DSJ Mini Project

Topic: Generation of a UML diagram for a given Java Code.

By:

Sharath K.P(USN:1PI13CS141)

Shashwath.S.Bharadwaj(USN:1PI13CS144)

Shreyas Gandhi(USN:1PI13CS154)

Anirudh Agarwal(USN:1PI13IS020)

Description:

The objective of the Mini Project was to generate a UML diagram for a given Java source code file. The UML diagram would be indicative of the features of the Client code(attributes and behaviour) as well as those pieces of code on which it depended(Inheritance, Implementation of interfaces etc).

The data structure used for the implementation of the UML generator is a Multi-List(Linked list in 2 Dimensions), with every node in the list representing a different class that the executing program would depend on.

The interaction of the user with the code will only be to enter the names of files which are to be used for the execution of the given program. The linked UML diagram will appear on the GUI.

Implementation:

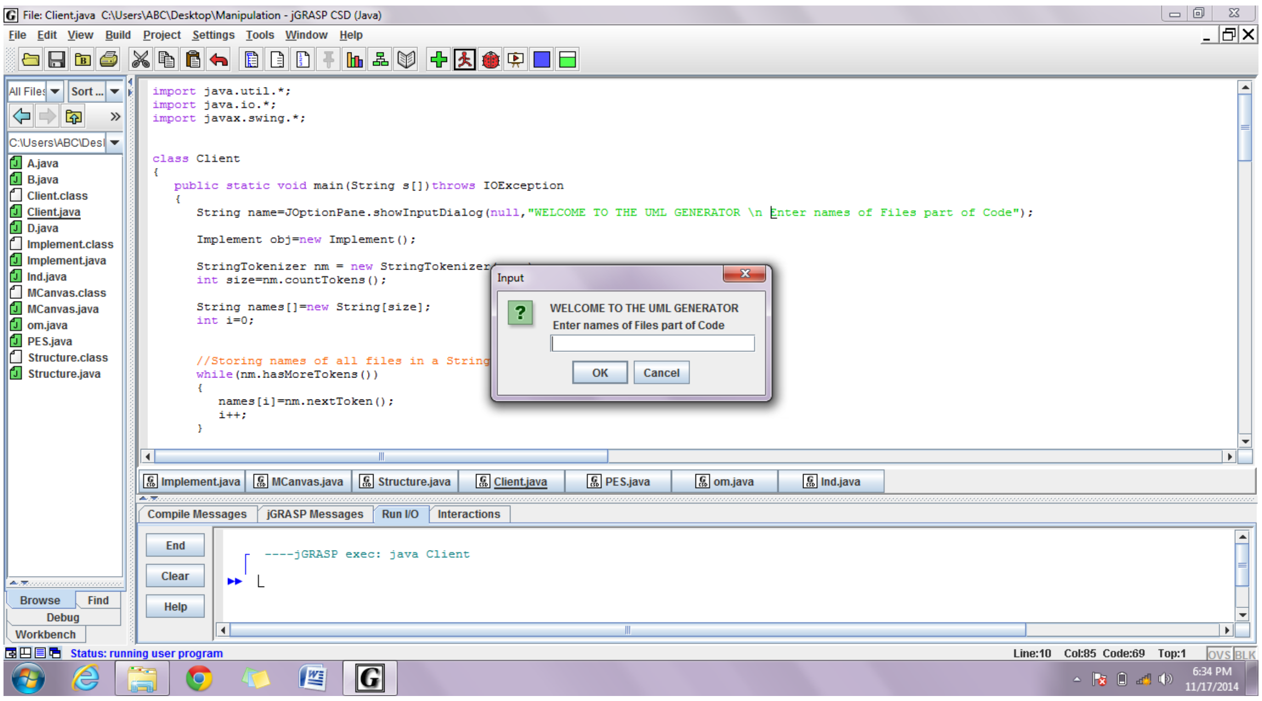
The implementation of the project is as follows:

1. The user is required to enter the names of files which contain the program to be executed as well as the names of all other files on which the program depends i.e if the class to be executed inherits from another, the name of the file containing the super class is also to be entered by the user.(Super class name is to be entered first)
2. The names of files are stored in an array and are read sequentially.
3. Upon reading a file, a node is created which contains the class name of the file read, it's attributes and behaviours.
4. If the file makes use of the code present in a separate file, the two nodes for the respective files are linked horizontally in case of a hierarchy(inheritance or interface implementation) and vertically otherwise.
5. The list is traversed starting with the root, vertically till no more nodes are to be traversed and then horizontally till no more nodes are to

be traversed. For each horizontally linked node encountered, a vertical traversal is done to read all the nodes that are vertically attached to it.

6. The UML diagram is generated during the traversal and is output once the traversal is complete.

Screenshots:



File: Client.java C:\Users\ABC\Desktop\Manipulation - jGRASP CSD (Java)

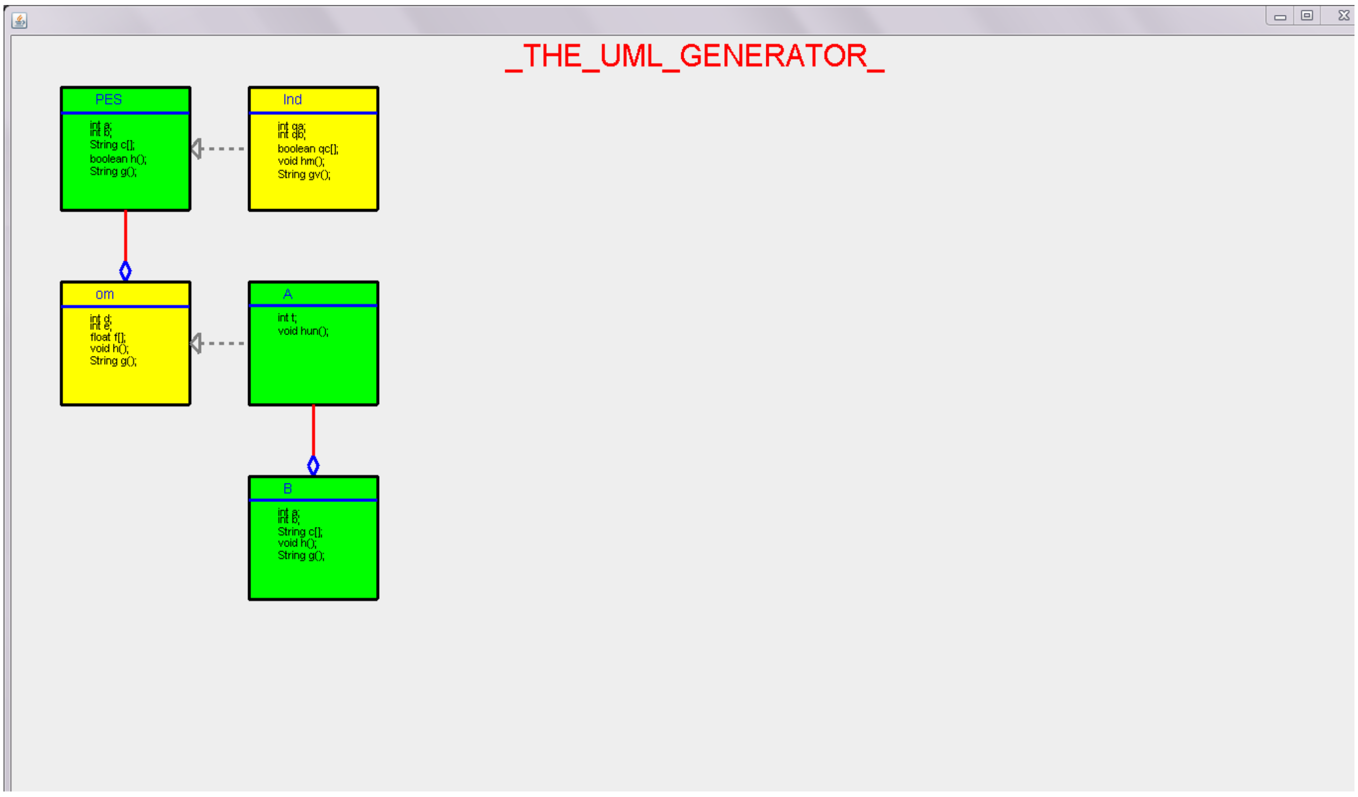
```java
import java.util.*;
import java.io.*;
import javax.swing.*;


class Client
{
   public static void main(String s[])throws IOException
   {
      String name=JOptionPane.showInputDialog(null,"WELCOME TO THE UML GENERATOR \n Enter names of Files part of Code");

      Implement obj=new Implement();

      StringTokenizer nm = new StringTokenizer
      int size=nm.countTokens();

      String names[]=new String[size];
      int i=0;


      //Storing names of all files in a String
      while(nm.hasMoreTokens())
      {
         names[i]=nm.nextToken();
         i++;
      }
```

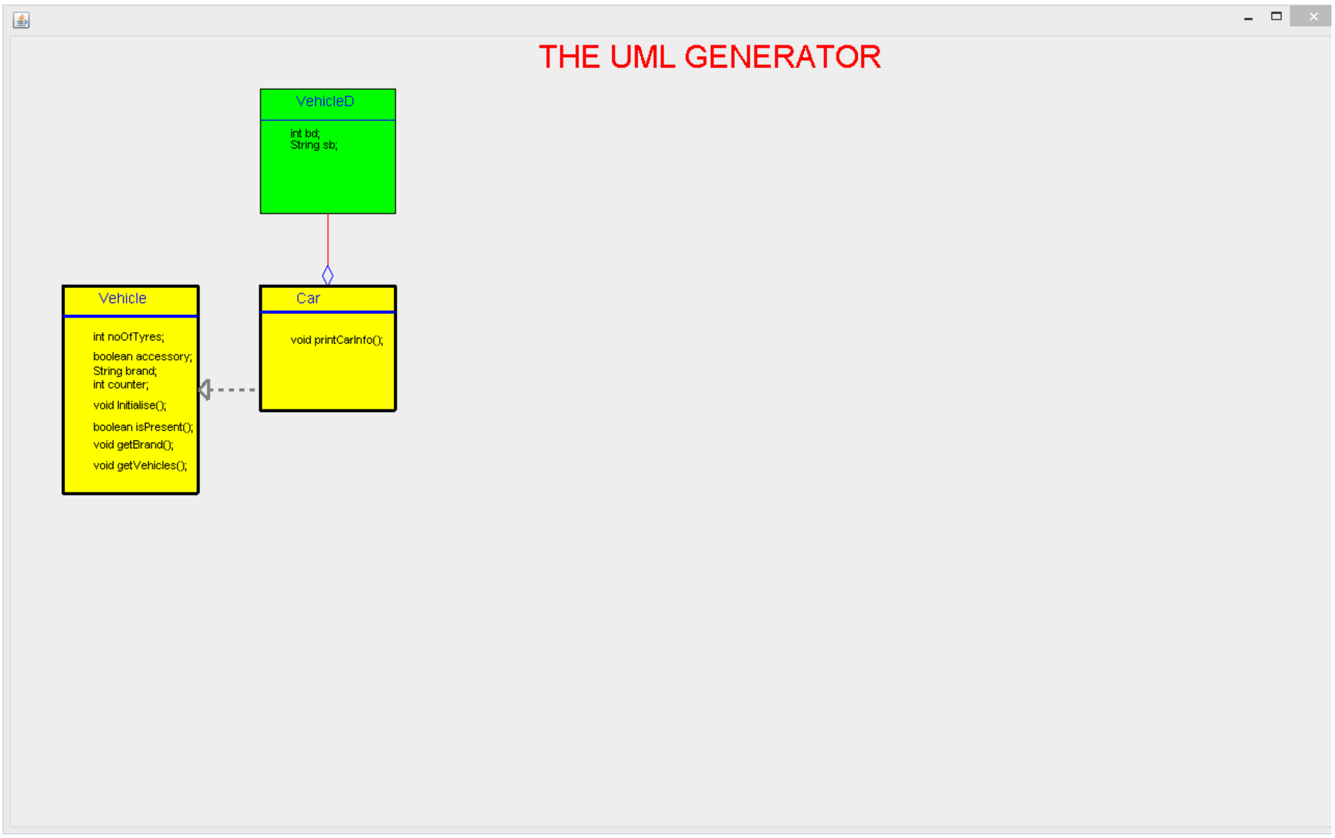
Input

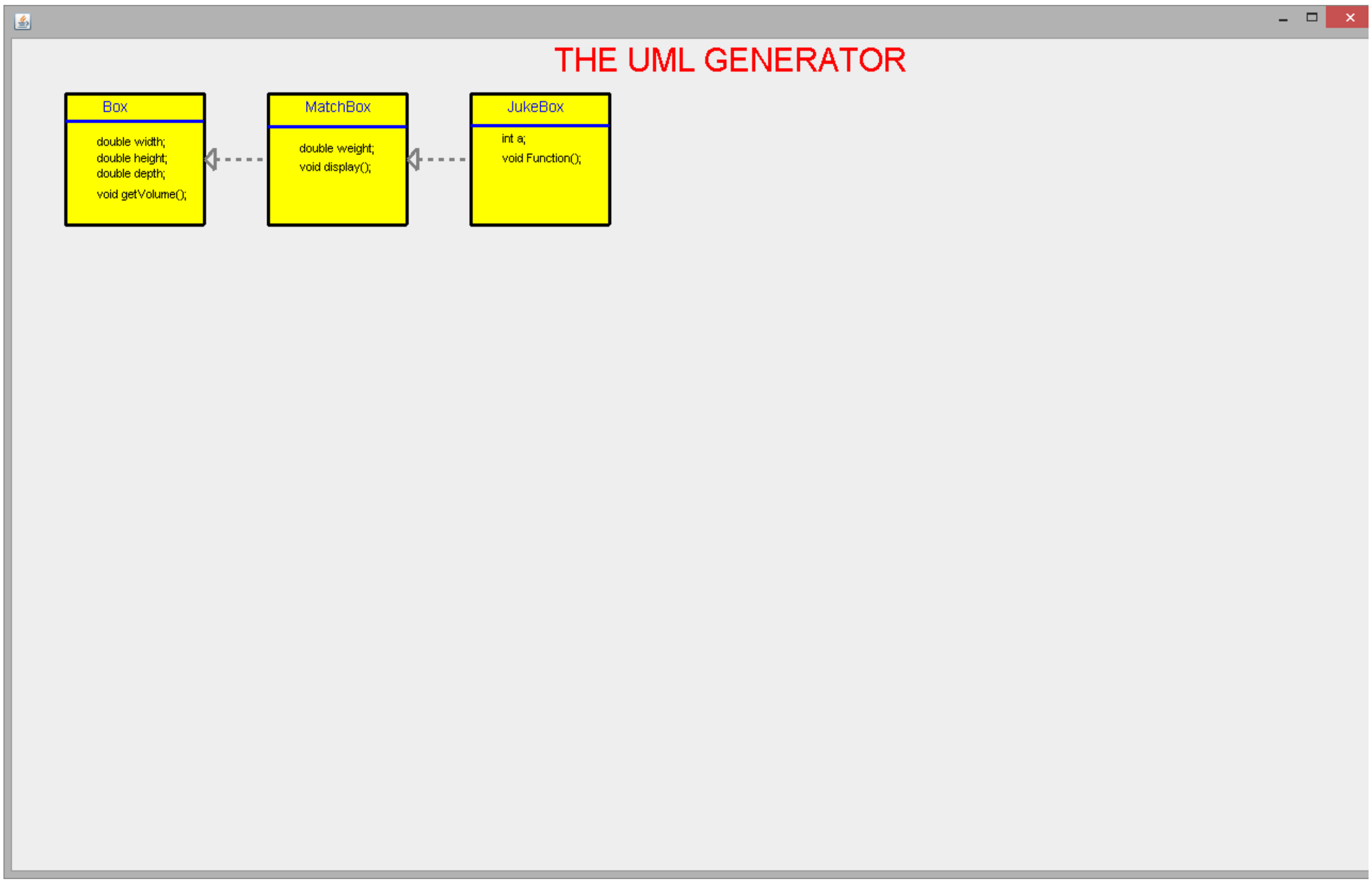
WELCOME TO THE UML GENERATOR
Enter names of Files part of Code

```
----jGRASP exec: java Client
```

Status: running user program   Line:10 Col:85 Code:69 Top:1




THE UML GENERATOR
VehicleD
int bd;
String sb;
Vehicle
int noOfTyres;
boolean accessory;
String brand;
int counter;
void Initialise();
boolean isPresent();
void getBrand();
void getVehicles();
Car
void printCarInfo();


THE UML GENERATOR
Box
double width;
double height;
double depth;
void getVolume();
MatchBox
double weight;
void display();
JukeBox
int a;
void Function();

Data Structure:

The data structure used in the project is a Multi list where each node has the following structure:

- Contains Strings which contain Class name, Attributes and behaviours of a program.
- Contains links to 2 nodes – a vertical node and a horizontal node.
- Each node in the multi-list contains the contents of a file being used in the program being executed.

The multi-list upon creation will contain those nodes linked horizontally which are related to each other either through inheritance or interface implementation. The remaining nodes are linked vertically to those nodes in which they are being used or are being referred to.

Accomplishments:

In the project we were successfully able to generate UML diagrams for programs with a simple hierarchy(inheritance) and composition. The challenges we faced while trying to implement the program were as follows:

- The linking of various nodes in the required order for the multi-list. This could be accomplished in multiple ways. However, traversal mechanisms and consequent output via a GUI for all linking mechanisms weren't as simple and often resulted in faulty outputs. The method eventually adopted for obtaining the output was to indicate the presence of a hierarchy by the detection of "Implements" or "extends" keyword in a file. If these keywords were present, linking would happen horizontally to the class name that follows.
- In the absence of keywords like "Extends" or "implements" the node would be linked vertically to the previous node that had it's reference.
- Linking of the code for the creation of the multi-list and for obtaining the output through a GUI was another challenge. We were able to produce

outputs of the two pieces of code separately and were able to verify the correctness of both. However, the two pieces of code had to be changed and modified several times to find the optimum linking mechanism that produced the right output through the traversal of the multi-list.

Conclusion:

In the course of this project we learnt several important lessons:

1. Usage of Java swing class and creation of the GUI. None of the team members were aware of how a GUI is prepared or used prior to this project. After having implemented the project, we now have a good understanding of how a GUI is to be prepared for a given application and various other features of the swing and AWT classes
2. Linking mechanisms based on keywords present in a file currently being read. As mentioned earlier, we were able to come up with several different mechanisms for the linking of the nodes. However, finding the most efficient way to do it so that the GUI could be linked to the back end properly to produce the correct output was a challenge.

Enhancements:

- The program has a limitation when it comes to depicting inheritances in the UML diagram, namely the Super class name is to be entered first and the sub class names are to be entered afterwards. A possible improvement could be to make the code work regardless of the order of the file names entered.
- The code doesn't cover a few special cases like a program containing a class within a class. Also the output when 2 completely unrelated file names are entered is not defined. The code could be improved to cover these cases as well.

Thus, the program to generate a UML diagram for a given Java source code file was implemented using a Multi list as the data structure. The code produced accurate results for simple programs involving concepts of inheritance and composition.