

# Malicious Email Detection & Filtering System Using Bayesian Machine Learning Algorithm



## Authors

Rao Muhammad Umer	10CS61
2010-IU-967	
Tayyeb Islam	10CS76
2010-IU-972	
Awais Anwar	10CS119
2010-IU-994	
Hussain Ahmed Madni	10CS67
2010-IU-970	

## Supervisor

Engr. Khurram Hameed  
Lecturer

DEPARTMENT OF COMPUTER SYSTEM ENGINEERING  
UNIVERSITY COLLEGE OF ENGINEERING AND TECHNOLOGY

**The Islamia University of Bahawalpur**

2010-2014

# Malicious Email Detection & Filtering System Using Bayesian Machine Learning Algorithm

## Authors

Rao Muhammad Umer	10CS61
2010-IU-967	
Tayyeb Islam	10CS76
2010-IU-972	
Awais Anwar	10CS119
2010-IU-994	
Hussain Ahmed Madni	10CS67
2010-IU-970	

A thesis submitted in partial fulfillment of the requirements for the degree of

B.Sc. Computer System Engineering

## Supervisor

Engr. Khurram Hameed  
Lecturer (UCET-IUB)

External Examiner Signature: \_\_\_\_\_

Thesis Supervisor Signature: \_\_\_\_\_

DEPARTMENT OF COMPUTER SYSTEM ENGINEERING  
UNIVERSITY COLLEGE OF ENGINEERING AND TECHNOLOGY  
**The Islamia University of Bahawalpur**

2010-2014

## ABSTRACT

**Suspicious or Malicious E-mail detection** is a kind of mailing system where suspicious users are identified by determining the keywords used by someone. The keywords such as bomb, RDX and attached harmful files are found in the mails which are sent by the user. All these blocked mails are checked by the administrator and identify the users who sent such mails.

Malicious E-mail is the term used to describe any code in any part of a software system or script that is intended to cause undesired effects, security breaches or damage to a system. Malicious code describes a broad category of system security terms that includes attack scripts, viruses, worms, Trojan horses, backdoors, and malicious active content.

## UNDERTAKING

I certify that research work titled “**Malicious Email Detection & Filtering System Using Bayesian Machine Learning Algorithm**” is my own work. The work has not been presented elsewhere for assessment. Where material has been used from other sources it has been properly acknowledged/referred.

Signature of Students

Rao Muhammad Umer 10CS61 .....  
2010-IU-967

Tayyeb Islam 10CS76 .....  
2010-IU-972

Awais Anwar 10CS119 .....  
2010-IU-994

Hussain Ahmed Madni 10CS67 .....  
2010-IU-970

## ACKNOWLEDGEMENTS

*We are grateful to Allah Almighty who provided us with this wonderful concept and bestowed us the ability to complete it successfully. We are thankful to our parents for their constant support and their financial help as the project wouldn't have been successful without their motivation and prayers.*

*We are grateful to our supervisor **Engr. Khurram Hameed** who helped us out of many hitches and provided us guidance at many instances and motivated us to carry on our work which obviously resulted in successful completion of our project. And finally, we thank all the faculty members of this great institution that helped us over the course of our studies. This project wouldn't have been made without their support.*

## Table of Contents

<b><i>Undertaking</i></b> .....	<b>2</b>
<b><i>Acknowledgements</i></b> .....	<b>3</b>
<b><i>List of figures</i></b> .....	<b>9</b>
<b><i>Chapter 1</i></b> .....	<b>12</b>
1.1 Statement of the Problem .....	12
1.2 Objectives .....	13
1.3 Block Diagram .....	14
<b><i>Chapter 2</i></b> .....	<b>15</b>
2.1 Login Module .....	15
2.2 Registration Module .....	16
2.3 Administration Module .....	16
2.4 Encryption Module .....	17
2.5 User Module .....	18
<b><i>Chapter 3</i></b> .....	<b>19</b>
3.1 N-Tier Architecture .....	19
3.1.1 2-Tier and 3-Tier Architecture .....	19
3.2 MVC Architecture .....	21
3.2.1 Model-1 .....	21
3.2.2 Model-2(MVC) .....	21
3.2.3 Advantages of MVC .....	23
<b><i>Chapter 4</i></b> .....	<b>24</b>
4.1 Agile software Development .....	24
4.1.1 Manifesto for Agile Software Development .....	24
4.1.2 Comparison between Water Fall and Agile Software Process Model.....	24
4.2 Scrum.....	25
4.2.1 User Story .....	26
4.3 Test Driven Development (TDD) .....	28
4.3.1 Software for TDD .....	28
4.3.2 The Three Laws of TDD .....	28
<b><i>Chapter 5</i></b> .....	<b>30</b>
5.1 Bottom-Up Approach.....	30
5.2 Top-Down Approach.....	30
5.3 UML Modeling .....	31
5.3.1 UNIFIED MODELING LANGUAGE .....	31
5.3.1.1 USER MODEL VIEW .....	31
5.3.1.2 STRUCTURAL MODEL VIEW .....	31
5.3.1.3 BEHAVIORAL MODEL VIEW .....	31
5.3.1.4 IMPLEMENTATION MODEL VIEW .....	31

5.3.1.5 ENVIRONMENTAL MODEL VIEW .....	31
5.3.2 Data Flow Diagrams .....	32
5.3.2.1 ADMIN .....	32
5.3.2.2 USER.....	32
5.3.3 Class Diagram.....	33
5.3.3.1 ADMIN .....	33
5.3.3.2USER.....	33
5.3.4 Sequence Diagram.....	34
5.3.4.1 ADMIN .....	34
5.3.4.2 USER.....	34
5.3.5 Collaboration Diagram .....	35
5.3.5.1 ADMIN .....	35
5.3.5.2 USER.....	35
5.3.6 Object Diagram.....	36
5.3.6.1 ADMIN .....	36
5.3.6.2 <i>USER</i> .....	36
5.3.7 Use Case Diagram.....	37
5.3.7.1 ADMIN .....	37
5.3.7.2 USER.....	37
5.3.8 Component Diagram.....	38
5.3.8.1 ADMIN .....	38
5.3.8.2 USER.....	38
5.3.9 Deployment Diagram.....	39
5.3.9.1 ADMIN .....	39
5.3.9.2 USER.....	39
DEVELOPMENT LANGUAGES .....	40
6.1 Java.....	40
6.1.1 Java Programming Language.....	40
6.1.2Java Platform.....	41
6.1.3What can Java Technology do?.....	42
6.1.4 Java as an Emerging Technology .....	43
6.2 SERVLET .....	44
6.2.1 Introduction to Servlets .....	44
6.2.2 Servlet Architecture .....	45
6.2.3Servlet Lifecycle .....	47
6.2.4Writing the Servlet .....	48
6.2.5 Interacting with Clients .....	49
6.2.6 Lifecycle Methods.....	52
Providing Information about the Servlet.....	57

6.3 JSP .....	58
6.3.1 JSP LIFE CYCLE .....	59
6.4 JSP - Expression Language (EL) .....	60
6.4.1 Simple Syntax .....	60
6.5 JSTL .....	61
6.6 JavaBeans:.....	62
6.6.1JavaBeans Conventions .....	63
6.7 POJO (Plain Old Java Object) .....	65
6.8 HTML .....	66
HTML Example .....	66
Example Explained .....	66
6.8.1 HTML Tags .....	67
6.8.2 HTML Page Structure.....	67
6.8.3 HTML Versions.....	67
6.9 CSS.....	68
Styles Solved a Big Problem .....	68
6.9.1 CSS Saves a Lot of Work! .....	69
6.9.2 CSS Syntax.....	69
CSS Example.....	69
Example .....	69
6.9.4 CSS Comments .....	70
Example .....	70
<b>Chapter 7 .....</b>	<b>71</b>
7.1 Database.....	71
7.2 Relational DBMS .....	72
7.2.1Integrity Rules .....	72
Employees Table 7.2.....	73
7.2.2 SELECT Statements .....	74
Table 7.3.....	74
7.2.3WHERE Clauses.....	75
7.2.4 Joins .....	76
Cars Table 7.4 .....	76
7.2.5 Common SQL Commands .....	77
7.2.6 Result Sets and Cursors .....	78
7.2.7 Transactions.....	79
7.2.8 Stored Procedures.....	80
7.3 SQL – Structured Query Language .....	81
7.4 JDBC Introduction.....	81
7.4.1 JDBC Product Components .....	83



7.4.2 JDBC Architecture .....	84
<b>Chapter 8 .....</b>	<b>86</b>
8.1 Bayes Theorem.....	86
8.1.1 Anti-Spam Filter .....	86
8.1.2 Advantages:.....	89
8.1.3 Disadvantages: .....	89
8.2 Unicode Encryption Algorithm .....	90
8.2.1 Code .....	90
<b>Chapter 9 .....</b>	<b>92</b>
9.1 TESTING IN STRATEGIES.....	92
9.1.1 UNIT TESTING .....	92
9.1.1.1 BLACK BOX TESTING.....	92
9.1.1.2 WHITE BOX TESTING.....	93
9.1.2 INTEGRATING TESTING.....	93
9.1.3 SYSTEM TESTING .....	93
9.1.4 ACCEPTANCE TESTING.....	93
9.2 TEST APPROACH .....	93
9.2.1 BOTTOM UP APPROACH .....	93
9.2.2 TOP DOWN APPROACH .....	94
9.3 VALIDATION .....	94
<b>Chapter 10 .....</b>	<b>95</b>
10.1 Hardware Requirements .....	95
10.2 Software Requirements.....	95
<b>Chapter 11 .....</b>	<b>96</b>
11.1 Eclipse JEE: .....	96
11.1.1 Package Description: .....	96
11.1.2 Package includes: .....	96
11.2 Java Development Kit: .....	96
11.3 MySQL Workbench:.....	97
11.3.1 Design: .....	97
11.3.2 Develop:.....	97
11.3.3 Administer: .....	97
11.3.4 New! Visual Performance Dashboard:.....	97
11.3.5 Database Migration:.....	98
11.4 Apache Tomcat: .....	98
11.4.2 Directories and Files:.....	98
11.5 Adobe Dreamweaver: .....	99
<b>Chapter 12 .....</b>	<b>100</b>
12.1 Hello World Application .....	100

<b>Chapter 13</b> .....	<b>110</b>
13.1 Conclusion .....	110
13.2 Future Enhancements .....	110
Abbreviations .....	111

## LIST OF FIGURES

<i>Number</i>	<i>Page</i>
Fig 1.1 Block diagram of Project .....	16
Fig 2.1 Login Module .....	17
Fig 2.2 Registration Module .....	18
Fig 2.3 Administration Module.....	19
Fig 2.4 Encryption Module. ....	19
Fig 2.5 User Module .....	20
Fig 3.1 2-Tier Architecture .....	21
Fig 3.2 3-Tier Architecture .....	22
Fig 3.3 Model-1(MVC).....	23
Fig 3.4 Model-2(MVC).....	25
Fig 4.1 Software Development Process Dependencies.....	27
Fig 4.2 Scrum.....	28
Fig 4.3 Test Driven Development Cycle .....	31
Fig 5.1 Data Flow Diagram (Admin) .....	35
Fig 5.2 Data Flow Diagram (User) .....	35
Fig 5.3 Class Diagram (Admin) .....	36
Fig 5.4 Class Diagram (User) .....	36
Fig 5.5 Sequence Diagram (Admin) .....	37

Fig 5.6Sequence Diagram (User).....	37
Fig5.7Collaboration Diagram (Admin) .....	38
Fig 5.8Collaboration Diagram (User) .....	38
Fig 5.9Object Diagram (Admin) .....	39
Fig 5.10Object Diagram (User) .....	39
Fig 5.11 Use Case Diagram (Admin) .....	40
Fig5.12Use Case Diagram (User) .....	40
Fig 5.13Component Diagram (Admin) .....	41
Fig 5.14Component Diagram (User) .....	41
Fig 5.15Development Diagram (Admin) .....	42
Fig 5.16Development Diagram (User) .....	42
Fig.6.1Overview of Software Development Process .....	44
Fig 6.2Java Virtual Machine (JVM) .....	44
Fig.6.3Java Platform .....	45
Fig 6.4JSP Life Cycle .....	53
Fig 6.5JSTL Tag Library Group .....	56
Fig7.1JDBC Architecture .....	78
Fig 7.2JDBC 3-Tier Architecture .....	79
Fig 8.1Probablity Graph.....	81
Fig 12.1Switch Eclipse to JEE.....	94
Fig 12.2Open Perspective .....	95
Fig12.3Select Dynamic Web Project.....	96
Fig 12.4Name to the Project .....	97

Fig 12.5Set Default Values .....	98
Fig 12.6Adding JSP .....	99
Fig 12.7Give Name to JSP .....	100
Fig12.8Run and Deploy the Project.....	102
Fig12.9Output of the Project.....	103

# CHAPTER 1

## INTRODUCTION

### 1.1 Statement of the Problem

Several articles, industry reports and congressional testimonies document the existence of targeted malicious email (TME) sent by malicious threat actors not necessarily motivated by profit alone. These malicious emails have been targeted at company executives, government personnel and other individuals with access to sensitive information useful by an opposing party to advance a cause. Current research and commercial methods for detecting illegitimate email are limited to addressing Internet scale email abuse such as spam, none seek to address targeted malicious emails.

For organizations targeted by these emails, detection is critically important since these emails can enable the installation of malicious software on the targeted user's computer system. This malicious software can contain a backdoor that allows a malicious threat actor to gain entrance to an organization's network and its sensitive information. Whereas conventional unwanted email, such as spam, is sent in bulk to a large number of people on the Internet, TME is sent to very specific individuals.

The techniques that malicious threat actors use to craft and send these targeted emails are different from the techniques used by spammers. Furthermore, since the targeted emails are sent to specific individuals, the characteristics of the recipient are relevant whereas with spam, they are less relevant. This dissertation exploits the differences between spam and TME by capturing features of TME and TME recipients and incorporating them into a decision classifier. The classifier is an algorithm that categorizes a given email as either TME or non-targeted malicious email (NTME).

This categorization allows an organization to decide whether to accept or reject the email coming into their network environment.

All organizations allow email to enter in their network some of the attackers target single user or small group and extract important information by injecting malicious code

in the email as well as in email attachment that creates backdoor in system. If we rely on current conventional detection methods, targeted email attack goes undetected and file attachment have malicious code that is also create trouble in network. A malicious executable is defined to be a program that performs a malicious function, such as compromising a system's security, damaging a system or obtaining sensitive information without the user's permission. Using data mining methods. Every day some malicious programs are created and most cannot be accurately detected until signatures have been generated for them.

## 1.2 Objectives

**Suspicious or Malicious E-mail detection** is a kind of mailing system where suspicious users are identified by determining the keywords used by him/her. The keywords such as bomb, RDX, are found in the mails which are sent by the user. All these blocked mails are checked by the administrator and identify the users who sent such mails.

Malicious E-mail is the term used to describe any code in any part of a software system or script that is intended to cause undesired effects, security breaches or damage to a system. Malicious code describes a broad category of system security terms that includes attack scripts, viruses, worms, Trojan horses, backdoors, and malicious active content.

### 1.3 Block Diagram

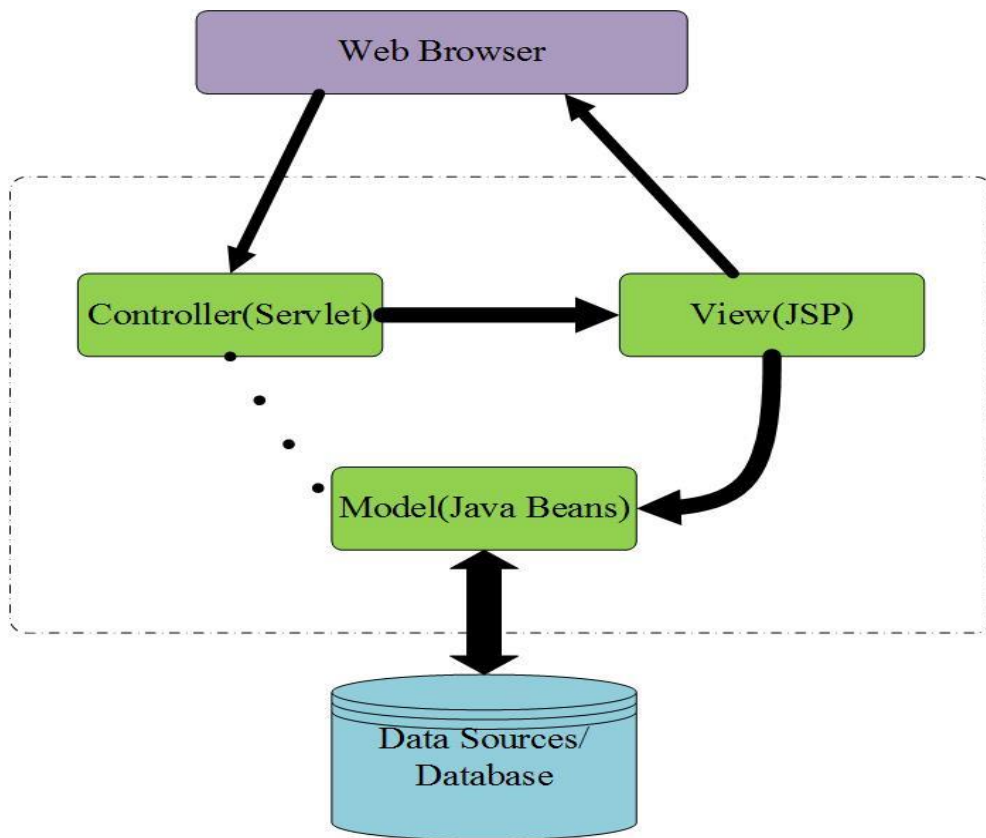


Figure 1.1



# CHAPTER 2

## PROJECT OVERVIEW

### 2.1 Login Module

This module is used by administrator and users (who are authenticated) to login into the Colors mail. The login details of the specified person will be entered and hence can enter into the Colors mail.

Login

Logging into secure mail

Email\*

Password\*

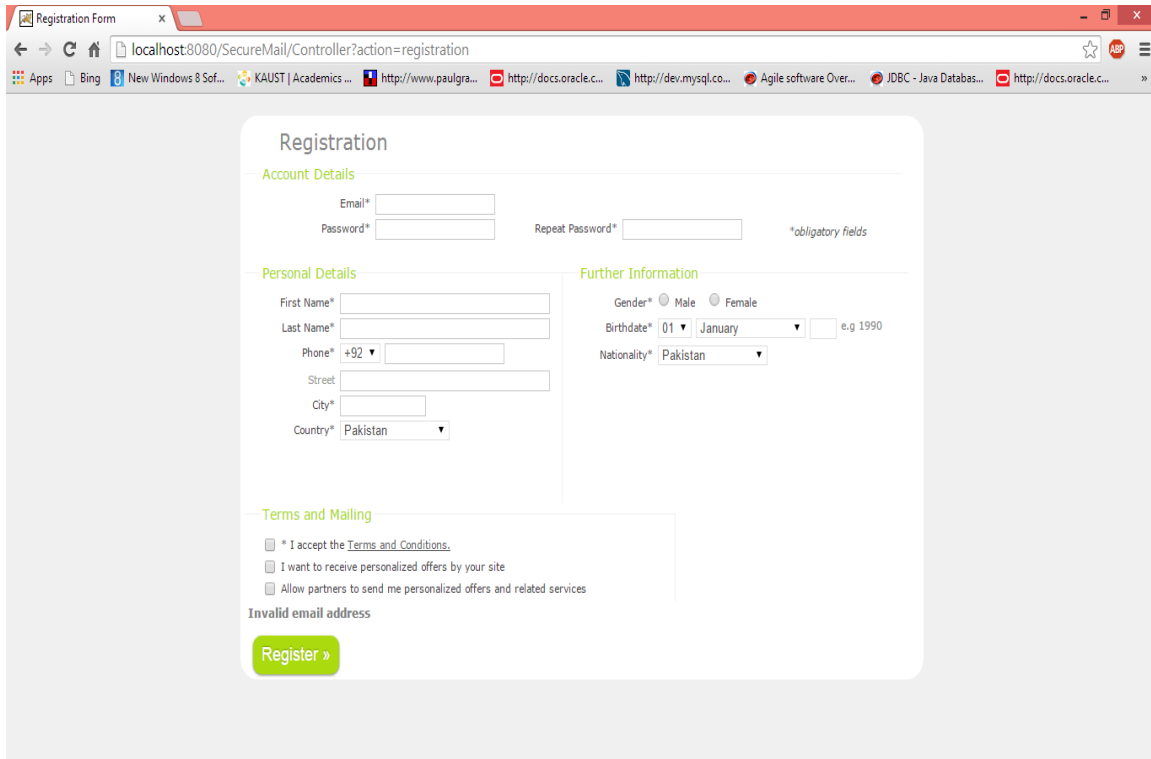
[New User? Register](#)

Login »

Figure 2.1

## 2.2 Registration Module

This module is used by the unauthenticated users who are unregistered. The users must register themselves such that they can login into the Colors mail.



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/SecureMail/Controller?action=registration'. The page title is 'Registration Form'. The main content area is a registration form titled 'Registration'. The form is divided into three sections: 'Account Details', 'Personal Details', and 'Further Information'. The 'Account Details' section includes fields for 'Email\*', 'Password\*', and 'Repeat Password\*', with a note '\*obligatory fields'. The 'Personal Details' section includes fields for 'First Name\*', 'Last Name\*', 'Phone\*' (with a dropdown for '+92'), 'Street', 'City\*', and 'Country\*' (with a dropdown for 'Pakistan'). The 'Further Information' section includes fields for 'Gender\*' (with radio buttons for 'Male' and 'Female'), 'Birthdate\*' (with a dropdown for '01' and a dropdown for 'January', and a text field for 'e.g 1990'), and 'Nationality\*' (with a dropdown for 'Pakistan'). Below these sections is a 'Terms and Mailing' section with three checkboxes: '\* I accept the Terms and Conditions.', 'I want to receive personalized offers by your site', and 'Allow partners to send me personalized offers and related services'. At the bottom of the form is a green 'Register »' button. The browser's taskbar shows several open applications, including 'Apps', 'Bing', 'New Windows 8 Sof...', 'KAUST | Academics ...', 'http://www.paulgra...', 'http://docs.oracle.c...', 'http://dev.mysql.co...', 'Agile software Over...', 'JDBC - Java Databas...', and 'http://docs.oracle.c...'.

Figure 2.2

## 2.3 Administration Module

This module is used by the administrator to perform the functions like managing the keywords, entering new keywords and to check out the block list of the discarded mails.

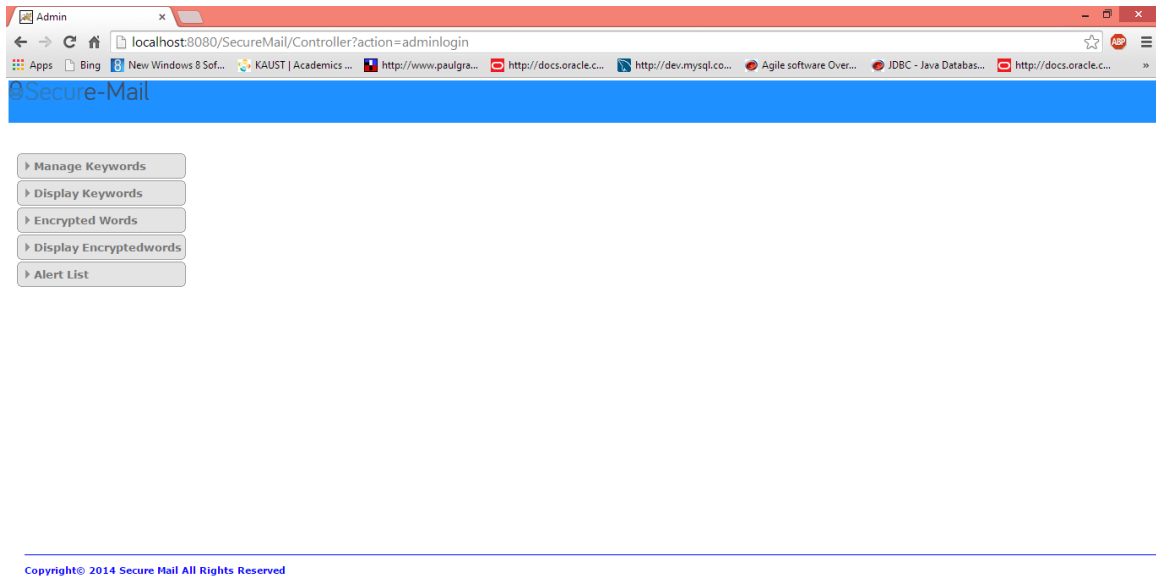


Figure 2.3

## 2.4 Encryption Module

This module is used by the administrator to perform the functions like encryption of the words. The encrypted words are sent to the database and hence managed by the administrator.

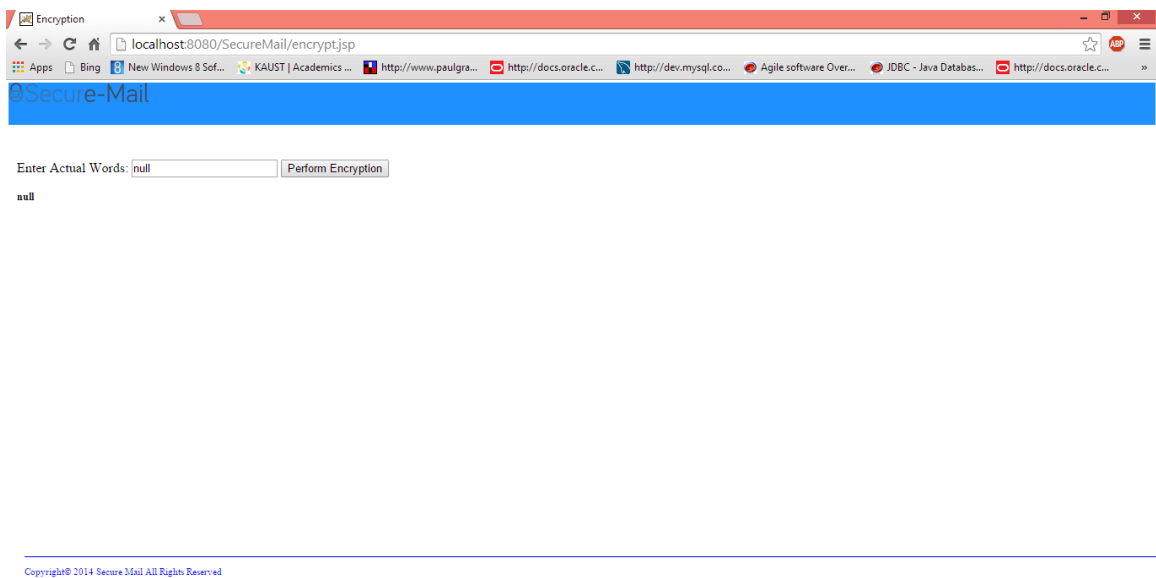


Figure 2.4

## 2.5 User Module

This module is used by the users to do operations like composing mail, checking out the mails in inbox and finally sending the mails to the authenticated users by attaching a message.

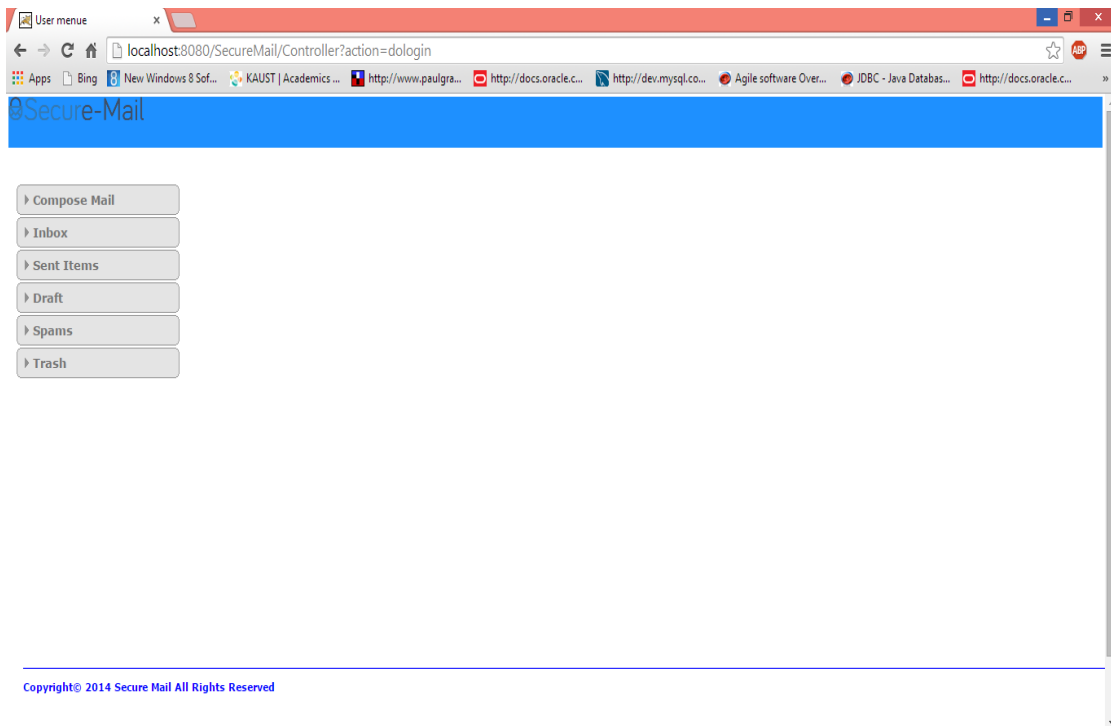


Figure 2.5

## CHAPTER 3

### System Architecture

#### 3.1 N-Tier Architecture

##### 3.1.1 2-Tier and 3-Tier Architecture

There are two types of client server architectures:

- **2-tier architectures:**

In this architecture, client directly interact with the server. This type of architecture may have some security holes and performance problems. Internet Explorer and Web Server work on two tier architecture. Here security problems are resolved using Secure Socket Layer (SSL).

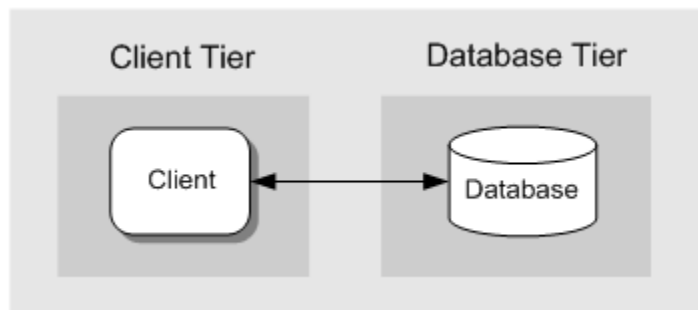


Figure 3.1

- **3-tier architectures:**

In this architecture, one more software sits in between client and server. This middle software is called middleware. Middleware are used to perform all the security checks and load balancing in case of heavy load. A middleware takes all requests from the client and after doing required authentication it passes that request to the server. Then server does required processing and sends response

back to the middleware and finally middleware passes this response back to the client. If you want to implement a 3-tier architecture then you can keep any middle ware like Web Logic or Web Sphere software in between your Web Server and Web Browsers.

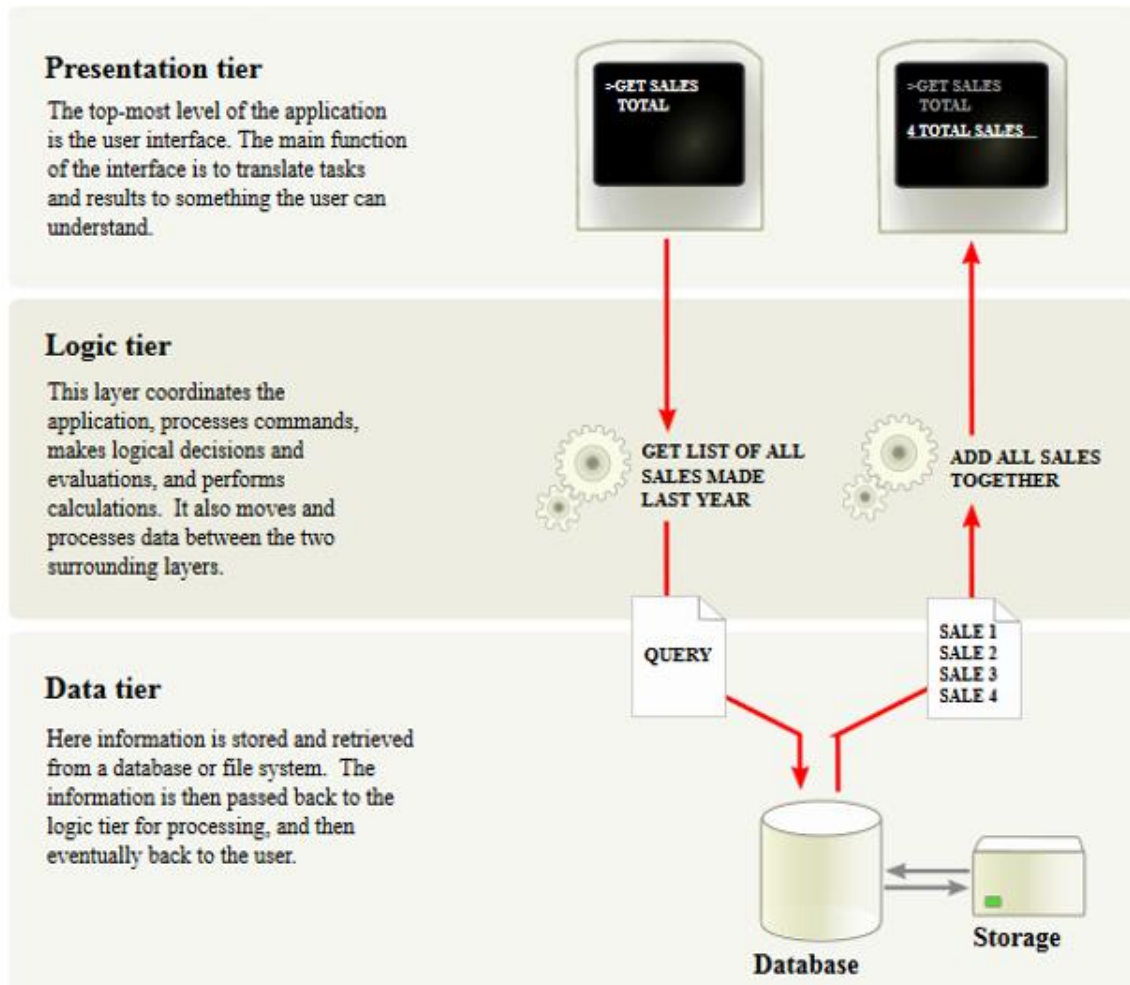


Figure 3.2

### Advantages of Tiers

- Layering helps you to maximize maintainability of the code, optimize the way that the application works when deployed in different ways, and provide a clear delineation between locations where certain technology or design decisions must be made.

- Placing your layers on separate physical tiers can help performance by distributing the load across multiple servers. It can also help with security by segregating more sensitive components and layers onto different networks or on the Internet versus an intranet.

## 3.2 MVC Architecture

### 3.2.1 Model-1

In Model 1, a request is made to a JSP or servlet and then that JSP or servlet handles all responsibilities for the request, including processing the request, validating data, handling the business logic, and generating a response. The Model 1 architecture is commonly used in smaller, simple task applications due to its ease of development. Also, the Model 1 architecture unnecessarily ties together the business logic and presentation logic of the application.

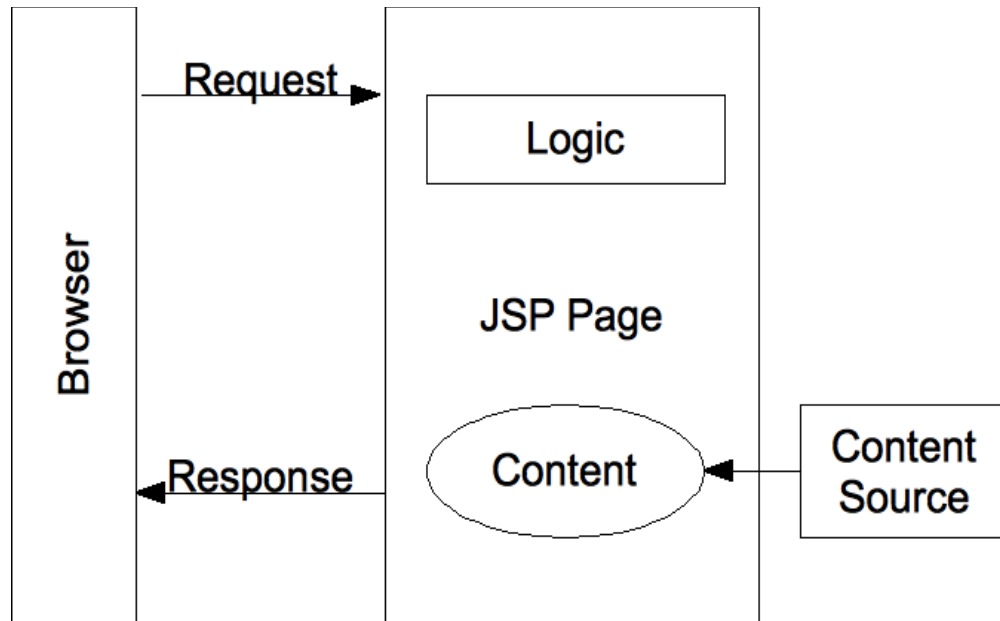


Figure 3.3

### 3.2.2 Model-2(MVC)

MVC stands for Model View Controller. Models are nothing but POJO (Plain Old Java Object). Views are any view technology like JSP, HTML, Velocity etc. and controller contains actual business logic.

MVC is a design pattern methodology which used to separate data away from view and business logic, as instructor said Model (data) never import from view or controller packages.

Model-2 is a complex design pattern used in the design of Java Web applications which separates the display of content from the logic used to obtain and

manipulate the content. Since Model 2 drives a separation between logic and display, it is usually associated with the model–view–controller (MVC) paradigm. While the exact form of the MVC "Model" was never specified by the Model 2 design, a number of publications recommend a formalized layer to contain MVC Model code. The Java Blue Prints, for example, originally recommended using EJBs to encapsulate the MVC Model.

In a Model 2 application, requests from the client browser are passed to the controller. The controller performs any logic necessary to obtain the correct content for display. It then places the content in the request (commonly in the form of a **JavaBean** or **POJO**) and decides which view it will pass the request to. The view then renders the content passed by the controller. Model 2 is recommended for medium- and large-sized applications.

MVC usually is model for single application, where all 3 parts are more or less connected between each other, but 3-tier architecture means, that it's something like 3 modules/different projects. And every of them will do its own part and will have own architecture, probably MVC.

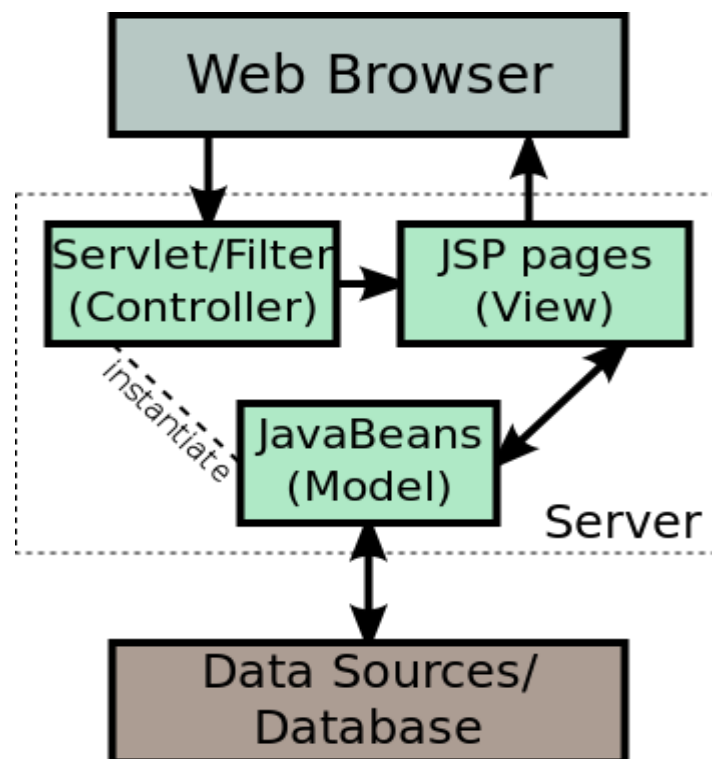


Figure 3.4



### 3.2.3 Advantages of MVC

MVC is a design pattern to develop web applications. Model, View and Controller are the three components to separate logics to reuse entire the applications. Since there are multiple layers representing multiple logics there will be clean separation of roles or logics.

- Modification of one logic does not affect other logics.
  - Easy to maintain and enhance the project.
  - Parallel development is possible due to this productivity is very good.
  - Project Leader divides a team into two parts.
    1. Web Authors
    2. J2EE Developers
1. **Web Authors:** These programmers use JSP in developer's presentation logic of the application.
  2. **J2EE Developers:** These programmers use J2EE technologies like servlets EJB and etc., to develop integration logic and business logic, persistence logic of the application.

## CHAPTER 4

# SOFTWARE PROCESS MODEL

### 4.1 Agile software Development

The Agile movement proposes alternatives to traditional project management. Agile approaches are typically used in software development to help businesses respond to unpredictability.

#### 4.1.1 Manifesto for Agile Software Development

- Individuals and Interactions over processes and tools
- Working Software over comprehensive documentation
- Customer Collaboration over contract negotiation
- Responding to Change over following a plan

#### 4.1.2 Comparison between Water Fall and Agile Software Process Model

Agile development provides opportunities to assess the direction throughout the development lifecycle. This is achieved through regular cadences of work, known as Sprints or iterations, at the end of which teams must present a potentially shippable product increment. By focusing on the repetition of abbreviated work cycles as well as the functional product they yield, agile methodology is described as “iterative” and “incremental.” In waterfall, development teams only have one chance to get each aspect of a project right. In an agile paradigm, every aspect of development — requirements, design, etc. — is continually revisited. When a team stops and re-evaluates the direction of a project every two weeks, there’s time to steer it in another direction.

## Software Development Process Dependencies

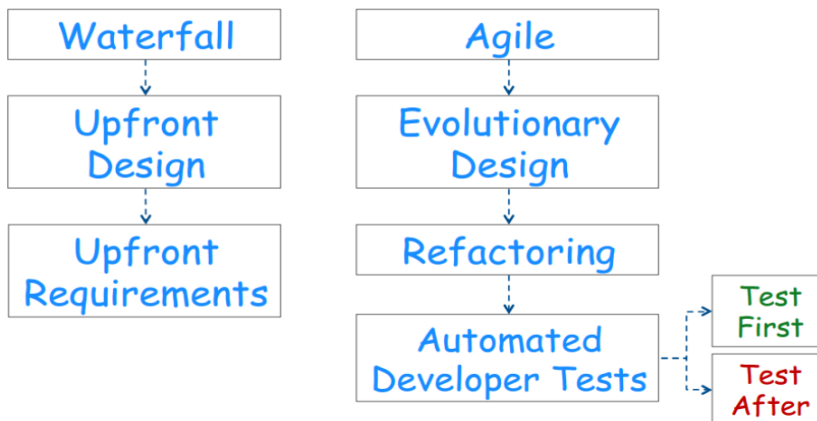


Figure 4.1

### 4.2 Scrum

Scrum emphasizes the idea of “empirical process control.” That is, Scrum uses the real-world progress of a project — not a best guess or uninformed forecast — to plan and schedule releases. In Scrum, projects are divided into succinct work cadences, known as sprints, which are typically one week, two weeks, or three weeks in duration. At the end of each sprint, stakeholders and team members meet to assess the progress of a project and plan its next steps. This allows a project’s direction to be adjusted or reoriented based on completed work, not speculation or predictions.

This emphasis on an ongoing assessment of completed work is largely responsible for its popularity with managers and developers. But what allows Scrum to work is a simple set of roles, responsibilities, and meetings that never change. If Scrum’s capacity for adaption and flexibility makes it an appealing option, the stability of its practices give teams something to lean on when development gets chaotic.

# SCRUM

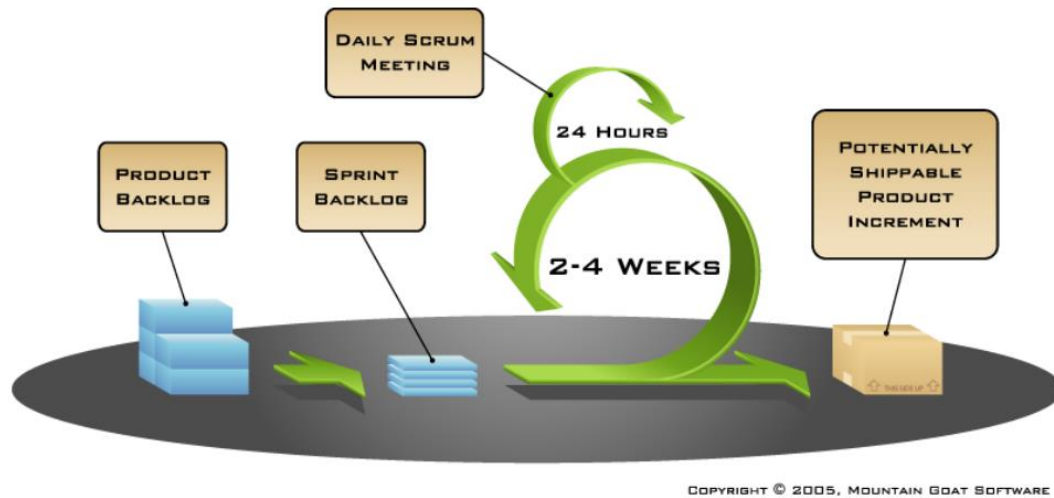


Figure 4.2

## 4.2.1 User Story

User stories are part of an agile approach that helps shift the focus from writing about requirements to talking about them. All agile user stories include a written sentence or two and, more importantly, a series of conversations about the desired functionality.

### User Story Format

- **Description:**

As a <role>

I want <feature>

So that <reason/value>

- **Acceptance Criteria:**

Given <Pre-condition>

When <User action>

Then <Expected Result>

- **Definition of Done**

- Passes all regression tests

- Passed testing per acceptance criteria items
- Peer code review
- Accepted by UI team
- Demo to business stakeholder

### **User Story Example:**

#### **User Story Title: Web Developer Managing the Admin Account**

**As a** web developer

**I want to** create administration account

**So that** I can manage keywords, adding new keywords and to block the harmful contents of email

#### **Acceptance Criteria:**

##### **Preconditions:**

- There will be connectivity of DB Server to admin account.
- An authorized developer can enter administration account.

##### **User Actions:**

- A developer can add the keywords (like bomb, RDX etc.) in the Alert List and block them and also identify such user who sent such emails.
- A developer can block harmful emails files like .bat, .exe etc. extension and identify user who sent such mails.
- A developer can encrypt the password, keywords and other file extensions.

##### **Expected Results:**

- Suspicious keywords and malicious files should be blocked.
- Malicious emails sender should be identified.
- Password, keywords and other file extension should be encrypted.

## **4.2.2 DAILY STANDUP**

- It lasts no more than 15 minutes.
- What you did yesterday?
- What are you planning to do today?
- Any roadblocks?

### **4.3 Test Driven Development (TDD)**

**Test-driven development (TDD)** is a software development process that relies on the repetition of a very short development cycle: first the developer writes an (initially failing) automated test case that defines a desired improvement or new function, then produces the minimum amount of code to pass that test, and finally refactors the new code to acceptable standards.

#### **4.3.1 Software for TDD**

##### **XUnit frameworks:**

Developers may use computer-assisted testing frameworks, such as XUnit created in 1998, to create and automatically run the test cases. Xunit frameworks provide assertion-style test validation capabilities and result reporting. These capabilities are critical for automation as they move the burden of execution validation from an independent post-processing activity to one that is included in the test execution. The execution framework provided by these test frameworks allows for the automatic execution of all system test cases or various subsets along with other features.

#### **4.3.2 The Three Laws of TDD**

1. You are not allowed to write any production code unless it is to make a failing unit test pass.
2. You are not allowed to write any more of a unit test than is sufficient to fail and compilation failures are failures.
3. You are not allowed to write any more production code than is sufficient to pass the one failing unit test.

### 4.3.2 TDD Cycle (Red-Green-Refactor)

## Red-Green-Refactor

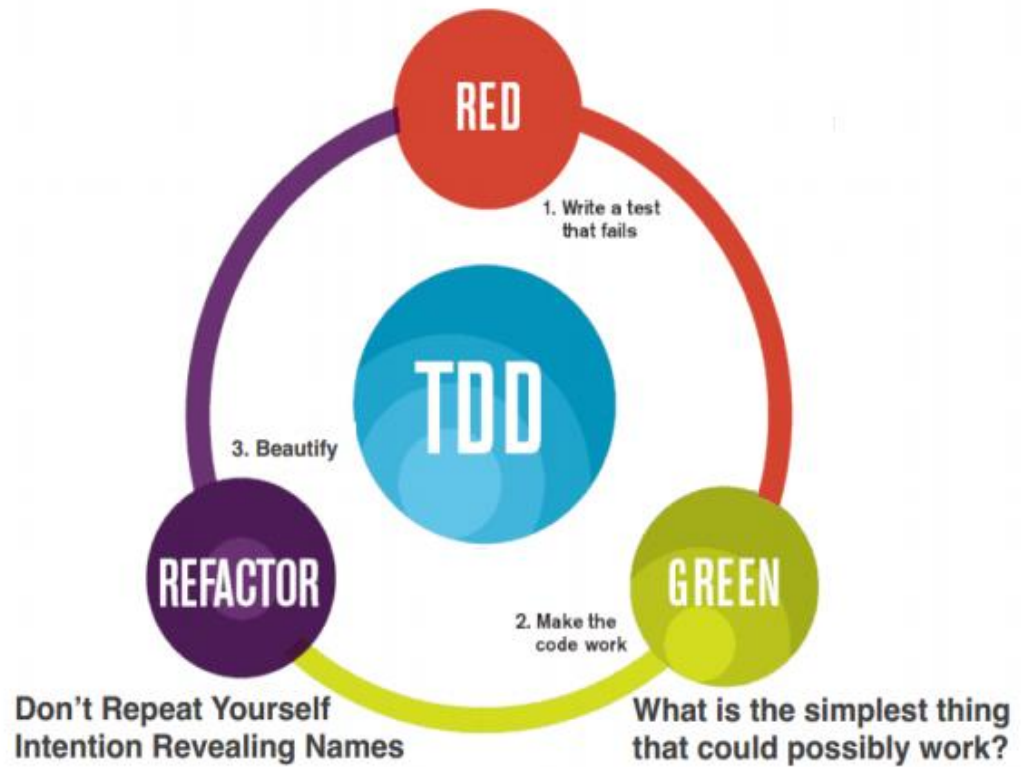


Figure 4.3

# CHAPTER 5

## SYSTEM DESIGN

### 5.1 Bottom-Up Approach

It consists of defining and coding the very basic, definite parts of the system to be designed, then linking these parts together to form the whole.

#### Why Bottom-Up?

- This approach allows teams to code functioning sub-systems quickly
- Testing can be done early and often, as first-level systems are defined first
- It encourages and leads to reusable code
- Pre-existing code is simpler to incorporate and test

### 5.2 Top-Down Approach

The top-down approach, also known as step-wise design, essentially breaks a system or model down into component sub-systems, each of which may further broken down further. However, no first-level system are defined – you won't describe for a loop, or define the attributes of an object in the top-down approach. Systems end up as a series of 'black boxes'; components that have specific inputs and outputs, but no definite internal structure.

#### Why Top-Down?

- This approach lends itself to most projects, and doesn't require a specific kind of team or breakdown
- It allows large teams to split themselves amongst subsystems
- Due to definite inputs and outputs of each subsystem, combining them is simplified
- It encourages object-oriented programming (OOP) via encapsulation
- It is still very successful for functional programming
- It works very well for small projects, by laying out specific 'tasks' to be done.



- Easy to maintain, because of this subsystem partitioning

## **5.3 UML Modeling**

### **5.3.1 UNIFIED MODELING LANGUAGE**

The Unified Modeling Language allows the software engineer to express an analysis model using the modeling notation that is governed by a set of syntactic semantic and pragmatic rules. A UML system is represented using five different views that describe the system from distinctly different perspective. Each view is defined by a set of diagram, which is as follows.

#### **5.3.1.1 USER MODEL VIEW**

- This view represents the system from the user's perspective.
- The analysis representation describes a usage scenario from the end-users perspective.

#### **5.3.1.2 STRUCTURAL MODEL VIEW**

- In this model the data and functionality are arrived from inside the system.
- This model view models the static structures.

#### **5.3.1.3 BEHAVIORAL MODEL VIEW**

It represents the dynamic behavioral as parts of the system, depicting the interactions of collection between various structural elements described in the user model and structural model view.

#### **5.3.1.4 IMPLEMENTATION MODEL VIEW**

In this the structural and behavioral as parts of the system are represented as they are to be built.

#### **5.3.1.5 ENVIRONMENTAL MODEL VIEW**

In this the structural and behavioral aspects of the environment in which the system is to be implemented are represented.

UML is specifically constructed through two different domains they are:

- UML Analysis modeling, this focuses on the user model and structural model views of the system.
- UML design modeling, which focuses on the behavioral modeling, implementation modeling and environmental model views.

Use case Diagrams represent the functionality of the system from a user's point of view. Use cases are used during requirements elicitation and analysis to represent the functionality of the system. Use cases focus on the behavior of the system from external point of view.

### 5.3.2 Data Flow Diagrams

#### 5.3.2.1 ADMIN

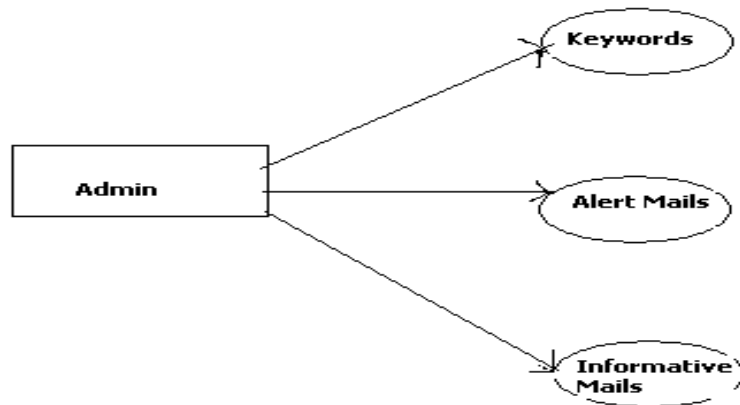


Figure 5.1

#### 5.3.2.2 USER

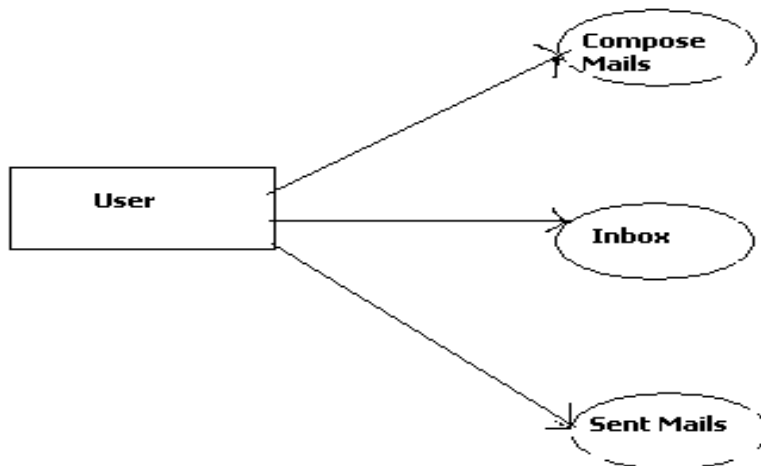


Figure 5.2

### 5.3.3 Class Diagram

#### 5.3.3.1 ADMIN

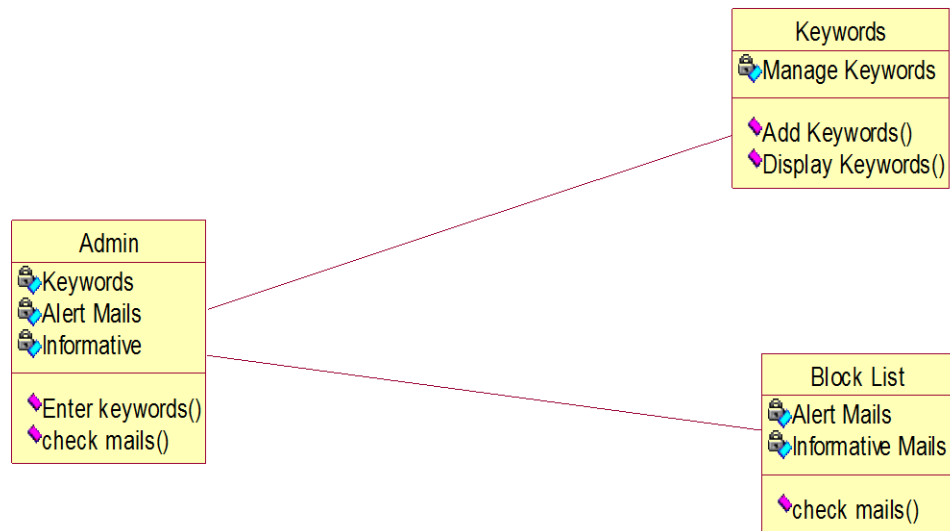


Figure 5.3

#### 5.3.3.2 USER

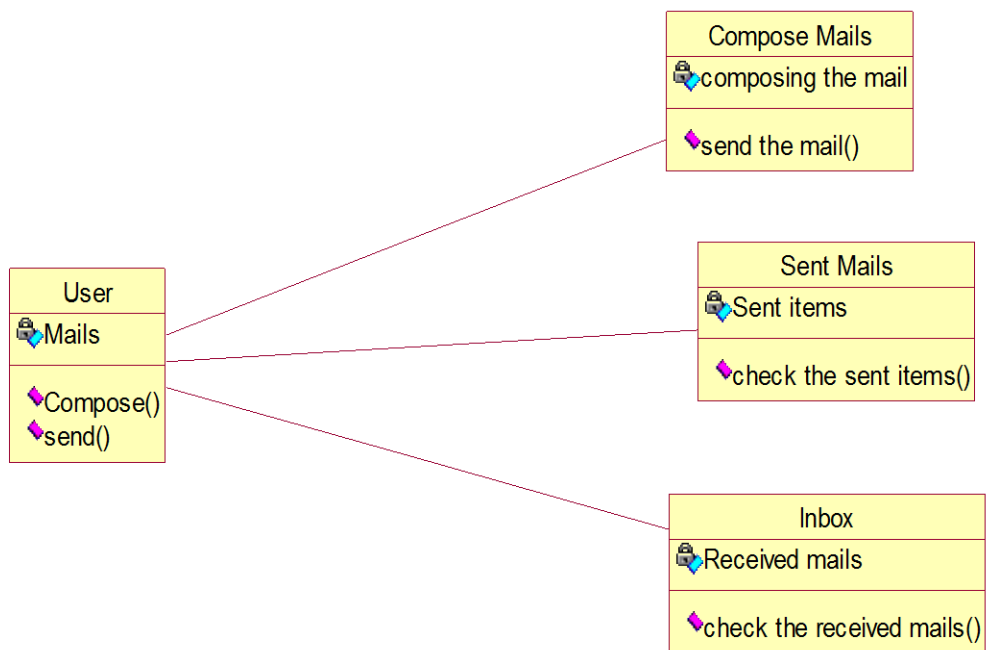


Figure 5.4

### 5.3.4 Sequence Diagram

#### 5.3.4.1 ADMIN

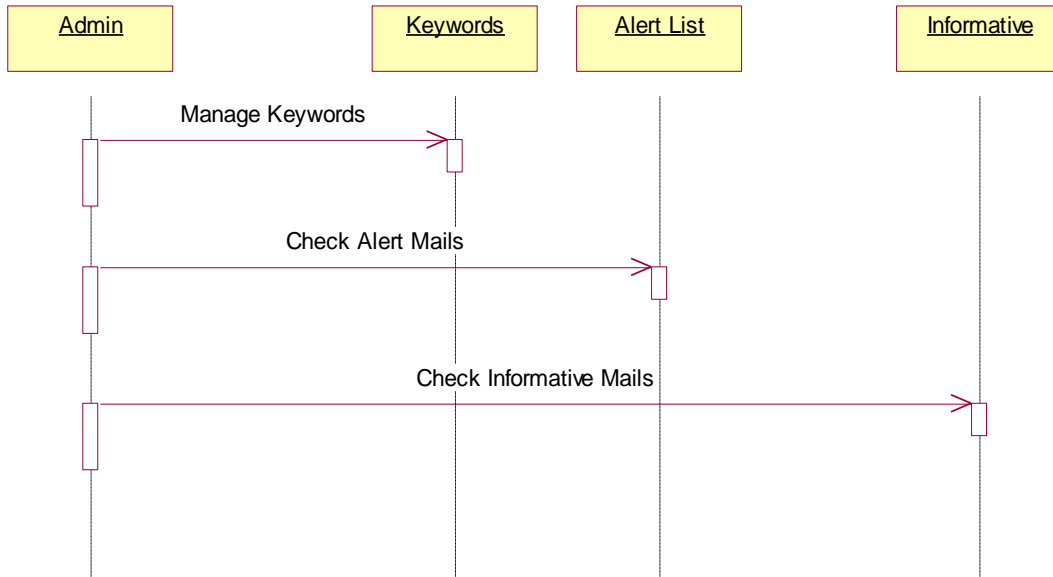


Figure 5.5

#### 5.3.4.2 USER

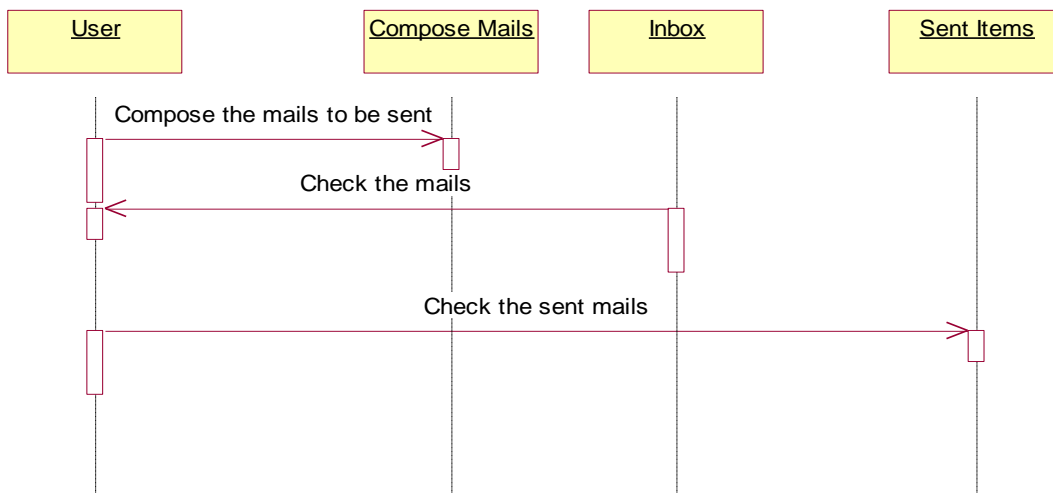


Figure 5.6

5.3.5 Collaboration Diagram

5.3.5.1 ADMIN

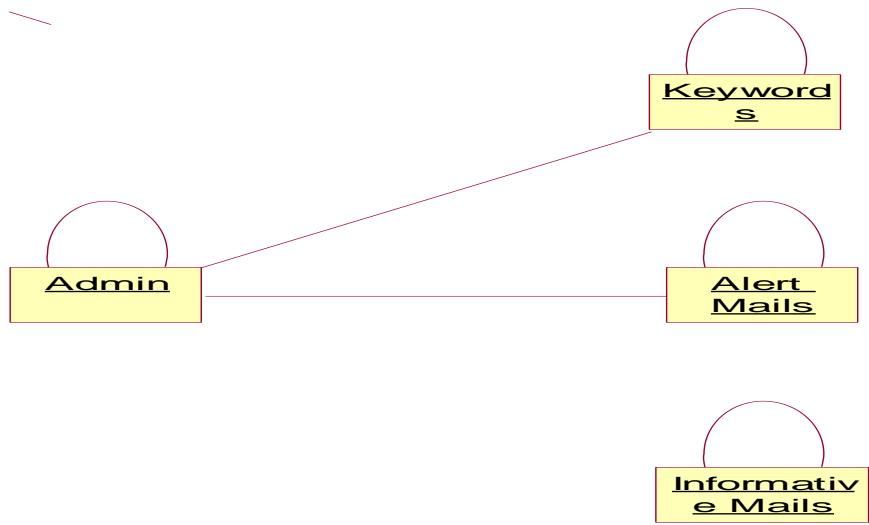


Figure 5.7

5.3.5.2 USER

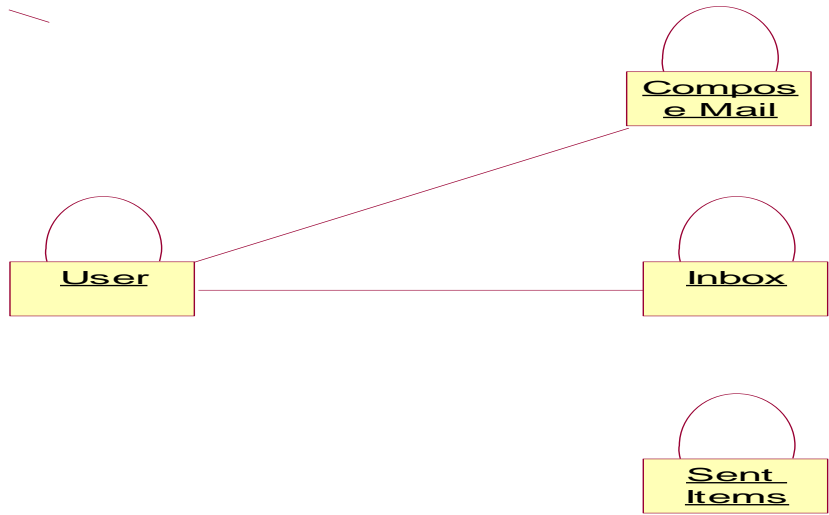


Figure 5.8

### 5.3.6 Object Diagram

#### 5.3.6.1 ADMIN

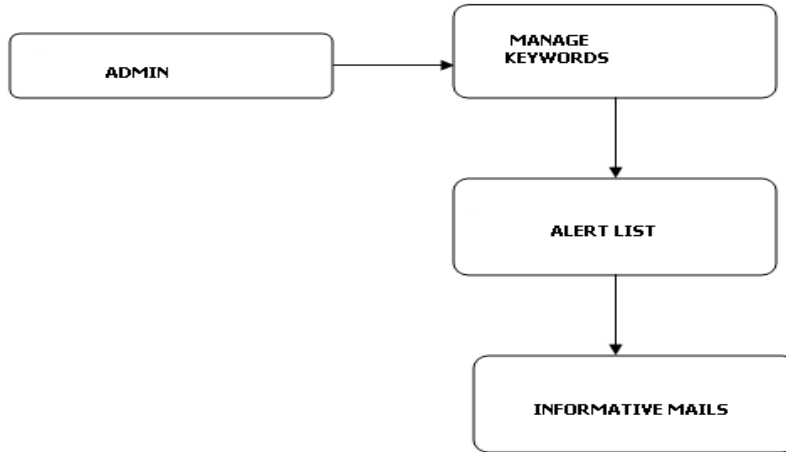


Figure 5.9

#### 5.3.6.2 USER

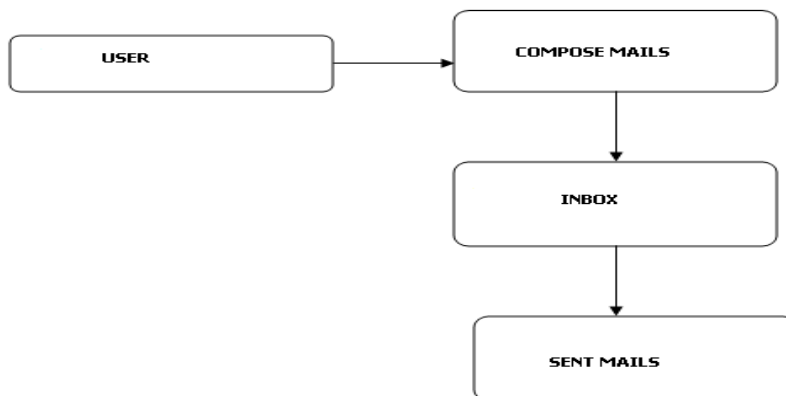


Figure 5.10

### 5.3.7 Use Case Diagram

#### 5.3.7.1 ADMIN

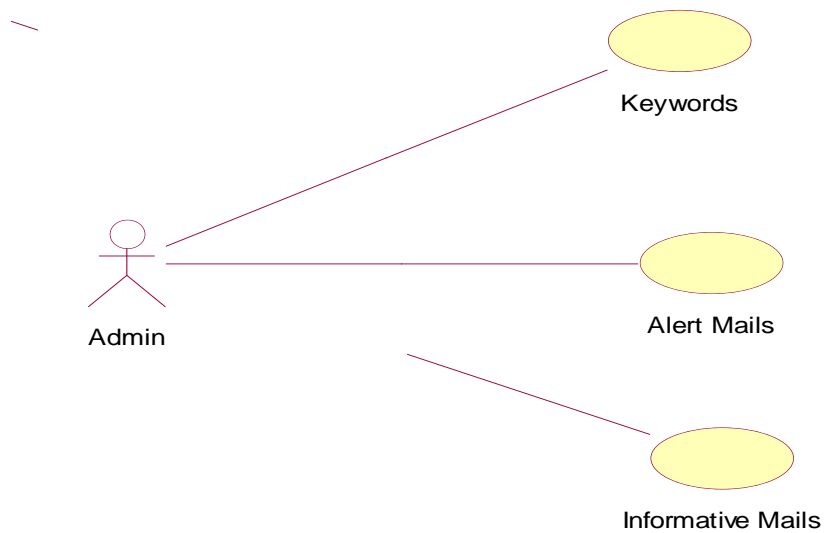


Figure 5.11

#### 5.3.7.2 USER

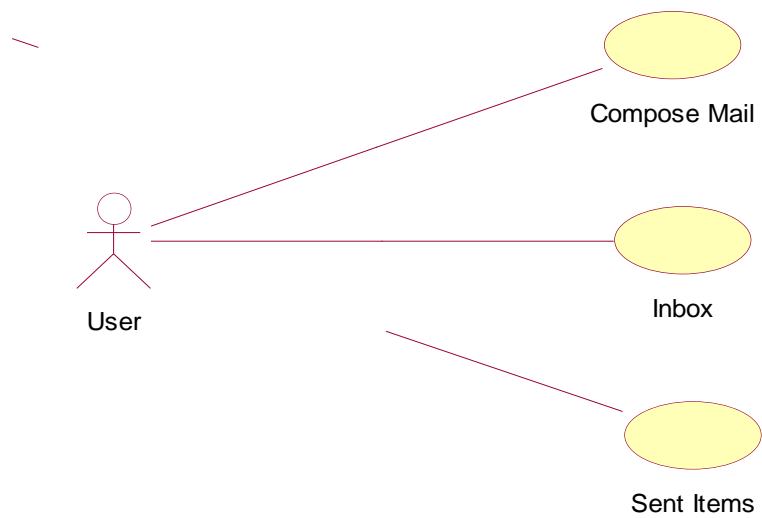


Figure 5.12

### 5.3.8 Component Diagram

#### 5.3.8.1 ADMIN

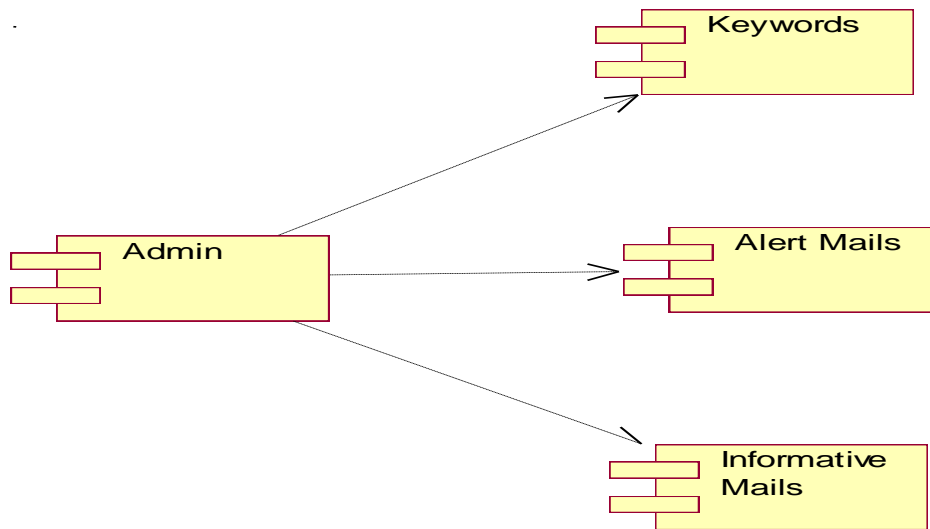


Figure 5.13

#### 5.3.8.2 USER

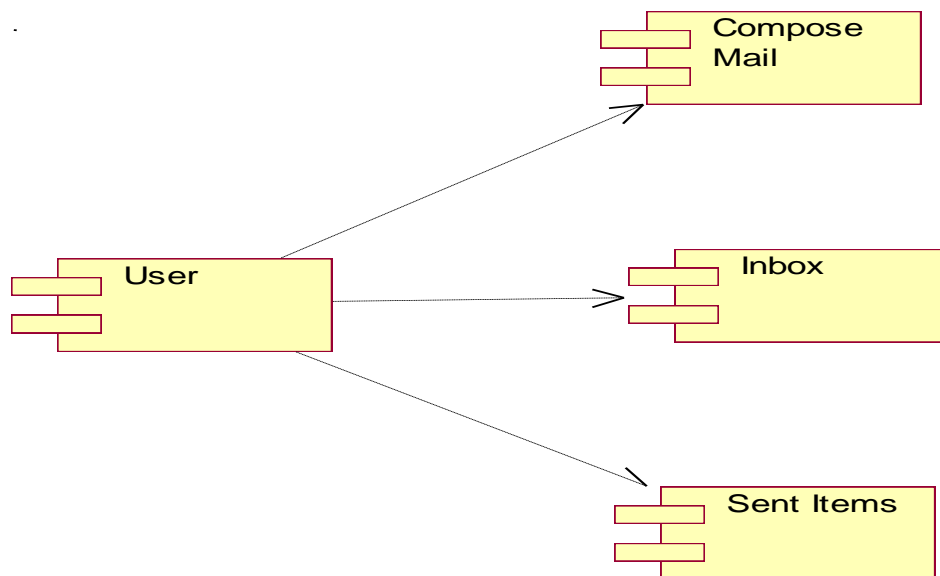


Figure 5.14



### 5.3.9 Deployment Diagram

#### 5.3.9.1 ADMIN

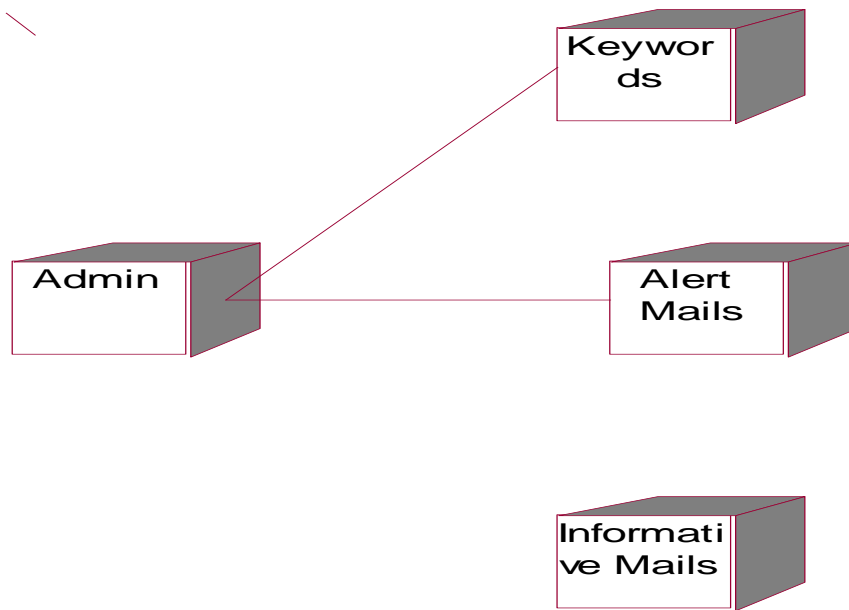


Figure 5.15

#### 5.3.9.2 USER

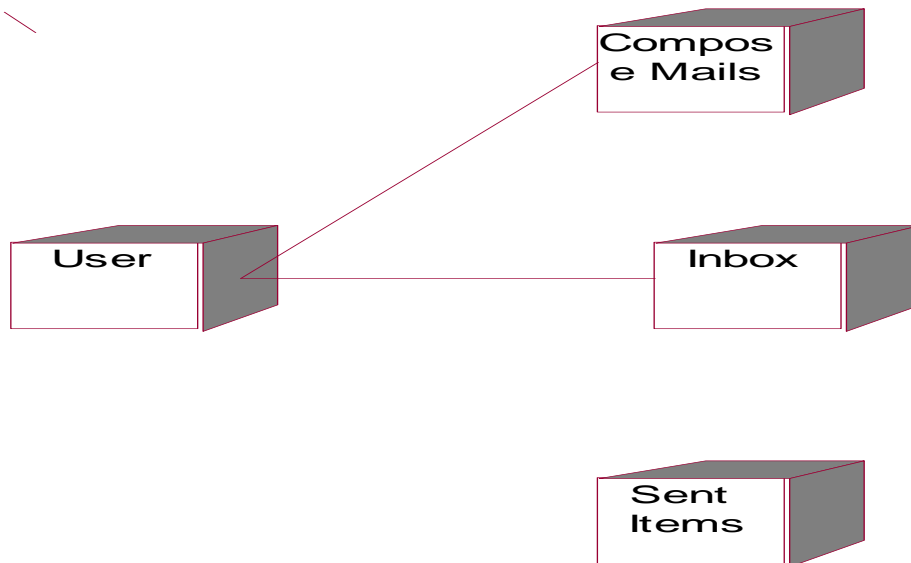


Figure 5.16

## Chapter 6

### DEVELOPMENT LANGUAGES

#### 6.1 Java

Java technology is both a programming language and a platform.

##### 6.1.1 Java Programming Language

The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

- Simple
- Object oriented
- Distributed
- Multithreaded
- Dynamic
- Architecture neutral
- Portable
- High performance
- Robust
- Secure

Each of the preceding buzzwords is explained in The Java Language Environment, a white paper written by James Gosling and Henry McGilton.

In the Java programming language, all source code is first written in plain text files ending with the .java extension. Those source files are then compiled into .Class files by the javac compiler. A .class file does not contain code that is native to your processor; it instead contains *bytecodes* — the machine language of the Java Virtual Machine (Java VM). The java launcher tool then runs your application with an instance of the Java Virtual Machine.

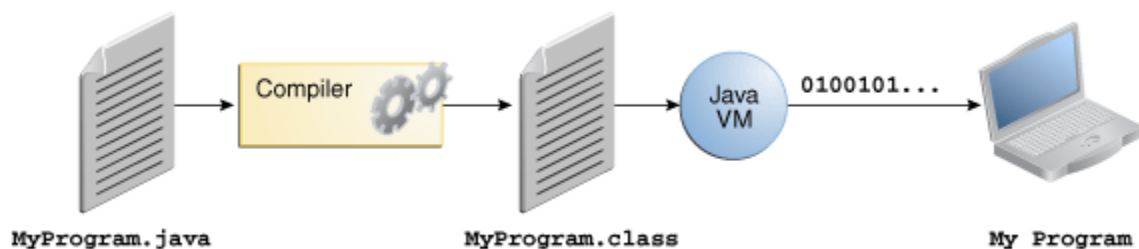


Figure 6.1

## An overview of the software development process

Because the Java VM is available on many different operating systems, the same .class files are capable of running on Microsoft Windows, the Solaris™ Operating System (Solaris OS), Linux, or Mac OS. Some virtual machines, such as the Java SE HotSpot at a Glance, perform additional steps at runtime to give your application a performance boost. This include various tasks such as finding performance bottlenecks and recompiling (to native code) frequently used sections of code.

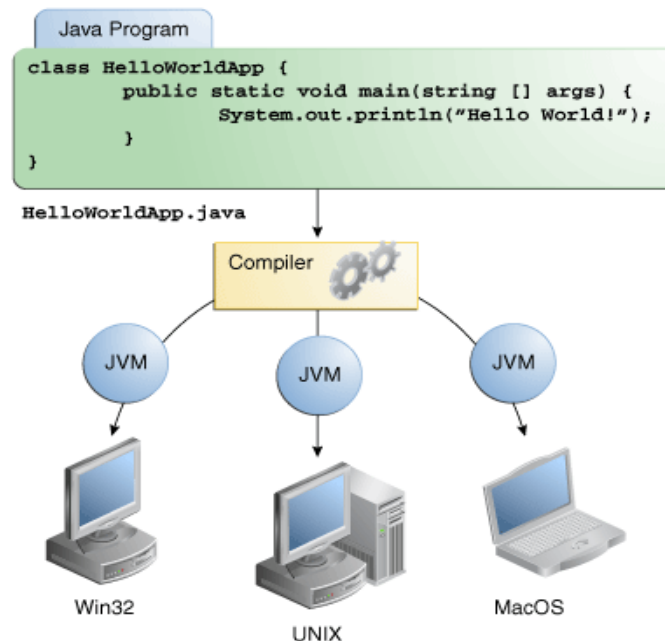


Figure 6.2

Through the Java VM, the same application is capable of running on multiple platforms.

### 6.1.2Java Platform

A platform is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Microsoft Windows, Linux, Solaris OS, and Mac OS. Most platforms can be described as a combination of the operating system and underlying hardware. The Java platform differs from most other

platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

The Java platform has two components:

- The Java Virtual Machine
- The Java Application Programming Interface (API)

The API is a large collection of ready-made software components that provide many useful capabilities. It is grouped into libraries of related classes and interfaces; these libraries are known as packages.

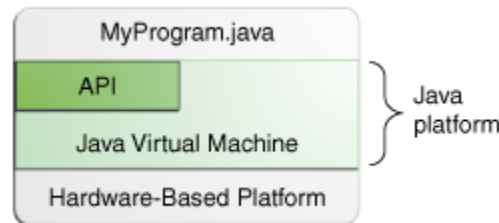


Figure 6.3

The API and Java Virtual Machine insulate the program from the underlying hardware.

As a platform-independent environment, the Java platform can be a bit slower than native code. However, advances in compiler and virtual machine technologies are bringing performance close to that of native code without threatening portability. The terms "Java Virtual Machine" and "JVM" mean a Virtual Machine for the Java platform.

### 6.1.3 What can Java Technology do?

The general-purpose, high-level Java programming language is a powerful software platform. Every full implementation of the Java platform gives you the following features:

- **Development Tools:** The development tools provide everything you'll need for compiling, running, monitoring, debugging, and documenting your applications. As a new developer, the main tools you'll be using are the javac compiler, the java launcher, and the javadoc documentation tool.
- **Application Programming Interface (API):** The API provides the core functionality of the Java programming language. It offers a wide array of useful classes ready for use in your own applications. It spans everything from basic objects, to networking and security, to XML generation and database access, and more. The core API is very large; to get an overview of what it contains, consult the Java Platform Standard Edition 7 Documentation.
- **Deployment Technologies:** The JDK software provides standard mechanisms such as the Java Web Start software and Java Plug-In software for deploying your applications to end users.
- **User Interface Toolkits:** The Swing and Java 2D toolkits make it possible to create sophisticated Graphical User Interfaces (GUIs).
- **Integration Libraries:** Integration libraries such as the Java IDL API, JDBC™ API, Java Naming and Directory Interface™ (JNDI) API, Java RMI, and Java Remote Method Invocation over Internet Inter-ORB Protocol Technology (Java RMI-IIOP Technology) enable database access and manipulation of remote objects.

#### 6.1.4 Java as an Emerging Technology

It is likely to make your programs better and requires less effort than other languages. We believe that Java technology will help you do the following:

- **Get started quickly:** Although the Java programming language is a powerful object-oriented language, it's easy to learn, especially for programmers already familiar with C or C++.
- **Write less code:** Comparisons of program metrics (class counts, method counts, and so on) suggest that a program written in the Java programming language can be four times smaller than the same program written in C++.

- **Write better code:** The Java programming language encourages good coding practices, and automatic garbage collection helps you avoid memory leaks. Its object orientation, its JavaBeans™ component architecture, and its wide-ranging, easily extendible API let you reuse existing, tested code and introduce fewer bugs.
- **Develop programs more quickly:** The Java programming language is simpler than C++, and as such, your development time could be up to twice as fast when writing in it. Your programs will also require fewer lines of code.
- **Avoid platform dependencies:** You can keep your program portable by avoiding the use of libraries written in other languages.
- **Write once, run anywhere:** Because applications written in the Java programming language are compiled into machine-independent bytecodes, they run consistently on any Java platform.
- **Distribute software more easily:** With Java Web Start software, users will be able to launch your applications with a single click of the mouse. An automatic version check at startup ensures that users are always up to date with the latest version of your software. If an update is available, the Java Web Start software will automatically update their installation.

## 6.2 SERVLET

### 6.2.1 Introduction to Servlets

Servlets are modules that run inside request/response-oriented servers, such as Java-enabled web servers. Functionally they operate in a very similar way to CGI scripts, however, being Java based they are more platform independent.

Some Example Applications

A few of the many applications for servlets include,

- Processing data POSTed over HTTPS using an HTML form, including purchase order or credit card data. A servlet like this could be part of an order-entry and processing system, working with product and inventory databases, and perhaps an on-line payment system.

- Allowing collaboration between people. A servlet can handle multiple requests concurrently; they can synchronize requests to support systems such as on-line conferencing.
- Forwarding requests. Servlets can forward requests to other servers and servlets. This allows them to be used to balance load among several servers that mirror the same content. It also allows them to be used to partition a single logical service over several servers, according to task type or organizational boundaries.

### 6.2.2 Servlet Architecture

The central abstraction in the Servlet API is the Servlet interface. All servlets implement this interface, either directly or, more commonly, by extending a class that implements it such as `HttpServlet`. The inheritance hierarchy looks as follows.

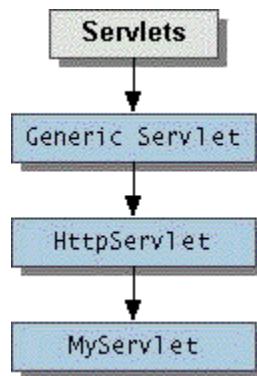


Figure 6.4

The Servlet interface provides the following methods that manage the servlet and its communications with clients.

- **destroy()**  
Cleans up whatever resources are being held and makes sure that any persistent state is synchronized with the servlet's current in-memory state.
- **getServletConfig()**  
Returns a servlet config object, which contains any initialization parameters and startup configuration for this servlet.

- **getServletInfo()**  
Returns a string containing information about the servlet, such as its author, version, and copyright.
- **init(ServletConfig)**  
Initializes the servlet. Run once before any requests can be serviced.
- **service(ServletRequest,ServletResponse)**  
Carries out a single request from the client.

Servlet writers provide some or all of these methods when developing a servlet. When a servlet accepts a service call from a client, it receives two objects, ServletRequest and ServletResponse. The ServletRequest class encapsulates the communication from the client to the server, while the ServletResponse class encapsulates the communication from the servlet back to the client.

The ServletRequest interface allows the servlet access to information such as the names of the parameters passed in by the client, the protocol (scheme) being used by the client, and the names of the remote host that made the request and the server that received it. It also provides the servlet with access to the input stream, ServletInputStream, through which the servlet gets data from clients that are using application protocols such as the HTTP POST and PUT methods. Subclasses of ServletRequest allow the servlet to retrieve more protocol-specific data. For example, HttpServletRequest contains methods for accessing HTTP-specific header information.

The ServletResponse interface gives the servlet methods for replying to the client. It allows the servlet to set the content length and mime type of the reply, and provides an output stream, ServletOutputStream, and a Writer through which the servlet can send the reply data. Subclasses of ServletResponse give the servlet more protocol-specific capabilities. For example, HttpServletResponse contains methods that allow the servlet to manipulate HTTP-specific header information.

The classes and interfaces described above make up a basic Servlet. HTTP servlets have some additional objects that provide session-tracking capabilities. The



servlet writer can use these APIs to maintain state between the servlet and the client that persists across multiple connections during some time period.

### **6.2.3 Servlet Lifecycle**

Servers load and run servlets, which then accept zero or more requests from clients and return data to them. They can also remove servlets. These are the steps of a servlets lifecycle. The next paragraphs describe each step in more detail, concentrating on concurrency issues.

When a server loads a servlet, it runs the servlet's init method. Even though most servlets are run in multi-threaded servers, there are no concurrency issues during servlet initialization. This is because the server calls the init method once, when it loads the servlet, and will not call it again unless it is reloading the servlet. The server can not reload a servlet until after it has removed the servlet by calling the destroy method. Initialization is allowed to complete before client requests are handled (that is, before the service method is called) or the servlet is destroyed.

After the server loads and initializes the servlet, the servlet is able to handle client requests. It processes them in its service method. Each client's request has its call to the service method run in its own servlet thread: the method receives the client's request, and sends the client its response.

Servlets can run multiple service methods at a time. It is important, therefore, that service methods be written in a thread-safe manner. If, for some reason, a server should not run multiple service methods concurrently, the servlet should implement the `SingleThreadModel` interface. This interface guarantees that no two threads will execute the servlet's service methods concurrently.

Servlets run until they are removed from the service. When a server removes a servlet, it runs the servlet's destroy method. The method is run once; the server will not run it again until after it reloads and reinitializes the servlet.

The diagram below shows the basic interactions between clients, a web server, and a servlet registered with the web server.

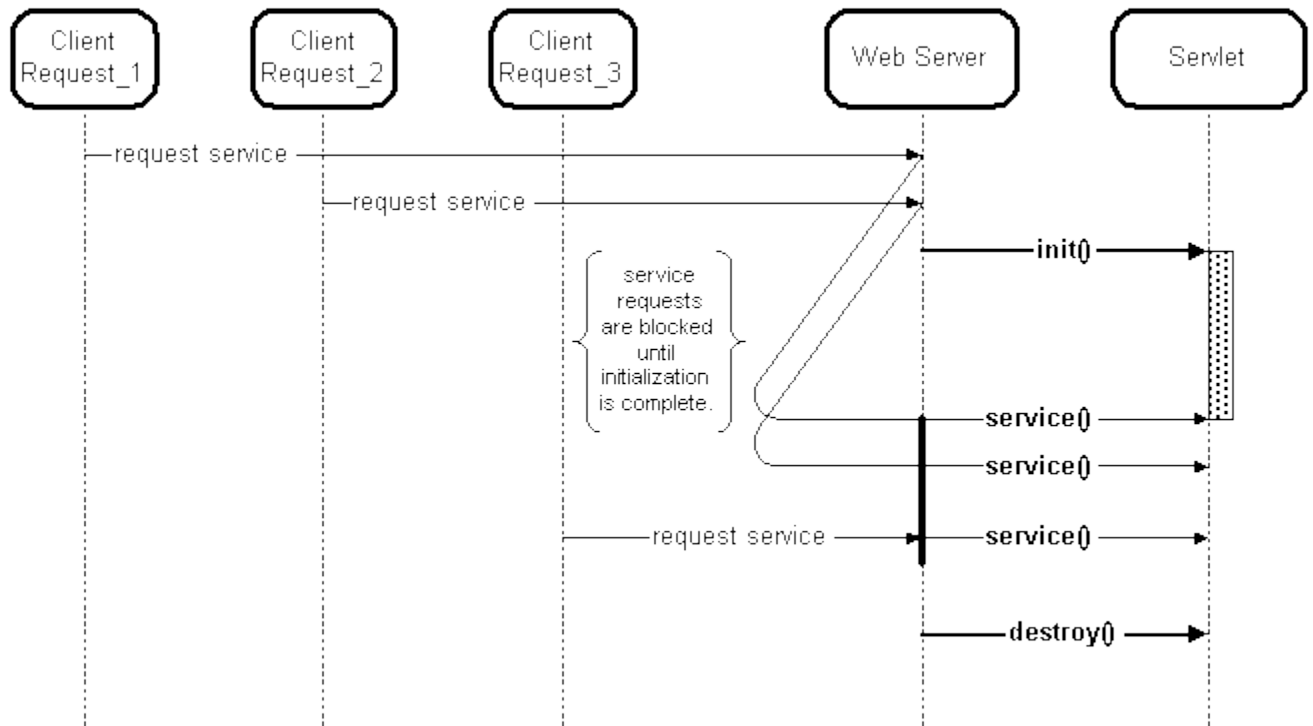


Figure 6.5

When the destroy method runs, however, other threads might be running service requests. If, in cleaning up, it is necessary to access shared resources, that access should be synchronized. During a servlet's lifecycle, it is important to write thread-safe code for destroying the servlet and, unless the servlet implements the `SingleThreadModel` interface, servicing client requests.

#### 6.2.4 Writing the Servlet

Servlets implement the `javax.servlet.Servlet` interface. While servlet writers can develop servlets by implementing the interface directly, this is usually not required. Because most servlets extend web servers that use the HTTP protocol to interact with clients, the most common way to develop servlets is by specializing the `javax.servlet.http.HttpServlet` class. This tutorial concentrates on describing this method of writing servlets.

The `HttpServlet` class implements the `Servlet` interface by extending the `GenericServlet` base class, and provides a framework for handling the HTTP protocol. Its service method supports standard HTTP/1.1 requests by dispatching each request to a method designed to handle it.

By default, servlets written by specializing the `HttpServlet` class can have multiple threads concurrently running its service method. If, you would like to have only a single thread running a single service method at a time, then in addition to extending the `HttpServlet`, your servlet should also implement the `SingleThreadModel` interface. This does not involve writing any extra methods, merely declaring that the servlet implements the interface. For example,

```
public class SurveyServlet extends HttpServlet
    implements SingleThreadModel {

    /* typical servlet code, with no threading concerns
     * in the service method. No extra code for the
     * SingleThreadModel interface. */
    ...

}
```

### 6.2.5 Interacting with Clients

Servlet writers who are developing HTTP servlets that specialize the `HttpServlet` class should override the method or methods designed to handle the HTTP interactions that their servlet will handle. The candidate methods include,

- `doGet`, for handling GET, conditional GET and HEAD requests
- `doPost`, for handling POST requests
- `doPut`, for handling PUT requests
- `doDelete`, for handling DELETE requests

By default, these methods return a `BAD_REQUEST` (400) error. An example HTTP servlet that handles `GET` and `HEAD` requests follows; it specializes the `doGet` method. The second example is also provided. It handles `POST` requests from a form by specializing the `doPost` method.

The `HttpServlet`'s service method, by default, also calls the `doOptions` method when it receives an `OPTIONS` request, and `doTrace` when it receives a `TRACE` request. The default implementation of `doOptions` automatically determines what HTTP options are supported and returns that information. The default implementation of `doTrace` causes a response with a message containing all of the headers sent in the trace request. These methods are not typically overridden.

Whatever method you override, it will take two arguments. The first encapsulates the data from the client, and is an `HttpServletRequest`. The second encapsulates the response to the client, and is an `HttpServletResponse`. The following paragraphs discuss their use.

An `HttpServletRequest` object provides access to HTTP header data, such as any cookies found in the request and the HTTP method with which the request was made. It, of course, allows the you to obtain the arguments that the client sent as part of the request. How you access the client data might depend on the HTTP method of the request.

- For any HTTP method, you can use the `getParameterValues` method, which will return the value of a named parameter. (The method `getParameterNames` provides the names of the parameters.) You can also manually parse the request.
- For requests using the HTTP `GET` method, the `getQueryString` method will return a `String` to be parsed.
- For HTTP methods `POST`, `PUT`, and `DELETE`, you have the choice between two methods. If you expect text data, then it can be read using the `BufferedReader` returned by the `getReader` method, if you expect binary data,

then it should be read with the `ServletInputStream` returned by the `getInputStream` method.

Note that you should use either the `getParameterValues` method or one of the methods that allow you to parse the data yourself. They can not be used together in a single request.

For responding to the client, an `HttpServletResponse` object provides two ways of returning the response data to the user. You can use the writer returned by the `getWriter` method or the output stream returned by the `getOutputStream` method. You should use `getWriter` to return text data to the user, and `getOutputStream` for binary data.

Before accessing the `Writer` or `OutputStream`, HTTP header data should be set. The `HttpServletResponse` class provides methods to access the header data, such as the content type and encoding, and content length of the response. After you set the headers, you may obtain the writer or output stream and send the body of the response to the user. Closing the writer or output stream after sending the response to the client allows the server to know when the response is complete.

### **Example of an HTTP Servlet that handles the GET and HEAD methods**

```
/**
 * This is a simple example of an HTTP Servlet. It responds to the GET
 * and HEAD methods of the HTTP protocol.
 */
public class SimpleServlet extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        // set header field first
        res.setContentType("text/html");

        // then get the writer and write the response data
```

```

    PrintWriter out = res.getWriter();
    out.println("<HEAD><TITLE>SimpleServlet
Output</TITLE></HEAD><BODY>");
    out.println("<h1> SimpleServlet Output </h1>");
    out.println("<P>This is output is from SimpleServlet.");
        out.println("</BODY>");
        out.close();
    }

    public String getServletInfo() {
        return "A simple servlet";
    }

}

```

### 6.2.6 Lifecycle Methods

Servlets that manage resources do so by overriding the lifecycle methods `init` and `destroy`. These servlets might need to be given arguments at startup, in order to initialize correctly.

#### Overriding the Init Method

During initialization, the servlet should prepare the resources it manages, to ready the servlet for accepting service requests. It can do this without regard for multi-threading concerns, since there is only a single thread running on behalf of the servlet during initialization. As soon as the `init` method returns, the servlet can receive client requests. If, for some reason, the servlet's required resources cannot be made available (for example, a required network connection cannot be established), or some other initialization error occurs that would make it impossible for the servlet to handle requests, the `init` method should throw an `UnavailableException` exception.

The `init` method takes a `ServletConfig` object as a parameter. The method should save this object, so that it can be returned by the `getServletConfig` method. The simplest

way to do this is to have the `newinit` method call `super.init`. If you do not do this, you should store the `ServletConfig` object yourself, and override the `getServletConfig` method so that it can obtain the object from its new location.

An example `init` method follows. It is the `init` method from the `Survey Servlet`, which accepts input from a form and stores it in a file. In order to store the survey information, it needs a directory. It receives the directory as an initialization parameter; initialization parameters are discussed in the next section.

```
public void init(ServletConfig config)
    throws ServletException
{
    super.init(config);

    //Store the directory that will hold the survey-results files
    resultsDir = getInitParameter("resultsDir");

    //If no directory was provided, can't handle clients
    if (resultsDir == null) {
        throw new UnavailableException (this,
            "Not given a directory to write survey results!");
    }
}
```

As you can see, this `init` method calls the `super.init` method to manage the `ServletConfig` object. The `init` method also sets a field, `resultsDir`, with the directory name that is provided as an initialization parameter. If no directory name was provided, the servlet throws an unavailable exception. If the `init` method completes successfully, the servlet can then handle client requests.

## Initialization Parameters

The specification of initialization parameters is server-specific. For example, they are specified with a property when a servlet is run with the servlet runner. This tutorial contains a general explanation of properties, and how to create them.

However the initialization parameters are specified, they are always obtained the same way: with the `getInitParameter` method. This method takes the parameter name as an argument. The example `initMethod` calls `getInitParameter`. If, for some reason, you need to get the parameter names, you can get them with the `getParameterNames` method.

## Overriding the Destroy Method

When a server unloads a servlet, it calls the servlet's destroy method. The destroy method should undo any initialization work and synchronize persistent state with the current in-memory state. This section begins with a description of how to write a simple destroy method, then describes how to structure a servlet if threads running its service method might still be running when the destroy method is called.

Though it is often the case that a servlet that overrides the `init` method must also override the destroy method to undo that initialization, this is not required. Because initialization involves reading a file and using its contents to initialize a shared data structure, there is no work to undo when the server is finished with the servlet.

For many servlets, however, there is initialization work that must be undone. For example, assume there is a servlet that opens a database connection during initialization. Its destroy method, shown as an example below, would close that connection.

```
/**
 * Cleans up database connection
 */
public void destroy() {
    try {
        con.close();
    }
```



```

    } catch (SQLException e) {
        while(e != null) {
            log("SQLException: " + e.getSQLState() + '\t' +
                e.getMessage() + '\t' +
                e.getErrorCode() + '\t');
            e = e.getNextException();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

### **Coping with Service Threads at Servlet Termination**

When a server removes a servlet, it typically calls `destroy` after all service calls have been completed, or a server-specific number of seconds have passed, whichever comes first. If your servlet has operations that take a long time to run (that is, they may run longer than the server's grace period), then threads could still be running when `destroy` is called. The servlet writer is responsible for making sure that any threads still handling client requests complete. The remainder of this section describes a technique for doing this.

A servlet with potentially long-running service requests should keep track of how many service methods are currently running. Its long-running methods should periodically poll to make sure that they should continue to run. If the servlet is being destroyed, then the long-running method should stop working, clean up if necessary, and return.

For example, the instance variable that counts the number of service methods running could be called `serviceCounter`, and the indicator of whether the servlet is being destroyed could be an instance variable called `shuttingDown`. Each variable should have its own set of access methods.

```

public ShutdownExample extends HttpServlet {

```

```

private int serviceCounter = 0;
private Boolean shuttingDown;
//Access methods for serviceCounter
protected synchronized void enteringServiceMethod() {
    serviceCounter++;
}
protected synchronized void leavingServiceMethod() {
    serviceCounter--;
}
protected synchronized int numServices() {
    return serviceCounter;
}
//Access methods for shuttingDown
protected setShuttingDown(Boolean flag) {
    shuttingDown = flag;
}
protected Boolean isShuttingDown() {
    return shuttingDown;
}
}

```

The service method should increment the service counter each time it is entered and decrement it each time it returns.

```

protected void service(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException
{
    enteringServiceMethod();
    try {
        super.service(req, resp);
    } finally {
        leavingServiceMethod();
    }
}

```

```

    }
}

```

The destroy method should check the serviceCounter, and if there are any long-running methods, set the shuttingDown variable. This variable will let the threads still handling client requests know that it is time to shut down. The destroy method should then wait for the service methods to complete, in order to provide a clean shutdown.

```

public void destroy() {
    /* Check to see whether there are still service methods running,
       * and if there are, tell them to stop. */
    if (numServices() > 0) {
        setShuttingDown(true);
    }

    /* Wait for the service methods to stop. */
    while(numService() > 0) {
        try {
            thisThread.sleep(interval);
        } catch (InterruptedException e) {
        }
    }
}
}

```

## Providing Information about the Servlet

Some applets and applications, for example, the Java Web Server Administration Tool, display information about a servlet. This information can include a short description of the purpose of the servlet, its author, and perhaps its version number. The Servlet API provides a method to return this information, `getServletInfo`. By default, this method returns null. While servlet writers are not required to override this method, it is strongly recommended. The simple servlet, shown as an example earlier, overrides this method:

```

/**
 * This is a simple example of an HTTP Servlet. It responds to the GET
 * and HEAD methods of the HTTP protocol.
 */

```

```

public class SimpleServlet extends HttpServlet {
    public String getServletInfo() {
        return "A simple servlet";
    }
}

```

The URL for a servlet has the following general form:

*http://machine-name:port/servlet/servlet-name*

where *servlet-name* corresponds to the name you have given your servlet.

### 6.3 JSP

The first Java Server Pages specification was released in 1999. Originally JSP was modeled after other server-side template technologies to provide a simple method of embedding dynamic code with static markup. When a request is made for the content of a JSP, a container interprets the JSP, executes any embedded code, and sends the results in a response. At the time this type of functionality was nothing terribly new, but it was and still is a helpful enhancement to Servlets.

JSP simply puts Java inside HTML pages. You can take any existing HTML page and change its extension to ".jsp" instead of ".html". In fact, this is the perfect exercise for your first JSP.

JSP has been revised several times since the original release, each adding functionality, and is currently in version 2.0. The JSP specifications are developed alongside the Servlet specifications and can be found on Sun Microsystems'

The functionality defined by the JSP 2.0 specifications can be broken down as follows:

The JSP specifications define the basic syntax and semantics of a Java Server Page. A basic Java Server Page consists of plain text and markup and can optionally take advantage of embedded scripts and other functionality for creating dynamic content.

JSP includes a mechanism for defining dynamic attributes for custom tags. Any scripting language can be used for this purpose; usually Java is implemented, but the JSP

specification defines a custom expression language designed specifically for the task. Often the JSP EL is a much simpler and more flexible solution, especially when combined with JSP design patterns that do not use embedded scripts.

Discussing the basics of JSP is the focus of this chapter. JavaBeans, Custom Tags, and the JSP Expression Language are all fully discussed in later chapters after a proper foundation of JSP is established.

### 6.3.1 JSP LIFE CYCLE

Much like Servlets, understanding JSP requires understanding the simple life cycle that JSP follows. JSP follows a three-phase life cycle: initialization, service, and destruction. This life cycle should seem familiar and is identical to the one described for Servlets.

While a JSP does follow the Servlet life cycle, the methods have different names. Initialization corresponds to the `jspInit()` method, service corresponds to the `jspService()` method, and destruction corresponds to the `jspDestroy()` method. The three phases are all used the same as a Servlet and allow a JSP to load resources, provide service to multiple client requests, and destroy loaded resources when the JSP is taken out of service.

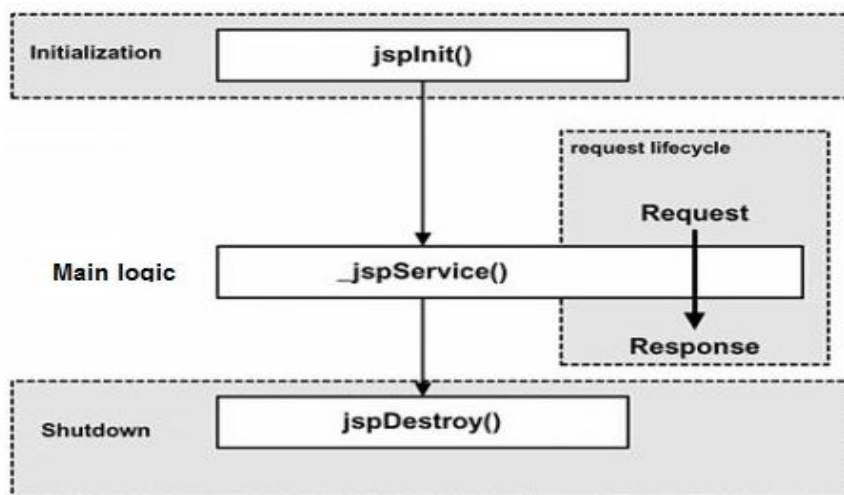


Figure 6.4

## 6.4 JSP - Expression Language (EL)

JSP Expression Language (EL) makes it possible to easily access application data stored in JavaBeans components. JSP EL allows you to create expressions both (a) arithmetic and (b) logical. Within a JSP EL expression, you can use integers, floating point numbers, strings, the built-in constants true and false for Boolean values, and null.

### 6.4.1 Simple Syntax

Typically, when you specify an attribute value in a JSP tag, you simply use a string. For example:

```
<jsp:setProperty name="box" property="perimeter" value="100"/>
```

JSP EL allows you to specify an expression for any of these attribute values. A simple syntax for JSP EL is as follows:

```
${expr}
```

Here **expr** specifies the expression itself. The most common operators in JSP EL are . and []. These two operators allow you to access various attributes of Java Beans and built-in JSP objects.

For example above syntax <jsp:setProperty> tag can be written with an expression like:

```
<jsp:setProperty name="box" property="perimeter"  
    value="${2*box.width+2*box.height}"/>
```

When the JSP compiler sees the \${ } form in an attribute, it generates code to evaluate the expression and substitutes the value of expression.

You can also use JSP EL expressions within template text for a tag. For example, the `<jsp:text>` tag simply inserts its content within the body of a JSP. The following `<jsp:text>` declaration inserts `<h1>Hello JSP!</h1>` into the JSP output:

```
<jsp:text>
<h1>Hello JSP!</h1>
</jsp:text>
```

You can include a JSP EL expression in the body of a `<jsp:text>` tag (or any other tag) with the same `${}` syntax you use for attributes. For example:

```
<jsp:text>
Box Perimeter is: ${2*box.width + 2*box.height}
</jsp:text>
```

EL expressions can use parentheses to group sub expressions. For example, `${(1 + 2) * 3}` equals 9, but `${1 + (2 * 3)}` equals 7.

To deactivate the evaluation of EL expressions, we specify the `isELIgnored` attribute of the `page` directive as below:

```
<%@ page isELIgnored = "true|false" %>
```

The valid values of this attribute are `true` and `false`. If it is `true`, EL expressions are ignored when they appear in static text or tag attributes. If it is `false`, EL expressions are evaluated by the container.

## 6.5 JSTL

The Java Server Pages Standard Tag Library (JSTL) is a collection of useful JSP tags which encapsulates core functionality common to many JSP applications.

JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags. It also provides a framework for integrating existing custom tags with JSTL tags.

The JSTL tags can be classified, according to their functions, into following JSTL tag library groups that can be used when creating a JSP page:

- **Core Tags**
- **Formatting tags**
- **SQL tags**
- **XML tags**
- **JSTL Functions**

Area	Subfunction	Prefix
Core	Variable support	c
	Flow control	
	URL management	
	Miscellaneous	
XML	Core	x
	Flow control	
	Transformation	
I18N	Locale	fmt
	Message formatting	
	Number and date formatting	
Database	SQL	sql
Functions	Collection length	fn
	String manipulation	

Thus, the tutorial references the JSTL core tags in JSP pages by using the following `taglib` directive:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

Figure 6.5

## 6.6 JavaBeans:

JavaBeans are reusable software components for Java. They are classes that encapsulate many objects into a single object (the bean). They are serializable have a 0-argument constructor, and allow access to properties using getter and setter methods.



### 6.6.1 JavaBeans Conventions

In order to function as a JavaBean class, an object class must obey certain conventions about method naming, construction, and behavior. These conventions make it possible to have tools that can use, reuse, replace, and connect JavaBeans.

The required conventions are as follows:

- The class must have a public default constructor (with no arguments). This allows easy instantiation within editing and activation frameworks.
- The class properties must be accessible using *get*, *set*, *is* (can be used for Boolean properties instead of *get*), and other methods (so-called accessor methods and mutator methods) according to a standard naming convention. This allows easy automated inspection and updating of bean state within frameworks, many of which include custom editors for various types of properties. Setters can have one or more than one argument.
- The class should be serializable. This allows applications and frameworks to reliably save, store, and restore the bean's state in a manner independent of the VM and of the platform.

```
package player;

public class PersonBean implements java.io.Serializable {

    /**
     * Property <code>name</code> (note capitalization) readable/writable.
     */
    private String name = null;

    private boolean deceased = false;

    /** No-arg constructor (takes no arguments). */
    public PersonBean() {
```

```

}

/**
 * Getter for property <code>name</code>
 */
public String getName() {
    return name;
}

/**
 * Setter for property <code>name</code>.
 * @param value
 */
public void setName(final String value) {
    name = value;
}

/**
 * Getter for property "deceased"
 * Different syntax for a boolean field (is vs. get)
 */
public boolean isDeceased() {
    return deceased;
}

/**
 * Setter for property <code>deceased</code>.
 * @param value
 */
public void setDeceased(final boolean value) {
    deceased = value;
}

```

## 6.7 POJO (Plain Old Java Object)

In computing software, POJO is an acronym for Plain Old Java Object. The name is used to emphasize that a given object is an ordinary Java Object, not a special object.

We wondered why people were so against using regular objects in their systems and concluded that it was because simple objects lacked a fancy name. So we gave them one, and it's caught on very nicely.

The term "POJO" is mainly used to denote a Java object which does not follow any of the major Java object models, conventions, or frameworks. The term continues the pattern of older terms for technologies that do not use fancy new features, such as POTS (Plain Old Telephone Service) in telephony, PODS (Plain Old Data Structures) that are defined in C++ but use only C language features, and POD (Plain Old Documentation) in Perl. The equivalent to POJO on the .NET framework is Plain Old CLR Object (POCO). For PHP, it is Plain Old PHP Object (POPO).

The POJO phenomenon has most likely gained widespread acceptance because of the need for a common and easily understood term that contrasts with complicated object frameworks.

Ideally speaking, a POJO is a Java object not bound by any restriction other than those forced by the Java Language Specification. I.e., a POJO **should not** have to

1. Extend prespecified classes, as in

```
Public class Foo extends javax.servlet.http.HttpServlet
```

2. Implement prespecified interfaces, as in

```
Public class Bar implements javax.ejb.EntityBean
```

3. Contain prespecified annotations, as in

```
@javax.persistence.Entity public class Baz
```

However, due to technical difficulties and other reasons, many software products or frameworks described as POJO-compliant actually still require the use of prespecified annotations for features such as persistence to work properly. The idea is that if the object

(actually class) was a POJO before any annotations were added, and would return to POJO status if the annotations are removed then it can still be considered a POJO. Then the basic object remains a POJO in that it has no special characteristics (such as an implemented interface) that makes it a "Specialized Java Object" (SJO or (sic) SoJO).

## 6.8 HTML

HTML is a **markup** language for **describing** web documents (web pages).

- HTML stands for **H**yper **T**ext **M**arkup **L**anguage
- A markup language is a set of **markup tags**
- HTML documents are described by **HTML tags**
- Each HTML tag **describes** different document content

HTML Example

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Heading</h1>
<p>My first paragraph.</p>
</body>
</html>
```

Example Explained

- The **DOCTYPE** declaration defines the document type
- The text between **<html>** and **</html>** describes the web document
- The text between **<body>** and **</body>** describes the visible page content
- The text between **<h1>** and **</h1>** describes a heading
- The text between **<p>** and **</p>** describes paragraph

Using the description, a web browser can display a document with a heading and a paragraph.

### 6.8.1 HTML Tags

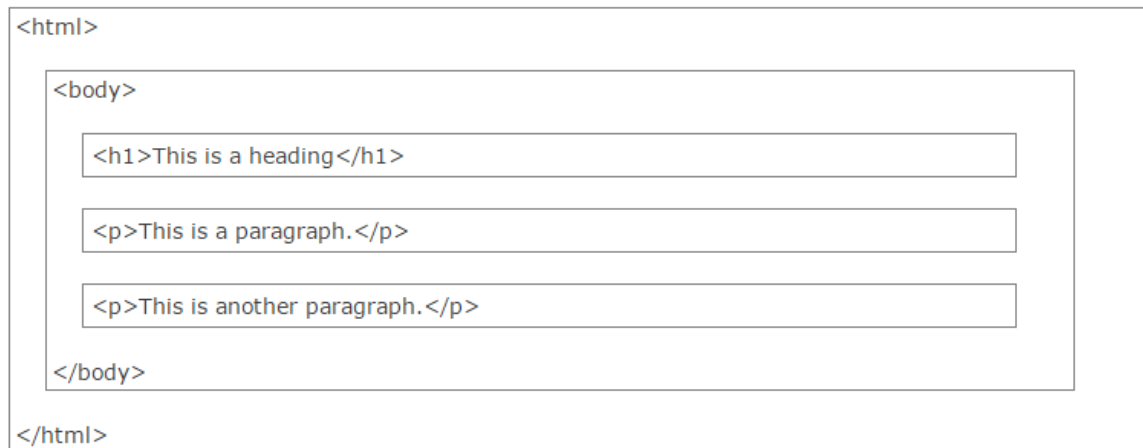
HTML tags are **keywords** (tag names) surrounded by **angle brackets**:

`<tagname>content</tagname>`

- HTML tags normally come **in pairs** like `<p>` and `</p>`
- The first tag in a pair is the **start tag**, the second tag is the **end tag**
- The end tag is written like the start tag, but with a **slash** before the tag name
- The start tag is often called the **opening tag**. The end tag is often called the **closing tag**.

### 6.8.2 HTML Page Structure

Below is a visualization of an HTML page structure:



### 6.8.3 HTML Versions

Since the early days of the web, there have been many versions of HTML:

Version	Year
HTML	1991
HTML+	1993

HTML 2.0	1995
HTML 3.2	1997
HTML 4.01	1999
XHTML	2000
HTML5	2012

## 6.9 CSS

- CSS stands for **Cascading Style Sheets**
- Styles define **how to display** HTML elements
- Styles were added to HTML 4.0 **to solve a problem**
- **External Style Sheets** can save a lot of work
- External Style Sheets are stored in **CSS files**

### Styles Solved a Big Problem

HTML was never intended to contain tags for formatting a document.

HTML was intended to define the content of a document, like:

```
<h1>This is a heading</h1>
```

```
<p>This is a paragraph.</p>
```

When tags like `<font>`, and color attributes were added to the HTML 3.2 specification, it started a nightmare for web developers. Development of large web sites, where fonts and color information were added to every single page, became a long and expensive process.

In HTML 4.0, all formatting could be removed from the HTML document, and stored in a separate CSS file.

All browsers support CSS today.

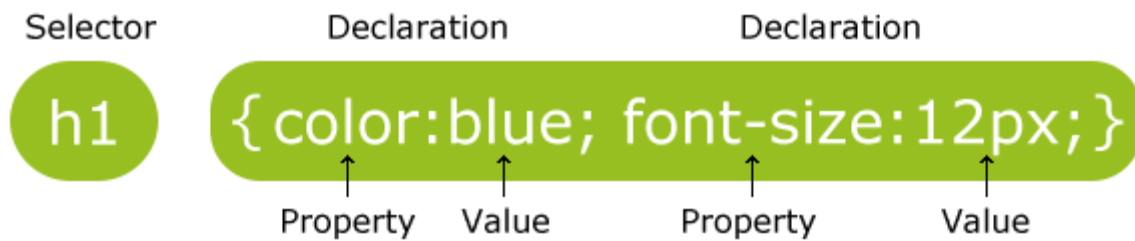
### 6.9.1 CSS Saves a Lot of Work!

CSS defines HOW HTML elements are to be displayed.

Styles are normally saved in external .css files. External style sheets enable you to change the appearance and layout of all the pages in a Web site, just by editing one single file!

### 6.9.2 CSS Syntax

A CSS rule set consists of a selector and a declaration block:



The selector points to the HTML element you want to style.

The declaration block contains one or more declarations separated by semicolons.

Each declaration includes a property name and a value, separated by a colon.

### CSS Example

A CSS declaration always ends with a semicolon, and declaration groups are surrounded by curly braces:

```
p {color:red;text-align:center;}
```

To make the CSS code more readable, you can put one declaration on each line, like this:

### Example

```
p {  
    color: red;  
    text-align: center;  
}
```

## 6.9.4 CSS Comments

Comments are used to explain your code, and may help you when you edit the source code at a later date. Comments are ignored by browsers.

A CSS comment starts with `/*` and ends with `*/`. Comments can also span multiple lines:

Example

```
p {  
    color: red;  
    /* This is a single-line comment */  
    text-align: center;  
}  
/* This is  
a multi-line  
comment */
```



# CHAPTER 7

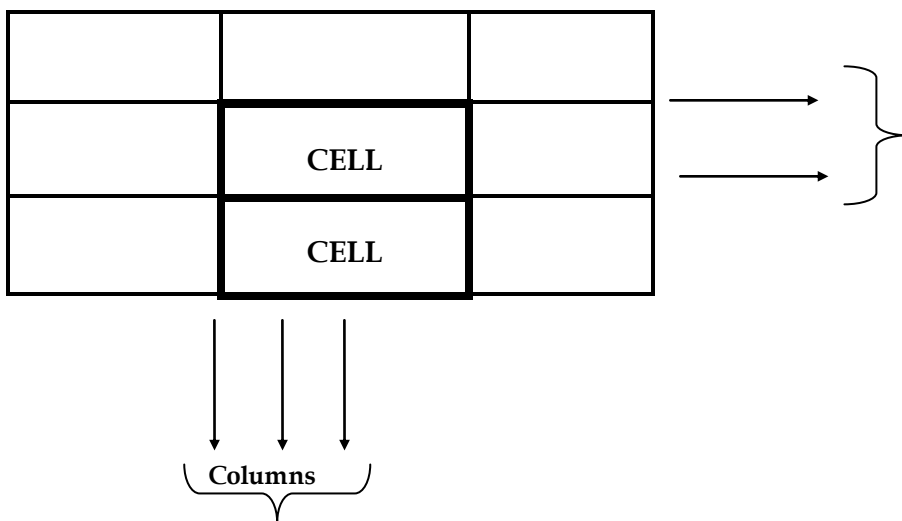
## DATABASE

### 7.1 Database

A database is the place of storage of the data in the form of tables

Data means information which is very useful. A database is also collection of 1 or more tables.

Table 7.1



A cell is an intersection of a row and a column

A column is also called as a field / attribute

A record is also called as a row / tuple.

A table is also called as an entity / relation.

#### **Note:-**

- If we install any of the database related software(s) – we can create our own database, we can create our own tables and we can store the data inside it.
- When we install any database s/w(s) – a part of hard disk will be designated / reserved to perform database related activities
- A database can also contain other database objects like views, indexes, stored procedures, functions, triggers etc, apart from tables.

Some of the database software(s) we have are,

Oracle, SQL Server, DB2, Sybase, Informix, MySQL, MS – Access, Foxbase, FoxPro

Among the above database software – some of them are DBMS and some of them are RDBMS.

The s/w which is widely used today is Oracle. The different versions of Oracle starting from the earliest to the latest are – Oracle 2, Oracle 3, Oracle 4, Oracle 5, Oracle 6, Oracle 7, Oracle 8i, Oracle 9i, Oracle 10g, and the latest to hit the market is Oracle 11g here ‘i’ stands for Internet and ‘g’ stands for Grid / Grid computing.

## **7.2 Relational DBMS**

A database is a means of storing information in such a way that information can be retrieved from it. In simplest terms, a relational database is one that presents information in tables with rows and columns. A table is referred to as a relation in the sense that it is a collection of objects of the same type (rows). Data in a table can be related according to common keys or concepts, and the ability to retrieve related data from a table is the basis for the term relational database. A Database Management System (DBMS) handles the way data is stored, maintained, and retrieved. In the case of a relational database, a Relational Database Management System (RDBMS) performs these tasks. DBMS as used in this book is a general term that includes RDBMS.

### **7.2.1 Integrity Rules**

Relational tables follow certain integrity rules to ensure that the data they contain stay accurate and are always accessible. First, the rows in a relational table should all be distinct. If there are duplicate rows, there can be problems resolving which of two possible selections is the correct one. For most DBMSs, the user can specify that duplicate rows are not allowed, and if that is done, the DBMS will prevent the addition of any rows that duplicate an existing row.

A second integrity rule of the traditional relational model is that column values must not be repeating groups or arrays. A third aspect of data integrity involves the concept of a null value. A database takes care of situations where data may not be available by using a null value to indicate that a value is missing. It does not equate to a

blank or zero. A blank is considered equal to another blank, a zero is equal to another zero, but two null values are not considered equal.

When each row in a table is different, it is possible to use one or more columns to identify a particular row. This unique column or group of columns is called a primary key. Any column that is part of a primary key cannot be null; if it were, the primary key containing it would no longer be a complete identifier. This rule is referred to as entity integrity.

The Employees table illustrates some of these relational database concepts. It has five columns and six rows, with each row representing a different employee.

**Employees Table 7.2**

Employee_Number	First_name	Last_Name	Date_of_Birth	Car_Number
10001	Axel	Washington	28-Aug-43	5
10083	Arvid	Sharma	24-Nov-54	Null
10120	Jonas	Ginsberg	01-Jan-69	Null
10005	Florence	Wojokowski	04-Jul-71	12
10099	Sean	Washington	21-Sep-66	Null
10035	Elizabeth	Yamaguchi	24-Dec-59	Null

The primary key for this table would generally be the employee number because each one is guaranteed to be different. (A number is also more efficient than a string for making comparisons.) It would also be possible to use First\_Name and Last\_Name because the combination of the two also identifies just one row in our sample database. Using the last name alone would not work because there are two employees with the last name of "Washington." In this particular case the first names are all different, so one could conceivably use that column as a primary key, but it is best to avoid using a column where duplicates could occur. If Elizabeth Yamaguchi gets a job at this company and the primary key is First\_Name, the RDBMS will not allow her name to be added (if it has been specified that no duplicates are permitted). Because there is already an Elizabeth in the table, adding a second one would make the primary key useless as a way of identifying just one row. Note that although

using First\_Name and Last\_Name is a unique composite key for this example, it might not be unique in a larger database. Note also that the Employee table assumes that there can be only one car per employee.

### 7.2.2 SELECT Statements

SQL is a language designed to be used with relational databases. There is a set of basic SQL commands that is considered standard and is used by all RDBMSs. For example, all RDBMSs use the SELECT statement.

A SELECT statement, also called a query, is used to get information from a table. It specifies one or more column headings, one or more tables from which to select, and some criteria for selection. The RDBMS returns rows of the column entries that satisfy the stated requirements. A SELECT statement such as the following will fetch the first and last names of employees who have company cars:

```
SELECT First_Name, Last_Name
FROM Employees
WHERE Car_Number IS NOT NULL
```

The result set (the set of rows that satisfy the requirement of not having null in the Car\_Number column) follows. The first name and last name are printed for each row that satisfies the requirement because the SELECT statement (the first line) specifies the columns First\_Name and Last\_Name. The FROM clause (the second line) gives the table from which the columns will be selected.

**Table 7.3**

FIRST_NAME	LAST_NAME
Axel	Washington
Florence	Wojokowski

The following code produces a result set that includes the whole table because it asks for all of the columns in the table Employees with no restrictions (no WHERE clause). Note that SELECT \* means "SELECT all columns."

```
SELECT *  
FROM Employees
```

### 7.2.3 WHERE Clauses

The WHERE clause in a SELECT statement provides the criteria for selecting values. For example, in the following code fragment, values will be selected only if they occur in a row in which the column Last\_Name begins with the string 'Washington'.

```
SELECT First_Name, Last_Name  
FROM Employees  
WHERE Last_Name LIKE 'Washington%'
```

The keyword LIKE is used to compare strings, and it offers the feature that patterns containing wildcards can be used. For example, in the code fragment above, there is a percent sign (%) at the end of 'Washington', which signifies that any value containing the string 'Washington' plus zero or more additional characters will satisfy this selection criterion. So 'Washington' or 'Washingtonian' would be matches, but 'Washing' would not be. The other wildcard used in LIKE clauses is an underbar (\_), which stands for any one character. For example,

```
WHERE Last_Name LIKE 'Ba_man'
```

would match 'Batman', 'Barman', 'Badman', 'Balman', 'Bagman', 'Bamman', and so on.

The code fragment below has a WHERE clause that uses the equal sign (=) to compare numbers. It selects the first and last name of the employee who is assigned car 12.

```
SELECT First_Name, Last_Name  
FROM Employees  
WHERE Car_Number = 12
```

The next code fragment selects the first and last names of employees whose employee number is greater than 10005:

```
SELECT First_Name, Last_Name
FROM Employees
WHERE Employee_Number > 10005
```

WHERE clauses can get rather elaborate, with multiple conditions and, in some DBMSs, nested conditions. This overview will not cover complicated WHERE clauses, but the following code fragment has a WHERE clause with two conditions; this query selects the first and last names of employees whose employee number is less than 10100 and who do not have a company car.

```
SELECT First_Name, Last_Name
FROM Employees
WHERE Employee_Number < 10100 and Car_Number IS NULL
```

A special type of WHERE clause involves a join, which is explained in the next section.

## 7.2.4 Joins

A distinguishing feature of relational databases is that it is possible to get data from more than one table in what is called a join. Suppose that after retrieving the names of employees who have company cars, one wanted to find out who has which car, including the make, model, and year of car. This information is stored in another table, Cars.

Cars **Table 7.4**

Car_Number	Make	Model	Year
5	Honda	Civic DX	1996
12	Toyota	Corolla	1999

There must be one column that appears in both tables in order to relate them to each other. This column, which must be the primary key in one table, is called the foreign key in the other table. In this case, the column that appears in two tables is Car\_Number, which is the primary key for the table Cars and the foreign key in the table Employees. If the 1996 Honda Civic were wrecked and deleted from the Cars table, then Car\_Number 5

would also have to be removed from the Employees table in order to maintain what is called referential integrity. Otherwise, the foreign key column (Car\_Number) in the Employees table would contain an entry that did not refer to anything in Cars. A foreign key must either be null or equal to an existing primary key value of the table to which it refers. This is different from a primary key, which may not be null. There are several null values in the Car\_Number column in the table Employees because it is possible for an employee not to have a company car.

The following code asks for the first and last names of employees who have company cars and for the make, model, and year of those cars. Note that the FROM clause lists both Employees and Cars because the requested data is contained in both tables. Using the table name and a dot (.) before the column name indicates which table contains the column.

```
SELECT Employees.First_Name, Employees.Last_Name,  
       Cars.Make, Cars.Model, Cars.Year  
FROM Employees, Cars  
WHERE Employees.Car_Number = Cars.Car_Number
```

This returns a result set that will look similar to the following: **Table 7.5**

FIRST_NAME	LAST_NAME	MAKE	MODEL	YEAR
Axel	Washington	Honda	Civic DX	1996
Florence	Wojokowski	Toyota	Corolla	1999

### 7.2.5 Common SQL Commands

SQL commands are divided into categories, the two main ones being Data Manipulation Language (DML) commands and Data Definition Language (DDL) commands. DML commands deal with data, either retrieving it or modifying it to keep it up-to-date. DDL commands create or change tables and other database objects such as views and indexes.

A list of the more common DML commands follows:

- **SELECT** — used to query and display data from a database. The **SELECT** statement specifies which columns to include in the result set. The vast majority of the SQL commands used in applications are **SELECT** statements.
- **INSERT** — adds new rows to a table. **INSERT** is used to populate a newly created table or to add a new row (or rows) to an already-existing table.
- **DELETE** — removes a specified row or set of rows from a table
- **UPDATE** — changes an existing value in a column or group of columns in a table

The more common DDL commands follow:

- **CREATE TABLE** — creates a table with the column names the user provides. The user also needs to specify a type for the data in each column. Data types vary from one RDBMS to another, so a user might need to use metadata to establish the data types used by a particular database. **CREATE TABLE** is normally used less often than the data manipulation commands because a table is created only once, whereas adding or deleting rows or changing individual values generally occurs more frequently.
- **DROP TABLE** — deletes all rows and removes the table definition from the database. A JDBC API implementation is required to support the **DROP TABLE** command as specified by SQL92, Transitional Level. However, support for the **CASCADE** and **RESTRICT** options of **DROP TABLE** is optional. In addition, the behavior of **DROP TABLE** is implementation-defined when there are views or integrity constraints defined that reference the table being dropped.
- **ALTER TABLE** — adds or removes a column from a table. It also adds or drops table constraints and alters column attributes

## 7.2.6 Result Sets and Cursors

The rows that satisfy the conditions of a query are called the result set. The number of rows returned in a result set can be zero, one, or many. A user can access the data in a result set one row at a time, and a cursor provides the means to do that. A cursor can be thought of as a pointer into a file that contains the rows of the result set, and that



pointer has the ability to keep track of which row is currently being accessed. A cursor allows a user to process each row of a result set from top to bottom and consequently may be used for iterative processing. Most DBMSs create a cursor automatically when a result set is generated.

Earlier JDBC API versions added new capabilities for a result set's cursor, allowing it to move both forward and backward and also allowing it to move to a specified row or to a row whose position is relative to another row.

### **7.2.7 Transactions**

When one user is accessing data in a database, another user may be accessing the same data at the same time. If, for instance, the first user is updating some columns in a table at the same time the second user is selecting columns from that same table, it is possible for the second user to get partly old data and partly updated data. For this reason, DBMSs use transactions to maintain data in a consistent state (data consistency) while allowing more than one user to access a database at the same time (data concurrency).

A transaction is a set of one or more SQL statements that make up a logical unit of work. A transaction ends with either a commit or a rollback, depending on whether there are any problems with data consistency or data concurrency. The commit statement makes permanent the changes resulting from the SQL statements in the transaction, and the rollback statement undoes all changes resulting from the SQL statements in the transaction.

A lock is a mechanism that prohibits two transactions from manipulating the same data at the same time. For example, a table lock prevents a table from being dropped if there is an uncommitted transaction on that table. In some DBMSs, a table lock also locks all of the rows in a table. A row lock prevents two transactions from modifying the same row, or it prevents one transaction from selecting a row while another transaction is still modifying it.

## 7.2.8 Stored Procedures

A stored procedure is a group of SQL statements that can be called by name. In other words, it is executable code, a mini-program, that performs a particular task that can be invoked the same way one can call a function or method. Traditionally, stored procedures have been written in a DBMS-specific programming language. The latest generation of database products allows stored procedures to be written using the Java programming language and the JDBC API. Stored procedures written in the Java programming language are bytecode portable between DBMSs. Once a stored procedure is written, it can be used and reused because a DBMS that supports stored procedures will, as its name implies, store it in the database.

The following code is an example of how to create a very simple stored procedure using the Java programming language. Note that the stored procedure is just a static Java method that contains normal JDBC code. It accepts two input parameters and uses them to change an employee's car number.

```
import java.sql.*;

public class UpdateCar {

    public static void UpdateCarNum(int carNo, int empNo)
        throws SQLException {

        Connection con = null;
        PreparedStatement pstmt = null;

        try {
            con = DriverManager.getConnection(
                "jdbc:default:connection");

            pstmt = con.prepareStatement(
```

```

        "UPDATE EMPLOYEES " +
        "SET CAR_NUMBER = ? " +
        "WHERE EMPLOYEE_NUMBER = ?");

    pstmt.setInt(1, carNo);
    pstmt.setInt(2, empNo);
    pstmt.executeUpdate();
}
finally {
    if (pstmt != null) pstmt.close();
}
}
}

```

### 7.3 SQL – Structured Query Language

- It is a language to talk to the database / to access the database
- It is a language, whereas SQL server is a database.
- To work on SQL, a DB software (RDBMS) is required.
- SQL is not case sensitive

**There are 3 types of statements in SQL**

- **DDL** – Data Definition Language – the various commands in DDL are :- Create, Drop, Truncate, Alter, Rename
- **DML** – Data Manipulation Language – the various commands in DML are :- Insert, Update, Delete
- **TCL** – Transaction Control Language – the various commands in TCL are :- Rollback, Commit, Savepoint

### 7.4 JDBC Introduction

The JDBC API is a Java API that can access any kind of tabular data, especially data stored in a Relational Database.

JDBC helps you to write Java applications that manage these three programming activities:

1. Connect to a data source, like a database
2. Send queries and update statements to the database
3. Retrieve and process the results received from the database in answer to your query

The following simple code fragment gives a simple example of these three steps:

```
public void connectToAndQueryDatabase(String username, String password) {  
    Connection con = DriverManager.getConnection(  
        "jdbc:myDriver:myDatabase",  
        username,  
        password);  
    Statement stmt = con.createStatement();  
    ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");  
  
    while (rs.next()) {  
        int x = rs.getInt("a");  
        String s = rs.getString("b");  
        float f = rs.getFloat("c");  
    }  
}
```

This short code fragment instantiates a Driver Manager object to connect to a database driver and log into the database, instantiates a Statement object that carries your SQL language query to the database; instantiates a ResultSet object that retrieves the results of your query, and executes a simple while loop, which retrieves and displays those results. It's that simple.

## 7.4.1 JDBC Product Components

JDBC includes four components:

1. **The JDBC API:** The JDBC™ API provides programmatic access to relational data from the Java™ programming language. Using the JDBC API, applications can execute SQL statements, retrieve results, and propagate changes back to an underlying data source. The JDBC API can also interact with multiple data sources in a distributed, heterogeneous environment.

The JDBC API is part of the Java platform, which includes the *Java™ Standard Edition* (Java™ SE ) and the *Java™ Enterprise Edition* (Java™ EE). The JDBC 4.0 API is divided into two packages: `java.sql` and `javax.sql`. Both packages are included in the Java SE and Java EE platforms.

2. **JDBC Driver Manager:** The JDBC DriverManager class defines objects which can connect Java applications to a JDBC driver. DriverManager has traditionally been the backbone of the JDBC architecture. It is quite small and simple.

The Standard Extension packages `javax.naming` and `javax.sql` let you use a DataSource object registered with a *Java Naming and Directory Interface™* (JNDI) naming service to establish a connection with a data source. You can use either connecting mechanism, but using a DataSource object is recommended whenever possible.

3. **JDBC Test Suite:** The JDBC driver test suite helps you to determine that JDBC drivers will run your program. These tests are not comprehensive or exhaustive, but they do exercise many of the important features in the JDBC API.
4. **JDBC-ODBC Bridge:** The Java Software bridge provides JDBC access via ODBC drivers. Note that you need to load ODBC binary code onto each client machine that uses this driver. As a result, the ODBC driver is most appropriate on a corporate network where client installations are not a major problem, or for application server code written in Java in a three-tier architecture.

This Trail uses the first two of these four JDBC components to connect to a database and then build a java program that uses SQL commands to communicate with a test Relational Database. The last two components are used in specialized environments to test web applications, or to communicate with ODBC-aware DBMSs.

### 7.4.2 JDBC Architecture

The JDBC API supports both two-tier and three-tier processing models for database access.

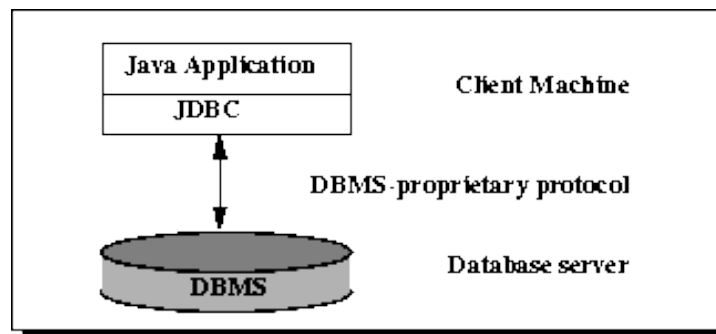


Figure 7.1

In the two-tier model, a Java applet or application talks directly to the data source. This requires a JDBC driver that can communicate with the particular data source being accessed. A user's commands are delivered to the database or other data source, and the results of those statements are sent back to the user. The data source may be located on another machine to which the user is connected via a network. This is referred to as a client/server configuration, with the user's machine as the client, and the machine housing the data source as the server. The network can be an intranet, which, for example, connects employees within a corporation, or it can be the Internet.

In the three-tier model, commands are sent to a "middle tier" of services, which then sends the commands to the data source. The data source processes the commands and sends the results back to the middle tier, which then sends them to the user. MIS directors find the three-tier model very attractive because the middle tier makes it

possible to maintain control over access and the kinds of updates that can be made to corporate data. Another advantage is that it simplifies the deployment of applications. Finally, in many cases, the three-tier architecture can provide performance advantages.

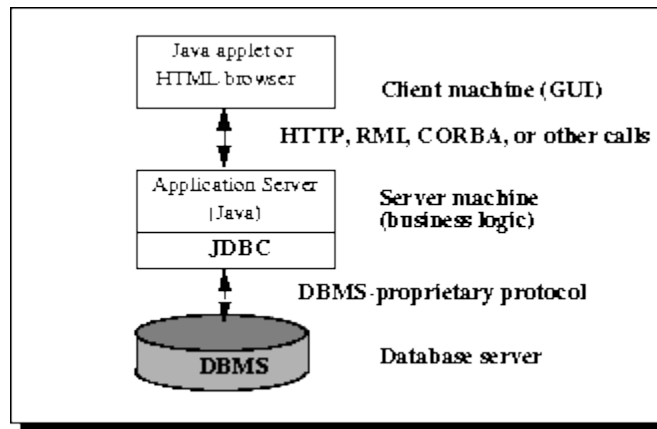


Figure 7.2

Until recently, the middle tier has often been written in languages such as C or C++, which offer fast performance. However, with the introduction of optimizing compilers that translate Java bytecode into efficient machine-specific code and technologies such as Enterprise JavaBeans™, the Java platform is fast becoming the standard platform for middle-tier development. This is a big plus, making it possible to take advantage of Java's robustness, multithreading, and security features.

With enterprises increasingly using the Java programming language for writing server code, the JDBC API is being used more and more in the middle tier of a three-tier architecture. Some of the features that make JDBC a server technology are its support for connection pooling, distributed transactions, and disconnected rowsets. The JDBC API is also what allows access to a data source from a Java middle tier.

# CHAPTER 8

## ALGORITHM DESIGN

### 8.1 Bayes Theorem

The basic formula for describing Bayes Theorem as:

$$\Pr(A | B) = \Pr(B | A) \times \Pr(A) / \Pr(B)$$

As well as a specific example relevant to document classification:

$$\Pr(\text{Category} | \text{Document}) = \Pr(\text{Document} | \text{Category}) \times \Pr(\text{Category}) / \Pr(\text{Document})$$

- $\Pr(A | B)$  = Probability of A happening given that B has already happened
- $\Pr(A)$  = Probability of A happening
- $\Pr(A | B)$ : Conditional probability of A : i.e. probability of A, given that all we know is B

#### 8.1.1 Anti-Spam Filter

Probability is defined as a quantitative measure of uncertainty state of information or event. It has an index which ranges from 0 to 1. It is also approximated through proportion of number of events over the total experiment. If the probability of a state is 0 (zero), we are certain the state will not happen. However if the probability is one the event will surely happen. A probability of 0.5 means we have maximum doubt about the state that will happen.

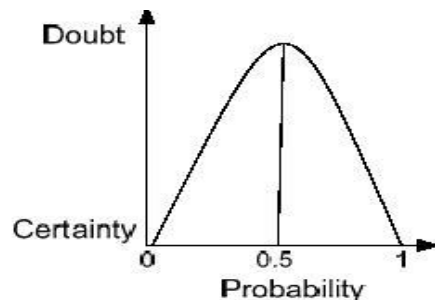


Figure 8.1



The following section describes some of the basic probability formula's that will be used:

Conditional probability: The probability of an event may depend on the occurrence or non-occurrence of another event. This dependency is written in terms of conditional probability  $P(A|B)$

“the probability that A will happen given that B already has” or “the probability to select A among B”

Notice that B is given first, and we find the proportion of A among B

$$P(A | B) = \frac{P(A \cap B)}{P(B)}$$

$$P(B|A) = P(A \cap B) / P(A)$$

$$P(A|B) = P(A \cap B) / P(B)$$

$$P(A \cap B) = P(B|A) P(A) = P(A|B) P(B)$$

Given the above formula': An event A is INDEPENDENT from event B if the conditional probability is the same as the marginal probability

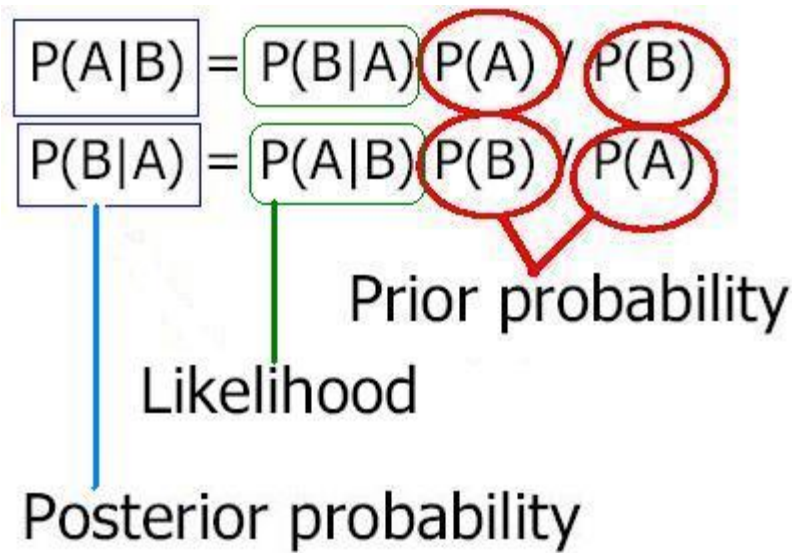
$$P(B|A) = P(B)$$

$$P(A|B) = P(A)$$

From the formulas the Bayes Theorem States the Prior probability. Unconditional probabilities of our hypothesis before we get any data or any NEW evidence. Simply speaking it is the state of our knowledge before the data is observed.

Also stated is the posterior probability: A conditional probability about our hypothesis (our state of knowledge) after we revised based on the new data.

Likelihood is the conditional probability based on our observation data given that our hypothesis holds.



The following are the mathematical formalisms, and the example on a spam filter, but keep in mind the basic idea.

The Bayesian classifier uses the Bayes theorem, Which says:

$$p(c_j | d) = \frac{p(d | c_j) p(c_j)}{p(d)}$$

Considering each attribute and class label as a random variable and given a record with attributes  $(A_1, A_2, \dots, A_n)$ . The goal is to predict class  $C$ . Specifically, we want to find the value of  $C$  that maximizes  $P(C | A_1, A_2, \dots, A_n)$

The approach take is to compute the posterior probability  $P(C | A_1, A_2, \dots, A_n)$  for all values of  $C$  using the Bayes theorem.

$$P(C | A_1 A_2 \dots A_n) = \frac{P(A_1 A_2 \dots A_n | C) P(C)}{P(A_1 A_2 \dots A_n)}$$

So you choose the value of  $C$  that maximizes  $P(C | A_1, A_2, \dots, A_n)$ . This is equivalent to choosing the value of  $C$  that maximizes  $P(A_1, A_2, \dots, A_n | C) P(C)$

So to simplify the task of Naïve Bayesian Classifiers we assume attributes have independent distributions.

The Naïve Bayes theorem has the following characteristics as advantages and disadvantages:

### **8.1.2 Advantages:**

- Handles quantitative and discrete data
- Robust to isolated noise points
- Handle missing values by ignoring the instance
- During probability estimate calculations
- Fast and space efficient
- Not sensitive to irrelevant features
- Quadratic decision boundary

### **8.1.3 Disadvantages:**

- If conditional probability is zero
- Assumes independence of features

Naïve Bayesian prediction requires each conditional probability be non zero. Otherwise, the predicted probability will be zero.

$$P(X | C_i) = \prod_{k=1}^n P(x_k | C_i)$$

In order to overcome this we use probability estimation from one of the following:

$$\text{Original : } P(A_i | C) = \frac{N_{ic}}{N_c}$$

$$\text{Laplace : } P(A_i | C) = \frac{N_{ic} + 1}{N_c + c}$$

$$\text{m - estimate : } P(A_i | C) = \frac{N_{ic} + mp}{N_c + m}$$

c: number of classes

p: prior probability

m: parameter

In order to classify and predict a spam email from a non-spam I will be using the following techniques and assumptions:

- Will be sorting according to language( spam or non-spam), then words, then count
- If a word does not exist, consider to approximate  $P(\text{word}|\text{class})$  using Laplacian
- Will be using the following table for my analysis

The antispam-table.txt file is the file that contains each word that content filtering uses to determine if a message is spam. Beside each word there are three numbers. The first number is an identifier assigned by the anti-spam engine. The second number is the number of times that the word has occurred in non-spam e-mail messages. The third number is the number of times that the word has occurred in spam e-mail messages.

## 8.2 Unicode Encryption Algorithm

We live in modern age where most of time we have to use electronic media . We transfer and receive data by using internet therefore a secure path is essential for this process. The way we use to secure our transmission is to use a key like password to encrypt a message before transmission and for the receiver to decrypt the message using the same shared key .An ingenious solution to this problem is to use a public key cryptosystem which replaces shared key with different for encryption and decryption. One way of breaking the public key cryptosystem is to factorize the public key.

### 8.2.1 Code

```
public String doEncryption(String actualWord) {
    short shiftKey = 7;
    for (int i = 0; i <= (actualWord.length() - 1); i++) {
```

```

    char c = actualWord.charAt(i);
    short k = (short) c;
    short k2 = (short) (k + shiftKey);

    if (k2 > 90) {
        k2 = (short) (k2 - 26);
    }

    char c2 = (char) k2;
    encryptedWord = encryptedWord + c2;
}

return encryptedWord;
}

```

# CHAPTER 9

## SOFTWARE TESTING

Testing is a process, which reveals errors in the program. It is the major quality measure employed during software development. During software development. During testing, the program is executed with a set of test cases and the output of the program for the test cases is evaluated to determine if the program is performing as it is expected to perform.

### 9.1 TESTING IN STRATEGIES

In order to make sure that the system does not have errors, the different levels of testing strategies that are applied at differing phases of software development are:

#### 9.1.1 UNIT TESTING

Unit Testing is done on individual modules as they are completed and become executable. It is confined only to the designer's requirements. Each module can be tested using the following two Strategies .For example JUnit Test.

##### 9.1.1.1 BLACK BOX TESTING

In this strategy some test cases are generated as input conditions that fully execute all functional requirements for the program. This testing has been uses to find errors in the following categories:

- Incorrect or missing functions
- Interface errors
- Errors in data structure or external database access
- Performance errors
- Initialization and termination errors.

In this testing only the output is checked for correctness. The logical flow of the data is not checked.

### **9.1.1.2 WHITE BOX TESTING**

In this the test cases are generated on the logic of each module by drawing flow graphs of that module and logical decisions are tested on all the cases. It has been used to generate the test cases in the following cases:

- Guarantee that all independent paths have been Executed.
- Execute all logical decisions on their true and false Sides.
- Execute all loops at their boundaries and within their operational bounds
- Execute internal data structures to ensure their validity.

### **9.1.2 INTEGRATING TESTING**

Integration testing ensures that software and subsystems work together as a whole. It tests the interface of all the modules to make sure that the modules behave properly when integrated together.

### **9.1.3 SYSTEM TESTING**

It involves in-house testing of the entire system before delivery to the user. Its aim is to satisfy the user the system meets all requirements of the client's specifications.

### **9.1.4 ACCEPTANCE TESTING**

It is a pre-delivery testing in which entire system is tested at client's site on real world data to find errors.

## **9.2 TEST APPROACH**

Testing can be done in two ways:

- Bottom up approach
- Top down approach

### **9.2.1 BOTTOM UP APPROACH**

Testing can be performed starting from smallest and lowest level modules and proceeding one at a time. For each module in bottom up testing a short program executes the module and provides the needed data so that the module is asked to perform the way it will when embedded within the larger system. When bottom level modules are tested attention turns to those on the next level that use the lower level ones they are tested individually and then linked with the previously examined lower level modules.

### **9.2.2 TOP DOWN APPROACH**

This type of testing starts from upper level modules. Since the detailed activities usually performed in the lower level routines are not provided stubs are written. A stub is a module shell called by upper level module and that when reached properly will return a message to the calling module indicating that proper interaction occurred. No attempt is made to verify the correctness of the lower level module.

### **9.3 VALIDATION**

The system has been tested and implemented successfully and thus ensured that all the requirements as listed in the software requirements specification are completely fulfilled. In case of erroneous input corresponding error messages are displayed



# **CHAPTER 10**

## **SYSTEM REQUIREMENTS**

### **10.1 Hardware Requirements**

- **PC / Laptop :: Dual Core or above**
- **Pentium processor :: 2.1 GHZ or above**
- **RAM Capacity :: 1GB**
- **Hard Disk :: 80GB**

### **10.2 Software Requirements**

- **Operating System :: Windows XP/7/8.1**
- **Server Side :: JSP ,JSTL, Servlets**
- **Client Side :: HTML, CSS**
- **HTML Designing :: Dream weaver Tool**
- **Web Server :: Tomcat v7.0**
- **Services :: JDBC**
- **Database :: MYSQL Workbench**
- **Integrated Development Environment :: Eclipse JE**

# CHAPTER 11

## TOOLS

### 11.1 Eclipse JEE:

#### 11.1.1 Package Description:

Eclipse JEE includes the tools for Java developers creating Java EE and Web applications, including a Java IDE, tools for Java EE, JPA, JSF, Mylyn, EGit and others.

#### 11.1.2 Package includes:

- Data Tools Platform
- Eclipse Git Team Provider
- Eclipse Java Development Tools
- Eclipse Java EE Developer Tools
- JavaScript Development Tools
- Maven Integration for Eclipse
- Mylyn Task List
- Eclipse Plug-in Development Environment
- Remote System Explorer
- Eclipse XML Editors and Tools

### 11.2 Java Development Kit:

The **Java Development Kit (JDK)** is an implementation of either one of the Java SE, Java EE or Java ME platforms released by Oracle Corporation in the form of a binary product aimed at Java developers on Solaris, Linux, Mac OS X or Windows. The JDK includes a private JVM and a few other resources to finish the recipe to a Java Application. Since the introduction of the Java platform, it has been by far the most widely used Software Development Kit (SDK). On 17 November 2006, Sun announced that it would be released under the GNU General Public License (GPL), thus making it free software. This happened in large part on 8 May 2007, when Sun contributed the source code to the Open JDK.

### **11.3 MySQL Workbench:**

MySQL Workbench is a unified visual tool for database architects, developers, and DBAs. MySQL Workbench provides data modeling, SQL development, and comprehensive administration tools for server configuration, user administration, backup, and much more. MySQL Workbench is available on Windows, Linux and Mac OS X.

#### **11.3.1 Design:**

MySQL Workbench enables a DBA, developer, or data architect to visually design, model, generate, and manage databases. It includes everything a data modeler needs for creating complex ER models, forward and reverse engineering, and also delivers key features for performing difficult change management and documentation tasks that normally require much time and effort.

#### **11.3.2 Develop:**

MySQL Workbench delivers visual tools for creating, executing, and optimizing SQL queries. The SQL Editor provides color syntax highlighting, auto-complete, reuse of SQL snippets, and execution history of SQL. The Database Connections Panel enables developers to easily manage standard database connections, including MySQL Fabric. The Object Browser provides instant access to database schema and objects.

#### **11.3.3 Administer:**

MySQL Workbench provides a visual console to easily administer MySQL environments and gain better visibility into databases. Developers and DBAs can use the visual tools for configuring servers, administering users, performing backup and recovery, inspecting audit data, and viewing database health.

#### **11.3.4 New! Visual Performance Dashboard:**

MySQL Workbench provides a suite of tools to improve the performance of MySQL applications. DBAs can quickly view key performance indicators using the Performance Dashboard. Performance Reports provide easy identification and access to IO hotspots, high cost SQL statements, and more. Plus, with 1 click, developers can see where to optimize their query with the improved and easy to use Visual Explain Plan.

### 11.3.5 Database Migration:

MySQL Workbench now provides a complete, easy to use solution for migrating Microsoft SQL Server, Microsoft Access, Sybase ASE, PostgreSQL, and other RDBMS tables, objects and data to MySQL. Developers and DBAs can quickly and easily convert existing applications to run on MySQL both on Windows and other platforms. Migration also supports migrating from earlier versions of MySQL to the latest releases.

## 11.4 Apache Tomcat:

Apache Tomcat is an open source software implementation of the Java Servlet and Java Server Pages technologies. The Java Servlet and Java Server Pages specifications are developed under the Java Community Process. Apache Tomcat is developed in an open and participatory environment and released under the Apache License version 2. Apache Tomcat is intended to be a collaboration of the best-of-breed developers from around the world. Apache Tomcat, Tomcat, Apache, the Apache feather, and the Apache Tomcat project logo are trademarks of the Apache Software Foundation.

### 11.4.2 Directories and Files:

Throughout the docs, you'll notice there are numerous references to `$CATALINA_HOME`. This represents the root of your Tomcat installation. When we say, "This information can be found in your `$CATALINA_HOME/README.txt` file" we mean to look at the `README.txt` file at the root of your Tomcat install. Optionally, Tomcat may be configured for multiple instances by defining `$CATALINA_BASE` for each instance. If multiple instances are not configured, `$CATALINA_BASE` is the same as `$CATALINA_HOME`.

These are some of the key tomcat directories:

- `/bin` - Startup, shutdown, and other scripts. The `*.sh` files (for Unix systems) are functional duplicates of the `*.bat` files (for Windows systems). Since the Win32 command-line lacks certain functionality, there are some additional files in here.
- `/conf` - Configuration files and related DTDs. The most important file in here is `server.xml`. It is the main configuration file for the container.

- **/logs** - Log files are here by default.
- **/webapps** - This is where your webapps go.

## **11.5 Adobe Dreamweaver:**

**Adobe Dreamweaver** is a proprietary web development tool developed by Adobe Systems. Dreamweaver was created by Macromedia in 1997, and was maintained by them until Macromedia was acquired by Adobe Systems in 2005. Adobe Dreamweaver is available for OS X and for Windows.

Following Adobe's acquisition of the Macromedia product suite, releases of Dreamweaver subsequent to version 8.0 have been more compliant with W3C standards. Recent versions have improved support for Web technologies such as CSS, JavaScript, and various server-side scripting languages and frameworks including ASP (ASP JavaScript, ASP VBScript, ASP.NET C#, ASP.NET VB), ColdFusion, Scriptlet, and PHP.

# CHAPTER 12

## WEB APPLICATIONS

### 12.1 Hello World Application

This application is a simple Hello World application which will output **Hello World!!** on the browser. It needs a dynamic Web project and a JSP associated with it.

To run this tutorial, as a minimum you will be required to have installed the following prerequisite software:

1. Sun JDK 6.0+ (J2SE 1.6)
2. Eclipse IDE for Java EE Developers, which is platform specific
3. Apache Geronimo Eclipse Plugin 2.1.x
4. Apache Geronimo Server 2.1.x

Geronimo version 2.1.x, Java 1.5 runtime, and Eclipse Ganymede are used in this tutorial but other versions can be used instead (e.g., Geronimo version 2.2, Java 1.6, Eclipse Europa)

Details on installing eclipse are provided in the Development environment section. This tutorial is organized in the following sections:

- Creating a dynamic Web project using Eclipse
- Adding a JSP to the project
- Making `hello.jsp` the welcome file
- Run and deploy
- Creating a dynamic Web project using Eclipse Launch Eclipse and Switch to *Java EE* perspective.

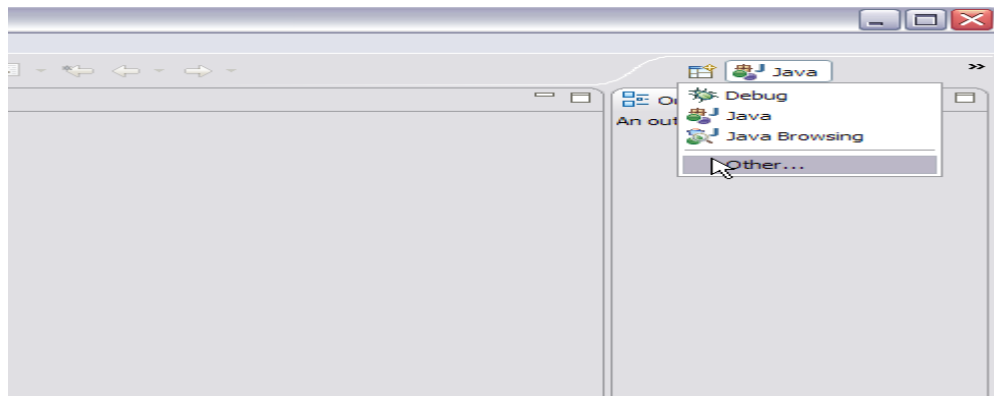


Figure 12.1

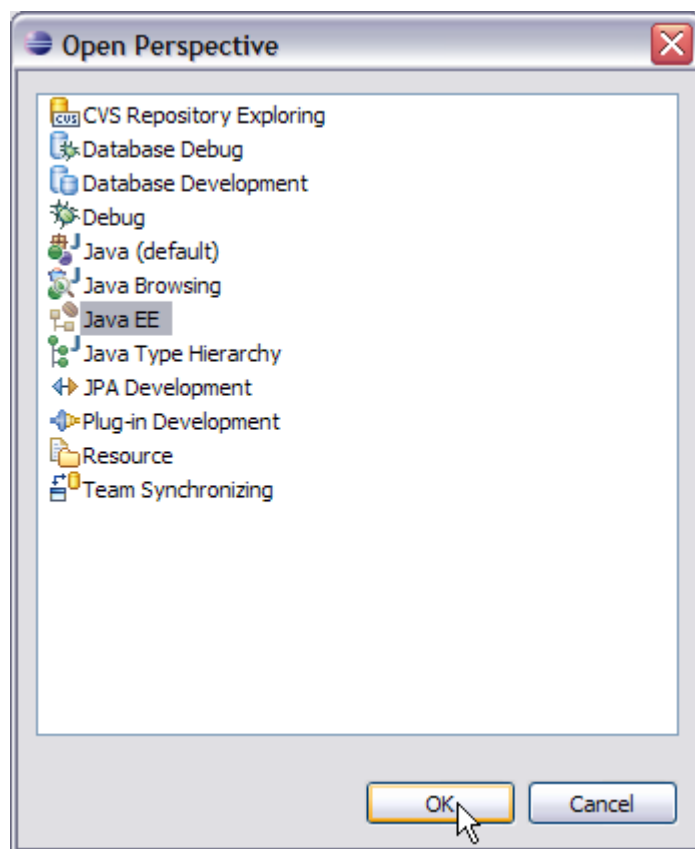


Figure 12.2

1. Right click under the project explorer and select **Dynamic Web Project** as shown in the figure

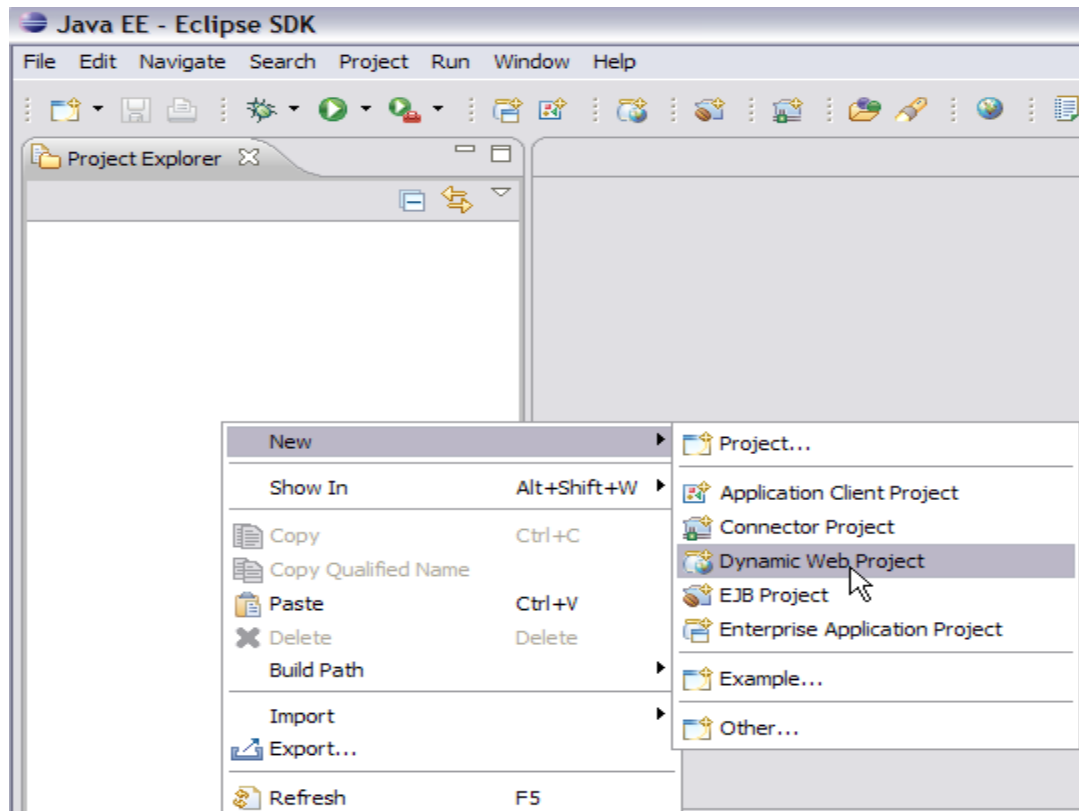


Figure 12.3



2. Name the project as *HelloWorld*.

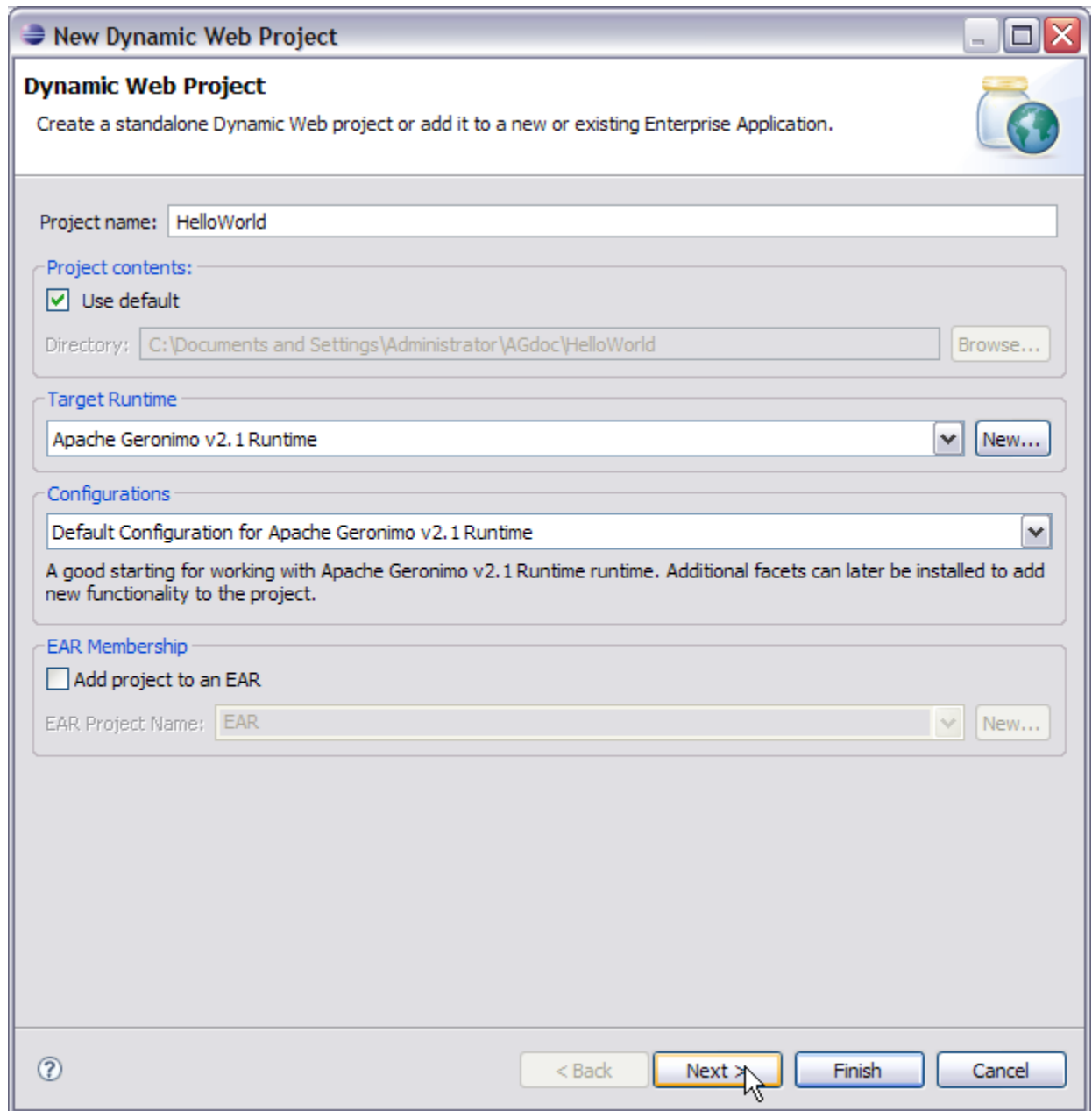


Figure 12.4

3. Keep default values for all the fields and select **Finish**.

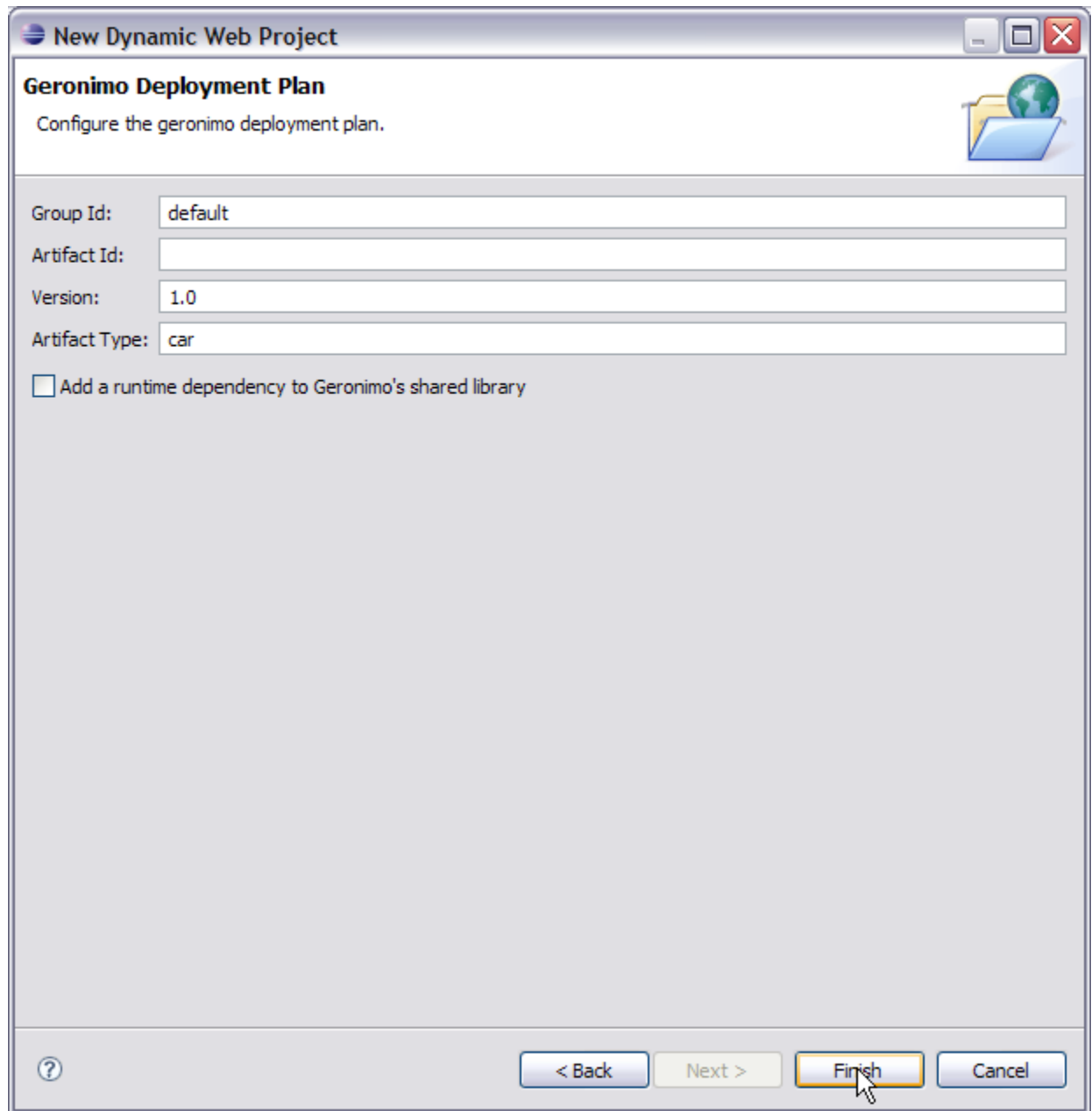


Figure 12.5

*Adding a JSP to the project*

1. Right-click on the project **HelloWorld** and create a new JSP as shown in the figure.

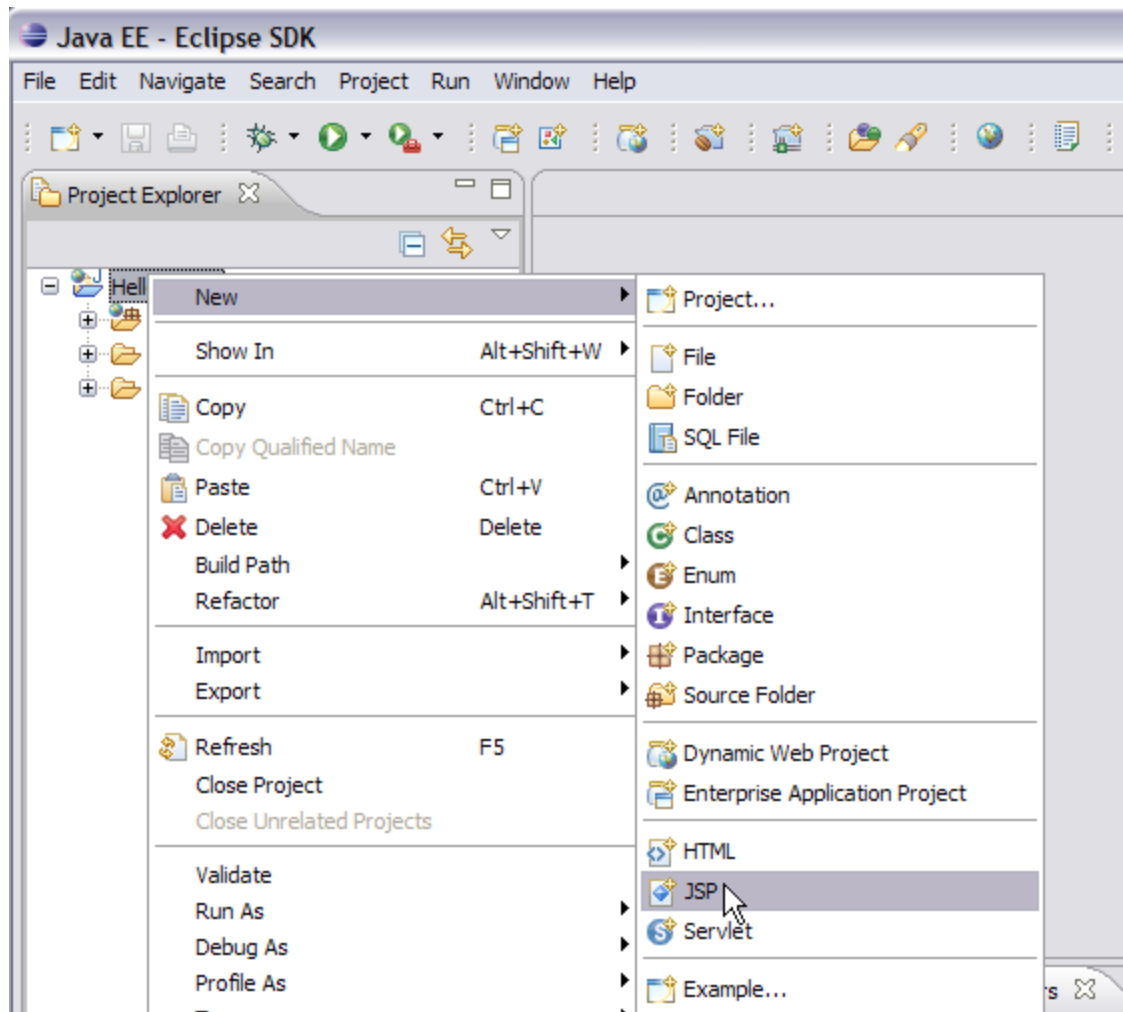


Figure 12.6

2. Give the name as **hello.jsp** and select **Next**. Select **Finish** on the next screen

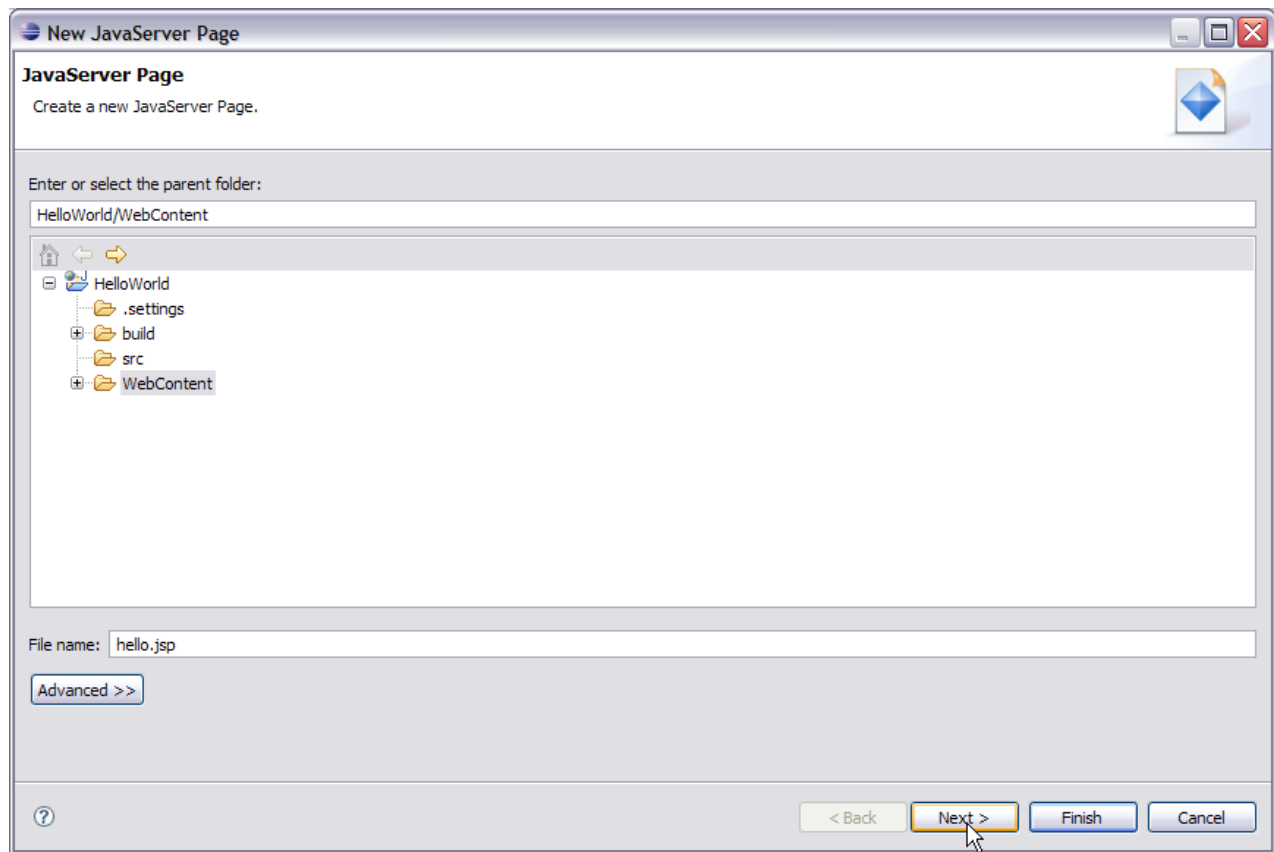


Figure 12.7

3. Modify the code of hello.jsp as follows:

#### **hello.jsp**

```
<% @ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"% >
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" >
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Hello World</title>
</head>
<body>
```

Hello World!!

</body>

</html>

*Making hello.jsp the welcome file*

1. Click **WebContent** -> **WEB-INF** and open web.xml.
2. Add hello.jsp as a welcome file under the <welcome-file-list> tag:

**web.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
    xmlns="http://java.sun.com/xml/ns/javaee"
```

```
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
```

```
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee id="WebApp_ID" version="2.5">
```

```
<display-name>HellowWorld</display-name>
```

```
<welcome-file-list>
```

```
<welcome-file>hello.jsp</welcome-file>
```

```
<welcome-file>index.html</welcome-file>
```

```
<welcome-file>index.htm</welcome-file>
```

```
<welcome-file>index.jsp</welcome-file>
```

```
<welcome-file>default.html</welcome-file>
```

```
<welcome-file>default.htm</welcome-file>
```

```
<welcome-file>default.jsp</welcome-file>
```

```
</welcome-file-list>
```

```
</web-app>
```

*Run and deploy*

1. Deploy the application on the server.

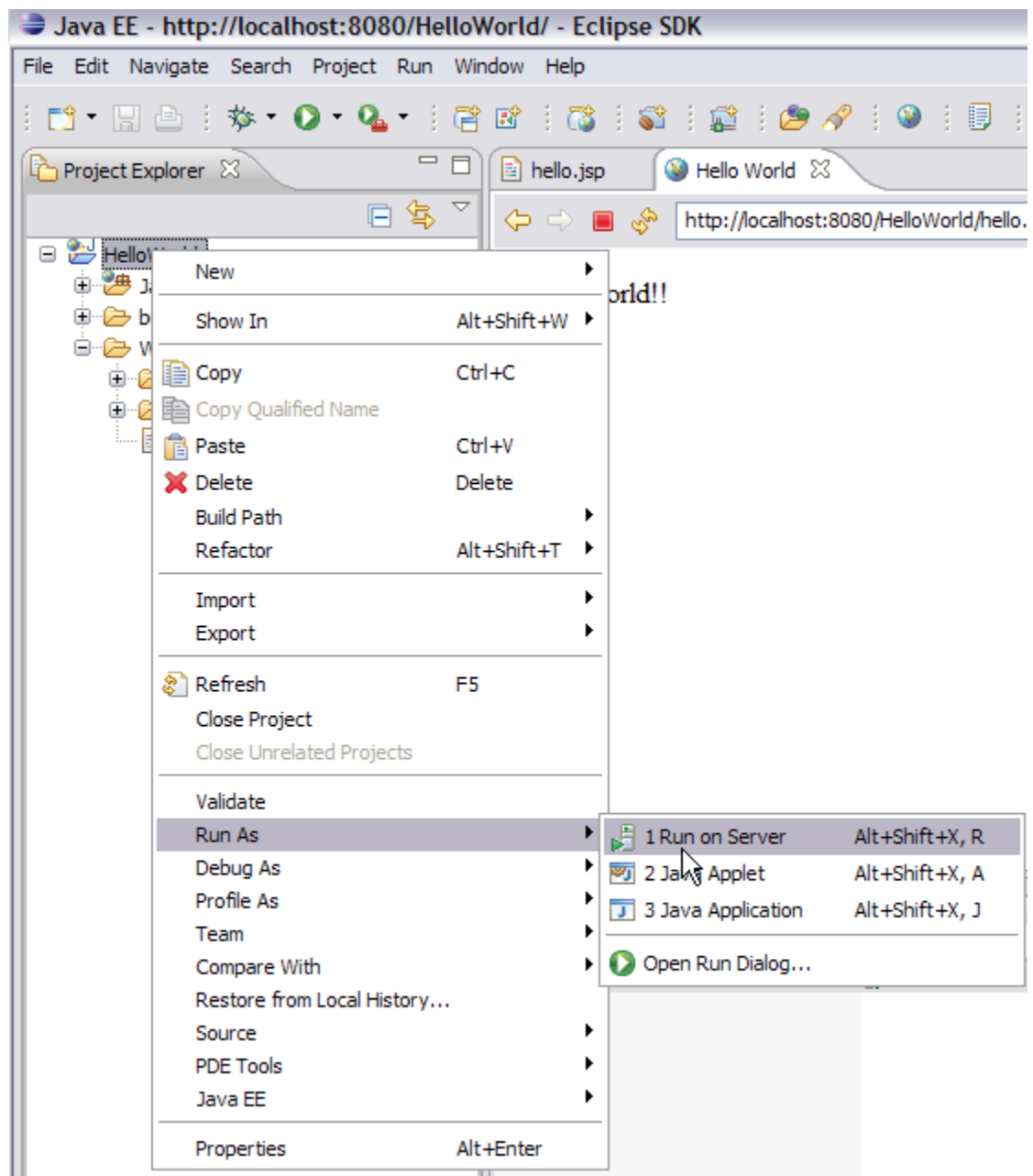


Figure 12.8

2. launch the application using <http://localhost:8080/HelloWorld/hello.jsp>

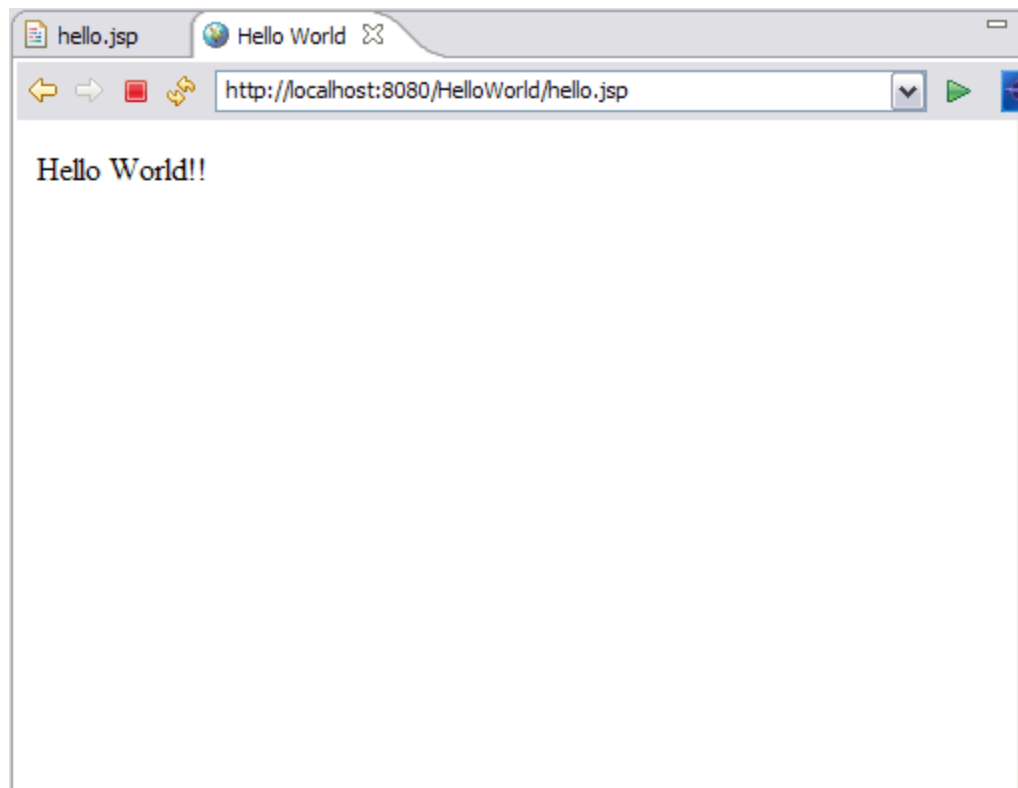


Figure 12.9

## CHAPTER 13

### CONCLUSION & FUTURE ENHANCEMENTS

#### 13.1 Conclusion

Email has been an efficient and popular communication mechanism as the number of Internet user's increase. In many security informatics applications it is important to detect deceptive communication in email. In this application the mails are classified as suspicious or normal using the key words and encrypted keywords. The mails containing these keywords and encrypted keywords are classified as suspicious mails and they can be blocked and verified by the administrator. The proposed work will be helpful for identifying the suspicious email and also assist the investigators to get the information in time to take effective actions to reduce the criminal activities.

#### 13.2 Future Enhancements

Even though the project fulfills the requirements of the present application there is always scope for further work. According to the emerging changes and new versions, further work can be done to improve the application. Since project is designed in a flexible software.

Whenever a new virus will be found on internet it can also be added into your database. This feature will be the future enhancement.

Whenever an attached file is RAR or Archive file then any extension file in the RAR or Archive file cannot be found. So, this task will be for future enhancement.



## **Abbreviations**

**RDX:** Research Department Explosive

**TME:** Targeted Malicious Email

**NTME:** Non-Targeted Malicious Email

**SSL:** Secure Socket Layer

**MVC:** Model View Controller

**JSP:** Java Server Pages

**POJO:** Plain Old Java Object

**HTML:** Hypertext Markup Language

**J2EE:** Java 2 Enterprise Edition

**TDD:** Test Driven Development

**UML:** Unified Modeling Language

**JVM:** Java Virtual Machine

**OS:** Operating System

**API:** Application Programming Interface

**XML:** Extensible Markup Language

**JDK:** Java Development Kit

**GUI:** Graphical User Interface

**JDBC:** Java Database Connectivity

**JNDI:** Java Naming and Directory Interface

**RMI:** Remote Method Invocation

**RMI-IIOP:** Remote Method Invocation over Internet Inter-ORB Protocol Technology

**CGI:** Common Gateway Interface

**HTTP:** Hypertext Transfer Protocol

**URL:** Universal Resource Locator

**JSP-EL:** Java Server Pages- Expression Language

**JSTL:** Java Server Pages Standard Tag Library

**SQL:** Structured Query Language

**POTS:** Plain Old Telephone Service

**PODS:** Plain Old Data Structure

**POD:** Plain Old Documentation  
**PHP:** Hypertext Preprocessor  
**POCO:** Plain Old CLR Object  
**POPO:** Plain Old PHP Object  
**SJO:** Specialized Java Object  
**RDBMS:** Relational Database Management System  
**DML:** Data Manipulation Language  
**DDL:** Data Definition Language  
**TCL:** Transaction Control Language  
**CSS:** Cascading Style-sheet  
**SDK:** Software Development Kit  
**GPL:** General Public License

## REFERENCES

### BOOKS REFERED

#### HTML

- [1] Holzner, HTML Black Book

#### JAVA TECHNOLOGIES

- [2] Larne Pekowsley, JAVA Server Pages
- [3] Nick Todd, JAVA Server Pages
- [4] Scott oaks, JAVA Security
- [5] Shadab siddiqui, J2EE Professional
- [6] Yehuda Shiran, Java Script Programming
- [7] JAVA Complete Reference

#### JDBC

- [8] Patel Moss, JAVA Database Programming with JDBC

#### SOFTWARE ENGINEERING

- [9] Sommerville, Software Engineering.

#### Web Link:

- [1]<http://tomcat.apache.org/> (Apache Tomcat Welcome)
- [2]<http://tomcat.apache.org/tomcat-7.0-doc/introduction.html> (Apache Tomcat Introduction and others)
- [3][http://en.wikipedia.org/wiki/Java\\_Development\\_Kit\\_\(JDK\)](http://en.wikipedia.org/wiki/Java_Development_Kit_(JDK))
- [3]<http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/keplersr2> (Eclipse package)
- [4]<http://www.mysql.com/products/workbench/> (MySQL Workbench)
- [5][http://en.wikipedia.org/wiki/Adobe\\_Dreamweaver](http://en.wikipedia.org/wiki/Adobe_Dreamweaver) (Adobe Dreamweaver)
- [6]<http://geronimo.apache.org/GMOxDOC30/developing-a-hello-world-web-application.html> (Web Application)

- 7- [http://www.w3schools.com/css/css\\_syntax.asp](http://www.w3schools.com/css/css_syntax.asp)
- 8- <http://www.w3schools.com/html/default.asp>
- 9- [www.jsptut.com](http://www.jsptut.com)
- 10- [www.mountangoatsoftware.com](http://www.mountangoatsoftware.com)
- 11- [www.princeton.edu](http://www.princeton.edu)
- 12- [www.cse.iitb.ac.in/servlettutorial](http://www.cse.iitb.ac.in/servlettutorial)
- 13- <http://www.paulgraham.com/spam.html>
- 14- <http://shiffman.net/>
- 15- <http://examples.javacodegeeks.com/core-java>