INTRODUCTION

1.1. INTRODUCTION TO PROJECT

Steganography with Audio, Video & Image is java console application of Steganography happens with Audio, Video & Image File for transferring more secure sensitive data. The Sensitive Data is Encoded with an Audio, Video & Image File and Passed over Insecure Channels to other end Systems. Here we can use any file Format for Encryption and Decryption of Message.

- It is a platform independent application so that it can be used on any operating system.
- There is no need to download and install as other software.
- Provides security for the information which is passed over the net.
- Information is embedded into the Audio, Video & Image file.

1.2. ORGANIZATION PROFILE

SOFTWARE SOLUTIONS

Software Solutions is an IT solution provider for a dynamic environment where business and technology strategies converge. Their approach focuses on new ways of business combining IT innovation and adoption while also leveraging an organization‟s current IT assets. Their work with large global corporations and new products or services and to implement prudent business and technology strategies in today‟s environment.

RANGE OF EXPERTISE INCLUDES

- Software Development Services
- Engineering Services
- Systems Integration
- Customer Relationship Management
- Product Development
- Electronic Commerce
- Consulting
- IT Outsourcing

We apply technology with innovation and responsibility to achieve two broad objectives:

- Effectively address the business issues our customers face today.
- Generate new opportunities that will help them stay ahead in the future.

THIS APPROACH RESTS ON

- A strategy where we architect, integrate and manage technology services and solutions - we call it aim for success.
- A robust offshore development methodology and reduced demand on customer resources.
- A focus on the use of reusable frameworks to provide cost and times benefits.

They combine the best people, processes and technology to achieve excellent results - consistency. We offer customers the advantages of:

SPEED

They understand the importance of timing, of getting there before the competition. A rich portfolio of reusable, modular frameworks helps jump-start projects. Tried and tested methodology ensures that we follow a predictable, low - risk path to achieve results. Our track record is testimony to complex projects delivered within and evens before schedule.

EXPERTISE

Our teams combine cutting edge technology skills with rich domain expertise. What‟s equally important - they share a strong customer orientation that means they actually start by listening to the customer. They‟re focused on coming up with solutions that serve customer requirements today and anticipate future needs.

A FULL SERVICE PORTFOLIO

They offer customers the advantage of being able to Architect, integrate and manage technology services. This means that they can rely on one, fully accountable source instead of trying to integrate disparate multi vendor solutions.

SERVICES

Mortgage is providing its services to companies which are in the field of production, quality control etc with their rich expertise and experience and information technology they are in best position to provide software solutions to distinct business requirements.

1.3. PURPOSE OF THE PROJECT

The purpose of the project is application is to provide the security for the confidential information.

LITERATURE SURVEY

S.No	TITLE	YEAR	OVERVIEW	DRAWBACKS
1.	A New Approach for Data Hiding with LSB Steganography	2015	Authors have done survey on recent achievement of LSB based image Steganography to improve current LSB based steganography methods. They also proposed two new techniques.	Its does not support all formats.
2.	Steganography an Art of Hiding Data	2009	Authors proposed new LSB based method in which common bit pattern is used to hide the data. Proposed system has low data hiding capacity.	Image may corrupted.

SYSTEM ANALYSIS

3.1. INTRODUCTION

After analyzing the requirements of the task to be performed, the next step is to analyze the problem and understand its context. The first activity in the phase is studying the existing system and other is to understand the requirements and domain of the new system. Both the activities are equally important, but the first activity serves as a basis of giving the functional specifications and then successful design of the proposed system. Understanding the properties and requirements of a new system is more difficult and requires creative thinking and understanding of existing running system is also difficult, improper understanding of present system can lead diversion from solution.

3.2. ANALYSIS MODEL

The model that is basically being followed is the SPIRAL MODEL, which states that the phases are organized in a linear order. First of all the feasibility study is done. Once that part is over the requirement analysis and project planning begins. If system exists one and modification and addition of new module is needed, analysis of present system can be used as basic model.

The design starts after the requirement analysis is complete and the coding begins after the design is complete. Once the programming is completed, the testing is done. In this model the sequence of activities performed in a software development project are: -

- Requirement Analysis
- Project Planning
- System design
- Detail design
- Coding
- Unit testing
- System integration & testing

Here the linear ordering of these activities is critical. End of the phase and the output of one phase is the input.

of other phase. The output of each phase is to be consistent with the overall requirement of the system.

SPIRAL MODEL was defined by Barry Boehm in his 1988 article, “A spiral Model of Software Development and Enhancement”. This model was not the first model to discuss iterative development, but it was the first model to explain why the iteration models.

As originally envisioned, the iterations were typically 6 months to 2 years long. Each phase starts with a design goal and ends with a client reviewing the progress thus far. Analysis and engineering efforts are applied at each phase of the project, with an eye toward the end goal of the project.

The following diagram shows how a spiral model acts like:

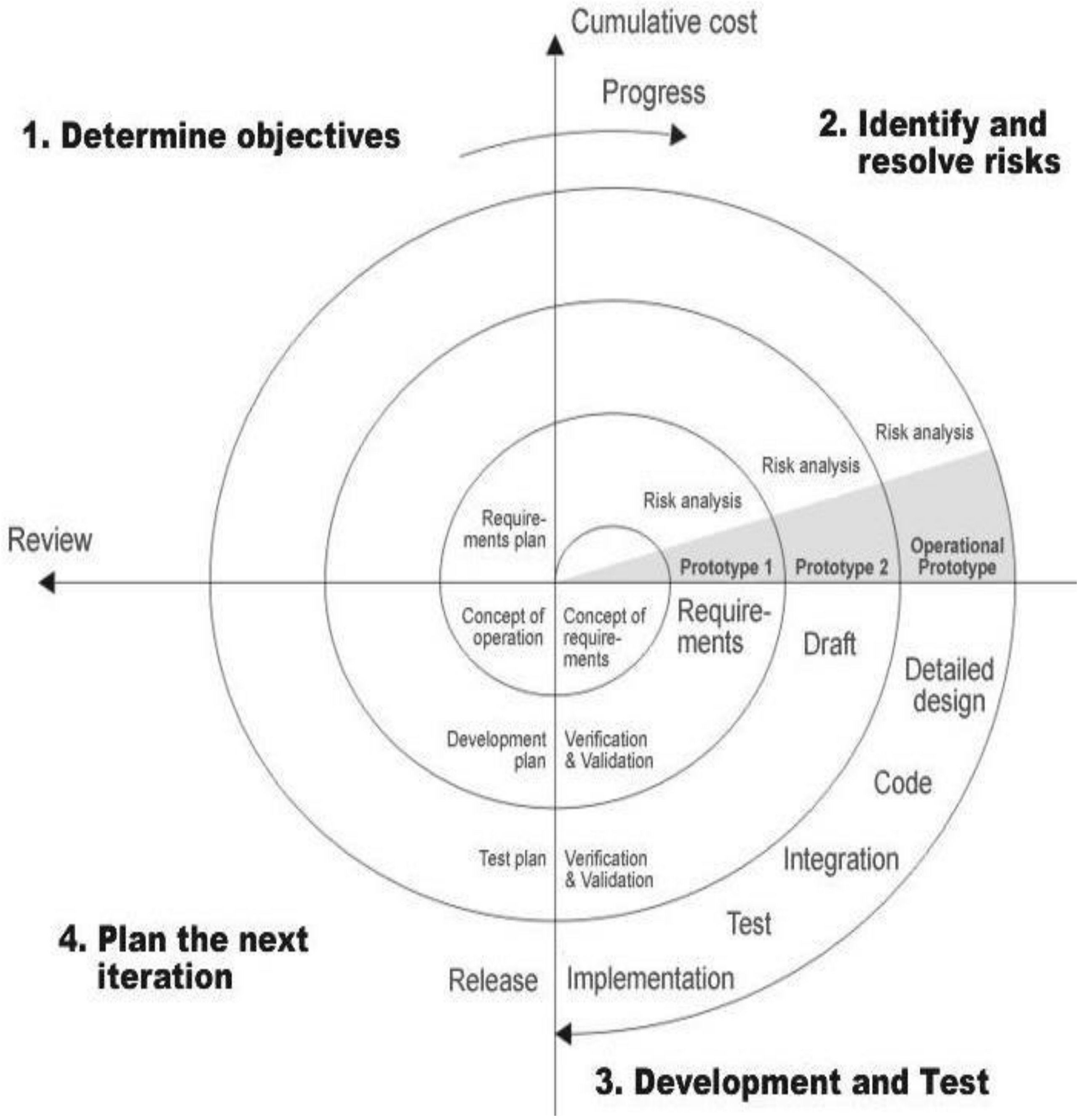


Fig 1.0: Spiral Model

3.3. STUDY OF THE SYSTEM

In the flexibility of uses the interface has been developed a graphics concepts in mind, associated through a browser interface. The GUI‟s at the top level has been categorized as follows

1. Administrative User Interface Design
2. The Operational and Generic User Interface Design

The administrative user interface concentrates on the consistent information that is practically, part of the organizational activities and which needs proper authentication for the data collection. The Interface helps the administration with all the transactional states like data insertion, data deletion, and data updating along with executive data search capabilities. The operational and generic user interface helps the users upon the system in transactions through the existing data and required services. The operational user interface also helps the ordinary users in managing their own information helps the ordinary users in managing their own information in a customized manner as per the assisted flexibilities.

Number of Modules

After careful analysis the system has been identified to have the following modules:

Administrator Module

Pickup Module
Verification Module
Legal Module
Customer Module

Administrator Module

This module is responsible for coordinating the other modules. It allows the administrator create, update/delete and view the banks information and it allows admin to create create/update/delete and view different departments and it can create logins for different employees in each and every department and it can manage loan interest rates of different banks etc. It facilitates to view the new applicant details and assign it to different employees in pickup department initially. It allows the administrator to view customer application finally from verification department and builder details from legal department and then store bank final opinion in this application. It also allows to generate different reports for business analysis. It provides messaging facility for the administrator for communication.

Pickup Module

This module allows the pickup department to view their applications which are assigned to them, collect the documents according to the checklist and forward it to verification department.

Verification Module

This module allows the employees of verification department employees to view the forwarded application from pickup department and check the details as per the documents and forward it to next level.

Legal Module

This module allows the employee of legal departments to check verify legal documents of the builder, verify the check list and then generate the APF no for the builder.

Customer Module

This module allows the customer to view the interest rates of the banks which we are dealing, apply for a loan, check the status of the loan at any point of time and communicate with the administrator if necessary. It allows messaging facility for communication.

3.4. HARDWARE & SOFTWARE SPECIFICATIONS

Hardware Requirements

Client Side			
	Processor	**RAM**	**Disk Space**
Internet Explorer 6.0 **Or Higher**	Computer with a 2.6GHz processor or higher (Pentium processor recommended)	512MB Minimum	Minimum 20 GB
Server Side			
Apache Tomcat	Intel Pentium processor at 2.6GHz or faster	Minimum 512 MB Physical Memory; 1 GB Recommended	Minimum 20 GB
Net Beans 6.8 **Oracle10g**	Intel Pentium Processor at 2.6GHz or faster	Minimum 512 MB Physical Memory; (1 GB Recommended)	Minimum 20 GB

Software Interface

- **Client on Internet/Intranet:** Any web browser.
- **Web Server:** Apache Tomcat.
- **Database Server:** Oracle10g.
- **Development End:** NetBeans6.x.

3.5. PROBLEMS IN EXISTING SYSTEM

Here the existing system is a manual one using which the banking agent can‟t maintain the effectively by sharing across different branches with proper security and can‟t track details easily. It doesn’t provide proper co-ordination between different departments of the company. It doesn’t allow the customer to check the status of his file in proper way which leads customer dies-satisfaction.

- Doesn’t provide faster and effective system
- Doesn’t provide good co-ordination between departments
- Doesn’t provide effective forwarding system tomove the file from one level to another
- Doesn’t user-friendly interface
- Difficulty in generating different reports as per the business requirement
- Doesn’t facilitate the services from online

3.6. PROPOSED SYSTEM

The online automated system with web-based architecture can support issues like.

- This system maintains the information related different departments and stored at a central DB, which leads easy accessibility and consistency
- Interest rates of different banks and the other details are also available at the click of a mouse.
- Customer can apply for a loan and track his file details from online.
- The decision process in faster and more consistent
- Provides good communication between two departments
- Provides a facility to generate the reports very easily.

3.7. INPUT AND OUTPUT

The major inputs and outputs and major functions of the system are follows:

Inputs

- Admin enter his user id and password for login.
- Customer will enter his user id and password for login.
- New users give his completed personnel, address and phone details for registration.
- Admin gives different kind of user information to the user to search data.
- Customer will put a request for a new connection to the admin.

Outputs

- Admin can have his own home page.
- Users enter their own home page.
- The new user’s (Customer) data will be stored in the centralized database.
- Admin get the search details of different criteria.
- Different kind of reports is generated by administrator.

FEASIBILITY REPORT

Preliminary investigation examine project feasibility, the likelihood the system will be useful to the organization. The main objective of the feasibility study is to test the Technical, Operational and Economical feasibility for adding new modules and debugging old running system. All system is feasible if they are unlimited resources and infinite time. There are aspects in the feasibility study portion of the preliminary investigation:

- Technical Feasibility
- Operation Feasibility
- Economical Feasibility

4.1. TECHNICAL FEASIBILITY

The technical issue usually raised during the feasibility stage of the investigation includes the following:

- Does the necessary technology exist to do what is suggested?
- Do the proposed equipments have the technical capacity to hold the data required to use the new system?
- Will the proposed system provide adequate response to inquiries, regardless of the number or location of users?
- Can the system be upgraded if developed?
- Are there technical guarantees of accuracy, reliability, ease of access and data security?

Earlier no system existed to cater to the needs of „Secure Infrastructure Implementation System‟. The current system developed is technically feasible. It is a web based user interface for audit workflow at ABC Tech. Thus it provides an easy access to the users. The database‟s purpose is to create, establish and maintain a workflow among various entities in order to facilitate all concerned users in their various capacities or roles. Permission to the users would be granted based on the roles specified. Therefore, it provides the technical guarantee of accuracy, reliability and security. The software and hard requirements for the development of this project are not many and are already available in-house at ABC Tech or are available as free as open source. The work for the project is done with the current equipment and existing software technology. Necessary bandwidth exists for providing a fast feedback to the users irrespective of the number of users using the system.

4.2. OPERATIONAL FEASIBILITY

Proposed projects are beneficial only if they can be turned out into information system. That will meet the organization‟s operating requirements. Operational feasibility aspects of the project are to be taken as an important part of the project implementation. Some of the important issues raised are to test the operational feasibility of a project includes the following: -

- Is there sufficient support for the management from the users?
- Will the system be used and work properly if it is being developed and implemented?
- Will there be any resistance from the user that will undermine the possible application benefits?

This system is targeted to be in accordance with the above-mentioned issues. Beforehand, the management issues and user requirements have been taken into consideration. So there is no question of resistance from the users that can undermine the possible application benefits.

The well-planned design would ensure the optimal utilization of the computer resources and would help in the improvement of performance status.

4.3. ECONOMIC FEASIBILITY

A system can be developed technically and that will be used if installed must still be a good investment for the organization. In the economical feasibility, the development cost in creating the system is evaluated against the ultimate benefit derived from the new systems. Financial benefits must equal or exceed the costs.

The system is economically feasible. It does not require any addition hardware or software. Since the interface for this system is developed using the existing resources and technologies available at ABC Tech, There is nominal expenditure and economical feasibility for certain.

SOFTWARE REQUIREMENT SPECIFICATIONS

5.1. FUNCTIONAL REQUIREMENTS

Following is a list of functionalities of the system

Functional requirement means interface between user and the hardware.

In this project, we discuss the functionalities of the each module. The functional requirements are:

- ➢ In the Encryption process each LSB (Least Significant Byte) will be replaced by the encrypted data.
- ➢ Embedding it into the file, encryption operation will be performed by using the encryption key which is provided by the source.
- ➢ During Decryption process each data will be extracted from each LSB and then performs decryption operation which results into a plain text which was sent by the source side.
- ➢ Destination side this data will be encrypted from Audio, Video & Image file and performs decryption to get original message.

In Graphical User Interface we design the GUI

5.2. NON-FUNCTIONAL REQUIREMENTS:

The system should be Stand Alone Application. The system should have Role based System functions access. Approval Process has to be defined.

These are the mainly following:

- Secure access of confidential data (user’s details). SSL ***(Secure Sockets Layer)*** can be used.
- 24 X 7 availability
- Better component design to get better performance at peak time
- Flexible service based architecture will be highly desirable for future extension

Performance

They understand the importance of timing, of getting there before the competition. A rich portfolio of reusable, modular frameworks helps jump-start projects. Tried and tested methodology ensures that we follow a predictable, low - risk path to achieve results. Our track record is testimony to complex projects delivered within and evens before schedule.

Security

Its provides more security by setting username and password.

Safety

This application provides more safety to the users for accessing the databases and for performing the operations on the databases.

Interfaces

It provides the interface for accessing the database and also allows the user to do the manipulations on the databases.

Reliability

This entire project is depends on the SQL Server.

Accuracy

Since the same table is created at different users account, the

Possibility of retrieving data wrongly increases. Also if the data is more,

Validations become difficult. This may result in loss of accuracy of data.

Ease of Use

Ever user should be comfortable of working with computer and internet browsing. He must have basic knowledge of English.

Interoperability

This provides the import and export facilities for sending one database to another database.

Maintainability

The key to reducing need for maintenance, while working, if possible to do essential tasks.

1. More accurately defining user requirement during system development.
2. Assembling better systems documentation.
3. Using more effective methods for designing, processing, login and communicating information with project team members.
4. Making better use of existing tools and techniques.
5. Managing system engineering process effectively.

Testability:

Testing is done in various ways such as testing the algorithm, programming code; sample data debugging is also one of following the above testing.

5.3. PERFORMANCE REQUIREMENTS

Performance is measured in terms of the output provided by the application. Requirement specification plays an important part in the analysis of a system. Only when the requirement specifications are properly given, it is possible to design a system, which will fit into required environment. It rests largely in the part of the users of the existing system to give the requirement specifications because they are the people who finally use the system. This is because the requirements have to be known during the initial stages so that the system can be designed according to those requirements. It is very difficult to change the system once it has been designed and on the other hand designing a system, which does not cater to the requirements of the user, is of no use.

The requirement specification for any system can be broadly stated as given below:

- The system should be able to interface with the existing system
- The system should be accurate
- The system should be better than the existing system

The existing system is completely dependent on the user to perform all the duties.

SYSTEM DEVELOPEMENT ENVIRONMENT

6.1. INTRODUCTION TO JAVA

About Java

Initially the language was called as "oak" but it was renamed as "java" in 1995.The primary motivation of this language was the need for a platform-independent (i.e. architecture neutral) language that could be used to create software to be embedded in various consumer electronic devices.

6.1.1. Java is a programmer's language

6.1.2. Java is cohesive and consistent

6.1.3. Except for those constraint imposed by the Internet environment. Java gives the programmer, full control

Finally Java is to Internet Programming where c was to System Programming.

Importance of Java to the Internet

Java has had a profound effect on the Internet. This is because; java expands the Universe of objects that can move about freely in Cyberspace. In a network, two categories of objects are transmitted between the server and the personal computer. They are passive information and Dynamic active programs. in the areas of Security and probability. But Java addresses these concerns and by doing so, has opened the door to an exciting new form of program called the Applet.

Applications and applets

An application is a program that runs on our Computer under the operating system of that computer. It is more or less like one creating using C or C++ .Java‟s ability to create Applets makes it important. An Applet I san application, designed to be transmitted over the Internet and executed by a Java-compatible web browser. An applet I actually a tiny Java program, dynamically downloaded across the network, just like an image. But the difference is, it is an intelligent program, not just a media file. It can be react to the user input and dynamically change.

Java Architecture

Java architecture provides a portable, robust, high performing environment for development. Java provides portability by compiling the byte codes for the Java Virtual Machine, which is then interpreted on each platform by the run-time environment. Java is a dynamic system, able to load code when needed from a machine in the same room or across the planet.

Compilation of code

When you compile the code, the Java compiler creates machine code (called byte code)for a hypothetical machine called Java Virtual Machine(JVM). The JVM is supposed executed the byte code. The JVM is created for the overcoming the issue of probability. The code is written and compiled for one machine and interpreted on all machines .This machine is called Java Virtual Machine.

Simple

Java was designed to be easy for the Professional programmer to learn and to use effectively. If you are an experienced C++ Programmer. Learning Java will oriented features of C++ . Most of the confusing concepts from C++ are either left out of Java or implemented in a cleaner, more approachable manner. In Java there are a small number of clearly defined ways to accomplish a given task.

Object oriented

Java was not designed to be source-code compatible with any other language. This allowed the Java team the freedom to design with a blank state. One outcome of this was a clean usable, pragmatic approach to objects. The object model in Java is simple and easy to extend, while simple types, such as integers, are kept as high-performance non-objects.

Robust

The multi-platform environment of the web places extraordinary demands on a program, because the program must execute reliably in a variety of systems. The ability to create robust programs. Was given a high priority in the design of Java. Java is strictly typed language; it checks your code at compile time and runtime.

Java virtually eliminates the problems of memory management and deal location, which is completely automatic. In a well-written Java program, all run-time errors can and should be managed by your program.

6.2. AWT

AWT stands for **Abstract Window Toolkit**. It is a portable GUI library between Solaris and Windows 95/NT and Mac System 7.X(soon) for stand-alone applications and/or applets. Since it can be used in applets it can be used on IRIX, SunOS, HP/UX, Linux which Netscape 2.0 supports.

The Abstract Window Toolkit provides many classes for programmers to use. It is your connection between your application and the native GUI. The AWT hides you from the underlying details of the GUI your application will be running on and thus is at very high level of abstraction. It takes the lowest common denominator approach to retain portability. No floating toolbars or Balloon help here...

It is a Java package and can be used in any Java program by importing java.awt.* via the **import** keyword. The documentation for the package is available at the Java hompage. The package will be covered briefly as this document is not considered advanced material because it does not discuss Peers, ImageConsumers/Producers, Toolkits and other advanced AWT ilk. It is recommend you look at the source code to see how the AWT really works.

Structure of the AWT

The structure of the AWT is rather simple: **Components** are added to and *then* laid out by layout managers in **Containers**. We have a variety of event handling, menu, fonts, graphics classes in addition to those two, but they'll be only briefly discussed here.

Nothing prevents us from using threads, audio, I/O, networking classes alongside the AWT. The AWT by itself is rather bare and empty. Imagine a text editor that couldn't save! As a side note, the AWT can be used in a stand-alone Java application or in an applet that is embedded within a HTML document.

The AWT hierarchy

Classes:

- BorderLayout
- CardLayout
- CheckboxGroup
- Color
- Component
    - Button
    - Canvas

- Checkbox
- Choice
- Container
  - Panel
  - Window
    - Dialog
    - Frame
- Label
- List
- Scrollbar
- TextCompoment
  - TextArea
  - TextField

- Dimension
- Event
- FileDialog
- FlowLayout
- Font
- FontMetrics
- Graphics
- GridLayout
- GridBagConstraints
- GridBagLayout
- Image
- Insets
- MediaTracker
- MenuComponent
  - MenuBar
  - MenuItem
    - CheckboxMenuItem
    - Menu
- Point
- Polygon
- Toolkit

6.3. SWING

Swing includes graphical user interface (GUI) widgets such as text boxes, buttons, split-panes, and tables.Swing widgets provide more sophisticated GUI components than the earlier Abstract Window Toolkit. Since they are written in pure Java, they run the same on all platforms, unlike the AWT which is tied to the underlying platform's windowing system. Swing supports pluggable look and feel – not by using the native platform's facilities, but by roughly emulating them. This means you can get any supported look and feel on any platform. The disadvantage of lightweight components is possibly slower execution. The advantage is uniform behavior on all platforms.

Architecture

The Swing library makes heavy use of the Model/View/Controller software design pattern, which attempts to separate the data being viewed from the method by which it is viewed. Because of this, most Swing components have associated models (typically as interfaces), and the programmer can use various default implementations or provide their own. For example, the JTable has a model called TableModel that describes an interface for how a table would access tabular data. A default implementation of this operates on a two-dimensional array.

Swing favors relative layouts (which specify the positional relationships between components), as opposed to absolute layouts (which specify the exact location and size of components). The motivation for this is to allow Swing applications to work and appear visually correct regardless of the underlying systems colors, fonts, language, sizes or I/O devices. This can make screen design somewhat difficult and numerous tools have been developed to allow visual designing of screens.

Swing also uses a publish subscribe event model (as does AWT), where listeners subscribe to events that are fired by the application (such as pressing a button, entering text or clicking a checkbox). The model classes typically include, as part of their interface, methods for attaching listeners (this is the publish aspect of the event model).

The frequent use of loose coupling within the framework makes Swing programming somewhat different from higher-level GUI design languages and 4GLs. This is a contributing factor to Swing having such a steep learning curve.

Look and feel

Swing allows one to specialize the look and feel of widgets, by modifying the default (via runtime parameters), deriving from an existing one, by creating one from scratch, or, beginning with J2SE 5.0, by using the skinnable Synth Look and Feel, which is configured with an XML property file. The look and feel can be changed at runtime, and early demonstrations of Swing would frequently provide a way to do this.

Relationship to AWT

Since early versions of Java, a portion of the Abstract Windowing Toolkit (AWT) has provided platform independent APIs for user interface components. In AWT, each component is rendered and controlled by a native peer component specific to the underlying windowing system.
By contrast, Swing components are often described as lightweight because they do not require allocation of native resources in the operating system's windowing toolkit. The AWT components are referred to as heavyweight components.
Much of the Swing API is generally a complementary extension of the AWT rather than a direct replacement. In fact, every Swing lightweight interface ultimately exists within an AWT heavyweight component because all of the top-level components in Swing (JApplet, JDialog, JFrame, and JWindow) extend an AWT top-level container. The core rendering functionality used by Swing to draw its lightweight components is provided by Java2D, another part of JFC. However, the use of both lightweight and heavyweight components within the same window is generally discouraged due to Z-order incompatibilities.

Relationship to SWT

The Standard Widget Toolkit (SWT) is a competing toolkit originally developed by IBM and now maintained by the Eclipse Foundation. SWT's implementation has more in common with the heavyweight components of AWT. This confers benefits such as more accurate fidelity with the underlying native windowing toolkit, at the cost of an increased exposure to the native resources in the programming model. The advent of SWT has given rise to a great deal of division among Java desktop developers with many strongly favouring either SWT or Swing. A renewed focus on Swing look and feel fidelity with the native windowing toolkit in the approaching Java SE 6 release (as of February 2006) is probably a direct result of this.

Example

The following is a Hello World program using Swing.

```
import javax.swing.JFrame;
import javax.swing.JLabel;

public final class HelloWorld extends JFrame {
private HelloWorld() {
setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
getContentPane().add(new JLabel("Hello, World!"));
pack();
setLocationRelativeTo(null);
}

public static void main(String[] args) {
new HelloWorld().setVisible(true);
}
}
```

About the JFC and Swing

JFC is short for Java Foundation Classes, which encompass a group of features for building graphical user interfaces (GUIs) and adding rich graphics functionality and interactivity to Java applications. It is defined as containing the features shown in the table below.

Features of the Java Foundation Classes

Feature	**Description**
Swing GUI Components	Includes everything from buttons to split panes to tables. Many components are capable of sorting, printing, and drag and drop, to name a few of the supported features.
Pluggable Look-and-Feel Support	The look and feel of Swing applications is pluggable, allowing a choice of look and feel. For example, the same program can use either the Java or the Windows look

	and feel. Additionally, the Java platform supports the GTK+ look and feel, which makes hundreds of existing look and feels available to Swing programs. Many more look-and-feel packages are available from various sources.
Accessibility API	Enables assistive technologies, such as screen readers and Braille displays, to get information from the user interface.
Java 2D API	Enables developers to easily incorporate high-quality 2D graphics, text, and images in applications and applets. Java 2D includes extensive APIs for generating and sending high-quality output to printing devices.
Internationalization	Allows developers to build applications that can interact with users worldwide in their own languages and cultural conventions. With the input method framework developers can build applications that accept text in languages that use thousands of different characters, such as Japanese, Chinese, or Korean.

This trail concentrates on the Swing components. We help you choose the appropriate components for your GUI, tell you how to use them, and give you the background information you need to use them effectively. We also discuss other JFC features as they apply to Swing components.

Which Swing Packages Should I Use?

The Swing API is powerful, flexible — and immense. The Swing API has 18 public packages:

javax.accessibility	javax.swing.plaf	javax.swing.text
javax.swing	javax.swing.plaf.basic	javax.swing.text.html
javax.swing.border	javax.swing.plaf.metal	javax.swing.text.html.parser
javax.swing.colorchooser	javax.swing.plaf.multi	javax.swing.text.rtf
javax.swing.event	javax.swing.plaf.synth	javax.swing.tree
javax.swing.filechooser	javax.swing.table	javax.swing.undo

Fortunately, most programs use only a small subset of the API. This trail sorts out the API for you, giving you examples of common code and pointing you to methods and classes you're likely to need. Most of the code in this trail uses only one or two Swing packages:

- javax.swing
- javax.swing.event (not always required)

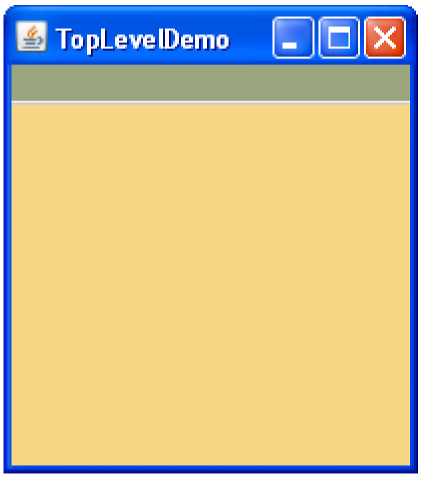
Using Top-Level Containers

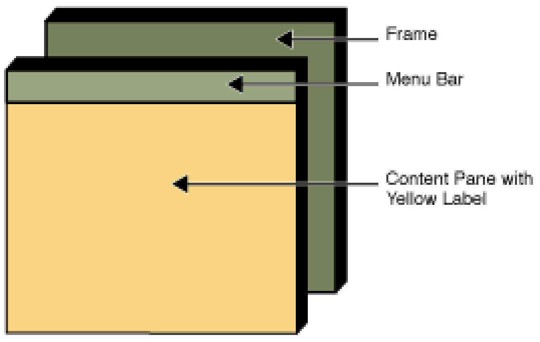
As we mentioned before, Swing provides three generally useful top-level container classes: JFrame, JDialog, and JApplet. When using these classes, you should keep these facts in mind:

- To appear onscreen, every GUI component must be part of a **containment hierarchy**. A containment hierarchy is a tree of components that has a top-level container as its root. We'll show you one in a bit.
- Each GUI component can be contained only once. If a component is already in a container and you try to add it to another container, the component will be removed from the first container and then added to the second.
- Each top-level container has a content pane that, generally speaking, contains (directly or indirectly) the visible components in that top-level container's GUI.
- You can optionally add a menu bar to a top-level container. The menu bar is by convention positioned within the top-level container, but outside the content pane. Some look and feels, such as the Mac OS look and feel, give you the option of placing the menu bar in another place more appropriate for the look and feel, such as at the top of the screen.

Note: Although JInternalFrame mimics JFrame, internal frames aren't actually top-level containers.

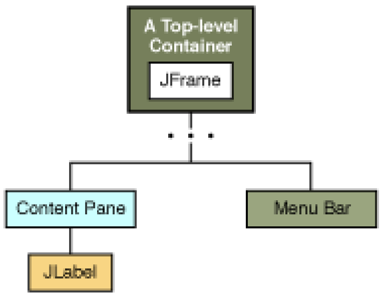
Here's a picture of a frame created by an application. The frame contains a green menu bar (with no menus) and, in the frame's content pane, a large blank, yellow label.





You can find the entire source for this example in TopLevelDemo.java. Although the example uses a JFrame in a standalone application, the same concepts apply to JApplets and JDialogs.

Here's the containment hierarchy for this example's GUI:



As the ellipses imply, we left some details out of this diagram. We reveal the missing details a bit later. Here are the topics this section discusses:

- Top-Level Containers and Containment Hierarchies
- Adding Components to the Content Pane
- Adding a Menu Bar
- The Root Pane (a.k.a. The Missing Details)

Top-Level Containers and Containment Hierarchies

Each program that uses Swing components has at least one top-level container. This top-level container is the root of a containment hierarchy — the hierarchy that contains all of the Swing components that appear inside the top-level container.

As a rule, a standalone application with a Swing-based GUI has at least one containment hierarchy with a JFrame as its root. For example, if an application has one main window and two dialogs, then the application has three containment hierarchies, and thus three

top-level containers. One containment hierarchy has a JFrame as its root, and each of the other two has a JDialog object as its root.

A Swing-based applet has at least one containment hierarchy, exactly one of which is rooted by a JApplet object. For example, an applet that brings up a dialog has two containment hierarchies. The components in the browser window are in a containment hierarchy rooted by a JApplet object. The dialog has a containment hierarchy rooted by a JDialog object.

Adding Components to the Content Pane

Here's the code that the preceding example uses to get a frame's content pane and add the yellow label to it:

```
frame.getContentPane().add(yellowLabel, BorderLayout.CENTER);
```

As the code shows, you find the content pane of a top-level container by calling the getContentPane method. The default content pane is a simple intermediate container that inherits from JComponent, and that uses a BorderLayout as its layout manager.

It's easy to customize the content pane — setting the layout manager or adding a border, for example. However, there is one tiny gotcha. The getContentPane method returns a Container object, not a JComponent object. This means that if you want to take advantage of the content pane's JComponent features, you need to either typecast the return value or create your own component to be the content pane. Our examples generally take the second approach, since it's a little cleaner. Another approach we sometimes take is to simply add a customized component to the content pane, covering the content pane completely.

Note that the default layout manager for JPanel is FlowLayout; you'll probably want to change it.

To make a component the content pane, use the top-level container's setContentPane method. For example:

```
//Create a panel and add components to it.
JPanel contentPane = new JPanel(new BorderLayout());
contentPane.setBorder(someBorder);
contentPane.add(someComponent, BorderLayout.CENTER);
contentPane.add(anotherComponent, BorderLayout.PAGE_END);


topLevelContainer.setContentPane(contentPane);
```

Note: As a convenience, the add method and its variants, remove and setLayout have been overridden to forward to the contentPane as necessary. This means you can write

```
frame.add(child);
```

and the child will be added to the contentPane.

Note that only these three methods do this. This means that getLayout() will not return the layout set with setLayout().

Adding a Menu Bar

In theory, all top-level containers can hold a menu bar. In practice, however, menu bars usually appear only in frames and applets. To add a menu bar to a top-level container, create a JMenuBar object, populate it with menus, and then call setJMenuBar. The TopLevelDemo adds a menu bar to its frame with this code:

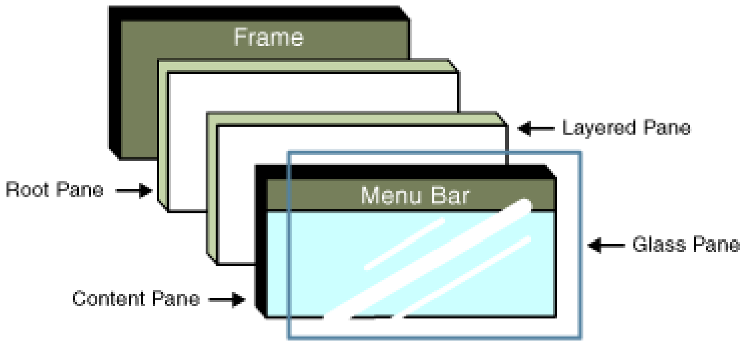
frame.setJMenuBar(greenMenuBar);

For more information about implementing menus and menu bars, see How to Use Menus.

The Root Pane

Each top-level container relies on a reclusive intermediate container called the **root pane**. The root pane manages the content pane and the menu bar, along with a couple of other containers. You generally don't need to know about root panes to use Swing components. However, if you ever need to intercept mouse clicks or paint over multiple components, you should get acquainted with root panes.

Here's a list of the components that a root pane provides to a frame (and to every other top-level container):



We've already told you about the content pane and the optional menu bar. The two other components that a root pane adds are a layered pane and a glass pane. The layered pane contains the menu bar and content pane, and enables Z-ordering of other components. The glass pane is often used to intercept input events occuring over the top-level container, and can also be used to paint over multiple components.

The JComponent Class

With the exception of top-level containers, all Swing components whose names begin with "J" descend from the JComponent class. For example, JPanel, JScrollPane, JButton, and JTable all inherit from JComponent. However, JFrame and JDialog don't because they implement top-level containers.

The JComponent class extends the Container class, which itself extends Component. The Component class includes everything from providing layout hints to supporting painting and events. The Container class has support for adding components to the container and laying them out. This section's API tables summarize the most often used methods of Component and Container, as well as of JComponent.

JComponent Features

The JComponent class provides the following functionality to its descendants:

- Tool tips
- Painting and borders
- Application-wide pluggable look and feel
- Custom properties
- Support for layout
- Support for accessibility
- Support for drag and drop
- Double buffering
- Key bindings

Tool tips

By specifying a string with the setToolTipText method, you can provide help to users of a component. When the cursor pauses over the component, the specified string is displayed in a small window that appears near the component. See How to Use Tool Tips for more information.

Painting and borders

The setBorder method allows you to specify the border that a component displays around its edges. To paint the inside of a component, override the paintComponent method. See How to Use Borders and Performing Custom Painting for details.

Application-wide pluggable look and feel

Behind the scenes, each JComponent object has a corresponding ComponentUI object that performs all the drawing, event handling, size determination, and so on for that JComponent. Exactly which ComponentUI object is used depends on the current look and feel, which you can set using the UIManager.setLookAndFeel method. See How to Set the Look and Feel for details.

Custom properties

You can associate one or more properties (name/object pairs) with any JComponent. For example, a layout manager might use properties to associate a constraints object with each JComponent it manages. You put and get properties using the putClientProperty and getClientProperty methods. For general information about properties, see Properties.

Support for layout

Although the Component class provides layout hint methods such as getPreferredSize and getAlignmentX, it doesn't provide any way to set these layout hints, short of creating a subclass and overriding the methods. To give you another way to set layout hints, the JComponent class adds setter methods — setMinimumSize, setMaximumSize, setAlignmentX, and setAlignmentY. See Laying Out Components Within a Container for more information.

Support for accessibility

The JComponent class provides API and basic functionality to help assistive technologies such as screen readers get information from Swing components, For more information about accessibility, see How to Support Assistive Technologies.

Support for drag and drop

The JComponent class provides API to set a component's transfer handler, which is the basis for Swing's drag and drop support. See Introduction to DnD for details.

Double buffering

Double buffering smooths on-screen painting. For details, see Performing Custom Painting.

Key bindings

This feature makes components react when the user presses a key on the keyboard. For example, in many look and feels when a button has the focus, typing the Space key is equivalent to a mouse click on the button. The look and feel automatically sets up the bindings between pressing and releasing the Space key and the resulting effects on the button. For more information about key bindings, see How to Use Key Bindings.

The JComponent API

The JComponent class provides many new methods and inherits many methods from Component and Container. The following tables summarize the methods we use the most.

- Customizing Component Appearance
- Setting and Getting Component State
- Handling Events
- Painting Components
- Dealing with the Containment Hierarchy
- Laying Out Components
- Getting Size and Position Information
- Specifying Absolute Size and Position

Customizing Component Appearance

Method	Purpose
void setBorder(Border)	Set or get the border of the component. See How to Use Borders for details.

Border getBorder()	
void setForeground(Color) void setBackground(Color)	Set the foreground or background color for the component. The foreground is generally the color used to draw the text in a component. The background is (not surprisingly) the color of the background areas of the component, assuming that the component is opaque.
Color getForeground() Color getBackground()	Get the foreground or background color for the component.
void setOpaque(boolean) boolean isOpaque()	Set or get whether the component is opaque. An opaque component fills its background with its background color.
void setFont(Font) Font getFont()	Set or get the component's font. If a font has not been set for the component, the font of its parent is returned.
void setCursor(Cursor) Cursor getCursor()	Set or get the cursor displayed over the component and all components it contains (except for children that have their own cursor set). Example: aPanel.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));

Setting and Getting Component State

Method	Purpose
void setComponentPopupMenu(JPopupMenu)	Sets the JPopupMenu for this JComponent. The UI is responsible for registering bindings and adding the necessary listeners such that the JPopupMenu will be shown at the appropriate time. When the JPopupMenu is shown depends upon the look and feel: some may show it on a mouse event, some may enable a key binding. If popup is null, and getInheritsPopupMenu returns true, then getComponentPopupMenu will be delegated to the parent. This provides for a way to make all child components inherit the popupmenu of the parent.
void setTransferHandler(TransferHandler)	Set or remove the transferHandler property. The TransferHandler supports exchanging data via cut, copy, or

TransferHandler getTransferHandler()	paste to/from a clipboard as well a drag and drop. See Introduction to DnD for more details.
void setToolTipText(String)	Set the text to display in a tool tip. See How to Use Tool Tips for more information.
void setName(String) String getName()	Set or get the name of the component. This can be useful when you need to associate text with a component that does not display text.
boolean isShowing()	Determine whether the component is showing on screen. This means that the component must be visible, and it must be in a container that is visible and showing.
void setEnabled(boolean) boolean isEnabled()	Set or get whether the component is enabled. An enabled component can respond to user input and generate events.
void setVisible(boolean) boolean isVisible()	Set or get whether the component is visible. Components are initially visible, with the exception of top-level components.

Handling Events

(see Writing Event Listeners for details)

Method	Purpose
void addHierarchyListener(hierarchyListener l) void removeHierarchyListener(hierarchyListener l)	Adds or removes the specified hierarchy listener to receive hierarchy changed events from this component when the hierarchy to which this container belongs changes. If listener l is null, no exception is thrown and no action is performed.
void addMouseListener(MouseListener) void removeMouseListener(MouseListener)	Add or remove a mouse listener to or from the component. Mouse listeners are notified when the user uses the mouse to interact with the listened-to component.
void addMouseMotionListener(MouseMotionListener) void	Add or remove a mouse motion listener to or from the component. Mouse motion listeners are notified when the user moves the mouse within

removeMouseMotionListener(MouseMotionListener)	the listened-to component's bounds.
void addKeyListener(KeyListener) void removeKeyListener(KeyListener)	Add or remove a key listener to or from the component. Key listeners are notified when the user types at the keyboard and the listened-to component has the keyboard focus.
void addComponentListener(ComponentListener) void removeComponentListener(ComponentListener)	Add or remove a component listener to or from the component. Component listeners are notified when the listened-to component is hidden, shown, moved, or resized.
boolean contains(int, int) boolean contains(Point)	Determine whether the specified point is within the component. The argument should be specified in terms of the component's coordinate system. The two int arguments specify **x** and **y** coordinates, respectively.
Component getComponentAt(int, int) Component getComponentAt(Point)	Return the component that contains the specified **x, y** position. The top-most child component is returned in the case where components overlap. This is determined by finding the component closest to the index 0 that claims to contain the given point via Component.contains().
Component setComponentZOrder(component comp, int index)	Moves the specified component to the specified z-order index in the container. If the component is a child of some other container, it is removed from that container before being added to this container. The important difference between this method and java.awt.Container.add(Component, int) is that this method doesn't call removeNotify on the component while removing it from its previous container unless necessary and when allowed by the underlying native windowing system. This

	way, if the component has the keyboard focus, it maintains the focus when moved to the new position. **Note:** The z-order determines the order that components are painted. The component with the highest z-order paints first and the component with the lowest z-order paints last. Where components overlap, the component with the lower z-order paints over the component with the higher z-order.
Component getComponentZOrder(component comp)	Returns the z-order index of the component inside the container. The higher a component is in the z-order hierarchy, the lower its index. The component with the lowest z-order index is painted last, above all other child components.

Painting Components

(see Performing Custom Painting for details)

Method	Purpose
void repaint() void repaint(int, int, int, int)	Request that all or part of the component be repainted. The four int arguments specify the bounds (**x**, **y**, width, height, in that order) of the rectangle to be painted.
void repaint(Rectangle)	Request that the specified area within the component be repainted.
void revalidate()	Request that the component and its affected containers be laid out again. You should not generally need to invoke this method unless you explicitly change a component's size/alignment hints after it's visible or change a containment hierarchy after it is visible. Always invoke repaint after revalidate.
void paintComponent(Graphics)	Paint the component. Override this method to implement painting for custom components.

Dealing with the Containment Hierarchy

(see Using Top-Level Containers for more information)

Method	Purpose
Component add(Component) Component add(Component, int) void add(Component, Object)	Add the specified component to this container. The one-argument version of this method adds the component to the end of the container. When present, the int argument indicates the new component's position within the container. When present, the Object argument provides layout constraints to the current layout manager.
void remove(int) void remove(Component) void removeAll()	Remove one of or all of the components from this container. When present, the int argument indicates the position within the container of the component to remove.
JRootPane getRootPane()	Get the root pane that contains the component.
Container getTopLevelAncestor()	Get the topmost container for the component — a Window, Applet, or null if the component has not been added to any container.
Container getParent()	Get the component's immediate container.
int getComponentCount()	Get the number of components in this container.
Component getComponent(int) Component[] getComponents()	Get the one of or all of the components in this container. The int argument indicates the position of the component to get.
Component getComponentZOrder(int) Component[] getComponentZOrder()	Returns the z-order index of the component inside the container. The higher a component is in the z-order hierarchy, the lower its index. The component with the lowest z-order index is painted last, above all other child components.

Laying Out Components

(see Laying Out Components Within a Container for more information)

Method	**Purpose**
void setPreferredSize(Dimension) void setMaximumSize(Dimension) void setMinimumSize(Dimension)	Set the component's preferred, maximum, or minimum size, measured in pixels. The preferred size indicates the best size for the component. The component should be no larger than its maximum size and no smaller than its minimum size. Be aware that these are hints only and might be ignored by certain layout managers.
Dimension getPreferredSize() Dimension getMaximumSize() Dimension getMinimumSize()	Get the preferred, maximum, or minimum size of the component, measured in pixels. Many JComponent classes have setter and getter methods. For those non-JComponent subclasses, which do not have the corresponding setter methods, you can set a component's preferred, maximum, or minimum size by creating a subclass and overriding these methods.
void setAlignmentX(float) void setAlignmentY(float)	Set the alignment along the **x-** or **y-** axis. These values indicate how the component would like to be aligned relative to other components. The value should be a number between 0 and 1 where 0 represents alignment along the origin, 1 is aligned the furthest away from the origin, and 0.5 is centered, and so on. Be aware that these are hints only and might be ignored by certain layout managers.
float getAlignmentX() float getAlignmentY()	Get the alignment of the component along the **x-** or **y-** axis. For non-JComponent subclasses, which do not have the corresponding setter methods, you can set a component's alignment

	by creating a subclass and overriding these methods.
void setLayout(LayoutManager) LayoutManager getLayout()	Set or get the component's layout manager. The layout manager is responsible for sizing and positioning the components within a container.
void applyComponentOrientation(ComponentOrientation) void setComponentOrientation(ComponentOrientation)	Set the ComponentOrientation property of this container and all the components contained within it. See Setting the Container's Orientation for more information.

Getting Size and Position Information

Method	Purpose
int getWidth() int getHeight()	Get the current width or height of the component measured in pixels.
Dimension getSize() Dimension getSize(Dimension)	Get the component's current size measured in pixels. When using the one-argument version of this method, the caller is responsible for creating the Dimension instance in which the result is returned.
int getX() int getY()	Get the current **x** or y coordinate of the component's origin relative to the parent's upper left corner measured in pixels.
Rectangle getBounds() Rectangle getBounds(Rectangle)	Get the bounds of the component measured in pixels. The bounds specify the component's width, height, and origin relative to its parent. When using the one-argument version of this method, the caller is responsible for creating the Rectangle instance in which the result is returned.
Point getLocation() Point getLocation(Point)	Gets the current location of the component relative to the parent's upper left corner measured in pixels. When using the one-argument version of getLocation method, the caller is responsible for creating the Point instance in which the result is returned.
Point getLocationOnScreen()	Returns the position relative to the upper left corner of the screen.

Specifying Absolute Size and Position

(see Doing Without a Layout Manager (Absolute Positioning) for more information)

Method	Purpose
void setLocation(int, int) void setLocation(Point)	Set the location of the component, in pixels, relative to the parent's upper left corner. The two int arguments specify **x** and **y**, in that order. Use these methods to position a component when you are not using a layout manager.
void setSize(int, int) void setSize(Dimension)	Set the size of the component measured in pixels. The two int arguments specify width and height, in that order. Use these methods to size a component when you are not using a layout manager.
void setBounds(int, int, int, int) void setBounds(Rectangle)	Set the size and location relative to the parent's upper left corner, in pixels, of the component. The four int arguments specify **x**, **y**, width, and height, in that order. Use these methods to position and size a component when you are not using a layout manager.

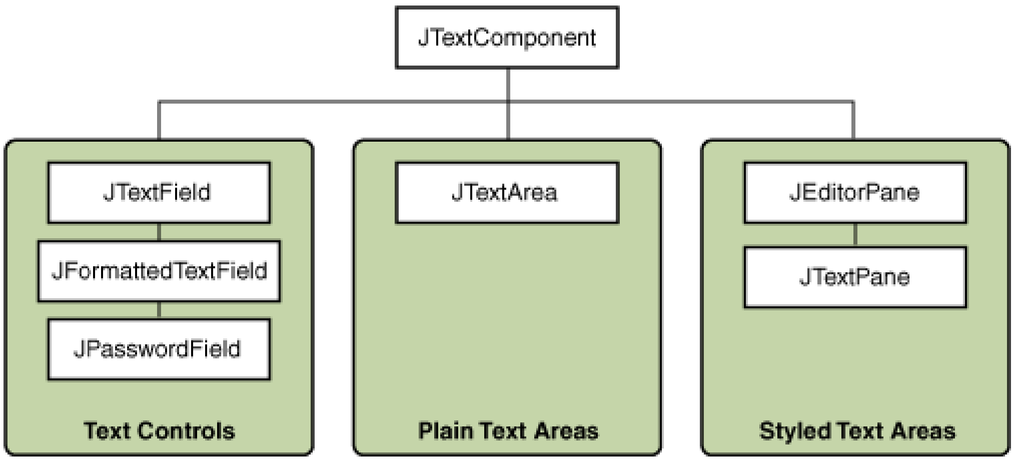
Using Text Components

This section provides background information you might need when using Swing text components. If you intend to use an unstyled text component — a text field, password field, formatted text field, or text area — go to its how-to page and return here only if necessary. If you intend to use a styled text component, see How to Use Editor Panes and Text Panes, and read this section as well. If you do not know which component you need, read on.

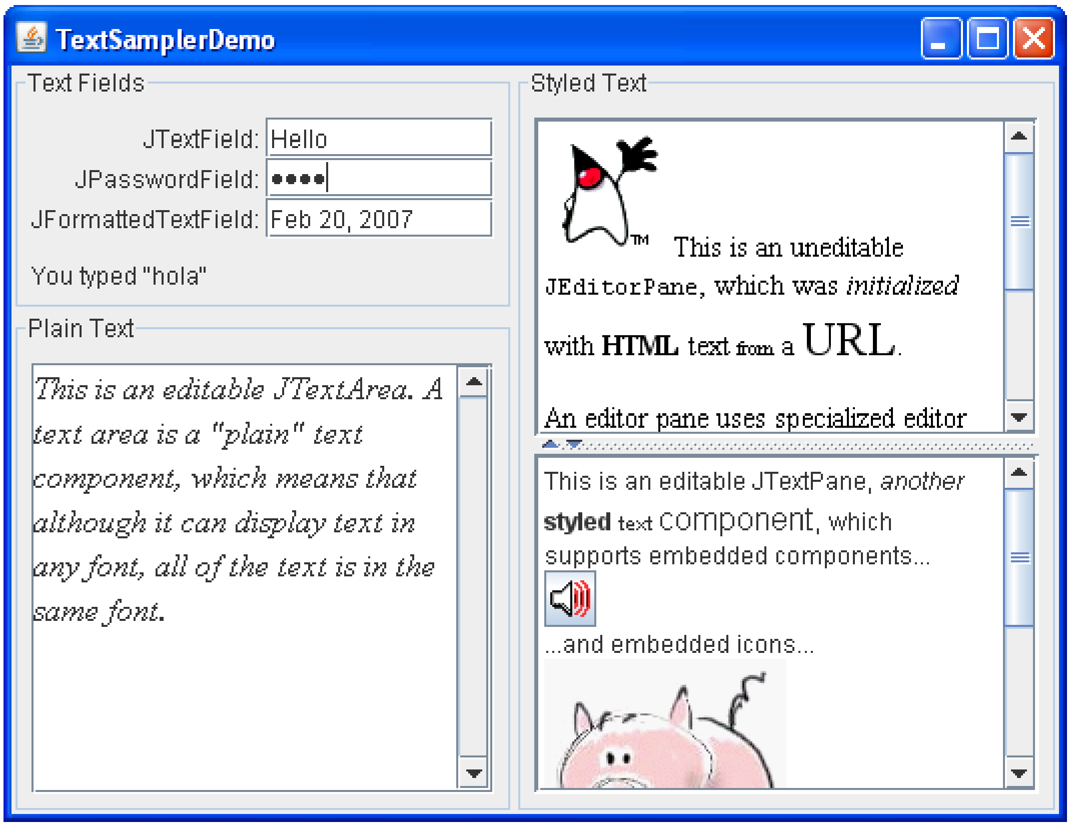
Swing text components display text and optionally allow the user to edit the text. Programs need text components for tasks ranging from the straightforward (enter a word and press Enter) to the complex (display and edit styled text with embedded images in an Asian language).

Swing provides six text components, along with supporting classes and interfaces that meet even the most complex text requirements. In spite of their different uses and capabilities, all Swing text components inherit from the same superclass, JTextComponent, which provides a highly-configurable and powerful foundation for text manipulation.

The following figure shows the JTextComponent hierarchy.



The following picture shows an application called TextSamplerDemo that uses each Swing text component.



JTextField: Hello
JPasswordField: ●●●●
JFormattedTextField: Feb 20, 2007

You typed "hola"

This is an editable JTextArea. A text area is a "plain" text component, which means that although it can display text in any font, all of the text is in the same font.

This is an uneditable JEditorPane, which was *initialized* with **HTML** text from a URL.

An editor pane uses specialized editor

This is an editable JTextPane, *another* **styled** text component, which supports embedded components...

...and embedded icons...

Try this:

1. Click the Launch button to run TextSamplerDemo using Java™ Web Start (download JDK 6). Alternatively, to compile and run the example yourself, consult the example index.

Launch

2. Type some text in the text field and press Enter. Do the same in the password field. The label beneath the fields is updated when you press Enter.
3. Try entering valid and invalid dates into the formatted text field. Note that when you press Enter the label beneath the fields is updated only if the date is valid.
4. Select and edit text in the text area and the text pane. Use keyboard bindings, Ctrl-X, Ctrl-C, and Ctrl-V, to cut, copy, and paste text, respectively.
5. Try to edit the text in the editor pane, which has been made uneditable with a call to setEditable.
6. Look in the text pane to find an example of an embedded component and an embedded icon.

The TextSamplerDemo example uses the text components in very basic ways. The following table tells you more about what you can do with each kind of text component.

Group	Description	Swing Classes
Text Controls	Also known simply as text fields, text controls can display only one line of editable text. Like buttons, they generate action events. Use them to get a small amount of textual information from the user and perform an action after the text entry is complete.	JTextField and its subclasses JPasswordField and JFormattedTextField
Plain Text Areas	JTextArea can display multiple lines of editable text. Although a text area can display text in any font, all of the text is in the same font. Use a text area to allow the user to enter unformatted text of any length or to display unformatted	JTextArea

	help information.	
Styled Text Areas	A styled text component can display editable text using more than one font. Some styled text components allow embedded images and even embedded components. Styled text components are powerful and multi-faceted components suitable for high-end needs, and offer more avenues for customization than the other text components. Because they are so powerful and flexible, styled text components typically require more initial programming to set up and use. One exception is that editor panes can be easily loaded with formatted text from a URL, which makes them useful for displaying uneditable help information.	JEditorPane and its subclass JTextPane

This Tutorial provides information about the foundation laid by the JTextComponent class and tells you how to accomplish some common text-related tasks. Because the JTextComponent class and its subclasses have too many features to be completely described in this Tutorial, please visit the Swing and AWT forum at java.net for help and information.

How to Use HTML in Swing Components

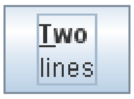
Many Swing components display a text string as part of their GUI. By default, a component's text is displayed in a single font and color, all on one line. You can determine the font and color of a component's text by invoking the component's setFont and setForeground methods, respectively. For example, the following code creates a label and then sets its font and color:

```
label = new JLabel("A label");
label.setFont(new Font("Serif", Font.PLAIN, 14));
label.setForeground(new Color(0xffffdd));
```

If you want to mix fonts or colors within the text, or if you want formatting such as multiple lines, you can use HTML. HTML formatting can be used in all Swing buttons, menu items, labels, tool tips, and tabbed panes, as well as in components such as trees and tables that use labels to render text.

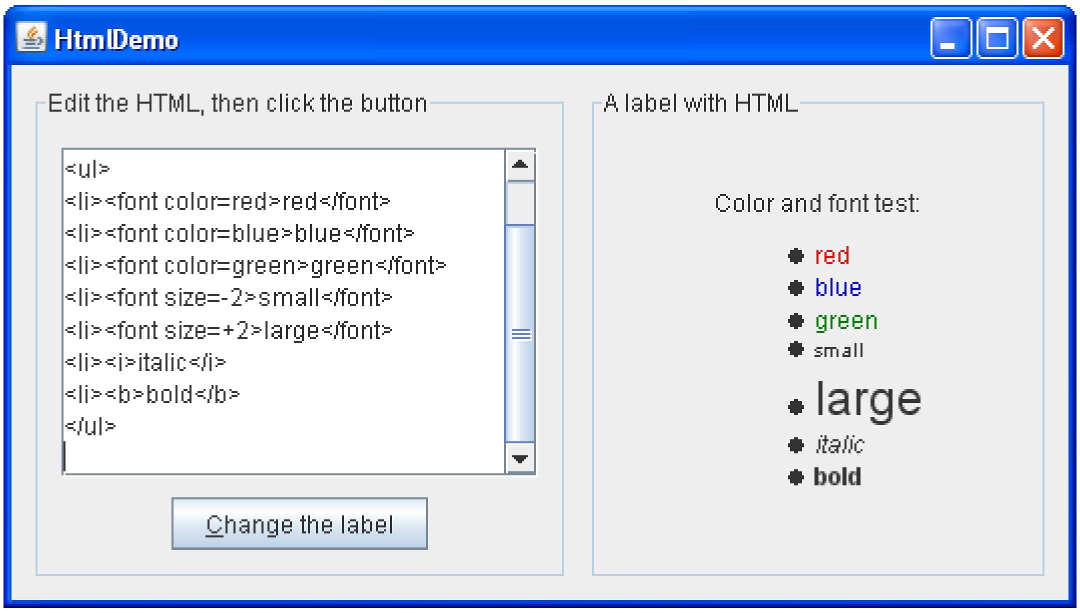
To specify that a component's text has HTML formatting, just put the <html> tag at the beginning of the text, then use any valid HTML in the remainder. Here is an example of using HTML in a button's text:

button = new JButton("<html><b><u>T</u>wo</b><br>lines</html>");

Here is the resulting button. 

An Example: HtmlDemo

An application called HtmlDemo lets you play with HTML formatting by setting the text on a label. You can find the entire code for this program in HtmlDemo.java. Here is a picture of the HtmlDemo example.



```
<ul>
<li><font color=red>red</font>
<li><font color=blue>blue</font>
<li><font color=green>green</font>
<li><font size=-2>small</font>
<li><font size=+2>large</font>
<li><i>italic</i>
<li><b>bold</b>
</ul>
```

Color and font test:

- red
- blue
- green
- small
- large
- *italic*
- **bold**

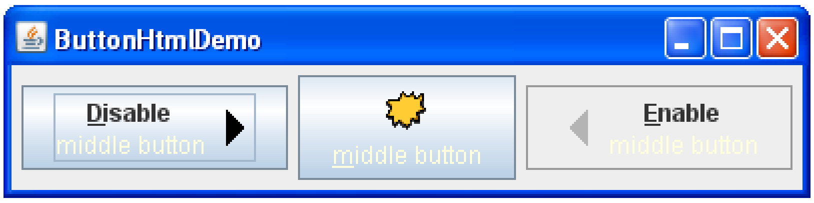
Try This:

1. Click the Launch button to run HtmlDemo using Java™ Web Start (download JDK 6). Alternatively, to compile and run the example yourself, consult the example index.

2. Edit the HTML formatting in the text area at the left and click the "Change the label" button. The label at the right shows the result.
3. Remove the <html> tag from the text area on the left. The label's text is no longer parsed as HTML.

Example 2: ButtonHtmlDemo

Let us look at another example that uses HTML. ButtonHtmlDemo adds font, color, and other text formatting to three buttons. You can find the entire code for this program in ButtonHtmlDemo.java. Here is a picture of the ButtonHtmlDemo example.



Click the Launch button to run ButtonHtmlDemo using Java™ Web Start (download JDK 6). Alternatively, to compile and run the example yourself, consult the example index.

Launch

The left and right buttons have multiple lines and text styles and are implemented using HTML. The middle button, on the other hand, uses just one line, font, and color, so it does not require HTML. Here is the code that specifies the text formatting for these three buttons:

```
b1 = new JButton("<html><center><b><u>D</u>isable</b><br>"
            + "<font color=#ffffdd>middle button</font>",
            leftButtonIcon);
Font font = b1.getFont().deriveFont(Font.PLAIN);
b1.setFont(font);
...
b2 = new JButton("middle button", middleButtonIcon);
b2.setFont(font);
b2.setForeground(new Color(0xffffdd));
...
b3 = new JButton("<html><center><b><u>E</u>nable</b><br>"
            + "<font color=#ffffdd>middle button</font>",
            rightButtonIcon);
b3.setFont(font);
```

Note that we have to use a <u> tag to cause the mnemonic characters "D" and "E" to be underlined in the buttons that use HTML. Note also that when a button is disabled, its HTML text unfortunately remains black, instead of becoming gray. (Refer to bug #4783068 to see if this situation changes.)

This section discussed how to use HTML in ordinary, non-text components. For information on components whose primary purpose is formatting text, see Using Text Components.

How to Use Models

Most Swing components have models. A button (JButton), for example, has a model (a ButtonModel object) that stores the button's state — what its keyboard mnemonic is, whether it's enabled, selected, or pressed, and so on. Some components have multiple models. A list (JList), for example, uses a ListModel to hold the list's contents, and a ListSelectionModel to track the list's current selection.

You often don't need to know about the models that a component uses. For example, programs that use buttons usually deal directly with the JButton object, and don't deal at all with the ButtonModel object.

Why then do models exist? The biggest reason is that they give you flexibility in determining how data is stored and retrieved. For example, if you're designing a spreadsheet application that displays data in a sparsely populated table, you can create your own table model that is optimized for such use.

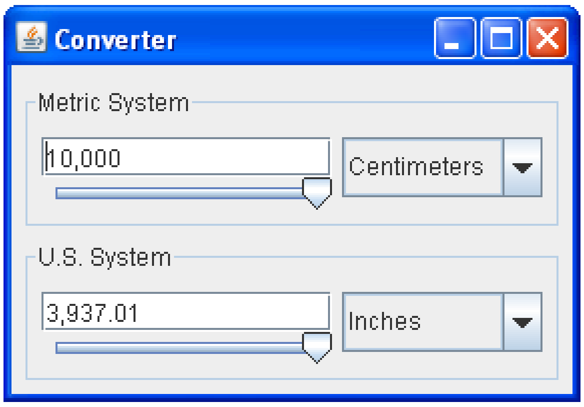
Models have other benefits, too. They mean that data isn't copied between a program's data structures and those of the Swing components. Also, models automatically propagate changes to all interested listeners, making it easy for the GUI to stay in sync with the data. For example, to add items to a list you can invoke methods on the list model. When the model's data changes, the model fires events to the JList and any other registered listeners, and the GUI is updated accordingly.

Although Swing's model architecture is sometimes referred to as a Model-View-Controller (MVC) design, it really isn't. Swing components are generally implemented so that the view and controller are indivisible, implemented by a single UI object provided by the look and feel. The Swing model architecture is more accurately described as a **separable model architecture**. If you're interested in learning more about the Swing model architecture, see A Swing Architecture Overview, an article in **The Swing Connection**.

An Example: Converter

This section features an example called Converter, which is an application that continuously converts distance measurements between metric and U.S. units. You can **run Converter** (download JDK 6). Or, to compile and run the example yourself, consult the example index.

As the following picture shows, Converter features two sliders, each tied to a text field. The sliders and text fields all display the same data — a distance — but using two different units of measure.



10,000

3,937.01

The important thing for this program is ensuring that only one model controls the value of the data. There are various ways to achieve this; we did it by deferring to the top slider's model. The bottom slider's model (an instance of a custom class called FollowerRangeModel) forwards all data queries to the top slider's model (an instance of a custom class called ConverterRangeModel). Each text field is kept in sync with its slider, and vice versa, by event handlers that listen for changes in value. Care is taken to ensure that the top slider's model has the final say about what distance is displayed.

When we started implementing the custom slider models, we first looked at the API section of How to Use Sliders. It informed us that all slider data models must implement the BoundedRangeModel interface. The BoundedRangeModel API documentation tells us that the interface has an implementing class named DefaultBoundedRangeModel. The API documentation for DefaultBoundedRangeModel shows that it's a general-purpose implementation of BoundedRangeModel.

We didn't use DefaultBoundedRangeModel directly because it stores data as integers, and Converter uses floating-point data. Thus, we implemented ConverterRangeModel as a subclass of Object. We then implemented FollowerRangeModel as a subclass of ConverterRangeModel

How to Use Borders

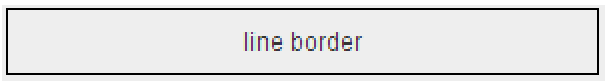
Every JComponent can have one or more borders. Borders are incredibly useful objects that, while not themselves components, know how to draw the edges of Swing components. Borders are useful not only for drawing lines and fancy edges, but also for providing titles and empty space around components.

Note: Our examples set borders on JPanels, JLabels, and custom subclasses of JComponent. Although technically you can set the border on any object that inherits from JComponent, the look and feel implementation of many standard Swing components doesn't work well with user-set borders. In general, when you want to set a border on a standard Swing component other than JPanel or JLabel, we recommend that you put the component in a JPanel and set the border on the JPanel.

To put a border around a JComponent, you use its setBorder method. You can use the BorderFactory class to create most of the borders that Swing provides. If you need a reference to a border — say, because you want to use it in multiple components — you can save it in a variable of type Border. Here is an example of code that creates a bordered container:

```
JPanel pane = new JPanel();
pane.setBorder(BorderFactory.createLineBorder(Color.black));
```

Here's a picture of the container, which contains a label component. The black line drawn by the border marks the edge of the container.



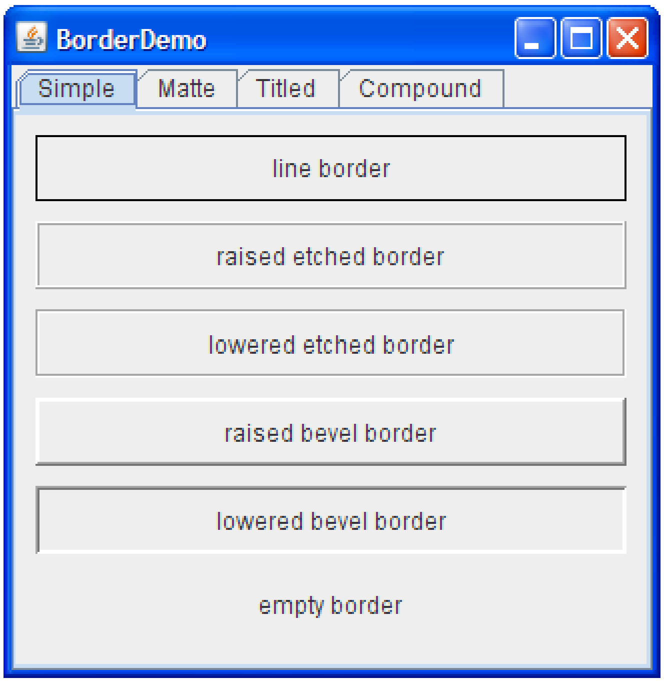
The rest of this page discusses the following topics:

- The BorderDemo Example
- Using the Borders Provided by Swing
- Creating Custom Borders
- The Border API
- Examples of Using Borders

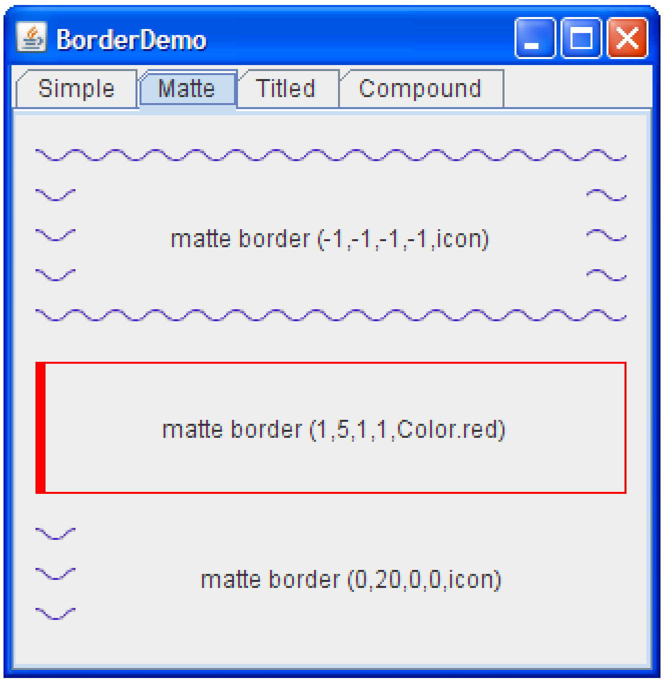
The BorderDemo Example

The following pictures show an application called BorderDemo that displays the borders Swing provides. We show the code for creating these borders a little later, in Using the Borders Provided by Swing.

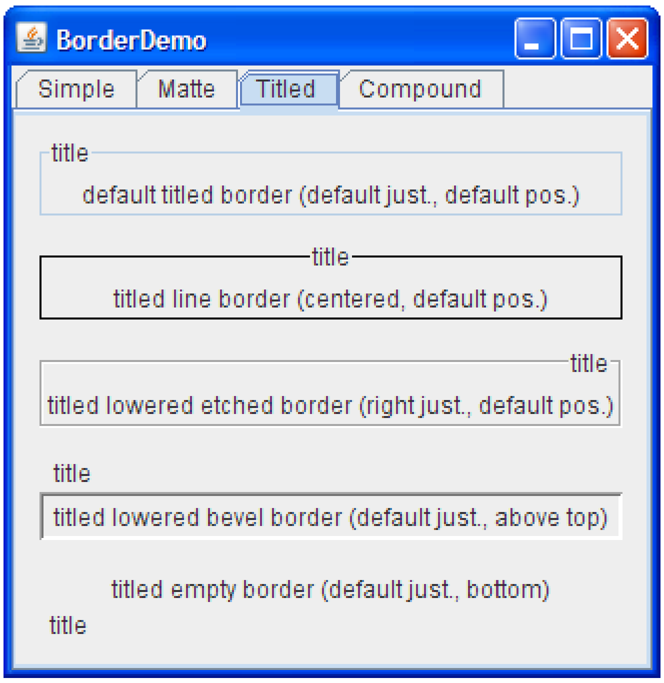
Click the Launch button to run the BorderDemo example using Java™ Web Start (download JDK 6). Alternatively, to compile and run the example yourself, consult the example index.



The next picture shows some matte borders. When creating a matte border, you specify how many pixels it occupies at the top, left, bottom, and right of a component. You then specify either a color or an icon for the matte border to draw. You need to be careful when choosing the icon and determining your component's size; otherwise, the icon might get chopped off or have mismatch at the component's corners.


BorderDemo
Simple
Matte
Titled
Compound
matte border (-1,-1,-1,-1,icon)
matte border (1,5,1,1,Color.red)
matte border (0,20,0,0,icon)

The next picture shows titled borders. Using a titled border, you can convert any border into one that displays a text description. If you don't specify a border, a look-and-feel-specific border is used. For example, the default titled border in the Java look and feel uses a gray line, and the default titled border in the Windows look and feel uses an etched border. By default, the title straddles the upper left of the border, as shown at the top of the following figure.



title
default titled border (default just., default pos.)

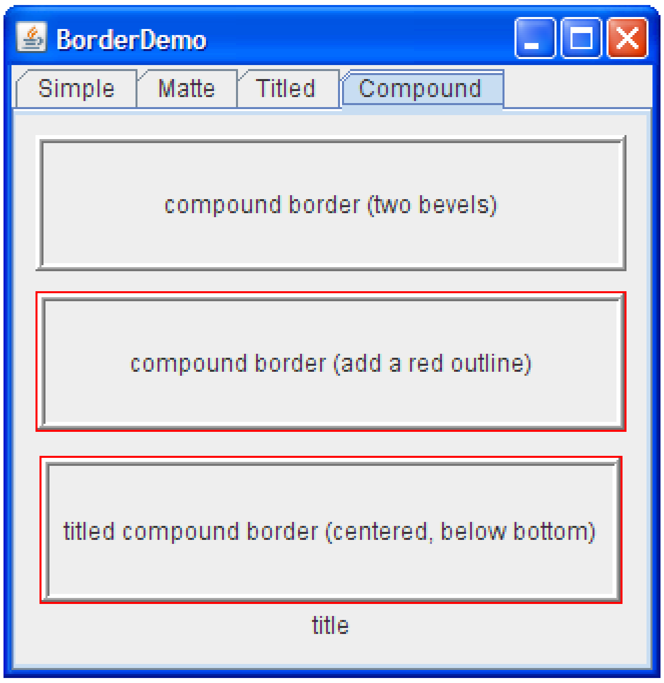
title
titled line border (centered, default pos.)

title
titled lowered etched border (right just., default pos.)

title
titled lowered bevel border (default just., above top)

titled empty border (default just., bottom)
title

The next picture shows compound borders. With compound borders, you can combine any two borders, which can themselves be compound borders.



compound border (two bevels)

compound border (add a red outline)

titled compound border (centered, below bottom)
title

Using the Borders Provided by Swing

The code that follows shows how to create and set the borders you saw in the preceding figures. You can find the program's code in BorderDemo.java.

```
//Keep references to the next few borders,
//for use in titles and compound borders.
Border blackline, raisedetched, loweredetched,
       raisedbevel, loweredbevel, empty;

blackline = BorderFactory.createLineBorder(Color.black);
raisedetched = BorderFactory.createEtchedBorder(EtchedBorder.RAISED);
loweredetched = BorderFactory.createEtchedBorder(EtchedBorder.LOWERED);
raisedbevel = BorderFactory.createRaisedBevelBorder();
loweredbevel = BorderFactory.createLoweredBevelBorder();
empty = BorderFactory.createEmptyBorder();

//Simple borders
jComp1.setBorder(blackline);
jComp2.setBorder(raisedbevel);
jComp3.setBorder(loweredbevel);
jComp4.setBorder(empty);

//Matte borders
ImageIcon icon = createImageIcon("images/wavy.gif",
                                 "wavy-line border icon"); //20x22

jComp5.setBorder(BorderFactory.createMatteBorder(
                                  -1, -1, -1, -1, icon));
jComp6.setBorder(BorderFactory.createMatteBorder(
                                   1, 5, 1, 1, Color.red));
jComp7.setBorder(BorderFactory.createMatteBorder(
                                   0, 20, 0, 0, icon));

//Titled borders
TitledBorder title;
```

```
title = BorderFactory.createTitledBorder("title");
jComp8.setBorder(title);

title = BorderFactory.createTitledBorder(
                                blackline, "title");
title.setTitleJustification(TitledBorder.CENTER);
jComp9.setBorder(title);

title = BorderFactory.createTitledBorder(
                                loweredetched, "title");
title.setTitleJustification(TitledBorder.RIGHT);
jComp10.setBorder(title);

title = BorderFactory.createTitledBorder(
                                loweredbevel, "title");
title.setTitlePosition(TitledBorder.ABOVE_TOP);
jComp11.setBorder(title);

title = BorderFactory.createTitledBorder(
                                empty, "title");
title.setTitlePosition(TitledBorder.BOTTOM);
jComp12.setBorder(title);

//Compound borders
Border compound;
Border redline = BorderFactory.createLineBorder(Color.red);

//This creates a nice frame.
compound = BorderFactory.createCompoundBorder(
                                        raisedbevel, loweredbevel);
jComp13.setBorder(compound);

//Add a red outline to the frame.
compound = BorderFactory.createCompoundBorder(
                                        redline, compound);
jComp14.setBorder(compound);

//Add a title to the red-outlined frame.
compound = BorderFactory.createTitledBorder(
                                        compound, "title",
                                        TitledBorder.CENTER,
                                        TitledBorder.BELOW_BOTTOM);
jComp15.setBorder(compound);
```

As you probably noticed, the code uses the BorderFactory class to create each border. The BorderFactory class, which is in the javax.swing package, returns objects that implement the Border interface.

The Border interface, as well as its Swing-provided implementations, is in the javax.swing.border package. You often don't need to directly use anything in the border package, except when specifying constants that are specific to a particular border class or when referring to the Border type.

Creating Custom Borders

If BorderFactory doesn't offer you enough control over a border's form, then you might need to directly use the API in the border package — or even define your own border. In addition to containing the Border interface, the border package contains the classes that implement the borders you've already seen: LineBorder, EtchedBorder, BevelBorder, EmptyBorder, MatteBorder, TitledBorder, and CompoundBorder. The border package also contains a class named SoftBevelBorder, which produces a result similar to BevelBorder, but with softer edges.

If none of the Swing borders is suitable, you can implement your own border. Generally, you do this by creating a subclass of the AbstractBorder class. In your subclass, you must implement at least one constructor and the following two methods:

- paintBorder, which contains the drawing code that a JComponent executes to draw the border.
- getBorderInsets, which specifies the amount of space the border needs to draw itself.

If a custom border has insets (and they typically have insets) you need to override both AbstractBorder.getBorderInsets(Component c) and AbstractBorder.getBorderInsets(Component c, Insets insets) to provide the correct insets.

For examples of implementing borders, see the source code for the classes in the javax.swing.border package.

The Border API

The following tables list the commonly used border methods. The API for using borders falls into two categories:

- Creating a Border with BorderFactory
- Setting or Getting a Component's Border

Creating a Border with BorderFactory

Method	**Purpose**
Border createLineBorder(Color) Border createLineBorder(Color, int)	Create a line border. The first argument is a java.awt.Color object that specifies the color of the line. The optional second argument specifies the width in pixels of the line.
Border createEtchedBorder() Border createEtchedBorder(Color, Color) Border createEtchedBorder(int) Border createEtchedBorder(int, Color, Color)	Create an etched border. The optional Color arguments specify the highlight and shadow colors to be used. In release 1.3, methods with int arguments were added that allow the border methods to be specified as either EtchedBorder.RAISED or EtchedBorder.LOWERED. The methods without the int arguments create a lowered etched border.
Border createLoweredBevelBorder()	Create a border that gives the illusion of the component being lower than the surrounding area.
Border createRaisedBevelBorder()	Create a border that gives the illusion of the component being higher than the surrounding area.
Border createBevelBorder(int, Color, Color) Border createBevelBorder(int, Color, Color, Color, Color)	Create a raised or lowered beveled border, specifying the colors to use. The integer argument can be either BevelBorder.RAISED or BevelBorder.LOWERED. With the three-argument constructor, you specify the highlight and shadow colors. With the five-argument constructor, you specify the outer highlight, inner highlight, outer shadow, and inner shadow colors, in that order.

Border createEmptyBorder() Border createEmptyBorder(int, int, int, int)	Create an invisible border. If you specify no arguments, then the border takes no space, which is useful when creating a titled border with no visible boundary. The optional arguments specify the number of pixels that the border occupies at the top, left, bottom, and right (in that order) of whatever component uses it. This method is useful for putting empty space around your components.
MatteBorder createMatteBorder(int, int, int, int, Color) MatteBorder createMatteBorder(int, int, int, int, Icon)	Create a matte border. The integer arguments specify the number of pixels that the border occupies at the top, left, bottom, and right (in that order) of whatever component uses it. The color argument specifies the color which with the border should fill its area. The icon argument specifies the icon which with the border should tile its area.
TitledBorder createTitledBorder(String) TitledBorder createTitledBorder(Border) TitledBorder createTitledBorder(Border, String) TitledBorder createTitledBorder(Border, String, int, int) TitledBorder createTitledBorder(Border, String, int, int, Font) TitledBorder createTitledBorder(Border, String, int, int, Font, Color)	Create a titled border. The string argument specifies the title to be displayed. The optional font and color arguments specify the font and color to be used for the title's text. The border argument specifies the border that should be displayed along with the title. If no border is specified, then a look-and-feel-specific default border is used. By default, the title straddles the top of its companion border and is left-justified. The optional integer arguments specify the title's position and justification, in that order. TitledBorder defines these possible positions: ABOVE_TOP, TOP (the default), BELOW_TOP, ABOVE_BOTTOM, BOTTOM, and BELOW_BOTTOM. You can specify the justification as LEADING (the default), CENTER, or TRAILING. In locales with Western alphabets LEADING is equivalent to LEFT and TRAILING is equivalent to RIGHT.
CompoundBorder createCompoundBorder(Border, Border)	Combine two borders into one. The first argument specifies the outer border; the second, the inner border.

Setting or Getting a Com+ponent's Border

Method	**Purpose**
void setBorder(Border) Border getBorder()	Set or get the border of the receiving JComponent.
void setBorderPainted(boolean) boolean isBorderPainted() **(in** *AbstractButton***,** *JMenuBar***,** *JPopupMenu***,** *JProgressBar***, and** *JToolBar***)**	Set or get whether the border of the component should be displayed.

Examples that Use Borders

Many examples in this lesson use borders. The following table lists a few interesting cases.

Example	**Where Described**	**Notes**
BorderDemo	This section	Shows an example of each type of border that BorderFactory can create. Also uses an empty border to add breathing space between each pane and its contents.
BoxAlignmentDemo	How to Use BoxLayout	Uses titled borders.
BoxLayoutDemo	How to Use BoxLayout	Uses a red line to show where the edge of a container is, so that you can see how the extra space in a box layout is distributed.
ComboBoxDemo2	How to Use Combo Boxes	Uses a compound border to combine a line border with an empty border. The empty border provides space between the line and the component's innards.

SYSTEM DESIGN

7.1. INTRODUCTION

Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm and area of application. Design is the first step in the development phase for any engineered product or system. The designer‟s goal is to produce a model or representation of an entity that will later be built. Beginning, once system requirement have been specified and analyzed, system design is the first of the three technical activities -design, code and test that is required to build and verify software.

The importance can be stated with a single word "Quality". Design is the place where quality is fostered in software development. Design provides us with representations of software that can assess for quality. Design is the only way that we can accurately translate a customer‟s view into a finished software product or system. Software design serves as a foundation for all the software engineering steps that follow. Without a strong design we risk building an unstable system – one that will be difficult to test, one whose quality cannot be assessed until the last stage.

During design, progressive refinement of data structure, program structure, and procedural details are developed reviewed and documented. System design can be viewed from either technical or project management perspective. From the technical point of view, design is comprised of four activities – architectural design, data structure design, interface design and procedural design.

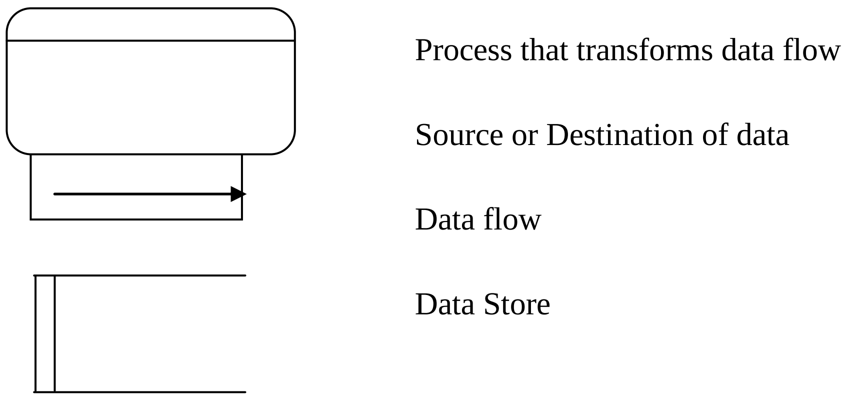
7.2. DATA FLOW DIAGRAM

A data flow diagram is graphical tool used to describe and analyze movement of data through a system. These are the central tool and the basis from which the other components are developed. The transformation of data from input to output, through processed, may be described logically and independently of physical components associated with the system. These are known as the logical data flow diagrams. The physical data flow diagrams show the actual implements and movement of data between people, departments and workstations. A full description of a system actually consists of a set of data flow diagrams. Using two familiar notations Yourdon, Gane and Sarson notation develops the data flow diagrams. Each component in a DFD is labeled with a descriptive name. Process is further identified with a number that will be used for identification purpose. The development of DFD'S is done in several levels. Each process in lower level diagrams can be broken down into a more detailed DFD in the next level. The lop-level diagram is often called context diagram. It consist a single process bit, which plays vital role in studying the current system. The process in the context level diagram is exploded into other process at the first level DFD.The idea behind the explosion of a process into more process is that understanding at one level of detail is exploded into greater detail at the next level. This is done until further explosion is necessary and an adequate amount of detail is described for analyst to understand the process.Larry Constantine first developed the DFD as a way of expressing system requirements in a graphical from, this lead to the modular design.A DFD is also known as a "bubble Chart" has the purpose of clarifying system requirements and identifying major transformations that will become programs in system design. So it is the starting point of the design to the lowest level of detail. A DFD consists of a series of bubbles joined by data flows in the system.

DFD SYMBOLS:

In the DFD, there are four symbols

1. A square defines a source(originator) or destination of system data
2. An arrow identifies data flow. It is the pipeline through which the information flows
3. A circle or a bubble represents a process that transforms incoming data flow into outgoing data flows.
4. An open rectangle is a data store, data at rest or a temporary repository of data


Process that transforms data flow
Source or Destination of data
Data flow
Data Store

CONSTRUCTING A DFD:

Several rules of thumb are used in drawing DFD'S:

1. Process should be named and numbered for an easy reference. Each name should be representative of the process.

2. The direction of flow is from top to bottom and from left to right. Data traditionally flow from source to the destination although they may flow back to the source. One way to indicate this is to draw long

flow line back to a source. An alternative way is to repeat the source symbol as a destination. Since it is used more than once in the DFD it is marked with a short diagonal.

3. When a process is exploded into lower level details, they are numbered.
4. The names of data stores and destinations are written in capital letters. Process and dataflow names have the first letter of each work capitalized

A DFD typically shows the minimum contents of data store. Each data store should contain all the data elements that flow in and out.

Questionnaires should contain all the data elements that flow in and out. Missing interfaces redundancies and like is then accounted for often through interviews.

SAILENT FEATURES OF DFD'S

1. The DFD shows flow of data, not of control loops and decision are controlled considerations do not appear on a DFD.
2. The DFD does not indicate the time factor involved in any process whether the dataflow take place daily, weekly, monthly or yearly.
3. The sequence of events is not brought out on the DFD.

TYPES OF DATA FLOW DIAGRAMS

1. Current Physical
2. Current Logical
3. New Logical
4. New Physical

CURRENT PHYSICAL:

In Current Physical DFD process label include the name of people or their positions or the names of computer systems that might provide some of the overall system-processing label includes an identification of the technology used to process the data. Similarly data flows and data stores are often labels with the names of the actual physical media on which data are stored such as file folders, computer files, business forms or computer tapes.

CURRENT LOGICAL:

The physical aspects at the system are removed as much as possible so that the current system is reduced to its essence to the data and the processors that transform them regardless of actual physical form.

NEW LOGICAL:

This is exactly like a current logical model if the user were completely happy with the user were completely happy with the functionality of the current system but had problems with how it was implemented typically through the new logical model will differ from current logical model while having additional functions, absolute function removal and inefficient flows recognized.

NEW PHYSICAL:

The new physical represents only the physical implementation of the new system.

RULES GOVERNING THE DFD'S

PROCESS

1) No process can have only outputs.

2) No process can have only inputs. If an object has only inputs than it must be a sink.

3) A process has a verb phrase label.

DATA STORE

1) Data cannot move directly from one data store to another data store, a process must move data.

2) Data cannot move directly from an outside source to a data store, a process, which receives, must move data from the source and place the data into data store

3) A data store has a noun phrase label.

SOURCE OR SINK

The origin and /or destination of data

1) Data cannot move direly from a source to sink it must be moved by a process

2) A source and /or sink has a noun phrase land

DATA FLOW

1) A Data Flow has only one direction of flow between symbols. It may flow in both directions between a process and a data store to show a read before an update. The later is usually indicated however by two separate arrows since these happen at different type.

2) A join in DFD means that exactly the same data comes from any of two or more different processes data store or sink to a common location.

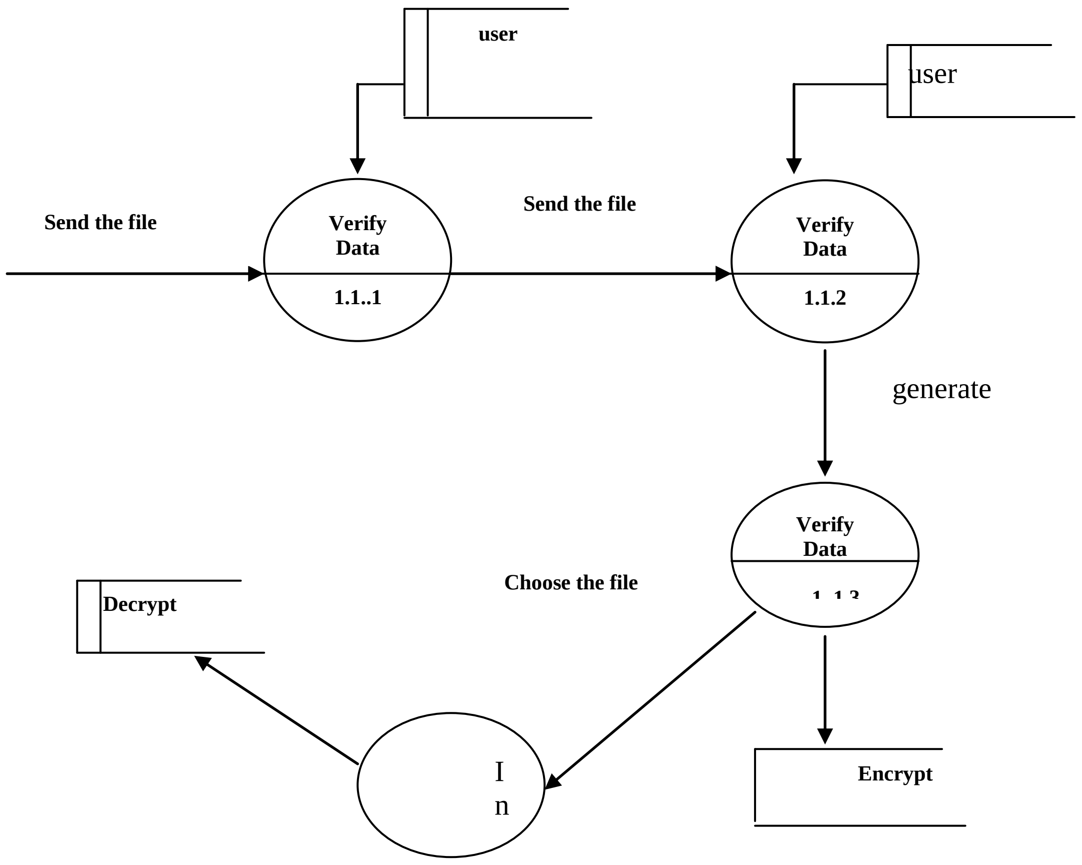
3) A data flow cannot go directly back to the same process it leads. There must be at least one other process that handles the data flow produce some other data flow returns the original data into the beginning process.

4) A Data flow to a data store means update (delete or change).

5) A data Flow from a data store means retrieve or use.

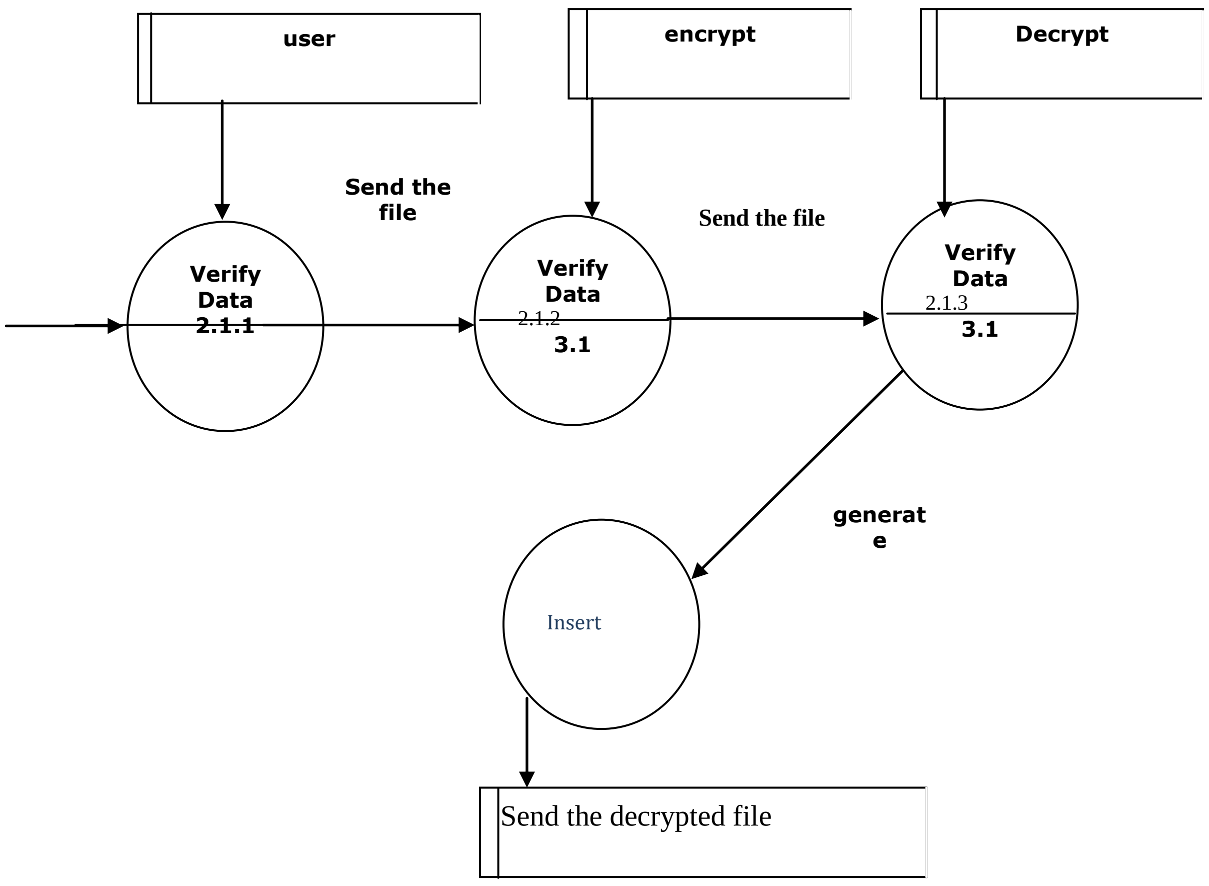
   A data flow has a noun phrase label more than one data flow noun phrase can appear on a single arrow as long as all of the flows on the same arrow move together as one package.

Level-1 **First level or SystemLevel Dataflow:**
+++
DFD for User Operation



Level 2: Second level or SubSystemLevel Dataflow:

DFD for User Operation



7.3. ACTIVITY DIAGRAM

A State diagram/Activity diagram is a specification of the sequences of states that an object or an interaction goes through in response to events during its life, together with its responsive action. Every state diagram is having one entry and exit state. And the state can have any number of sub-states. The above state diagram represents, how admin will interact with other objects, and how he will perform actions and change his state.

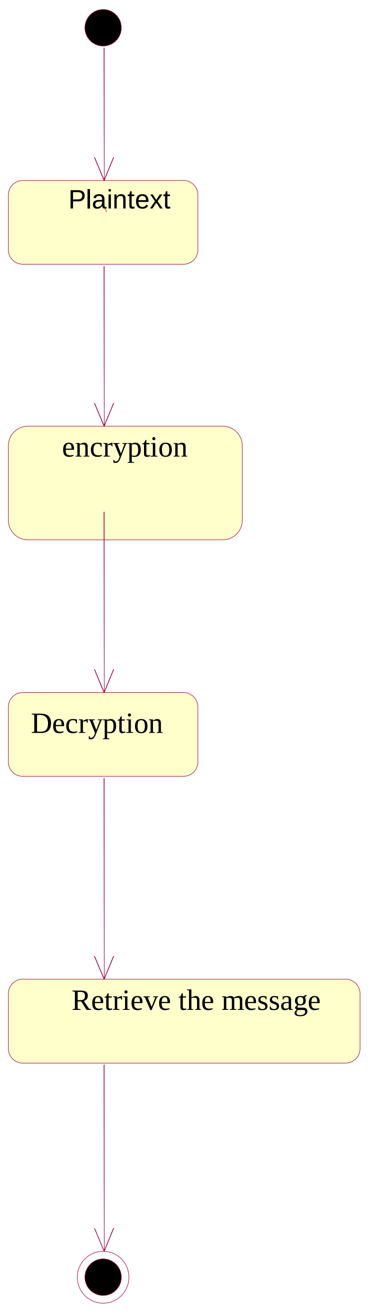


Figure 6.6: activity diagram

7.4. SEQUENCE DIAGRAM

A sequence diagram is a graphical view of a scenario that shows object interaction in a time-based sequence what happens first, what happens next. Sequence diagrams establish the roles of objects and help provide essential information to determine class responsibilities and interfaces.

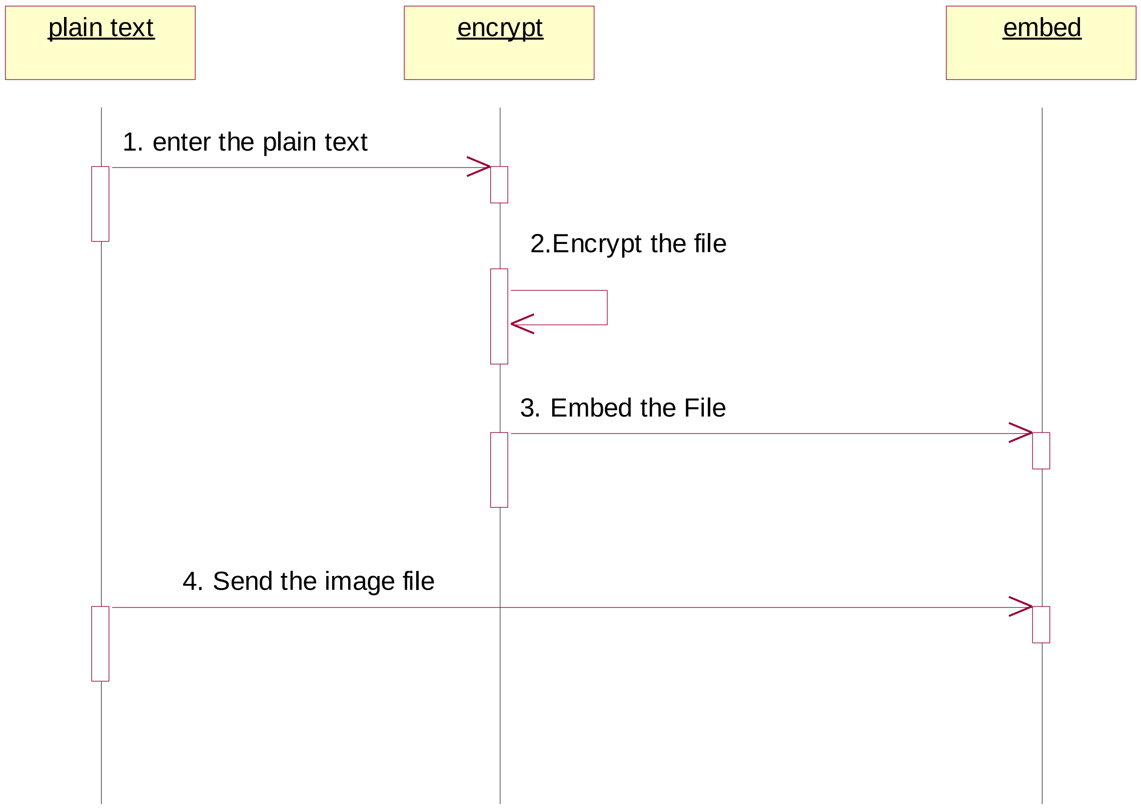
There are two main differences between sequence and collaboration diagrams: sequence diagrams show time-based object interaction while collaboration diagrams show how objects associate with each other.

Object: An object has state, behavior, and identity. The structure and behavior of similar objects are defined in their common class. Each object in a diagram indicates some instance of a class. An object that is not named is referred to as a class instance. The object icon is similar to a class icon except that the name is underlined. An object's concurrency is defined by the concurrency of its class.

Message: A message is the communication carried between two objects that trigger an event. A message carries information from the source focus of control to the destination focus of control. The synchronization of a message can be modified through the message specification. Synchronization means a message where the sending object pauses to wait for results.

Link: A link should exist between two objects, including class utilities, only if there is a relationship between their corresponding classes. The link is depicted as a straight line between objects or objects and class instances in a collaboration diagram. If an object links to itself, use the loop version of the icon.

SEQUENCE DIAGRAM FOR EMBEDDING:



Sequence diagram for embedding.

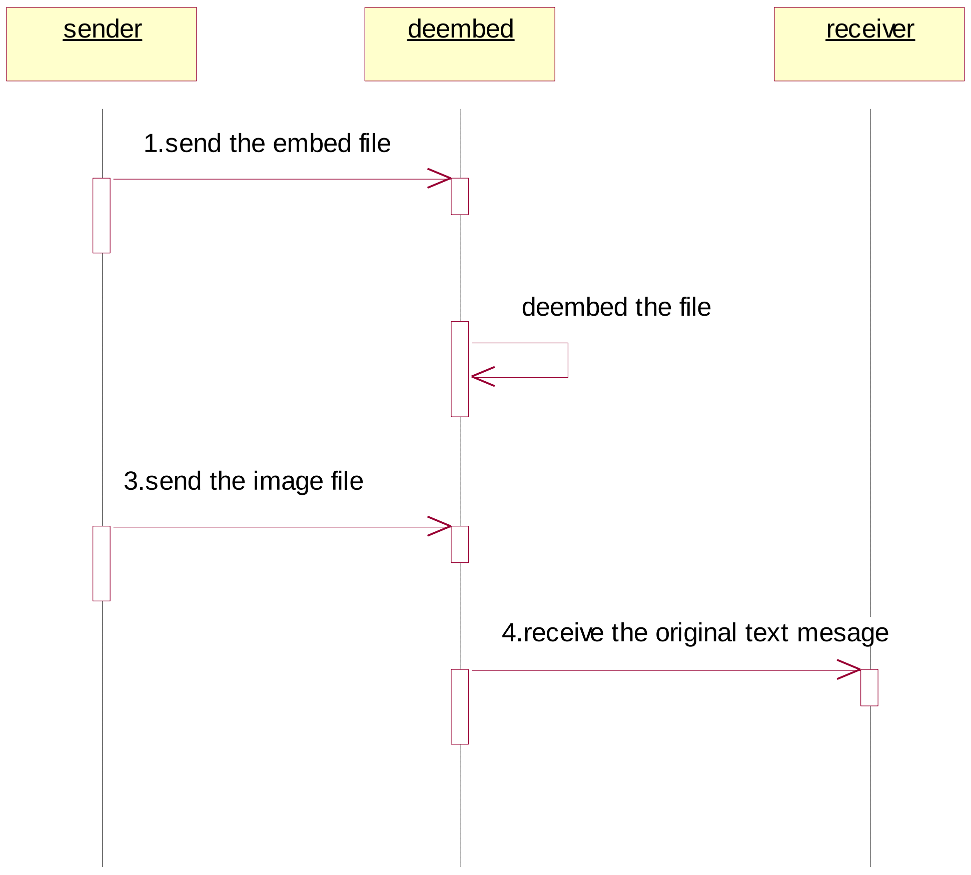
Objects:

Embed

Messages:

Image file is selected & text file is embedded into it.

SEQUENCE DIAGRAM FOR DEEMBEDING:



:Sequence diagram for Deembeding

Objects:

Deembed

Messages:

Embeded file is deembeded & we get the separated text file.

7.5. CLASS DIAGRAM

Identification of analysis classes:

A class is a set of objects that share a common structure and common behavior (the same attributes, operations, relationships and semantics). A class is an abstraction of real-world items.

There are 4 approaches for identifying classes:

1. Noun phrase approach:
2. Common class pattern approach.
3. Use case Driven Sequence or Collaboration approach.
4. Classes , Responsibilities and collaborators Approach

1. Noun Phrase Approach:

The guidelines for identifying the classes:

a. Look for nouns and noun phrases in the use cases.
b. Some classes are implicit or taken from general knowledge.
c. All classes must make sense in the application domain; Avoid computer implementation classes – defer them to the design stage.
d. Carefully choose and define the class names.

After identifying the classes we have to eliminate the following types of classes:

a. Redundant classes.
b. Adjective classes.

2. Common class pattern approach:

The following are the patterns for finding the candidate classes:

a. Concept class.
b. Events class.

c. Organization class

d. Peoples class

e. Places class

f. Tangible things and devices class.

3. **Use case driven approach:**

We have to draw the sequence diagram or collaboration diagram. If there is need for some classes to represent some functionality then add new classes which perform those functionalities.

4. **CRC approach:**

The process consists of the following steps:

a. Identify classes' responsibilities (and identify the classes)
b. Assign the responsibilities
c. Identify the collaborators.

Identification of responsibilities of each class:

The questions that should be answered to identify the attributes and methods of a class respectively are:

1. What information about an object should we keep track of?
2. What services must a class provide?

Identification of relationships among the classes:

Three types of relationships among the objects are:

Association: How objects are associated?

Super-sub structure: How are objects organized into super classes and sub classes?

Aggregation: What is the composition of the complex classes?

Guidelines for identifying the tentative associations:

- A dependency between two or more classes may be an association. Association often corresponds to a verb or prepositional phrase.
- A reference from one class to another is an association. Some associations are implicit or taken from general knowledge.

Super-sub class relationships

Super-sub class hierarchy is a relationship between classes where one class is the parent class of another class (derived class).This is based on inheritance. This hierarchy is represented with Generalization.

Guidelines for identifying the super-sub relationship, a generalization are

1**. Top-down*:*** Look for noun phrases composed of various adjectives in a class name. Avoid excessive refinement. Specialize only when the sub classes have significant behavior.

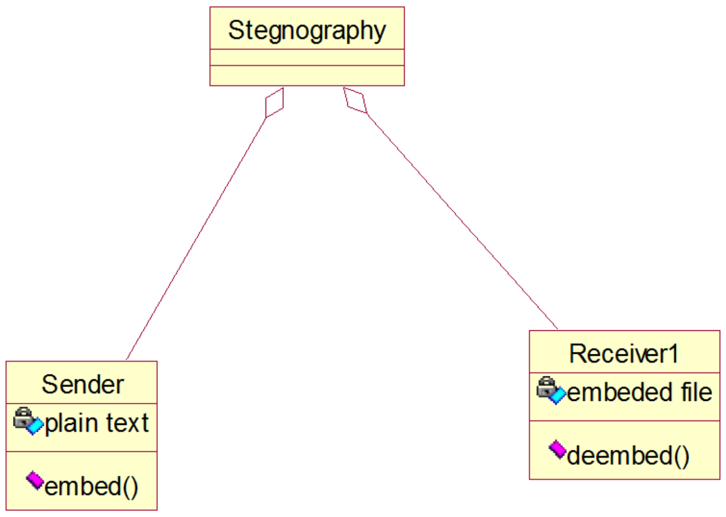
2. **Bottom-up*:*** Look for classes with similar attributes or methods. Group them by moving the common attributes and methods to an abstract class. You may have to alter the definitions a bit.

3. **Reusability*:*** Move the attributes and methods as high as possible in the hierarchy.

4**. Multiple inheritances*:*** Avoid excessive use of multiple inheritances. One way of getting benefits of multiple inheritances is to inherit from the most appropriate class and add an object of another class as an attribute`The class diagram is core to object-oriented design. It describes the types of objects in the system and the static relationships between them.

Packages

Packages allow you to break up a large number of objects into related groupings. In many object oriented languages (such as Java), packages are used to provide scope and division to classes and interfaces. In the UML, packages serve a similar, but broader purpose


Stegnography
Sender
plain text
embed()
Receiver1
embeded file
deembed()

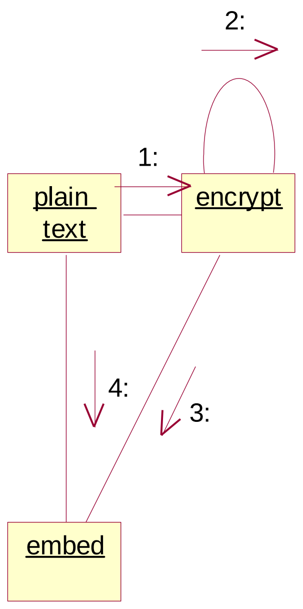
7.6. COLLABORATION DIAGRAM

A Communication diagram models the interactions between objects or parts in terms of sequenced messages. Communication diagrams represent a combination of information taken from Class, Sequence, and Use Case Diagrams describing both the static structure and dynamic behavior of a system.

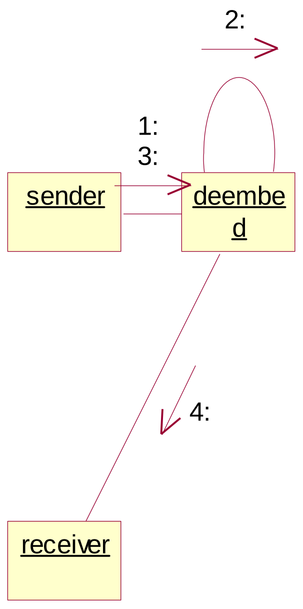
However, communication diagrams use the free-form arrangement of objects and links as used in Object diagrams. In order to maintain the ordering of messages in such a free-form diagram, messages are labeled with a chronological number and placed near the link the message is sent over. Reading a communication diagram involves starting at message 1.0, and following the messages from object to object.

Communication diagrams show a lot of the same information as sequence diagrams, but because of how the information is presented, some of it is easier to find in one diagram than the other. Communication diagrams show which elements each one interacts with better, but sequence diagrams show the order in which the interactions take place more clearly.

Collaboration Diagram for Embedding



Collaboration Diagram for DeEmbedding



CODING

```
import javax.swing.*;
import javax.swing.border.TitledBorder;
import java.awt.*;
import java.awt.event.*;

public class MainClient extends WindowAdapter implements ActionListener
{
        private JFrame              mainFrame;
        private JMenuBar            menuBar;
        private JMenu               menuFile, menuEdit, menuView, menuHelp, menuLookAndFeel;
        private JMenuItem           mnuExit, mnuEmbedMessage, mnuEmbedFile, mnuHelp, mnuAbout;
        private JMenuItem           mnuRetrieveMessage, mnuRetrieveFile, mnuModifyMaster;
        private JRadioButtonMenuItem    mnuTonicFeel, mnuMetalFeel, mnuMotifFeel, mnuWindowsFeel;
        private ButtonGroup lookAndFeelButtonGroup;

        private JPanel mainPanel, panelAbout, panelButtons;
        private JLabel lblLogo;
  private JLabel lblFiller[], lblName, lblEmail, lblPhone;
        private GridBagLayout gbl;
        private GridBagConstraints gbc;
        private MyJButton btnEmbedFile, btnRetrieveFile, btnEmbedMessage, btnRetrieveMessage;
        private MyJButton btnHelp, btnAbout;
        private BackEndHandler back;

        private MainClient()
        {
                mainFrame= new JFrame("Steganography With Audio, Video and Image");
                mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                mainFrame.addWindowListener(this);

                // Setup the menu bar
                mnuExit= new MyJMenuItem("Exit", 1, 'x');
                mnuEmbedMessage= new MyJMenuItem("Embed Message", 6, 'm');
```

```
//mnuEmbedFile= new MyJMenuItem("Embed File", 7, 'i');
mnuRetrieveMessage= new MyJMenuItem("Retrieve Message", 0, 'r');
//mnuRetrieveFile= new MyJMenuItem("Retrieve File", 2, 't');
mnuModifyMaster= new MyJMenuItem("Modify Master file settings", 2, 'd');
mnuHelp= new MyJMenuItem("Help", 0, 'h');
mnuAbout= new MyJMenuItem("About", 0, 'a');
mnuTonicFeel= new MyJRadioButtonMenuItem("Plastic XP", 8, 'x');
mnuMetalFeel= new MyJRadioButtonMenuItem("Metal", 0, 'm');
mnuMotifFeel= new MyJRadioButtonMenuItem("Motif", 2, 't');
mnuWindowsFeel= new MyJRadioButtonMenuItem("Windows", 0, 'w');

// Add item listener for Look and feel menu items
RadioListener radioListener= new RadioListener();
mnuTonicFeel.addItemListener(radioListener);
mnuMetalFeel.addItemListener(radioListener);
mnuMotifFeel.addItemListener(radioListener);
mnuWindowsFeel.addItemListener(radioListener);
mnuTonicFeel.setSelected(true);

lookAndFeelButtonGroup= new ButtonGroup();
lookAndFeelButtonGroup.add(mnuTonicFeel);
lookAndFeelButtonGroup.add(mnuMetalFeel);
lookAndFeelButtonGroup.add(mnuMotifFeel);
lookAndFeelButtonGroup.add(mnuWindowsFeel);

// Add action listeners for other menu items
mnuEmbedMessage.addActionListener(this);
//mnuEmbedFile.addActionListener(this);
mnuRetrieveMessage.addActionListener(this);
//mnuRetrieveFile.addActionListener(this);
mnuModifyMaster.addActionListener(this);
mnuExit.addActionListener(this);
mnuHelp.addActionListener(this);
mnuAbout.addActionListener(this);

menuFile= new MyJMenu("File", 0, 'f');
menuFile.add(mnuEmbedMessage);
```

```
//menuFile.add(mnuEmbedFile);
menuFile.add(mnuRetrieveMessage);
//menuFile.add(mnuRetrieveFile);
menuFile.add(mnuExit);

menuEdit= new JMenu("Edit");
menuEdit.add(mnuModifyMaster);

menuLookAndFeel= new MyJMenu("Look and Feel...", 0, 'l');
menuLookAndFeel.add(mnuTonicFeel);
menuLookAndFeel.add(mnuMetalFeel);
menuLookAndFeel.add(mnuMotifFeel);
menuLookAndFeel.add(mnuWindowsFeel);
menuView= new MyJMenu("View", 0, 'v');
menuView.add(menuLookAndFeel);

menuHelp= new MyJMenu("Help", 0, 'h');
menuHelp.add(mnuHelp);
//menuHelp.add(mnuAbout);

menuBar= new JMenuBar();
menuBar.add(menuFile);
//menuBar.add(menuEdit);
//menuBar.add(menuView);
menuBar.add(menuHelp);
mainFrame.setJMenuBar(menuBar);

mainPanel= new JPanel();
panelAbout= new JPanel();
panelButtons= new JPanel();



// Create filler labels
lblFiller= new JLabel[4];
for(int i=0; i<4; i++)
        lblFiller[i]= new JLabel(" ");
```

```
                // Prepare About panel
                gbl= new GridBagLayout();
                gbc= new GridBagConstraints();
                panelAbout.setLayout(gbl);
                panelAbout.setBackground(Color.white);
                Color myColor= new Color(50, 153, 237);
                Font arialFont= new Font("Arial", Font.PLAIN, 14);
                Font myFont= new Font("Monotype Corsiva", Font.PLAIN, 18);

                // Prepare the Buttons panel
                panelButtons.setBackground(Color.white);
                gbl= new GridBagLayout();
                panelButtons.setLayout(gbl);
                //panelButtons.setBorder(new TitledBorder("Supported operations"));

                //lblLogo= new JLabel(new ImageIcon("D://Nidheesh//jins//Images//nnn.jpeg"));
     lblLogo= new JLabel(new ImageIcon("./Images/Logo.bmp"));
                btnEmbedMessage= new
MyJButton("./Images/EmbedMessage.bmp","./Images/EmbedMessageHover.bmp");
                btnEmbedFile= new MyJButton("./Images/EmbedFile.bmp", "./Images/EmbedFileHover.bmp");
                btnRetrieveMessage= new MyJButton("./Images/RetrieveMessage.bmp",
"./Images/RetrieveMessageHover.bmp");
                btnRetrieveFile= new MyJButton("./Images/RetrieveFile.bmp",
"./Images/RetrieveFileHover.bmp");
                btnHelp= new MyJButton("./Images/Help.bmp", "./Images/HelpHover.bmp");
                btnAbout= new MyJButton("./Images/About.bmp", "./Images/AboutHover.bmp");

                // Add action listeners for the buttons
                btnEmbedMessage.addActionListener(this);
                btnEmbedFile.addActionListener(this);
                btnRetrieveMessage.addActionListener(this);
                btnRetrieveFile.addActionListener(this);
                btnHelp.addActionListener(this);
                btnAbout.addActionListener(this);

                // Add filler for rows 1 and 2
                gbc.weightx= 4;gbc.weighty= 2;gbc.fill= gbc.BOTH;
gbc.gridx= 6;    gbc.gridy= 1;    gbl.setConstraints(lblFiller[0], gbc);
panelButtons.add(lblFiller[0]);

gbc.weightx= 1;gbc.weighty= 1;gbc.fill= gbc.NONE;
gbc.gridx= 3;    gbc.gridy= 3;    gbl.setConstraints(btnHelp, gbc);
//panelButtons.add(btnHelp);

gbc.gridx= 5;    gbl.setConstraints(btnAbout, gbc);
//panelButtons.add(btnAbout);

// Add filler for rows 4 and 5
gbc.fill = gbc.BOTH;
gbc.gridx= 1;    gbc.weighty= 2;gbc.gridy= 4;    gbl.setConstraints(lblFiller[1], gbc);
panelButtons.add(lblFiller[1]);

gbc.fill= gbc.NONE;
gbc.gridx= 2;    gbc.weighty= 1;gbc.gridy= 6;    gbl.setConstraints(btnEmbedMessage, gbc);
panelButtons.add(btnEmbedMessage);

gbc.gridx= 4;    gbl.setConstraints(btnRetrieveMessage, gbc);
panelButtons.add(btnRetrieveMessage);

// Add filler for row 7 and 8
gbc.fill = gbc.BOTH;
gbc.gridx= 6;    gbc.weighty= 2;gbc.gridy= 7;    gbl.setConstraints(lblFiller[2], gbc);
panelButtons.add(lblFiller[2]);

gbc.fill= gbc.NONE;
gbc.gridx= 3;    gbc.weighty= 1;gbc.gridy= 9;    gbl.setConstraints(btnEmbedFile, gbc);
panelButtons.add(btnEmbedFile);

gbc.gridx= 5;    gbl.setConstraints(btnRetrieveFile, gbc);
panelButtons.add(btnRetrieveFile);

// Add the lblLogo and two Panels to the mainPanel
gbl= new GridBagLayout();
mainPanel.setLayout(gbl);
```

```
            JLabel label= new JLabel(new ImageIcon("Tech Innova Logo New.jpeg"));
            Box b1= new Box(BoxLayout.X_AXIS);
            b1.add(Box.createHorizontalStrut(300));
            b1.add(Box.createVerticalStrut(700));
            b1.add(label);
            mainPanel.add(b1);

            mainPanel.setBackground(Color.gray);



            gbc.anchor= gbc.CENTER;
            gbc.gridx= 1;    gbc.gridy= 1;    gbc.weighty= 2;gbc.fill= gbc.VERTICAL;
            gbl.setConstraints(lblLogo, gbc);
            mainPanel.add(lblLogo);

            gbc.gridy= 3;    gbc.weighty= 2;
            gbl.setConstraints(panelAbout, gbc);
            mainPanel.add(panelAbout);

            gbc.gridy= 5;    gbc.weighty= 1;
            gbl.setConstraints(panelButtons, gbc);
            mainPanel.add(panelButtons);

            gbc.gridy= 6;    gbc.weighty= 2;
            gbl.setConstraints(lblFiller[3], gbc);
            mainPanel.add(lblFiller[3]);

            JPanel tempPanel= (JPanel) mainFrame.getContentPane();
            tempPanel.add(mainPanel, BorderLayout.CENTER);
            tempPanel.add(new MyJLabel(" ", Color.black, Color.darkGray), BorderLayout.SOUTH);

            Dimension d= Toolkit.getDefaultToolkit().getScreenSize();
            mainFrame.setSize(d.width, (int) (d.height-(d.height*.03)));
            mainFrame.setResizable(false);
            mainFrame.setVisible(true);
    }
```

```
// Listener methods
public void actionPerformed(ActionEvent e)
{
        Object source= e.getSource();

        // Embed message operation
        if(source== mnuEmbedMessage || source== btnEmbedMessage)
        {
                back= new BackEndHandler(this, BackEndHandler.EMBED_MESSAGE);
                back.start();
        }

        // Retrieve message operation
        if(source== mnuRetrieveMessage || source== btnRetrieveMessage)
        {
                back= new BackEndHandler(this, BackEndHandler.RETRIEVE_MESSAGE);
                back.start();
        }

        // Embed file operation



        // Modify Master file operation
        if(source== mnuModifyMaster)
        {
                back= new BackEndHandler(this, BackEndHandler.EDIT_MASTER);
                back.start();
        }



        if(source== mnuHelp || source==btnHelp)
                Steganograph.showHelpDialog();

        if(source== mnuAbout || source== btnAbout)
                Steganograph.showAboutDialog();
```

```
		if(source== mnuExit)
		{
			int result= JOptionPane.showConfirmDialog(mainFrame, "Are you sure that you want to
close .", "Confirm Exit", JOptionPane.YES_NO_OPTION);
			if(result== JOptionPane.YES_OPTION)
			{

				System.exit(0);
			}
		}
	}



	// Class for lissoning to Look and feel radio menu events
	private class RadioListener implements ItemListener
	{
		public void itemStateChanged(ItemEvent e)
		{
			JMenuItem item= (JMenuItem) e.getSource();
			try
			{
				if(item== mnuTonicFeel && mnuTonicFeel.isSelected())

	UIManager.setLookAndFeel("com.jgoodies.looks.plastic.PlasticXPLookAndFeel");

				if(item== mnuMetalFeel && mnuMetalFeel.isSelected())

	UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");

				if(item== mnuMotifFeel && mnuMotifFeel.isSelected())

	UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");

				if(item== mnuWindowsFeel && mnuWindowsFeel.isSelected())

	UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
```

```
                                        SwingUtilities.updateComponentTreeUI(mainFrame);
                                        Steganograph.updateUserInterface();
                              }
                              catch(Exception ex)
                                   {
                                        }
                    }
          }

          // Main method
          public static void main(String args[])
          {
                    new MainClient();
          }
}
```

SYSTEM TESTING AND IMPLEMENTATION

9.1. INTRODUCTION

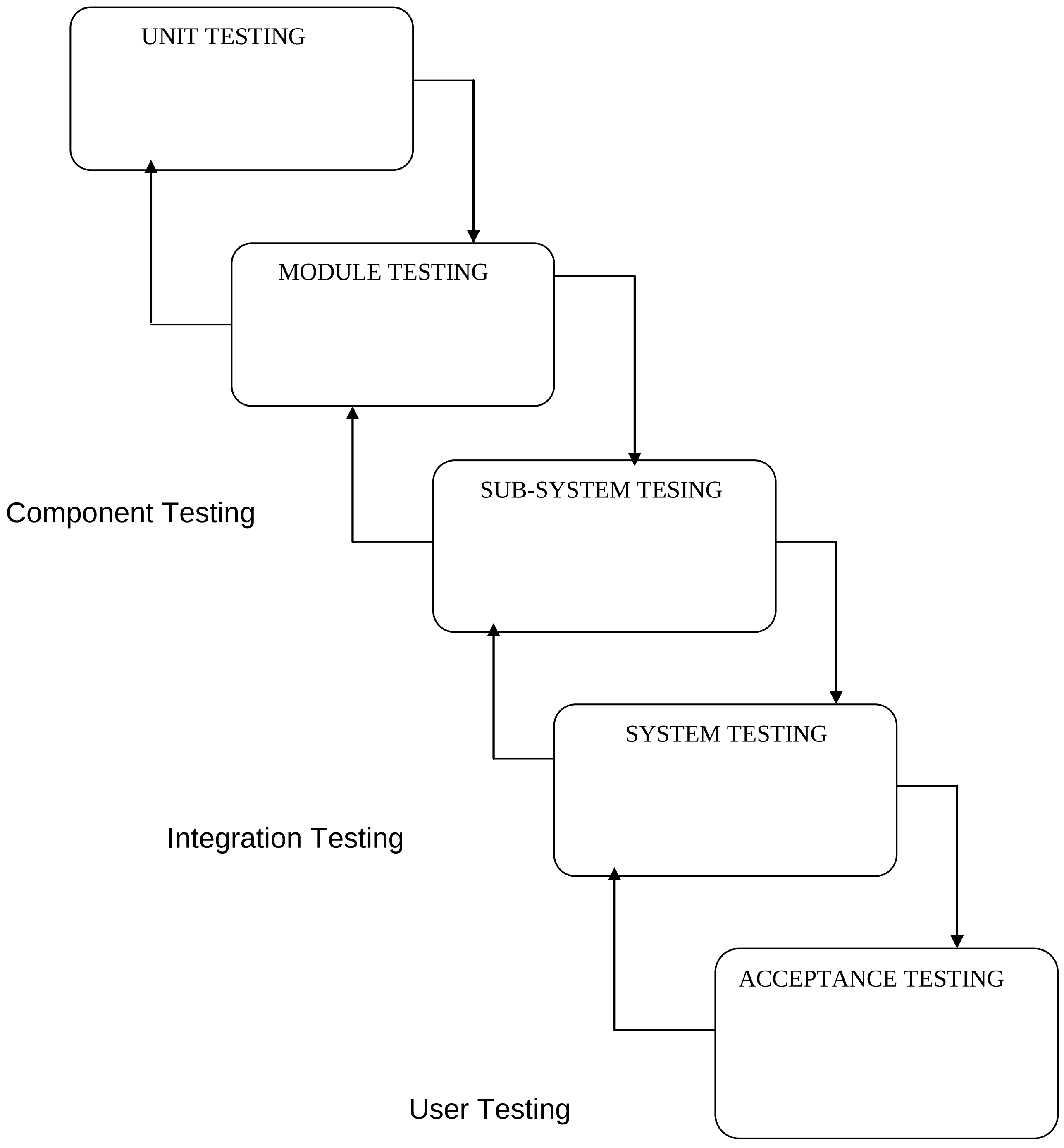
Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. In fact, testing is the one step in the software engineering process that could be viewed as destructive rather than constructive.

A strategy for software testing integrates software test case design methods into a well-planned series of steps that result in the successful construction of software. Testing is the set of activities that can be planned in advance and conducted systematically. The underlying motivation of program testing is to affirm software quality with methods that can economically and effectively apply to both strategic to both large and small-scale systems.

9.2. STRATEGIC APPROACH TO SOFTWARE TESTING

The software engineering process can be viewed as a spiral. Initially system engineering defines the role of software and leads to software requirement analysis where the information domain, functions, behavior, performance, constraints and validation criteria for software are established. Moving inward along the spiral, we come to design and finally to coding. To develop computer software we spiral in along streamlines that decrease the level of abstraction on each turn.

A strategy for software testing may also be viewed in the context of the spiral. Unit testing begins at the vertex of the spiral and concentrates on each unit of the software as implemented in source code. Testing progress is done by moving outward along the spiral to integration testing, where the focus is on the design and the construction of the software architecture. Talking another turn on outward on the spiral we encounter validation testing where requirements established as part of software requirements analysis are validated against the software that has been constructed. Finally we arrive at system testing, where the software and other system elements are tested as a whole.


UNIT TESTING
MODULE TESTING
SUB-SYSTEM TESING
Component Testing
SYSTEM TESTING
Integration Testing
ACCEPTANCE TESTING
User Testing

9.3. UNIT TESTING

Unit testing focuses verification effort on the smallest unit of software design, the module. The unit testing we have is white box oriented and some modules the steps are conducted in parallel.

1. WHITE BOX TESTING

This type of testing ensures that

- All independent paths have been exercised at least once
- All logical decisions have been exercised on their true and false sides
- All loops are executed at their boundaries and within their operational bounds
- All internal data structures have been exercised to assure their validity.

To follow the concept of white box testing we have tested each form .we have created independently to verify that Data flow is correct, All conditions are exercised to check their validity, All loops are executed on their boundaries.

2. BASIC PATH TESTING

Established technique of flow graph with Cyclomatic complexity was used to derive test cases for all the functions. The main steps in deriving test cases were:
Use the design of the code and draw correspondent flow graph.

Determine the Cyclomatic complexity of resultant flow graph, using formula:
V(G)=E-N+2 or
V(G)=P+1 or

V(G)=Number Of Regions

Where V(G) is Cyclomatic complexity,
E is the number of edges,
N is the number of flow graph nodes,
P is the number of predicate nodes.
Determine the basis of set of linearly independent paths.

3. CONDITIONAL TESTING

In this part of the testing each of the conditions were tested to both true and false aspects. And all the resulting paths were tested. So that each path that may be generate on particular condition is traced to uncover any possible errors.

4. DATA FLOW TESTING

This type of testing selects the path of the program according to the location of definition and use of variables. This kind of testing was used only when some local variable were declared. The definition-use chain method was used in this type of testing. These were particularly useful in nested statements.\

5. LOOP TESTING

In this type of testing all the loops are tested to all the limits possible. The following exercise was adopted for all loops:

- All the loops were tested at their limits, just above them and just below them.
- All the loops were skipped at least once.
- For nested loops test the inner most loop first and then work outwards.
- For concatenated loops the values of dependent loops were set with the help of connected loop.
- Unstructured loops were resolved into nested loops or concatenated loops and tested as above.
- Each unit has been separately tested by the development team itself and all the input.

9.4. TEST CASES

Test Case Report 1

(Use one template for each test case)

GENERAL INFORMATION			
Test Stage:	☐Unit ☐Functionality ☐Interface ☐Performance ☐Acceptance		
Test Date:	30/11/2011	**System Date, if applicable:**	30/11/2011
Tester:	Anne Gowthami	**Test Case Number:**	1
Test Case Description:	Unit testing focuses on verifying the effort on the smallest unit of software-module. The local data structure is examined to ensure that the date stored temporarily maintains its integrity during all steps in the algorithm's execution. Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing.		
Results:	☐Pass(OK) ☐Fail		
INTRODUCTION			
Requirement(s) to be tested:	Username Text field and Password Text field and the Authority Text Area.		
Roles and Responsibilities:	Gathering the Requirements of the Project Designing and Testing.		
Set Up Procedures:	By Installing Net Beans.		
ENVIRONMENTAL NEEDS			
Hardware:	PC with Minimum 20GB Hard Disk and 1GB RAM.		
Software:	Windows XP/2000, Net Beans IDE 6.0.		
TEST			
Test Items and Features:	Username and Password		
Procedural Steps:	If the User enters the correct username and password it will be redirected to another appropriate page so that we can confirm test is accepted.		
Expected Results of Case:	If the page is redirected we can confirm the result of this Test case is succeeded.		

Test Case Report 2

(Use one template for each test case)

GENERAL INFORMATION			
Test Stage:	☐Unit ☐Functionality ☐Interface ☐Performance ☐Acceptance		
Test Date:	30/11/2011	**System Date, if applicable:**	30/11/2011
Tester:	Anne Gowthami	**Test Case Number:**	2
Test Case Description:	Unit testing focuses on verifying the effort on the smallest unit of software-module. The local data structure is examined to ensure that the date stored temporarily maintains its integrity during all steps in the algorithm's execution. Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing.		
Results:	☐Pass(OK) ☐Fail		
INTRODUCTION			
Requirement(s) to be tested:	Issuing the books and adding members, calculating the fine etc		
Roles and Responsibilities:	Gathering the Requirements of the Project Designing and Testing.		
Set Up Procedures:	By Installing Net Beans.		
ENVIRONMENTAL NEEDS			
Hardware:	PC with Minimum 20GB Hard Disk and 1GB RAM.		
Software:	Windows XP/2000, Net Beans IDE 6.0.		
TEST			
Test Items and Features:	Issuing books and adding members etc		
Procedural Steps:	If the User enters the correct username and password it will be redirected to another appropriate page so that we can confirm test is accepted.		
Expected Results of Case:	If the page is redirected we can confirm the result of this Test case is succeeded.		

SYSTEM SECURITY

10.1. INTRODUCTION

The protection of computer based resources that includes hardware, software, data, procedures and people against unauthorized use or natural
Disaster is known as System Security.
System Security can be divided into four related issues:

- Security
- Integrity
- Privacy
- Confidentiality

SYSTEM SECURITY refers to the technical innovations and procedures applied to the hardware and operation systems to protect against deliberate or accidental damage from a defined threat.

DATA SECURITY is the protection of data from loss, disclosure, modification and destruction.

SYSTEM INTEGRITY refers to the power functioning of hardware and programs, appropriate physical security and safety against external threats such as eavesdropping and wiretapping.

PRIVACY defines the rights of the user or organizations to determine what information they are willing to share with or accept from others and how the organization can be protected against unwelcome, unfair or excessive dissemination of information about it.

CONFIDENTIALITY is a special status given to sensitive information in a database to minimize the possible invasion of privacy. It is an attribute of information that characterizes its need for protection.

10.2. SECURITY IN SOFTWARE

To set up authentication for Web Applications:

1. Open the web.xml deployment descriptor in a text editor or use the Administration Console.

2. Specify the authentication method using the <auth-method> element. The available options are:

BASIC

Basic authentication uses the Web Browser to display a username/password dialog box. This username and password is authenticated against the realm.

FORM

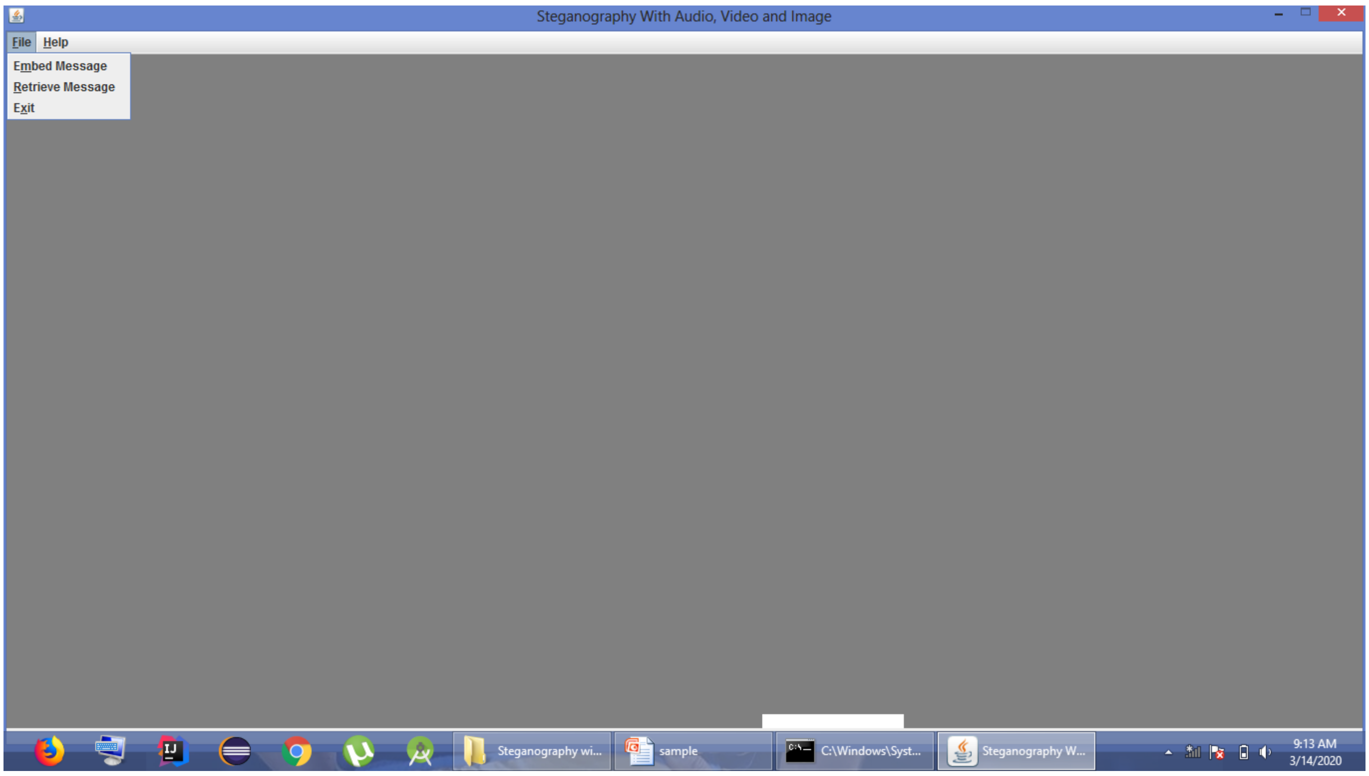
Form-based authentication requires that you return an HTML form containing the username and password. The fields returned from the form elements must be: j_username and j_password, and the action attribute must be j_security_check. Here is an example of the HTML coding for using FORM authentication:

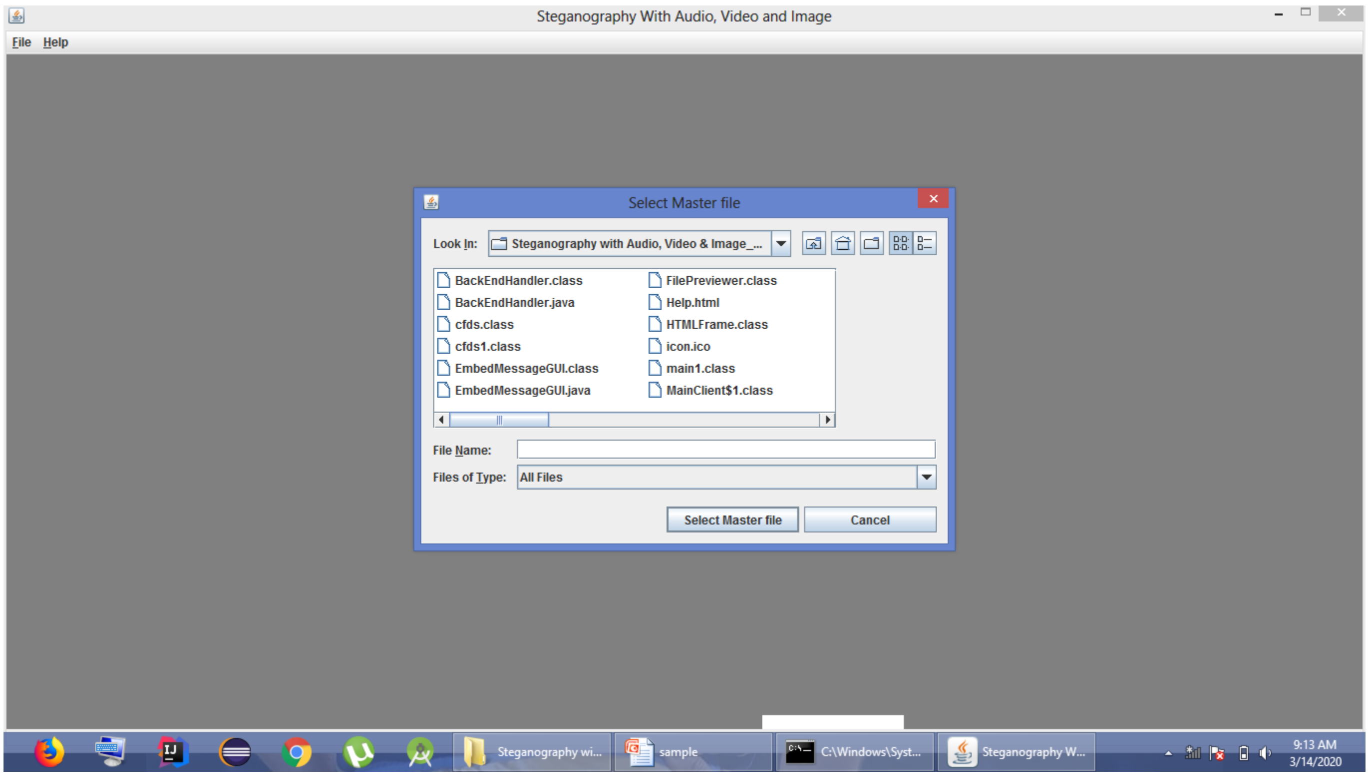
```
<form method="POST" action="j_security_check">
  <inputtype="text"name="j_username">
  <input type="password" name="j_password">
</form>
```

The resource used to generate the HTML form may be an HTML page, a JSP, or a servlet. You define this resource with the <form-login-page> element.

The HTTP session object is created when the login page is served. Therefore, the session.isNew() method returns FALSE when called from pages served after successful authentication.

SCREEN SHOTS

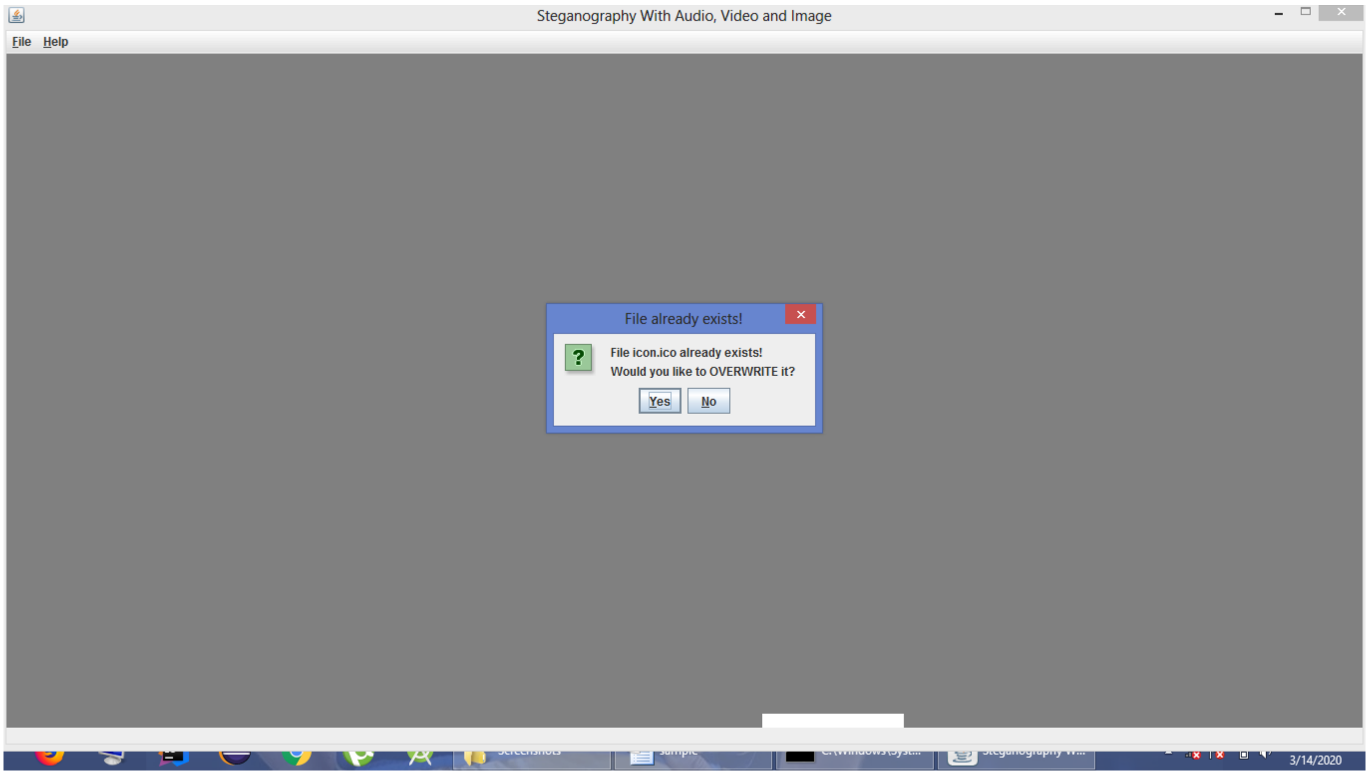



Steganography With Audio, Video and Image
File
Help
Select Master file
Look In:
Steganography with Audio, Video & Image_...

BackEndHandler.class
BackEndHandler.java
cfds.class
cfds1.class
EmbedMessageGUI.class
EmbedMessageGUI.java
FilePreviewer.class
Help.html
HTMLFrame.class
icon.ico
main1.class
MainClient$1.class

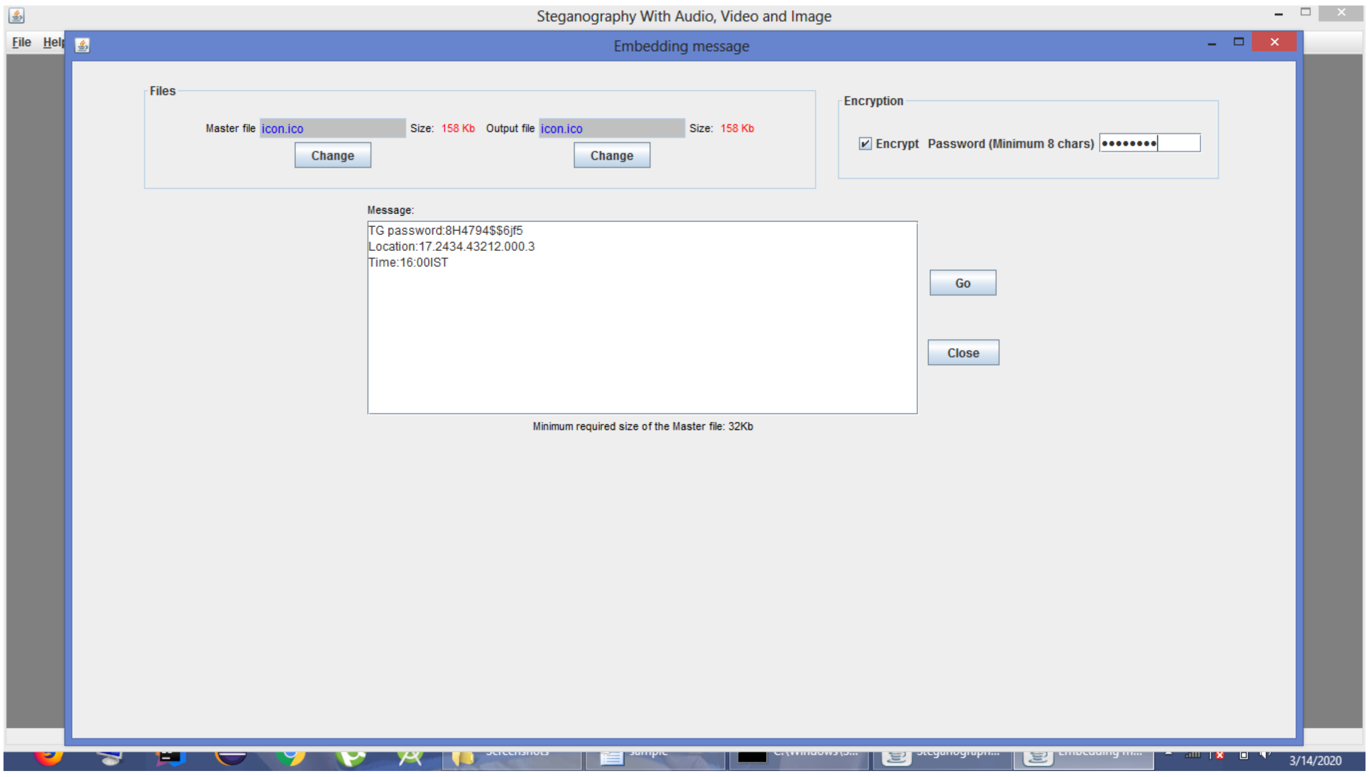
File Name:
Files of Type: All Files

Select Master file
Cancel



File already exists!

File icon.ico already exists!
Would you like to OVERWRITE it?



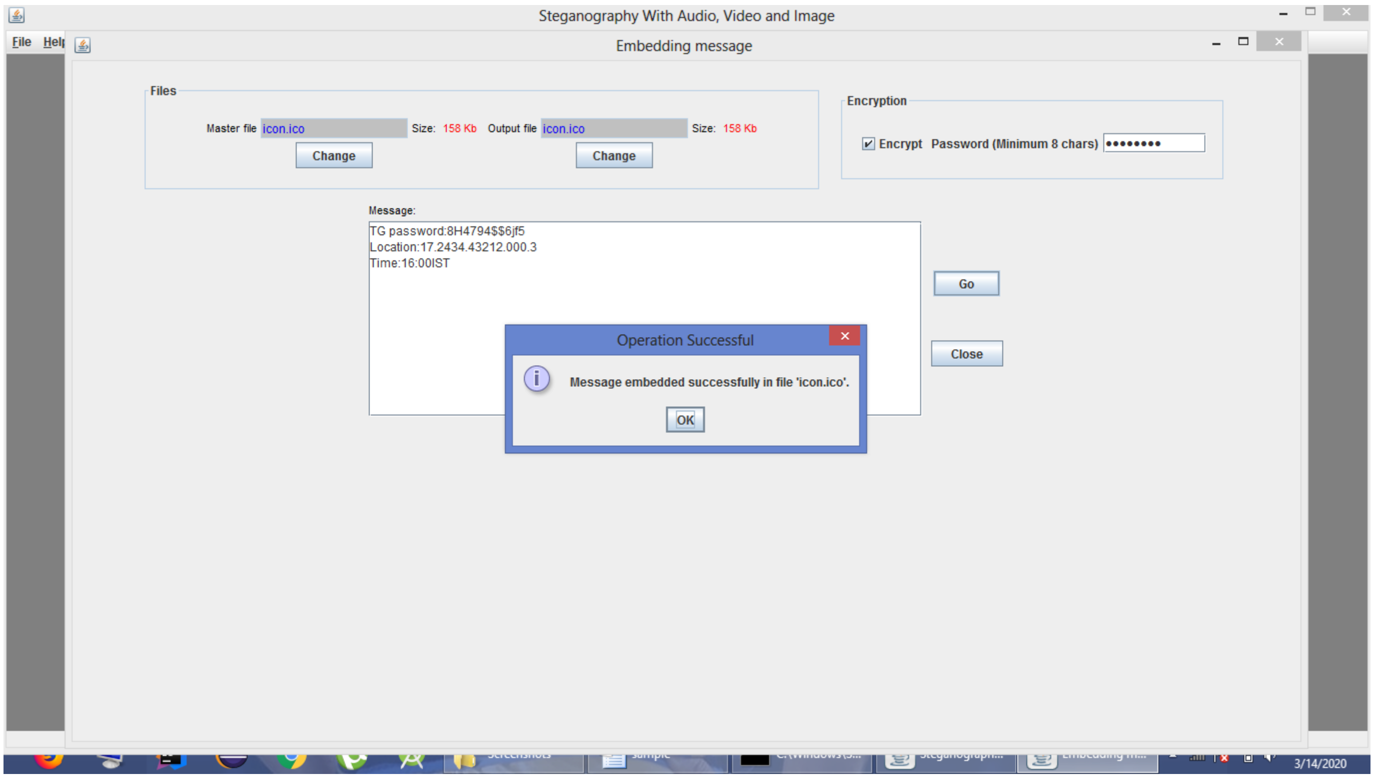
Master file icon.ico Size: 158 Kb Output file icon.ico Size: 158 Kb

✓ Encrypt Password (Minimum 8 chars) ••••••••

Message:

```
TG password:8H4794$$6jf5
Location:17.2434.43212.000.3
Time:16:00IST
```

Minimum required size of the Master file: 32Kb



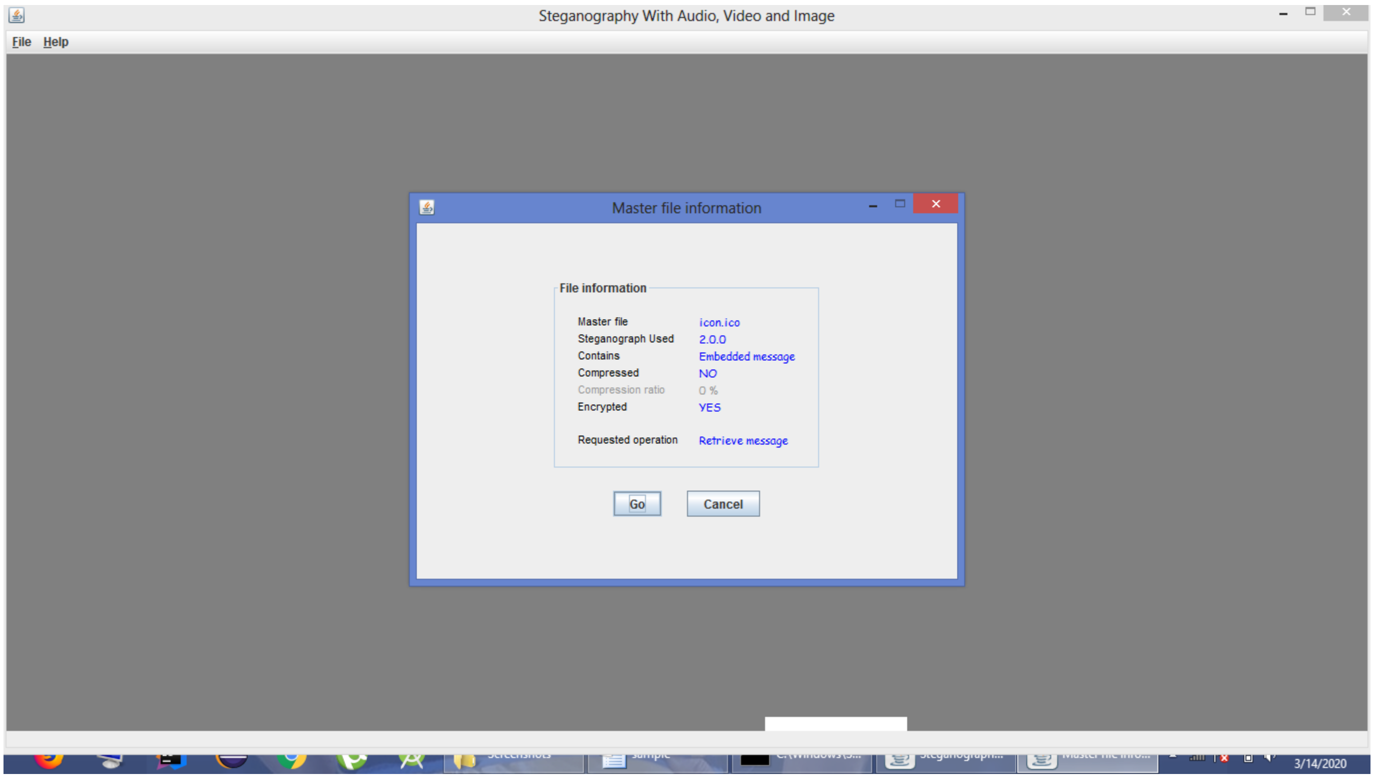
Master file icon.ico Size: 158 Kb Output file icon.ico Size: 158 Kb

✓ Encrypt Password (Minimum 8 chars) ••••••••

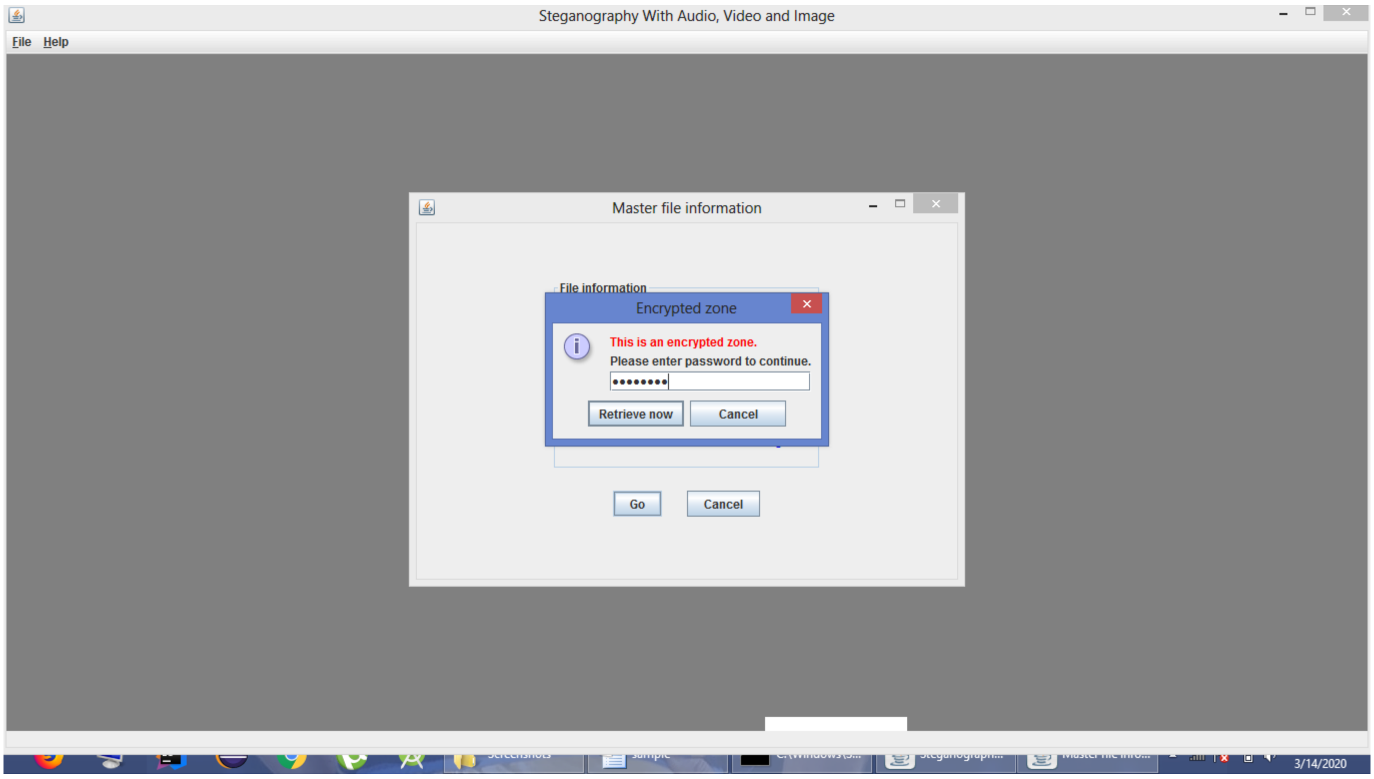
Message:

```
TG password:8H4794$$6jf5
Location:17.2434.43212.000.3
Time:16:00IST
```

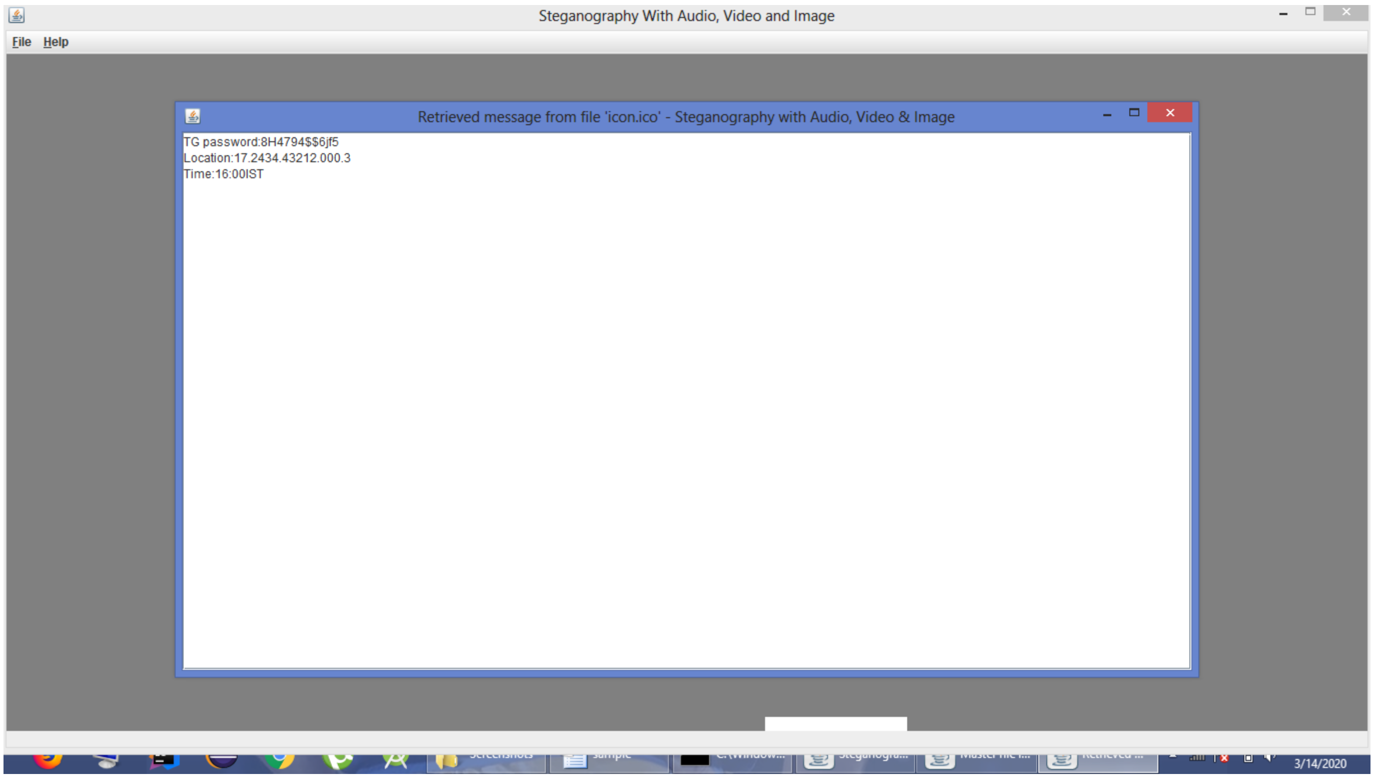
Message embedded successfully in file 'icon.ico'.



Master file	icon.ico
Steganograph Used	2.0.0
Contains	Embedded message
Compressed	NO
Compression ratio	0 %
Encrypted	YES
Requested operation	Retrieve message



This is an encrypted zone.
Please enter password to continue.


Steganography With Audio, Video and Image
File Help
Retrieved message from file 'icon.ico' - Steganography with Audio, Video & Image

```
TG password:8H4794$$6jf5
Location:17.2434.43212.000.3
Time:16:00IST
```

CONCLUSION

Conclusions and Recommendations

This proposed system enables to add, remove node by changing the weight i.e., by moving the cursor. We can add new node by double clicking, all the process can be achieved by following the process mention in the text area.

The system deals with providing an easy and secures files transmissions across the network. The data is embedded with an Image file and transmitted through the network.. It is going to be reliable and the confirmation status of the transfer is returned. It retrieves data from the image file. It deals with Embedding, Retrieving and Authentication of data.

FUTURE ENHANCEMENT

Future Enhancement

Different kind of statistical data can be fetched from the database and the results can be presented in the form of reports. Request for password changes can be automated so that the indication need not be done manually for the given point of time.

The efficiency of any system designed to suit an organization depends cooperation during the implementation stage and also flexibility of the system to adopt itself to the organization. Stegnography has been developed to overcome the problems with the security.

- In future this project offers user to enter the data through simple and interactive forms. This is very helpful for the client to enter the desired information through so much simplicity.

- The user is mainly more concerned about the validity of the data, whatever he is entering. There are checks on every stages of any new creation, data entry or updating so that the user cannot enter the invalid data, which can create problems at later date.

BIBLIOGRAPHY

BIBLIOGRAPHY

References for the Project Development Were Taken From the following Books and Web Sites.

<u>**JAVA Technologies:**</u>

JAVA Complete Reference

Mastering JAVA Security

Head First EJB Sierra Bates

Java Script Programming -Yehuda Shiran

JAVA2 Networking -Pistoria

JAVA Security-Scotl oaks

J2EE Professional -Shadab siddiqui