CPS 2004
OOP Assignment

Name & Surname: Deborah Vella

Table of contents

Java

Assumptions

Before coding the Java Program, I made some assumptions and programmed according to them.

First assumption is that the text file provided, must have each command or word separated by a space as the delimiter for the file reader is the space.

Second assumption is that the commands are always given in the same order i.e:

- BeginRestaurant
- Item
- EndRestaurant
- BeginOrderList
- BeginOrder
- OrderItem
- EndOrder
- EndOrderList

Third assumption is that, each restaurant should have all its details inside only one BeginRestaurant and EndRestaurant commands. Which means that if I added a Restaurant called ZeroTre, in the next restaurants to be read there should not be another restaurant named ZeroTre.

Fourth assumption is about the linked list. In the assignment specification there is written that the linked list implemented in the tutorial should be used. I assumed that functions such as deleting, finding, inserting at a specific position and returning the size of list can be discarded for this assignment. Therefore, I did not implement all the functions the linked list in the tutorial offers and implemented only those that are needed.

Program Overview

For this question I created 5 different Object classes which are listed below:

- Restaurants: used to create an object for each Restaurant
- Order: used to create an object for each Order
- myLinkedList: implements the algorithm of a linked list
- ListNode: contains the information each node should have
- Item: used to create items for the Restaurants' menus and also used to create the items for each order's item list.

Most object classes make use of **encapsulation** as they contain private variables therefore can be accessed from that class only. There are public methods to modify or get these attributes. This automatically enforces **data hiding**. Furthermore, **no inheritance** was implemented as each Object defines something totally different from every other object and should not be of the same type. Also, there are **no abstract classes** in this program as each Object class needs to be instantiated at one point or another.

Two more classes (non-Object classes) were created:

- Main: reads text file and the commands contained in it, takes in all the details of the restaurants and the orders, create objects of Item, Order and Restaurants and store them in their own data structure (Linked list for orders and ArrayLists for the rest). Calls the observer whenever an arithmetic calculation should be done.
- Observer: calculates highest revenue and the total price of each order.

Class Breakdown

Object Classes

Each object class has its own getters and setters even if some of them are never used.

Restaurants		
Attribute	Variable Type	Brief Description
Name	String	Stores Restaurant name
methodOfServing	String	Store: delivery, takeaway, both
ItemList	ArrayList<Item>	Store the list of menu items
revenue	Double	Stores restaurant's revenue

Method Name	Return Type	Parameter Type	Purpose
Restaurants	\	String, String, ArrayList<Item>	Constructor
getName	String	\	Returns Restaurant name
getRevenue	Double	\	Returns revenue
getItemList	ArrayList<Item>	\	Returns the list of menu items
getMethodOfServing	String	\	Returns one of these: delivery/takeaway/both
setItemList	\	ArrayList<Item>	Modifies menu list
setName	\	String	Sets Restaurant name
setRevenue	\	Double	Updates Revenue by adding old revenue sum with the number passed.
setMethodOfServing	\	String	Sets Method of serving

Order		
Attribute	Variable Type	Brief Description
restaurantName	String	Stores Restaurant name
service	String	Store: delivery, takeaway, both
ItemList	ArrayList<Item>	Store the list of ordered items
totalOrderPrice	Double	Stores the order price

Method Name	Return Type	Parameter Type	Purpose
Order	\	String, String, ArrayList<Item>, double	Constructor
getRestaurantName	String	\	Returns Restaurant name
getTotalPrice	Double	\	Returns total price of order
getItemList	ArrayList<Item>	\	Returns the list of ordered items
getService	String	\	Returns one of these: delivery/takeaway/both
setOrderList	\	ArrayList<Item>	Modifies ordered list
setRestaurantName	\	String	Sets Restaurant name
setTotalPrice	\	Double	Updates order price
setService	\	String	Sets method of serving

Item		
Attribute	Variable Type	Brief Description
name	String	Stores Restaurant name
price	double	Stores Item price

Method Name	Return Type	Parameter Type	Purpose
Item	\	String, double	Constructor
geName	String	\	Returns Item name
getPrice	Double	\	Returns Item price
setName	\	String	Sets Item name
setPrice	\	Double	Updates Item price

myLinkedList			
Attribute	Variable Type	Brief Description	
head	ListNode	Points to the head of the list	
sizeOfList	int	Stores linked list size	
Method Name	Return Type	Parameter Type	Purpose
myLinkedList	\	\	Constructor. Sets head equal to Null and sizeOfList equal to 0
add	\	Order	Inserts a new Order inside the linked list
writeAll	\	\	Writes all orders stored in the linked list to the file called Orders.txt

ListNode			
Attribute	Variable Type	Brief Description	
next	ListNode	Points to the next node	
order	Order	Stores the order	
Method Name	Return Type	Parameter Type	Purpose
ListNode	\	Order	Constructor. Sets next equal to Null and order to the passed order

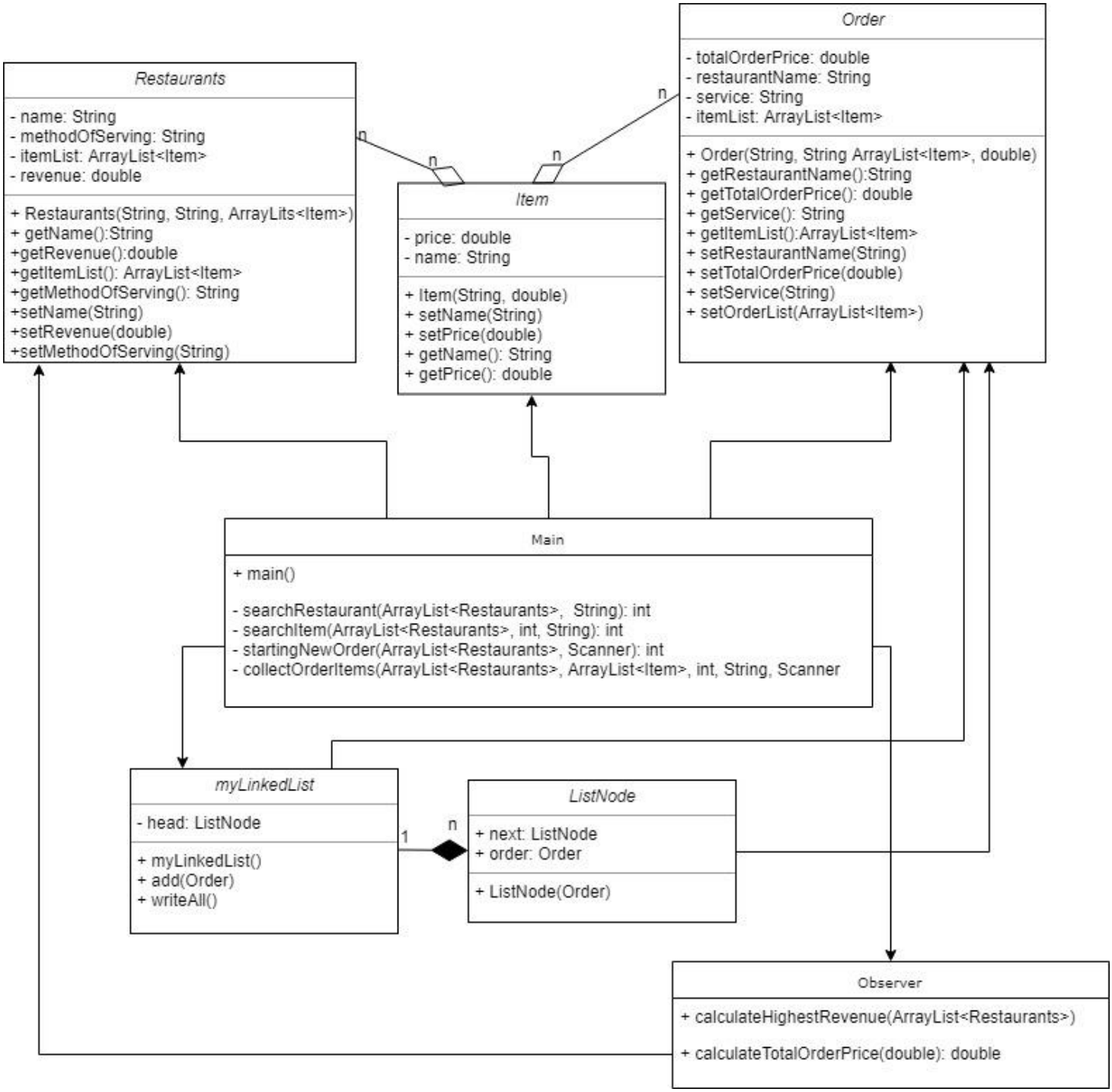
Main and Observer Classes

Main			
Method Name	Return Type	Parameter Type	Purpose
Main	\	String[]	Opens file passed as command line argument. Reads words from the file and carries out particular tasks according to specific commands.
searchRestaurant	int	ArrayList<Restaurants>, String	Searches for a specific restaurant and returns the index at which the restaurant is found
searchItem	int	ArrayList<Restaurants>, int, String	Searches for a specific Item for a specific Restaurant and returns the index at which the item is found

startingNewOrder	Int	ArrayList<Restaurants>, Scanner	Reads the restaurant name, searches for it inside the Arraylist, returns index.
collectOrderItems	\	ArrayList<Restaurants>, ArrayList<Item>, int, String, Scanner	Collects all order Items, passes each order to the linked list for storage. Recursively calls itself whenever a new Order is to be read. Finishes by calling the writeAll() function found inside myLinkedList class

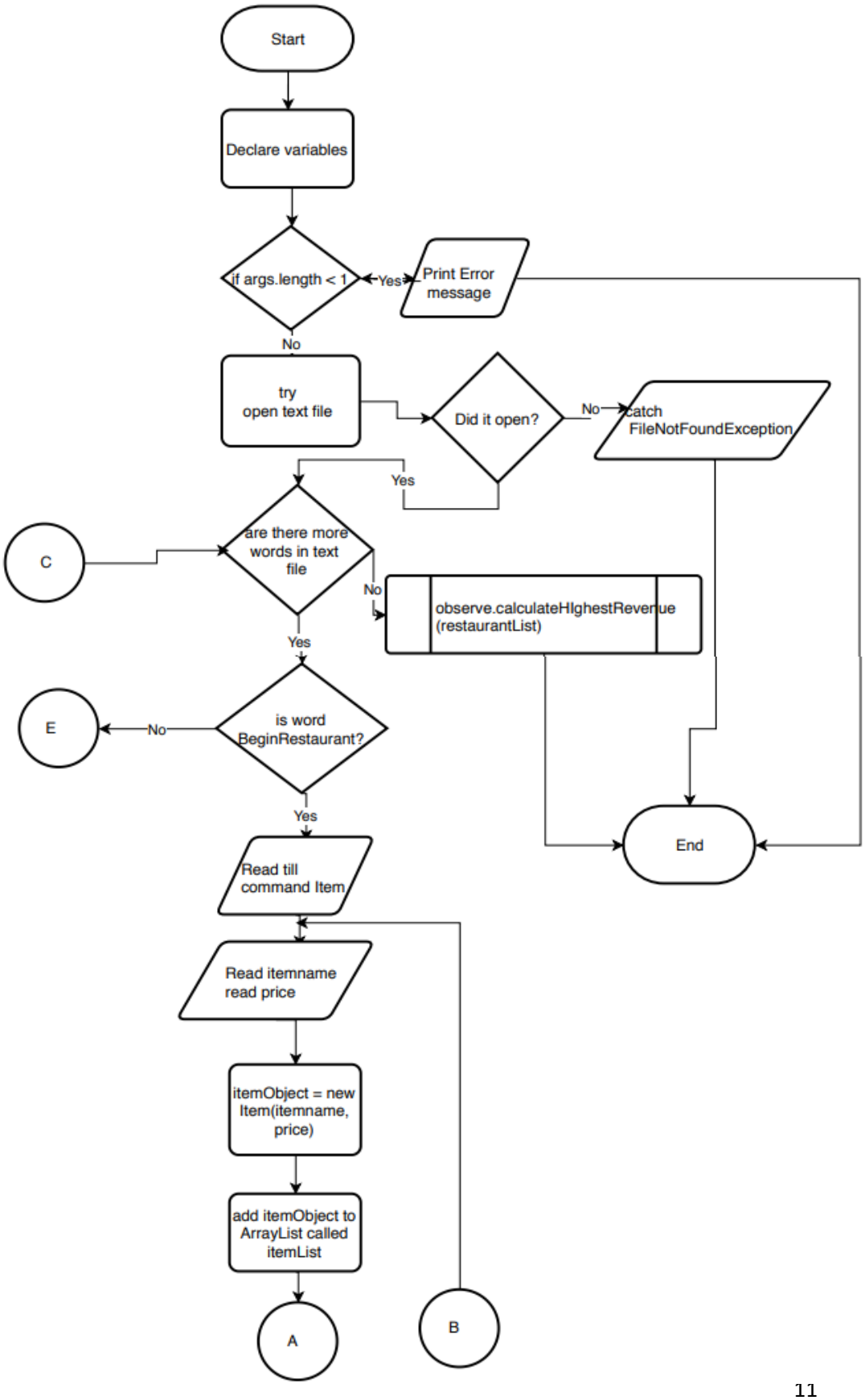
Observer			
Method Name	Return Type	Parameter Type	Purpose
calculateHighestRevenue	\	ArrayList<Restaurants>	Compares each restaurant's revenue to finally calculate which of them has the highest revenue and writes it to file.
calculateTotalOrderPrice	Double	Double	Performs the arithmetical work to get the total price of an order

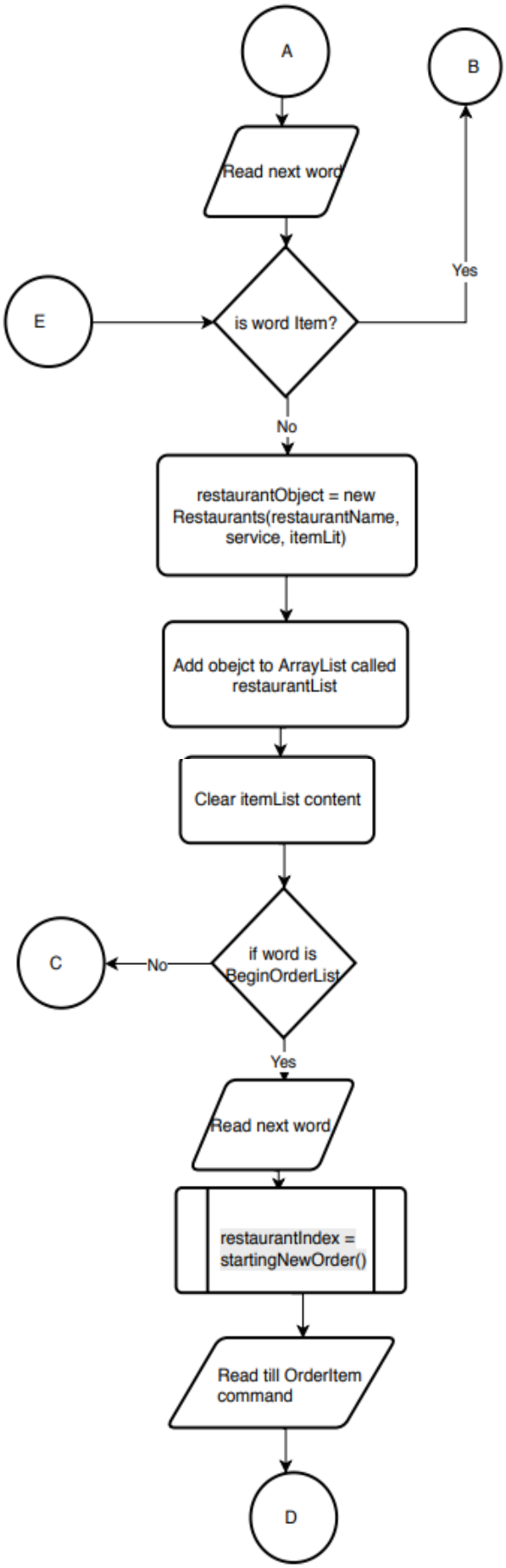
Class Diagram

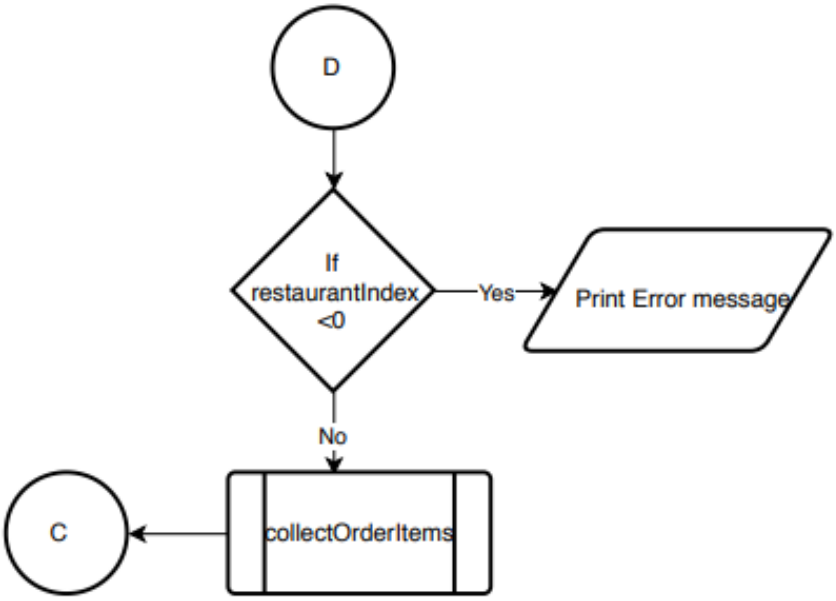


Flowchart

Main()




A
B
Read next word
Yes
E
is word Item?
No
restaurantObject = new Restaurants(restaurantName, service, itemLit)
Add obejct to ArrayList called restaurantList
Clear itemList content
C
No
if word is BeginOrderList
Yes
Read next word
restaurantIndex = startingNewOrder()
Read till OrderItem command
D


D
If
restaurantIndex
<0
Yes
Print Error message
No
C
collectOrderItems

Error Handling

Throughout the program exceptions and validations were implemented to prevent the program from crashing if an error occurs. (Only the parts which do not have a lot of code in the implementation have the Code section below. The rest are explained by pseudo code only)

Exceptions

FileNotFoundExeption

This exception was used whenever the text file was to be read. It was implemented inside the main method of the Main class as the file is to be opened there.

Pseudo Code:

```
try{
        Open file passed through the command line
        /* rest of the main method body
        */
}Catch(FileNotFoundException f){
        Print an error message
}
```

IndexOutOfBoundsException

This exception was used so that if the search for an item returns index -1, which means, it isn't found, the program will not try to fetch it to get its price. It is found inside the collectOrderItems() function in the Main class.

Pseudo Code:

```
try{
        Get the item index returned by the function searchItem().
        Get the price of that item using the item index to immediately find the item needed.
}catch(IndexOutOfBounds a){
        Print an error message notifying the user the item is not offered by that specific restaurant
}
```

Code:

```
try {
    itemIndex = searchItem(restaurantsList, restaurantIndex, itemname); //gets index where the item is stored
    price = restaurantsList.get(restaurantIndex).getItemList().get(itemIndex).getPrice();  //get the price of the item
}catch (IndexOutOfBoundsException a){
    System.out.println("The item: "+itemname+" is not offered by "+restaurantsList.get(restaurantIndex).getName());
}
```

IOException

This exception is used whenever the program needs to write to the Orders.txt file. It is found inside the Observer Class when writing the highest revenue to file and in the myLinkedList class when writing all orders to the file.

Pseudo Code:

try{

Open file to write in.

Write in the file appending the new text with the already existing text in the file.

}catch(IOException e)

Print an error message

}

Validation

- First thing that is checked is whether a file was passed as an argument or not.

Pseudo Code:

If length of args is less than one {

Print error message notifying the user that no file was passed

Exit the program

}

Code:

```
if(args.length<1){  //checks if an argument was passed in the command line
    //if there is no argument output a message and exit
    System.out.println("Error! No file was passed in the command line");
    System.exit( status: 1);
}
```

- Another validation is found when reading the order list and need to search for the Restaurant, the order is for, inside the ArrayList. If the Restaurant is not found the user needs to notified.

Pseudo Code:

If the returned restaurant index is equal to -1{

Print a message to tell the user to check the information inside the text file.

}else{

Proceed to collecting the order items.

}

Code:

```
restaurantIndex = startingNewOrder(restaurantsList, scanner);
service = scanner.next();      //has to be method to be served by
token = scanner.next();        //has to be 'OrderItem' command
if(restaurantIndex == -1){
    System.out.println("Please make sure the information in the text file is correct!");
}else {
    collectOrderItems(restaurantsList, itemList, restaurantIndex, service, scanner);
}
```

- Inside the startingNewOrder() method, there is another validation which caters for when the Restaurant cannot be found inside the ArrayList.

Pseudo Code:

Search for the Restaurant and return the index where it is found

If restaurant index is equal to -1 {

Print message to let the user know the restaurant is not found

}

Code:

```
int restaurantIndex = searchRestaurant(restaurantsList, restaurantName); //gets index where the restaurant is stored
if(restaurantIndex ==-1){
    System.out.println("Restaurant name cannot be found!");}

return restaurantIndex;
```

Proof of Running Application

The program can be run by opening the src file in terminal and then running the following commands:

source ./compile.sh

source ./run.sh

1. When the text file to be read has the correct information without any spelling mistakes. Each Restaurant has one order.

```
Orders.txt   oop-java-example.txt
---NEW ORDER---
Cikku   delivery
Ordered Items:ftira 2.0euros
ftira   2.0euros
chips   1.0euros
water-small 1.0euros
Total Order Price6.0

---NEW ORDER---
ZeroTre delivery
Ordered Items:chips 1.5euros
capricciosa 6.5euros
Total Order Price8.0

Restaurant with highest revenue is: ZeroTre
Revenue: 8.0
```

2. When the text file to be read has the correct information without any spelling mistakes. This time one of the restaurants has more than one order (Cikku).

```
Orders.txt   oop-java-example.txt
---NEW ORDER---
Cikku   delivery
Ordered Items:ftira 2.0euros
water-small 1.0euros
Total Order Price3.0

---NEW ORDER---
Cikku   delivery
Ordered Items:ftira 2.0euros
ftira   2.0euros
chips   1.0euros
water-small 1.0euros
Total Order Price6.0

---NEW ORDER---
ZeroTre delivery
Ordered Items:chips 1.5euros
capricciosa 6.5euros
Total Order Price8.0

Restaurant with highest revenue is: Cikku
Revenue: 9.0
```

3. When no file is passed as argument

```
jonvel@ubuntu:~/Assignment OOP/JavaProgram/src$ java Main
Error! No file was passed in the command line
```

4. When a restaurant name is incorrect in the Begin Order. In this case I changed the name from Cikku to Cikka in the text file. Terminal output:

```
jonvel@ubuntu:~/Assignment OOP/JavaProgram/src$ source ./CompileAndRun.sh
Restaurant name cannot be found!
Please make sure the information in the text file is correct!
```

In Orders.txt only the restaurant that could be found is written.

```
---NEW ORDER---
ZeroTre delivery
Ordered Items:chips 1.5euros
capricciosa 6.5euros
Total Order Price8.0

Restaurant with highest revenue is: ZeroTre
Revenue: 8.0
```

5. When an order item is written incorrectly in the text file. In this case I changed one of the ftira to ftajjar. Terminal output:

```
jonvel@ubuntu:~/Assignment OOP/JavaProgram/src$ source ./CompileAndRun.sh
The item: ftajjar is not offered by Cikku
```

Orders.txt :

```
---NEW ORDER---
Cikku   delivery
Ordered Items:ftajjar   0.0euros
ftira   2.0euros
chips   1.0euros
water-small 1.0euros
Total Order Price4.0

---NEW ORDER---
ZeroTre delivery
Ordered Items:chips 1.5euros
capricciosa 6.5euros
Total Order Price8.0

Restaurant with highest revenue is: ZeroTre
Revenue: 8.0
```

C++

Assumptions

First assumption is that each word in the text file should be separated by a space so that the program can move from word to word having the space as a delimiter. Secondly, the words after each command should always have the same order for example for the mammal, the litter size is always written after the length.

Program Overview

For this program I created 4 files having two of them being .cpp and the other two being .h. These are listed below:

- main.cpp: handles the reading from file and calling the right methods from other classes, creating objects and passing them as references to the correct functions.

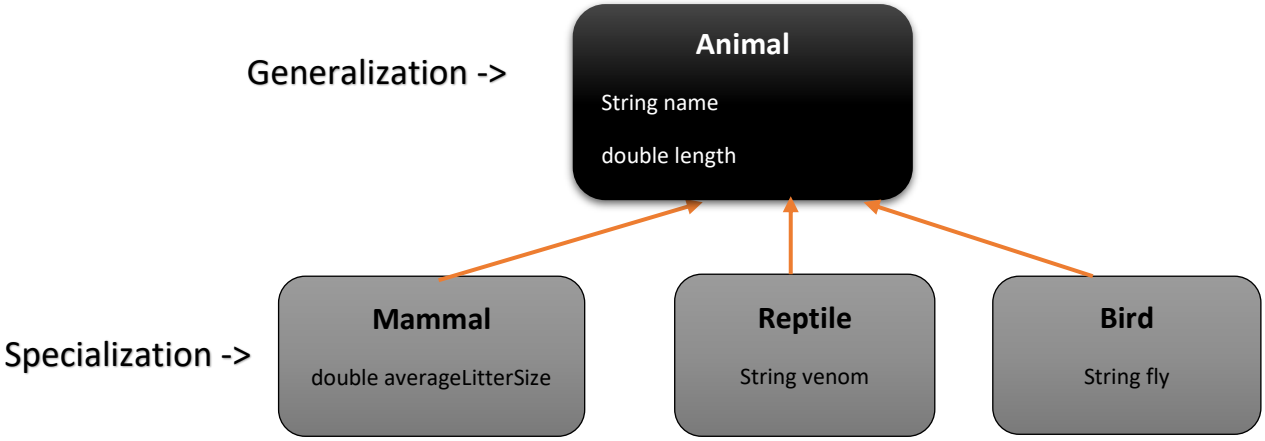
- Animal.cpp: contains the implementation of the methods declared inside the classes coded in Animal.h file.

- Animal.h: contains the declaration of the **base class** Animal and the **derived classes** Mammal, Bird and Reptile. Each class contains its own attributes and methods. The Animal class is made to be **abstract** as it contains methods which are not implemented inside of it but are implemented in the derived classes.

- BST.h: this file is made up of, both the declaration of the classes needed for the binary search tree data structure, and the implementation of the methods inside these classes. I decided to put all of this in one header file to prevent any linking problems as I made use of **templates** to accept different object types.

Inheritance

When creating the object classes of the animals I decided to make the Mammal, Bird and Reptile classes, inherit from one Base class which I called Animal. This is because all the species must have two same attributes which are the name and the length and these are properties of each and every animal that exists. Therefore, the base class contains a generalization of all animals and each derived class specializes more according to what species type it is representing. The inheritance implemented is a **single inheritance model** because the derived classes inherit from one class only.



Abstraction

The base class Animal was automatically made abstract by implementing abstract methods inside of it. The below code shows which methods are abstract.

```
//the below make Animal class an Abstract class
virtual string getVenomous()=0;
virtual string getFly()=0;
virtual double getLitterSize()=0;
```

As one may notice these are methods that should not be implemented in Animal as not every animal has those properties (venomous, fly, litter size), but the Animal class needs to have their declarations existing in it. This is because, in the main method, the type Animal is passed to the binary search tree (BST<Animal> bst;) and later on, the program needs to fetch the methods shown above. Hence, if they do not exist in the class Animal, an error will occur as the methods would only exist in the derived classes.

Abstraction was the solution to this problem, by creating pure virtual abstract methods in Animal class and implement them in the derived classes. Given that the derived classes have to implement every abstract method they inherit, it was made sure that the methods that do not belong to that class return an empty string or a 0 (in case of double). The table below gives you a clearer picture of how this was coded.

Bird class in Animal.h	Inherited abstract methods implementation I Animal.cpp

`//CLASS BIRD`
`class Bird: public Animal{`
`private:`
`  string noReturn = "";`
`  double noReturn1 = 0;`
`  string fly;`
 	
`public:`
`  Bird(string, double, string);`
`  void setFly(string);`
`  string getFly() override;`
`  double getLitterSize() override;`
`  string getVenomous() override;`
`};`
	`string Bird::getFly(){`
	`  return fly;`
	`}`
	
	`double Bird::getLitterSize() {`
	`  return noReturn1;`
	`}`
	`string Bird::getVenomous() {`
	`  return noReturn;`
	`}`

The attributes declared in the base class have a public access modifier as it is an abstract class. On the other hand, the derived classes have a private access modifier for the variables declared or initialized. This hides data from other classes, therefore, making use of encapsulation. All classes have their methods with a public access modifier so that they can be accessed from other classes.

Template Classes

The binary search tree data structure was constructed with the use of template classes so that the tree can accept different object types (Mammal, Bird, Reptile) without having to code a BST for each type that is used. This data structure was built with two classes:

- BstNode template class: contains any data which is related to nodes:
  - pointer to the data to be store
  - pointers to the left and right children.
- BST template class: contains all operations that the binary search tree is needed to have:
  - Insertion
  - Creating a new node
  - Searching for an animal
  - Delete a node
  - Get minimum data in the tree
  - Output all traversals

BstNode class

```
template <class T>
class BstNode {
public:
    T* animal;    //will store objects
    //will store the address to the node on the right and left
    BstNode<T>* left;
    BstNode<T>* right;
};
```

BST class

```
template <class T>
class BST{
private:
    BstNode<T> * root = NULL; //set root equal to NULL
public:

    BST();
    ~BST();
    void Traversals(); //called from main to run BST traversal
    //BstNode* getRoot();
    void startInsert(T*); //called from main to start the insertion of object process
    void startSearch(string);// called from main to start finding the sepcies read from file
    void startDeletion(string);
    BstNode<T>* CreateNode(T*); //creates a new node whenever needed
    BstNode<T>* Insert(BstNode<T>*, T*); //carries out the process to insert data in a node
    BstNode<T>* Find(BstNode<T>*, string);//performs the searching
    BstNode<T>* RemoveNode(BstNode<T>*, string);
    BstNode<T>* GetMinimum(BstNode<T>*);
    //BstNode* Delete(BstNode*, Animal);
    //BstNode* FindMin(BstNode*);
    //The below perform the tree traversals
    void OutputInOrder(BstNode<T>*);
    void OutputPostOrder(BstNode<T>*);
    void OutputPreOrder(BstNode<T>*);
    //called to output the details found in each root while traversing the tree
    void outputDetails(BstNode<T>*);
};
```

Class Breakdown

Object Classes

Animal			
Attribute	Variable Type	Brief Description	
Name	string	Stores animal name	
species	string	Store: mammal, reptile or bird	
length	double	Stores animal's length	
Method Name	Return Type	Parameter Type	Purpose
Animal	\	\	Constructor
~Animal	\	\	Destructor
getName	string	\	Returns animal name
getLength	double	\	Returns animal length
getSpecies	string	\	Returns animal kind
setName	\	string	Sets animal name
setLength	\	double	Sets animal length
getVenomous	string	\	Pure virtual abstract method
getFly	string	\	Pure virtual abstract method
getLitterSize	double	\	Pure virtual abstract method

Mammal			
Attribute	Variable Type	Brief Description	
noReturn	string	Stores an empty string: ""	
averageLitterSize	double	Stores mammal's litter size	
Method Name	Return Type	Parameter Type	Purpose
Mammal	\	string, double, double	Constructor
~Mammal	\	\	Destructor
getLitterSize	double	\	Returns litter size
setLitterSize	\	double	Sets animal litter size
getVenomous	string	\	Returns empty string
getFly	string	\	Returns empty string

Reptile		
Attribute	Variable Type	Brief Description
noReturn	string	Stores an empty string: “”
noReturn2	double	Stores 0
venom	string	Stores if venomous or not

Method Name	Return Type	Parameter Type	Purpose
Reptile	\	string, double, string	Constructor
~Reptile	\	\	Destructor
setVenomous	\	string	Sets venomous or not
getVenomous	string	\	Returns venomous or not
getFly	string	\	Returns empty string
getLitterSize	double	\	Returns 0

Bird		
Attribute	Variable Type	Brief Description
noReturn	string	Stores an empty string: “”
noReturn2	double	Stores 0
fly	string	Stores if flies or not

Method Name	Return Type	Parameter Type	Purpose
Bird	\	string, double, string	Constructor
~Bird	\	\	Destructor
setFly	\	string	Sets fly or not
getVenomous	string	\	Returns empty string
getFly	string	\	Returns fly or not
getLitterSize	double	\	Returns 0

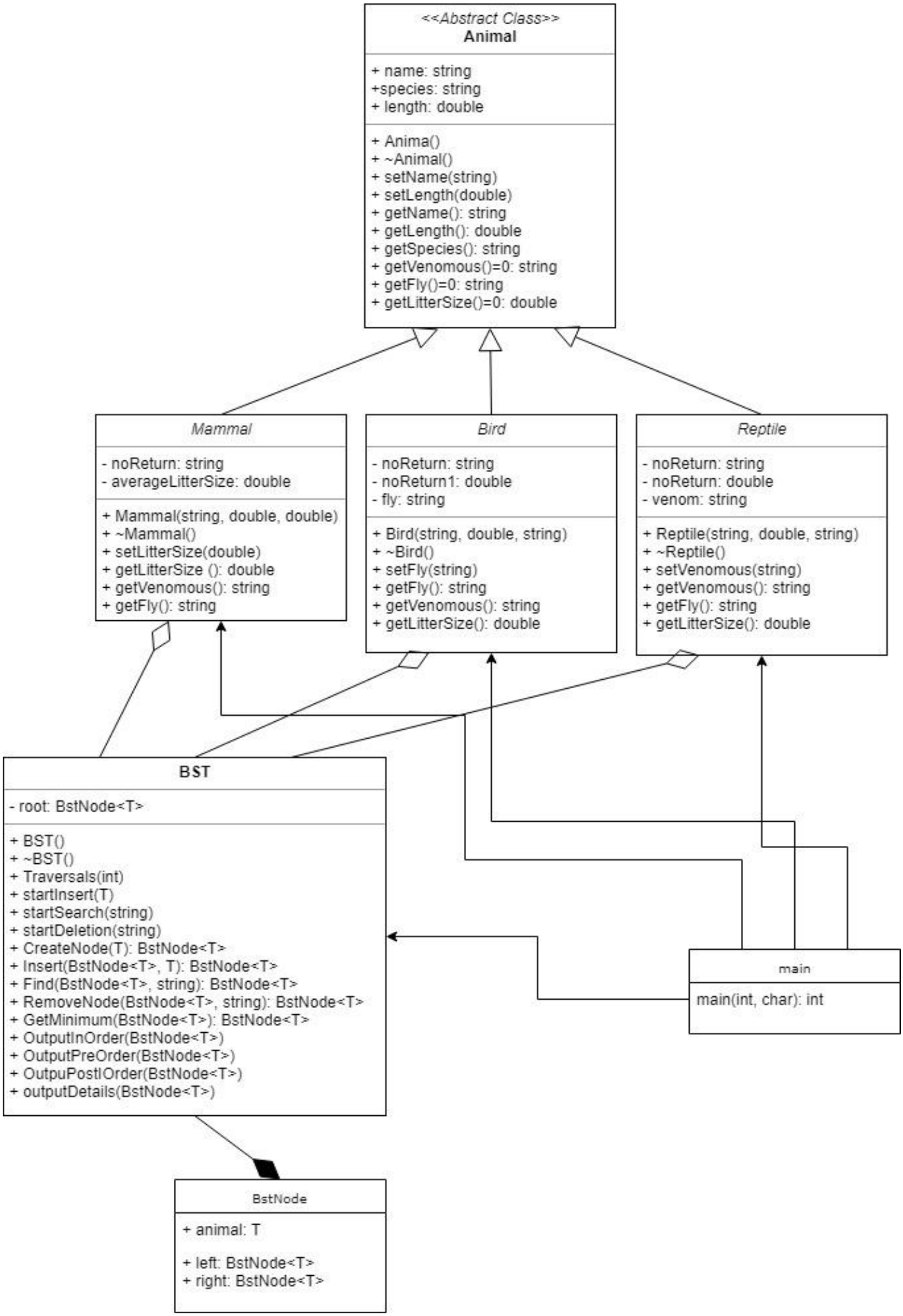
BST Object Class and BstNode Class

BST			
Attribute	Variable Type		Brief Description
Root	BstNode<T> *		Root points to NULL
Method Name	**Return Type**	**Parameter Type**	**Purpose**
BST	\	\	Constructor
~BST	\	\	Destructor
Traversals	\	Int	Called from main method and then calls one of the functions that carry out the tree traversals, passing the root. This method keeps root inside BST.h only instead of passing root from main method.
startInsert	\	T*	Called from main method and then calls the Insert function passing the data it gets from its parameter and the root. This method keeps root inside BST.h only instead of passing root from main method.
startSearch	\	string	Called from main method and then calls the Find function passing the word it gets from its parameter and the root. This method keeps root inside BST.h only instead of passing root from main method.
startDeletion	\	string	Called from main method and then calls the RemoveNode function passing the word it gets from its parameter and the root. This method keeps root inside BST.h only instead of passing root from main method.
CreateNode	BstNode<T>*	T*	Creates a new node whenever needed
Insert	BstNode<T>*	BstNode<T>*, T*	Carries out the process to insert new data into the bst
Find	BstNode<T>*	BstNode<T>*, string	Searches for the animal passed
RemoveNode	BstNode<T>*	BstNode<T>*, string	Deletes data from the tree
GetMinimum	BstNode<T>*	BstNode<T>*	Gets smallest data in the bst
OutputInOrder	\	BstNode<T>*	Performs inOrder traversal
OutputPreOrder	\	BstNode<T>*	Performs preOrder traversal

OutputPostOrder	\	BstNode<T>*	Performs postOrder traversal
outputDetails	\	BstNode<T>*	Called fromm the functions that perform the traversals to output the data found at each node

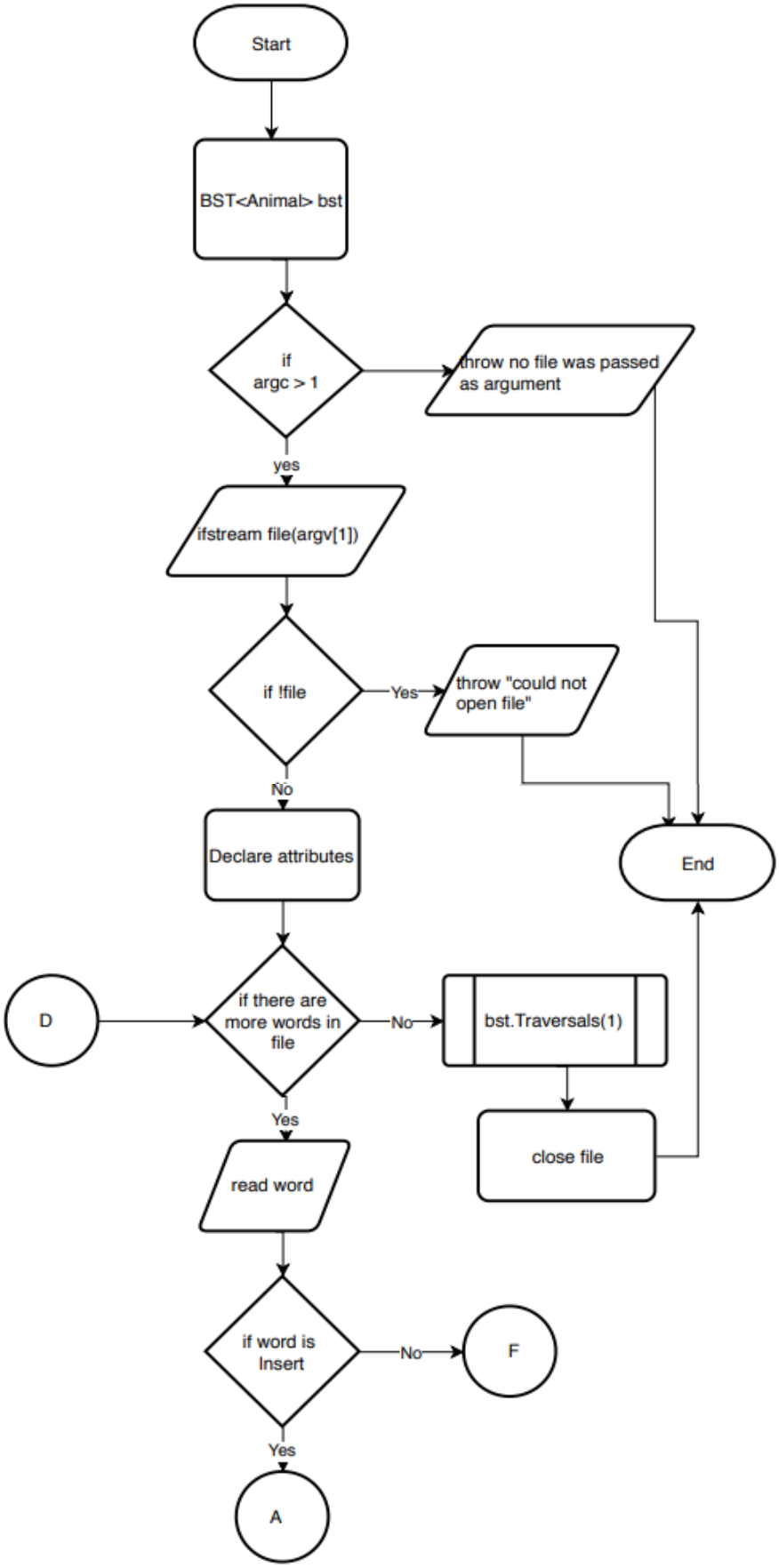
BstNode		
Attribute	Variable Type	Brief Description
animal	T*	Points to data stored in node
left	BstNode<T>*	Points to left child
right	BstNode<T>*	Points to right child

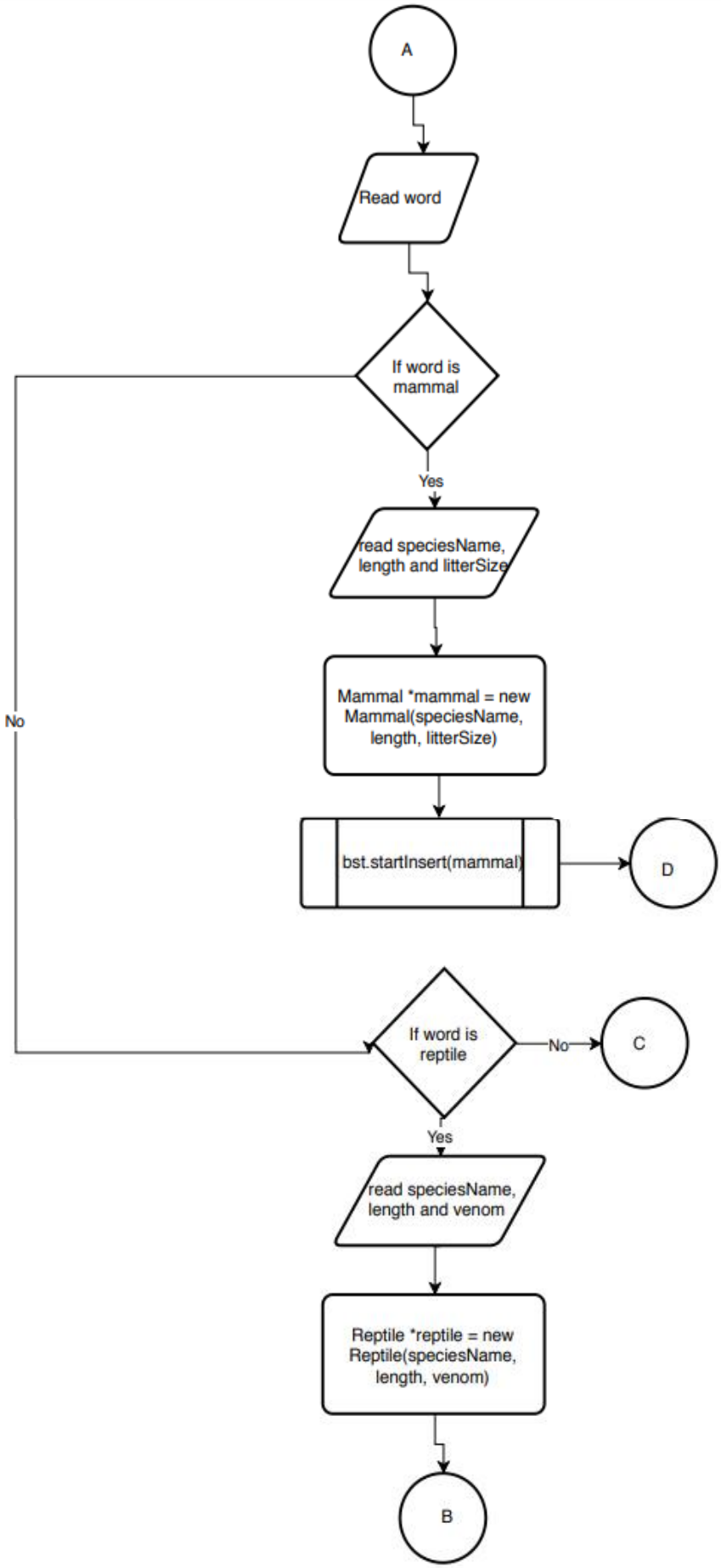
Class Diagram

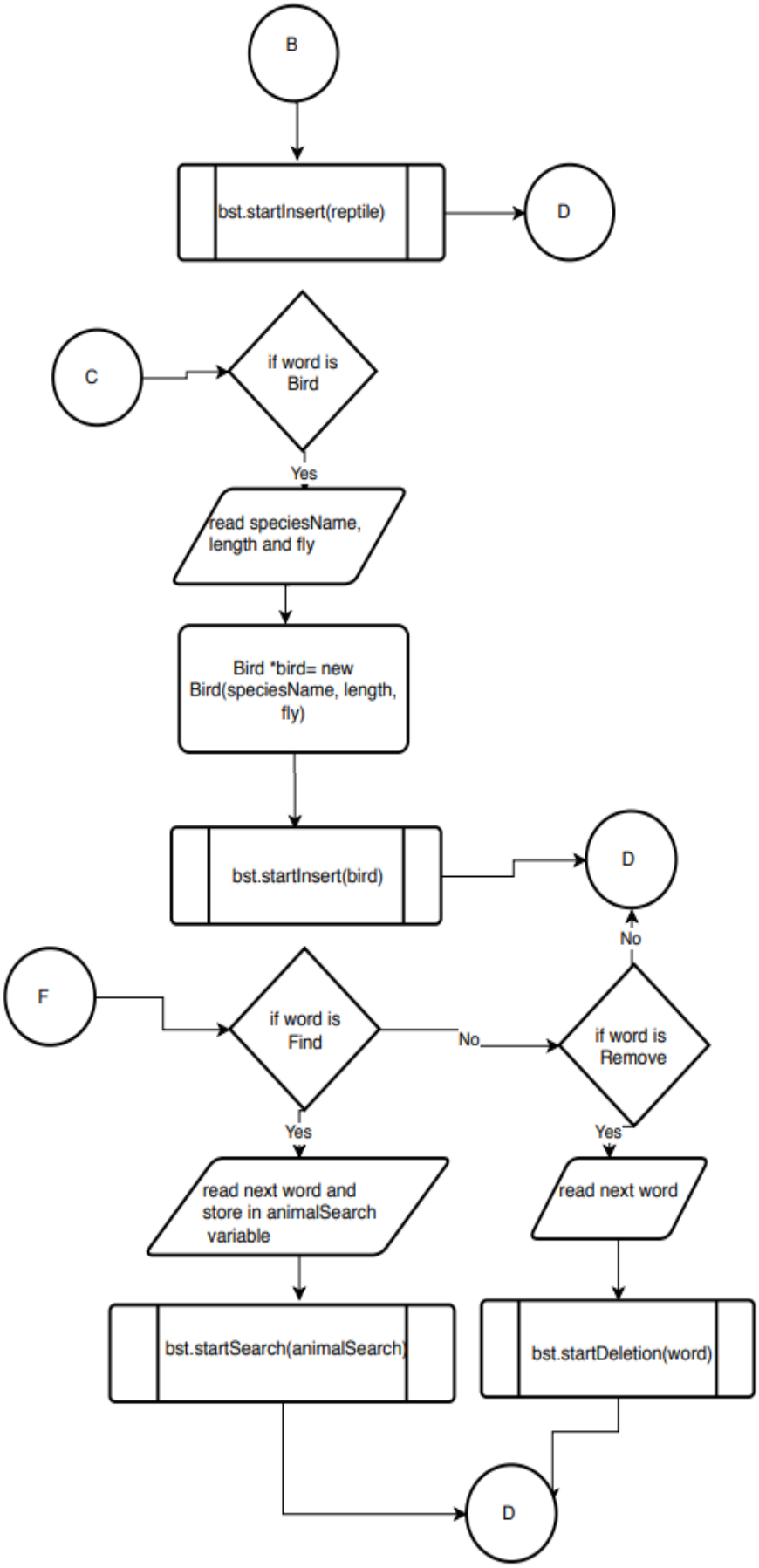


Flowchart

main




A
Read word
If word is mammal
Yes
No
read speciesName, length and litterSize
Mammal *mammal = new Mammal(speciesName, length, litterSize)
bst.startInsert(mammal)
D
If word is reptile
No
C
Yes
read speciesName, length and venom
Reptile *reptile = new Reptile(speciesName, length, venom)
B


B
bst.startInsert(reptile)
D
C
if word is Bird
Yes
read speciesName, length and fly
Bird *bird= new Bird(speciesName, length, fly)
bst.startInsert(bird)
D
No
F
if word is Find
No
if word is Remove
Yes
Yes
read next word and store in animalSearch variable
read next word
bst.startSearch(animalSearch)
bst.startDeletion(word)
D

Error Handling

Validation

In main.cpp

- First the program checks if an argument was pass in the command line. If not, the user is notified.

Pseudo Code:

If argc is greater than 1{

/*main body*/

}else{

throw an exception and tell user that nothing was passed as a parameter

}

- Next validation is when it tries to open the file. If it fails a message should be printed on screen to notify user.

Code:

```
ifstream file(argv[1]);
if (!file) {
    throw "could not open file";
}
```

In BST.h

- Inside the startSearch() function, the code checks if the animal was found or not so that the user will know what happened if no details are outputted.

Pseudo Code:

Get the pointer to the address returned by the Find() method and store it in nodeFound

If nodeFound is not equal to null{

Print the animal found details.

}else inform the user that the animal could not be found

- Inside the RemoveNode() function, the code checks if the animal to be deleted is found in the binary search tree or not.

Pseudo Code:

```
If root is equal to NULL{
        Print message on screen to inform the user that it could not be deleted.
}else
        Continue with the searching and deletion..
```

Proof of Running Application

This program should be run and compiled by the commands:

source ./compile.sh

source ./run.sh

1. When the text file has everything written down as it should be without any spelling mistakes:

```
viper was found
Length: 200     Species: Reptile        Venomous: venomous

Binary Search tree printed with InOrder Traversal:
Name: cat       Length: 60      Species: Mammal Litter Size: 4
Name: chameleon Length: 12      Species: Reptile        Venomous: non-venomous
Name: ostrich   Length: 150     Species: Bird   Fly: cannot-fly
Name: viper     Length: 200     Species: Reptile        Venomous: venomous
```

2. Exchanged the word eagle with eagles after the Remove command in the text file. Therefore, it could not be found in the binary search tree to eventually get deleted. Below screenshot is the output:

```
viper was found
Length: 200     Species: Reptile        Venomous: venomous

eagles could not be deleted! Make sure it is written correctly in .txt file

Binary Search tree printed with InOrder Traversal:
Name: cat       Length: 60      Species: Mammal Litter Size: 4
Name: chameleon Length: 12      Species: Reptile        Venomous: non-venomous
Name: eagle     Length: 80      Species: Bird   Fly: can-fly
Name: ostrich   Length: 150     Species: Bird   Fly: cannot-fly
Name: viper     Length: 200     Species: Reptile        Venomous: venomous
```

3. Exchanged the word viper with vipers after the Find command in the text file. As a result it can never be found in the tree and a message is printed to alert the user about it.

```
vipers could Not be Found!

Binary Search tree printed with InOrder Traversal:
Name: cat       Length: 60      Species: Mammal Litter Size: 4
Name: chameleon Length: 12      Species: Reptile        Venomous: non-venomous
Name: ostrich   Length: 150     Species: Bird   Fly: cannot-fly
Name: viper     Length: 200     Species: Reptile        Venomous: venomous
```

4. Ran the program without passing an argument. Error message is printed as shown below.

```
jonvel@ubuntu:~/Assignment OOP/CPlusPlusProgram$ ./main
No file was passed as an argument!jonvel@ubuntu:~/Assignment
```

Source Code

Java

Main.java

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

public class Main {

    public static void main(String[] args){

        Observer observe = new Observer();
        ArrayList<Restaurants> restaurantsList = new ArrayList<>(); //stores restaurants
objects
        ArrayList<Item> itemList = new ArrayList<>(); //used to store the list of the restaurants'
menu items and also used
        //to store the list of items being ordered

        Restaurants restaurantObject;
        Item itemObject;

        String token;
        String restaurantName;
        String service;
        String itemname;
        int restaurantIndex;
        double price;

        if(args.length<1){  //checks if an argument was passed in the command line
            //if there is no argument output a message and exit
            System.out.println("Error! No file was passed in the command line");
            System.exit(1);
        }

        try {
            Scanner scanner = new Scanner(new FileInputStream(args[0]));

            while (scanner.hasNext()) {
                token = scanner.next();

                if(token.equals("BeginRestaurant")){
                    //ASSUMING THAT THE COMMANDS ARE GIVEN IN THE ORDER FOUND
IN THE ASSIGNMENT SPECIFICATION

                    restaurantName = scanner.next();  //get restaurant name from text file
                    service = scanner.next();         //takeaway, delivery or both?
                    token = scanner.next();           //command has to be 'Item'

                    do{
                        itemname= scanner.next();     //after command Item, there has to be the
name
```

```
                    price = scanner.nextDouble(); //followed by the price
                    itemObject = new Item(itemname,price);
                    itemList.add(itemObject);
                    token = scanner.next();

                    //itemObject = new Item(itemname, price); //create an object for the item
                    //itemList.add(itemObject);   //add item to an Array list of items
                }while(token.equals("Item"));   //If command is 'Item' iterate again else it is
'EndRestaurant'

                restaurantObject = new Restaurants(restaurantName, service, itemList);
//create an object for the restaurant
                restaurantsList.add(restaurantObject);  //add object to the restaurant array list

            }
            itemList.clear(); //clear list to be able to start a new one for a new Restaurants
Object

            if(token.equals("BeginOrderList")){
                //ASSUMING THAT THE COMMANDS ARE GIVEN IN THE ORDER FOUND
IN THE ASSIGNMENT SPECIFICATION

                token = scanner.next();     //has to be 'BeginOrder'

                restaurantIndex = startingNewOrder(restaurantsList, scanner);
                service = scanner.next();     //has to be method to be served by
                token = scanner.next();       //has to be 'OrderItem' command
                if(restaurantIndex < 0){
                    System.out.println("Please make sure the information in the text file is
correct!");
                }else {
                    collectOrderItems(restaurantsList, itemList, restaurantIndex, service,
scanner);
                }
            }
        }

        observe.calculateHighestRevenue(restaurantsList);
    }catch(FileNotFoundException f){
        System.out.println("File could not be found!");
    }
}

  private static int searchRestaurant(ArrayList<Restaurants>restaurantsListIn, String
nameIn){
    int indexFound = -1;
    for(int i = 0; i<restaurantsListIn.size(); i++){
        if (nameIn.equals(restaurantsListIn.get(i).getName())){
            indexFound = i;
        }
    }
    return indexFound;
}
```

```
    private static int searchItem(ArrayList<Restaurants>restaurantsListIn, int indexFound,
String itemName){
        boolean flag =false;
        int i=0;
        int itemIndexFound= -1;
        do{
            if
(itemName.equals(restaurantsListIn.get(indexFound).getItemList().get(i).getName())){
                itemIndexFound = i;
                flag = true;
            }
            i++;
        }while(!flag);
        return itemIndexFound;
    }

    private static int startingNewOrder( ArrayList<Restaurants> restaurantsList, Scanner
scanner)
    {
        String restaurantName;
        restaurantName = scanner.next();     //has to be restaurant name

        int restaurantIndex = searchRestaurant(restaurantsList, restaurantName); //gets index
where the restaurant is stored
        if(restaurantIndex ==-1){
            System.out.println("Restaurant name cannot be found!");}

        return restaurantIndex;
    }

    private static void collectOrderItems( ArrayList<Restaurants>restaurantsList,
ArrayList<Item>itemList, int restaurantIndex, String service, Scanner scanner){

        Item itemObject;
        Order orderObject;

        myLinkedList linkedList = new myLinkedList();
        Observer observe = new Observer();

        String token;
        String restaurantName = restaurantsList.get(restaurantIndex).getName();
        String itemname;
        int itemIndex ;
        double price=0.0;
        double totalOrderPrice;


        do{
            itemname = scanner.next();  //has to be item name
            try {
                itemIndex = searchItem(restaurantsList, restaurantIndex, itemname); //gets index
where the item is stored
```

```
            price =
restaurantsList.get(restaurantIndex).getItemList().get(itemIndex).getPrice();  //get the price of
the item
        }catch (IndexOutOfBoundsException a){
            System.out.println("The item: "+itemname+" is not offered by
"+restaurantsList.get(restaurantIndex).getName());
        }
        totalOrderPrice = observe.calculateTotalOrderPrice(price);
        //totalOrderPrice = totalOrderPrice + price;  //calculating total order price with each
loop
        itemObject = new Item(itemname, price); //create an object for the item
        itemList.add(itemObject);   //add item to an Array list of items

        token = scanner.next(); //either another 'OrderItem' command or a 'EndOrder'
command

        if(token.equals("EndOrder")) {   //if it is not an Order item but the order is ended take
the next input
            token = scanner.next();     //either 'BeginOrder' command or 'EndOrderList'
        }
    }while(token.equals("OrderItem"));  //checks if the last read command is another
OrderItem if not exit loop


    restaurantsList.get(restaurantIndex).setRevenue(totalOrderPrice);

    orderObject = new Order(restaurantName, service, itemList, totalOrderPrice); //create
an object for the order
    linkedList.add(orderObject);

    itemList.clear(); //clear list to be able to start a new one for a new Restaurants Object

    if(token.equals("BeginOrder")){
        restaurantIndex = startingNewOrder(restaurantsList, scanner);
        service = scanner.next();     //has to be method to be served by
        token = scanner.next();       //has to be 'OrderItem' command
        if(restaurantIndex < 0){
            System.out.println("Please make sure the information in the text file is correct!");
        }else {
            collectOrderItems(restaurantsList, itemList, restaurantIndex, service, scanner);
        }
    }
    linkedList.writeAll();
  }
}
```

Restaurants.java

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

public class Restaurants implements Serializable{
    private String name;
    private String methodOfServing;
    private ArrayList<Item> itemList = new ArrayList<>();
    private double revenue = 0.0;

    public Restaurants(String nameIn, String methodOfServingIn, ArrayList<Item>itemListIn){
        name = nameIn;
        methodOfServing = methodOfServingIn;
        itemList.addAll(itemListIn);
    }

    //getters
    public String getName(){
        return name;
    }

    public double getRevenue() {
        return revenue;
    }

    public ArrayList<Item> getItemList() {
        return itemList;
    }

    public void setItemList(ArrayList<Item> itemListIn) {
        ArrayList<Item> itemList = new ArrayList<Item>(itemListIn);
    }

    public String getMethodOfServing(){
        return methodOfServing;
    }


    //setters
    public void setName(String nameIn){
        name = nameIn;
    }

    public void setRevenue(double revenueIn) {
        revenue = revenue + revenueIn;
    }
    public void setMethodOfServing(String methodOfServingIn) {
        methodOfServing = methodOfServingIn;
    }

}
```

Order.java

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

public class Order implements Serializable { //inherits the restaurants' variables

    private double totalOrderPrice;
    private String restaurantName;
    private String service;
    private ArrayList<Item> itemList = new ArrayList<>();

    public Order(String restaurantNameIn, String serviceIn, ArrayList<Item> itemListIn, double
totalOrderPriceIn){
        totalOrderPrice = totalOrderPriceIn;
        restaurantName = restaurantNameIn;
        service = serviceIn;
        itemList.addAll(itemListIn);
    }

    public void setRestaurantName(String restaurantName) {
        this.restaurantName = restaurantName;
    }

    public void setOrderList(ArrayList<Item> orderListIn) {
        ArrayList<Item> orderList = new ArrayList<>(orderListIn);
    }

    public void setService(String service) {
        this.service = service;
    }

    public void setTotalOrderPrice(double totalOrderPrice) {
        this.totalOrderPrice = totalOrderPrice;
    }

    public String getRestaurantName() {
        return restaurantName;
    }

    public double getTotalOrderPrice() {
        return totalOrderPrice;
    }

    public String getService() {
        return service;
    }

    public ArrayList<Item> getItemList() {
        return itemList;
    }
}
```

Item.java

```
public class Item {

    private double price;
    private String name;

    public Item(String nameIn, double priceIn){
        price = priceIn;
        name= nameIn;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public double getPrice() {
        return price;
    }

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

myLinkedList.java

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;

class myLinkedList {

    private ListNode head; //variable of type ListNode points to the head of the list
    private int sizeOfList;

    myLinkedList() {
        head = null; //point the linked list head to null
        sizeOfList = 0;
    }


    public void add(Order newOrder) { //adds objects of type Order into the linked list
        ListNode newNode = new ListNode(newOrder); //the new node to store the order to be
added
        if (head == null) {
            head = newNode;  //if there is no node in the linked list yet, create the first one
        } else {
            ListNode temp = head; //else start from the head node of the list
            while (temp.next != null) { //while the linked list is not fully traversed
                temp = temp.next; //set temp to the next node of the linked list
            }
            temp.next = newNode; //creates the last node of the list after temp.next reaches null
        }
        sizeOfList++;
    }


    public void writeAll() { //this writes all orders stored in the linked list to the file called
Orders.txt

        if (head != null) { //checks if there is anything inside the list
            ListNode temp = head; //starting from the first node
            try {
                Writer wr = new FileWriter("Orders.txt", true); //create or open file Orders.txt
                do {
                    wr.write("---NEW ORDER---\n" + temp.order.getRestaurantName() + "\t");
//writes restaurant name
                    wr.write(temp.order.getService() + "\nOrdered Items:"); //writes mode of service

                    for (Item item : temp.order.getItemList()) { //loops through the whole Arraylist
which contains ordered items
                        wr.write(item.getName() + "\t");  //writes name of food
                        wr.write(item.getPrice() + "euros\n");//writes price of food
                    }

                    wr.write("Total Order Price" + temp.order.getTotalOrderPrice() + "\n\n"); //writes
the total price of the order
```

```
                temp = temp.next; //move to the next node
            } while (temp != null); //continue looping until temp reaches null

            wr.flush();
            wr.close(); //closing file

        } catch (IOException e) {
            System.out.println("There was a problem in opening or creating the file!");
        }
    }
  }
}
```

ListNode.java

```
public class ListNode {


    ListNode next;  //used to point to the next node
    Order order;

    public ListNode(Order newOrder){
        next = null;
        order = newOrder;
    }

}
```

Observer.java

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;
import java.util.ArrayList;

public class Observer {

    private double revenue = 0.0;
    private double highestRevenue = 0.0;
    private int indexOfHighestRevenue;
    private double totalOrderPrice = 0.0;

    public void calculateHighestRevenue(ArrayList<Restaurants>restaurantsList){
        for (int i=0; i < restaurantsList.size(); i++){
            revenue = restaurantsList.get(i).getRevenue();
            if(revenue>highestRevenue){
                highestRevenue = revenue;
                indexOfHighestRevenue = i;
            }
        }
```

```
        try {
            Writer wr = new FileWriter("Orders.txt",true);
            // wr.write("PRICE: " + totalOrderPrice +"\n");  // write int
            wr.write("Restaurant with highest revenue is: "+
restaurantsList.get(indexOfHighestRevenue).getName() +"\n");  // write int
            wr.write("Revenue: "+ highestRevenue);
            wr.flush();
            wr.close();
        }catch(IOException e)
        {
            System.out.println("There was a problem writing the file");
        }
    }


    public double calculateTotalOrderPrice(double priceIn){
        totalOrderPrice = priceIn + totalOrderPrice;
        return totalOrderPrice;
    }

}
```

C++

Main.cpp

```
#include<iostream>
#include "BST.h"
#include "fstream"
#include "Animal.h"
using namespace std;

int main(int argc, char* argv[]) {

    BST<Animal> bst;

    if (argc > 1) {
        ifstream file(argv[1]);
        if (!file) {
            throw "could not open file";
        }

        string speciesName;
        double length;
        double litterSize;
        string venom;
        string fly;
        string animalSearch;

        for (string word; file >> word;) { //loops through all the words inside the file
            if (word == "Insert") { //if word is Insert

                file >> word; //read the next word which specifies what type of animal it is

                //start checking if the word is mammal, reptile or bird
                if (word == "mammal") {

                    file >> word;
                    speciesName = word; //storing the word inside species variable
                    file >> word; // reads next word
                    length = stod(word);//stores the length
                    file >> word; //reads next word
                    litterSize = stod(word);// stores the average litter size

                    Mammal *mammal = new Mammal(speciesName, length, litterSize);
                    bst.startInsert(mammal);

                } else if (word == "reptile") {

                    file >> word;
                    speciesName = word; //storing the word inside species variable
                    file >> word; // reads next word
                    length = stod(word);//stores the length
                    file >> word; //reads next word
                    venom = word;// stores the average litter size
```

```
                Reptile *reptile = new Reptile(speciesName, length, venom);
                bst.startInsert(reptile);

            } else if (word == "bird") {
                file >> word;
                speciesName = word; //storing the word inside species variable
                file >> word; // reads next word
                length = stod(word);//stores the length
                file >> word; //reads next word
                fly = word;// stores the average litter size

                Bird *bird = new Bird(speciesName, length, fly);
                bst.startInsert(bird);
            }
        } else if (word == "Find") { //if the word read is Find then start the search for the
animal

            file >> word;
            animalSearch = word;
            bst.startSearch(animalSearch);

        } else if (word == "Remove") {//if the word is Remove start the deletion
            file >> word;
            bst.startDeletion(word);

        }
    }

    cout << "Binary Search tree printed with InOrder Traversal:" << endl;

    bst.Traversals(1);
    file.close();

  } else { //if argc is <1
      throw "No file was passed as an argument!";

  }

  return 0;
}
```

Animal.h

```
#ifndef CPLUSPLUSPROGRAM_ANIMAL_H
#define CPLUSPLUSPROGRAM_ANIMAL_H


#include <string>
using namespace std;

//CLASS ANIMAL
class Animal{

public:
    string name;
    string species;
    double length;

    Animal();
    virtual ~Animal()=0; //destructor

    //setters
    void setName(string);
    void setLength(double);

    //getters
    string getName();
    double getLength();
    string getSpecies();

    //the below make Animal class an Abstract class
    virtual string getVenomous()=0;
    virtual string getFly()=0;
    virtual double getLitterSize()=0;
};

//THE FOLLOWING ARE ALL DERIVED CLASSES FROM THE ANIMALS CLASS

//CLASS MAMMAL
class Mammal: public Animal {
private:
    string noReturn = "";
    double averageLitterSize;
public:

    Mammal(string , double ,double );
    ~Mammal()override;
    void setLitterSize(double);
    double getLitterSize() override;
    string getVenomous() override;
    string getFly()override;

};

//CLASS REPTILE
```

```
class Reptile: public Animal{
private:
    string noReturn = "";
    double noReturn1 = 0;
    string venom;

public:
    Reptile(string, double, string);
    ~Reptile()override;
    void setVenomous(string);
    string getVenomous() override;
    string getFly()override;
    double getLitterSize() override;
};

//CLASS BIRD
class Bird: public Animal{
private:
    string noReturn = "";
    double noReturn1 = 0;
    string fly;

public:
    Bird(string, double, string);
    ~Bird()override;
    void setFly(string);
    string getFly() override;
    double getLitterSize() override;
    string getVenomous() override;
};

#endif //CPLUSPLUSPROGRAM_ANIMAL_H
```

Animal.cpp

```
#include <iostream>
#include "Animal.h"
//FOR BASE CLASS ANIMAL

Animal::Animal(){}

Animal::~Animal() { //destructor
}

//SETTERS
void Animal::setName(string nameIn){
   name = nameIn;
}

void Animal::setLength(double lengthIn){
   length = lengthIn;
}

//GETTERS
string Animal::getName(){
   return name;
}

double Animal::getLength(){
   return length;
}

string Animal::getSpecies(){
   return species;
}

//FOR CLASS MAMMAL
Mammal::~Mammal() {}


Mammal::Mammal(string nameIn, double lengthIn,double averageLitterSizeIn){
   this->name=nameIn;
   this->length=lengthIn;
   averageLitterSize = averageLitterSizeIn;
   species = "Mammal";
}

void Mammal::setLitterSize(double litterSizeIn){
   averageLitterSize = litterSizeIn;
}
string Mammal:: getVenomous() {
   return noReturn;
}

string Mammal::getFly(){
   return  noReturn;
```

```
}

double Mammal:: getLitterSize(){
   return averageLitterSize;
}

//FOR CLASS REPTILE

Reptile::Reptile(string nameIn, double lengthIn,string venomIn){
   this->name=nameIn;
   this->length=lengthIn;
   venom = venomIn;
   species = "Reptile";
}

Reptile::~Reptile() {}

void Reptile::setVenomous(string venomIn){
   venom = venomIn;
}

string Reptile::getVenomous(){
   return venom;
}

string Reptile::getFly(){
   return  noReturn;
}
double Reptile::getLitterSize() {
   return noReturn1;
}


//FOR CLASS BIRD

Bird::Bird(string nameIn, double lengthIn,string flyIn){
   this->name=nameIn;
   this->length=lengthIn;
   fly = flyIn;
   species = "Bird";
}

Bird::~Bird(){}

void Bird::setFly(string flyIn) {
   fly = flyIn;
}

string Bird::getFly(){
   return fly;
}

double Bird::getLitterSize() {
   return noReturn1;
}
```

```
string Bird::getVenomous() {
    return noReturn;
}
#ifndef CPLUSPLUSPROGRAM_BST_H
#define CPLUSPLUSPROGRAM_BST_H
```

BST.h

```
//includes
#include "Animal.h"
#include <iostream>

//class for Nodes
template <class T>
class BstNode {
public:
    T* animal;    //will store objects
    //will store the address to the node on the right and left
    BstNode<T>* left;
    BstNode<T>* right;
};

template <class T>
class BST{
private:
    BstNode<T> * root = NULL; //set root equal to NULL
public:

    BST();
    ~BST();
    void Traversals(int); //called from main to run BST traversal
    void startInsert(T*); //called from main to start the insertion of object process
    void startSearch(string);// called from main to start finding the sepcies read from file
    void startDeletion(string);
    BstNode<T>* CreateNode(T*); //creates a new node whenever needed
    BstNode<T>* Insert(BstNode<T>*, T*); //carries out the process to insert data in a node
    BstNode<T>* Find(BstNode<T>*, string);//performs the searching
    BstNode<T>* RemoveNode(BstNode<T>*, string); //deletes data and its respective node
from the bst
    BstNode<T>* GetMinimum(BstNode<T>*);//gets smallest data inside the bst

    //The below perform the tree traversals
    void OutputInOrder(BstNode<T>*);
    void OutputPostOrder(BstNode<T>*);
    void OutputPreOrder(BstNode<T>*);
    //called to output the details found in each root while traversing the tree
    void outputDetails(BstNode<T>*);
};

#endif //CPLUSPLUSPROGRAM_BST_H


template <class T>
BST<T>::BST(){

}

template <class T>
BST<T>::~BST(){}
```

```
template <class T>
void BST<T>::startInsert(T* animal){ //called from main whenever an object is to be added to
the tree
   //this method is called first instead of the Insert() so that information of the root stays
inside BST.h file
   root = Insert(root, animal); //update root with new root
}

template <class T>
//Inserts objects in the tree. Uses Recursion to do so
BstNode<T>* BST<T>::Insert(BstNode<T>* root,T *animal) {

   if(root == nullptr) { //no node exists at the address to which the root points to
      root = CreateNode(animal); //therefore, create a new node and pass the data to store in
it
   }
      // if the name of the animal to be inserted is less than the data inside the current root
   else if(animal->getName() <= root->animal->getName()) {
      root->left = Insert(root->left,animal); //recursive call to Insert and update the left child of
the root
   }
      // else, recursive call to insert in right subtree.
   else {
      root->right = Insert(root->right,animal);
   }
   return root;
}

template <class T>
BstNode<T>* BST<T>::CreateNode(T *animal) { //called when data needs to be added in a
new node
   BstNode<T>* newNode = new BstNode<T>(); //points to the Node created
   newNode->animal = animal; //set the animal in node to the object passed
   newNode->left = newNode->right = nullptr; //children pointing to NULL
   return newNode;
}

template <class T>
void BST<T>:: startSearch(string species){ //called from main to start the search
   BstNode<T>* nodeFound = Find(root, species); //gets the address where the species is, if
found
   if(nodeFound!=nullptr) { //if it is not null then it means it was found
      cout<<species<<" was found"<<endl;
      cout<<"Length: "<<nodeFound->animal->getLength(); //get length and species type
      cout<<"\tSpecies: "<<nodeFound->animal->getSpecies();

      if(nodeFound->animal->getSpecies() == "Mammal"){
         cout<<"\tLitter Size: "<<nodeFound->animal->getLitterSize()<<"\n"<<endl;
      }else if(nodeFound->animal->getSpecies() == "Reptile"){
         cout<<"\tVenomous: "<<nodeFound->animal->getVenomous()<<"\n"<<endl;
      }else{
         cout<<"\tFly: "<<nodeFound->animal->getFly()<<"\n"<<endl;
      }
   }
```

```
    else cout<<species<<" could Not be Found!\n"<<endl; //output error message
}

template <class T>
BstNode<T>* BST<T>::Find(BstNode<T>* root,string animal) { //searches for the species
    if(root == nullptr) { //no tree
        return nullptr;
    }
    else if(root->animal->getName() == animal) { //if found
        return root; //returns address of the node which contains the searched animal
    }
    else if(animal <= root->animal->getName()) { //traversing the tree
        return Find(root->left,animal); //search in the left subtree of the current root
    }
    else {
        return Find(root->right,animal);//search in the right subtree of the current root
    }
}

template <class T>
void BST<T>:: startDeletion(string species) { //called from main to start the search
    RemoveNode(root, species);
}

template <class T>
BstNode<T>* BST<T>::RemoveNode(BstNode<T>* root, string species){
    if(root == nullptr) //animal not found in tree
    {   cout<<species<<" could not be deleted! Make sure it is written correctly in .txt
file\n"<<endl;
        return root;
    }
    else if (species > root->animal->getName()) { //if animal passed is greater than that found
in root
        root->right = RemoveNode(root->right,species); //search in right child recursively
    }
    else if(species < root->animal->getName(){//if animal passed is less than animal in root
        root->left = RemoveNode(root->left,species); //search in left child recursively
    }
    //when animal is found execute following code
    else{

        if (root->left == nullptr && root->right == nullptr) { //if the node has no children (leaf
node)
            delete root; //deallocate memory of root from heap
            root = nullptr; //set the root to now point to NULL
        }
        else if (root->left == nullptr) { //if the node has one child in this case right child
            BstNode<T>* temporaryNode = root;//store address of node that will be deleted in a
temporary pointer to root
            root = root->right; //make the right child the root of the subtree
            delete temporaryNode; //deallocate memory of temporaryNode from heap
        }
        else if (root->right == nullptr) { //if the node has one child in this case left child
```

```
            BstNode<T>* temporaryNode = root;//store address of node that will be deleted in a
temporary pointer to root
            root = root->left; //make the left child the root of the subtree
            delete temporaryNode; //deallocate memory of temporaryNode from heap
        }

        else { //when the node has 2 children
            BstNode<T>* temporaryNode = GetMinimum(root->right); //search for minimum data
in the right subtree
            root->animal = temporaryNode->animal; //set the data to be deleted to the minimum
value found
            string species = root->animal->getName();
            root->right = RemoveNode(root->right, species);
        }
    }
    return root;
}

template <class  T>
BstNode<T>* BST<T>::GetMinimum(BstNode<T>* root)
{
    while(root->left != NULL){
        root = root->left; //gets the bottom most node on the left side of the root
        //i.e gets the far left leaf node
    }
    return root; //returns leaf node
}



template <class T>
void  BST<T>::Traversals(int num){ //called from main
    if(num==1) {
        OutputInOrder(root); //only calls the method that performs the inOrder traversal as only
that should be printed throughout the whole program
    }
    if (num == 2){
        OutputPostOrder(root);
    }
    if(num == 3){
        OutputPreOrder(root);
    }
}

//traverse subtree, then the root of the subtree, then right subtree
template <class T>
void BST<T>::OutputInOrder(BstNode<T>* root){
    if(root != nullptr){
        OutputInOrder(root->left); //recursive call to traverse the left subtree
        outputDetails(root);
        OutputInOrder(root->right);//recursive call to traverse the right subtree
    }
}
```

```
//traverse the left subtree followed by all nodes in the right subtree, then the root
template <class T>
void BST<T>::OutputPostOrder(BstNode<T>* root) {
    if(root!= nullptr){
        OutputPostOrder(root->left);//recursive call to traverse the left subtree
        OutputPostOrder(root->right);//recursive call to traverse the right subtree
        outputDetails(root);
    }
}

//traverse the root then left subtree then right subtree
template <class T>
void BST<T>::OutputPreOrder(BstNode<T>* root){
    if(root!= nullptr){
        outputDetails(root);
        OutputPreOrder(root->left);//recursive call to traverse the left subtree
        OutputPreOrder(root->right);//recursive call to traverse the right subtree
    }
}

template <class T>
void BST<T>::outputDetails(BstNode<T>* root){

    cout<<"Name: "<<root->animal->getName()<<"\tLength: "<<root->animal->getLength();
    cout<<"\tSpecies: "<<root->animal->getSpecies();
    if(root->animal->getSpecies() == "Mammal"){
        cout<<"\tLitter Size: "<<root->animal->getLitterSize()<<endl;
    }else if(root->animal->getSpecies() == "Reptile"){
        cout<<"\tVenomous: "<<root->animal->getVenomous()<<endl;
    }else{
        cout<<"\tFly: "<<root->animal->getFly()<<endl;
    }
}
```