of SVM and fit out data. We do not scale our
# data since we want to plot the support vectors
C = 1.0 # SVM regularization parameter
svc = svm.SVC(kernel='linear', C=1,gamma=10).fit(X, y)
# create a mesh to plot in
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
h = (x_max / x_min)/100
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
 np.arange(y_min, y_max, h))
plt.subplot(1, 1, 1)
Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('SVC with linear kernel')
plt.show()
svc = svm.SVC(kernel='rbf', C=1,gamma=10).fit(X, y)
plt.subplot(1, 1, 1)
Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
svc = svm.SVC(kernel='poly', C=1,gamma=100).fit(X, y)
plt.subplot(1, 1, 1)
Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
```

3) Design a Django application: A public site in which user can pick their favourite

programming language and vote.

```
var http = require('http');  // includes the http module
var uc = require('upper-case'); // include the upper-case module
http.createServer(function (req, res) {
   res.writeHead(200, {'Content-Type': 'text/html'});
   res.write(("hello world!").toUpperCase()); // assign the upper-case module
   res.end();
}).listen(8080);
```

Slip-22

Q1 Design simple HR Application using Spring Framework.

-spring

2) Write a Python program to prepare Scatter Plot for Iris Dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
iris = pd.read_csv("Iris.csv")   # Reading the dataset “Iris.csv”.
print (iris.head(10))   # head() will display the top rows of the dataset, the default value of this function is 5,
#that is it will show top 5 rows when no argument is given to it.
plt.plot(iris.Id, iris["SepalLengthCm"],"r--")
plt.show    #plt.show () will display the current figure that you are working on
iris.plot(kind ="scatter", x ='SepalLengthCm', y ='PetalLengthCm')
plt.grid()    # grid () function to add grid lines to the plot
```

3) Design a Django application: An admin site that lets you add, change and delete programming languages.

```
const fs = require('fs');

console.log("\nFile Contents of file before append:",
a=fs.readFileSync("file1.txt", "utf8"));

fs.appendFile("file2.txt", a, (err) => {
if (err) {
```

```
    console.log(err);
}
else {
    console.log("\nFile Contents of file after append:",
    fs.readFileSync("file2.txt", "utf8"));
}
});
```

Slip-23

Q1 Write a Java Program to implement State Pattern for Gumball Machine. Create instance variable that holds current state from there, we just need to handle all actions, behaviors and state transition that can happen.

State.java

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

SoldStat.java

```
public 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";
        }
}
```

SoldOutState.java

```
public 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";
        }
}
```

NoQuarterState.java

```
public 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";
	}
}
```

HasQuarterState.java

```
public 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";
        }

}
```

GumballMachine.java

```
public class GumballMachine {

        State soldOutState;
        State noQuarterState;
        State hasQuarterState;
        State soldState;

        State state;
        int count = 0;

        public GumballMachine(int numberGumballs) {
                soldOutState = new SoldOutState(this);
                noQuarterState = new NoQuarterState(this);
                hasQuarterState = new HasQuarterState(this);
                soldState = new SoldState(this);

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

        public void insertQuarter() {
```

```
            state.insertQuarter();
        }

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

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

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

        int getCount() {
            return count;
        }

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

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

```
    public State getSoldOutState() {
        return soldOutState;
    }

    public State getNoQuarterState() {
        return noQuarterState;
    }

    public State getHasQuarterState() {
        return hasQuarterState;
    }

    public State getSoldState() {
        return soldState;
    }

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

GumballMachineTestDrive.java

```
public class GumballMachineTestDrive{

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

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

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

                System.out.println(gumballMachine);

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

                gumballMachine.refill(3);
                gumballMachine.insertQuarter();
                gumballMachine.turnCrank();

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

2) Write a python program to find all null values in a given dataset and remove them.

```
import pandas as pd
import numpy as np
dict={'first score':[100,90,np.nan,95],
     'second score':[30,45,56,np.nan],
     'third score':[np.nan,40,80,98]}
df=pd.DataFrame(dict)
df.head()
df.isnull()
df.notnull()
#df=pd.DataFrame(dict)
df.fillna(0)
df.fillna(method='pad')
df.fillna(method='bfill')
df.replace(to_replace=np.nan,value=-99)
df.dropna()
df.dropna(axis=1)
new_data=df.dropna(axis=0)
```

new_data

3) Create your own blog using Django.

```
var fs = require('fs');
//var http = require('http');
//http.createServer(function (req, res) {
// Use fs.readFile() method to read the file
fs.readFile('file1.txt', 'utf8', function(err, data){
   /*if (err){
      res.writeHead(404, {'Content-Type': 'text/html'});
      return res.end("404 Not Found");
    }*/
   // Display the file content
   if(err) return console.error(err);
   console.log(data);
});
console.log('readFile called');
```

Slip-24

Q1 Write a Java Program to implement Iterator Pattern for Designing Menu like Breakfast,Lunch or Dinner Menu.

Menu.java

```
import java.util.Iterator;

public interface Menu
{
public Iterator<MenuItem> createIterator();
}
```

CafeMenu.java

```
import java.util.*;
public class CafeMenu implements Menu
{
HashMap<String, MenuItem> menuItems = new HashMap<String,
MenuItem>();
public CafeMenu()
{
```

```
addItem("Veggie Burger and French Fries","Veggie burger on a whole wheat bun, lettuce, tomato, and fries",true, 40);
addItem("Soup of the day","A cup of the soup of the day, with a side salad",false, 70);
addItem("Burrito","A large burrito, with whole pinto beans, salsa, guacamole",true, 80);
}
public void addItem(String name, String description,boolean vegetarian, double price)
{
MenuItem menuItem = new MenuItem(name, description, vegetarian, price);
menuItems.put(name, menuItem);
}
public Map<String, MenuItem> getItems()
{
return menuItems;
}
public Iterator<MenuItem> createIterator()
{
return menuItems.values().iterator();
}
}
```

DinerMenu.java

```
import java.util.Iterator;

public class DinerMenu implements Menu
{
static final int MAX_ITEMS = 6;
int numberOfItems = 0;
MenuItem[] menuItems;

public DinerMenu()
{
menuItems = new MenuItem[MAX_ITEMS];
```

```
addItem("Vegetarian BLT","(Fakin') Bacon with lettuce & tomato on whole
wheat", true, 80);
addItem("BLT","Bacon with lettuce & tomato on whole wheat", false, 90);
addItem("Soup of the day","Soup of the day, with a side of potato salad", false,
70);
addItem("Hotdog","A hot dog, with sauerkraut, relish, onions, topped with
cheese",false, 60);
addItem("Steamed Veggies and Brown Rice","A medly of steamed vegetables
over brown rice", true, 99);
addItem("Pasta","Spaghetti with Marinara Sauce, and a slice of sourdough
bread",true, 89);
}
public void addItem(String name, String description,boolean vegetarian, double
price)
{
MenuItem menuItem = new MenuItem(name, description, vegetarian, price);
if (numberOfItems >= MAX_ITEMS)
{
System.err.println("Sorry, menu is full!  Can't add item to menu");
}
else
{
menuItems[numberOfItems] = menuItem;
numberOfItems = numberOfItems + 1;
}
}
public MenuItem[] getMenuItems()
{
return menuItems;
}
public Iterator<MenuItem> createIterator()
{
return new DinerMenuIterator(menuItems);
//return new AlternatingDinerMenuIterator(menuItems);
}
// other menu methods here
}
```

DinerMenuIterator.java

```
import java.util.Iterator;

public class DinerMenuIterator implements Iterator<MenuItem> {
	MenuItem[] list;
	int position = 0;

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

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

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

	public void remove() {
		if (position <= 0) {
			throw new IllegalStateException
				("You can't remove an item until you've done at least
one next()");
		}
		if (list[position-1] != null) {
			for (int i = position-1; i < (list.length-1); i++) {
				list[i] = list[i+1];
			}
			list[list.length-1] = null;
		}
```

```
        }
}
```

MenuItem.java

```
public class MenuItem {
        String name;
        String description;
        boolean vegetarian;
        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 double getPrice() {
                return price;
        }

        public boolean isVegetarian() {
                return vegetarian;
        }
}
```

MenuTestDrive.java

```
public class MenuTestDrive {
	public static void main(String args[]) {
		PancakeHouseMenu pancakeHouseMenu = new
PancakeHouseMenu();
		DinerMenu dinerMenu = new DinerMenu();
		CafeMenu cafeMenu = new CafeMenu();

		Waitress waitress = new Waitress(pancakeHouseMenu,
dinerMenu, cafeMenu);

		waitress.printMenu();
		waitress.printVegetarianMenu();

		System.out.println("\nCustomer asks, is the Hotdog vegetarian?");
		System.out.print("Waitress says: ");
		if (waitress.isItemVegetarian("Hotdog")) {
			System.out.println("Yes");
		} else {
			System.out.println("No");
		}
		System.out.println("\nCustomer asks, are the Waffles
vegetarian?");
		System.out.print("Waitress says: ");
		if (waitress.isItemVegetarian("Waffles")) {
			System.out.println("Yes");
		} else {
			System.out.println("No");
		}
	}
}
```

PancakeHouseMenu.java

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

public class PancakeHouseMenu implements Menu {
	ArrayList<MenuItem> menuItems;
```

```
	public PancakeHouseMenu() {
		menuItems = new ArrayList<MenuItem>();

		addItem("K&B's Pancake Breakfast",
			"Pancakes with scrambled eggs and toast",
			true,
			90);

		addItem("Regular Pancake Breakfast",
			"Pancakes with fried eggs, sausage",
			false,
			80);

		addItem("Blueberry Pancakes",
			"Pancakes made with fresh blueberries and blueberry syrup",
			true,
			60);

		addItem("Waffles",
			"Waffles with your choice of blueberries or strawberries",
			true,
			70);
	}

	public void addItem(String name, String description,
			boolean vegetarian, double price)
	{
		MenuItem menuItem = new MenuItem(name, description,
vegetarian, price);
		menuItems.add(menuItem);
	}

	public ArrayList<MenuItem> getMenuItems() {
		return menuItems;
	}

	public Iterator<MenuItem> createIterator() {
```

```
        return menuItems.iterator();
    }

    // other menu methods here
}
```

Waitress.java

```
import java.util.Iterator;

public class Waitress {
    Menu pancakeHouseMenu;
    Menu dinerMenu;
    Menu cafeMenu;

    public Waitress(Menu pancakeHouseMenu, Menu dinerMenu, Menu
cafeMenu) {
        this.pancakeHouseMenu = pancakeHouseMenu;
        this.dinerMenu = dinerMenu;
        this.cafeMenu = cafeMenu;
    }

    public void printMenu() {
        Iterator<MenuItem> pancakeIterator =
pancakeHouseMenu.createIterator();
        Iterator<MenuItem> dinerIterator = dinerMenu.createIterator();
        Iterator<MenuItem> cafeIterator = cafeMenu.createIterator();

        System.out.println("MENU\n----\nBREAKFAST");
        printMenu(pancakeIterator);
        System.out.println("\nLUNCH");
        printMenu(dinerIterator);
        System.out.println("\nDINNER");
        printMenu(cafeIterator);
    }

    private void printMenu(Iterator<MenuItem> iterator) {
        while (iterator.hasNext()) {
```

```
                MenuItem menuItem = iterator.next();
                System.out.print(menuItem.getName() + ", ");
                System.out.print(menuItem.getPrice() + " -- ");
                System.out.println(menuItem.getDescription());
            }
        }

        public void printVegetarianMenu() {
            System.out.println("\nVEGETARIAN MENU\n----------");
            printVegetarianMenu(pancakeHouseMenu.createIterator());
            printVegetarianMenu(dinerMenu.createIterator());
            printVegetarianMenu(cafeMenu.createIterator());
        }

        public boolean isItemVegetarian(String name) {
            Iterator<MenuItem> pancakeIterator =
pancakeHouseMenu.createIterator();
            if (isVegetarian(name, pancakeIterator)) {
                return true;
            }
            Iterator<MenuItem> dinerIterator = dinerMenu.createIterator();
            if (isVegetarian(name, dinerIterator)) {
                return true;
            }
            Iterator<MenuItem> cafeIterator = cafeMenu.createIterator();
            if (isVegetarian(name, cafeIterator)) {
                return true;
            }
            return false;
        }


        private void printVegetarianMenu(Iterator<MenuItem> iterator) {
            while (iterator.hasNext()) {
                MenuItem menuItem = iterator.next();
                if (menuItem.isVegetarian()) {
                    System.out.print(menuItem.getName() + ", ");
```

```
                        System.out.print(menuItem.getPrice() + " -- ");
                        System.out.println(menuItem.getDescription());
                    }
                }
        }

        private boolean isVegetarian(String name, Iterator<MenuItem> iterator) {
                while (iterator.hasNext()) {
                        MenuItem menuItem = iterator.next();
                        if (menuItem.getName().equals(name)) {
                                if (menuItem.isVegetarian()) {
                                        return true;
                                }
                        }
                }
                return false;
        }
}
MenuTestDrive.java
public class MenuTestDrive {
        public static void main(String args[]) {
                PancakeHouseMenu pancakeHouseMenu = new
PancakeHouseMenu();
                DinerMenu dinerMenu = new DinerMenu();
                CafeMenu cafeMenu = new CafeMenu();

                Waitress waitress = new Waitress(pancakeHouseMenu,
dinerMenu, cafeMenu);

                waitress.printMenu();
                waitress.printVegetarianMenu();

                System.out.println("\nCustomer asks, is the Hotdog vegetarian?");
                System.out.print("Waitress says: ");
                if (waitress.isItemVegetarian("Hotdog")) {
                        System.out.println("Yes");
                } else {
```

```
                System.out.println("No");
            }
            System.out.println("\nCustomer asks, are the Waffles
vegetarian?");
            System.out.print("Waitress says: ");
            if (waitress.isItemVegetarian("Waffles")) {
                System.out.println("Yes");
            } else {
                System.out.println("No");
            }
        }
}
```

2) Write a python program to make Categorical values in numeric format for a given dataset.

```
import pandas as pd
#load Iris data set
iris = pd.read_csv('Iris.csv')
iris.head()
iris['code']=pd.factorize(iris.Species)[0]
iris.tail()
iris.code.value_counts()
```

3) Implement Login System using Django.

```
var http = require('http');
var formidable = require('formidable');
http.createServer(function (req, res) {
  if (req.url == '/fileupload') {
    var form = new formidable.IncomingForm();
    form.parse(req, function (err, fields, files) {
      res.write('File uploaded');
      res.end();
    });
  } else {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write('<form action="fileupload" method="post"
enctype="multipart/form-data">');
    res.write('<input type="file" name="filetoupload"><br>');
```

```
    res.write('<input type="submit">');
    res.write('</form>');
    return res.end();
  }
}).listen(8080);
```

Slip-25

Q1 Write a Java Program to implement Singleton pattern for multithreading.

```
class SingleObject
{
        private static SingleObject instance = new SingleObject();

        private SingleObject(){}

   //Get the only object available
   public static SingleObject getInstance(){
      return instance;
   }

   public void showMessage(){
      System.out.println("Hello Everyone!!!!");
   }
}
 class singletone{
   public static void main(String[] args) {
      //Get the only object available
      SingleObject object = SingleObject.getInstance();
      object.showMessage();
   }
}
```

2) Write a python program to Implement Simple Linear Regression for predicting house price.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_predict
data = pd.read_csv(r'kc_house_data.csv')
data.head(5)
print(data.shape)
# Make a list of importatnt feature which is needed to be incuding in training
data
f = ['price', 'bedrooms', 'bathrooms', 'sqft_living', 'floors', 'condition',
'sqft_above', 'sqft_basement', 'yr_built',
    'yr_renovated']
data = data[f]
print(data.shape)
# Drop the missing values
data = data.dropna()
print(data.shape)
# Get the statictial information of the dataset
data.describe()
# Now,Divide the dataset into two parts:independent variable and dependent
variable
X = data[f[1:]]
y = data['price']
# Split the dataset into training data and testing data
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2,
random_state=42)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
# Fit the regression model
lr = LinearRegression() # Create object of linear regression class
lr.fit(X_train,y_train) #fit training data
print(lr.coef_)
# Create the Prediction
y_test_predict = lr.predict(X_test)
print(y_test_predict.shape)
# Plot the error
g=plt.plot((y_test - y_test_predict),marker='o',linestyle='')
```

```
# # Fit the regression model without b(w0)
lr = LinearRegression(fit_intercept=False)
lr.fit(X_train,y_train)
y_test_predict = lr.predict(X_test)
g=plt.plot((y_test - y_test_predict),marker='o',linestyle='')
```

3) Create a Simple Web Server using node js.

```
var http = require('http'); // 1 - Import Node.js core module
var server = http.createServer(function (req, res) {   // 2 - creating server
   //handle incomming requests here..
});
server.listen(5000); //3 - listen for any incoming requests
console.log('Node.js web server at port 5000 is running..')
```

Slip-26

Q1 Write a Java Program to implement Strategy Pattern for Duck Behavior. Create instance variable that holds current state of Duck from there, we just need to handle all Flying Behaviours and Quack Behavior.

Duck.Java

```
public abstract class Duck {
  FlyBehaviour flyBehaviour;
  QuackBehaviour quackBehaviour;
  public Duck() {
  }
  public abstract void display();
  public void performFly() {
    flyBehaviour.fly();
  }

  public void performQuack() {
    quackBehaviour.quack();
  }
  public void swim() {
    System.out.println("All ducks float, even decoys!);
  }
  public void setFlyBehaviour(FlyBehaviour fb) {
    flyBehaviour = fb;
```

```
  }
  public void setQuackBehaviour(QuackBehaviour qb) {
    quackBehaviour qb;
  }
}
```

MallardDuck.Java

```
public class MallardDuck extends Duck {

  public MallardDuck() {
    quackBehaviour = new Quack();
    flyBehaviour = new FlyWithWings();
  }
  public void display() {
    System.out.println("I'm a real Mallard duck");
  }
}
```

ModelDuck.Java

```
public class ModolDuck extends Duck {
  public ModelDuck() {
    flyBehaviour = new FlyNoWay();
    quackBehaviour = new Quack();
  }
  public void display() {
    System.out.println("I'm a model duck");
  }
}
```

FlyBehaviour.Java

```
public interface FlyBehaviour {
  public void fly():
}
```

FlyWithWings.Java

```
public class FlyWithWings implements FlyBehaviour {
  public void fly() {
```

```
    System.out.println("I'm flying!!");
  }
}
```

FlyNoWay.Java

```
public class FlyNoWay implements FlyBehaviour {
  public void fly() {
    System.out.println("I can't fly");
  }
}
```

FlyRocketPowered.Java

```
public class FlyRocketPowered implements FlyBehaviour {
  public void fly() {
    System.out.println("I'm flying with a rocket!");
  }
}
```

QuackBehaviour.Java

```
public interface QuackBehaviour {
  public void quack() {
    System.out.println("Quack");
  }
}
```

Quack.Java

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

MuteQuack.Java

```
public class MuteQuack implements QuackBehaviour {
  public void quack() {
    System.out.println("<< Silence >>");
  }
```

```
}
```

Squeak.Java

```
public class Squeak implements QuackBehaviour {
  public void quack() {
    System.out.println("Squeak");
  }
}
```

MiniDuckSimulator.Java

```
public class MiniDuckSimulator {
  public static void main(String[] args) {
    Duck mallard = new MallardDuck();
    mallard.performQuack();
    mallard.performFly();
  }
}
```

MiniDuckSimulator.Java - Model Duck Edition

```
public class MiniDuckSimulator {
  public static void main(String[] args) {
    Duck mallard = new MallardDuck();
    mallard.performQuack();
    mallard.performFly();

    Duck model = new ModelDuck();
    model.performFly();
    model.setFlyBehaviour(new FlyRocketPowered());
    model.performFly();
  }
}
```

2) Write a python program to implement Multiple Linear Regression for given dataset.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
dataset = pd.read_csv('50_Startups.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [3])],
remainder='passthrough')
X = np.array(ct.fit_transform(X))
print(X)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
df = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})
df
```

3) Create a Node.js file that demonstrates create database and table in MySQL.

```
var mysql = require('mysql');
var con = mysql.createConnection({
host: "localhost",
user: "root",
password: "Root@123",
database: "node"
});
con.connect(function(err) {
if (err) throw err;
console.log("Connected!");
con.query("CREATE DATABASE Noded", function (err, result) {
if (err) throw err;
console.log("Database created");
});
});
var sql = "CREATE TABLE customers (name VARCHAR(25), address
VARCHAR(25))";
```

```
con.query(sql, function (err, result) {
  if (err) throw err;
  console.log("Table created");
});
```

Slip-27

Q1 Write a Java Program to implement Abstract Factory Pattern for Shape interface.

Shape.java

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

RoundedSquare.java

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

Rectangle.java

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

Step 3

Create an Abstract class to get factories for Normal and Rounded Shape Objects.

AbstractFactory.java

```
public abstract class AbstractFactory {
  abstract Shape getShape(String shapeType) ;
}
```

Step 4

Create Factory classes extending AbstractFactory to generate object of concrete class based on given information.

ShapeFactory.java

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

RoundedShapeFactory.java

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

Step 5

Create a Factory generator/producer class to get factories by passing an information such as Shape

FactoryProducer.java

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

Step 6

Use the FactoryProducer to get AbstractFactory in order to get factories of concrete classes by passing an information such as type.

AbstractFactoryPatternDemo.java

```
public class AbstractFactoryPatternDemo {
  public static void main(String[] args) {
    //get shape factory
    AbstractFactory shapeFactory = FactoryProducer.getFactory(false);
    //get an object of Shape Rectangle
    Shape shape1 = shapeFactory.getShape("RECTANGLE");
    //call draw method of Shape Rectangle
    shape1.draw();
    //get an object of Shape Square
    Shape shape2 = shapeFactory.getShape("SQUARE");
    //call draw method of Shape Square
    shape2.draw();
    //get shape factory
    AbstractFactory shapeFactory1 = FactoryProducer.getFactory(true);
    //get an object of Shape Rectangle
    Shape shape3 = shapeFactory1.getShape("RECTANGLE");
    //call draw method of Shape Rectangle
```

```
      shape3.draw();
      //get an object of Shape Square
      Shape shape4 = shapeFactory1.getShape("SQUARE");
      //call draw method of Shape Square
      shape4.draw();

   }
}
```

2) Write a python program to implement Polynomial Linear Regression for given dataset

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('Position_Salaries.csv')
X = dataset.iloc[:, 1:-1].values
y = dataset.iloc[:, -1].values
dataset.head(5)
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
poly_reg = PolynomialFeatures(degree = 4)
X_poly = poly_reg.fit_transform(X)
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y)
y_pred = lin_reg.predict(X_poly)
df = pd.DataFrame({'Real Values':y, 'Predicted Values':y_pred})
df
X_grid = np.arange(min(X), max(X), 0.1)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X, y, color = 'red')
plt.scatter(X, y_pred, color = 'green')
plt.plot(X_grid,
lin_reg.predict(poly_reg.fit_transform(X_grid)), color =
'black')
plt.title('Polynomial Regression')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```

3) Create your Django app in which after running the server, you should see on the browser, the text “Hello! I am learning Django”, which you defined in the index view.

```
var mysql = require('mysql');
var con = mysql.createConnection({
host: "localhost",
user: "root",
password: "Root@123",
database: "node"
});
con.connect(function(err) {
if (err) throw err;
console.log("Connected!");
con.query("CREATE DATABASE Noded", function (err, result) {
if (err) throw err;
console.log("Database created");
});
});
var sql = "CREATE TABLE customers (name VARCHAR(25), address
VARCHAR(25))";
con.query(sql, function (err, result) {
  if (err) throw err;
  console.log("Table created");
});
```

Slip-28

Q1 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().

```
import java.util.*;
interface Observer {
    public void update(float temp, float humidity, float pressure);
}
```

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

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

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

    public WeatherData() {
        observers = new ArrayList<>();
    }

    public void registerObserver(Observer o) {
        observers.add(o);
    }

    public void removeObserver(Observer o) {
        int i = observers.indexOf(o);
        if (i >= 0) {
            observers.remove(i);
        }
    }

    public void notifyObservers() {
        for (int i = 0; i < observers.size(); i++) {
            Observer observer = (Observer) observers.get(i);
            observer.update(temperature, humidity, pressure);
        }
    }
```

```
    public void measurementsChanged() {
        notifyObservers();
    }

    public void setMeasurements(float temperature, float humidity, float
pressure) {
        this.temperature = temperature;
        this.humidity = humidity;
        this.pressure = pressure;
        measurementsChanged();
    }

    public float getTemperature() {
        return temperature;
    }

    public float getHumidity() {
        return humidity;
    }

    public float getPressure() {
        return pressure;
    }
}

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

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

    public void update(float temp, float humidity, float pressure) {
```

```
        lastPressure = currentPressure;
        currentPressure = pressure;

        display();
    }

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

class HeatIndexDisplay implements Observer, DisplayElement {
    float heatIndex = 0.0f;
    private WeatherData weatherData;

    public HeatIndexDisplay(WeatherData weatherData) {
        this.weatherData = weatherData;
        weatherData.registerObserver(this);
    }
    public void update(float t, float rh, float pressure) {
        heatIndex = computeHeatIndex(t, rh);
        display();
    }
    private float computeHeatIndex(float t, float rh) {
        float index = (float) ((16.923 + (0.185212 * t) + (5.37941 * rh) - (0.100254
* t * rh)
                + (0.00941695 * (t * t)) + (0.00728898 * (rh * rh))
                + (0.000345372 * (t * t * rh)) - (0.000814971 * (t * rh * rh)) +
                (0.0000102102 * (t * t * rh * rh)) - (0.000038646 * (t * t * t)) +
(0.0000291583 *
```

```
                    (rh * rh * rh))
            + (0.00000142721 * (t * t * t * rh)) +
            (0.000000197483 * (t * rh * rh * rh)) - (0.0000000218429 * (t * t * t *
rh * rh)) +
            0.000000000843296 * (t * t * rh * rh * rh)) -
            (0.0000000000481975 * (t * t * t * rh * rh * rh)));
        return index;
    }
    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;
    private WeatherData weatherData;

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

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

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

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

        display();
```

```
    }

    public void display() {
        System.out.println("Avgerage/Maximum/Minimum temperature = " +
(tempSum / numReadings)
                + "/" + maxTemp + "/" + minTemp);
    }
}

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

    public CurrentConditionsDisplay(Subject weatherData) {
        this.weatherData = weatherData;
        weatherData.registerObserver(this);
    }

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

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

class weather {

    public static void main(String[] args) {
        WeatherData weatherData = new WeatherData();
```

```
        CurrentConditionsDisplay currentDisplay = new
CurrentConditionsDisplay(weatherData);
        StatisticsDisplay statisticsDisplay = new StatisticsDisplay(weatherData);
        ForecastDisplay forecastDisplay = new ForecastDisplay(weatherData);

        weatherData.setMeasurements(70, 55, 40.4f);
        weatherData.setMeasurements(72, 60, 39.2f);
        weatherData.setMeasurements(68, 80, 39.2f);
    }
}
```

2) Write a python program to implement Naive Bayes.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
data=pd.read_csv('pima-indians-diabetes.csv')
data.shape
data.isnull().sum()
data.isnull().values.any()
data.dtypes
#visualisation
data.hist()
columns=list(data)[0:-1]
data[columns].hist()
#identifty the correlation
data.corr()
sns.heatmap(data.corr(),annot=True)
sns.pairplot(data)
#calculate diabetes ratio of true or false target varible
n_true=len(data.loc[data['class']==True])
n_false=len(data.loc[data['class']==False])
print("No.of true cases:{0}
{1}%".format(n_true,(n_true/(n_true+n_false))*100))
print("No.of false cases:{0}
{1}%".format(n_false,(n_false/(n_true+n_false))*100))
```

```
#split the data
from sklearn.model_selection import train_test_split
x=data.drop('class',axis=1)
y=data['class']
X_train,X_test,Y_train,Y_test=train_test_split(x,y,test_size=0.30,random_state
=1)
from sklearn.impute import SimpleImputer
rep_0=SimpleImputer(missing_values=0,strategy='mean')
cols=X_train.columns
X_train=pd.DataFrame(rep_0.fit_transform(X_train))
X_test=pd.DataFrame(rep_0.fit_transform(X_test))
X_train.columns=cols
X_test.columns=cols
X_train.head()
from sklearn.naive_bayes import GaussianNB
diab_mode=GaussianNB()
diab_mode.fit(X_train,Y_train)
diab_train_predict=diab_mode.predict(X_train)
from sklearn import metrics
print("Model
Accuracy:{0}".format(metrics.accuracy_score(Y_train,diab_train_predict)))
diab_train_predict=diab_mode.predict(X_test)
from sklearn import metrics
print("Model
Accuracy:{0}".format(metrics.accuracy_score(Y_test,diab_train_predict)))
cm1=metrics.confusion_matrix(Y_test,diab_train_predict,labels=[1,0])
df_cm1=pd.DataFrame(cm1,index=[i for i in['1','0']],columns=[i for i in['predict
1','predict o']] )
df_cm1
```

3) Create your own blog using Django

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: 'localhost',
  user: "root",
  password: "Root@123",
  database:'node'
```

```
});
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
});
con.query('SELECT * FROM customers', (err,rows) => {
    if(err) throw err;
    console.log('Data received from Db:');
    console.log(rows);
  });
```

Slip-29

Q1 Write a Java Program to implement State Pattern for Gumball Machine. Create instance variable that holds current state from there, we just need to handle all actions, behaviors and state transition that can happen.

State.java

```
public interface State {

        public void insertQuarter();
        public void ejectQuarter();
        public void turnCrank();
        public void dispense();
        public void refill();
}
```

SoldStat.java

```
public 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";
        }
}
SoldOutState.java
public 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";
        }
}
```

NoQuarterState.java

```
public 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";
        }
```

```
}
```
HasQuarterState.java
```
public 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";
	}

}
```
GumballMachine.java
```
public class GumballMachine {

	State soldOutState;
	State noQuarterState;
	State hasQuarterState;
	State soldState;

	State state;
```

```
int count = 0;

public GumballMachine(int numberGumballs) {
        soldOutState = new SoldOutState(this);
        noQuarterState = new NoQuarterState(this);
        hasQuarterState = new HasQuarterState(this);
        soldState = new SoldState(this);

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

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

int getCount() {
        return count;
}
```

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

    public State getSoldOutState() {
        return soldOutState;
    }

    public State getNoQuarterState() {
        return noQuarterState;
    }

    public State getHasQuarterState() {
        return hasQuarterState;
    }

    public State getSoldState() {
        return soldState;
    }

        public String toString() {
                StringBuffer result = new StringBuffer();
                result.append("\nMighty Gumball, Inc.");
                result.append("\nJava-enabled Standing Gumball Model #2004");
                result.append("\nInventory: " + count + " gumball");
                if (count != 1) {
                        result.append("s");
                }
```

```
            result.append("\n");
            result.append("Machine is " + state + "\n");
            return result.toString();
        }
}
```

GumballMachineTestDrive.java

```
public class GumballMachineTestDrive{

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

        System.out.println(gumballMachine);

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

        System.out.println(gumballMachine);

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

        gumballMachine.refill(3);
        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();

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

2) Write a python program to implement Decision Tree whether or not to play Tennis.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.model_selection import train_test_split
from scipy.stats import zscore
import seaborn as sns
data=pd.read_csv('play_tennis.csv')
data.isnull().sum()
data.dtypes
data.head()
data.outlook.value_counts()
from sklearn.preprocessing import LabelEncoder
l=LabelEncoder()
for i in data.columns:
    if data[i].dtypes=='object' or data[i].dtypes=='bool':
        data[i]=pd.Categorical(data[i])

for i in data.columns:
        data[i]=l.fit_transform(data[i])
data.dtypes
data.head()
x=data.drop(['play'],axis=1)
y=data['play']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=1
)
dtree=DecisionTreeClassifier(criterion='gini',random_state=1)
dtree.fit(x_train,y_train)
print(dtree.score(x_train,y_train)) #data is over fitted so we use max_depth =5
means prunning technique
print(dtree.score(x_test,y_test))
dtree1=DecisionTreeClassifier(criterion='gini',max_depth=5,random_state=1)
dtree1.fit(x_train,y_train)
print(dtree1.score(x_train,y_train)) #data is over fitted so we use max_depth =5
means prunning technique
print(dtree1.score(x_test,y_test))
y_predict=dtree.predict(x_test)
from sklearn import metrics
cm=metrics.confusion_matrix(y_test,y_predict,labels=[1,0])
df_cm=pd.DataFrame(cm,index=[i for i in['1','0']],columns=[i for i in['predicted
1','predicted 0']] )
```

```
df_cm
sns.heatmap(df_cm,annot=True)
from sklearn.metrics import classification_report
m=classification_report(y_test,y_predict)
print(m)
```

3) Create a clone of the “Hacker News” website.

```
var mysql = require('mysql');
var con = mysql.createConnection({

  host: "localhost",
  user: "root",
  password: "Root@123",
  database: "node"
});
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "INSERT INTO student (rollno,namee, percentage) VALUES ?";
var values = [  [1,'abc', 77.6],  [2,'def', 89.6],  [3,'ghi', 91.6]  ];
con.query(sql, [values], function (err, result)
 {
    if (err) throw err;
    console.log("Number of records inserted: " + result.affectedRows);
  });
con.query("SELECT * FROM student", function (err, result, fields) {
    if (err) throw err;
    console.log(result);
  });
});
```

Slip-30

Q1 Write a Java Program to implement Factory method for Pizza Store with createPizza(), orederPizza(), prepare(), Bake(), cut(), box(). Use this to create variety of pizza’s like NyStyleCheesePizza, ChicagoStyleCheesePizza etc.

```
import java.util.ArrayList;
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 ChicagoStyleCheesePizza extends Pizza
{public ChicagoStyleCheesePizza()
{
name = "Chicago Style Deep Dish Cheese Pizza";
dough = "Extra Thick Crust Dough";
sauce = "Plum Tomato Sauce";
toppings.add("Shredded Mozzarella Cheese");
}

void cut()
{System.out.println("Cutting the pizza into square slices");
}}
class ChicagoStyleClamPizza extends Pizza
{public ChicagoStyleClamPizza()
{
name = "Chicago Style Clam Pizza";
```

```
dough = "Extra Thick Crust Dough";
sauce = "Plum Tomato Sauce";
toppings.add("Shredded Mozzarella Cheese");
toppings.add("Frozen Clams from Chesapeake Bay");
}
void cut()
{System.out.println("Cutting the pizza into square slices");
}}
class ChicagoStylePepperoniPizza extends Pizza
{public ChicagoStylePepperoniPizza()
{
name = "Chicago Style Pepperoni Pizza";
dough = "Extra Thick Crust Dough";
sauce = "Plum Tomato Sauce";
toppings.add("Shredded Mozzarella Cheese");
toppings.add("Black Olives");
toppings.add("Spinach");
toppings.add("Eggplant");
toppings.add("Sliced Pepperoni");
}
void cut()
{System.out.println("Cutting the pizza into square slices");
}}
class ChicagoStyleVeggiePizza extends Pizza
{public ChicagoStyleVeggiePizza()
{name = "Chicago Deep Dish Veggie Pizza";
dough = "Extra Thick Crust Dough";
sauce = "Plum Tomato Sauce";
toppings.add("Shredded Mozzarella Cheese");
toppings.add("Black Olives");
toppings.add("Spinach");
toppings.add("Eggplant");
}
```

```
void cut()
{System.out.println("Cutting the pizza into square slices");
}}
class DependentPizzaStore
{public Pizza createPizza(String style, String type)
{ Pizza pizza = null;
if (style.equals("NY"))
{if (type.equals("cheese"))
{pizza = new NYStyleCheesePizza();
}
else if (type.equals("veggie"))
{pizza = new NYStyleVeggiePizza();
}
else if (type.equals("clam"))
{pizza = new NYStyleClamPizza();
}
else if (type.equals("pepperoni"))
{pizza = new NYStylePepperoniPizza();
}}
else if (style.equals("Chicago"))
{if (type.equals("cheese"))
{pizza = new ChicagoStyleCheesePizza();
}
else if (type.equals("veggie"))
{pizza = new ChicagoStyleVeggiePizza();
}
else if (type.equals("clam"))
{pizza = new ChicagoStyleClamPizza();
}
else if (type.equals("pepperoni"))
{pizza = new ChicagoStylePepperoniPizza();
}}
else
```

```
{System.out.println("Error: invalid type of pizza");
return null;
}
pizza.prepare();
pizza.bake();
pizza.cut();
pizza.box();
return pizza;
}}
class NYPizzaStore extends PizzaStore
{Pizza createPizza(String item)
{if (item.equals("cheese"))
{return new NYStyleCheesePizza();
}
else if (item.equals("veggie"))
{return new NYStyleVeggiePizza();
}
else if (item.equals("clam"))
{return new NYStyleClamPizza();
}
else if (item.equals("pepperoni"))
{return new NYStylePepperoniPizza();
}
else return null;
}}
class NYStyleCheesePizza extends Pizza
{public NYStyleCheesePizza()
{
name = "NY Style Sauce and Cheese Pizza";
dough = "Thin Crust Dough";
sauce = "Marinara Sauce";
toppings.add("Grated Reggiano Cheese");
}}
```

```
class NYStyleClamPizza extends Pizza
{public NYStyleClamPizza()
{
name = "NY Style Clam Pizza";
dough = "Thin Crust Dough";
sauce = "Marinara Sauce";
toppings.add("Grated Reggiano Cheese");
toppings.add("Fresh Clams from Long Island Sound");
}}
class NYStylePepperoniPizza extends Pizza
{public NYStylePepperoniPizza()
{
name = "NY Style Pepperoni Pizza";
dough = "Thin Crust Dough";
sauce = "Marinara Sauce";
toppings.add("Grated Reggiano Cheese");
toppings.add("Sliced Pepperoni");
toppings.add("Garlic");
toppings.add("Onion");
toppings.add("Mushrooms");
toppings.add("Red Pepper");
}}
class NYStyleVeggiePizza extends Pizza
{public NYStyleVeggiePizza()
{
name = "NY Style Veggie Pizza";
dough = "Thin Crust Dough";
sauce = "Marinara Sauce";
toppings.add("Grated Reggiano Cheese");
toppings.add("Garlic");
toppings.add("Onion");
toppings.add("Mushrooms");
toppings.add("Red Pepper");
```

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

```
for (int i = 0; i < toppings.size(); i++)
{display.append((String )toppings.get(i) + "\n");
}
return display.toString();
}}
abstract class PizzaStore
{abstract Pizza createPizza(String item);
public Pizza orderPizza(String type)
{Pizza pizza = createPizza(type);
System.out.println("--- Making a " + pizza.getName() + " ---");
pizza.prepare();
pizza.bake();
pizza.cut();
pizza.box();
return pizza;
}}
public class Main
{public static void main(String[] args)
{
PizzaStore nyStore = new NYPizzaStore();
PizzaStore chicagoStore = new ChicagoPizzaStore();
Pizza pizza = nyStore.orderPizza("cheese");
System.out.println("Ethan ordered a " + pizza.getName() + "\n");
pizza = chicagoStore.orderPizza("cheese");
System.out.println("Joel ordered a " + pizza.getName() + "\n");
pizza = nyStore.orderPizza("clam");
System.out.println("Ethan ordered a " + pizza.getName() + "\n");
pizza = chicagoStore.orderPizza("clam");
System.out.println("Joel ordered a " + pizza.getName() + "\n");
pizza = nyStore.orderPizza("pepperoni");
System.out.println("Ethan ordered a " + pizza.getName() + "\n");
pizza = chicagoStore.orderPizza("pepperoni");
System.out.println("Joel ordered a " + pizza.getName() + "\n");
```

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

2) Write a python program to implement Linear SVM.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn import svm
data=pd.read_csv('letterdata.csv')
data.head()
def getaccuracy(testset,prediction):
    correct=0
    for x in range(len(testset)):
        if(testset[x]==prediction[x]):
            correct=correct+1
    return (correct/float(len(testset)))*100.0
data.isnull().values.any()
X,Y=np.array(data)[:,1:16],np.array(data.letter)[:]
X_train=X[:16000,:]
X_test=X[16001:,:]
Y_train=Y[:16000]
Y_test=Y[16001:]
clf=svm.SVC(gamma=0.025,C=3)
clf.fit(X_train,Y_train)
Y_predict=clf.predict(X_test)
getaccuracy(Y_test,Y_predict)
y_g=(np.column_stack([Y_test,Y_predict]))
```

```
#column stack used for matching supose x=1,2,3,4,5 and y= 10 20 30 40 match x axis on 1 check on y axis on 10 ...
print(y_g)
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2] # we only take the first two features. We could
 # avoid this ugly slicing by using a two-dim dataset
y = iris.target
# we create an instance of SVM and fit out data. We do not scale our
# data since we want to plot the support vectors
C = 1.0 # SVM regularization parameter
svc = svm.SVC(kernel='linear', C=1,gamma=10).fit(X, y)
# create a mesh to plot in
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
h = (x_max / x_min)/100
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
 np.arange(y_min, y_max, h))
plt.subplot(1, 1, 1)
Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('SVC with linear kernel')
plt.show()
svc = svm.SVC(kernel='rbf', C=1,gamma=10).fit(X, y)
plt.subplot(1, 1, 1)
```

```
Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
svc = svm.SVC(kernel='poly', C=1,gamma=100).fit(X, y)
plt.subplot(1, 1, 1)
Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
```

3) Implement Login System using Django.

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "Root@123",
  database: "node"
});

con.connect(function(err) {
  if (err) throw err;

  var sql = "DELETE FROM customers WHERE name = 'pooja'";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Number of records deleted: " + result.affectedRows);
  });
});
```

MSc(CS) Lab Work

Q1. 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()

Source code
Step 1: Create Observer interface and DisplayElement Interface for display common method

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

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

Step 2: Create Observable interface Subject.java

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

Step 3: Create WeatherData class to implement Subject interface.

```
import java.util.*;

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

```
public WeatherData() {
 observers = new ArrayList<>();
}

public void registerObserver(Observer o) {
 observers.add(o);
}

public void removeObserver(Observer o) {
 int i = observers.indexOf(o);
 if (i >= 0) {
  observers.remove(i);
 }
}

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

public void measurementsChanged() {
 notifyObservers();
}

public void setMeasurements(float temperature, float humidity, float pressure)
{
 this.temperature = temperature;
 this.humidity = humidity;
 this.pressure = pressure;
 measurementsChanged();
}

public float getTemperature() {
 return temperature;
```

```
 }

 public float getHumidity() {
  return humidity;
 }

 public float getPressure() {
  return pressure;
 }
}
```

Step 4: Create Observer ForecastDisplay class.

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

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

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

  display();
 }

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

```
 }
}
```
Step 5: Create second Observer HeatIndexDisplay class.
```
public class HeatIndexDisplay implements Observer, DisplayElement {
 float heatIndex = 0.0f;
 private WeatherData weatherData;

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

 public void update(float t, float rh, float pressure) {
  heatIndex = computeHeatIndex(t, rh);
  display();
 }

 private float computeHeatIndex(float t, float rh) {
  float index = (float)((16.923 + (0.185212 * t) + (5.37941 * rh) - (0.100254 * t
* rh)
   + (0.00941695 * (t * t)) + (0.00728898 * (rh * rh))
   + (0.000345372 * (t * t * rh)) - (0.000814971 * (t * rh * rh)) +
   (0.0000102102 * (t * t * rh * rh)) - (0.000038646 * (t * t * t)) +
(0.0000291583 *
   (rh * rh * rh)) + (0.00000142721 * (t * t * t * rh)) +
   (0.000000197483 * (t * rh * rh * rh)) - (0.0000000218429 * (t * t * t * rh *
rh)) +
   0.000000000843296 * (t * t * rh * rh * rh)) -
   (0.0000000000481975 * (t * t * t * rh * rh * rh)));
  return index;
 }

 public void display() {
  System.out.println("Heat index is " + heatIndex);
 }
}
```
Step 6: Create third Observer StatisticsDisplay class.

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

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

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

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

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

  display();
 }

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

Step 7: Create fourth Observer CurrentConditionsDisplay class.

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

```
 private Subject weatherData;

 public CurrentConditionsDisplay(Subject weatherData) {
  this.weatherData = weatherData;
  weatherData.registerObserver(this);
 }

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

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

Step 8: Create WeatherStation class to test observer design pattern.

```
public class WeatherStation {

 public static void main(String[] args) {
  WeatherData weatherData = new WeatherData();

  CurrentConditionsDisplay currentDisplay =
   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);
 }
}
```

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

lower case letters.

```
import java.io.*;

public class LowerCaseInputStream extends FilterInputStream
{


   public LowerCaseInputStream(InputStream is)
   {
      super(is);
   }
   public int read() throws IOException
   {
      int charIn = super.read();  //grab a character as an int
      if (charIn != -1)
        return Character.toLowerCase((char)charIn);
      else
        return charIn;
   }
   public int read(byte[] b, int offset, int len) throws IOException
   {
      int noBytes = super.read(b,offset,len);
      for (int i = offset; i < offset + noBytes; i++)
      {
         b[i] = (byte)Character.toLowerCase((char)b[i]);
      }
      return noBytes;
   }
}

import java.io.*;

public class LowerCaseInputStreamRunner
{
public static void main(String[] args) throws IOException
```

```
{
        int charIn;
        try {
                        FileInputStream fis = new
FileInputStream("c:/aaa/upper.dat");
                        BufferedInputStream bis = new BufferedInputStream(fis);
                        InputStream is = new LowerCaseInputStream(bis);
                        while((charIn = is.read()) >= 0)
                        System.out.print((char)charIn);
                }catch(IOException excp)
                {
                        System.err.println("An IOException occurred");
                        System.out.println(excp);
                }
}
}
```

Q3). Write a Java Program to implement Factory method for Pizza Store with createPizza(),
orederPizza(), prepare(), Bake(), cut(), box(). Use this to create variety of pizza's like
NyStyleCheesePizza, ChicagoStyleCheesePizza etc.

Step 1: Create an abstract class called Pizza which abstract pizza-related data:

```
import java.util.ArrayList;

abstract public class Pizza {
   String name;
   String dough;
   String sauce;
   ArrayList toppings = new ArrayList();

   public String getName() {
      return name;
   }
```

```
    public void prepare() {
        System.out.println("Preparing " + name);
    }

    public void bake() {
        System.out.println("Baking " + name);
    }

    public void cut() {
        System.out.println("Cutting " + name);
    }

    public void box() {
        System.out.println("Boxing " + name);
    }

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

Step 2: Create Concrete Pizza classes which extends abstract Pizza class - CheesePizza, ClamPizza, VeggiePizza, and PepperoniPizza class:

```
public class CheesePizza extends Pizza {
    public CheesePizza() {
        name = "Cheese Pizza";
        dough = "Regular Crust";
        sauce = "Marinara Pizza Sauce";
        toppings.add("Fresh Mozzarella");
```

```
        toppings.add("Parmesan");
    }
}

public class ClamPizza extends Pizza {
    public ClamPizza() {
        name = "Clam Pizza";
        dough = "Thin crust";
        sauce = "White garlic sauce";
        toppings.add("Clams");
        toppings.add("Grated parmesan cheese");
    }
}

public class VeggiePizza extends Pizza {
    public VeggiePizza() {
        name = "Veggie Pizza";
        dough = "Crust";
        sauce = "Marinara sauce";
        toppings.add("Shredded mozzarella");
        toppings.add("Grated parmesan");
        toppings.add("Diced onion");
        toppings.add("Sliced mushrooms");
        toppings.add("Sliced red pepper");
        toppings.add("Sliced black olives");
    }
}

public class PepperoniPizza extends Pizza {
    public PepperoniPizza() {
        name = "Pepperoni Pizza";
        dough = "Crust";
        sauce = "Marinara sauce";
        toppings.add("Sliced Pepperoni");
        toppings.add("Sliced Onion");
        toppings.add("Grated parmesan cheese");
    }
```

```
}
```
Step 3: Create a SimplePizzaFactory class which produces pizza object based on the type of the pizza - SimplePizzaFactory java class.

```
public class SimplePizzaFactory {

    public Pizza createPizza(String type) {
        Pizza pizza = null;

        if (type.equals("cheese")) {
            pizza = new CheesePizza();
        } else if (type.equals("pepperoni")) {
            pizza = new PepperoniPizza();
        } else if (type.equals("clam")) {
            pizza = new ClamPizza();
        } else if (type.equals("veggie")) {
            pizza = new VeggiePizza();
        }
        return pizza;
    }
}
```
Step 4: Let's create PizzaStore to order the Pizza:

```
package com.ramesh.gof.factory.pizzas;

public class PizzaStore {
    SimplePizzaFactory factory;

    public PizzaStore(SimplePizzaFactory factory) {
        this.factory = factory;
    }

    public Pizza orderPizza(String type) {
        Pizza pizza;

        pizza = factory.createPizza(type);
```

```
        pizza.prepare();
        pizza.bake();
        pizza.cut();
        pizza.box();

        return pizza;
    }
}
```

Step 5: Let's test the Factory Pattern with below PizzaTestDrive:

```
public class PizzaTestDrive {

    public static void main(String[] args) {
        SimplePizzaFactory factory = new SimplePizzaFactory();
        PizzaStore store = new PizzaStore(factory);

        Pizza pizza = store.orderPizza("cheese");
        System.out.println("We ordered a " + pizza.getName() + "\n");

        pizza = store.orderPizza("veggie");
        System.out.println("We ordered a " + pizza.getName() + "\n");
    }
}
```

Q4).Write a Java Program to implement Singleton pattern for multithreading.

Q5).

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

// Light class and its corresponding command
// classes
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;

    // The constructor is passed the light it
    // is going to control.
    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();
    }
}

// Stereo and its command classes
```

```
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);
     }
}

// A Simple remote control with one button
class SimpleRemoteControl
{
    Command slot;  // only one button

    public SimpleRemoteControl()
    {
    }

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

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

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

        // we can change command dynamically
        remote.setCommand(new
                 LightOnCommand(light));
        remote.buttonWasPressed();
        remote.setCommand(new
        StereoOnWithCDCommand(stereo));
        remote.buttonWasPressed();
        remote.setCommand(new
                StereoOffCommand(stereo));
        remote.buttonWasPressed();
     }
  }
```

Q6).Write a Java Program to implement undo command to test Ceiling fan.

Q7). Write a Java Program to implement Adapter pattern for Enumeration iterator.

```
  interface Bird
{
    // birds implement Bird interface that allows
```

```
    // them to fly and make sounds adaptee interface
    public void fly();
    public void makeSound();
}

class Sparrow implements Bird
{
    // a concrete implementation of bird
    public void fly()
    {
        System.out.println("Flying");
    }
    public void makeSound()
    {
        System.out.println("Chirp Chirp");
    }
}

interface ToyDuck
{
    // target interface
    // toyducks dont fly they just make
    // squeaking sound
    public void squeak();
}

class PlasticToyDuck implements ToyDuck
{
    public void squeak()
    {
        System.out.println("Squeak");
    }
}

class BirdAdapter implements ToyDuck
{
    // You need to implement the interface your
```

```
    // client expects to use.
    Bird bird;
    public BirdAdapter(Bird bird)
    {
        // we need reference to the object we
        // are adapting
        this.bird = bird;
    }

    public void squeak()
    {
        // translate the methods appropriately
        bird.makeSound();
    }
}

class Main
{
    public static void main(String args[])
    {
        Sparrow sparrow = new Sparrow();
        ToyDuck toyDuck = new PlasticToyDuck();

        // Wrap a bird in a birdAdapter so that it
        // behaves like toy duck
        ToyDuck birdAdapter = new BirdAdapter(sparrow);

        System.out.println("Sparrow...");
        sparrow.fly();
        sparrow.makeSound();

        System.out.println("ToyDuck...");
        toyDuck.squeak();

        // toy duck behaving like a bird
        System.out.println("BirdAdapter...");
        birdAdapter.squeak();
```

```
    }
}
```

Q8). Write a Java Program to implement Iterator Pattern for Designing Menu like Breakfast,
Lunch or Dinner Menu.

```
import java.util.Iterator;

public interface Menu {
	public Iterator<?> createIterator();

	String name;
	public String getName() {
		return name;
	}
}
public class MenuItem {
	String name;
	String description;
	boolean vegetarian;
	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 double getPrice() {
		return price;
	}

	public boolean isVegetarian() {
		return vegetarian;
	}
}
public class PancakeHouseMenu implements Menu {
	ArrayList<MenuItem> menuItems;

	public PancakeHouseMenu() {
		name = "BREAKFAST";
		menuItems = new ArrayList<MenuItem>();

		addItem("K&B's Pancake Breakfast",
			"Pancakes with scrambled eggs, and toast",
			true,
			2.99);

		addItem("Regular Pancake Breakfast",
			"Pancakes with fried eggs, sausage",
			false,
			2.99);

		addItem("Blueberry Pancakes",
			"Pancakes made with fresh blueberries, and blueberry
syrup",
			true,
			3.49);
```

```
                addItem("Waffles",
                        "Waffles, with your choice of blueberries or strawberries",
                        true,
                        3.59);
        }

        public void addItem(String name, String description,
                        boolean vegetarian, double price)
        {
                MenuItem menuItem = new MenuItem(name, description,
vegetarian, price);
                menuItems.add(menuItem);
        }

        public ArrayList<MenuItem> getMenuItems() {
                return menuItems;
        }

        public Iterator<MenuItem> createIterator() {
                return menuItems.iterator();
        }

        // other menu methods here
}
import java.util.Iterator;

public class DinerMenu implements Menu {
        static final int MAX_ITEMS = 6;
        int numberOfItems = 0;
        MenuItem[] menuItems;

        public DinerMenu() {
                name = "LUNCH";
                menuItems = new MenuItem[MAX_ITEMS];

                addItem("Vegetarian BLT",
```

```
                "(Fakin') Bacon with lettuce & tomato on whole wheat",
true, 2.99);
        addItem("BLT",
                "Bacon with lettuce & tomato on whole wheat", false, 2.99);
        addItem("Soup of the day",
                "Soup of the day, with a side of potato salad", false, 3.29);
        addItem("Hotdog",
                "A hot dog, with saurkraut, relish, onions, topped with
cheese",
                false, 3.05);
        addItem("Steamed Veggies and Brown Rice",
                "Steamed vegetables over brown rice", true, 3.99);
        addItem("Pasta",
                "Spaghetti with Marinara Sauce, and a slice of sourdough
bread",
                true, 3.89);
    }

    public void addItem(String name, String description,
                 boolean vegetarian, double price)
    {
        MenuItem menuItem = new MenuItem(name, description,
vegetarian, price);
        if (numberOfItems >= MAX_ITEMS) {
            System.err.println("Sorry, menu is full!  Can't add item to
menu");
        } else {
            menuItems[numberOfItems] = menuItem;
            numberOfItems = numberOfItems + 1;
        }
    }

    public MenuItem[] getMenuItems() {
        return menuItems;
    }

    public Iterator<MenuItem> createIterator() {
```

```
            return new DinerMenuIterator(menuItems);
            //return new AlternatingDinerMenuIterator(menuItems);
        }

        public

        // other menu methods here
}
import java.util.Iterator;

public class DinerMenuIterator implements Iterator<MenuItem> {
        MenuItem[] list;
        int position = 0;

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

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

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

        public void remove() {
            if (position <= 0) {
                throw new IllegalStateException
                    ("You can't remove an item until you've done at least
one next()");
```

```
            }
            if (list[position-1] != null) {
                for (int i = position-1; i < (list.length-1); i++) {
                    list[i] = list[i+1];
                }
                list[list.length-1] = null;
            }
        }
}
public class Waitress {
        ArrayList<Menu> menus;


        public Waitress(ArrayList<Menu> menus) {
            this.menus = menus;
        }

        public void printMenu() {
            Iterator<?> menuIterator = menus.iterator();

            System.out.print(MENU\n----\n);
            while(menuIterator.hasNext()) {
                Menu menu = (Menu)menuIterator.next();
                System.out.print("\n" + menu.getName() + "\n");
                printMenu(menu.createIterator());
            }
        }

        void printMenu(Iterator<?> iterator) {
            while (iterator.hasNext()) {
                MenuItem menuItem = (MenuItem)iterator.next();
                System.out.print(menuItem.getName() + ", ");
                System.out.print(menuItem.getPrice() + " -- ");
                System.out.println(menuItem.getDescription());
            }
        }
}
```

```
public class MenuTestDrive {
	public static void main(String args[]) {
		PancakeHouseMenu pancakeHouseMenu = new
PancakeHouseMenu();
		DinerMenu dinerMenu = new DinerMenu();
		ArrayList<Menu> menus = new ArrayList<Menu>();
		menus.add(pancakeHouseMenu);
		menus.add(dinerMenu);
		Waitress waitress = new Waitress(menus);
		waitress.printMenu();

	}
}
```

Q9).Write a java program to implement Adapter pattern to design Heart Model to Beat
Model.

Q10).Write a java program to implement Adapter pattern to design Heart Model to Beat
Model.