

MTC'roid ROUTE MAESTRO

A PROJECT REPORT

Submitted by

DIVYA.E (50810205015)

KALAIARASI.S (50810205027)

NIVETHA.E (50810205041)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

ARUNAI COLLEGE OF ENGINEERING

TIRUVANNAMALAI



ANNA UNIVERSITY:: CHENNAI 600 025

APRIL 2014

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report “ **MTC’roid ROUTE MAESTRO** ” is the bonafide work of **DIVYA.E(50810205015), KALAIARASI.S (50810205027), NIVETHA.E (50810205041)** who carried out the project work under my supervision.

SIGNATURE

Mr.N.ANANDA KUMAR,M.E.,
Associate Professor

HEAD OF THE DEPARTMENT

Dept of Information Technology,
Arunai College of Engineering,
Velu Nagar, Mathur,
Tiruvannamalai-606 603.

SIGNATURE

Mr.S.SUNDARAPANDIYAN, M.Tech.,
Assistant Professor

SUPERVISOR

Dept of Information Technology,
Arunai College of Engineering,
Velu Nagar, Mathur,
Tiruvannamalai-606 603.

Submitted for VIVA-VOCE held on at Arunai College of Engineering,
Tiruvannamalai.

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We are extremely thankful to our Management for having included us to annex the golden opportunity of studying this course in this college. We wish to thank our Chairman **Mr.E.V.Velu, MA., M.L.A.**, Vice-Chairman **Mr.E.V.KUMARAN,ME.**, and Managing Director **Dr.E.V.Kamban,M.D.**, for their innovative suggestion and encouragement.

We wish to thank our Secretary **Mr.R.Selvaraj** and **Mr.D.Dhandapani, Ph.D.**, Principal of being a source of inspiration. We extend our heartfelt and genuine gratitude to our Dean **Dr.S.Selvakumar Raja, Ph.D.**, and our Head of Department **Mr.N.Ananda Kumar, M.E.**, and Guide **Mr.S.Sundarapandiyan M.Tech.**, for their valuable suggestion throughout this project.

We express our heartiest gratitude to all faculty members of **IT Department** and other Staffs of our college for providing their valuable suggestion at different stage of the project.

Finally, We express our thanks to our **Parents** and **Friends** for their prayers, enthusiasm and endless support without which this project wouldn't have been completed.

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	ABSTRACT	i
	LIST OF FIGURES	ii
	LIST OF ABBREVIATIONS	iii
1.	INTRODUCTION	1
2.	LITERATURE SURVEY	2
3.	SYSTEM DESCRIPTION	5
	3.1 Existing System	5
	3.1.1 Disadvantages	6
	3.2 Proposed System	6
	3.2.1 Advantages	7
	3.3 Software Requirements	7
	3.4 Hardware Requirements	7
4.	SYSTEM DESIGN	8
	4.1 Architecture Diagram	8
	4.2 ER Diagram	9
	4.3 Data Flow Diagram	10
	4.3.1 DFD Level 0	
	4.3.2 DFD Level 1	
	4.3.3 DFD Level 2	

5.	PROJECT DESCRIPTION	11
	5.1Module Description	11
	5.1.1 Data collection	11
	5.1.2 PHP Server	11
	5.1.3 Route Management Module	11
	5.1.4 Fare Calculation	11
	5.1.5 Notification	11
6.	UML DIAGRAM	12
	6.1 Use case	12
	6.2 Sequence Diagram	13
	6.3 Class Diagram	14
7.	SYSTEM TESTING	15
	7.1 Testing Objectives	15
	7.2 Developing Methodologies	16
	7.3 Testing Technique Tool Selection process	16
	7.4 Unit Testing	17
	7.4.1 Functional Testing	18
	7.4.2 Performance Testing	18
	7.4.3 Structured Testing	18
	7.4.4 Stress Testing	19
	7.5 Integration Testing	19
	7.6 Acceptance Testing	19
	7.7 Validation Testing	20
	7.8 Black Box Testing	20

8.	SOFTWARE DESCRIPTION	21
9.	CONCLUSION	44
10.	FUTURE ENHANCEMENT	45
	APPENDICES	
	A-Sample coding	46
	B- Screen shots	54
	BIBLIOGRAPHY	61

ABSTRACT

The advancement in the android mobile devices, wireless and web technologies given rise to the new application that will make the user feel comfortable with their travelling. Travelling is a large growing business in various countries also. It becomes very difficult to keep bus services records and other information. This application shows the bus services with fare for different category of buses and also the route for that bus to the passenger, we can install this application in mobile. Our project will be useful in the knowing the route of the services and also to analyze the fare between the intermediate places. This project main aim is to satisfy a passenger to view the route and the services and also the possible buses and nearby famous places around the destination places and also calculate the fare between the intermediate station.

LIST OF FIGURES

FIGURE NO	TITLE	PAGE NO
4.1	Architecture Diagram	8
4.2	ER Diagram	9
4.3	Data Flow Diagram	10
6.1	Use Case Diagram	12
6.2	Sequence Diagram	13
6.3	Class Diagram	14

LIST OF ABBREVIATIONS

PHP	-	Personal Home page
GPS	-	Global Position System
GNSS	-	Global Navigation Satellite System

INTRODUCTION

CHAPTER 1

INTRODUCTION

Metropolitan Transport Corporation (MTC) is the governmental agency that operates public bus services in and around the city of Chennai with around 3500 buses covering 850 routes. In 1947, 30 buses introduced madras state transport department. The Metropolitan Transport Corporation (formerly known as Pallavan Transport Corporation) sometimes known as the MTC, is the agency that operates the public transport bus service in Chennai, India. It operates in a area of 3,929 square kilometer. It contains 25 depots, 771 routes, 23,519 employees, passenger/day is 5 million. MTC provides informations such as administration, fares, long term process, bus routes with 'stage stops',tenders and complaint redressal.

The information that was ultimately obtained from the MTC was an updated list of fares,a list of termini and the numbers of bus registered to each bus[terminus] and of all the buses with their starting and ending points.This helped in creating the final database and also assisted in data collection.

LITERATURE SURVEY

CHAPTER 2

LITERATURE SURVEY

2.1 Title	ES Based Automatic ticket vending machine for modern transport system
Author	M.Bhuvaneswari,S.Sukhumar

Description

Ticket friend solution through automated machine enables the passenger to predetermine the transport details. In this automated system we replace the traditional ticket system by smart card that contains all details of the user including bank account information. which is similar to the atm card. This automatic ticket vending machine consists of display which shows the availability of buses for all destinations. The person can find out the destination place by pressing the buttons available on that machine with the help of zig bee. If the location is selected then the availability of buses along with the time is displayed.

If the people confirm to go in certain bus, by using smart card the person can receive the tickets employing RFID technique and by showing the ticket in front of the bus the door opens automatically and after some predetermined seconds it gets closed. GPS is placed Inside the bus and the display shows the route map. By using this we can minimize manpower in buses and ticket counters, predetermining of the bus can be done to find the destination exactly, safe journey can be assured without any disturbance and system based booking for easy usage. Voice talking GPS proposed in the transport make the passenger to identify their departing location.

2.2 Title GPS and GPRS Based cost Effective Human Tracking System Using Mobile Phone.

Author Ruchika Gupta and BVR Reddy

Description

This cost effective method of tracking a human's mobility using two technologies viz General Packet Radio Service (GPRS) and Global Positioning System (GPS). The whole system allows the user's mobility to be tracked using a mobile phone which is equipped with an internal GPS receiver and a GPRS transmitter. A mobile phone application has been developed and deployed on an Android Phone whose responsibility is to track the GPS location and send it to a remote location by creating a GPRS packet. As unique identifier we have used mobile's International Mobile Equipment Identity (IMEI) number which will be sent along with the coordinates.

The person's position is further saved in a Mobile Object Database (MOD) for live tracking which is created in MySQL. From MOD the data will be first transferred into an XML file which will be fed as an input to a web application which is developed with JavaScript Ajax based Google Map API integrated into it which will be responsible for the showing the current location of the mobile phone. Most of the applications developed so far use a handheld GPS receiver device for tracking the location, but we have reduced the cost of device by using the mobile phone which has an inbuilt GPS receiver. And further the cost is reduced by using GPRS rather than using Short Message Service (SMS) for communicating the information to the server.

2.3 Title Electronic Ticketing machine

Author keith E.Mayes.

Description

Electronic Ticketing Machine (ETM) is a Hand Held Computer, in which the program is stored along with all the relevant data, for issue of tickets in the bus during the journey. The storage of program and data is done through a Personal Computer (Host PC), and is called pre-journey configuration for the specific route, bus service, fare, concessions, conductor, driver and day of operation, for handling the passenger and luggage ticketing. The master data pertaining to the unit fare charges for adult/child and luggage, concessions, route, bus service are stored on the Host PC, and made available for uploading on to the ETM at the time of configuration. Once the pre-configuration is over the ETM is ready for use during the journey. A collection report can be prepared at the end of trip. After a journey is over the transaction data files are transferred from ETM to the Host PC, over a communication cable connected to serial communication port. The ETM requires about three hours for full charging after which it can be used continuously for 7-8 hours.

SYSTEM DESCRIPTION

CHAPTER 3

SYSTEM DESCRIPTION

3.1 Existing system

In the existing system all the jobs of the bus route management is done manually. This is very difficult to the operators who want to handle hundreds of trips and many buses in a day. The current system is that an operator wants to keep the Electronic Ticketing Machine of the bus route in MTC office and a separate record for the passengers. So many complaints against staff can arise from the passenger's side. More over there is no detailed record of the bus and routes.

Chennai route check

In this application, user can get the route only by giving the source and destination.

- It doesn't connect with google map
- It works both on online and also in offline
- It doesn't shows all the routes
- It doesn't provides detail information about the bus.

MTC Bus

In this application, user can get the route and intermediate places.

- It connects with google map.
- It doesn't shows all the routes.
- It doesn't provides detail information about the bus.
- In existing system there is no features of fare calculations.
- Internet connection is needed in the existing system.
- It doesn't identify the famous places.
- It will not provide the notifications for destination.

3.1.1Disadvantage

In existing application there is no fare and luggage calculation for a bus, Notification, famous place of the destination and no tracking of the location.

3.2 Proposed system

The proposed system is very useful for the passengers. This avoids the overheads for the mobile operators. They can minimize the working stress and can keep essential documents related to the bus and the route as a softcopy. The advantage of the proposed system is the reduction in the cost of the office equipments(luggage) and the transaction is done quickly. In this application user can calculate different categories of bus fare calculations and destination favorite place. This project aims to use both GPS location and google maps in the development of an android application for the Metropolitan Transport corporation (MTC) chennai.

STAGE FARE[VALUES IN RUPEES]

stage	fare	stage	fare	stage	fare	stage	fare	stage	fare	stage	fare
1.	3.00	6.	6.00	11.	8.00	16.	9.00	21.	11.0	26.	13.0
2.	4.00	7.	6.00	12.	8.00	17.	9.00	22.	11.0	27.	14.0
3.	5.00	8.	7.00	13.	9.00	18.	10.0	23.	12.0	28.	14.0
4.	5.00	9.	7.00	14.	9.00	19.	10.0	24.	12.0		
5.	6.00	10.	8.00	15.	9.00	20.	10.0	25.	13.0		

Express =ordinary fare*1.5+(0.50paisa)

Deluxe=ordinary fare*2+(1Rs)

Night service=double the ordinary fare

VolvoA/c=2.5times the deluxe fare(min 15RS)

3.2.1 Advantage

- User friendly interface
- Fast access to database
- More Storage Capacity
- Search facility
- Look and Feel Environment

3.3 Software requirements

- Operating system : Windows XP
- Technology Used : Android 2.2
- IDE : Eclipse 3.4 (min)
- Emulators : Micro emulator 5055
- Plug-in : ADT plug-in
- Tools used : Android SDK1.2,GoogleAPIv8 or minv7

3.4 Hardware requirements

- Processor : Pentium P4
- Motherboard : Genuine Intel
- RAM : Min 1 GB
- Hard Disk : 80 GB

SYSTEM DESIGN

CHAPTER 4

SYSTEM DESIGN

4.1 ARCHITECTURE DIAGRAM

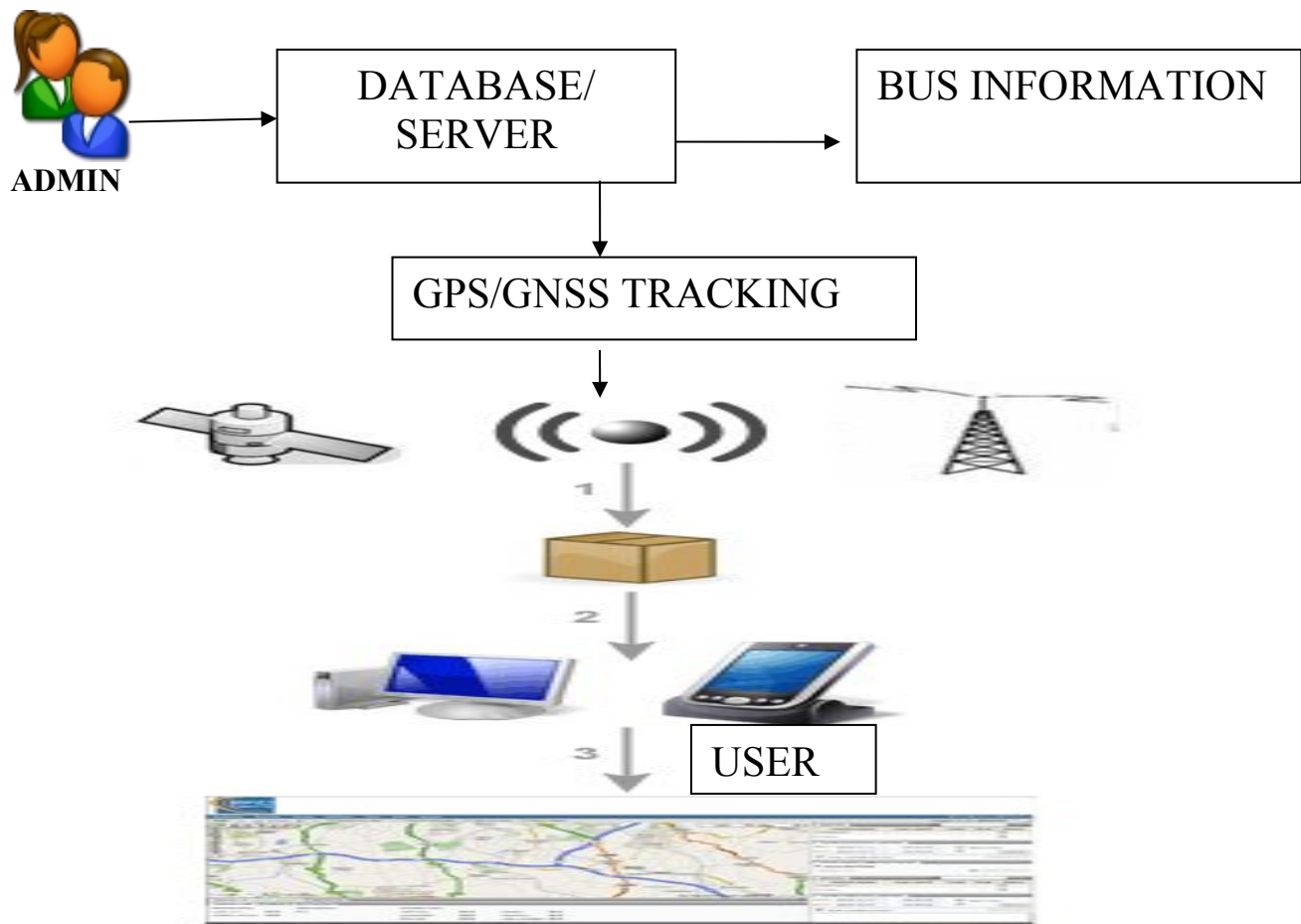


Fig 4.1- Architecture diagram

4.2 ER DIAGRAM

Entity Relationship contains attributes, entity and how they are related.

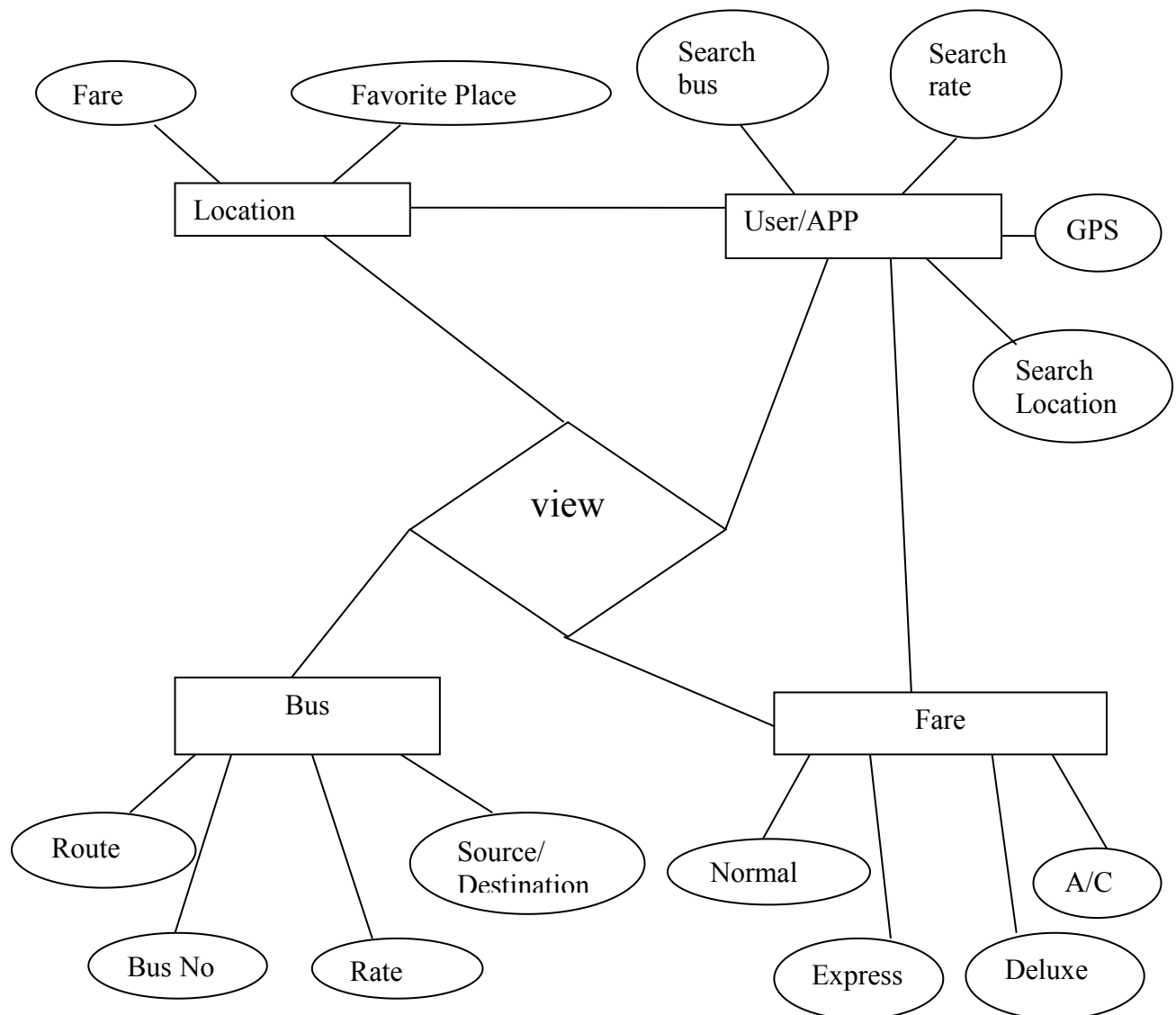
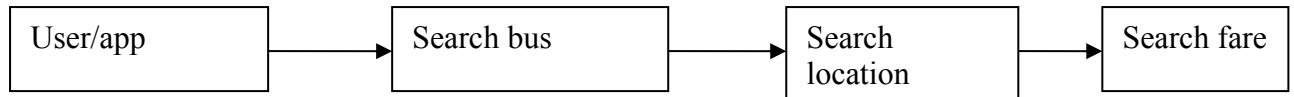


Fig 4.2- ER diagram

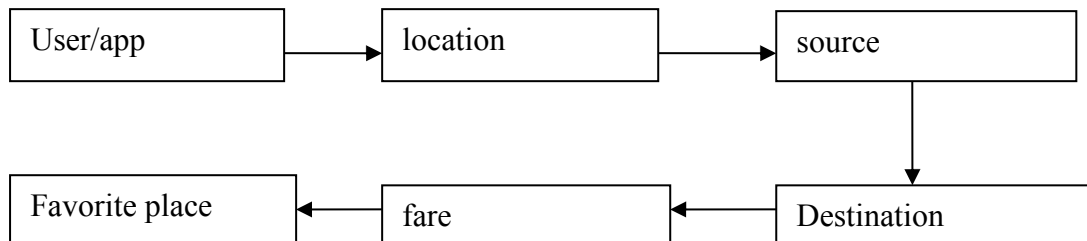
4.3 Data Flow Diagram

Data flow diagram is a graphical representation of the flow of data through information system.

Level 0:



Level 1:



Level 2:

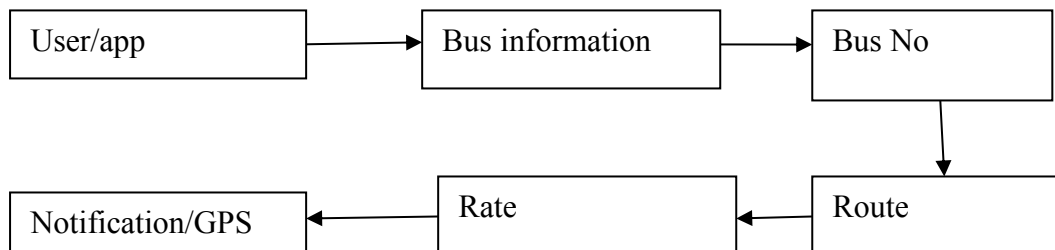


Fig 4.3- Data flow diagram

PROJECT DESCRIPTION

CHAPTER 5

PROJECT DESCRIPTION

5.1 Module Description

5.1.1 Data Collection

This process is to extract information from the data set and transform it into an understandable structure for further use.

5.1.2 PHP Server

This server retrieves the data's from the database management system.

5.1.3 Route Management Module

In this module, if any routes are changed in the MTC Chennai bus service it will updates in route management module.

5.1.4 Fare calculation

In these modules the user can see the type of buses which run between the source and destination places and also to see the fare for both type buses and also to see the luggage fare for that bus in our own app.

5.1.5 Notification

GPS tracker

LBS

Alarm

UML DIAGRAM

CHAPTER 6

UML DIAGRAM

6.1 USE CASE DIAGRAM

A Use case diagram main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

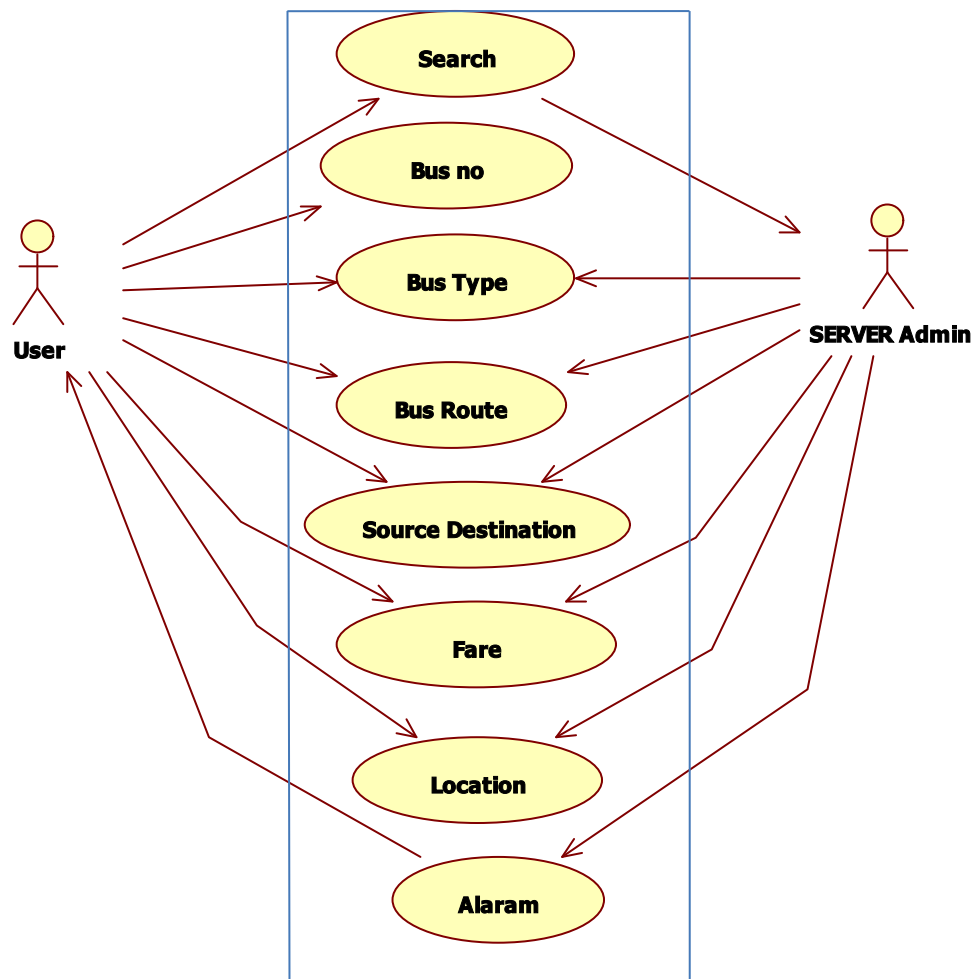


Fig 6.1- Use case diagram

6.2 SEQUENCE DIAGRAM

Sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one and in what order. Sequence diagram are sometimes called Event-trace diagram, event scenarios, and timing diagrams.

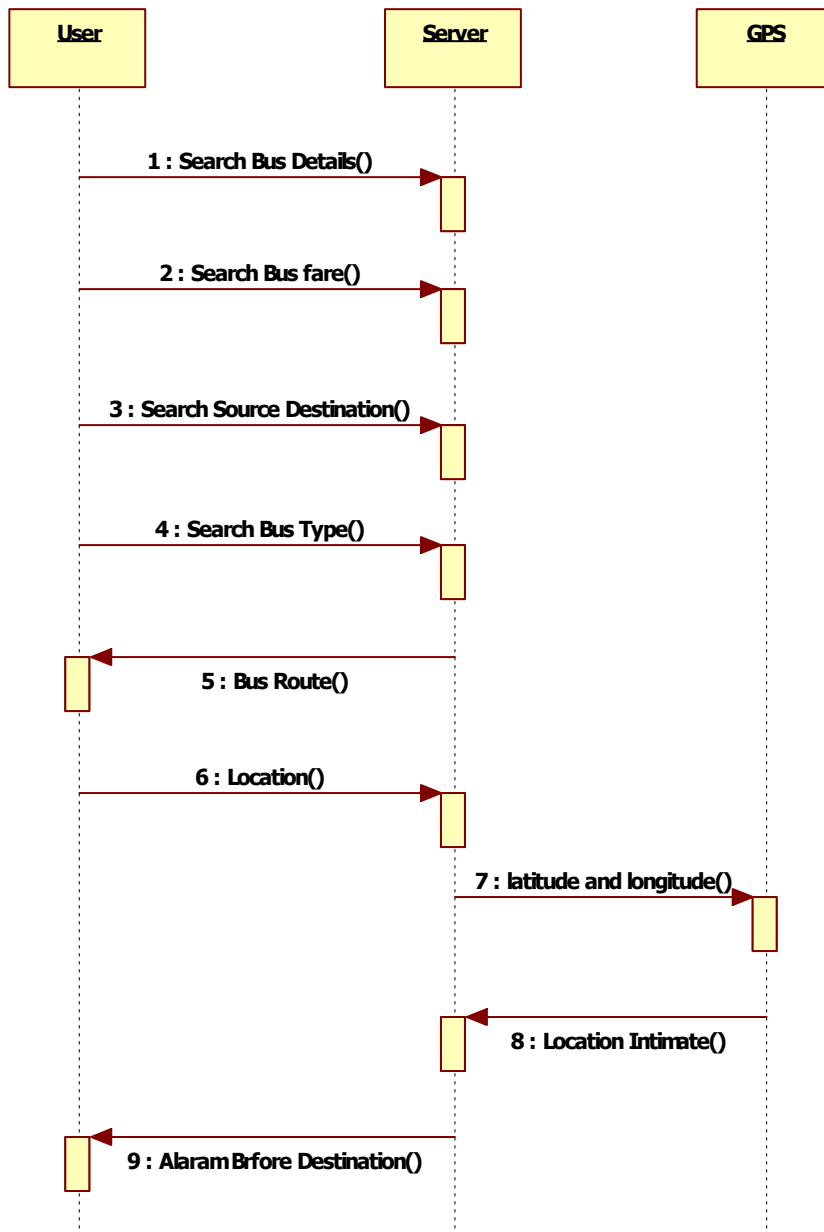


Fig 6.2- Sequence diagram

6.3 CLASS DIAGRAM

It contain class name, attributes, operations and some other components. Class name and Object names are same. Class diagram show the classes of the system, their interrelationships including inheritance, aggregation, and association.

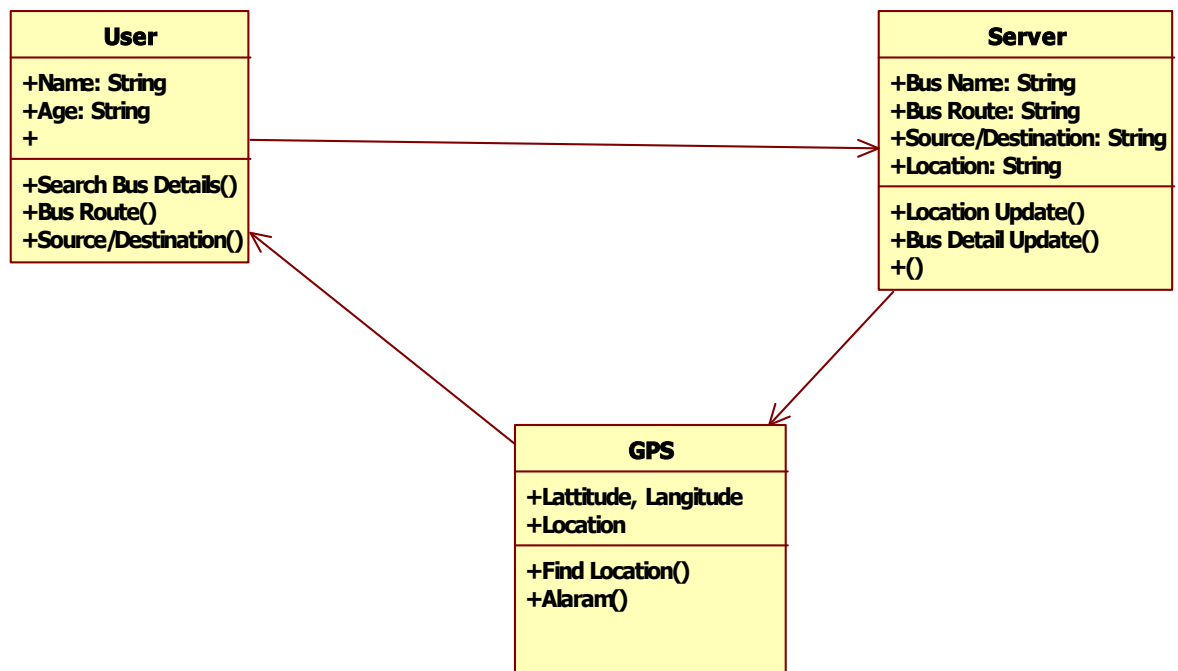


Fig 6.3- Class diagram

SYSTEM TESTING

CHAPTER 7

SYSTEM TESTING

7.1 TESTING OBJECTIVES

Testing is a set of activities that can be planned in advance and conducted systematically. For this reason a template for software testing, a set of steps into which we can place specific test case design techniques and testing methods should be defined for software process. Testing often accounts for more effort than any other software engineering activity. If it is conducted haphazardly, time is wasted, unnecessary effort is expended, and even worse, errors sneak through undetected. It would therefore seem reasonable to establish a systematic strategy for testing software

Type of Testing

There are two type of testing according their behaviors

1. Unconventional Testing
2. Conventional Testing

Unconventional Testing

Unconventional testing is a process of verification which is doing by SQA (Software Quality Assurance) team. It is a prevention technique which is performing from begging to ending of the project development. In this process SQA team verifies project development activities and insuring that developing project is fulfilling the requirement of the client or not. In this testing the SQA team follows these methods:

- | | |
|------------------------|--------------------------|
| 1. Peer review | 3. Inspection |
| 2. Code walk and throw | 4. Document Verification |

Conventional Testing

Conventional Testing is a process of finding the bugs and validating the project. Testing team involves in this testing process and validating that developed project is according to client requirement or not. This process is a correction technique where testing team find bugs and reporting to the development team for correction on developed project built.

7.2 Developing Methodologies

Testing is vital to the success of the system. System testing makes a logical assumption that if all the parts of the system are correct, the goal will be successfully achieved. It is the major quality measure used to determine the status and usefulness of the system. Its basic function is to find the error in the software by examine all possible loopholes. The goal of testing is to point out uncovered requirements, design Or coding errors or invalid acceptance or storage of data. During system testing, the system is used to experimentally to ensure that The software does not run according to its specification and in the way users expect. Special test data is the input for processing ,and the results examined. The process of testing has been divided into three distinctive stages. Different tests are performed at different levels.

7.3 Testing Technique Tool Selection Process

The most important phase in system development life cycle is system testing. The number and nature of errors in a newly designed system depends on the system specifications and the time frame given for the design.

A newly designed system should have all the subsystems working together, but in reality each subsystems work independently. During this phase, all the subsystems are gathered into one pool and tested to determine whether it meets the user requirements.

Testing is done at two level -Testing of individual modules and testing the entire system. During the system testing, the system is used experimentally to ensure that the software will run according to the specifications and in the way the user expects.

Testing plays a very critical role in determining the reliability and efficiency of software and hence is a very important stage in software development. Software testing is done at different levels. They are the unit testing and system testing which comprises of integration testing and acceptance testing.

7.4 UNIT TESTING

This is the first level of testing. The different modules are tested against the specifications produced during the integration. This is done to test the internal logic of each module. Those resulting from the interaction between modules are initially avoided. The input received and output generated is also tested to see whether it falls in the expected range of values. Unit testing is performed from the bottom up, starting with the smallest and lowest modules and proceeding one at a time.

The units in a system are the modules and routines that are assembled and integrated to perform a specific function. The programs are tested for correctness of logic applied and detection of errors in coding. Each of the modules was tested and errors are rectified. They were then found to function properly.

The testing plays an important role in the development of the project. The unit testing comprises the set of tests performed by an individual programmer. The situation is illustrated as follows

Coding → Debugging → Unit testing → Integration testing

The program unit is usually small enough that the programmer who developed it can test it in great details when the unit is integrated into evolving software product. This project is divided into various modules and unit testing has been done.

There are 4 categories in unit testing

- | | |
|------------------------|-----------------------|
| a) Functional testing | c) Stress testing |
| b) Performance testing | d) Structured testing |

7.4.1 Functional Testing

Functional testing cases involves executing the code with normal input values for which the expected result are known as well as boundary values as modules of X Matrices.

7.4.2 Performance Testing

It determines the amount of execution time spent in various parts of the units. The certain amount of performance testing is done during unit testing.

7.4.3 Structure Testing

Structured test are concerned with exercising the internal logic of the program and traversing particular execution parts. Some author refers “Black box testing” as stress test and “white box testing or Glass box testing” as structured testing. The unit testing like functional testing, performance testing, stress testing and structured testing has done all the modules of the system. Unit testing has the goal of discovered error in the individual modules of the system. Unit testing should be exhaustive, as possible to ensure that each representative case handled by each module has been tested. Unit testing is eased by the System structure, which is composed of small loosely counted modules.

7.4.4 Stress Testing

Stress testing is the test designed to intentionally break the system as a unit into many small parts or units.

7.5 Integration Testing

BOTTOM- UP

Bottom-up is the traditional strategy used to integrate the components of software system into a functioning whole? It consists of unit testing followed by the testing of entire system. Disadvantage of the bottom-up testing include the necessity to write and debug test harness fir the modules and level of the complexity that results into larger and larger units. The extreme case of complexity result when each module is unit tested in isolated and all modules are linked and executed in one single integration run. This is the “Big-Bank” approach to integration testing. The disadvantage is the difficulty of isolating errors.

TOP-DOWN

Top-down integration starts with the main routine and one or two immediately subordinate routines in the system structure. The top-level skeleton has been thoroughly tested. It becomes the test harness for its immediately subordinate routines.

Top-down integration requires the use of the program stubs to simulate the effect of lower level routines that are called by those being tested. While integrating the system as a whole, the bottom-up and top-down testing has been done and no error was found.

7.6 Acceptance Testing

Acceptance testing is the test, where the actual client will test the program. This is the final phase, before releasing the program.

7.7 Validation Testing

Validation Testing is occurred after the Integration Testing. The Validation Testing may be defined in many ways, but a simple definition is that the Validation succeeds when software functions in a manner that can be reasonably expected by the Jobseekers, Companies, etc.

7.8 BLACK BOX TESTING

Black-box testing focuses on the functional requirements of the software. Black-Box testing attempts to find errors in the following categories: incorrect or missing functions, interface errors, errors in data structures or database access, performance errors, and initialization and termination errors. Black-box testing on the other hand broadens the focus and might be called testing in the large.

Black-box testing are designed to validate functional requirements without regard to the internal working of the program. Black-box techniques focus on the information domain of the software, deriving test case by partitioning input and output in a manner that provides through test coverage. The requirements for higher quality software demands more systematic approach to testing. The specifications states what the program should do, and how it should perform under various conditions are examined. The multilane function has undergone the black-box testing.

SOFTWARE DESCRIPTION

CHAPERT 8

SOFTWARE DESCRIPTION

Introduction to Java's Architecture

At the heart of Java technology lies the Java virtual machine--the abstract computer on which all Java programs run. Although the name "Java" is generally used to refer to the Java programming language, there is more to Java than the language. The Java virtual machine, Java API, and Java class file work together with the language to make Java programs run. Java's Architecture show how the Java virtual machine fits into the big picture. They show how the virtual machine relates to the other components of Java's architecture: the class file, API, and language. They describe the motivation behind--and the implications of--the overall design of Java technology. This chapter gives an introduction to Java as a technology. It gives an overview of Java's architecture, discusses why Java is important, and looks at Java's pros and cons.

Why Java?

Over the ages people have used tools to help them accomplish tasks, but lately their tools have been getting smarter and interconnected. Microprocessors have appeared inside many commonly used items, and increasingly, they have been connected to networks. As the heart of personal computers and workstations, for example, microprocessors have been routinely connected to networks. They have also appeared inside devices with more specific functionality than the personal computer or the workstation.

Televisions, VCRs, audio components, fax machines, scanners, printers, cell phones, personal digital assistants, pagers, and wrist-watches--all have been enhanced with microprocessors; most have been connected to networks. Given the increasing capabilities and decreasing costs of information processing and data networking technologies, the network is rapidly extending its reach.

The emerging infrastructure of smart devices and computers interconnected by networks represents a new environment for software--an environment that presents new challenges and offers new opportunities to software developers. Java is well suited to help software developers meet challenges and seize opportunities presented by the emerging computing environment, because Java was designed for networks. Its suitability for networked environments is inherent in its architecture, which enables secure, robust, platform-independent programs to be delivered across networks and run on a great variety of computers and devices.

The Challenges and Opportunities of Networks

One challenge presented to software developers by the increasingly network-centric hardware environment is the wide range of devices that networks interconnect. A typical network usually has many different kinds of attached devices, with diverse hardware architectures, operating systems, and purposes. Java addresses this challenge by enabling the creation of platform-independent programs. A single Java program can run unchanged on a wide range of computers and devices. Compared with programs compiled for a specific hardware and operating system, platform-independent programs written in Java can be easier and cheaper to develop, administer, and maintain.

Another challenge the network presents to software developers is security. In addition to their potential for good, networks represent an avenue for malicious programmers to steal or destroy information, steal computing resources, or simply be a nuisance. Virus writers, for example, can place their wares on the network for unsuspecting users to download. Java addresses the security challenge by providing an environment in which programs downloaded across a network can be run with customized degrees of security.

One aspect of security is simple program robustness. Like devious code written by malicious programmers, buggy code written by well-meaning programmers can potentially destroy information, monopolize compute cycles, or cause systems to crash. Java's architecture guarantees a certain level of program robustness by preventing certain types of pernicious bugs, such as memory corruption, from ever occurring in Java programs. This establishes trust that downloaded code will not inadvertently (or intentionally) crash, but it also has an important benefit unrelated to networks: it makes programmers more productive. Because Java prevents many types of bugs from ever occurring, Java programmers need not spend time trying to find and fix them.

One opportunity created by an omnipresent network is online software distribution. Java takes advantage of this opportunity by enabling the transmission of binary code in small pieces across networks. This capability can make Java programs easier and cheaper to deliver than programs that are not network-mobile. It can also simplify version control. Because the most recent version of a Java program can be delivered on-demand across a network, you needn't worry about what version your end-users are running. They will always get the most recent version each time they use your program.

Mobile code gives rise to another opportunity: mobile objects, the transmission of both code and state across the network. Java realizes the promise of object mobility in its APIs for object serialization and RMI (Remote Method Invocation). Built on top of Java's underlying architecture, object serialization and RMI provide an infrastructure that enables the various components of distributed systems to share objects. The network-mobility of objects makes possible new models for distributed systems programming, effectively bringing the benefits of object-oriented programming to the network.

Platform independence, security, and network-mobility--these three facets of Java's architecture work together to make Java suitable for the emerging networked computing environment. Because Java programs are platform independent, network-mobility of code and objects is more practical. The same code can be sent to all the computers and devices the network interconnects. Objects can be exchanged between various components of a distributed system, which can be running on different kinds of hardware. Java's built-in security framework also helps make network-mobility of software more practical. By reducing risk, the security framework helps to build trust in a new paradigm of network-mobile software.

The Architecture

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

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

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

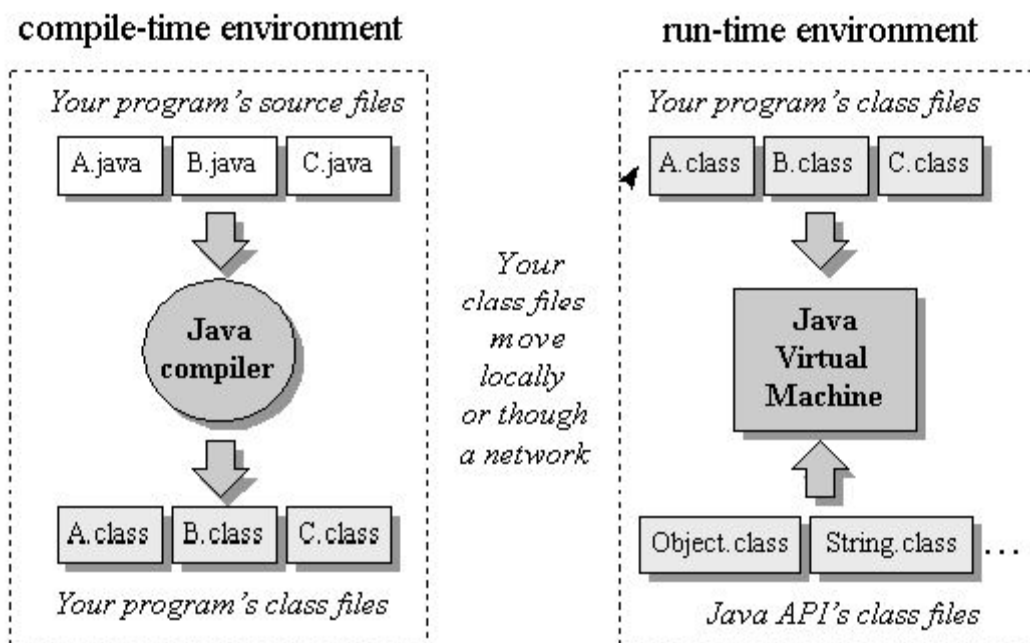


Fig 8.1- The Java programming environment

Together, the Java virtual machine and Java API form a "platform" for which all Java programs are compiled. In addition to being called the *Java runtime system*, the combination of the Java virtual machine and Java API is called the *Java Platform* (or, starting with version 1.2, the *Java 2 Platform*).

Java programs can run on many different kinds of computers because the Java Platform can itself be implemented in software. As you can see in Figure 8.2, a Java program can run anywhere the Java Platform is present.

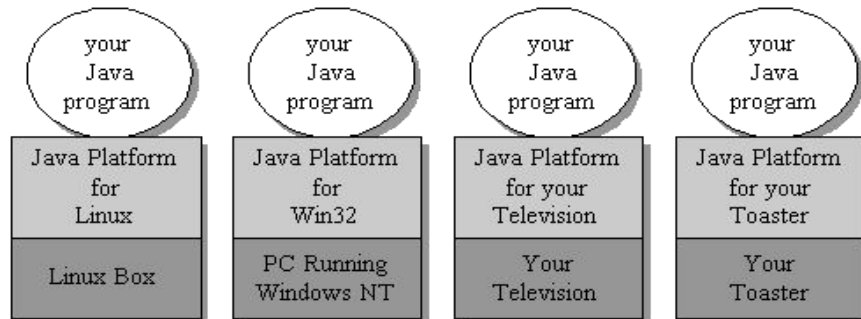


Fig 8.2- Java programs run on top of the Java Platform

The Java Virtual Machine

At the heart of Java's network-orientation is the Java virtual machine, which supports all three prongs of Java's network-oriented architecture: platform independence, security, and network-mobility. The Java virtual machine is an abstract computer. Its specification defines certain features every Java virtual machine must have, but leaves many choices to the designers of each implementation. For example, although all Java virtual machines must be able to execute Java byte codes, they may use any technique to execute them. Also, the specification is flexible enough to allow a Java virtual machine to be implemented either completely in software or to varying degrees in hardware. The flexible nature of the Java virtual machine's specification enables it to be implemented on a wide variety of computers and devices. A Java virtual machine's main job is to load class files and execute the bytecodes they contain. As you can see in Figure 8.3, the Java virtual machine contains a *class loader*, which loads class files from both the program and the Java API. Only those class files from the Java API that are actually needed by a running program are loaded into the virtual machine. The bytecodes are executed in an *execution engine*.

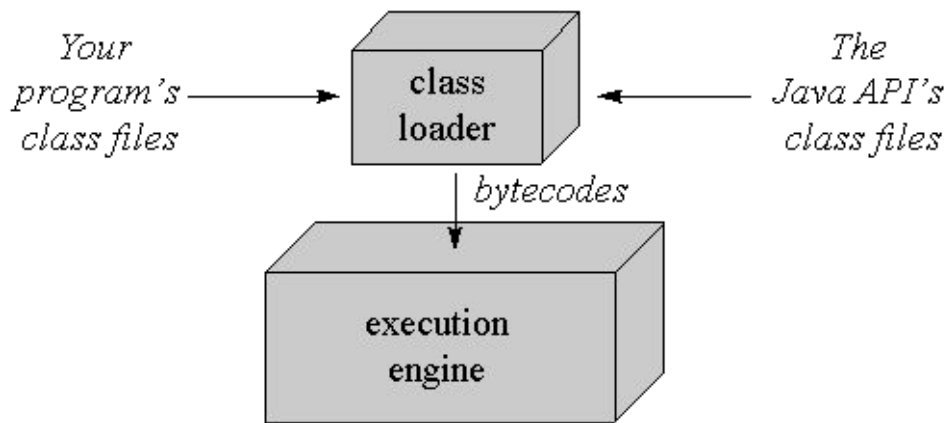


Fig 8.3- A basic block diagram of the Java virtual machine

The execution engine is one part of the virtual machine that can vary in different implementations. On a Java virtual machine implemented in software, the simplest kind of execution engine just interprets the bytecodes one at a time. Another kind of execution engine, one that is faster but requires more memory, is a *just-in-time compiler*. In this scheme, the bytecodes of a method are compiled to native machine code the first time the method is invoked. The native machine code for the method is then cached, so it can be re-used the next time that same method is invoked. A third type of execution engine is an *adaptive optimizer*. In this approach, the virtual machine starts by interpreting bytecodes, but monitors the activity of the running program and identifies the most heavily used areas of code. As the program runs, the virtual machine compiles to native and optimizes just these heavily used areas. The rest of the code, which is not heavily used, remain as byte codes which the virtual machine continues to interpret.

This adaptive optimization approach enables a Java virtual machine to spend typically 80 to 90% of its time executing highly optimized native code, while requiring it to compile and optimize only the 10 to 20% of the code that really matters to performance. Lastly, on a Java virtual machine built on top of a chip that executes Java byte codes natively, the execution engine is actually embedded in the chip.

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

When running on a Java virtual machine that is implemented in software on top of a host operating system, a Java program interacts with the host by invoking *native methods*. In Java, there are two kinds of methods: Java and native. A Java method is written in the Java language, compiled to byte codes, and stored in class files. A native method is written in some other language, such as C, C++, or assembly, and compiled to the native machine code of a particular processor. Native methods are stored in a dynamically linked library whose exact form is platform specific. While Java methods are platform independent, native methods are not. When a running Java program calls a native method, the virtual machine loads the dynamic library that contains the native method and invokes it. As you can see in Figure 8.4, native methods are the connection between a Java program and an underlying host operating system.

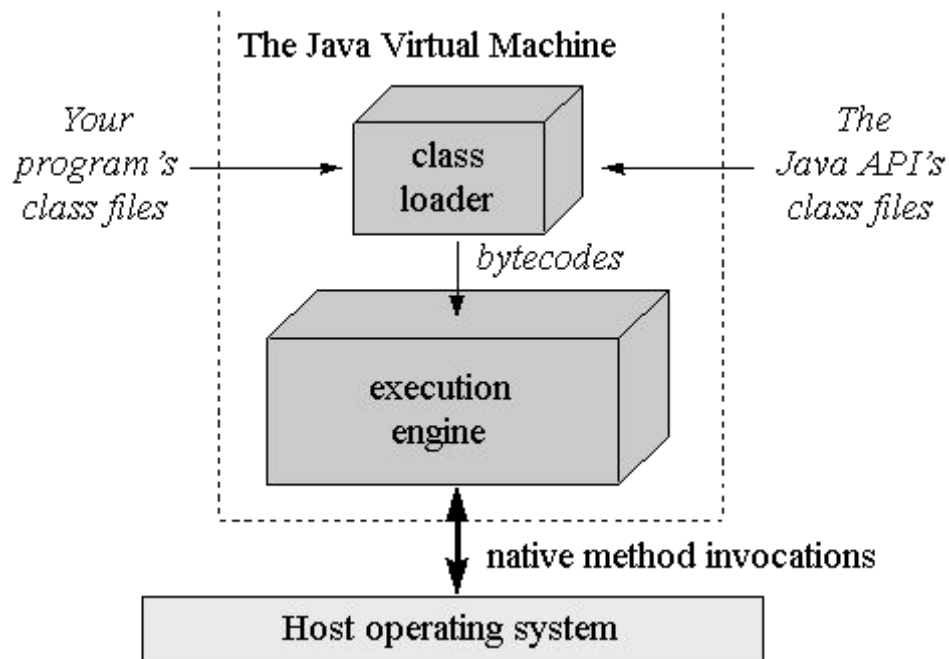


Fig 8.4- A Java virtual machine implemented in software on top of a host operating system

You can use native methods to give your Java programs direct access to the resources of the underlying operating system. Their use, however, will render your program platform specific, because the dynamic libraries containing the native methods are platform specific. In addition, the use of native methods may render your program specific to a particular implementation of the Java Platform. One native method interface--the *Java Native Interface*, or *JNI*--enables native methods to work with any Java Platform implementation on a particular host computer. Vendors of the Java Platform, however, are not necessarily required to support JNI. They may provide their own proprietary native method interfaces in addition to (or depending on their contract, in place of) JNI.

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

The Class Loader Architecture

One aspect of the Java virtual machine that plays an important role in both security and network- mobility is the class loader architecture. In the block diagrams of Figures 8.3 and 8.4, a single mysterious cube identifies itself as "the class loader," but in reality there may be more than one class loader inside a Java virtual machine. Thus the class loader cube of the block diagram actually represents a subsystem that may involve many class loaders. The Java virtual machine has a flexible class loader architecture that allows a Java application to load classes in custom ways.

A Java application can use two types of class loaders: a "bootstrap" class loader and user-defined class loaders. The bootstrap class loader (there is only one of them) is a part of the Java virtual machine implementation. For example, if a Java virtual machine is implemented as a C program on top of an existing operating system, then the bootstrap class loader will be part of that C program. The bootstrap class loader loads classes, including the classes of the Java API, in some default way, usually from the local disk. (The bootstrap class loader has also been called the primordial class loader, system class loader, or default class loader). At run-time, a Java application can install user-defined class loaders that load classes in custom ways, such as by downloading class files across a network.

While the bootstrap class loader is an intrinsic part of the virtual machine implementation, user-defined class loaders are not. Instead, user-defined class loaders are written in Java, compiled to class files, loaded into the virtual machine, and instantiated just like any other object. They are really just another part of the executable code of a running Java application. You can see a graphical depiction of this architecture in Figure 8.5.

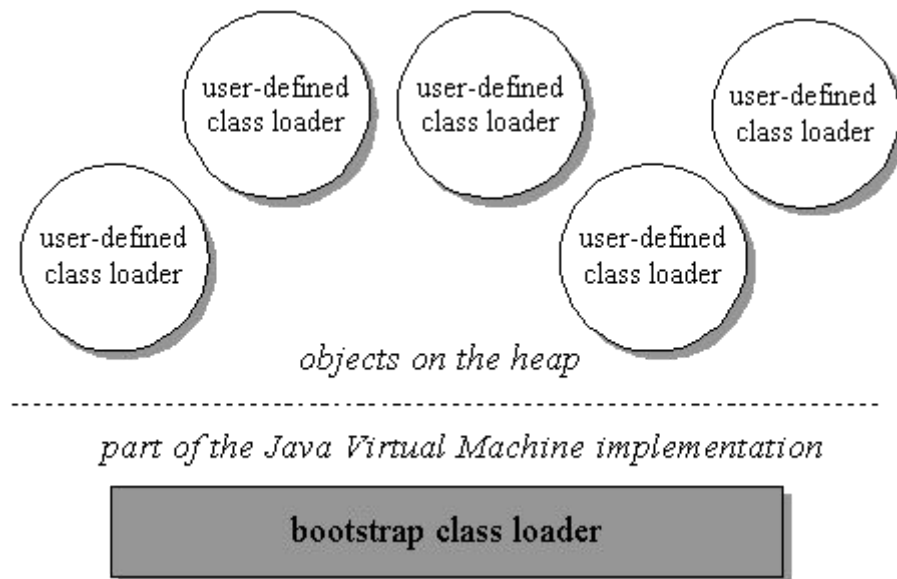


Fig 8.5-Java's class loader architecture

Because of user-defined class loaders, you don't have to know at compile-time all the classes that may ultimately take part in a running Java application. User-defined class loaders enable you to dynamically extend a Java application at run-time. As it runs, your application can determine what extra classes it needs and load them through one or more user-defined class loaders. Because you write the class loader in Java, you can load classes in any manner expressible in Java code. You can download them across a network, get them out of some kind of database, or even calculate them on the fly.

For each class it loads, the Java virtual machine keeps track of which class loader--whether bootstrap or user-defined--loaded the class. When a loaded class first refers to another class, the virtual machine requests the referenced class from the same class loader that originally loaded the referencing class. For example, if the virtual machine loads class `Volcano` through a particular class loader, it will attempt to load any classes `Volcano` refers to through the same class loader. If `Volcano` refers to a class named `Lava`, perhaps by invoking a method in class `Lava`, the virtual machine will request `Lava` from the class loader that loaded `Volcano`. The `Lava` class returned by the class loader is dynamically linked with class `Volcano`.

Because the Java virtual machine takes this approach to loading classes, classes can by default only see other classes that were loaded by the same class loader. In this way, Java's architecture enables you to create multiple *name-spaces* inside a single Java application. Each class loader in your running Java program has its own name-space, which is populated by the names of all the classes it has loaded.

A Java application can instantiate multiple user-defined class loaders either from the same class or from multiple classes. It can, therefore, create as many (and as many different kinds of) user-defined class loaders as it needs. Classes loaded by different class loaders are in different name-spaces and cannot gain access to each other unless the application explicitly allows it. When you write a Java application, you can segregate classes loaded from different sources into different name-spaces. In this way, you can use Java's class loader architecture to control any interaction between codes loaded from different sources. In particular, you can prevent hostile code from gaining access to and subverting friendly code.

One example of dynamic extension is the web browser, which uses user-defined class loaders to download the class files for applets across a network. A web browser fires off a Java application that installs a user-defined class loader--usually called an *applet class loader*-- that knows how to request class files from an HTTP server. Applets are an example of dynamic extension, because the Java application doesn't know when it starts which class files the browser will ask it to download across the network. The class files to download are determined at run-time, as the browser encounters pages that contain Java applets.

The Java application started by the web browser usually creates a different user-defined class loader for each location on the network from which it retrieves class files. As a result, class files from different sources are loaded by different user-defined class loaders. This places them into different name-spaces inside the host Java application. Because the class files for applets from different sources are placed in separate name-spaces, the code of a malicious applet is restricted from interfering directly with class files downloaded from any other source.

By allowing you to instantiate user-defined class loaders that know how to download class files across a network, Java's class loader architecture supports network-mobility. It supports security by allowing you to load class files from different sources through different user-defined class loaders. This puts the class files from different sources into different name-spaces, which allows you to restrict or prevent access between code loaded from different sources.

The Java Class File

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

When you compile and link a C++ program, the executable binary file you get is specific to a particular target hardware platform and operating system because it contains machine language specific to the target processor. A Java compiler, by contrast, translates the instructions of the Java source files into byte codes, the "machine language" of the Java virtual machine.

In addition to processor-specific machine language, another platform-dependent attribute of a traditional binary executable file is the byte order of integers. In executable binary files for the Intel X86 family of processors, for example, the byte order is *little-endian*, or lower order byte first. In executable files for the Power PC chip, however, the byte order is *big-endian*, or higher order byte first. In a Java class file, byte order is big-endian irrespective of what platform generated the file and independent of whatever platforms may eventually use it.

In addition to its support for platform independence, the Java class file plays a critical role in Java's architectural support for network-mobility. First, class files were designed to be compact, so they can more quickly move across a network. Also, because Java programs are dynamically linked and dynamically extensible, class files can be downloaded as needed. This feature helps a Java application manage the time it takes to download class files across a network, so the end-user's wait time can be kept to a minimum.

The Java API

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

The class files of the Java API are inherently specific to the host platform. The API's functionality must be implemented expressly for a particular platform before that platform can host Java programs. To access the native resources of the host, the Java API calls native methods. As you can see in Figure 1-6, the class files of the Java API invoke native methods so your Java program doesn't have to.

In this manner, the Java API's class files provide a Java program with a standard, platform-independent interface to the underlying host. To the Java program, the Java API looks the same and behaves predictably no matter what platform happens to be underneath. Precisely because the Java virtual machine and Java API are implemented specifically for each particular host platform, Java programs themselves can be platform independent.

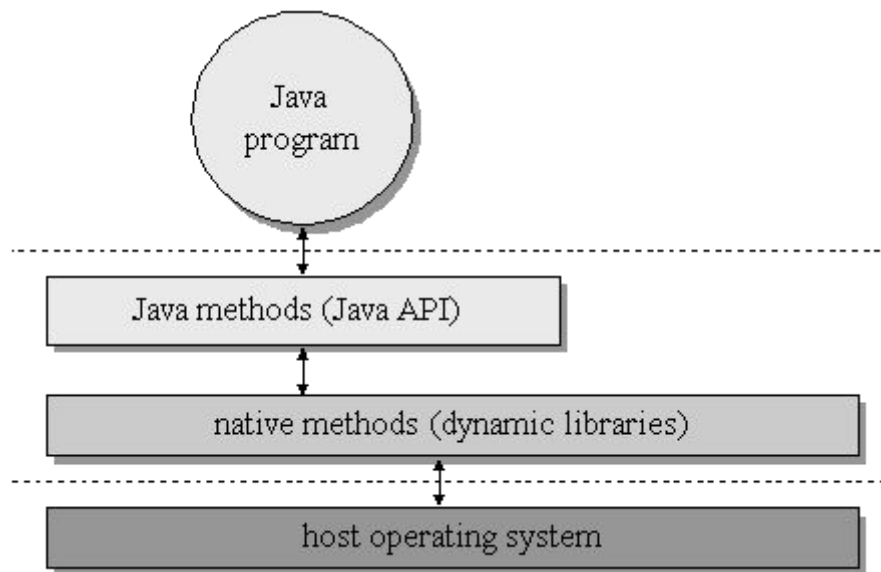


Fig 8.6-A platform-independent Java program

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

Instead, it encourages implementations to adopt the look and feel of the underlying platform. The Swing library offers even more flexibility -- enabling the look and feel to be chosen by the programmer. Also, because the size of fonts, buttons, and other user interface components will vary from platform to platform, the AWT and Swing include *layout managers* to position the elements of a window or dialog box at run-time. Rather than forcing you to indicate exact X and Y coordinates for the various elements that constitute, say, a dialog box, the layout manager positions them when your dialog box is displayed. With the aim of making the dialog look its best on each platform, the layout manager will very likely position the dialog box elements slightly differently on different platforms. In these ways and many others, the internal architecture of the Java API is aimed at facilitating the platform independence of the Java programs that use it. In addition to facilitating platform independence, the Java API contributes to Java's security model. The methods of the Java API, before they perform any action that could potentially be harmful (such as writing to the local disk), check for permission. In Java releases prior to 1.2, the methods of the Java API checked permission by querying the *security manager*. A security manager could, for example, forbid access to the local disk. If the application requested a local disk write by invoking a method from the pre-1.2 Java API, that method would first check with the security manager. Upon learning from the security manager that disk access is forbidden, the Java API would refuse to perform the write. In Java 1.2, the job of the security manager was taken over by the *access controller*, a class that performs stack inspection to determine whether the operation should be allowed. (For backwards compatibility, the security manager still exists in Java 1.2.) By enforcing the security policy established by the security manager and access controller, the Java API helps to establish a safe environment in which you can run potentially unsafe code.

The Java Programming Language

Although Java was designed for the network, its utility is not restricted to networks. Platform independence, network-mobility, and security are of prime importance in a networked computing environment, but you may not always find yourself facing network-oriented problems. As a result, you may not always want to write programs that are platform independent. You may not always want to deliver your programs across networks or limit their capabilities with security restrictions. There may be times when you use Java technology primarily because you want to get the advantages of the Java programming language.

As a whole, Java technology leans heavily in the direction of networks, but the Java programming language is quite general-purpose. The Java language allows you to write programs that take advantage of many software technologies:

- object-orientation
- multi-threading
- structured error-handling
- garbage collection
- dynamic linking
- dynamic extension

Instead of serving as a test bed for new and experimental software technologies, the Java language combines in a new way concepts and techniques that had already been tried and proven in other languages. These concepts and techniques make the Java programming language a powerful general-purpose tool that you can apply to a variety of situations, independent of whether or not they involve a network.

At the beginning of a new project, you may be faced with the question, "Should I use C++ (or some other language) for my next project, or should I use Java?" As an implementation language, Java has some advantages and some disadvantages over other languages. One of the most compelling reasons for using Java as a language is that it can enhance developer productivity. The main disadvantage is potentially slower execution speed. Java is, first and foremost, an object-oriented language. One promise of object-orientation is that it promotes the re-use of code, resulting in better productivity for developers. This may make Java more attractive than a procedural language such as C, but doesn't add much value to Java over C++. Yet compared to C++, Java has some significant differences that can improve a developer's productivity.

In Java, there is no way to directly access memory by arbitrarily casting pointers to a different type or by using pointer arithmetic, as there is in C++. Java requires that you strictly obey rules of type when working with objects. If you have a *reference* (similar to a pointer in C++) to an object of type `Mountain`, you can only manipulate it as a `Mountain`. You can't cast the reference to type `Lava` and manipulate the memory as if it were a `Lava`. Neither can you simply add an arbitrary offset to the reference, as pointer arithmetic allows you to do in C++. You can, in Java, cast a reference to a different type, but only if the object really is of the new type. For example, if the `Mountain` reference actually referred to an instance of class `Volcano` (a specialized type of `Mountain`), you could cast the `Mountain` reference to a `Volcano` reference. Because Java enforces strict type rules at run-time, you are not able to directly manipulate memory in ways that can accidentally corrupt it. As a result, you can't ever create certain kinds of bugs in Java programs that regularly harass C++ programmers and hamper their productivity.

Another way Java prevents you from inadvertently corrupting memory is through automatic garbage collection. Java has a `new` operator, just like C++, that you use to allocate memory on the heap for a new object. But unlike C++, Java has no corresponding `delete` operator, which C++ programmers use to free the memory for an object that is no longer needed by the program. In Java, you merely stop referencing an object, and at some later time, the garbage collector will reclaim the memory occupied by the object.

The garbage collector prevents Java programmers from needing to explicitly indicate which objects should be freed. As a C++ project grows in size and complexity, it often becomes increasingly difficult for programmers to determine when an object should be freed, or even whether an object has already been freed. This results in memory leaks, in which unused objects are never freed, and memory corruption, in which the same object is accidentally freed multiple times. Both kinds of memory troubles cause C++ programs to crash, but in ways that make it difficult to track down the exact source of the problem. You can be more productive in Java primarily because you don't have to chase down memory corruption bugs. But also, you can be more productive because when you no longer have to worry about explicitly freeing memory, program design becomes easier. A third way Java protects the integrity of memory at run-time is array bounds checking. In C++, arrays are really shorthand for pointer arithmetic, which brings with it the potential for memory corruption. C++ allows you to declare an array of ten items, then write to the eleventh item, even though that tramples on memory. In Java, arrays are full-fledged objects, and array bounds are checked each time an array is used. If you create an array of ten items in Java and try to write to the eleventh, Java will throw an exception. Java won't let you corrupt memory by writing beyond the end of an array.

One final example of how Java ensures program robustness is by checking object references, each time they are used, to make sure they are not `null`. In C++, using a null pointer usually results in a program crash. In Java, using a null reference results in an exception being thrown.

The productivity boost you can get just by using the Java language results in quicker development cycles and lower development costs. You can realize further cost savings if you take advantage of the potential platform independence of Java programs. Even if you are not concerned about a network, you may still want to deliver a program on multiple platforms. Java can make support for multiple platforms easier, and therefore, cheaper.

PHP Server

PHP is a server-side scripting language designed for web development but also used as a general-purpose programming language. PHP is now installed on more than 244 million websites and 2.1 million web servers. Originally created by Rasmus Lerdorf in 1995, the reference of PHP is now produced by The PHP Group. While PHP originally stood for *Personal Home Page*, it now stands for *PHP: Hypertext Preprocessor*, a recursive backronym.

PHP code is interpreted by a web server with a PHP processor module, which generates the resulting web page: PHP commands can be embedded directly into an HTML source document rather than calling an external file to process data. It has also evolved to include a command-line interface capability and can be used in standalone graphical applications.

PHP is free software released under the PHP License. PHP can be deployed on most web servers and also as a standalone shell on almost every operating system and platform, free of charge.

PHP development began in 1994 when the developer Rasmus Lerdorf wrote a series of Common Gateway Interface (CGI)Perl scripts, which he used to maintain his personal homepage. The tools performed tasks such as displaying his résumé and recording his web traffic.

He rewrote these scripts in C for performance reasons, extending them to add the ability to work with web forms and to communicate with databases, and called this implementation "Personal Home Page/Forms Interpreter" or PHP/FI.

PHP/FI could be used to build simple, dynamic web applications. Lerdorf initially announced the release of PHP/FI as "Personal Home Page Tools (PHP Tools) version 1.0" publicly to accelerate bug location and improve the code, on the comp.infosystems.www.authoring.cgi Usenet discussion group on June 8, 1995. This release already had the basic functionality that PHP has as of 2013. This included Perl-like variables, form handling, and the ability to embed HTML.

Early PHP was not intended to be a new programming language, and grew organically, with Lerdorf noting in retrospect: "I don't know how to stop it, there was never any intent to write a programming language [...] I have absolutely no idea how to write a programming language, I just kept adding the next logical step on the way." A development team began to form and, after months of work and beta testing, officially released PHP/FI 2 in November 1997.

Zeev Suraski and Andi Gutmans rewrote the parser in 1997 and formed the base of PHP 3, changing the language's name to the recursive acronym *PHP: Hypertext Preprocessor*. Afterwards, public testing of PHP 3 began, and the official launch came in June 1998. Suraski and Gutmans then started a new rewrite of PHP's core, producing the Zend Engine in 1999. They also founded Zend Technologies in Ramat Gan, Israel.

On May 22, 2000, PHP 4, powered by the Zend Engine 1.0, was released. As of August 2008 this branch reached version 4.4.9. PHP 4 is no longer under development nor will any security updates be released.

On July 13, 2004, PHP 5 was released, powered by the new Zend Engine II. PHP 5 included new features such as improved support for object-oriented programming, the PHP Data Objects (PDO) extension (which defines a lightweight and consistent interface for accessing databases), and numerous performance enhancements. In 2008 PHP 5 became the only stable version under development. Late static binding had been missing from PHP and was added in version 5.3.

CONCLUSION

CHAPTER 9

CONCLUSION

Android is the latest technology, which has its own specification and many applications for real time use. Main reason to use the android is it's a user friendly. Now a day's PHP is the most securable language in technology side. Combinable both platforms give easy survey, design, implementation and finally security. As a result we get an exactly result of transportation System. And this result is more useful for the business people, passengers and also for tourist. It alert the passengers when destination come using GPS tracking is a method of finding out exactly location of the place. GPS tracking System uses the Global Navigation Satellite System(GNSS) network.

CHAPTER 10

FUTURE ENHACEMENT

Nowadays mobile devices became part of our life and we have much comfortable while using smart phones. There are lot of application satisfied our daily uses. MTC'roid Route Maestro is a bus app's .we done this application in android in future it will further develop to support in all other operating system like ios, Symbian, bada, blackberry, window etc. This app's can used in corporation eg: Neyveli, Bangalore MTC and it combine with train.

APPENDICES

APPENDICES

10.1 APPENDIX1-SAMPLE CODE-A

GPS TRACKER

```
import android.content.Context;
import android.media.MediaPlayer;
import android.util.Log;
public class AlarmScreen {
    MediaPlayer mPlayer;
    static Context context;
    static double destLat, destLng;
    GPSTracker gps;
    public AlarmScreen() {
        checkCurrentLocation();
    } public void checkCurrentLocation() {
        gps = new GPSTracker(context);
        if (gps.canGetLocation()) {
            Log.d("destLat alarm screen", ""+destLat);
            Log.d("destLng alarm screen", ""+destLng);
            double latitude = gps.getLatitude();
            double longitude = gps.getLongitude();
            Log.d("current-Lat alarm screen", ""+latitude);
            Log.d("current-Lng alarm screen", ""+longitude);
            if(destLat-latitude<.1000000||destLng-longitude<.1000000){
                mPlayer = MediaPlayer.create(context, R.raw.alert);
                mPlayer.start();
                Log.d("alert called", "alert called");
            }
        }
    }
}
```

FARE CALCULATE ACTIVITY

```
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import com.example.mtc.utils.Constants;
import com.example.mtc.utils.XMLfunctions;
import android.app.Activity;
import android.app.ProgressDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.Button;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;
public class FareCalculateActivity extends Activity{
    Spinner mBuses;
    String[] mBusNumbers;
    ArrayAdapter<String> arrayAdapter;
    String mSource,mDestination;
    TextView mFare;
    Button mGetFare;
    Spinner mBusType;
```

```

String[] mBusTypes = { "White", "Green", "Blue", "Night", "Volvo" };
ArrayAdapter<String> busAdapter;
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_farecalculate);
    mBuses = (Spinner) findViewById(R.id.buses);
    mBusType = (Spinner) findViewById(R.id.bustype);
    mGetFare = (Button) findViewById(R.id.getfare);
    mFare = (TextView) findViewById(R.id.fare);
    busAdapter = new ArrayAdapter<String>(FareCalculateActivity.this,
        android.R.layout.simple_spinner_item, mBusTypes);
    busAdapter.setDropDownViewResource(android.R.layout.simple_spinner_d
        ropdown_item);
    mBusType.setAdapter(busAdapter);
    mBusType.setPrompt("Select Bus Type");
    Bundle sender = getIntent().getExtras();
    mSource = sender.getString("source");
    mDestination = sender.getString("destination");
    mBuses.setPrompt("Choose a Bus");
    loadResponse(mSource, mDestination);
    mGetFare.setOnClickListener(new View.OnClickListener() {
        public void onClick(View arg0) {
            String mBusno = arrayAdapter.getItem(mBuses.getSelectedItemPosition());
            String mBustype =
                busAdapter.getItem(mBusType.getSelectedItemPosition());
            loadCost(mSource, mDestination, mBusno, mBustype);
        }
    });
}

```

```

public void loadCost(String mSur, String mDestn, String mBusno ,String
mBustyp) {
String url = Constants.COST_API + "BusNo="+mBusno + "&source=" +
mSur
+ "&Dest=" + mDestn + "&BusType=" + mBustyp;
url = url.replace(" ", "%20");
Log.d("url ", "" + url);
DownloadWebPageTask01 task = new DownloadWebPageTask01();
task.execute(new String[] { url.toString() });}

public class DownloadWebPageTask01 extends AsyncTask<String, Void,
String> {
final ProgressDialog dialog = new
ProgressDialog(FareCalculateActivity.this);
private volatile boolean running = true;
public DownloadWebPageTask01() {
running = true;}
protected void onPreExecute() {
this.dialog.setCancelable(true);
this.dialog.setMessage("Loading data...");
this.dialog.show();
this.dialog.setOnCancelListener(new DialogInterface.OnCancelListener() {
public void onCancel(DialogInterface dialog) {
cancel(true);}
});}
protected String doInBackground(String... urls) {
String response = "";
while (running) {
response = XMLfunctions.getXML(urls[0]);
Log.d("response", "" + response);

```

```

running = false;}
return response;}

protected void onPostExecute(String result) {
    if (!result.startsWith("<html>")) {
        Document doc = XMLfunctions.XMLfromString(result);
        int numResults = XMLfunctions.numResults(doc);
        Log.d("numResult", "" + numResults);
        if (numResults == -1 || numResults == 0) {
            Toast toast = Toast.makeText(getApplicationContext(),
                "Cannot Connect to Server", Toast.LENGTH_SHORT);
            toast.show(); } else {
                NodeList nodes = doc.getElementsByTagName("Database");
                Element d = (Element) nodes.item(0);
                String status = (XMLfunctions.getValue(d, "Message"));
                if (status.equals("Success")) {
                    String cost = (XMLfunctions.getValue(d, "Cost"));
                    mFare.setText(cost);
                } else {
                    Toast.makeText(getApplicationContext(),
                        "Cannot Connect to Server", Toast.LENGTH_SHORT).show();
                } } if (dialog.isShowing()) {
                    dialog.dismiss();}
                } else {
                    if (dialog.isShowing()) {
                        dialog.dismiss();}
                    Toast toast = Toast.makeText(getApplicationContext(),
                        "Cannot Connect to Server", Toast.LENGTH_SHORT);
                    toast.show();
                } }
    }
}

```

```

protected void onCancelled() {
    running = false;}}
public void loadResponse(String mSource,String mDestination) {
    String url =
    Constants.BUSNO_API+"source="+mSource+"&Dest="+mDestination;
    url = url.replace(" ", "%20");
    Log.d("url ", "" + url);
    DownloadWebPageTask task = new DownloadWebPageTask();
    task.execute(new String[] { url.toString() });}
public class DownloadWebPageTask extends AsyncTask<String, Void,
String> {
    final ProgressDialog dialog = new
    ProgressDialog(FareCalculateActivity.this);
    private volatile boolean running = true;
    public DownloadWebPageTask() {
        running = true;}
    protected void onPreExecute() {
        this.dialog.setCancelable(true);
        this.dialog.setMessage("Loading data...");
        this.dialog.show();
        this.dialog.setOnCancelListener(new DialogInterface.OnCancelListener() {
        public void onCancel(DialogInterface dialog) {
            cancel(true);
        }});}
    protected String doInBackground(String... urls) {
        String response = "";
        while (running) {
            response = XMLfunctions.getXML(urls[0]);
            Log.d("response", "" + response);

```

```

running = false;}
return response;}

protected void onPostExecute(String result) {
    if (!result.startsWith("<html>")) {
        Document doc = XMLfunctions.XMLfromString(result);
        int numResults = XMLfunctions.numResults(doc);
        Log.d("numResult", "" + numResults);
        if (numResults == -1 || numResults == 0) {
            Toast toast = Toast.makeText(getApplicationContext(),
                "Cannot Connect to Server", Toast.LENGTH_LONG);
            toast.show();} else {
                NodeList nodes = doc.getElementsByTagName("Database");
                Element d = (Element) nodes.item(0);
                String status = (XMLfunctions.getValue(d, "Message"));
                if (status.equals("Success")) {
                    NodeList nodes_list = doc.getElementsByTagName("Node");
                    int nodesSize = nodes_list.getLength();
                    Log.d("nodes size", "" + nodesSize);
                    if (nodesSize > 0) {
                        mBusNumbers=new String[nodesSize];
                        for (int i = 0; i < nodesSize; i++) {
                            Element e = (Element) nodes_list.item(i);
                            String source = (XMLfunctions.getValue(e, "BusNo"));
                            mBusNumbers[i]=source;}}
                        arrayAdapter = new ArrayAdapter<String>(FareCalculateActivity.this,
                            android.R.layout.simple_spinner_item,mBusNumbers);
                        arrayAdapter.setDropDownViewResource(android.R.layout.simple_spinner_
                            dropdown_item);
                        mBuses.setAdapter(arrayAdapter);

```

```

    }else
    {
        Toast.makeText(getApplicationContext(),"Cannot Connect to Server",
        Toast.LENGTH_SHORT).show();
    }
}
if (dialog.isShowing()) {
    dialog.dismiss();}
} else {
    if (dialog.isShowing()) {
        dialog.dismiss();}
    Toast toast = Toast.makeText(getApplicationContext(),"Cannot Connect to
    Server", Toast.LENGTH_SHORT);
    toast.show();
}
}
protected void onCancelled() {
    running = false;
}
}
public boolean isOnline(Context c) {
    ConnectivityManager cm = (ConnectivityManager) c
    .getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo ni = cm.getActiveNetworkInfo();
    if (ni != null && ni.isConnected())
        return true;
    else
        return false;}
}

```


B-SCREEN SHOTS

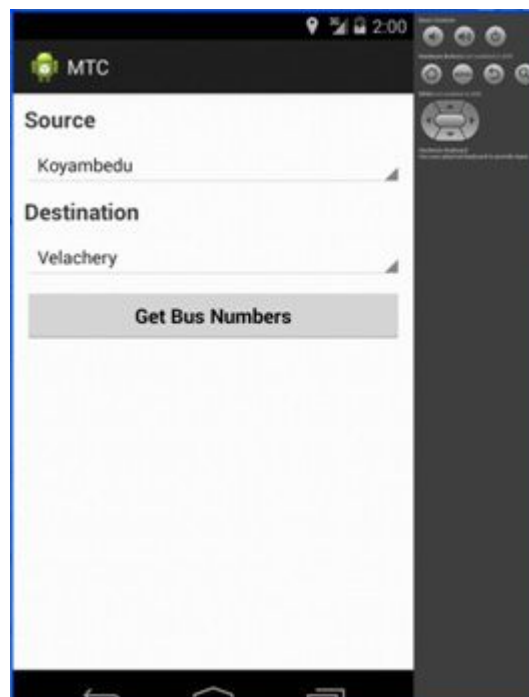
MOBILE MENU SCREEN



APPLICATION HOME SCREEN



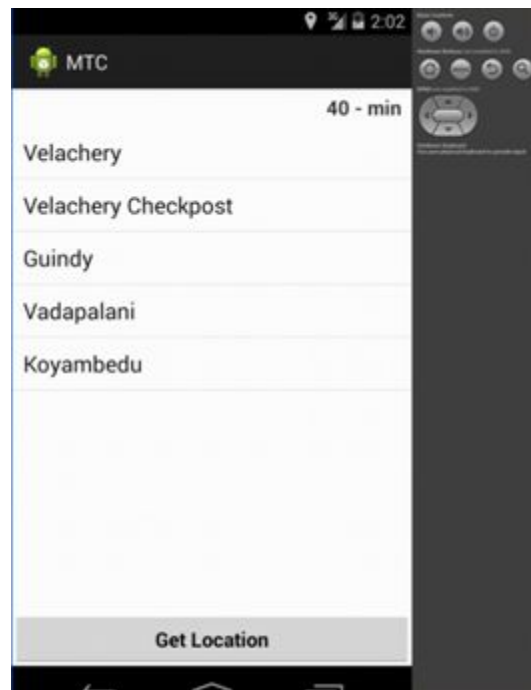
SEARCH BUS ROUTE



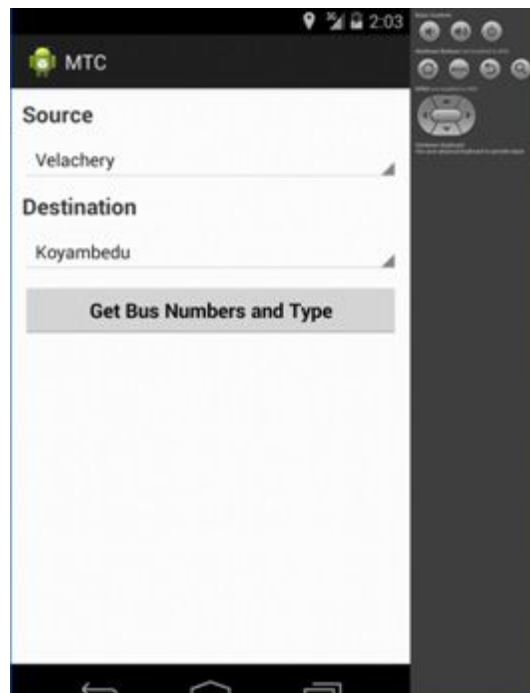
BUS LISTS



ROUTE



FARE CALCULATION



MTC

Source
Velachery

Destination
Koyambedu

Get Bus Numbers and Type

CLASSIFICATION OF BUS



MTC

Buses :
D70

Bus Type :
Green
White
Green
Blue
Night
Volvo

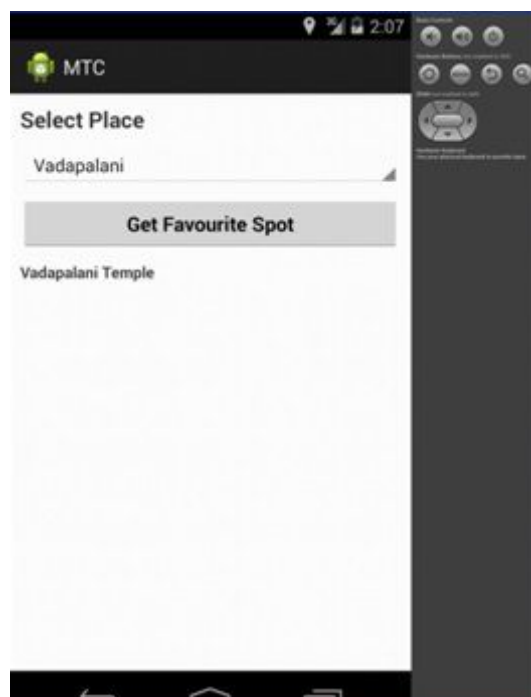
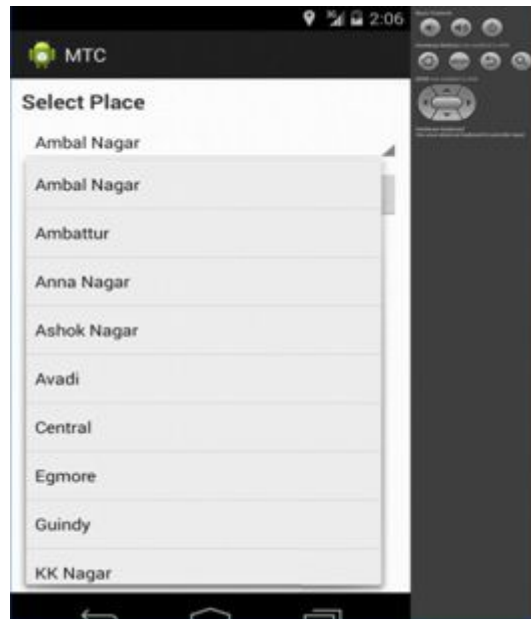
FARE AMOUNT (WHITE)

The screenshot shows the MTC app interface. At the top, there is a status bar with the time 2:04 and various icons. Below the status bar, the app header displays the MTC logo and name. The main content area has two dropdown menus: 'Buses :' with 'D70' selected and 'Bus Type :' with 'White' selected. Below these is a grey button labeled 'Get Fare'. Underneath the button, the text 'Fare : Rs. 10' is displayed. On the right side of the screen, there is a vertical sidebar with several circular icons and a larger circular icon with a bus symbol.

The screenshot shows the MTC app interface. At the top, there is a status bar with the time 2:05 and various icons. Below the status bar, the app header displays the MTC logo and name. The main content area has two dropdown menus: 'Buses :' with 'D70' selected and 'Bus Type :' with 'Blue' selected. Below these is a grey button labeled 'Get Fare'. Underneath the button, the text 'Fare : Rs. 21.0' is displayed. On the right side of the screen, there is a vertical sidebar with several circular icons and a larger circular icon with a bus symbol.



FAVOURITE PLACES



BIBLIOGRAPHY

- 1.[http://en.wikipedia.org/wiki/Metropolitan_Transport_Corporation_\(Chennai\)](http://en.wikipedia.org/wiki/Metropolitan_Transport_Corporation_(Chennai))
i)
- 2.<http://www.mtcbus.org/MTCchennai> Website
- 3.<http://play.google.com/store/search?q=Chennai+MTC&c=apps>(List of
previously created apps)
- 4.<http://www.livechennai.com/detailnews.asp?newsid=1609>
- 5.<http://articles.timesofindia.indiatimes.com>