<Drake: Route, Place>

<Me: Bus, BusType, BusesDatabase>

<?: DijkstraStation>

**Pseudocode for classes LongDistStationsDatabase, BusStation, GasStation, BusesDatabase, Bus, and BusType**

**Drake, Jeffrey, and Joseph**

Class LongDistStationsDatabase:

Let constant *DATA\_FILE* refer to file LongDistStationsText.txt

Declare a list *lDStations* that will store BusStation objects (some of which will be GasStation objects, since GasStation is a child class of BusStation). This list will be initialized in the constructor.

Void method createConnection ( integer *i0*, integer *i1* ):

Let variables *bs0* and *bs1* store the *i0*th and *i1*th stations in *lDStations*, respectively

Using the BusStation.connect method, connect *bs0* to *bs1* and *bs1* to *bs0*

Constructor:

Let string *line* store the first line of *DATA\_FILE*. (Eventually it will iterate through all the lines in the file.)

Initialize *lDStations* with a new, empty list of BusStations

Until *line* reaches a blank line:

Let 1-dimensional, 4-element string array *splitLine* store the 4 substrings of *line* representing its 4 columns, which are separated by 3 substrings “, ”

Let real numbers *lat* and *long* store the numbers parsed from the 0th and 1st elements of *splitLine*

Let boolean *isGasStation* be false if the 2nd element of *splitLine* is “City” and true if the 2nd element of *splitLine* is “Gas”. Throw an exception if the 2nd element of *splitLine* is neither “City” nor “Gas”.

Let string *name* store the 3rd element of *splitLine*

Create a BusStation *lDStation* using either the BusStation constructor or the GasStation constructor – depending on *isGasStation*’s value – and passing *lat*, *long*, and *name* to that constructor

Add *lDStation* to *lDStations*

Overwrite *line* with the next line in the file

*line* currently stores a blank line. Overwrite *line* with the next line in the file.

Until *line* reaches another blank line or the end of the file:

Let 1-dimensional, 2-element string array *splitLine* store the 2 substrings of *line* representing its 2 columns, which are separated by a space

Call this class’s createConnection method, parsing the 2 elements of *splitLine* as integers for the 2 arguments

Overwrite *line* with the next line in the file

Function toArray() returning a 1-dimensional array of BusStations:

Return *lDStations*.toArray()

Void method update():

Overwrite *DATA\_FILE* entirely as follows:

For each BusStation in *lDStations*:

Write a line containing its latitude; its longitude; either of the strings “Gas” or “City”, depending on whether it is a GasStation or a normal BusStation, respectively; and its name. Separate each of these 4 items with a comma and a space – the string “, ”.

Write a black line to indicate the separation between the two parts of the file

For ( integer *i* starting at 0 and incrementing through every index valid for *lDStations* (ending at one less than the size of *lDStatiions*) ):

For ( integer *j* starting one greater than *i* and incrementing through every index greater than i valid for *lDStations* ):

If the *i*th and *j*th stations in *lDStations* are connected, then write a line containing *i* and *j*, separated by a space

Void method addLDStation ( BusStation *lDStation* ):

If *lDStation*’s *name* contains a comma, then throw an exception

Otherwise, add *lDStation* to *lDStations*

Void method addNewLDStation taking four parameters:

Parameters:

Real numbers *latitude* and *longitude*

Boolean *isGasStation*

String *name*

If *name* contains a comma, then throw an exception

Otherwise:

Create a BusStation *lDStation* using either the BusStation constructor or the GasStation constructor – depending on *isGasStation*’s value – and passing the other three arguments to that constructor

Add *lDStation* to *lDStations*

Void method delete ( integer *index* ):

Let string *name* store the name of the *index*th station in *lDStations*

Delete the *index*th station in *lDStations*

For each remaining station *bs* in *LDStations*:

For each station in *bs*.connectedStations:

If that station’s name is identical to the variable *name* created at the beginning of this method, then remove that station from *bs*.connectedStations.

Class BusStation, a child class of the Place class:

Declare a string *name* that will store the name of the bus station, and declare a list *connectedStations* that will store other BusStation objects. Both will be initialized in the constructor.

Function getName():

The accessor function for *name*

Constructor ( real number *latitude*, real number *longitude*, string *name* ):

Call Place’s constructor with arguments *latitude* and *longitude*

Set this object’s *name* to the argument *name* of this method

Initialize *connectedStations* with a new, empty list of BusStations

Void method connect ( BusStation *station* ):

Add *station* to *connectedStations*

Function toString() overriding Place’s toString() function:

Return a string containing this BusStation’s *name*, *latitude*, and *longitude*, in that order, with *latitude* and *longitude* in parentheses and preceded by the word at, as in this example:

Abc Xyz Station at (0, 0)

Function hoursTo( Place *place*, Bus *bus* ) returning a real number:

Use Place’s milesTo function to find the distance (in miles) between this BusStation and *place.* Divide that distance by the cruising speed of *bus* to find the time required (in hours) to travel that distance in the *bus*. Return that time.

Class GasStation, a child class of the BusStation class:

This class has no fields besides those inherited from BusStation

Constructor ( real number *latitude*, real number *longitude*, string *name* ):

Call BusStation’s constructor with these arguments

Function toString() overriding BusStation’s toString() function:

Return BusStation’s toString() value preceded by string “GAS STATION: ”, as in this example:

GAS STATION: Abc Xyz Station at (0, 0)

Class BusesDatabase:

Let constant *DATA\_FILE* refer to file BusesText.txt

Declare a list *buses* that will store Bus objects. This list will be initialized in the constructor.

Constructor:

Let string *line* store the first line of *DATA\_FILE*. (Eventually it will iterate through all the lines in the file.)

Initialize *buses* with a new, empty list of Buses

Until *line* reaches a blank line or the end of the file:

Let 1-dimensional, 4-element string array *splitLine* store the 4 substrings of *line* representing its 4 columns, which are separated by 3 substrings “, ”

Let string *makeAndModel* store the 0th element of *splitLine*

Let BusType *type* store BusType.*city* if the 1st element of *splitLine* is “City” and store BusType.*longDistance* if the 1st element of *splitLine* is “LD”. Throw an exception if the 1st element of *splitLine* is neither “City” nor “LD”.

Let real numbers *tankSize*, *cruisingConsumption*, and *cruisingSpeed* store the real numbers parsed from the 2nd, 3rd, and 4th elements of *splitLine*

Create a Bus *bus* by passing *makeAndModel*, *type*, *tankSize*, *cruisingConsumption*, and *cruisingSpeed* to the Bus constructor.

Add *bus* to *buses*

Overwrite *line* with the next line in the file

Function toArray() returning a 1-dimensional array of Buses:

Return *buses*.toArray()

Void method update():

Overwrite *DATA\_FILE* entirely as follows:

For each Bus in *buses*:

Write a line containing its make and model, its type (converted to a string with the Bus.toString() function), its tank size, its cruising consumption, and its cruising speed. Separate each of these 5 items with a comma and a space – the string “, ”.

Void method addBus ( Bus *bus* ):

If *bus*’s *makeAndModel* contains a comma, then throw an exception.

Otherwise, add *bus* to *buses*

Void method addNewBus taking 5 parameters:

Parameters:

String *mm*

BusType *bt*

Real numbers *ts, cc, cs*

If *mm* contains a comma, then throw an exception

Otherwise, create a new Bus by passing the 5 arguments to the Bus constructor, and add that Bus to *buses*.

Void method delete ( integer *index* ):

Delete the *index*th Bus in *buses*

Class Bus:

Declare string *makeAndModel*; BusType *type*; and real numbers *tankSize* (in gallons), *cruisingConsumption* (in gallons per hour), and *cruisingSpeed* (in miles per hour). All five will be initialized in the constructor.

This class has 5 accessor functions, one for each of the above fields: getMakeAndModel(), getType(), getTankSize(), getCruisingConsumption, and getCruisingSpeed()

Constructor taking 5 parameters:

Parameters:

String *mm*

BusType *b*

Real numbers *ts, cc, cs*

Initialize *makeAndModel* to *mm*, *type* to *t*, *tankSize* to *ts*, *cruisingConsumption* to *cc*, and *cruisingSpeed* to *cs*.

Let constant *DEFAULT\_BUS* store a Bus with *makeAndModel* “The Magic School Bus”, *type* BusType.*city*, *tankSize* 60 gallons, *cruisingConsumption* 4.5 gallons per hour, and *cruisingSpeed* 30 mph

Function maxTime() returning a real number:

Divide *tankSize* by *cruisingConsumption* to find the number of hours this Bus can run before needing to refuel. Return that time.

Function maxRange() returning a real number:

Call the maxTime() method and multiply its result by *cruisingSpeed* to find the distance in miles this bus can go before needing to refuel. Return that distance .

Enumerated class BusType:

This enumeration has two elements: *city* and *longDistance*

Function toString():

If this object is *city,* then return “City”. If this object is rather *longDistance*, then return “LD”.