Volume: 12 Issue:04 Pages: 4644-4649(2021) ISSN: 0975-0290

Abstraction of UML Class Diagram from the Input Java Program

Dr. R.N. Kulkarni,
Professor and Head, Dept. of C.S.E, BITM, Ballari.
rn_kulkarni@rediffmail.com
P. Pani Rama Prasad[2]
Asst. Professor, Dept. of C.S.E, BITM, Ballari and Research Scholar, VTU Belagavi.
phanirama75@gmail.com

ABSTRACT

The Unified Modeling Language (UML) is a design tool developed by IBM and today it is most commonly used in all software industries for design of software systems. The UML comprises thirteen different diagrams out of which the class diagram is one. The class diagram represents the static behavior of the software system, which comprises the class name, attributes, operations and relationships such as generalization, Aggregation, Association, composition and Interfaces. In this paper, we are proposing a novel approach and developed the tool for the abstraction of the class diagram from the input executable java program. The abstraction of class diagram comprises steps such as restructuring the input program, identifying the reusable components and finally representing the class diagram in the form of a table.

Keywords – **Unified Modeling Language (UML), 'C' Program.**

Date of Submission: Jan 19, 2021 Date of Acceptance: Feb 18, 2021

I. INTRODUCTION

Nowadays many organizations are using java programming language for the development of software applications, because of its naturalness. It supports major features like inheritance, polymorphism, information hiding, object oriented and platform independent. These features are not supported in the procedural oriented programming languages.

Unified Modeling Language (UML) is one of the design tool kit to design any software system. These design diagrams are independent of programming language used for software development. It comprises of thirteen different diagrams which represents either static or dynamic behavior of the system. The class diagram represents the static behavior of the application system. This diagram represents the details such as class name, attributes, operations, visibility and association. Other than that the class diagram also represents generalization, interfaces, and aggregation and composition relationships among the classes present in the class diagram.

2. LITERATURE SURVEY

In paper [1], the authors proposed a methodology for abstraction of functionalities from the legacy 'C' program using the technique of program slicing. The proposed approach is appropriately modified and used in our work for the restructuring of java program.

In paper [2], authors discussed about restructuring the java program by converting multiline to the single statement, multiline statement to single statement line, Removing comment lines and blank lines. However, the author has not addressed the issues related to the files which are externally linked to the existing file using the import directive. In our proposed work, we are addressing the issues related to linking of external files.

In the paper[3], the author explained the software metrics tool benefits of extraction of UML diagrams. The software metrics tool collects the information after parsing the XML format generated by UML tool. The class diagram and its XML representation are helpful in the proposed work.

In paper [4], authors proposed tool which takes the UML class diagram as an input and represent it in a table format. This approach discussed in this paper may be helpful in representing the class components in a tabular format.

In paper [5], the author discussed about the diagrammatic description of the class diagram that includes flows of attributes, thus providing a basic representation for specifying behavior and control flow from program. The methodology proposed in the paper is appropriately modified to identify the attributes and operations from the program.

In paper [6], the authors study the metrics for UML structural and behavioral diagrams from different viewpoints, relationships, types of metric values, and their validations. This work is useful in abstraction of relationships of a class.

In paper [7], authors argue that the philosophies behind object-oriented programming are and the claims of Languages that support object-oriented ontology. Since Java is an object-oriented programming language, the topics discussed in the paper are useful in abstraction of class components from java program.

In paper [8], authors developed an application which would convert the given code into UML diagrams class diagram and package diagram and also measure coupling in java object oriented software. The process discussed in

Volume: 12 Issue:04 Pages: 4644-4649(2021) ISSN: 0975-0290

this article is useful to understand conversion process from java program to class diagram.

In paper [9], authors present an approach to automatically generate structural and behavioral code from UML class and sequence diagrams. This approach is useful in realizing the static behavior of the class diagram

In paper [10], the authors developed a tool for automatic generation of UML class diagrams from Java byte-code. To detect the class diagram elements such association multiplicities, compositions and query methods. This concept is useful in extracting class relationships from java program.

In Paper [11], authors present a discussion on problematic stages and possible element transformations into software components. Several conclusions are drawn on potential usage of the class diagram in industry. From this work we can analyze the reusable components from the class diagram.

In paper [12], the authors used UML class graph (UCG) method is used to represent the class diagram for the structural similarity between other classes. This work is useful in depicting the static features of classes inside the class diagram.

In paper [13], authors discussed about restructuring the C program by removing comment lines and blank lines converting multi-line to the single statement, multiline statement to single statement line. This approach is useful for restructuring java program.

3. METHODOLOGY

The proposed methodology to abstract the design information from the input java program is as shown in the fig 3.1.

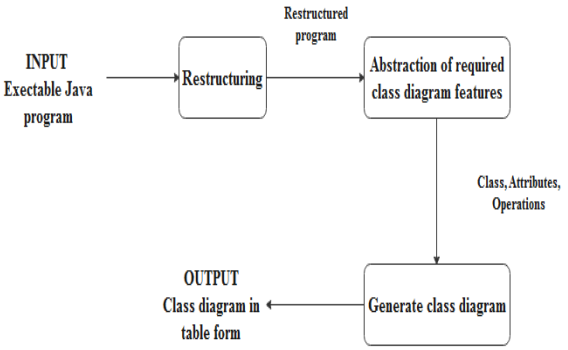


Fig3.1 Block Diagram to abstract the design information from the java program

The figure 3.1 illustrates the process of abstracting the design information from the java program. The executable java program is act as input to the restructuring process, which generates a restructured java program. From this restructured program, the class information is abstracted and represented in the in tabular form called as a class table. The class table consists of class name, attributes present in the class, operations and relationships of the class.

3.1 The block diagram contains three important stages such as:

1. Restructuring of input java program.
2. Abstraction of reusable class components from java program
3. Representation of class diagram in tabular form.

Step 1: **Restructuring of input java program**

Restructuring is a process of making changes to the program at the same abstraction level without changing its functionality. During the restructuring the following tasks are performed on the input java program [1].

It involves the steps such as

(i) Converting multi-statement line to the single statement,

(ii) Multiline statement to single statement line,

(iii) Removing comment lines and blank lines from the java program.

(iv) Inserting the line number to each executable statement in the program.

(v) Minimization of attributes used in a class.

If the java program has external linked classes imported from other packages, the corresponding classes are processed and included in the existing program [1].

Step 2: **Abstraction of reusable class components from java program**.

Each line of the restructured java program is scanned from first line to last line. This process includes six steps:-

Step 2.1: When the key word *class* is encountered in a line, identify the word next to class as a class name. The attributes can be identified, when line beings with data type read all the corresponding elements until it is encountered with semicolon. Here each element present in that line is considered as an *attribute*. If the attribute is declared inside the class and it is accessed in its operations, then such attribute is considered as valuable and hence store them in the format *<visibility type><attribute>:<data-type>* inside the class table. Otherwise eliminate the attribute and it is not part of class table. The visibility type is *private(-), public(+), protected(#) and package(~)* are considered for attributes and operations. When the scanned line is encountered with data-type operation name along with braces without semi-colon, it is to be considered as a function name. Store them as operations in the format *<data-type><operation_name>:return_type* inside the class table.

Step 2.2: Inside the class definition, when we encountered another class name in the linewith keyword *list<another class_name> object*. Then identify this relationship as Association between the classes. Store the association relationship in the format A_n*: another class_name* inside the class table, where A_n indicates *Association* and integer value of n will be in the range from 1..n.

Step 2.3: Inside the class definition, when we encountered another class name in the line with keyword

Volume: 12 Issue:04 Pages: 4644-4649(2021) ISSN: 0975-0290

private *<another class_name> object;* then identify this relationship as an Aggregation between the classes. An Aggregation relationship represents the weak assembly of the classes. Store the aggregation relationship as AG_n: *another class_name* inside the class table, where AG_n indicates *Aggregation* and integer value of n will be in the range from 1..n.

Step 2.4: In this step Inside the class definition, we encountered another class name in the line with keyword ***private final*** *<another class_name> object1* andif theobject1 is declared inside the constructor of existing class, then identify this relationship as Composition between the classes. Composition represents the strong assembly of the classes. Store the association relationship as C_n: *another class_name* inside the class table, where C_n indicates *composition* relationship and integer value of n will be in the range 1..n.

Step 2.5: When the scanned line in a program is encountered with keyword *class* along with class name and another keyword ***extends*** with sub class name in the same line. Then identify this relationship as an generalization between the classes. The generalization relationship represents the super-sub class relationship. Store the generalization relationship as G_n: *sub class name* inside the class table, where G_n indicates *generalization* relationship and integer value of n will be in the range 1..n.

Step 2.6: When the scanned line in a program is encountered with keyword ***class*** along with class name and another keyword ***implements*** with interface name in the same line, then identify it as an interface relationship. The interface relationship represents interface name with declaration of operation. Store the interface relationship as I_n: *interface name* inside the class table, where I_n indicates *Interface* relationship and integer value of n will be in the range 1..n.

Step 3: Representation of class diagram in Tabular form.

The class information is abstracted and represented in the in tabular form called as a class table. The class table consists of class name, attributes present in the class, operations and relationships of the class as shown in fig 3.2.

Class Name	Attributes	Operations	Relationship
Class name1	*<visibility type> <attribute>: <data-type>*	*<data-type> <operation name>: return type*	*Relationship: class name*

fig 3.2 class table

4. RESULTS & DISCUSSION

The methodology proposed in this paper is applied to an input java program and abstracted class components are represented in tabular form.

Class name: Department		
attribute	**project()**	**putdept()**
id	id	--
dept	dept	--
~~time~~	--	--
~~cost~~	--	--
Unused attributes		time, cost

Table 4.1 Attributes Minimization Table.

4.1. Step1: **Restructuring of input java program**
Consider the employee java program as shown in fig 4.1.

Import Employee_info.empDetails; class **Employee** { int empid; String name; **empDetails emp;** **//Association relationship** /*operations of employee class*/ private location <list> loc; **//Aggregation** void input_emp(){ loc.inlocation(); System. out. println("Employee details…"); } }	package **Employee_info;** class **empDetails** { String Address, mobileNo; void empindata(){ } void empoutdata(){ } }	Class **Location** { String addrs; Location() { final private Department <list> dept; } **//Composition** void inlocatio(){ } void outlocation(){ } }
class **salaryDetails** extends **Employee {** **//Generalization relationship** **float hra, da, gross;** void putsal(){ } void putsal(){ } **}**	Class **Department** implements **project** { int id; float cost, time; string dept; void project(){ Id=111; Cost="sales"; } void putdept(){ } }	Interface **project**{ //Interface int id; string name; void getproject(){ } Void putproject(){ } }

Fig 4.1. Employee java program

The Restructured student java program is as shown in figure 4.2. Restructuring of java program involves the steps such as

(i) Removing the blank lines and comment lines.

(ii) Converting multiple statements in a single line to a multiple statement,.

Volume: 12 Issue:04 Pages: 4644-4649(2021) ISSN: 0975-0290

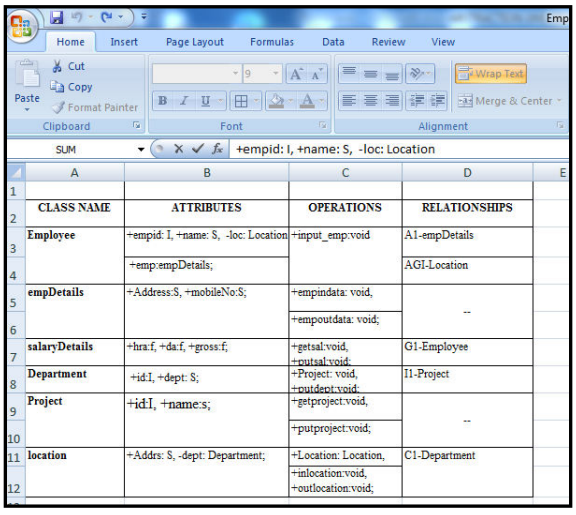
(iii) Multi line statement to a single line, (iv) Include line numbers.
(v) Process the external linked classes into existing class and
(vi) Minimization of attributes used in a class.

Import Employee_info.empDetails; 1. Clas **Employee** { 2. int empid; 3. String name; **4. empDetails emp;** private location <list> loc; 5. void input_emp(){ 6.loc.inlocation(); 7.System.out.println("Employee details…"); 8.} 9.}	package **Employee_info;** 1. class **empDetails** { 2. String Address, mobileNo; 3. void empindata(){ 4. } 5. void empoutdata(){ 6. } 7. }	1. Class **Location** { 2. String addrs; 3. Location() { 4. final private Department <list> dept; 5. } 6. Void inlocatio(){ 7. } 8. Void outlocation(){ 9. } 10.}
1. class **salaryDetails** extends **Employee {** 2. float hra, da, gross; 3. void getsal(){ 4. { 5. Void putsal(){ 6. } 7. **}**	1.Class **Department** implements **project** { 2. int id; 3.float cost, time; 4. string dept; 5.void project(){ 6. Id=111; 7. Cost="sales"; 8. } 9.void putdept() { 10. } 11.}	1. Interface **project**{ 2. int id; 3. string name; 4. void getproject(){ 5. } 6. Void putproject(){ 7. } 8. }

Fig 4.2. Restructured student java program

Table 4.2. Employee class table

In case of minimization of attributes used in a class, the attribute which is declared inside the class, must be accessed in its operations, then such attribute is considered as valuable attribute. Then only store the attribute inside the class table. Otherwise do not include the attribute, inside the class table. In this method, unused attributes are eliminated from the class table. Consider the class Department in the given java program as shown in fig 4.1. The attribute minimization table is as shown table 4.1. The class Table for department class will store attributes id and dept



	A	B	C	D
1				
2	CLASS NAME	ATTRIBUTES	OPERATIONS	RELATIONSHIPS
3	Employee	+empid: I, +name: S, -loc: Location	+input_emp:void	A1-empDetails
4		+emp:empDetails;		AGI-Location
5	empDetails	+Address:S, +mobileNo:S;	+empindata: void,	--
6			+empoutdata: void;	
7	salaryDetails	+hra:f, +da:f, +gross:f;	+getsal:void, +putsal:void;	G1-Employee
8	Department	+id:I, +dept: S;	+Project: void, +putdept:void;	I1-Project
9	Project	+id:I, +name:s;	+getproject:void,	--
10			+putproject:void;	
11	location	+Addrs: S, -dept: Department;	+Location: Location,	C1-Department
12			+inlocation:void, +outlocation:void;	

Volume: 12 Issue:04 Pages: 4644-4649(2021) ISSN: 0975-0290

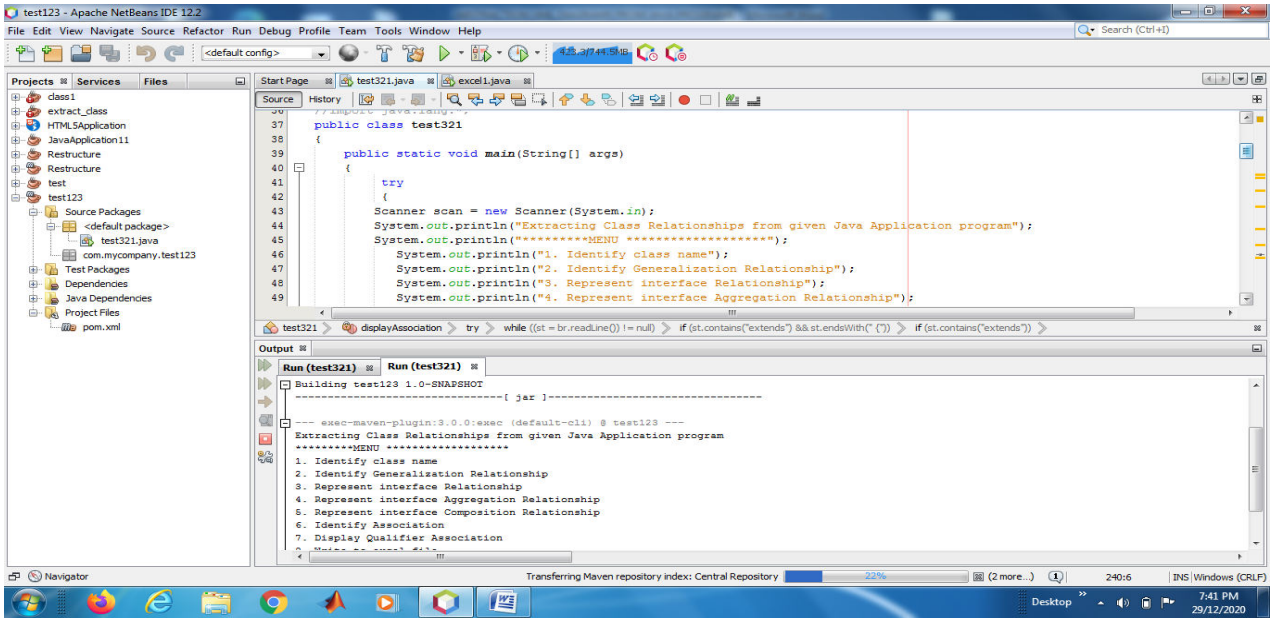


Fig 4.3. Output Screen shot of the tool

ii. Identification of Interface: Project is an interface which is implemented inside an **Employee** class. The interface relationship is represented in table 4.2.

iii. Identification of Association, Aggregation and Composition Relationships

4.2. Step 2: Abstraction of class components and storing in a class table

i. Identification of generalization: Represents relationship between super class and sub classes. **Employee** is the super class and **salaryDetails** is sub class. The generalization relationship is represented in table 4.2.

From the given java program as shown in fig 4.1, the following relationships are identified.

a. Employee is the base class and **empDetails** is the **associated class**.

b. Employee is the base class and **location** is an **aggregate class**.

c. Location is the base class and **Department** is the **composition class**.

All the three relationships are represented in table 4.2. The output screen shot is as shown is as shown in fig 4.3

4.3 Step 3: Representation of class diagram in Tabular form.

The class information is abstracted from the java program and represented in the tabular form called as a class table. The class table is an excel file and it is shown in table 4.2.

All the three relationships are represented in table 4.2. The output screen shot is as shown is as shown in fig 4.3

In this paper we have presented an automated tool to abstract the class diagram from the input executable java program. The proposed tool performs restructuring of input executable java program and the abstracts the required components from the restructured program and finally represents the class diagram in a tabular form. The tool is tested for its completeness and correctness as an input up to 500 lines of code by giving different java programs.

6. REFERENCES

[1] Dr. R.N. Kulkarni, Padmapriya Patil, , “Abstraction of Functional Modules from a Legacy ‘C’Program using Program Slicing”, Perspectives in Communication, Embedded-systems and Signal-processing-PiCES-2020, 4(4),p-39-44 , August 2020.

[2] Dr. R.N. Kulkarni, P. Pani Rama Prasad, “Restructuring of Java Program to be amenable for Reengineering”, Journal of Engineering Science and Technology, Vol 02(06), May 2019.

[3] Daljeet Singh, “A scrutiny study of various unified modeling language (UML) diagrams, software metrics tool and program slicing technique”, Journal of Emerging Technologies and Innovative Research (JETIR). Vol 5(6), June 2018.

[4] Dr. R N Kulkarni, C K Srinivasa, “An Ameliorated Approach to Represent UML Class Diagram in the Table Format”, International Journal of Computer Applications 182(7):5-9, August 2018.

[5] Sabah Al-Fedaghi, “Diagramming the Class Diagram: Toward a Unified Modeling Methodology”, International Journal of Computer Science and Information Security(IJCSIS), Vol. 15(9), September 2017.

[6] Dr. Amit Kamra, "Measuring Software Design Metrics of UML Structural and Behavioural Diagrams", ICRTESM-2017 at IETE, Erandwane, Pune, Maharashtra, May 2017.

[7] Justin Joque, "The Invention of the Object: Object Orientation and the Philosophical Development of Programming Languages", DOI 10.1007/s13347-016-0223-5, Springer Science+Business Media Dordrecht, 2016, P-335–356.

[8] Shubhangi Sakore1 , Ravina Kudale, "Tool for Converting Source Code to UML Diagrams & Measuring Object Oriented Metrics in OO Java Software", International Journal of Science and Research (IJSR) Vol 5(4), April 2016, p-1797-1800

[9] Abilio G. Parada, Eliane Siegert, Lisane B. de Brisolara, "Generating Java code from UML Class and Sequence Diagrams", DOI: 10.1109/SBESC.2011.22, IEEE Explore, December 2014

[10] Martin Keschenau, "Reverse Engineering of UML Specifications from Java Programs", OOPSLA'14, Oct. 2428, 2014, Vancouver, British Columbia, Canada. ACM 1581138334/ 04 /0010. October 2014

[11] Oksana Nikiforova1, Janis Sejans, Antons Cernickins, "Role of UML Class Diagram in Object-Oriented Software Development", Scientific Journal of Riga Technical University Computer Science. Applied Computer Systems, DOI: 10.2478/v10143-011-0023-4, vol(4) 44, 2011.

[12] Zhongchen Yuan, · Li Yan, Zongmin Ma, "Structural similarity measure between UML class diagrams based on UCG", Vol(25) Requirement Engineering, Springer Nature 2019, p 213–229.

[13] R.N.Kulkarni and Shivanand M. Handigund, "Moulding The Legacy C Programs For Reengineering", International Conference on "Advances in Computer Vision and Information Technology (ACVIT -07)", Aurangabad, India, November, 2007, p1531-1537.

[14] Rajkumar N Kulkarni, Padmapriya Patil, "Restructuring of Legacy 'C'Program to be Amenable for Multicore Architecture", ICRTEST Elsevier energy procedia proceedings-2016, issue-1, p 201-207.

[15] Shaik Ismail, Sai Koparthi, "Automatically restructuring of java comments", Published in conference'17, Washington D.C, U.S.A@2017 ACM PP-1-4, july 2017.

[16] Core and Advanced Java Black Book", edition 2017, Dreamtech press, ISBN:978-93-5119-940-3.

[17] Mubarak Albarka Umar, Chen Zhanfang. A Comparative study of Dynamic Software Testing Techniques. Int. J. Advanced Networking and Applications Volume: 12 Issue: 03 Pages: 4575-4584(2020) ISSN: 0975-0290.

[18] Dr Vipin Saxena, Deepa Raj. Local Area Network Performance Using UML. Int. J. of Advanced Networking and Applications Volume: 02, Issue: 02, Pages:614-620 (2010).