USER FRIENDLY FRAME BUILDER ENVIRONMENT FOR JAVA SWING

By

Mythily.D
Registration Number: 71205621027

Of
KUMARAGURU COLLEGE OF TECHNOLOGY
COIMBATORE

A PROJECT REPORT
Submitted to the

FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

In partial fulfillment of the requirements
For the award of the degree

Of

MASTER OF COMPUTER APPLICATION

ANNA UNIVERSITY
CHENNAI 600 025

June 2008

BONAFIDE CERTIFICATE

Certified that this project report titled User Friendly Frame Builder Environment for Java Swing is the bonafide work of Ms. Mythily.D (Registration Number: 71205621027) who carried out the research under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Supervisor **Head of the Department**

Submitted to Project and Viva Examination held on 01-07-2008

Internal Examiner External Examiner

OSPROSYS TECHNOLOGIES

I I/5, Vasan Apartments, IInd Floor, Ayyavu Street, Aminijikarai, Chennai - 29.
Ph : 044 - 4260 4240, www.osprosys.net

June 2nd, 2008

To Whomsoever It May Concern

This is to certify that Miss.MYTHILY.D (71205621027), Student of Kumaraguru College of Technology, Coimbatore doing her Final Year, M.C.A (Computer Application) has successfully completed her project entitled as "USER FRIENDLY FRAME BUILDER ENVIRONMENT FOR JAVA SWING" under the guidance of Mr.Karthikeyan.A, Senior Software Engineer, from 04th Jan 2008 to 2nd June 2008.

During her Project duration her conduct and contribution has been excellent.

We wish her all the best for her future Endeavors.

OSPROSYS TECHNOLOGIES
11/5, Vasan Apartments, 2nd Floor, Ayyavu St, Aminijikarai, Ph-044-42604240
CHENNAI-29

For Osprosys Technologies

(Project Co-ordinator)

ABSTRACT

The software **User Friendly Frame Builder Environment for Java Swing** is capable of developing java application using swing from Java. This project automatically generates Java source codes for the design, designed by the user. On behalf of writing source codes for designing, this project uses design-oriented approach to develop applications easily. The design will produce Java source code as an output. The output source will be saved to the disk as a Java file. The file can be edited with other source editors or text editors.

The Development of projects or program in a Language, simply in Text mode is very hard and takes lot of time to develop. This leads to delay in developing the projects. To simplify the tasks and to create projects in quick time, this IDE helps programmers. Using these IDE we can design the components or controls in Java, very easily, quickly and efficiently. If we select any component from the toolbox that will appear on the design window and also we can set the property of each component from the property window.

The Code for creating the swing controls and to handle the swing components like List box and Image List Boxes are tedious and laborious. To minimize these works and to simplify this task, we can use this IDE. Using this, code for the package is automatically generated. We can use this and implement this integrated environment to build various applications and programs which supports GUI (Graphical user Interface) environment.

For Java beginners or users, it is too tedious to design a form using a Java program because Java is a programming language which is case sensitive. So typing a lengthy Java program in editor such as notepad, MS-DOS editor or WordPad increases user's burden. While compile this large program, it may produce a lot of error instructions. Debugging such a huge amount of errors is impatient work. So this project helps beginners to develop project in quick time.

ACKNOWLEDGEMENT

I wish to express my sincere thanks to **Dr.JOSEPH V.THANIKAL** Principal, Kumaraguru College of Technology, Coimbatore, for giving me the consent to undertake this project.

My deepest acknowledgement to **Dr.M.GURURAJAN** Head of the Department, Computer Applications, Kumaraguru College of Technology, Coimbatore, for his timely help and guidance throughout this project.

I am greatly indebited to **Mrs.V.GEETHA** Project Coordinator, Assistant Professor, Department of computer Applications, Kumaraguru College of Technology, Coimbatore and my sincere thanks to her for guiding and encouraging me at every stage of this project.

I am greatly indebted to my guide **Mr.M.MANIKANTAN** Senior Lecturer, Department of computer Applications, Kumaraguru College of Technology, Coimbatore, for his valuable guidance and encouragement at every stage of this project

I express my sincere thanks to **Mr.KARTHIKEYAN**, Senior Software Engineer, Project Leader, OSPROSYS TECHNOLOGIES, Chennai for his support and assistance at various levels of my project work.

Finally, I owe my great deal of gratitude to my parents for helping me to overwhelm in all my proceedings. I bow my heart and head with heartfelt thanks to my department staffs and all those who taught me their warm service to succeed and achieve my work.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 ORGANIZATION PROFILE:

Osprosys Technologies is a leading Software Development firm since the year 2000. Osprosys Technologies located at Chennai, Tamil Nadu. Domestic operation had already commenced in 1998 with emphasis in software engineering on time material basis largely catering to the market. We have client in India and Abroad.

Osprosys Technologies is a leading international business technology company with products backed by a strong domain expertise to provide cost-effective comprehensive solutions for businesses. Use of strong domain knowledge, deployment of the our robust technology frame work, a client centric, template free, approach to solutions deployed and a sound delivery model form the basic tenets of our organization. Osprosys Technologies has had the privilege of serving a marquee customer base with a strong financial record and robust year-over-year growth.

The Services Offered is:

1 Software Development – Off Shore / On- Site on Java, Telecom, DSP, Embedded
2 Product on DSP, Embedded Systems, Telecom,
3 IT Enable Services
4 Product Related Solutions
5 Web Designing

1.2 ABSTRACT

The software **User Friendly Frame Builder Environment for Java Swing** is capable of developing java application using swing from Java. This project automatically generates Java source codes for the design, designed by the user. On behalf of writing source codes for designing, this project uses design-oriented approach to develop applications easily. The design will produce Java source code as an output. The output source will be saved to the disk as a Java file. The file can be edited with other source editors or text editors.

The Development of projects or program in a Language, simply in Text mode is very hard and takes lot of time to develop. This leads to delay in developing the projects. To simplify the tasks and to create projects in quick time, this IDE helps programmers. Using these IDE we can design the components or controls in Java, very easily, quickly and efficiently. If we select any component from the toolbox that will appear on the design window and also we can set the property of each component from the property window.

The Code for creating the swing controls and to handle the swing components like List box and Image List Boxes are tedious and laborious. To minimize these works and to simplify this task, we can use this IDE. Using this, code for the package is automatically generated. We can use this and implement this integrated environment to build various applications and programs which supports GUI (Graphical user Interface) environment.

For Java beginners or users, it is too tedious to design a form using a Java program because Java is a programming language which is case sensitive. So typing a lengthy Java program in editor such as notepad, MS-DOS editor or WordPad increases user's burden. While compile this large program, it may produce a lot of error instructions. Debugging such a huge amount of errors is impatient work. So this project helps beginners to develop project in quick time.

1.3 PROBLEM DEFINITION

User need identification and analysis are concerned with what the user needs rather than what they want. Not until the problem has been identified, defined & evaluate should the solutions be thought about & whether the problem is worth solving.

Determining the information each user needs is a particularly difficult task. In fact, it is recognized as one of the most difficult tasks in system development. The usual approach is to ask the user what information is currently available & what other information is required, with the needs & requirements of the user & concern. The system is to be defined clearly & previously.

CHAPTER 2

SYSTEM ANALYSIS

2.1 EXISTING SYSTEM

This type of IDE has already developed under java. We have enhanced and remove some of the disadvantages present in the existing system.

Some of existing IDE developed in java are

1 Net Beans

2 JDeveloper

3 KAWA

There are some limitations in those existing system. Few disadvantages are

1 It occupies more memory

2 They require Pentium IV processor and 128MB RAM to perform execution.

3 This software takes more execution time.

2.2 PROPOSED SYSTEM

This software is capable of developing java application using swing from Java. This project automatically generates Java source codes for the design designed by the user. On behalf of writing source codes for designing, this project uses design-oriented approach to develop applications easily. The design will produce Java source code as an output. The output source will be saved to the disk as a Java file. The file can be edited with other source editors or text editors.

This software is designed in 4 phases they are

1. Designing Phase
2. Component Adding Phase
3. Property Modification Phase
4. Automatic Code Generation Phase

New system is required to satisfy the following requirements,

- To provide an easy and efficient environment
- Fast, feasible and flexible
- To provide an easy user interactive area
- To reduce errors and so easy the debugging
- To reduce required time

Usually, the Java program is coded in an editor like notepad or edited in the Dos prompt, compiled &executed in the Dos prompt and lacks development environment. Determining the information each user needs is a particularly difficult task. In fact it is recognized as one of the most difficult tasks in system Development. Writing manually, Error checking, Time consuming is difficult to Java beginners. The approach is to ask the user what information is currently available and what other information is required.

With the needs and requirements of the user and concern, the system is to be defined clearly and precisely.

Design a GUI interface.

Generate code on designing.

2.3 USER INTERFACE REQUIREMENTS

Module I (Designing Phase)

In this phase we design menu bar, toolbar, property window, main frame and code window to provide a user interactive area.

Module II (Component Adding phase)

The job of component adding phase is to merge the selected tools into the design frame

Module III (Property Modification)

This phase contains code to set the properties on runtime. User can change the properties of the components at runtime so that it is altered in code window.

Module IV (Automatic Code Generation)

This is the final phase of this project which generate coding when the user double clicked on the design frame

CHAPTER 3

DEVELPOMENT ENVIRONMENT

3.1 HARDWARE ENVIRONMENT

Processor	: Intel Pentium IV
Clock Speed	: 700 MHZ
RAM	: 256 MB
Monitor	: 14" SVGA Digital Color Monitor
Keyboard	: 107 Keys Keyboard
Floppy Drive	: 1.44MB
Compact Disk Drive	: 700MB
Hard Disk	: 40GB
Printer	: Canon BJC 2100 SP
Mouse	: Logitech Mouse

3.2 SOFTWARE ENVIRONMENT

Operating System	: Windows 98/2000/XP
Software	: Jdk 1.4 /J2sdk1.4.1_06

3.3 SOFTWARE OVERVIEW

3.3.1 The Java Architecture

Java's architecture arises out of four distinct but interrelated technologies:

- The Java programming language
- The Java class file format
- The Java Application Programming Interface
- The Java virtual machine

When we write and run a Java program, we are tapping the power of these four technologies. We express the program in source files written in the Java programming language, compile the source to Java class files, and run the class files on a Java virtual machine. When we write our program, we access system resources (such as I/O, for example) by calling methods in the classes that implement the Java Application Programming Interface, or Java API. As our program runs, it fulfills program's Java API calls by invoking methods in class files that implement the Java API. We can see the relationship between these four parts in Figure 3.3.1.1

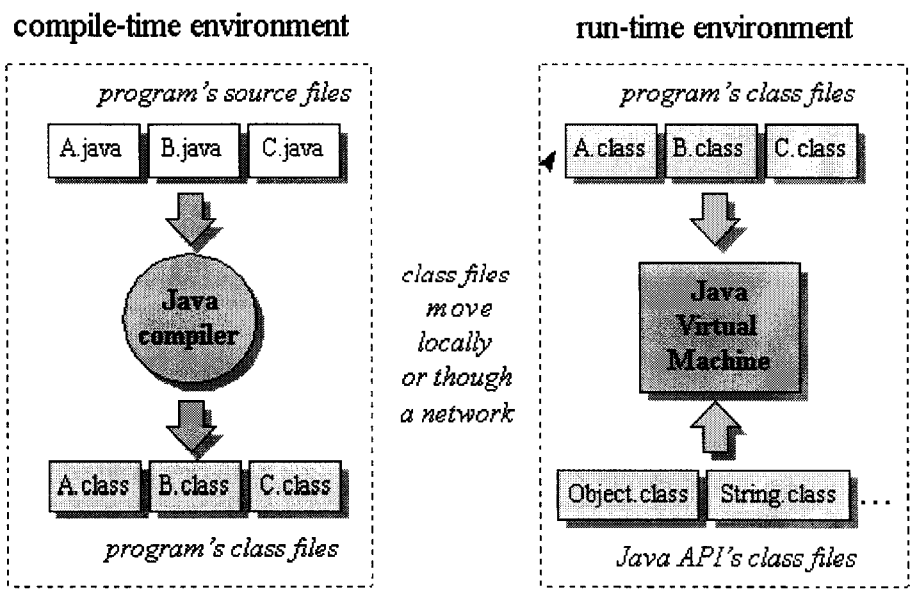


Figure 3.3.1.1 The Java programming environment.

Together, the Java virtual machine and Java API form a "platform" for which all Java programs are compiled. In addition to being called the *Java runtime system*, the combination of the Java virtual machine and Java API is called the *Java Platform* (or, starting with version 1.2, the *Java 2 Platform*). Java programs can run on many different kinds of computers because the Java Platform can itself be implemented in software. As you can see in Figure 3.3.1.2, a Java program can run anywhere the Java Platform is present.

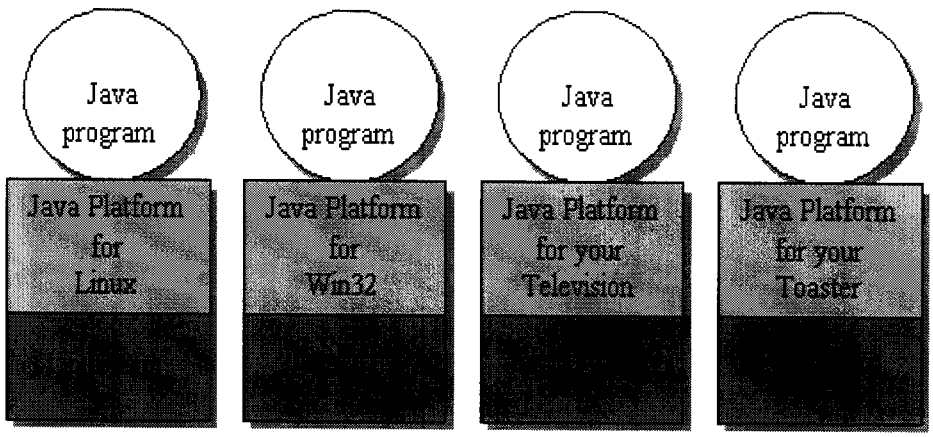


Figure 3.3.1.2 Java programs run on top of the Java Platform.

3.3.2 The Java Virtual Machine

At the heart of Java's network-orientation is the Java virtual machine, which supports all three prongs of Java's network-oriented architecture: platform independence, security, and network-mobility.

The Java virtual machine is an abstract computer. Its specification defines certain features every Java virtual machine must have, but leaves many choices to the designers of each implementation. For example, although all Java virtual machines must be able to execute Java bytecodes, they may use any technique to execute them. Also, the specification is flexible enough to allow a Java virtual machine to be implemented either completely in software or to varying degrees in hardware. The flexible nature of the Java virtual machine's specification enables it to be implemented on a wide variety of computers and devices.

A Java virtual machine's main job is to load class files and execute the bytecodes they contain. As you can see in Figure 3.3.2.1, the Java virtual machine contains a *class loader*, which loads class files from both the program and the Java API. Only those class files from the Java API that are actually needed by a running program are loaded into the virtual machine. The bytecodes are executed in an *execution engine*.

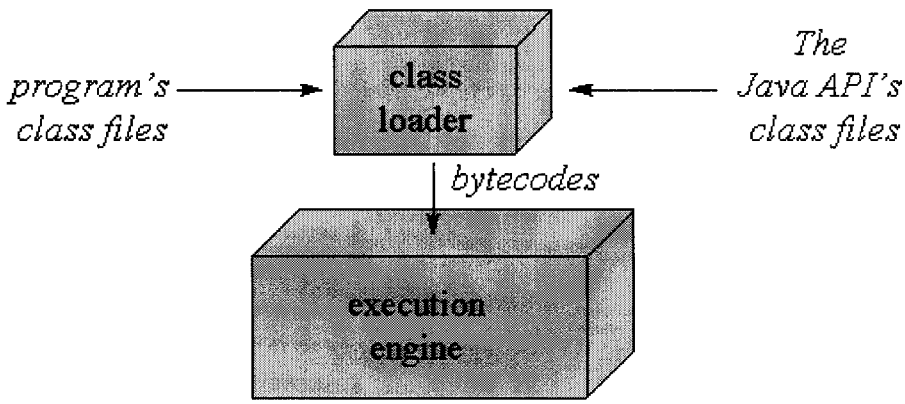


Figure 3.3.2.1 A basic block diagram of the Java virtual machine.

The execution engine is one part of the virtual machine that can vary in different implementations. On a Java virtual machine implemented in software, the simplest kind of execution engine just interprets the bytecodes one at a time. Another kind of execution engine, that is faster but requires more memory, is a *just-in-time compiler*. In this scheme, the bytecodes of a method are compiled to native machine code the first time the method is invoked. The native machine code for the method is then cached, so it can be re-used the next time that same method is invoked. A third type of execution engine is an *adaptive optimizer*. In this approach, the virtual machine starts by interpreting bytecodes, but monitors the activity of the running program and identifies the most heavily used areas of code. As the program runs, the virtual machine compiles to native and optimizes just these heavily used areas. The rest of the code, which is not heavily used, remains as bytecodes, which the virtual machine continues to interpret. This adaptive optimization approach enables a Java virtual machine to spend typically 80 to 90% of its time executing highly optimized native code, while requiring it to compile and optimize only the 10 to 20% of the code that really matters to performance. Lastly, on a Java virtual machine built on top of a chip that executes Java bytecodes natively, the execution engine is actually embedded in the chip.

Sometimes the Java virtual machine is called the *Java interpreter*; however, given the various ways in which bytecodes can be executed, this term can be misleading. While "Java interpreter" is a reasonable name for a Java virtual machine that interprets bytecodes, virtual machines also use other techniques (such as just-in-time compiling) to execute bytecodes. Therefore, although all Java interpreters are Java virtual machines, not all Java virtual machines are Java interpreters.

When running on a Java virtual machine that is implemented in software on top of a host operating system, a Java program interacts with the host by invoking *native methods*. In Java, there are two kinds of methods: Java and native. A Java method is written in the Java language, compiled to bytecodes, and stored in class files. A native method is written in some other language, such as C, C++, or assembly, and compiled to the native machine code of a particular processor. Native methods are stored in a dynamically linked library whose exact form is platform specific. While Java methods are platform independent, native methods are not. When a running Java program calls a native method, the virtual machine loads the dynamic library that contains the native method and invokes it. As you can see in Figure 3.3.2.1, native methods are the connection between a Java program and an underlying host operating system.

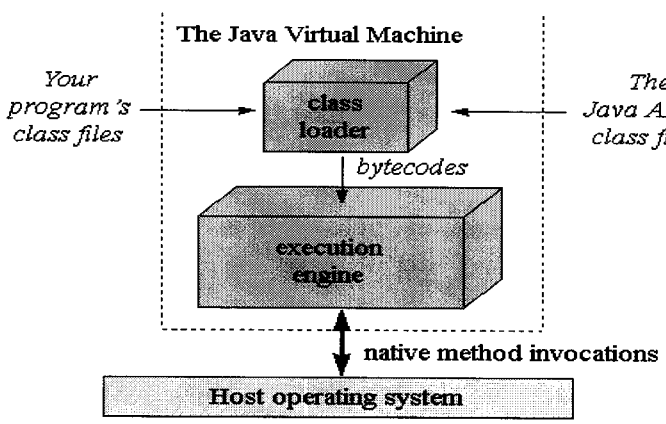


Figure 3.3.2.2 A Java virtual machine implemented in software on top of a host operating system.

You can use native methods to give your Java programs direct access to the resources of the underlying operating system. Their use, however, will render your program platform specific, because the dynamic libraries containing the native methods are platform specific. In addition, the

use of native methods may render your program specific to a particular implementation of the Java Platform. One native method interface--the *Java Native Interface*, or *JNI*--enables native methods to work with any Java Platform implementation on a particular host computer. Vendors of the Java Platform, however, are not necessarily required to support JNI. They may provide their own proprietary native method interfaces in addition to (or depending on their contract, in place of) JNI.

Java gives you a choice. If you want to access resources of a particular host that are unavailable through the Java API, you can write a platform-specific Java program that calls native methods. If you want to keep your program platform independent, however, you must access the system resources of the underlying operating system only through the Java API.

3.3.3 The Java Class File

The Java class file helps make Java suitable for networks mainly in the areas of platform-independence and network-mobility. Its role in platform independence is serving as a binary form for Java programs that is expected by the Java virtual machine but independent of underlying host platforms. This approach breaks with the tradition followed by languages such as C or C++. Programs written in these languages are most often compiled and linked into a single binary executable file specific to a particular hardware platform and operating system. In general, a binary executable file for one platform won't work on another. The Java class file, by contrast, is a binary file that can be run on any hardware platform and operating system that hosts the Java virtual machine.

3.3.4 The Java API

The Java API helps make Java suitable for networks through its support for platform independence and security. The Java API is set of runtime libraries that give you a standard way to access the system resources of a host computer. When you write a Java program, you assume the class files of the Java API will be available at any Java virtual machine that may ever have the privilege of running your program. This is a relatively safe assumption because the Java virtual machine and the class files for the Java API are the required components of any implementation of the Java Platform. When you run a Java program, the virtual machine loads the Java API class files that are referred to by your program's class files. The combination of all loaded class files (from

your program and from the Java API) and any loaded dynamic libraries (containing native methods) constitute the full program executed by the Java virtual machine.

The class files of the Java API are inherently specific to the host platform. The API's functionality must be implemented expressly for a particular platform before that platform can host Java programs. To access the native resources of the host, the Java API calls native methods. As you can see in Figure 3.3.4.1, the class files of the Java API invoke native methods so your Java program doesn't have to. In this manner, the Java API's class files provide a Java program with a standard, platform-independent interface to the underlying host. To the Java program, the Java API looks the same and behaves predictably no matter what platform happens to be underneath. Precisely because the Java virtual machine and Java API are implemented specifically for each particular host platform, Java programs themselves can be platform independent.

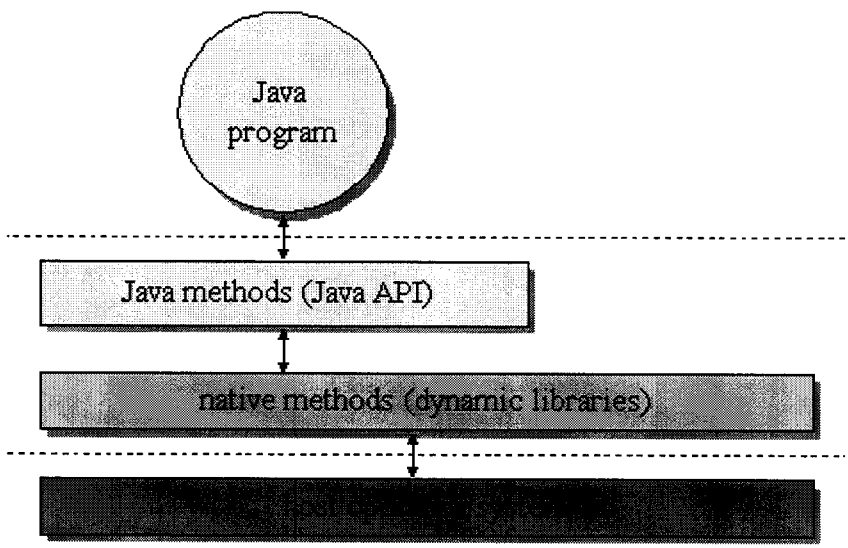


Figure 3.3.4.1 A platform-independent Java program.

The internal design of the Java API is also geared towards platform independence. For example, the graphical user interface libraries of the Java API, the Abstract Windows Toolkit (or AWT) and Swing, are designed to facilitate the creation of user interfaces that work on all platforms. Creating platform- independent user interfaces is inherently difficult, given that the native look and feel of user interfaces vary greatly from one platform to another. The AWT library's architecture does not coerce implementations of the Java API to give Java programs a user interface that looks exactly the same everywhere. Instead, it encourages implementations to adopt the look

and feel of the underlying platform. The Swing library offers even more flexibility -- enabling the look and feel to be chosen by the programmer. Also, because the size of fonts, buttons, and other user interface components will vary from platform to platform, the AWT and Swing include *layout managers* to position the elements of a window or dialog box at run- time. Rather than forcing you to indicate exact X and Y coordinate for the various elements that constitute, say, a dialog box, the layout manager positions them when your dialog box is displayed. With the aim of making the dialog look its best on each platform, the layout manager will very likely position the dialog box elements slightly differently on different platforms. In these ways and many others, the internal architecture of the Java API is aimed at facilitating the platform independence of the Java programs that use it.

In addition to facilitating platform independence, the Java API contributes to Java's security model. The methods of the Java API, before they perform any action could potentially be harmful (such as writing to the local disk), check for permission. In Java releases prior to 1.2, the methods of the Java API checked permission by querying the *security manager*. The security manager is a special object that defines a custom security policy for the application. A security manager could, for example, forbid access to the local disk. If the application requested a local disk write by invoking a method from the pre-1.2 Java API, that method would first check with the security manager. Upon learning from the security manager that disk access is forbidden, the Java API would refuse to perform the write. In Java 1.2, the job of the security manager was taken over by the *access controller*, a class that performs stack inspection to determine whether the operation should be allowed. (For backwards compatibility, the security manager still exists in Java 1.2.) By enforcing the security policy established by the security manager and access controller, the Java API helps to establish a safe environment in which you can run potentially unsafe code.

3.4 JAVA SWING

Swing is a set of classes that provides more powerful and flexible components than are possible with the AWT. The familiar components, such as buttons, check boxes, and labels, swing supplies several exciting additions, including tabbed panes, scroll panes, trees and tables.

Familiar components such as buttons have more capabilities in swing. Swing components are not implemented by platform – specific code. Instead, they are written entirely in JAVA and, therefore, are platform-independent.

3.4.1 Javax.Swing package

The Swing API has been included in the Java 2 platform. JFC is short for Java Foundation Classes, which encompass a group of features for building graphical user interfaces GUI. Use Swing components in building a user interface. First we examine the simplest Swing application you can write. Then we present several progressively complicated examples of creating user interfaces using components in the javax.swing package. We cover several Swing components, such as buttons, labels, and text areas.

javax.swing	Provides a set of "lightweight" (all-Java language) components that, to the maximum degree possible, work the same on all platforms.
javax.swing.border	Provides classes and interface for drawing specialized borders around a Swing component.
javax.swing.colorchooser	Contains classes and interfaces used by the JColorChooser component.
javax.swing.event	Provides for events fired by Swing components.
javax.swing.filechooser	Contains classes and interfaces used by the JfileChooser component.

3.4.2 Compiling and Running Swing Programs

The swing programs how to compile and run a Swing application. Here are the steps you need to follow,

Install the latest release of the Java 2 platform for swing application and Swing program is in a single file. When you save this file, you must match the spelling and capitalization of its name exactly for saving swing program.

The swing program is compiled using the compiler in a recent release of the Java platform. Once you can update your SDK, you should be able to use the programs in this trail without changes. Another common mistake is installing the Java Runtime Environment (JRE) and not the full Software Development Kit (SDK) needed to compile these programs.

The swing program can compile successfully, you can run it. Assuming that your program uses a standard look and feel, you can use the interpreter to run the program without adding anything to your class path. Programs that use a nonstandard look and feel or any other nonstandard code package, you must make sure that the necessary classes are in the class path.

Java Web Start is a technology that simplifies the distribution of applications. The first time you launch an application, Java Web Start automatically downloads all necessary files. It then caches them on your computer so the application can be quickly relaunched from a desktop shortcut or from a Web browser. Java Web Start checks for updates each time you run a remote application and download any updated files automatically.

3.4.3 Swing Application

Swing program each time the user clicks the button (JButton), the label (JLabel) is updated. Swing allows you to specify which look and feel your program uses--Java, GTK+, Windows, and so on.

The Swing Application GUI contains a button and a label. The first creates the button. The user can use to simulate a click of the button an event handler for the button click. How to set up buttons check boxes and radio buttons, as they all inherit from the AbstractButton class.

Swing Application groups its label and button in a container (a JPanel) before adding the components to the frame. The first line creates the container and assigns it a layout manager--an object that determines the size and position of each component added to the container every component having the same size.

The code creates and sets a border that provides some empty space around the containers. Borders are a feature that JPanel inherits from the JComponent class. A Border JComponent instead of using by one or more JComponents object to paint the component's edges.

3.4.4 The Swing event model

In the Swing event model a component can initiate ("fire") an event. A distinct class represents each type ofevent. When an event is fired, it is received by one or more "listeners," which act on that event. Thus, the source of an event and the place where the event is handled can be separate. Since you typically use Swing components as they are, but need to write code that is called when the components receive an event, this is an excellent example of the separation of interface and implementation. Each event listener is an object of a class that implements a particular type of listener **interface**. So as a programmer, all you do is create a listener object and register it with the component that's firing the event.

This registration is performed by calling an **addXXXListener( )** method in the event-firing component, in which "**XXX**" represents the type of event listened for. You can easily know what types of events can be handled by noticing the names of the "addListener" methods, and if you try to listen for the wrong events you'll discover your mistake at compile time.

Act that results in the event	Listener type
User clicks a button, presses Return while typing in a text field, or chooses a menu item	ActionListener
User closes a frame (main window)	WindowListener
User presses a mouse button while the cursor is over a component	MouseListener
User moves the mouse over a component	MouseMotionListener
Component becomes visible	ComponentListener
Component gets the keyboard focus	FocusListener
Table or list selection changes	ListSelectionListener

You can create a global listener class, but this is a situation in which inner classes tend to be quite useful, not only because they provide a logical grouping of your listener classes inside the UI or business logic classes they are serving, but because (as you shall see later) the fact that an inner class object keeps a reference to its parent object provides a nice way to call across class and subsystem boundaries. All the examples so far in this chapter have been using the Swing event model, but the remainder of this section will fill out the details of that model. Event and listener types

All Swing components include **addXXXListener( )** and **removeXXXListener( )** methods so that the appropriate types of listeners can be added and removed from each component. You'll notice that the "**XXX**" in each case also represents the argument for the method, for example: **addMyListener(MyListener m)**. The following table includes the basic associated events, listeners, and methods, along with the basic components that support those particular events by providing the **addXXXListener( )** and **removeXXXListener( )** methods.

You can see that each type of component supports only certain types of events. It turns out to be rather difficult to look up all the events supported by each component. A simpler approach is to modify the **ShowMethodsClean.java** all the event listeners supported by any Swing component that you enter. There is no **MouseMotionEvent** even though it seems like there ought to be. Clicking and motion is combined into **MouseEvent**, so this second

appearance of **MouseEvent** in the table is not an error. whose names match a keyword that you provide. The magic of this is that it can automatically show you *all* the methods for a class without forcing you to walk up the inheritance hierarchy examining the base classes at each level.

Thus, it provides a valuable timesaving tool for programming: because the names of most Java methods are made nicely verbose and descriptive, you can search for the method names that contain a particular word of interest. When you find what you think you're looking for, check the online documentation.

The GUI contains a **JTextField name** in which you can enter the Swing class name you want to look up. The results are displayed in a **JTextArea**. You'll notice that there are no buttons or other components by which to indicate that you want the search to begin. That's because an ActionListener monitors the JtextField. Whenever you make a change and press ENTER, the list is immediately updated. If the text isn't empty, it is used inside Class.forName( ) to try to look up the class.

If the name is incorrect, **Class.forName( )** will fail, which means that it throws an exception. This is trapped and the **JTextArea** is set to "No match." But if you type in a correct name (capitalization counts), **Class.forName( )** is successful and **getMethods( )** will return an array of **Method** objects. Each of the objects in the array is turned into a **String** via **toString( )** (this produces the complete method signature) and added to **n**, a **String** array. The array **n** is a member of **class ShowAddListeners** and is used in updating the display whenever **reDisplay( )** is called. **reDisplay( )** creates an array of **String** called **rs** (for "result set"). The result set is conditionally copied from the **String**s in **n** that contain "add" and "Listener." **indexOf( )** and **substring( )** are then used to remove the qualifiers like **public**, **static**, etc.

Finally, **StripQualifiers.strip( )** removes the extra name qualifiers. This program is a convenient way to investigate the capabilities of a Swing component. Once you know which events a particular component supports, you don't need to look anything up to react to that event. You simply:

1. Take the name of the event class and remove the word "**Event**." Add the word "**Listener**" to what remains. This is the listener interface you must implement in your inner class.

2. Implement the interface above and write out the methods for the events you want to capture. For example, you might be looking for mouse movements, so you write code for the **mouseMoved( )** method of the **MouseMotionListener** interface. (You must implement the other methods, of course, but there's often a shortcut for that which you'll see soon.)

Create an object of the listener class in Step 2. Register it with your component with the method produced by prefixing "**add**" to your listener name. For example, **addMouseMotionListener( )**.

This is not an exhaustive listing, partly because the event model allows you to create your own event types and associated listeners. Thus, you'll regularly come across libraries that have invented their own events, and the knowledge gained in this chapter will allow you to figure out how to use these events.

Using listener adapters for simplicity In the table above, you can see that some listener interfaces have only one method. These are trivial to implement since you'll implement them only when you want to write that particular method. However, the listener interfaces that have multiple methods can be less pleasant to use. For example, something you must always do when creating an application is provide a **WindowListener** to the **JFrame** so that when you get the windowClosing( ) event you can call **System.exit( )** to exit the application. But since **WindowListener** is an **interface**, you must implement all of the other methods even if they don't do anything. This can be annoying. To solve the problem, some (but not all) of the listener interfaces that have more than one method are provided with *adapters*, the names of which you can see in the table above. Each adapter provides default empty methods for each of the interface methods. Then all you need to do is inherit from the adapter and override only the methods you need to change. For example, the typical **WindowListener** you'll use looks like this (remember that this has been wrapped inside the **Console** class in **com.bruceeckel.swing**):

This doesn't work, but it will drive you crazy trying to figure out why, since everything will compile and run fine—except that closing the window won't exit the program. Can you see the problem? It's in the name of the method: **WindowClosing( )** instead of **windowClosing( )**. A simple slip in capitalization results in the addition of a completely new method. However, this is not the method that's called when the window is closing, so you don't get the desired results.

Despite the inconvenience, an **interface** will guarantee that the methods are properly implemented. Tracking multiple events to prove to yourself that these events are in fact being fired, and as an interesting experiment, it's worth creating an applet that tracks extra behavior in a **JButton** (other than just whether it's pressed or not). This example also shows you how to inherit your own button object because that's what is used as the target of all the events of interest. To do so, you can just inherit from **JButton.9** The **MyButton** class is an inner class of **TrackEvent**, so **MyButton** can reach into the parent window and manipulate its text fields, which is what's necessary to be able to write the status information into the fields of the parent. Of course this is a limited solution, since **myButton** can be 9 In Java 1.0/1.1 you could *not* usefully inherit from the button object. This was only on of numerous fundamental design flaws.

CHAPTER 4

SYSTEM DESIGN

4.1 DESIGN PROCESS

4.1.1 Input Design

Input design is the process of converting user-oriented input to computer-based format. The goal of input data design is to make data entry as easy, logical and free from errors as possible. Inputs are raw data that are acceptable by the system and processed it to produce the output. It also includes determining media, method of input, speed of capture and entry into the system. Errors entered are controlled by the input design.

Four objectives guiding the design of input focus on:

1 Controlling the amount of input required.

2 Avoid delay.

3 Controlling errors.

4 Keeping the steps simple.

Main screen is designed with Menu bar, Tool box, Property window, Design frame and Code window. The skeleton structure of the form design required by the user is given as the input. Components can be added to the design frame from the Toolbox window and the user required form design can be designed.

This design will be saved to the disk as a Java file. The file can be edited with other source editors or text editors.

4.1.2 Output Design

After designing and saving the form design required by the user, code is created. The errors will be displayed. If the source file generates no errors, the program can be executed.

In this project, the Design frame acts as an interface. So the Design frame must be double clicked to get the code on the code window as an output.

Code window is mainly used to generate the code for each and every component in the design frame. Whenever the changes made in the Property Window or in the Design Frame it is reflected in the code window. Modification in the program can be performed in the code window. This coding helps the user to compile & execute easily

The user can simply insert their code if they want to perform any actions using Java listeners options, which we have provided through the project. After the insertion of the user required code, it can be compiled and executed to get the desired output. Thus the project aims at reducing the user difficulties, error, and time.

4.2 COMPONENTS AND METHODS

This project is used to work JAVA in very efficient way. This project contains the following phases,

- Designing phase
- Component adding phase
- Runtime Property modification phase
- Automatic code generation phase

Designing Phase:

This Phase contains the following components,

1. Menu bar Window
2. Toolbar Window
3. Toolbox Window

4. Main Screen Window
5. Code Window
6. Property Window

1. Menu bar Window: This component contains the following menus: File, Edit, View, Option, and Help

2. Toolbar Window: It has all the tools that are performed the same function as menu items do in the Menu bar Window.

3. Toolbox Window: The Toolbox Window is very helpful for designing and generates the code automatically. The Toolbox Window contains,

- JButton
- JLabel
- JTextField
- JtextAarea
- JCheckbox
- JRadioButton
- JList
- JComboBox
- JTabbedPane
- JScrollBar
- JSlider
- JProgrerssBar
- JMenu bar
- JMenu
- JmenuItem

- Separator
- JPanel
- Flow Layout
- Grid Layout
- Border Layout
- Null Layout

JButton:

This Button object creates a labeled button. The application can cause some action to happen when the button is pressed.

The gesture of clicking on a button with the mouse is associated with one instance of ActionEvent, which is sent out when the mouse is both pressed and released over the button.

When a button is pressed and released, AWT sends an instance of Action Event to the button, by calling process Event on the button. The button's process Event method receives all events for the button; it passes an action event along by calling its own processActionEvent method.

JLabel:

A Label object is a component for placing text in a container. A label displays a single line of read-only text. The text can be changed by the application, but a user cannot edit it directly.

JText field:

A Text Field object is a text component that allows for the editing of a single line of text. Every time the user types a key in the text field, one or more key events are sent to the text field. It is also possible to fire an Action Event.

The JTextField component displays a single line of editable text, using a single font and a single color. Horizontal alignment either LEFT, CENTER, or RIGHT can be specified for a text field's text. By default, pressing the Enter key

(KeyEvent.VK_ENTER) while a text field has focus causes the field to fire action event. The JTextField class is nearly source compatible with the java.awt.TextField class. The properties maintained by the JTextField class are actionCommmand, columns, horizontalAlignment, horizontalVisibility, scrollOffset.

JText area:

A Text Area object is a multi-line region that displays text. It can be set to allow editing or to be read-only. Every time the user types a key in the text field, one or more key events are sent to the text field. It is also possible to fire an Action Event.

The properties maintained by the JTextArea class are columns, lineCount, lineWrap, rows, tabSize, wrapStyleWord.

JCheckbox:

A check box object is a graphical component that can be in either an "on" (true) or "off" (false) state. Clicking on a check box changes its state from "on" to "off," or from "off" to "on."

JRadio Button:

Several check boxes can be grouped together under the control of a single object, using the Checkbox Group class. In a check box group, at most one button can be in the "on" state at any given time. Clicking on a check box to turn it on, forces any other check box in the same group that is on into the "off" state.

JList:

The List object presents the user with a scrolling list of text items. The list can be set up so that the user can choose either one item or multiple items. Selecting an item causes any other selected item to be automatically deselected.

When an item is selected or deselected by the user, AWT sends an instance of Item Event to the list. When the user double-clicks on an item in a scrolling list, AWT sends an instance of Action Event to the list following the item event.

For multiple-selection scrolling lists, it is considered a better user interface to use an external gesture to trigger the action.

JScrollbar:

The Scrollbar object embodies a scroll bar, a familiar user-interface object. A scroll bar provides a convenient means for allowing a user to select from a range of values. Normally, the user changes the value of the scroll bar by making a gesture with the mouse. When the user changes the value of the scroll bar, the scroll bar receives an instance of Adjustment Event.

JMenu bar:

JMenu:

A Menu object is a pull-down menu component that is deployed from a menu bar.A menu can optionally be a tear-off menu. A tear-off menu can be opened and dragged away from its parent menu bar or menu. It remains on the screen after the mouse button has been released. The mechanism for tearing off a menu is platform dependent, since the look and feel of the tear-off menu is determined by its peer. On platforms that do not support tear-off menus, the tear-off property is ignored.

Each item in a menu must belong to the Menu Item class. It can be an instance of Menu Item, a submenu (an instance of Menu), or a check box (an instance of CheckboxMenuItem).

JMenu Item:

All items in a menu must belong to the class Menu Item, or one of its subclasses. The default Menu Item object embodies a simple labeled menu item. When a menu item is selected, AWT sends an action event to the menu item. Since the event is an instance of Action Event, the process Event method examines the event and passes it along to processActionEvent. The latter method redirects the event to any Action Listener objects that have registered an interest in action events generated by this menu item.

JPanel:

Panel object is the simplest container class. A panel provides space in which an application can attach any other component, including other panels. The default layout manager for a panel is the Flow Layout layout manager.

Flow Layout:

A flow layout arranges components in a left-to-right flow, much like lines of text in a paragraph. Each line is centered.

Grid Layout:

The Grid Layout object is a layout manager that lays out a container's components in a rectangular grid. The container is divided into equal-sized rectangles, and one component is placed in each rectangle.

Border Layout:

A border layout lays out a container, arranging and resizing its components to fit in five regions: north, south, east, west, and center. Each region may contain no more than one component, and is identified by a corresponding constant: NORTH, SOUTH, EAST, WEST, and CENTER.

Null Layout:

A null layout arranges components in a user-defined position.

4. Main Screen Window: Main screen is mainly used for design process. Components can be added to the main screen from the Toolbox window.

Select a component from the Toolbox by a single click and click anywhere in the main screen then components will be placed on the position with default size.

After placing the components in the main screen, we can drag and drop the component in the frame.

If necessary the corresponding component properties can be modified.

5. Property Window: The property window is used to change the properties of each component. The Properties listed in the property window are:

- Caption
- Name
- Background
- Enable
- Foreground
- Font type
- Font style
- Font size
- Height
- X Pos
- Y Pos
- Visible
- Width

Caption: The component caption must be changed using the caption property.

Name: The component object must be changed using the name property.

Background: The background color for each component should be changed using this property.

Enable: Set the enabled property using this property.

Font type: Set the font name for each component using this property.

Font style: Set the font style (Plain, Bold, Italic) for each component using this property.

Font size: Set the size of the font for each component using this property.

Foreground: The foreground color for each component should be changed using this property.

Height: Set the height of the component using this property.

X Pos: The component x position should be modified using this property.

Y Pos: The Component y position should be modified using this property.

Visible: Set the visible property using this property.

Width: Set the width of the component using this property.

6. **Code Window:** This window is mainly used for generate the code for each and every component in the main screen. When new Frame window and dialog box are added, code is generated. Whenever the changes made in the Property Window or in the Main Screen it is reflected in the code window. Modification in the program can be performed in the code window. This coding helps the user to compile & execute easily.

Component Adding Phase:

Here we have written coding for merging the selected tools into the design frame. This phase is just connecting the tool window were tools such as JButton , JLabel, JTextField, JComboBox, etc.., are placed here. This tools can be put in the Design frame by Drag and drop method.

Components including for the design window should be created. To create every component we have used a class name AddingComponentEvent. This class is inherited with MouseListener and MouseMotionListener. MouseListener is used to sense various mouse action such as pressed, clicked, entered etc. Similarly MouseMotionListener is used to sense

Mouse drag, MouseMoveEvents. This is useful the AddingComponentEvent class for initializing Events for every Components.

AddingComponentEvent contains objects of every Component. These objects are private since it should be created within the class. MousePressedEvents pass the corresponding X,Y position of the Mouse position.

AddingComponentEvents also checks for creating coding for particular Components. It listen the MouseEvent for double click and when Event is triggered codes will be generated.

Object is used for identifying particular Components. This object can be retrieved using getSource( ) function.

JMenu has a special character, that it should be placed once in design window. So design window is checked while adding JMenu to the design window. Presents of JMenu in design window can be verified using getJMenuBar.

ComponentMotionEvent class links the particular Components and property window. It plays the works of displaying the property of selected window. For this the selected Component property should be passed into the property window.

Here getSource( ) is used to get object of particular Component and this object is passed into the property window using setComponentproperty method. This class Listens to the dragging event of any component.

Runtime Property Modification Phase:

This phase contain codes for modifying the property of the all the components during runtime. By making this options make user more flexible and user friendly.

Property that can be modified during runtime are name, caption, background color, foreground color, font type, font size, font style, height, width, x-position, y-position, tool tip, border. Modification of every component is carried out under particular listener.

Property is designed in two methods, they are

- Text Field
- ComboBox

Text field property is used to change name, caption, height, width, top etc..

Combo box is used to modifying character such as background color, foreground color, font type, font style, font size, border.

ActionListner is a method which looks for action on all components. setText( ) method is used to change the property of name.

Caption property is present in JLlabel, JMenu, JMenuitem, JToggleButton, JRadioButton, JCheckBox, JTextField, JTextArea, other components such as progress bar doesn't have this property. Run time modification in the property of caption is similar to name.

Foreground & background color property can be modified using getBackground(), setBackground( ), setforeground( ), getforeground( ) method. Here some colors are set default for users convince it can be choosing combo box. For choosing combination of colors we used Color Chooser function.

For changing the font type here we include graphics environment class, which loads various font type. Font size is set a maximum of 100. There is various font styles they are bold, italic, bold italic, plain.

Height, width of any component can be obtained through a method called setSize(). Similarly top, left position of an component can be given through setLocation()

Tool tip makes the application more user friendly, because it guide the user at each step. This property can be modified by obtaining the object of particular component through getText( ) method & new tool tip is inserted through a function called setToolTipText( ).

SetComponent property is a class, which is used to display the property of any component when it gets clicked.

Automatic code generation:

Final phase of our project is to generate coding when the user is double clicked on the design frame. Generally as you know JAVA code is in the form of

- Header files
- Class definition
- Declaration
- Constructor
- Main function

For generating code for component, we use append( ) method. for code generation we create a String Buffer. This append( ) method contain a parameter of text which will be replaced on corresponding code window.

Header files of JAVA are included using append( ) method, usually class name is similar to the file name. So we obtain the file name using LastIndexof( ) method for the file name object.

Declaration of all components is done through getComponentCount( ) method. the getComponentCount( ) method list all the declared variable of the container. Then this variable is passed in to declaration generator class where objects are checked for the components and variables are declared there.

To generate code, we create a constructor. Here code should be generated for the initialized components. Here the objects of particular components are passed and the corresponding code will be generated.

The generated code should be permanent so that it should be saved .For saving the code, we link File chooser in to the code.

At last Main method is generated for the compilation of the program. In this method, object is created for the corresponding class.

4.3 SYSTEM ARCHITECTURE

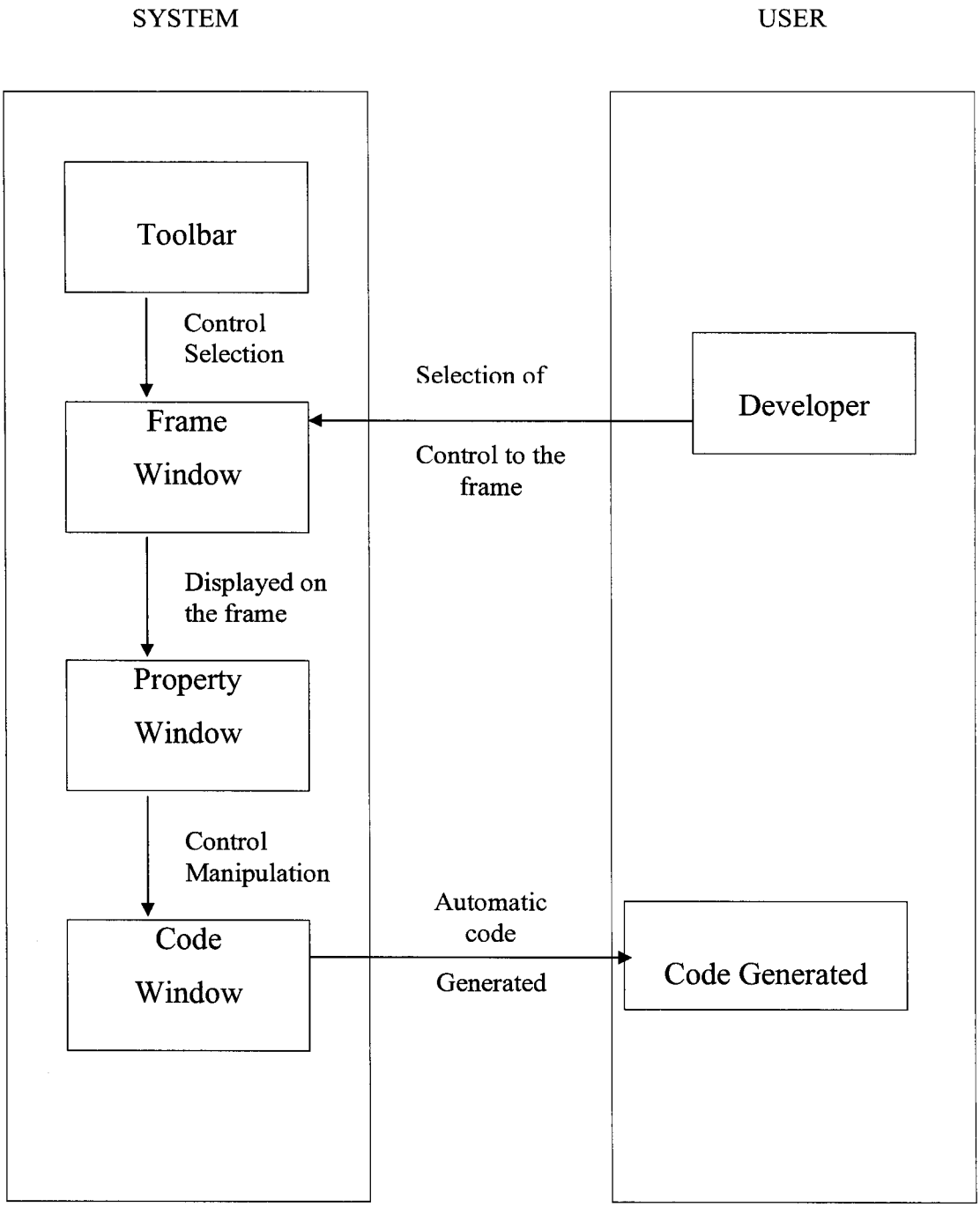


Figure 4.3.1 System Flow Diagram

CHAPTER 5

SYSTEM IMPLEMENTATION

System Implementation is the stage in the project where the theoretical design is turned into a working system. The most critical stage is achieving a successful system and in giving confidence on the new system for the user that it will work efficiently and effectively.

The existing system was long time process. The proposed system was developed using JDK1.4 and Java swings. The existing system caused long time transmission process but the system developed now has a very good user-friendly tool, which has a menu-based interface, graphical interface for the end user.

After coding and testing, the project is to be installed on the necessary system. The executable file is to be created and loaded in the system. Again the code is tested in the installed system. Installing the developed code in system in the form of executable file is implementation.

5.1 PSEUDO CODE

Pseudo Code for designing Tool box

- DECLARE a frame
- INITIALIZE all the components in an array.
- INITIALIZE for loop for the length of the array.
- DECLARE & DEFINE button
- LABEL the button with the string in array.
- COUNT the mouse click on toolbox components.
- If COUNT equals to 2 display the component in design frame.

DECLARE Frame

INITIALIZE array

DO
    DECLARE Button
    DEFINE Button
    LABEL Button
WHILE (arraycount not greater than count)
END

Pseudo code for designing property window

- DECLARE a frame.
- INITIALIZE an array with property names.
- INITIALIZE for loop with the length of the array.
- DECLARE and DEFINE a TEXTFIELD.
- Name each with LABEL.

DECLARE Frame

INITIALIZE array

DO
    DECLARE TextField
    DEFINE TextField
    LABEL TextField
WHILE (arraycount not greater than count)
END

Pseudo code for runtime property modification phase

- SET the DEFAULT properties for the components in design frame.
- GET the user property values
- SET the user property values

GET number of components

FOR all the components

    GET the property values

    SET the property values

END

IF any property change

FOR all the components

GET the property values from user

SET the user values

END

Pseudo code for code generation

- GET the filename from designframe.
- CONSTRUCT class with the filename.
- GET the components from Design Frame.
- FUNCTION to all toolbox components declaration,definition and Properties.

GET new/open file name

CONSTRUCT class with the file name

GET the number of components in the design frame

FOR all the components

CALL FUNCTION to generate code

END

CHAPTER 6

TESTING AND IMPLEMENTATION

6.1 SYSTEM TESTING

Testing is conducted to check whether the design system works properly based on the requirements of the user. A system is tested based on the on-line responses, volume of the transactions recovery from failure and usability.

The main purpose of conducting test is to find errors in the software by examine all the possible loopholes. Testing of the new system has been done successfully in different levels. There are so many testing activities that help to find the error and other quality factors to reach a safer state in the system implementation.

This system was tested against the testing that are listed below

- Acceptance Testing
- Stress Testing
- Unit Testing
- Integration Testing
- String Testing
- Program Testing
- Validation Testing

Acceptance Testing

Acceptance Testing is running the system with live data by the actual user. The software was given with the actual user data and the responsible staff fed the data.

The system is tested with the data and the code is executed for the various types of design inputs.

Stress Testing

The purpose of stress testing is to prove that the newly developed system does not malfunction under peak loads. For stress testing, the system was dumped with large data the result was observed carefully. The system is running under peak loads.

The system is tested by giving the 'n' number of controls and the coding is generated for the above controls and the listener object are created.

Unit Testing

Unit testing is testing the system individually. The new system was tested individually and all the modules work fine. Unit testing was conducted to check the individual working of the system.

The system is tested for the individual module like property window, toolbar, design window and code window.

Integration Testing

Integration Testing is combining all the modules together and tests the system as a whole. All the modules that undergone Unit testing is integrated together to get the whole software as a single module. This Integration testing is a formal procedure that must be executed carefully according to the need. The new system was tested after combining all the individual modules and it works fine.This testing is done for the whole system by giving any type of design and the code is generated automatically.

String Testing

Programs are invariably related to one another and interact in a total system. Each program is tested to see whether it confirms to related programs in the system. Each portion of

the new system was tested against the entire module with both test data and live data.The system is tested by interacting each module with one another. The controls from the toolbar is dragged and placed on the design frame and the consecutive coding is generated.

Program Testing

Program testing check for two types of errors, Syntax error and logical error. The new system was tested against syntax errors and logic errors to check both were not occurred.The project is tested by giving wrong input values and the proper warning messages are generated from the respective control.

Validation Testing

Validation Testing is the testing system from uncovers functional errors. To check whether the functional characteristics confirm to the specification or not. The proposed system was tested to see whether the entire user requirements were attained or not.

The system is tested for the validation also, (i.e.) the controls are dragged on the design frame, and the controls are tested by giving invalid input on the property window and analyzed for the respective output.

6.2 SYSTEM MAINTENANCE

System maintenance refers to the step necessary to install a new system to put into operation. The new system may be totally new replacing an existing manual or automated system or it may be major modification to an existing system. The method of maintenance and time scale to be adopted found out initially. The system is tested properly and at the same time the users are trained in the same procedure. Proper maintenance is essential to provide a reliable system to meet organization requirements.

The Main phase in the system life cycle is the maintenance of the new system. The major part of the maintenance of the system is played by the user who uses the software frequently. Successful maintenance may not guarantee improvement in the organization, but it prevents improper errors. Thus it can be considered as the most crucial stage in achieving a successful new system and in giving the user confidence that the new system will work and be effective.

CHAPTER 7

PERFORMANCE AND LIMITATIONS

7.1 MERITS OF THE SYSTEM

Some of the merits of the system are given bellow,

- All the GUI components in this system can be reused again and again.
- It is faster and more accurate system.
- User-Friendliness is obtained.
- As tool tips and help are available it is easy.
- We can interact with the system.
- Pleasant environment is provided.

7.2 LIMITATIONS OF THE SYSTEM

The limitation of this system is that, even though the system works efficiently with user given data, it does not have a database to store the data. If this system has database it can be applied to many applications like banking, marketing, and sales.

7.3 FUTURE ENHANCEMENTS

- The project can be evolved using SWING, which will provide a look and feel environment through a set of user interface components.
- If it is enhanced with DATABASE SYSTEMS, it can be used in any application where databases are important such as banking, marketing, sales details.
- It can also easily be adapted in NETWORK APPLICATION
- By using all advanced Java concepts, this can be used as a BEAN to construct new systems.

CHAPTER 8

CONCLUSION

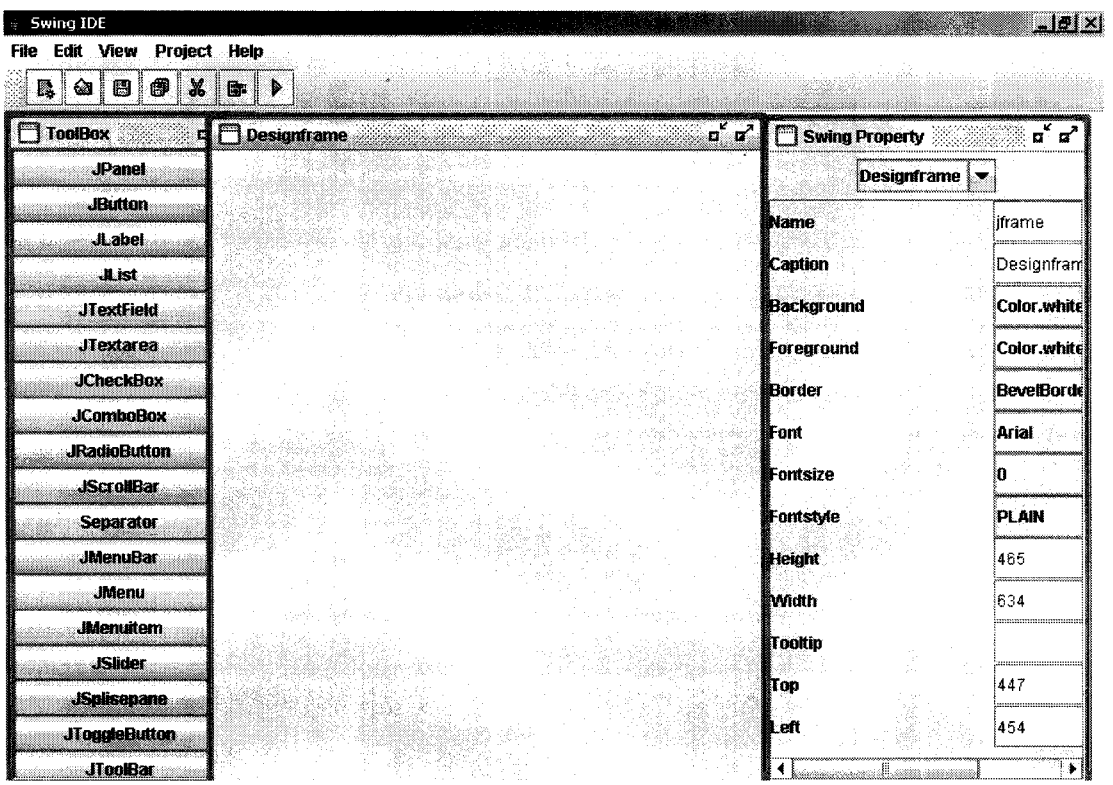
The project titled as **"INTEGRATED DEVELOPMENT ENVIRONMENT FOR JSWING "**is thus provide a user interactive area and components for Java programming. These provides easy way of designing the form and needs a very little number of user coding. Thus this project is developed using Java version jdk1.3. This uses only core Java concepts. The four main modules interlinked to main to get the finished work. User can generate the required program codes even if it is larger in number of lines. This must result in no or very least number of errors. Thus it reduces the difficulties and time. If this is enhanced with database, then this project can be used for any database application.

CHAPTER 9

APPENDICES

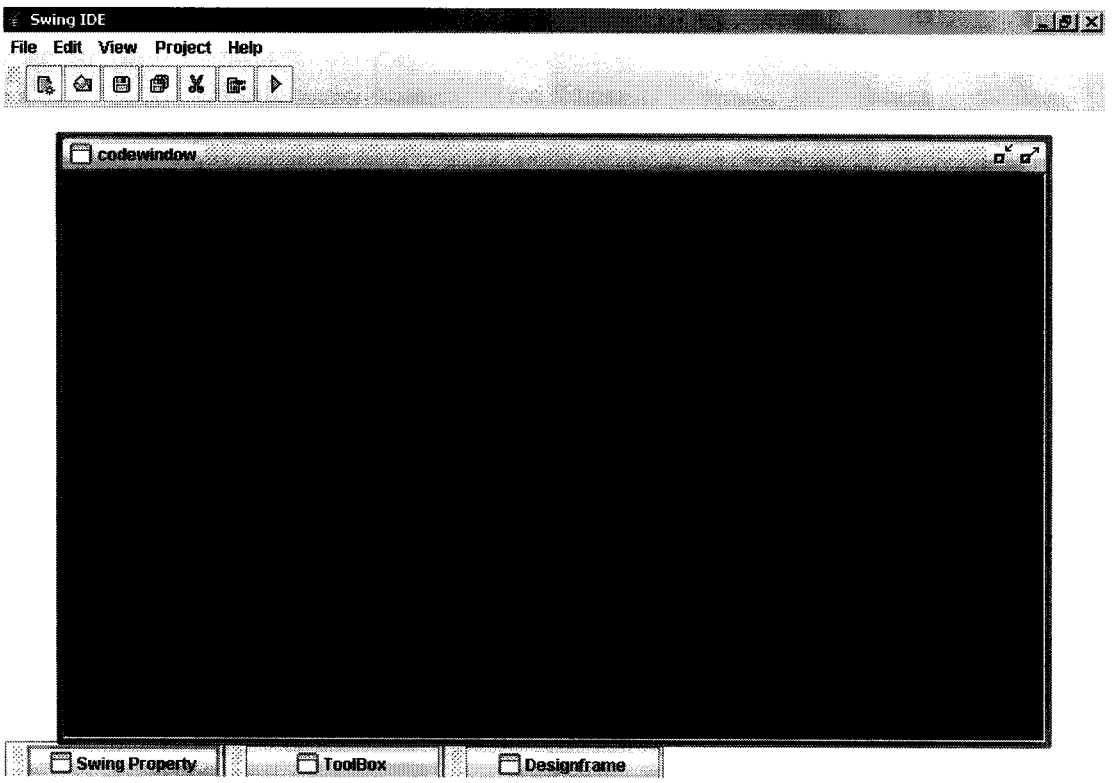
9.1 SAMPLE SCREENS

Output



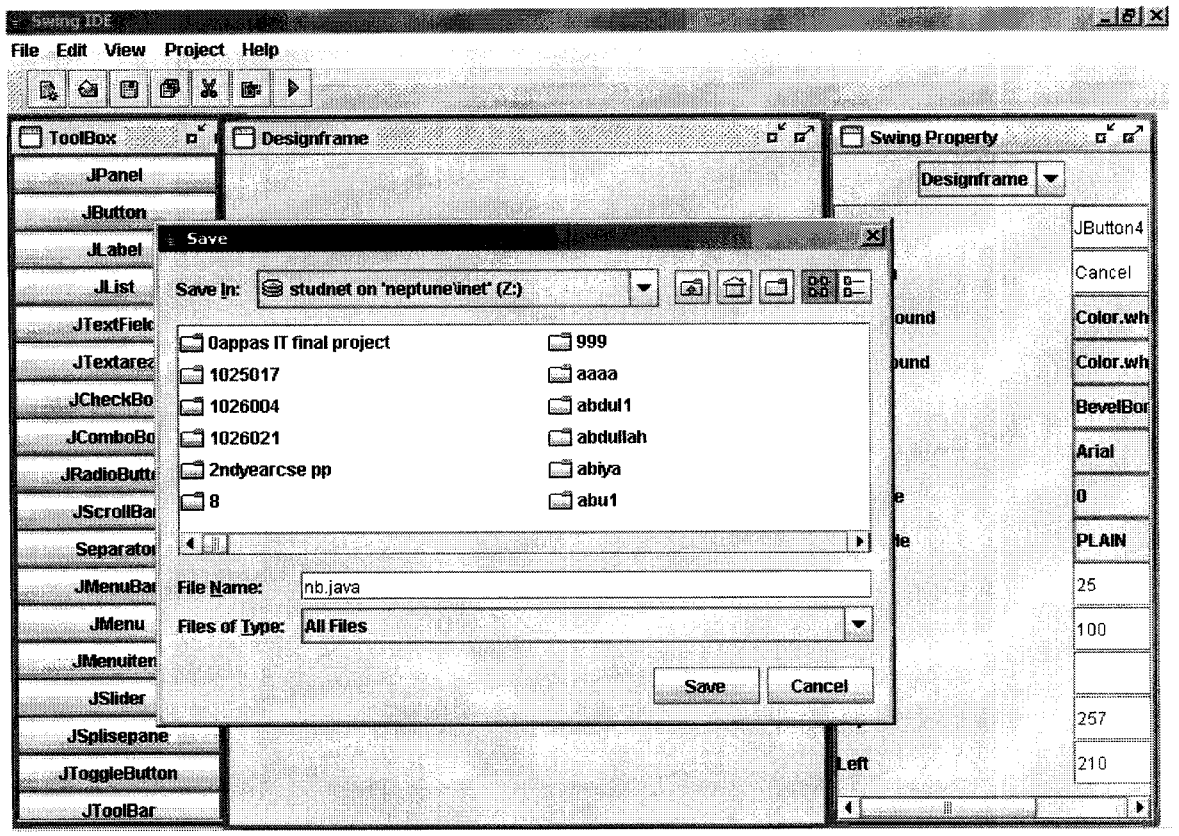
Screen shot for output

Code Window



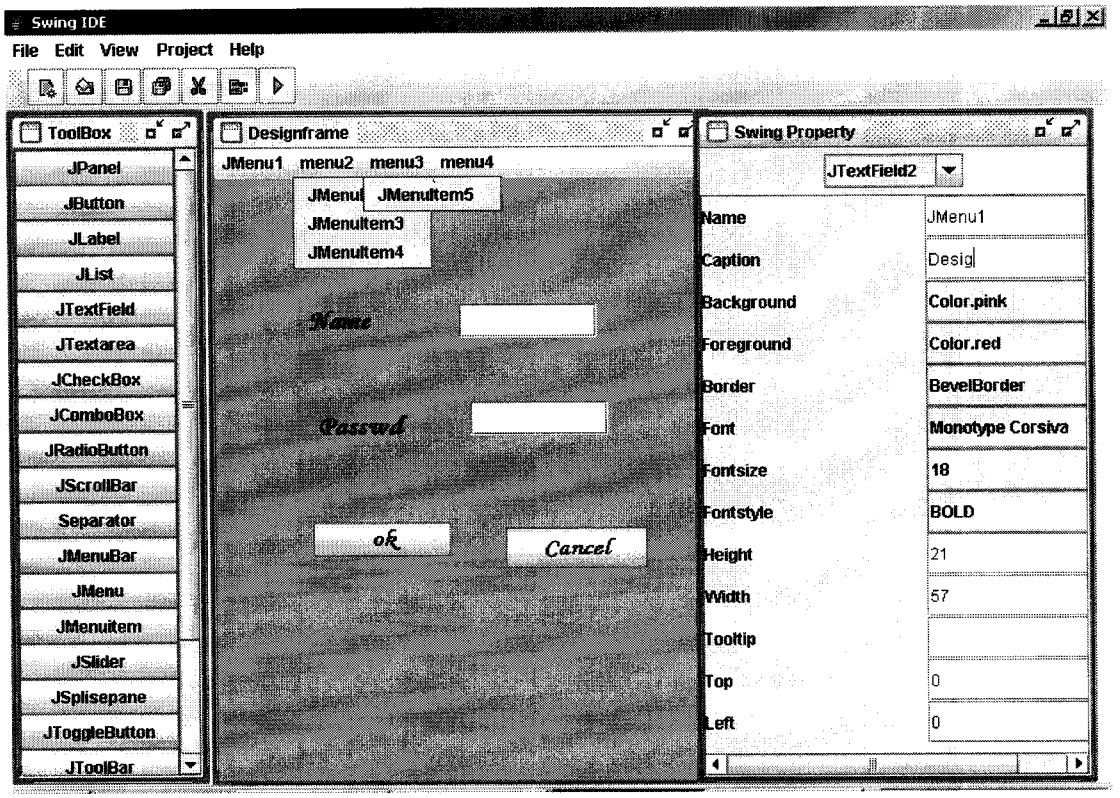
Screen shot for code window

Save Option



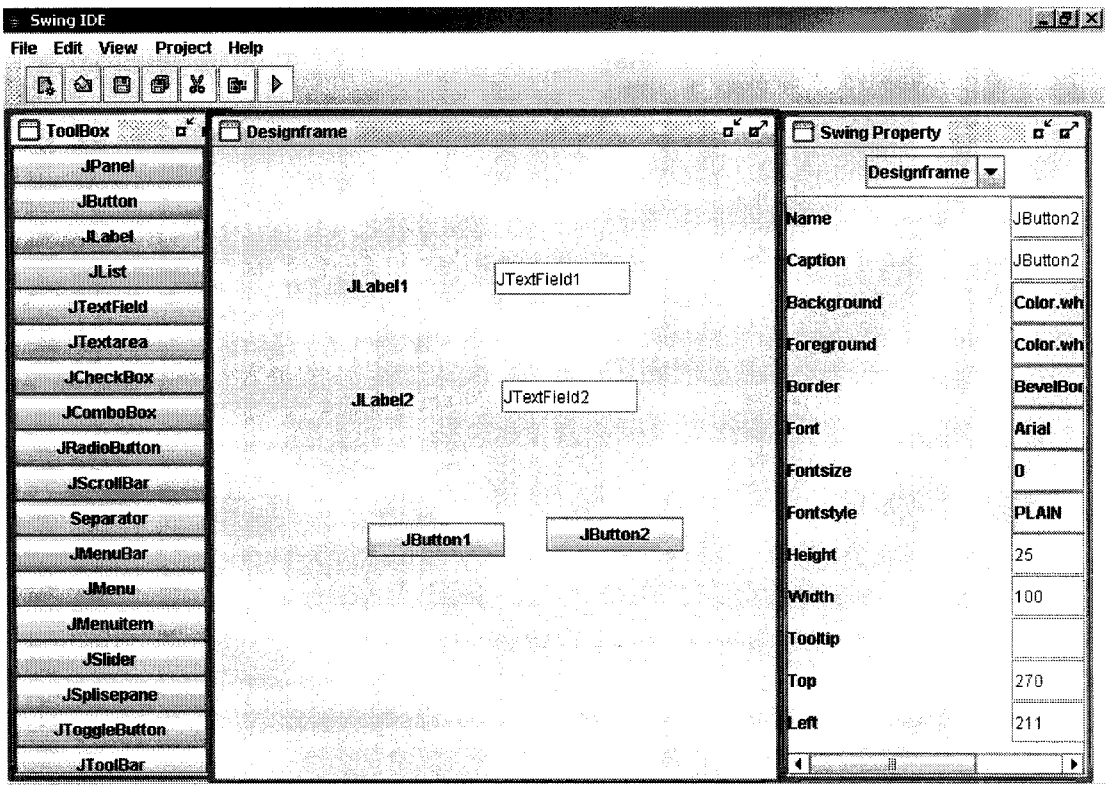
Screen Shot for Save Option

Menubar option



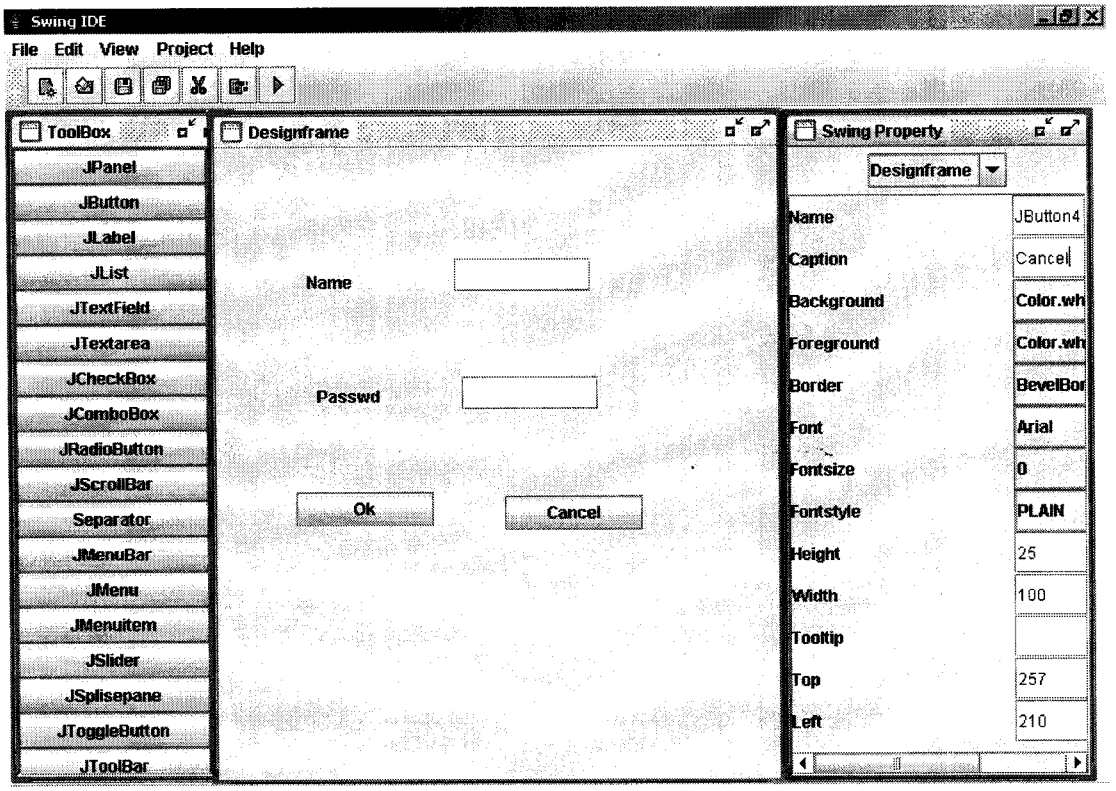
Screen shot for Menubar option

Component Adding



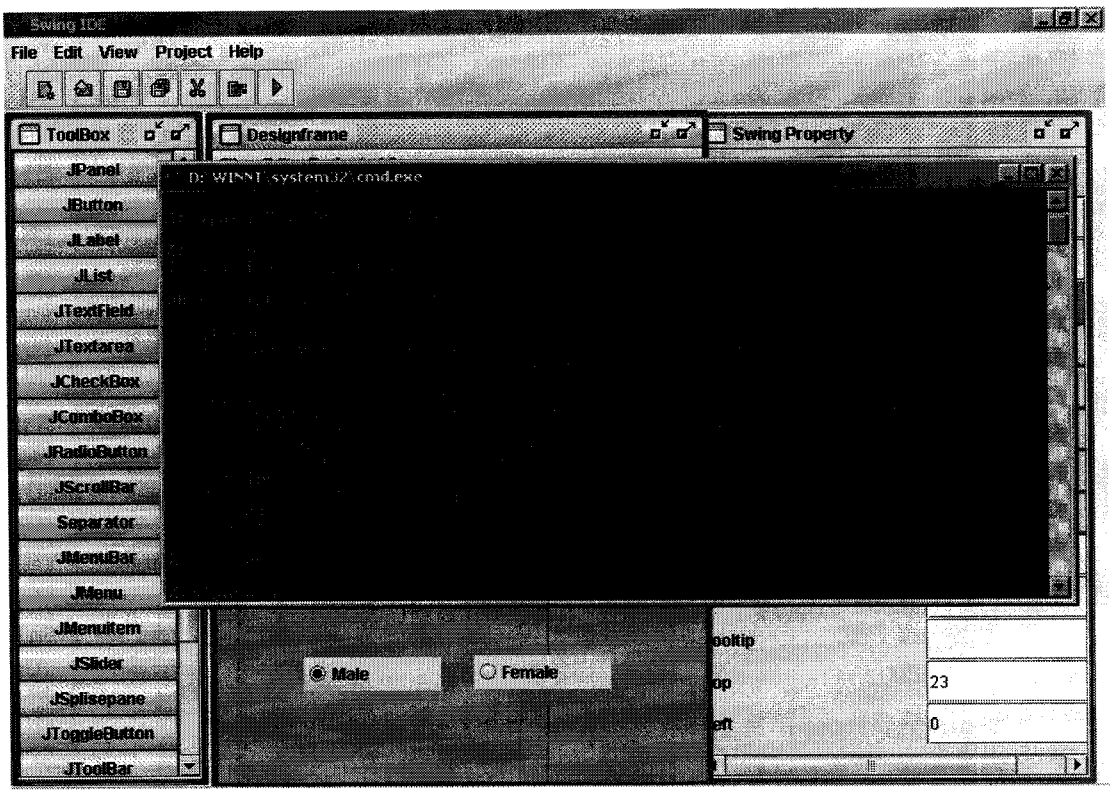
Screen shot for Component Adding

Property Change



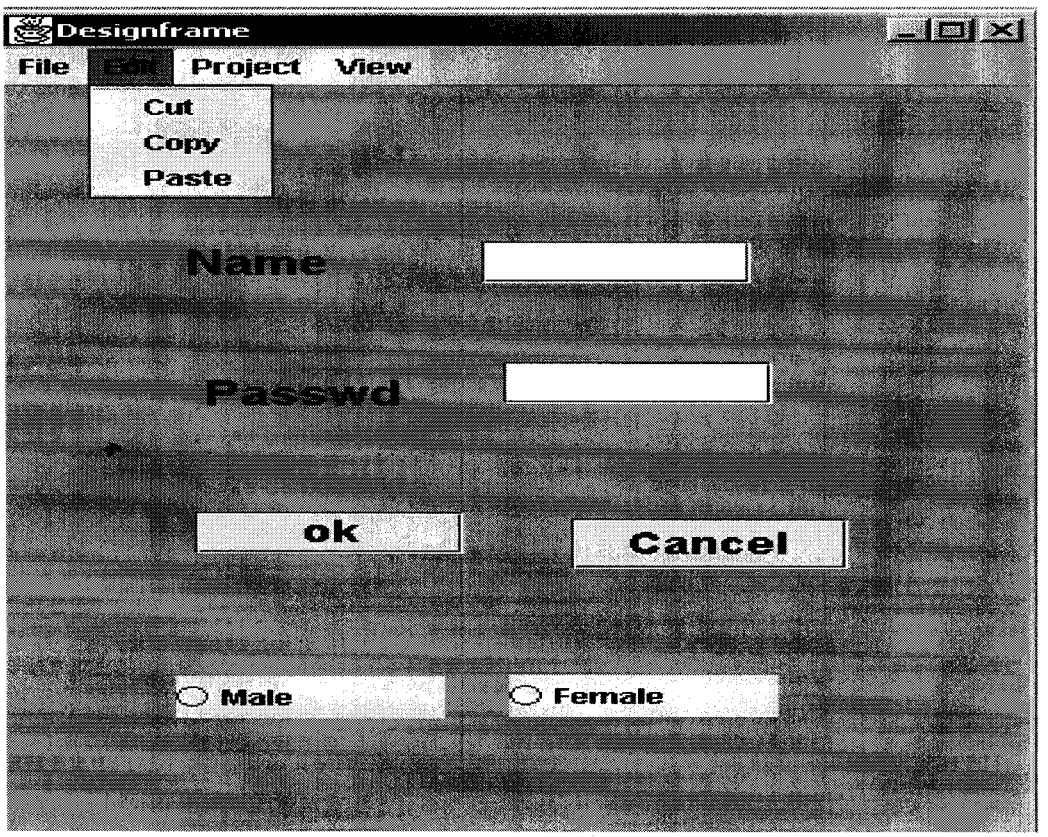
Screen Shot for Property changed

Compilation



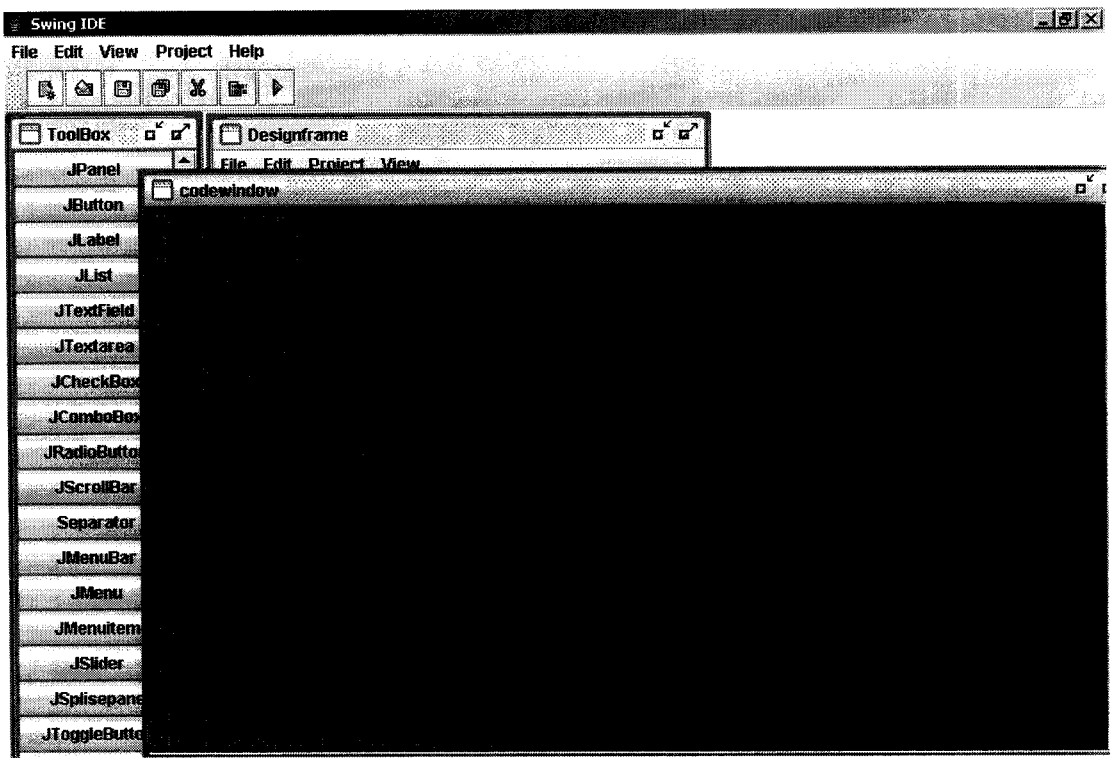
Screen shot for Compilation

Output window



Screen shot for Output Window

Code window



Screen shot for Code window

CHAPTER 10

REFERENCE

The relevant referred books for the project are the following

- The complete reference for Java2 – 3rd Editions by Patrick Naughton & Herbert Schildt.

- Java Foundation Classes by Matthew T. Nelson

- Programming Java-An introduction to programming using Java by Decker, Hirsh field, 2nd edition

- Core Java –Cay S.Horstmann, Gary Cornell.