Exercise 1 - Classes, Aggregation, Inheritance, Abstract Classes

Define in Java the classes representing the attributes (Attribute, ContinuousAttribute, DiscreteAttribute classes) of a transaction (or tuple) and the collection of transactions (Data class)

The visibility of attributes, methods and classes must be decided by the Student from time to time

■ *Define the abstract* ***Attribute*** *class (in the default package) that models the attribute entity.*

abstract class *Attribute {...}*

Members Attributes

String name; // *symbolic name of the attribute*

int index; // *numerical identifier of the attribute*

Members Methods

Attribute(String name, **int** index)

Input: attribute name and numeric attribute identifier (first, second ... attribute of the tuple)

Output: //

Behaviour: initialise member values name, index

String getName()

Input:

Output : attribute name

Behaviour: returns name;

int getIndex()

Input:

Output : numeric attribute identifier

Behaviour: returns index;

public **String** toString()

Input:

Output : overrides method inherited from the superclass and restores the string representing the state of the object

Behaviour: returns name;

■ *Define the concrete* ***Continuous Attribute*** *class that extends the* ***Attribute*** *class and models a continuous (numeric) attribute. This class includes methods for "normalising" the attribute's domain in the interval [0,1] in order to make attributes with different domains comparable.*

Members Attributes

double max;

double min ;// *represent the extremes of the value range (domain) that the attribute can actually take.*

Members Methods

ContinuousAttribute(String name, int index, double min, double max)

Input: name, numeric identifier, minimum and maximum value of attribute Output:

//

Behaviour: Invokes the constructor of the parent class and initialises members added by extension

double getScaledValue(**double** v)

Input: value of attribute to be normalised

Output : normalised value

Behaviour: Calculates and returns the normalised value of the parameter passed as input. The normalisation has the interval [0,1] as its codomain. The normalisation of v is then calculated as follows:

v'=(v-min)/(max-min)

Define the concrete class ***DiscreteAttribute*** *that extends the* ***Attribute*** *class and represents a discrete (categorical) attribute*

Members Attributes

String values[ ];// *array of String objects, one for each discrete domain value. The domain values are stored in values following a lexicographic order.*

Members Methods

DiscreteAttribute(String name, **int** index, String values[ ])

Input: attribute name, numeric attribute identifier and string array representing the attribute domain

Output : //

Behaviour: Invokes the constructor of the parent class and

initialises the values member with the input parameter.

int getNumberOfDistinctValues()

Input: //

Output : number of discrete values in the attribute domain

Behaviour: Returns the size of values

String getValue(**int** i)

Input: position of a value in values

Output : discrete value at position 'i' of values

Behaviour: Returns values[i]

- *Defining the concrete class* ***Data*** *to model the set of transactions (or tuples)*

Members Attributes

Object data [ ] [ ]; // *an nXm array of type Object where each row models a transaction*

int numberOfExamples; // *cardinality of transaction set (number of rows in date)*

Attribute attributeSet [ ]; // *a vector of the attributes in each tuple (data table schema)*

Members Methods

Data()

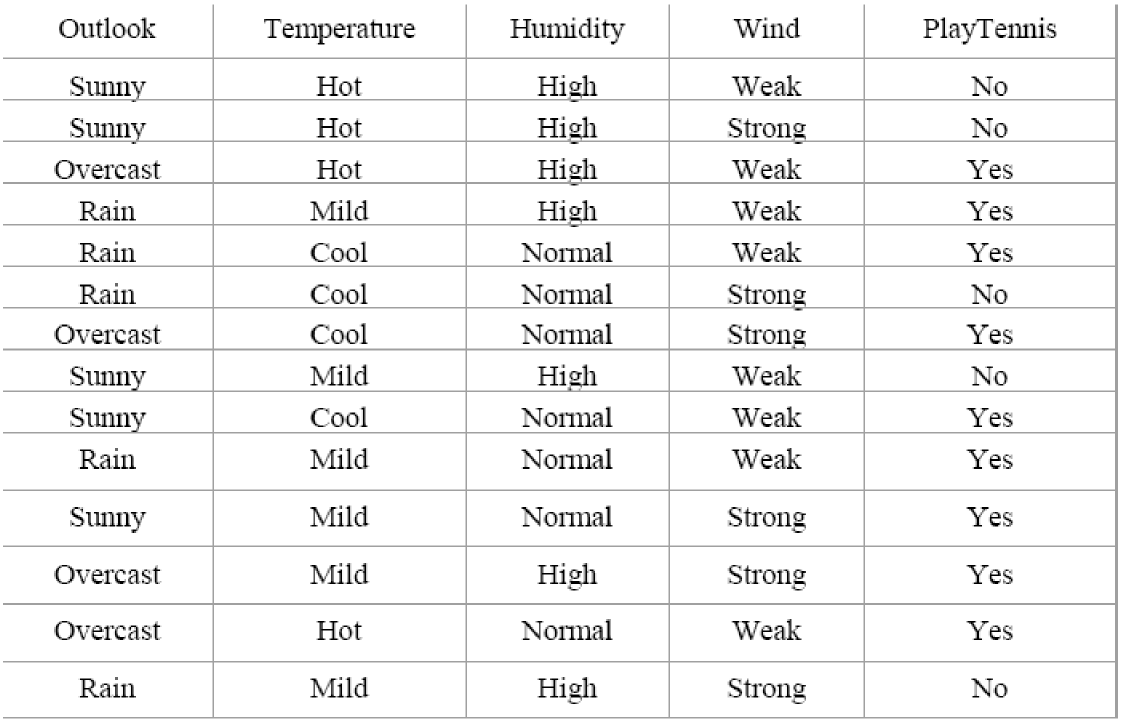
Input:

Output :

Behaviour: Initialise the given [ ][ ] array with example transactions (at this time, 14 examples and 5 attributes as shown in the table below);

Initialise attributeSet by creating five objects of type DiscreteAttribute, one for each attribute (in the table below). Take care to correctly model the name, index and domain of each attribute.

Initialise numberOfExamples



Outlook	Temperature	Humidity	Wind	PlayTennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

int getNumberOfExamples()

Input://

Output: cardinality of transaction set Behaviour:

returns numberOfExamples

int getNumberOfAttributes()

Input://

Output: cardinality of attribute set Behaviour: returns

attributeSet size

Attribute[] getAttributeSchema()

Input: //

Output: returns data schema Behaviour:

returnsattri

buteSet

Object getAttributeValue(**int** exampleIndex, **int** attributeIndex)

Input: row index , column index with reference to the matrix stored in date

Output: value taken on date by the attribute at attributeIndex position, in the line at exampleIndex position

Behaviour: returns data[exampleIndex][attributeIndex].

public *String toString()*

Input: //

Output: string modelling the state of the object

Behaviour: Creates a string in which it stores the table schema (see attributeSet) and the transactions stored in date, enumerated accordingly. Returns this string

- *Define a main method in* ***Data*** *that allows testing of the implemented classes, in particular allowing the printing of the set of transactions.*

Example output:

```
Outlook,Temperature,Humidity,Wind Playtennis

1:sunny,hot,high,weak,no,

2:sunny,hot,high,strong,no,

3:overcast,hot,high,weak,yes,

4:rain,mild,high,weak,yes,

5:rain,cool,normal,weak,yes,

6:rain,cool,normal,strong,no,

7:overcast,cool,normal,strong,yes,

8:sunny,mild,high,weak,no,

9:sunny,cool,normal,weak,yes,

10:rain,mild,normal,weak,yes,

11:sunny,mild,normal,strong,yes,

12:overcast,mild,high,strong,yes,

13:overcast,hot,normal,weak,yes,

14:rain,mild,high,strong,no,
```