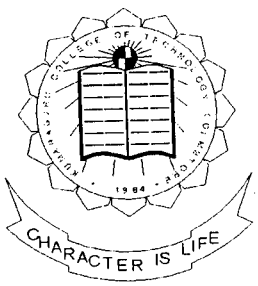
Code Generator

P-494

Project Report



Submitted By
Jayaram .V
Siva Kumaran .J
Siva Kumar .K
Somu .L

2000 - 2001

Under the guidance of
Mrs. D.Chandrakala , M.E,

In partial fulfillment of the requirements for the award of the degree of Bachelor of Engineering (**B.E**) in Computer Science and Engineering of the Bharathiar University, Coimbatore.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE – 641 006.

Department of Computer Science and Engineering
KUMARAGURU COLLEGE OF TECHNOLOGY
Coimbatore-641 006.

CERTIFICATE

This is to certify that the Project Report entitled

CODE GENERATOR

Has been submitted by

Mr. L. Somu, K. Sivakumar, J. Sivakumaran, U. Jayaram.

In partial fulfillment of the requirements for the award of Degree in Bachelor of Engineering in Computer Science and Engineering of the Bharathiar University, Coimbatore-641006 during the year 2000-2001.

D. Chandru 10/3/2001
(GUIDE)

S. Thangasamy 10/3/01
(HEAD OF DEPARTMENT)

Certified that the candidate was examined by us in the Project Viva-Voce Examination held on 12-03-2001 and the University Register Number is .....

S. Duraisamy 12/3/01
(INTERNAL EXAMINER)

(EXTERNAL EXAMINER)

DECLARATION

We, J. SivaKumaran, K. Sivakumar, U. Jayaram and L. Somu here by declare that this project work entitled "Code Generator" submitted to Kumaraguru College of Technology, Coimbatore (Affiliated to Bharathiar University) is a record of original work done by us under the supervision and guidance of Ms.Chandrakala M.E Department of Computer Science and Engineering.

Name of The Candidate	Register Number	Signature
1. U. Jayaram	9727K0146	
2. K. Sivakumar	9727K0178	
3. J. Sivakumaran	9727K0179	
4. L. Somu	9727K0180	

Countersigned: Staff in Charge

Mrs.D.Chandrakala M.E

Lecturer

Department of Computer Science and Engineering

Kumaraguru College of Technology

Coimbatore – 641 006

Place : Coimbatore

Date :

ACKNOWLEDGEMENT

We express our deep sense of gratitude to **Dr. K.K. Padmanabhan .Ph.D.** , Principal , Kumaraguru College of Technology , Coimbatore , for providing permission to carry out this project work.

With profound sense of gratitude and regards , we acknowledge with great pleasure the guidance and support extended by **Prof. S.Thangasamy Ph.D.** , Head of the Department of Computer Science and Engineering , for his valuable and continuous guidance , suggestions , constructive criticisms and persistent encouragement.

We express our deep sense of respectful gratitude to our guide **Mrs. D.Chandrakala M.E., Lecturer,** Department of Computer Science and Engineering and **Ms . A. Lavanya B.E** , Lecturer , Department of Computer Science and Engineering for their valuable guidance , keen suggestions , innovative ideas , inspiration , discussions , helpful criticisms and kind encouragement in all the phases of this project work . It had been indeed a great pleasure to work under her guidance.

It is our duty to express our thanks to **Mr.V.Krishnan , Project Manager , Atna Technologies India Pvt Ltd.** , Coimbatore & all IBM ACE faculties for guiding us to do this project work and his kind encouragement during the course of the project.

We also express our sincere thanks to **Mr. Shoban & Mr. Julian, Project manager ,Think Business Network Pvt Ltd.**, Coimbatore for their support.

We like to express our special thanks to all staffs and lab technicians in the Department of Computer Science and Engineering who helped us for the successful completion of the project.

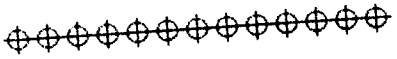
Finally we express our deep sense of gratitude to our parents, friends and all other persons who directly or indirectly involved with this project , for their invaluable help and consideration towards us.

Synopsis

Code generator is a software that can be used by team members, analyst, developers, project managers and chief information officer. End users can visualize the class diagrams and understand the interactions, they will make with the system from looking at the model. Analyst can visualize the interactions between objects from the model. Developers can visualize the objects that need to be developed and what each one needs to accomplish. Project managers can see the whole system and how the parts interact. Chief information officers can look at high-level models and see how system in their organization interacts with one another.

In the case of code generator, the project leader will provide the skeletal code to the team members. The team members will do the necessary coding by looking at the comments given by the leader. This can be greatly used in the software industry. Since the hierarchy of the entire project is generated. It greatly helps in further development of the project with ease.

CONTENTS



INTRODUCTION

1.1.Existing System

The following are the Existing Systems that are similar to Code Generator.

- "Structure Builder " [Version 4.0, Build 385], http://www.webgain.com ,
  WebGain Inc.
  5425 Stevens Creek Blvd
  Santa Clara, CA 95051
- "Rational Rose" , http://www.Rational.com .
- "Visual Age for Professional Java ", IBM , http://www.ibm.com .

Structure Builder:

This is a product developed by web gain Inc. This includes various features like EJB Module, Persistence Builder, Deployment Builder, Explorer View, Sub-projects, MANY other small enhancements and bug fixes.

Here in order to generate a skeletal code the user has to fill in the meta data properties related to the class. All the classes & interfaces contained in the project come under a single project name. Once the details are filled it gets automatically saved in the respective Java file .On completion of the entire project the hierarchy relation can be obtained by clicking the tab pane. This product supports dynamic change of coding when the hierarchy diagram relationship is changed.

Rational Rose:

This product supports both engineering and reverse engineering concepts.This products takes care of all UML diagrams.It provides support for object languages like C++,java,PowerBuilder ..,

Visual Age:

This product Developed by IBM , generates the code in a similar fashion as that of structure builde.But its exection speed is faster compared to it.It also has some added features.

1.2.Proposed System

This project "Code Generator" produces skeletal code and project hierarchy in different files.Here it prompts the user to create a project with different class and interfaces on it.Latter the meta data property for each class is obtained and saved in the respective files.. Special feature that is additionally implemented in this project is dynamically adding of field and user defined method names to the class /interface as child in the tree structure. Here all kinds of validations are done.

for example : if other than true / false is typed as initial value for Boolean it displays a message saying "Boolean can have values only true or false". Similarly validations are done for access specifies .The entire hierarchy of the project is presented in the text format on clicking the hierarchy button.This detail is also saved in a file.

System Requirements

2.1 Hardware Requirements

Processor : Pentium III 600 MHz.
Memory : 4.3 GB.
RAM :128 MB.
Floppy Disk Drive.

2.2 Software Requirements

Java 2.0.

2.3 Language Description

Introduction to Java 2.0

Java is a language for programming on the Internet developed by Sun Microsystems. It incorporates OOPS, multi threading and is platform independent. It is designed to be small, simple and portable across both platforms as well as operating systems at source as well as binary level. The popularity of java is due to its design mainly based on the combination of three key elements.

- Applets,
- Powerful Programming Language constructs (Application) and
- Significant object classes.

What's New in JDK 1.2

JDK 1.2 comes with a new set of packages JFC (Java Foundation Classes) that includes an improved user interface called *SWING* components. Swing includes many more components than those in AWT of

the first version of JDK so that sophisticated interfaces could be built. The Swing components are the JFC's lightweight user interface components.

Quick Tour Of AWT

Abstract Window Toolkit (AWT) is a package of classes that help in creating full-fledged windowing applications and applets. The component class is the base class of all user interface components and containers, which provides the basic functionality needed for display and event handling. The containers are generic AWT components that can contain other components including other containers.

It has several subsystems that support the development of GUI. Those subsystems include:

- Graphical primitives that allows the drawing and rendering of lines and images.
- Components such as Labels,Buttons and TextFields.
- Containers that include Frames,Panels and Dialogs.

- Layout Managers that control the display in a portable manner.
    - Flow Layout.
    - Border Layout.
    - Grid Layout.
    - Grid Bag Layout.

- Event system, which allows the users to respond the interactions between the components and containers in the application.

Layout manager classes are set of classes that help in arranging the user interface components in a container.

JFC at a Glance

JDK 1.2 was introduced with a new set of packages – Java Foundation Classes (JFC). – That includes an improved user interface called the swing components.

JFC were developed, to address the shortcomings of AWT. The development of JFC was unique. JFC 1.2 is an extension of the AWT, not a replacement for it. The JFC visual components extend the AWT container class. The methods contained in the Component and Container classes the AWT programmers are familiar with are still valid for JFC visual classes.

The AWT user interface classes are now superseded by classes provided by the JFC . The support classes play an important part in JFC applications. AWT support classes, those that do not create a native window are not replaced by JFC classes.

Sailing Through Swing

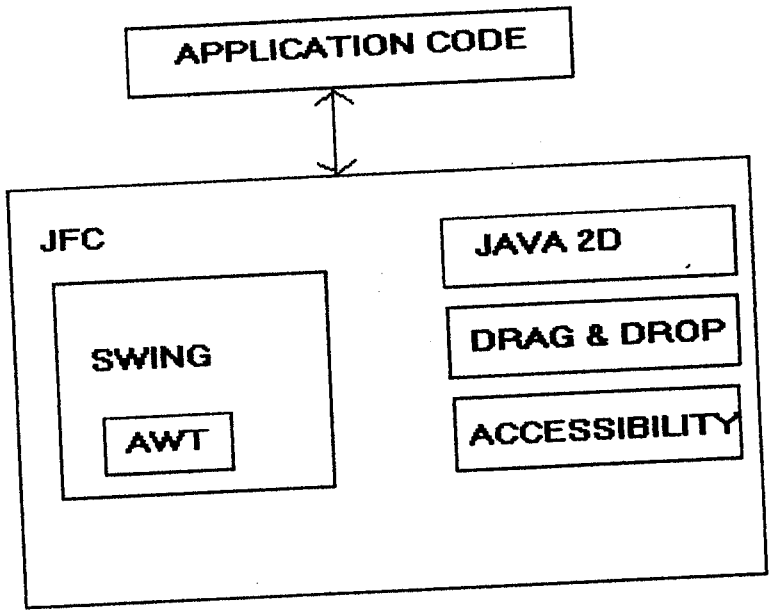
Swing components facilitate efficient graphical user interface (GUI) development. These components are a collection of lightweight

visual components. Swing components contain a replacement for the heavyweight AWT components as well as complex user-interface components such as trees and tables.

Swing components contain a plug gable look and feel (PL & F). This allows all applications to run with the native look and feel on different platforms. PL & F applications to have the same behavior on various platforms. JFC contains operating system neutral look and feel. Swing components do not contain peers. Swing components allow mixing AWT heavyweight and Swing lightweight components in an application.

The major difference between lightweight and heavyweight components is that the former can have transparent pixels while the later are always opaque. Lightweight components can be non-rectangular while heavyweight components are always rectangular.

Swing components are JavaBean compliant. This allows components to be used easily in a Bean aware application-building program. The root of the majority of the Swing hierarchy Swing architecture is shown in the figure given below.



Swing components comprise of a large percentage of the JFC release. The Swing component toolkit consists of over 250 pure Java Classes and 75 interfaces contained in about 10 packages. They are used to built lightweight user interfaces. Swing consists of user interfaces (UI) classes and non-user interface classes. The non-UI classes display something on the screen but provide services and other operations for the UI classes.

Swing Surmounts AWT

Swing offers a number of advantages than AWT.

1. Wide variety of Components
2. Plug gable Look and Feel
3. MVC Architecture
4. Keystroke Handling

5. Action Objects
6. Nested Containers
7. Virtual Desktops
8. Compound Borders
9. Customized Dialogs
10. Standard Dialog Classes
11. Standard Table and Tree Components
12. Powerful Text Manipulation
13. Generic Undo Capabilities
14. Accessibility Support etc…
15. Double Buffering.
16. Scrolling Support.

Wide variety of Components

Class names that start with 'J' are the components that are added to an application. Examples include JButton, JList, JPanel etc The remaining files in the swing package contain the utility classes and interfaces that the components use to function.

Action Objects

Action-interface objects provide a single point of control for program actions. An example of this would be a toolbar icon and menu item referencing the same Action Object. When the Action object is disabled, the GUI items that reference it are automatically disabled. The Action interface extends ActionListener, specifying an

enabled property as well as properties for text-description and graphic icons.

Nested Containers

Swing was designed to manage nested containers gracefully. The main 'heavyweight' containers (JWindow, JFrame, JDialog and JApplet) as well as the major 'light weight' containers (JInternalFrame and JComponent) all delegate their operations to a JRootPane. This commonality produces a high degree of regularity in container nesting. In particular, since the fundamental component class (JComponent) contains a JRootPane, virtually any component can be nested within another. It means, for example, that a graphic can be nested in a list, and a combo box can be nested in a tool bar. The JRootPane class uses a JLayeredPane to manage a content pane and an optional menu bar in a way that is virtually transparent to the user.

Customized Dialogs

The JOptionPane class provides a variety of static methods that the user can invoke to create and display both message dialogs and user-choice dialogs in a variety of formats. The 'message' displayed in the dialog can be a string, a string-generating object, or an arbitrary component. Choice buttons can be replaced with components that are specified for user selections.

Generic Undo capabilities

The Undo package provides generic undo capabilities that can be used in a variety of situations.

Main classes used in this project

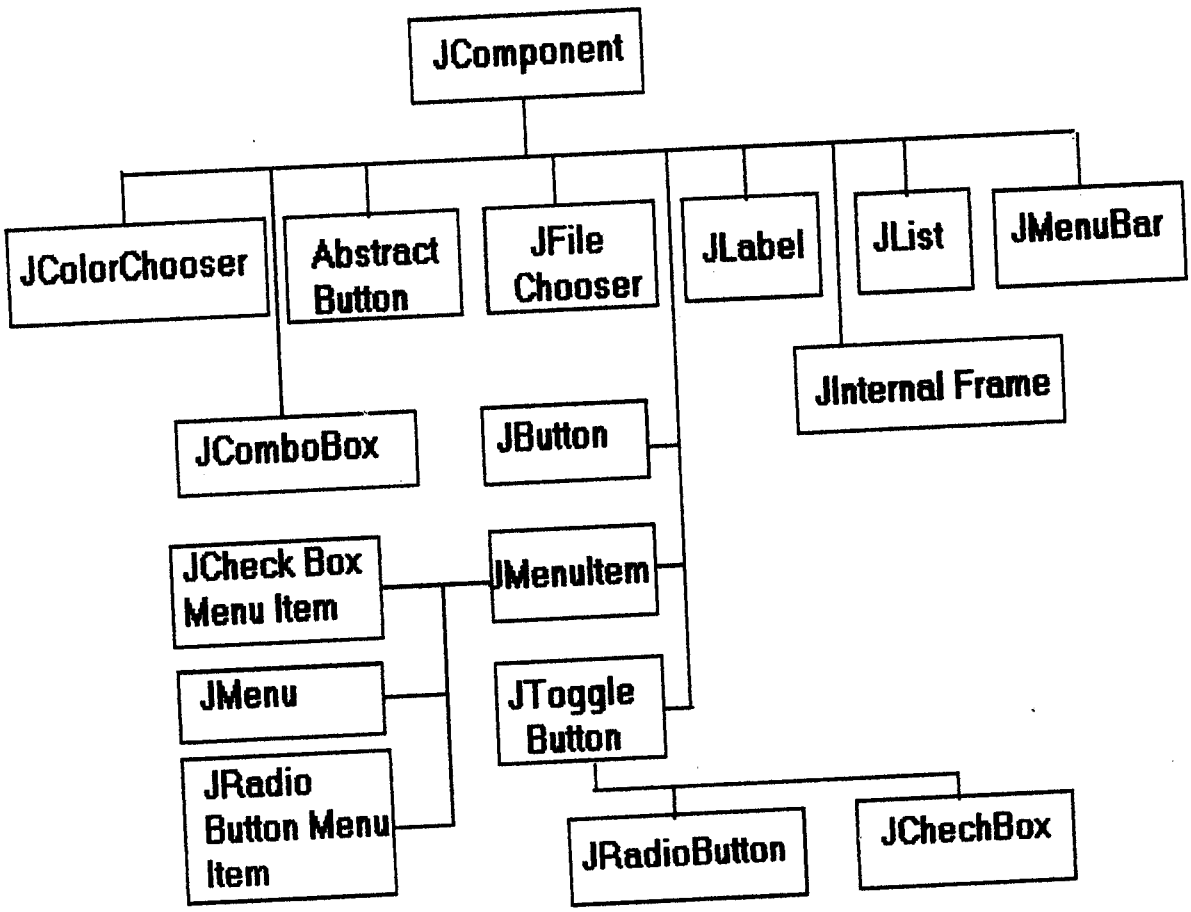
JPanel

JPanel is a generic lightweight container. The primary purpose of the class is to provide a concrete container for the JFC. The JPanel class is provided to give a concrete container class. Being an extension of the JComponent class, JPanel is a container and inherits the features contained in that class.

JComponent

The JComponent class is the root of the visual component class hierarchy in the JFC. The visual components are known as the "J" classes. The functionality contained in the JComponent class is available to all the visual components contained in the JFC. The JComponent class is a repository of functionality for all visual components.

The JComponent class is at the top of the hierarchy of all visual components contained in the JFC. The hierarchy is shown :



JTabbedPane

A component which lets the user switch between a group of components by clicking on a tab with a given title and/or icon is JTabbedPane.

JToolBar

JToolBar provides a component, which is useful for display commonly, used Actions or controls. The user can drag it out into a separate window. By default, the JToolBar uses the BoxLayout. JToolBars generally contain icon only JButtons or JToggleButton.

JScrollPane

JScrollPane is a specialized container that manages a view port, optional vertical and horizontal scrollbars and optional row and column heading view ports. The JViewPort provides a window or 'view port' onto a data source – for example text file. That data source is the 'scrollable client' displayed by the JViewport view. A JScrollPane basically consists of JscrollBars, JViewPort and the wiring between them. By default, the cornets are empty. A Component can be placed into a corner. The size of cornet components is entirely determined by the size of the headers and scroll bars that surround them. It is typically useful for scrolling purpose. It is an extension of JComponent. The JViewPort class manages components that have to be scrolled. The JScrollPane class manages a single viewport in the center of its area.

Other Components Available in Swing

- ✓ **JButton**
- ✓ **JCheckBox**
- ✓ **JRadioButton**
- ✓ **JTextArea**
- ✓ **JTextComponent**
- ✓ **JTextField**
- ✓ **JLabel**
- ✓ **JMenuBar**
- ✓ **JMenu**
- ✓ **JPopupMenu**
- ✓ **JMenuItem**
- ✓ **JCheckBoxMenuItem**
- ✓ **JRadioButtonMenuItem**
- ✓ **JComboBox**
- ✓ **JList**
- ✓ **JOptionPane**

System Design and Development

This chapter explains the Design and development of the system. It tells about how the inputs are given, how the given inputs are collected and finally how they are displayed are given in detail in this chapter.

3.1 Over View of the system

Code Generator is Software using which the skeletal code of the project is obtained. In companies the synopsis is given to the project head and the project head divides analysis the project then prepares the SRS and finally divides them into modules and these modules are given to team members for writing codes. With the help of code generator the project head can generate an outline skeletal code with necessary comments, which can be given to the members for further development.

Code generator mainly consists of two parts.

1. The first main page
2. The Property page

3.1.1 First Page

The first page consists of a menu bar with some menus, some buttons and a text area, which is, used for displaying the skeletal code and also the hierarchy. The first page is opened default with a tree structure starting with project. If any class has to be added it can be added below the project as a child. We can also add child to the child of project and the hierarchy can be extended. The child can be added by just clicking a button at the bottom.

3.1.1.1 Menu Bar

Menu bar consist of two drop down menus.

1. File
2. Edit

File menu consist of open close and Exit.

Open:

When the open option is clicked a file dialog box will be opened. Using the file dialogue box the respective file can be selected. If the open button in file dialogue box is clicked the particular selected file will be displayed in the text area, which is available at the bottom of the main page.

Save:

Similarly we have another option called save. Save is used to save the content in the text area in a file. The finally generated skeletal code, which is displayed in the text area, has to be saved. So for that save option is used. When save is clicked a file dialog box again opens and after entering the file name and if the save button is clicked the contents of the text area is stored under that particular file name.

Exit:

This exit option is to end the application. As soon as the exit button is clicked the application gets terminated.

Edit menu consist of cut, copy and paste. What ever in the text area can be edited using this Edit menu. Particular portion of the coding in the text area

can be highlighted and it can be either deleted or copied or it can be pasted. Any editing operation can be performed using this menu.

3.1.1.2 Hierarchy Button

This button when clicked the entire hierarchy is obtained in the text area. The hierarchy in the sense what are all the classes available, from which are all classes it is extending and which are all classes it is implementing etc. All these details can be obtained using the hierarchy button.

3.1.1.3 Add Child

The add sibling button is used to add child to a selected parent. After a particular parent is clicked and when this button is clicked a dialog box opens which asks for the name for the child. After giving the name if ok button is clicked a child is created with the specified name under the parent.

3.1.1.4 Delete

The Delete button is used to delete a child in the tree. The tree structure has lot of child. The child can be easily deleted by just highlighting the class and by just clicking this button the child gets automatically deleted.

3.1.1.5 Meta Properties

This button on clicking after a particular class is selected opens a new window, which is the property window where the details about the class are given. After all the details are entered the data's are carried to the main page and is displayed in a formatted manner.

3.1.1.6 Text Area

This is the place where the new files are opened or the skeletal code is generated and displayed. From this place the codes are saved. The hierarchy also displayed in this text area.

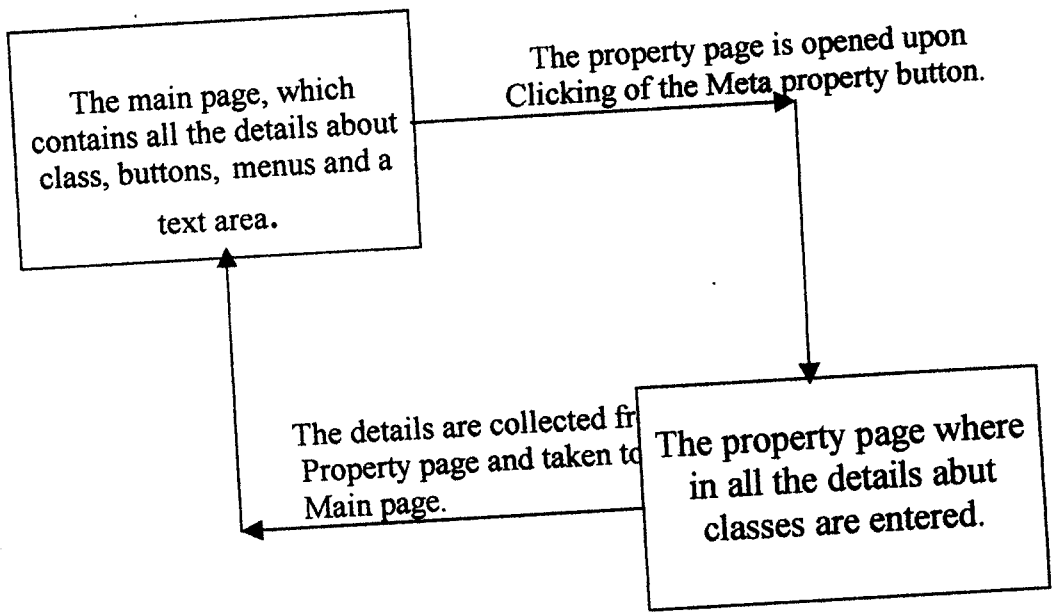
3.1.2 Property Page

The property page is the place where all the details about the class are given. A JTabbed Pane is used to display the properties in order. The details, which are given in this page, are listed below.

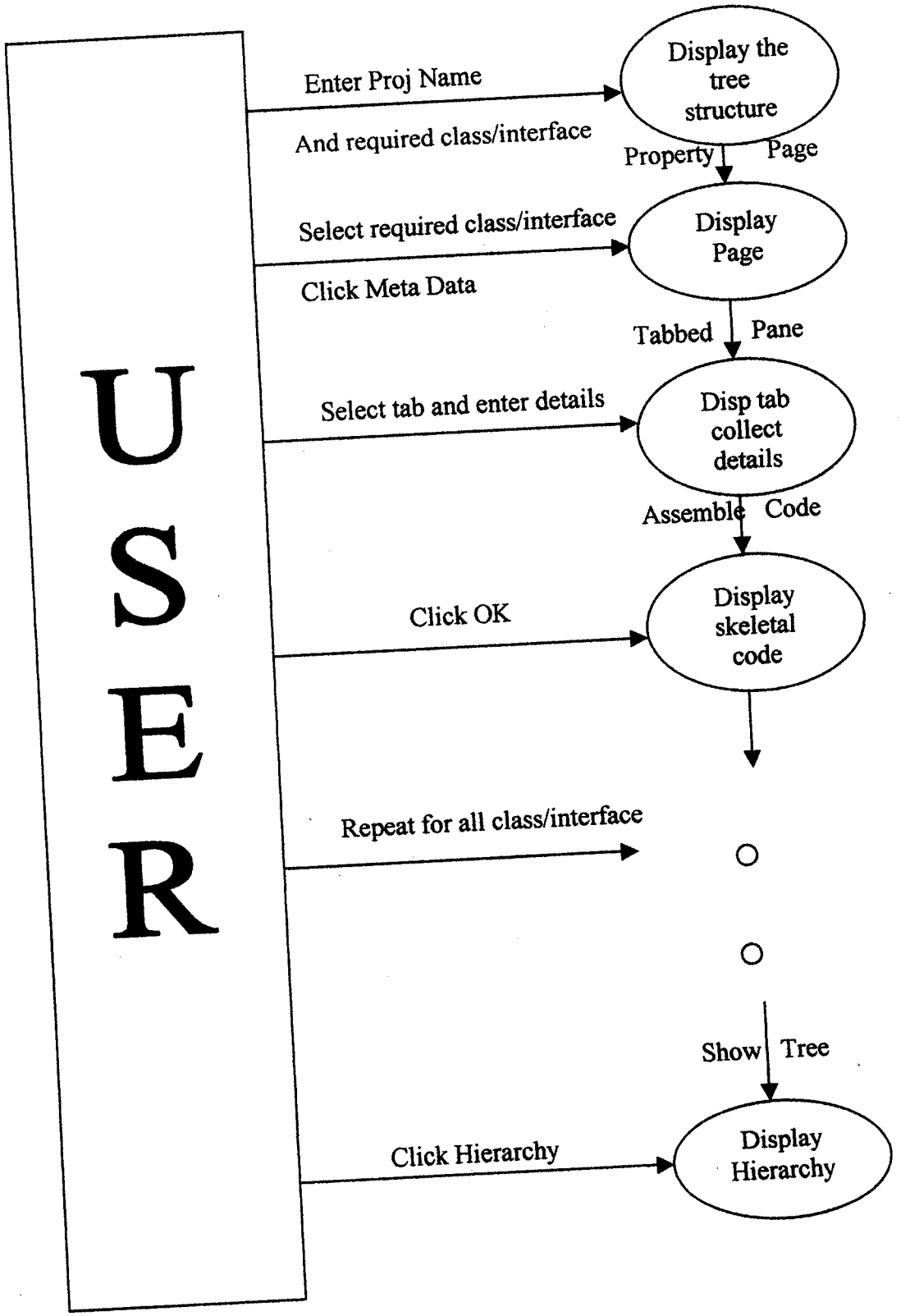
1. Class details which includes
    a. Name
    b. Extends which class
    c. Implements which listener
2. Variables which includes
    a. Type
    b. Name
    c. Initial value
    d. Comments
3. Function which includes
    a. Name
    b. Return type
    c. Arguments

d. Whether it is a user defined function or interface or main method

4. Comments in which general comment about the class is given
5. An 'OK' button when clicked all the details is collected and displayed in a particular format in the main page.



DATA FLOW DIAGRAM


USER
Enter Proj Name
And required class/interface
Display the tree structure
Property
Page
Select required class/interface
Click Meta Data
Display Page
Tabbed
Pane
Select tab and enter details
Disp tab collect details
Assemble
Code
Click OK
Display skeletal code
Repeat for all class/interface
Show
Tree
Click Hierarchy
Display Hierarchy

Implementation

Code generator a powerful tool for getting the skeletal code had been implemented using SWING. The main reason for going to swing is because of the JTabbed Pane used for the property page and also the JTree. Just by clicking the tabs the property page can be changed.

The main method is contained in codegen.java file. This is the main file, which is runned to start the application. This file consist of the inner classes Show, which basically does the job of displaying the Meta properties page. It does the process by calling the constructor of the Meta properties file. The following are the dot class files generated on compilation of the file.

1. Codegen.class
2. Codegen$1.class
3. Codegen$2.class
4. Codegen$3.class
5. Codegen$4.class
6. Codegen$5.class
7. Codegen$6.class
8. Codegen$7.class
9. Codegen$8.class
10. Codegen$show.class

The codegen$1-8.class files are generated due to the various action events used in the program. The object of the Meta file is received in the codegen in

order to display the skeletal code in the text area that is present in the codegen file.

The second file named the meta.java contains the input area in which the details needed to generate the skeletal code. JTabbed Pane is used in this in order to collect the Meta details in easier manner. It consists of the classes.

1. Clspanel
2. Medpanel
3. Varpanel
4. Hipanel
5. Companel
6. Okpanel

Clspanel:

Clspanel class is the place where the details about the class such as class/interface, access specifiers, package information etc are collected.

MedPanel:

This is the place where the method details such as name of the method, access specifiers, exceptions, parameters etc are collected.

Varpanel:

Here in this class the details about the variable such as initial value, comment etc are collected.

Hipanel:

In this the details related to the extends and implements are collected.

Companel:

Companel is the place where the details about the overall comments about the project are given and also the details needed for documentations are given.

Okpanel:

Okpanel consist of a single button which on clicked generates the entire code of the project.

On compilation of this java file generates 22 dot class files as a result of various events in the program.

Thus the project Code Generator was completed successfully. Now this can be used even by the beginner for the development of skeletal code. The skeletal code generated by this will be free from errors as it handles various type of validation. This will also be of great useful for further development of the code as the hierarchy of the entire project will be saved in a file. At a single glance the entire project details including the method names ,field names & hierarchy can be obtained.

Future enhancement :

- Can be used to develop skeletal code for other object oriented languages.
- Produce Visual representation of the hierarchy between the class/interfaces.
- Add advanced features like cardinal links info ,link labels .
- Provide facilities for printing, undo,properties,different color to indicate variable name,identifiers .
- Provide facilities for implementing inner classes, filling the body of the method .
- Support for dynamic change of code ,when the class diagram is manipulated .

SAMPLE SOURCE CODE

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.tree.*;
import java.io.*;

/**
 * @version 2.0 03/03/2001
 * @author  Jayaram.U
 * @author  SivaKumar.K
 * @author  SivaKumaran.J
 * @author  Somu.L
 */


public class codegen implements ActionListener
  {
    JFrame frame ;
    String input=null;
    protected DefaultTreeModel treeModel;
    private meta m = new meta(this) ;
  /** This constructor is used to build the tree structure,Buttons,file menu & Text area */
    public codegen()
    {
      frame = new JFrame("Code Generator Tree View ");
      JPanel panel = new JPanel(true);

      JMenuBar menuBar = constructMenuBar();
      frame.setJMenuBar(menuBar);
      frame.getContentPane().add("Center",panel);

      frame.addWindowListener(new WindowAdapter()
      {
       public void windowClosing(WindowEvent e)
       {
        File f = new File("e:\\97cse54\\newfol~1\\details.txt");
        f.delete();
        System.exit(0);
       }
      });

      frame.setVisible(true);
      frame.pack();
      frame.show();
      frame.setSize(800,700);
```

```
// construct tree

TreeNode root = makeSampleTree();
model = new DefaultTreeModel(root);
tree = new JTree(model);
tree.setEditable(true);

// add scroll pane with tree to content pane

JScrollPane scrollPane = new JScrollPane(tree);
panel.setLayout(new BorderLayout());
panel.add("Center", scrollPane);
panel.add("South",constructOptionsPanel() );
panel.setVisible(true);

 // make button panel

}

/** This function contains two panels . One for controls & other for text area*/
   private  JPanel constructOptionsPanel()
     {
     JPanel retPanel = new JPanel(false);
     JPanel borderPane = new JPanel(false);
     borderPane.setLayout(new BorderLayout());
     retPanel.setLayout(new FlowLayout());

    addSiblingButton = new JButton(" Hierarchy ");
    addSiblingButton.addActionListener(this);
    retPanel.add(addSiblingButton);
    addChildButton = new JButton("Add Child");
    addChildButton.addActionListener(this);
    retPanel.add(addChildButton);
    deleteButton = new JButton("Delete");
    deleteButton.addActionListener(this);
    retPanel.add(deleteButton);
    metaData = new JButton("Meta Data");
    metaData.addActionListener(new cd());
    retPanel.add(metaData);
    jtext = new JTextArea(9,30);
    JScrollPane scp = new JScrollPane(jtext);
    retPanel.add(scp);
    borderPane.add(retPanel,BorderLayout.CENTER);
    return borderPane;
   }
  /** This function handles menu item & its functions */
```

```
public JMenuBar constructMenuBar()
  {
    JMenu menu;
    JMenuBar menuBar = new JMenuBar();
    JMenuItem menuItem;

    menu = new JMenu("File");
    menuBar.add(menu);

    menuItem = menu.add(new JMenuItem("Open"));
    menuItem.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e)
    {
     FileDialog fd = new FileDialog(frame, "File Dialog",0);
     fd.setVisible(true);
     String str = new String();
     str = fd.getFile();
     try
     {
     FileInputStream ft = new FileInputStream(fd.getDirectory()+str);
     byte arr[] = new byte[ft.available()];
     ft.read(arr);
     jtext.append("");
     jtext.append(new String(arr));
     System.out.println(str);
     }
     catch(IOException ee){}
    }});

    menuItem = menu.add(new JMenuItem("Save"));
    menuItem.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e)
    {
     FileDialog fd = new FileDialog(frame, "File Dialog",1);
     fd.setVisible(true);
     String str = new String();
     str = fd.getFile();
     try
     {
     FileOutputStream fo = new FileOutputStream(fd.getDirectory()+str);
     byte arr1[] ;
     String stri = jtext.getText();
     arr1 = stri.getBytes();
     fo.write(arr1);
     System.out.println(str);
     }
```

```
    catch(IOException eee){}
  }});

  menuItem = menu.add(new JMenuItem("Exit"));
  menuItem.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)
{
        System.exit(0);
}});

   menu = new JMenu("Edit");
   menuBar.add(menu);

   menuItem = menu.add(new JMenuItem("Copy"));
   menuItem.addActionListener(new ActionListener(){
   public void actionPerformed(ActionEvent e)
   {

   }});

   menuItem = menu.add(new JMenuItem("Cut"));
   menuItem.addActionListener(new ActionListener(){
   public void actionPerformed(ActionEvent e)
   {

   }});
   menuItem = menu.add(new JMenuItem("Paste"));
   menuItem.addActionListener(new ActionListener(){
   public void actionPerformed(ActionEvent e)
   {

   }});
   menuItem = menu.add(new JMenuItem("Clear"));
   menuItem.addActionListener(new ActionListener(){
   public void actionPerformed(ActionEvent e)
   {

   }});

    return menuBar;
  }
/** This function returns the selected node */
   private DefaultMutableTreeNode getSelectedNode()
   {
     TreePath   selPath = tree.getSelectionPath();
```

```
 if(selPath != null)
  return (DefaultMutableTreeNode)selPath.getLastPathComponent();
  return null;
}



class cd implements ActionListener
 {

  public void actionPerformed(ActionEvent e)
  {
  m.setSize(600,450);
  m.show();

   }
 }
 class cd1 implements ActionListener
  {

    public void actionPerformed(ActionEvent e)
    {
     jtext.setText("");

    }
  }
/** This function get the values from the meta file & attaches to the tree structure*/
  public void setValue(String st)
   {
      DefaultMutableTreeNode selectedNode
       = (DefaultMutableTreeNode)
       tree.getLastSelectedPathComponent();

       DefaultMutableTreeNode newNode
        = new DefaultMutableTreeNode(st);

       System.out.println("newnode" + newNode);

       model.insertNodeInto(newNode, selectedNode,
        selectedNode.getChildCount());

        System.out.println("count  "+selectedNode.getChildCount());

        TreeNode[] nodes = model.getPathToRoot(newNode);
            TreePath path = new TreePath(nodes);
```

```
            System.out.println("path  " + path);
                tree.scrollPathToVisible(path);
    }
/** This function gets the code from mata file & displays the code in the area */
    public void setDescription(String c)
    {
    jtext.setText(c);
    }
/** This is the main method that invokes the constructor */
    public static void main(String[] args)
    {
      new codegen();
    }


    public TreeNode makeSampleTree()
    {
      DefaultMutableTreeNode root
        = new DefaultMutableTreeNode("Project");
      return root;
    }
/** This function handles the actions */
    public void actionPerformed(ActionEvent event)
    {
      DefaultMutableTreeNode selectedNode
          = (DefaultMutableTreeNode)
          tree.getLastSelectedPathComponent();

      System.out.println("selectednode " + selectedNode);

      if (selectedNode == null) return;

      if (event.getSource().equals(deleteButton))
      {
        if (selectedNode.getParent() != null)
          model.removeNodeFromParent(selectedNode);
        return;
      }

      // add new node as sibling or child

      if (event.getSource().equals(addSiblingButton))
      {
        jtext.setText("");
        try{
        FileInputStream io ;
```

```
io = new FileInputStream("details.txt");
byte arr[] = new byte[io.available()];
io.read(arr);
jtext.append(new String(arr));
}
catch(IOException ee){}
}
else if (event.getSource().equals(addChildButton))
{
System.out.println("Now for child");

input = JOptionPane.showInputDialog(frame,"Enter the class name","Input Dialog",
    JOptionPane.QUESTION_MESSAGE);
if(input != null)
{
    DefaultMutableTreeNode newNode
     = new DefaultMutableTreeNode(input);

    System.out.println("newnode" + newNode);

    model.insertNodeInto(newNode, selectedNode,
     selectedNode.getChildCount());

     System.out.println("count  "+selectedNode.getChildCount());

     TreeNode[] nodes = model.getPathToRoot(newNode);
          TreePath path = new TreePath(nodes);
          System.out.println("path  " + path);
             tree.scrollPathToVisible(path);

    }
  }
      // now display new node
}

private DefaultTreeModel model;
private JTree tree;
private JButton addSiblingButton;
private JButton addChildButton;
private JButton deleteButton;
private JButton editButton;
private JTextArea jtext;
private JButton metaData;
}
```

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.text.*;
import java.util.*;
import java.io.*;


public class meta extends JFrame
{
  JButton b;
  public codegen caller;
  clspanel cp = new clspanel();
  medpanel mp = new medpanel(cp);
  varpanel vp = new varpanel();
  compane1 cop = new compane1();
  hipanel hp  = new hipanel();
  okpanel op = new okpanel(cp,vp,mp,cop,hp);
  int xx=0;
  /** This constructor is used to create the Tab controls  */
    public meta(codegen caller)
    {
      super("MetaData Propeties");
      Container f = getContentPane();
      b =new JButton("OK");
      f.setSize(850,700);
      JTabbedPane jtp= new JTabbedPane();
      jtp.addTab("Class",cp);
      jtp.addTab("Hierarchy",hp);
      jtp.addTab("Variables",vp);
      jtp.addTab("Methods",mp);
      jtp.addTab("Comments",cop);
      jtp.addTab("OK",op);
      f.add(jtp,BorderLayout.CENTER);

     /** Window closing Event it Handled here */
      addWindowListener(new WindowAdapter()
        {
         public void windowClosing(WindowEvent e)
         {
          removeNotify();
         }
        });

        f.setVisible(true);
        this.caller=caller;
```

```
}

class clspanel extends JPanel
{
 JLabel l1,l2,l3,l4,l5;
 JTextField tf1,tf2,tf3;
 JButton jb1,jb2;
 JComboBox co,co1;
 public  String s=new String();
 JCheckBox jc1,jc2,jc3;

 public clspanel()
 {
  setLayout(null);

  l1= new JLabel("Class Name");
  l2= new JLabel("Package");
  l3= new JLabel("File Name");
  l4= new JLabel("Directory");
  l5= new JLabel("Class / Interface");

  jc1 = new JCheckBox("Public");
  jc2 = new JCheckBox("Abstract");
  jc3 = new JCheckBox("Final");

  jc2.addItemListener(new ItemListener()
  {
   public void itemStateChanged(ItemEvent ie)
   {
    if ((JCheckBox)ie.getItem()==jc2)
     {
      if(ie.getStateChange()==ie.SELECTED)
        jc3.setEnabled(false);
      else
        jc3.setEnabled(true);
     }
    }
   });

   jc3.addItemListener(new ItemListener()
   {
    public void itemStateChanged(ItemEvent ie)
     {
      if ((JCheckBox)ie.getItem()==jc3)
       {
        if(ie.getStateChange()==ie.SELECTED)
```

```
    jc2.setEnabled(false);
   else
    jc2.setEnabled(true);
   }
  }
});

tf1 = new JTextField(10);
tf2 = new JTextField(20);
tf3 = new JTextField(20);

jb1= new JButton("New");
jb2= new JButton("...");

co = new JComboBox();
co.addItem("");
co.addItem("java");
co.addItem("java.lang");
co.addItem("java.util");

co1 = new JComboBox();
co1.addItem("class");
co1.addItem("interface");

l1.setBounds(20,50,80,20);
l5.setBounds(20,20,120,20);
co1.setBounds(150,20,100,20);
tf1.setBounds(20,80,120,30);
l2.setBounds(20,120,80,20);
co.setBounds(20,150,120,30);
jb1.setBounds(160,150,80,30);
jc1.setBounds(350,50,80,20);
jc2.setBounds(350,100,80,20);
jc3.setBounds(350,150,80,20);
l3.setBounds(20,180,80,20);
tf2.setBounds(20,210,120,30);
l4.setBounds(20,250,80,20);
tf3.setBounds(20,280,120,30);
jb2.setBounds(250,280,80,20);

add(l1);    add(l2);    add(l3);    add(l4);add(l5);

add(tf1);    add(tf2);    add(tf3);add(co1);

add(jb1); add(jb2);
```

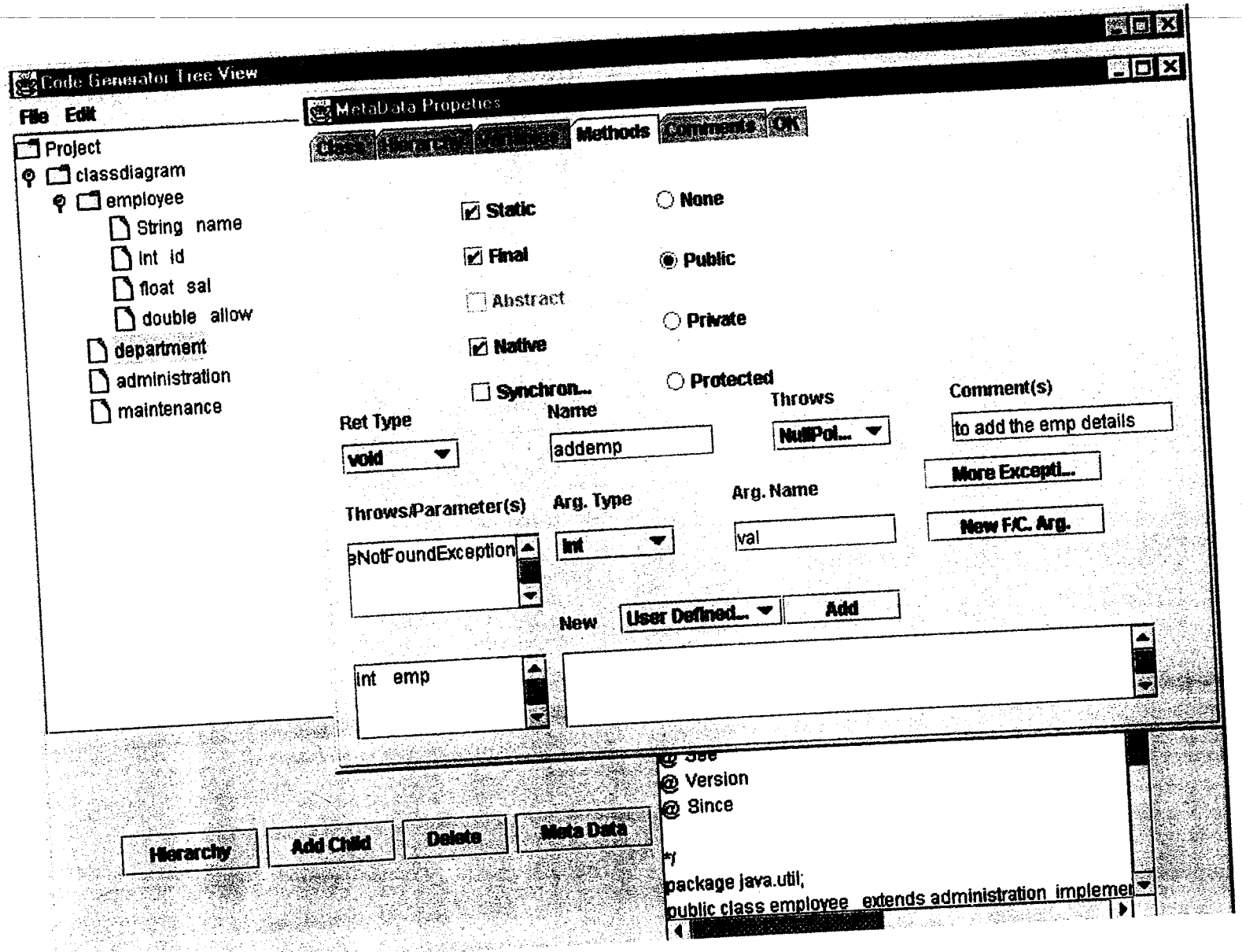
```
add(jc1); add(jc2); add(jc3);

 add(co);
}

public String getString()
{
 s=tf1.getText();
 return(s);
}

 /** @return "True if selected one is an interface ,false for class" */

 public boolean getType()
 {
  if(co1.getSelectedItem().equals("interface"))
  return(true);
  else
  return(false);
 }
/** collecting the values from this class in order to generate the skeletal code*/
  public String getDescription()
   {
     StringBuffer toReturn=new StringBuffer();

      if (co.getSelectedItem()!=null)
        toReturn.append("package "+co.getSelectedItem()+";"+"\n");

       if (jc1.isSelected())
         toReturn.append("public ");

       if (jc2.isSelected())
         toReturn.append("abstract ");

       else if (jc3.isSelected())
          toReturn.append("final ");

        if(co1.getSelectedItem().equals("class"))
         toReturn.append("class ");

        else
         toReturn.append("interface ");

        toReturn.append(tf1.getText());

        return(toReturn.toString());
```

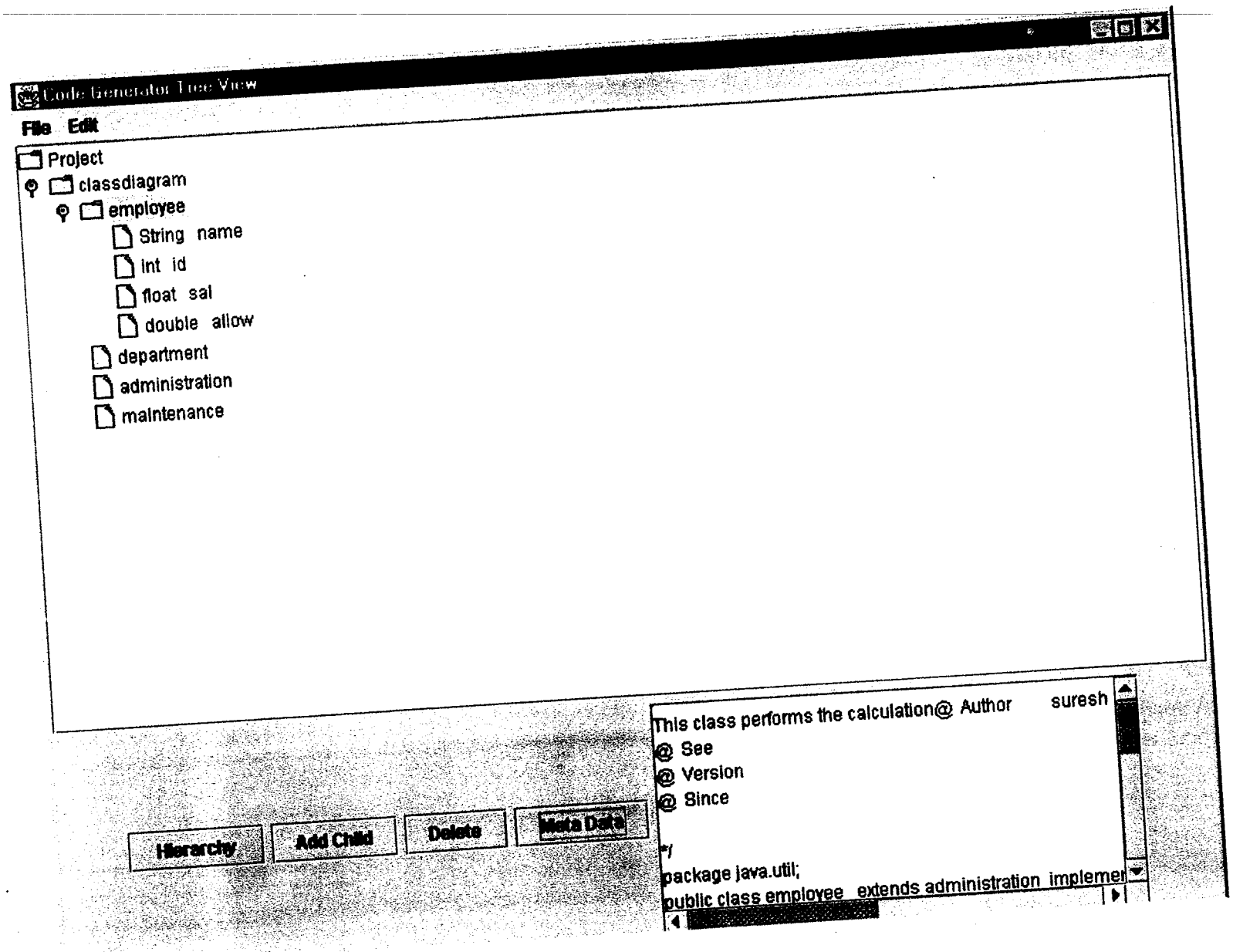
```
    }
}

    }


class okpanel extends JPanel
{
 JButton b;
 JLabel l;
 String s= new String();
 String hir[] = new String[15];
 String app;
 clspanel z1 = new clspanel();
 varpanel z2 = new varpanel();
 medpanel z3 = new medpanel(z1);
 companel z4 = new companel();
 hipanel z5  = new hipanel();
 FileOutputStream fo;

 public okpanel(clspanel c,varpanel c1,medpanel c2,companel c3,hipanel c4)
  {
   l = new JLabel("Display The Code");
   b= new JButton("OK");

   z1 = c;
   z2 = c1;
   z3 = c2;
   z4 = c3;
   z5 = c4;
   add(l);
   add(b);
   try{
   fo = new FileOutputStream("details.txt");
    }catch(IOException e2){}
   b.addActionListener(new ActionListener()
   {
   public void actionPerformed(ActionEvent e)
   {
    if((e.getActionCommand()).equals("OK"))
     {
     app=" " ;
     int j= z5.i;
     hir[xx]=z1.tf1.getText() +"  "+ z5.tf1.getText() +"  " +
z5.j1.getText()+"\n";
```

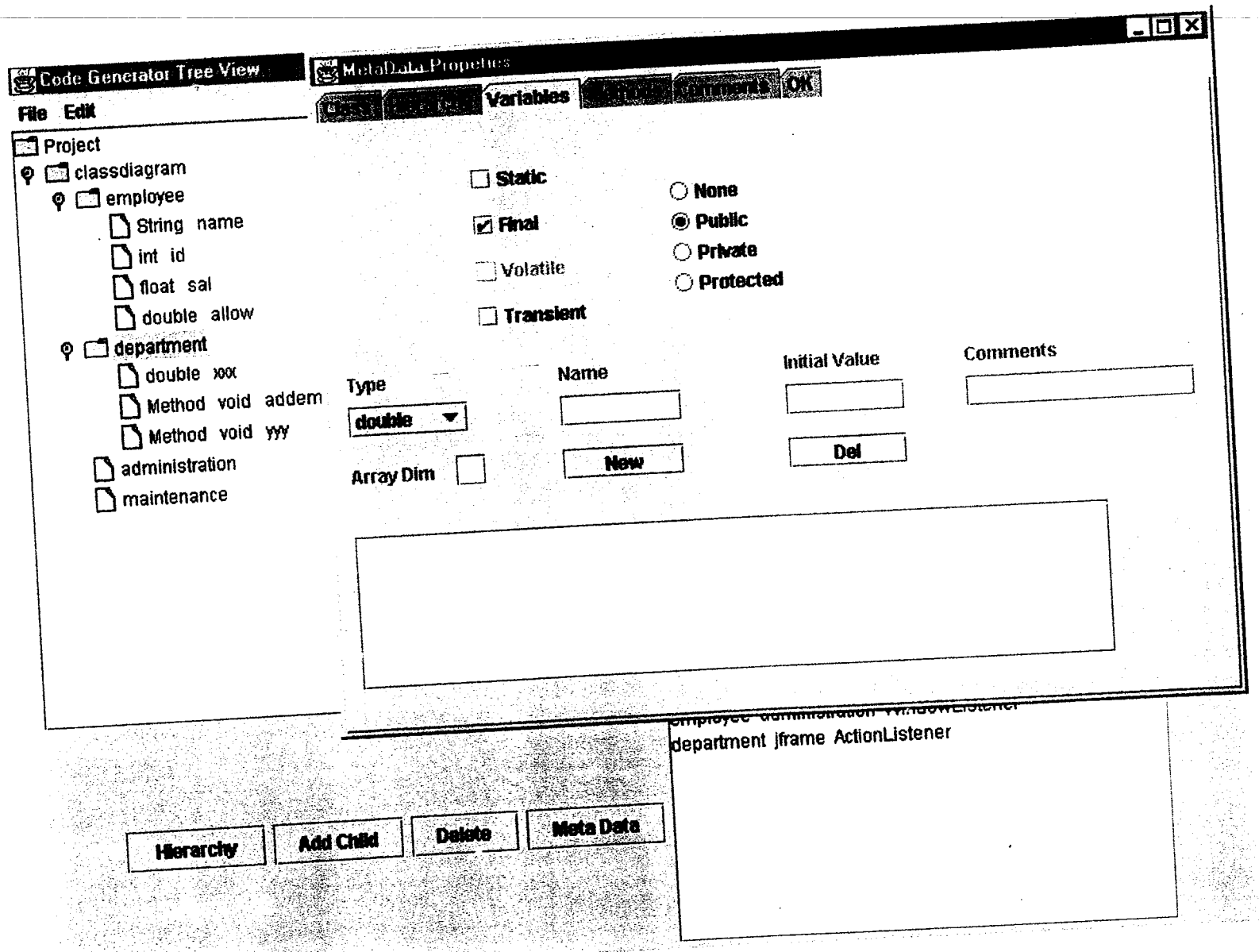
```
try
{
byte arr1[] ;
String stri = hir[xx];
arr1 = stri.getBytes();
fo.write(arr1);
}
catch(IOException eee){}

System.out.println(hir[xx]);
xx++;
for(int z=0;z<j;z++)
{
if(z5.str[z].equals("ActionListener"))
{
  app="\tpublic void actionPerformed(ActionEvent e)\n\t{\n\t}";
  break;
}
else
  app="";
 }

setDescription(z4.getDescription()+"\n"+
z1.getDescription() + z5.getDescription() +"\n"+
z2.getDescription() +"\n"+ z3.getDescription()+"\n"+app+"\n }");
setValue(z2.x,z2.arr);
setValue(z3.x,z3.str);

z1.tf1.setText(""); z1.tf2.setText("");z1.tf3.setText("");
z5.tf1.setText("");z5.j1.setText("");
z2.ja.setText("");
z3.ja1.setText("");z3.ja2.setText("");z3.ja3.setText("");
z4.ta1.setText("");z4.ta2.setText("");


for( int k=0;k < (z2.x);k++)
z2.arr[k]=" ";
z2.x=0;

for( int k=0;k < (z3.x);k++)
z3.str[k]=" ";
z3.x=0;
 }
}
});
```

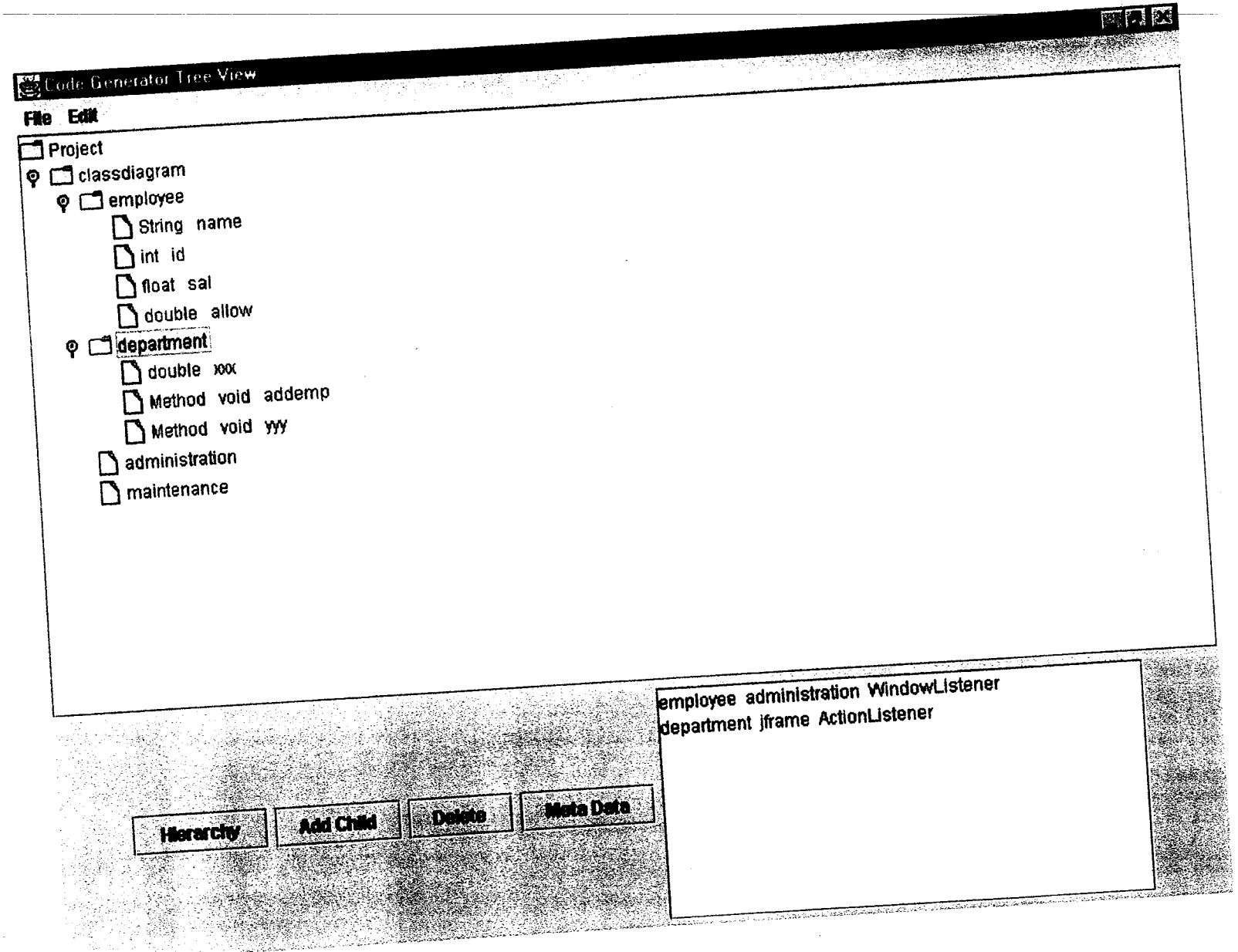
```
    }
  }
/**  This function is used to display fields & user defined methods in the root hierarchy
*/
  private void setValue(int i,String arr[])
  {
  for (int j =0;j<i;j++)
  {
  caller.setValue(arr[j]);
  }
  }

 /** This function is used to display the skeletal code to the text area in codegen*/

   private void setDescription(String s) {
      caller.setDescription(s);
   }
}
```

SAMPLE I/


Code Generator Tree View
File Edit
Project
classdiagram
employee
String name
int id
float sal
double allow
department
administration
maintenance
MetaData Propeties
Methods
Comments
OK
Static
Final
Abstract
Native
Synchron...
None
Public
Private
Protected
Ret Type
void
Name
addemp
Throws
NullPoi...
Comment(s)
to add the emp details
More Excepti...
Throws/Parameter(s)
eNotFoundException
Arg. Type
int
Arg. Name
val
New F/C. Arg.
New
User Defined...
Add
int emp
@ Version
@ Since
*/
package java.util;
public class employee extends administration implemen
Hierarchy
Add Child
Delete
Meta Data


Code Generator Tree View
File Edit
Project
classdiagram
employee
String name
int id
float sal
double allow
department
administration
maintenance
Hierarchy
Add Child
Delete
Meta Data
This class performs the calculation@ Author suresh
@ See
@ Version
@ Since
*/
package java.util;
public class employee extends administration implemen


Code Generator Tree View
File Edit
Project
classdiagram
employee
String name
int id
float sal
double allow
department
double xxx
Method void addem
Method void yyy
administration
maintenance
MetaData Properties
Variables
OK
Static
Final
Volatile
Transient
None
Public
Private
Protected
Type
double
Name
Initial Value
Comments
Array Dim
New
Del
department jframe ActionListener
Hierarchy
Add Child
Delete
Meta Data


Code Generator Tree View
File Edit
Project
classdiagram
employee
String name
int id
float sal
double allow
department
double xxx
Method void addemp
Method void yyy
administration
maintenance
employee administration WindowListener
department jframe ActionListener
Hierarchy
Add Child
Delete
Meta Data

```
/**
This class performs the calculation@ Author       suresh
@ See
@ Version
@ Since

*/
package java.util;
public class employee   extends administration  implements
WindowListener
 {
        public  static volatile    String        name;              // name of
 the employee
        public  final    int        id;                 // id of the emp
        public    final   float        sal[ ][ ];              // salary in this
 value
        public   final    double        allow  =   100 ;       // allowance of
 the emp

        //
        public    static final native synchronized void   main ( String[ ]
  args)  throws NullPointerException,IOException {  }
        //
        public  employee ( ) {  }
        //cla the salary
        public  employee ( boolean    ff
   , int    ss
   ) {  }


}
```

JAVA DOC

Class codegen

```
java.lang.Object
  |
  +--codegen
```

public class **codegen**
extends java.lang.Object
implements java.awt.event.ActionListener

DynamicTreeNode illustrates one of the possible ways in which dynamic loading can be used in tree. The basic premise behind this is that getChildCount() will be messaged from JTreeModel before any children are asked for. So, the first time getChildCount() is issued the children are loaded.

It should be noted that isLeaf will also be messaged from the model. The default behavior of TreeNode is to message getChildCount to determine this. As such, isLeaf is subclassed to always return false.

There are others ways this could be accomplished as well. Instead of subclassing TreeNode you could subclass JTreeModel and do the same thing in getChildCount(). Or, if you aren't using TreeNode you could write your own TreeModel implementation. Another solution would be to listen for TreeNodeExpansion events and the first time a node has been expanded post the appropriate insertion events. I would not recommend this approach though, the other two are much simpler and cleaner (and are faster from the perspective of how tree deals with it). NOTE: getAllowsChildren() can be messaged before getChildCount(). For this example the nodes always allow children, so it isn't a problem, but if you do support true leaf nodes you may want to check for loading in getAllowsChildren too.

Author:
Jayaram.U, SivaKumar.K, SivaKumaran.J, Somu.L

Field Summary

protected javax.swing.tree.DefaultTreeModel	treeModel

Constructor Summary

Method Summary

void	actionPerformed(java.awt.event.ActionEvent event) This function handles the actions
javax.swing.JMenuBar	constructMenuBar() This function handles menu item & its functions
static void	main(java.lang.String[] args) This is the main method that invokes the constructor
javax.swing.tree.TreeNode	makeSampleTree()
void	setDescription(java.lang.String c) This function gets the code from mata file & displays the code in the area
void	setValue(java.lang.String st) This function get the values from the meta file & attaches to the tree structure

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

treeModel

```
protected javax.swing.tree.DefaultTreeModel treeModel
```

Constructor Detail

codegen

```
public codegen()
```

This constructor is used to build the tree structure,Buttons,file menu & Text area

Method Detail

constructMenuBar

```
public javax.swing.JMenuBar constructMenuBar()
```

This function handles menu item & its functions

setValue

```
public void setValue(java.lang.String st)
```

This function get the values from the meta file & attaches to the tree structure

setDescription

```
public void setDescription(java.lang.String c)
```

This function gets the code from mata file & displays the code in the area

main

```
public static void main(java.lang.String[] args)
```

This is the main method that invokes the constructor

makeSampleTree

```
public javax.swing.tree.TreeNode makeSampleTree()
```

actionPerformed

```
public void actionPerformed(java.awt.event.ActionEvent event)
```

This function handles the actions

Specified by:

actionPerformed in interface java.awt.event.ActionListener

Hierarchy For All Packages

Class Hierarchy

- class java.lang.Object
  - class codegen (implements java.awt.event.ActionListener)
  - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - class java.awt.Container
      - class java.awt.Window
        - class java.awt.Frame (implements java.awt.MenuContainer)
          - class javax.swing.JFrame (implements javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants)
            - class **meta**

Class meta

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
              |
              +--java.awt.Window
                    |
                    +--java.awt.Frame
                          |
                          +--javax.swing.JFrame
                                |
                                +--meta
```

public class **meta**
extends javax.swing.JFrame

Author:
Jayaram.U, SivaKumar.K, SivaKumaran.J, Somu.L

See Also:
Serialized Form

Inner classes inherited from class javax.swing.JFrame
`javax.swing.JFrame.AccessibleJFrame`

Field Summary

codegen	`caller`

Fields inherited from class javax.swing.JFrame

accessibleContext, rootPane, rootPaneCheckingEnabled

Fields inherited from class java.awt.Frame

CROSSHAIR_CURSOR, DEFAULT_CURSOR, E_RESIZE_CURSOR, HAND_CURSOR, ICONIFIED, MOVE_CURSOR, N_RESIZE_CURSOR, NE_RESIZE_CURSOR, NORMAL, NW_RESIZE_CURSOR, S_RESIZE_CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR, TEXT_CURSOR, W_RESIZE_CURSOR, WAIT_CURSOR

Fields inherited from class java.awt.Component

BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Constructor Summary

meta(codegen caller)

This constructor is used to create the Tab controls

Methods inherited from class javax.swing.JFrame

addImpl, createRootPane, frameInit, getAccessibleContext, getContentPane, getDefaultCloseOperation, getGlassPane, getJMenuBar, getLayeredPane, getRootPane, isRootPaneCheckingEnabled, paramString, processKeyEvent, processWindowEvent, remove, setContentPane, setDefaultCloseOperation, setGlassPane, setJMenuBar, setLayeredPane, setLayout, setRootPane, setRootPaneCheckingEnabled, update

Methods inherited from class java.awt.Frame

addNotify, finalize, getCursorType, getFrames, getIconImage, getMenuBar, getState, getTitle, isResizable, remove, removeNotify, setCursor, setIconImage, setMenuBar, setResizable, setState, setTitle

Methods inherited from class java.awt.Window

addWindowListener, applyResourceBundle, applyResourceBundle, dispose, getFocusOwner, getInputContext, getLocale, getOwnedWindows, getOwner, getToolkit, getWarningString, hide, isShowing, pack, postEvent, processEvent, removeWindowListener, setCursor, show, toBack, toFront

Methods inherited from class java.awt.Container

add, add, add, add, add, addContainerListener, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getInsets, getLayout, getMaximumSize, getMinimumSize, getPreferredSize, insets, invalidate, isAncestorOf, layout, list, list, locate, minimumSize, paint, paintComponents, preferredSize, print, printComponents, processContainerEvent, remove, removeAll, removeContainerListener, setFont, validate, validateTree

Methods inherited from class java.awt.Component

action, add, addComponentListener, addFocusListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addPropertyChangeListener, addPropertyChangeListener, bounds, checkImage, checkImage, coalesceEvents, contains, contains, createImage, createImage, disable, disableEvents, dispatchEvent, enable, enable, enableEvents, enableInputMethods, firePropertyChange, getBackground, getBounds, getBounds, getColorModel, getComponentOrientation, getCursor, getDropTarget, getFont, getFontMetrics, getForeground, getGraphics, getHeight, getInputMethodRequests, getLocation, getLocation, getLocationOnScreen, getName, getParent, getPeer, getSize, getSize, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent, hasFocus, imageUpdate, inside, isDisplayable, isDoubleBuffered, isEnabled, isFocusTraversable, isLightweight, isOpaque, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, prepareImage, prepareImage, printAll, processComponentEvent, processFocusEvent, processInputMethodEvent, processMouseEvent, processMouseMotionEvent, removeComponentListener, removeFocusListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, repaint, requestFocus, reshape, resize, resize, setBackground, setBounds, setBounds, setComponentOrientation, setDropTarget, setEnabled, setForeground, setLocale, setLocation, setLocation, setName, setSize, setSize, setVisible, show, size, toString, transferFocus

Methods inherited from class java.lang.Object

clone, equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

caller

public codegen **caller**

Constructor Detail

meta

public **meta**(codegen caller)

This constructor is used to create the Tab controls

A

actionPerformed(ActionEvent) - Method in class codegen
This function handles the actions

C

caller - Variable in class meta

codegen - class codegen.
DynamicTreeNode illustrates one of the possible ways in which dynamic loading can be used in tree.
codegen() - Constructor for class codegen
This constructor is used to build the tree structure,Buttons,file menu & Text area
constructMenuBar() - Method in class codegen
This function handles menu item & its functions

M

main(String[]) - Static method in class codegen
This is the main method that invokes the constructor
makeSampleTree() - Method in class codegen

meta - class meta.
Copyright (c) 1997-1999 by Cybersoft Microsystems, Inc.
meta(codegen) - Constructor for class meta
This constructor is used to create the Tab controls

S

setDescription(String) - Method in class codegen
This function gets the code from mata file & displays the code in the area
setValue(String) - Method in class codegen
This function get the values from the meta file & attaches to the tree structure

Serialized Form

Class meta implements Serializable

Serialized Fields

b

javax.swing.JButton b

---

caller

codegen **caller**

---

cop

meta.companel cop

---

cp

meta.clspanel cp

---

hp

meta.hipanel hp

---

mp

meta.medpanel mp

---

op

meta.okpanel op

vp

meta.varpanel **vp**

xx

int **xx**

Bibliography

IEEE Std 830, Guide to Software Requirements Specification.

1 **Websites :**

- "Structure Builder " [Version 4.0, Build 385], http://www.webgain.com , WebGain Inc. 5425 Stevens Creek Blvd Santa Clara, CA 95051
- "Rational Rose" , http://www.Rational.com .
- "Visual Age for professional java ",IBM , http://www.ibm.com .

2 **Books :**

- Patrick Naughton & Herbert Schildt .,"*The complete reference java 2*",The McGraw-Hill companies inc,New York,Third Edition.
- John Zukowski .,"*Mastering Java2*", 1999BPB Publications, New Delhi,
- Bruce Eckel.,*"Thinking in Java"*, 1999 BPB Publications, New Delhi