Remote Compiler

By

PRAPHULL PUROHIT-(051221)
ABHISHEK BAJPAI-(051253)
GARIMA SINGH-(051286)

MAY – 2009

Submitted in partial fulfillment of the Degree of Bachelor of Technology

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the work entitled, ***"Remote Compiler"*** submitted by **Praphull Purohit (051221), Abhishek Bajpai (051253)** and **Garima Singh (051286)** in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science and Engineering of Jaypee University of Information Technology has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

[illegible]

Brig (retd.) S.P Ghrera

(Head of Department – Computer Science and I.T.)

[illegible]

Mr. Satish Chandra.

(Project Coordinator)

Infosys®

Infosys Technologies Limited
No. 350, Hebbal Electronics City
Hootagalli, Mysore - 570 018
Tel: 91 821 240 4101 Fax: 91 821 240 4200
www.infosys.com

Certificate of Project Completion at Infosys

This is to certify that **Mr./Ms. Praphull Purohit** has undertaken the project titled "Remote Compiler" at our organization **Infosys Technologies Limited, Mysore,** under the guidance of Mr. Ananth Kota for the period **19 January 2009** to 12 May **2009.**

[illegible]

Signature of Project Manager

ANANTH KOTA

Registered Office : Plot No. 44, Electronics City, Hosur Road, Bangalore - 560 100, India

BVQI UKAS IT

ACKNOWLEDGMENT

I extend my warm and sincere thanks to my project manager **Mr. Ananth Kota** who was a staunch supporter and motivator of this project. Right from the inception of this project work, my project manager guided me till the very end in the true sense of the word. He always came up with innovative ways and creative terms thus also helping me to instill and enhance the quality of creative thinking within myself.

Sincere thanks to **Brig (retd.) S.P. Ghrera , HOD, CSE & IT Department**, and **Mr. Satish Chandra**, Project Coordinator for being cooperative to the students of the department and providing relevant guidance in their endeavors.

Thanks to all the teaching and non-teaching staff of the CSE department who have helped in every possible way throughout the 4 years of the B. Tech. academic program.

I would also like to express my gratitude to this alma mater **JUIT, Waknaghat** for providing proper resources as and when required such as an all time internet facility and other resources.

Hence without giving a warm thanks to all of them who made this project work a reality my work would be incomplete.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ACRONYM	STANDS FOR
JS	JavaScript
JSP	Java Server Pages
XHTML	eXtensible HyperText Markup Language
AJAX	Asynchronous JavaScript and XML
SQL	Structured Query Language
XML	eXtensible Markup Language
WAR	Web Archive
JAR	Java Archive
JSON	JavaScript Object Notation
HTTP	HyperText Transfer Protocol

ABSTRACT

Code compilation is one of the most important phases of software development life cycle. Despite being this critical to the development of any software, least time is devoted by any development team to this phase. In an environment where compilation is not an everyday task, de-localizing compilation systems and providing remote access to compilation environments not only reduces the cost incurred for compilers but also saves time and manpower in maintaining and upgrading compilation systems.

Remote compiler is a software product which provides centralization of compilation environments and delocalization of compilation systems to web-based systems. This software surpasses the existing remote compilation systems by allowing for the use of input statements in source programs that, traditionally, would have caused the programs to wait indefinitely for user response at server end. A user can create or upload a source file to the system through a web-based interface, which is analyzed, compiled and executed at a remote system and results are displayed to the user, thereby eliminating the need for the presence of compiler at user's end. This system can be used by colleges for training purpose and for various programming competitions.

CHAPTER 1

INTRODUCTION

1.1 Business Requirements

The customer is a programmer in the network based environment who develops programs (in c, c++ or java) in the local system and tries to execute them on the centralized system. The local systems do not have the compilers to execute the programs. All the compilers are available on the centralized system.

1.2 Remote Compiler

- The Remote Compiler is a network based multi-user supported program that helps programmers to compile and execute programs coded in different programming languages(c, c++ and java) on a remote system.
- It avoids the need of various compilers in local system.
- The remote system comprises of required compilers to execute the program.
- The local system sends the programs, after modifying the input/output statements, to the remote system for compilation and execution.
- The remote system compiles/executes the program and sends the errors/output to the local system.
- Remote compiler helps to compile and execute the programs coded in any language(c, c++ or java) irrespective of the compiler being available in local system.
- To obtain the output from a program even when the environment needed to run the program is not available in the local system.

1.3 Functional Requirements

1.3.1 Requirements in scope

- The local system must be able to send files to the remote system.
- The remote system should compile the files received and send the results to the local system.
- The output should be returned to the local system when file is executed on the remote system.
- Security must be maintained through authenticated userId and password.
- Proper validations should be done before compiling the files.
- Future enhancements should be considered.

1.3.2 Requirements out of scope

- The remote system will not compile the programs that contain input statements nested inside looping structures.
- The remote system will not compile the programs that contain goto statements.
- The remote system will not compile the programs that contain multiple statements in single line.
- The remote system will not compile the programs that contain Structures.

1.3.3 User Interface

1.3.3.1 Current Features

- User logs in to the Remote System by providing a username and password.
- User can either upload a pre-existing source code file from his local system or can create his/her own source code file by typing the code in the provided text area which in turn will be sent to the remote system.
- User can also modify the source code and recompile it.
- User can create his own directory structure to store the files.
- User will be provided with an option to compile and execute his program.
- Results will be displayed on the user's local system after the program execution.
- If during execution of the program user needs to supply any input values, then he will be prompted before the compilation of the source to executable.

1.3.3.2 Future Enhancements

- The user can run SQL queries and view its results without having Database Server on the local system.
- The user can be assisted by an automated code generator tool while writing the code in the specific programming language.
- This project can be extended to support more programming languages by implementing the interfaces provided and configuring the compilation/execution system through the user interface provided.

1.4 Non functional requirements

- **Security**: UserId and password should be verified for accessing remote system.Only authenticated users can access the system.

- **Audit Trail**: Not Applicable.
- **Error logging**: Not Applicable.
- **Multi language Support**: Not Applicable.
- **Performance**: Multiple users can use the system simultaneously. Even when the file name of different users is same, the ambiguity of compiled filename is resolved depending upon the username and hence the unique filepath.
- **Scalability**: The project is scalable. It supports languages like c, c++ and java. But we can extend its functionality to other programming languages also by implementing the interfaces provided.
- **Availability / reliability**: The remote system compiler is available 24x7. Any authorised user can use it anytime. The compilation will be consistent across all the errors and deviations.
- **Data migration**: The programs written by the client will be modified, if required, and will be sent to the remote compiler for compilation and hence execution. The errors/output so generated will be sent back to the local machine.
- **Data Retention**: The files containing code (.c, .cpp, .java) and their respective executable files will be stored in the remote system.

CHAPTER 2

OVERVIEW OF REMOTE COMPILER

2.1 Requirements of Remote Compiler

2.1.1 Server Hardware requirements

- **Processor/RAM/HDD** : P4/1GB/10GB
- **Web server** : Tomcat/JBOSS
- **Database Server** : Oracle

2.1.2 Server Software requirements

- **OS for Web server** : Windows XP/Vista/2003/NT
- **OS for Database Server** : Windows XP/Vista/2003/NT
- **DBMS** : Oracle/DB2

2.1.3 Client Software requirements

- **Web browser**: Mozilla Firefox 3+, Internet Explorer 6+ (With JavaScript/AJAX enabled and ActiveX/MSXML runtime enabled for IE 6)

2.2 Overview

The main objective of this software is to compile the code sent by local system on the remote system compiler. The remote compiler should execute or run the code and send the error messages or the output to the local system.

2.3 Subsystems

The various subsystems identified in the project are:

- **Login**: This subsystem deals with allowing only authorized users to use the remote compiler. If the user is authorized, then he/she is allowed to access the compiler and other subsystems. It interacts with JavaScript component and database.
- **Graphical User Interface**: This subsystem provides a user interface that allows the user to create his/her own directory structure, provide the inputs to the program if any, send the files to remote compiler, compile and execute the user program and display the relevant outputs and errors, if any. To carry out its operations it interacts with JavaScript component.
- **File Manager**: This subsystem manages the file operations like getting the list of files corresponding to the user. It saves the modified source code and deletes the irrelevant temporary files created in the process.
- **Code Analyser**: It searches for the input statements in the source code file and, if any exists, then it prompts the user to enter the appropriate values before actual compilation begins. Then its replaces the source code with appropriate assignments and saves the

modified source code in a temporary file for its compilation and execution. It interacts with JavaScript component.

- **Compiler**: This subsystem compiles and executes the user files and sends back the results to the user's system. It interacts with JavaScript component.
- **JavaScript Component**: This subsystem provides the functionality for displaying the directory structure for the user when he/she logs on to the remote compiler; also it displays the custom dialog boxes. It interacts with subsystems login, code analyser, compiler GUI.
- **Database**: This subsystem stores the authorised user's details, the configuration details of languages, compilers, source files, temporary files, executable files which are required to make the project portable. It interacts with login.

2.4 Use case design

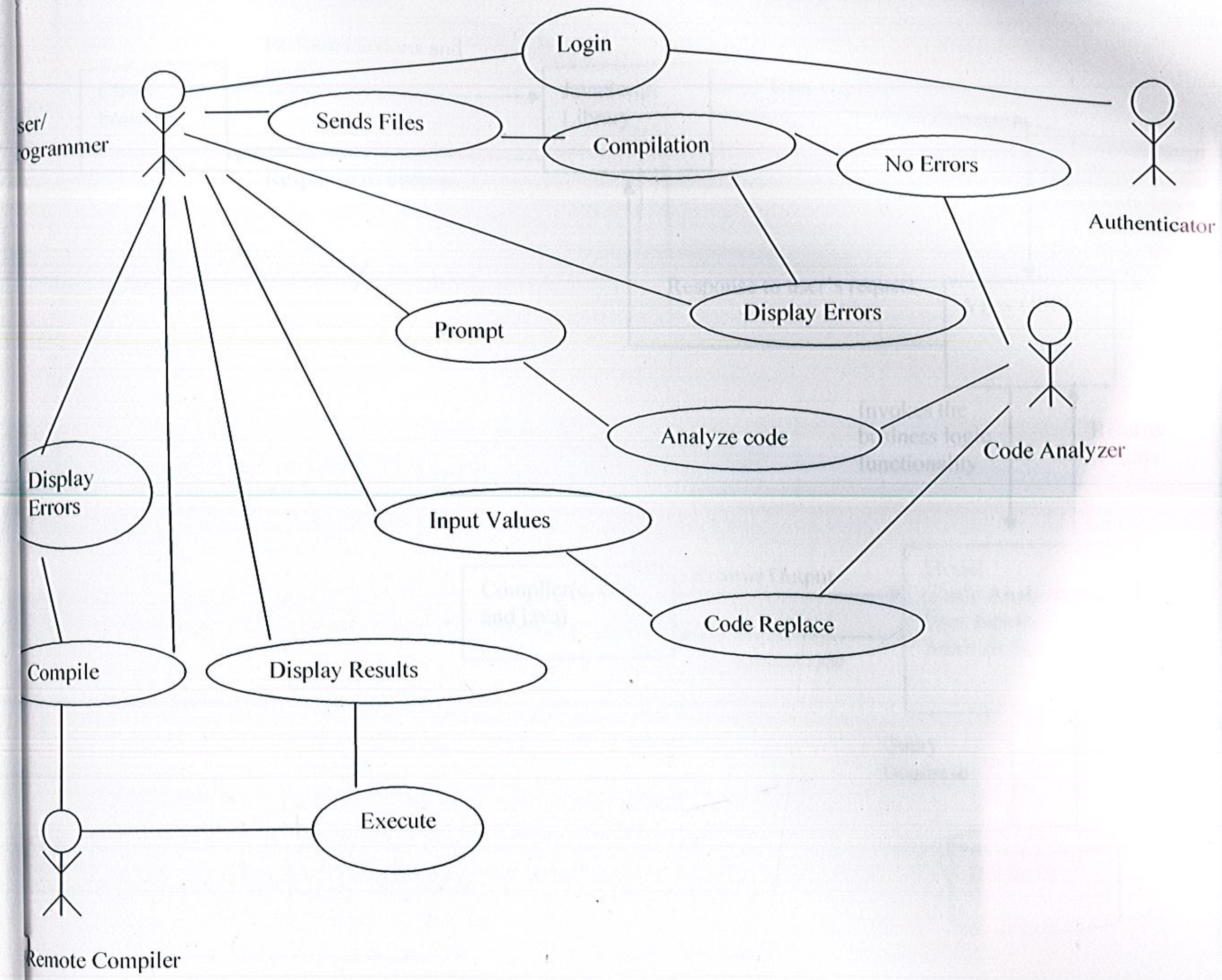
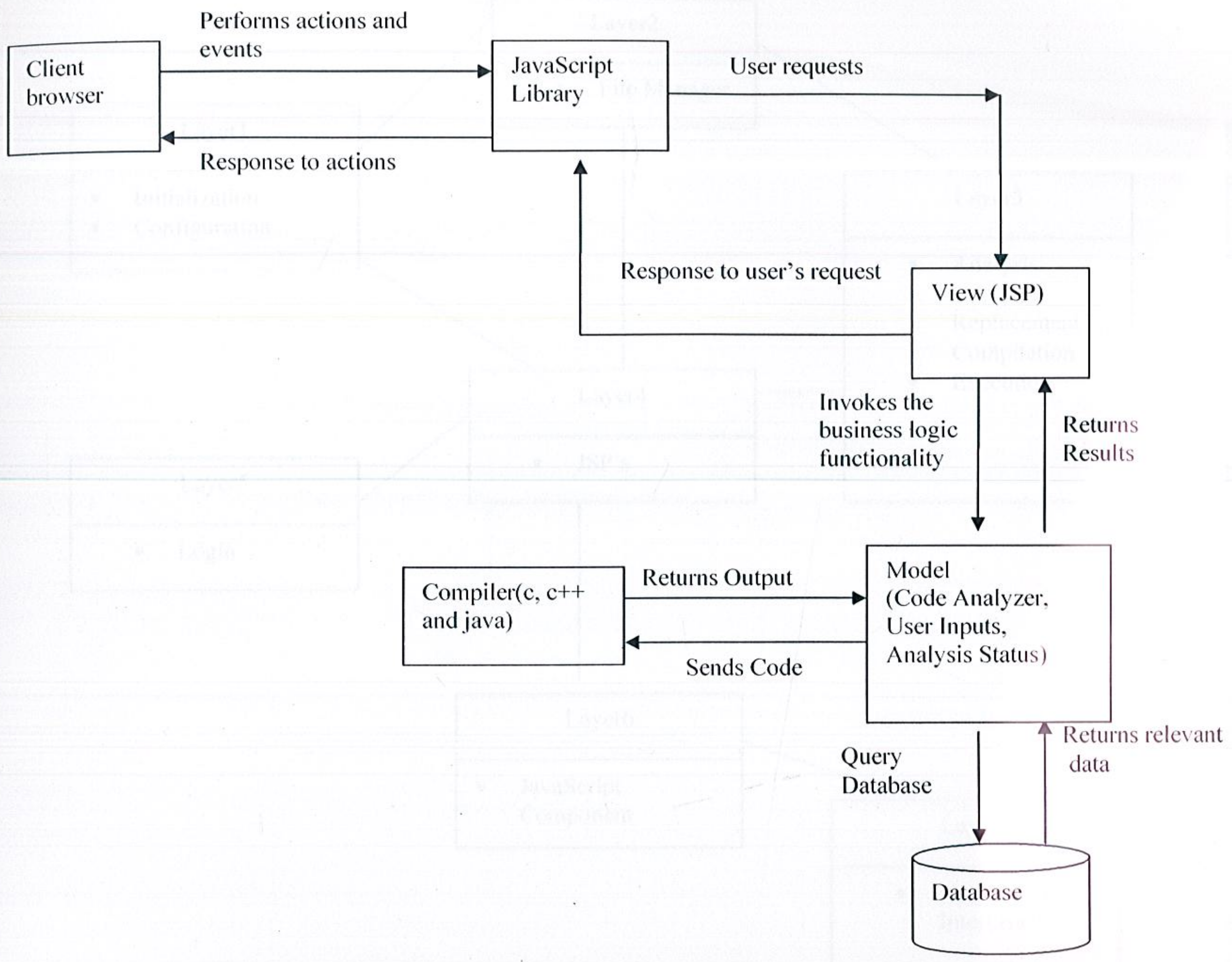


Fig 1 – Use Case Diagram

2.5 Architecture Design



Performs actions and events
Client browser
JavaScript Library
User requests
Response to actions
Response to user's request
View (JSP)
Invokes the business logic functionality
Returns Results
Compiler(c, c++ and java)
Returns Output
Sends Code
Model (Code Analyzer, User Inputs, Analysis Status)
Query Database
Returns relevant data
Database

Fig. 2 – Architecture Diagram

2.6 Layering and partitioning

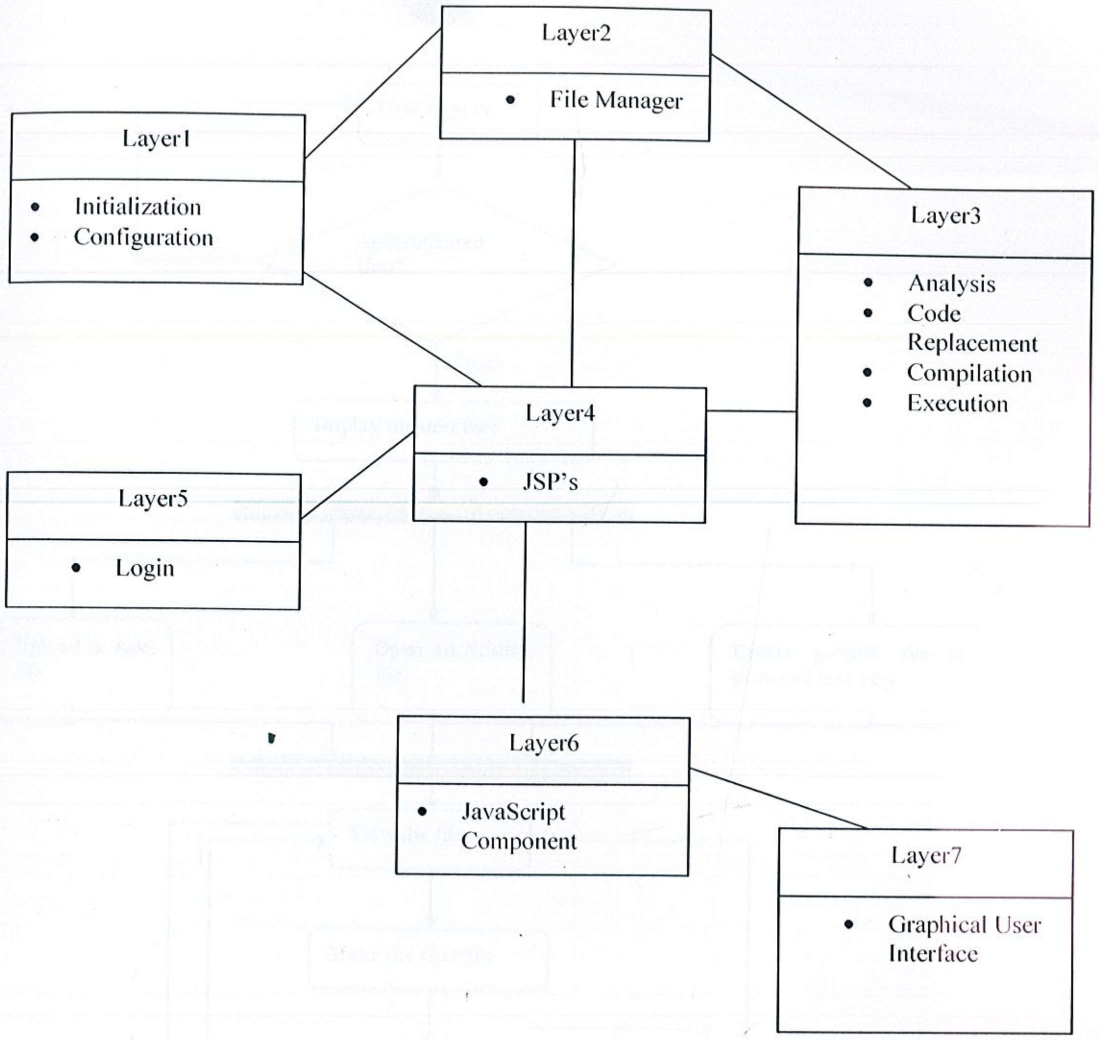
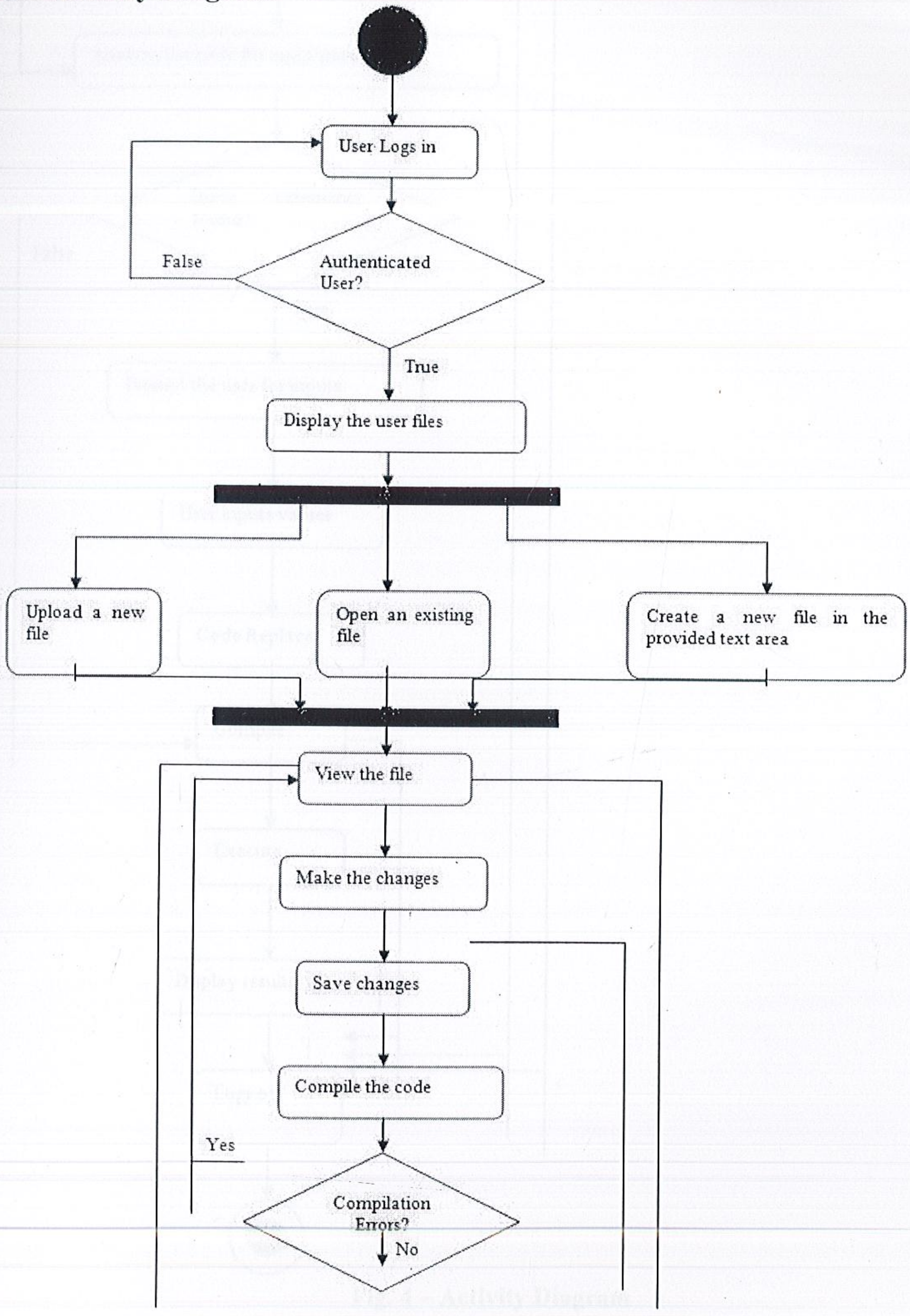


Fig. 3 – Layering and Partition

2.7 Activity Diagram



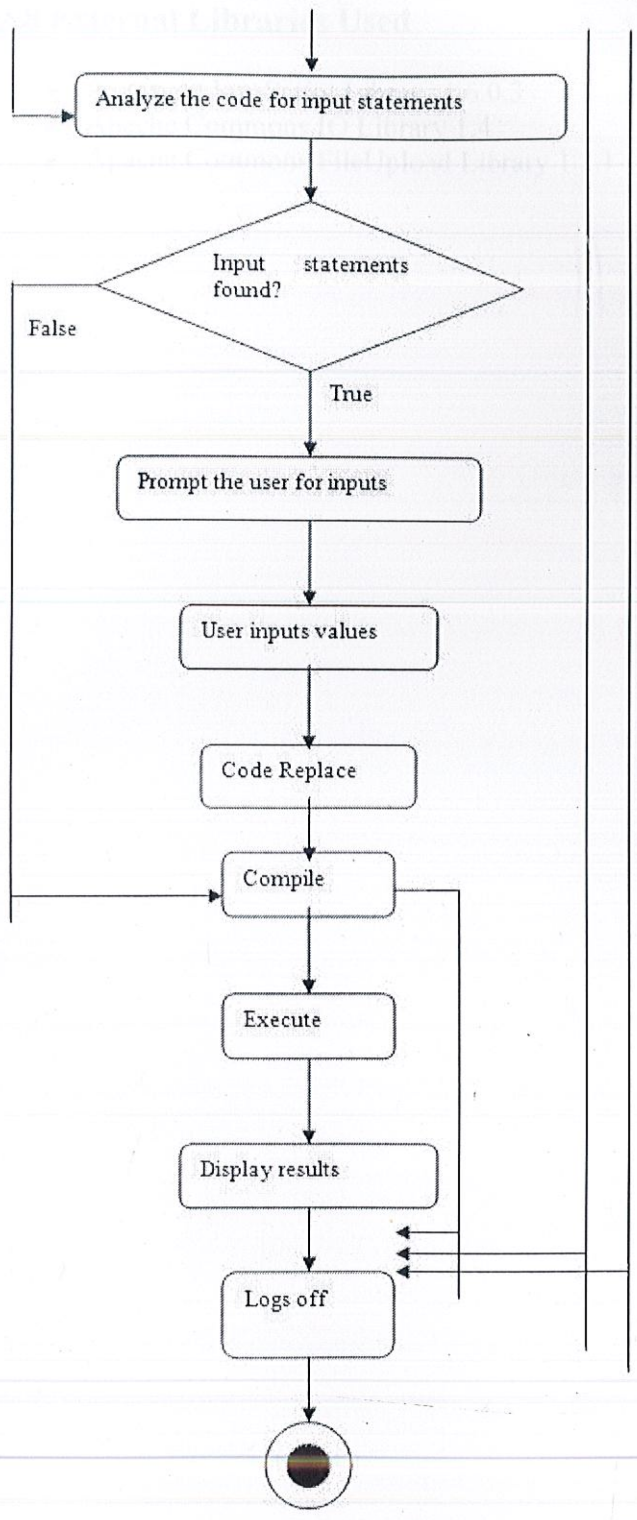


Fig. 4 – Activity Diagram

2.8 External Libraries Used

- Prototype JavaScript Library 1.6.0.3
- Apache Commons IO Library 1.4
- Apache Commons FileUpload Library 1.2.1

CHAPTER 3

DETAILED DESIGN

3.1 Implementation Elements

3.1.1 Login

Its specific implementation element is the database, to which the connection is established, that has users table through which authentication is done and has attributes:

- UserId
- Username
- Password

3.1.2 Graphical User Interface

Its various implementation elements are:

- Various File options such as:
  - Open File
  - Upload File
  - Create File
  - Save File
  - Compile File
  - Execute File
- Customized Directory structure creation
- Folder within a folder
- Files within a folder

3.1.3 File Manager

Its various implementation elements are:

- Get the list of files and directories corresponding to the user.
- Deletes temporary files created in the process of compilation.
- Modifies the source file.

3.1.4 Code Analyzer

Its various implementation elements are:

- Searching input statements in the source code file
- Replacing the source code with appropriate assignments
- Saving the modified source code for its compilation and execution.

3.1.5 Compiler

Its various implementation elements are:

- Compile the source code.
- Execute the compiled code.

3.1.6 JavaScript Component

Its various implementation elements are:

- Displays the user interface.

- Handles various events and actions of the user.
- Prompts the user for input values, if any.

3.1.7 Database

Its various implementation elements are:

- It stores the authorised user's details.
- It stores the configuration details of languages, compilers, source files, temporary files, executable files that are required to make the project portable.

3.2 Algorithm Design

3.2.1 Code Analysis

- Read the source file from selected by user for compilation.
- Search for all Input statements inside the program and make a list of matched occurrences along with the line of code where statement was found, input variable to receive the data into and the variable data type (language specific).
- Set this list of occurrences as a session object and prompt the user to enter values for each of the variable

3.2.2 Code Replacement

- Retrieve the session variable holding list of variables in input statements.
- Retrieve the values entered by user for each of the variable.
- Replace each input statement in the code by proper assignment statements for each variable.
- Original source file is stored in a temporary folder and changes are made to the file in original location.

3.2.3 Compilation

- File is compiled.
- In case compilation errors are found, a list of errors is returned.

- In case of no errors, all the modified source code is replaced by the original source file from temporary folder and all the temporary files are deleted.

3.3 Database Design

Our database has following tables:

- **Users**:
  Its attributes are:
  - userId: it is the user's id and also a primary key.
  - userName : it is the name of the user.
  - password: password of the user.

- **Languages**:
  Its attributes are:
  - languageId: it is the language's id.
  - languageName : it is the name of the language.
  - extension: it is the extension of the file and can be either c, cpp or java.

  The languageId and extension constitute the primary key.

- **TemporaryFiles**:
  Its attributes are:
  - languageId: it is the language's id.
  - extension: it is the extension of the file and can be either c, cpp or java.

  The languageId and extension constitute the primary key.

- **Compiler**:
  Its attributes are:
  - languageId: it is the language's id.
  - stepId: it constitutes the steps required to compile the source code.
  - compilerPath: it is the path required to compile the source code.

  The languageId and stepId constitute the primary key.

- **SourceFiles**:
  Its attributes are:
  - languageId: it is the language's id.
  - extension: it is the extension of the file and can be either c, cpp or java.

  The languageId and extension constitute the primary key.

3.4 Optimization of algorithms and data access

The optimization techniques used in the project are:

- Deletion of the intermediate files generated during compilation.
- The users' table is normalized up to 3NF.

3.5 Screen field validations and defaults

- **Login Screen**:
  The validations for the login screen are:
    - The userId, userName and password should not be blank.
    - The userId should only contain digits.
    - The password should be at least four characters.

  The login screen does not have any default values.

- **The User Input Screen**:
  The validations for user input screen are:
    - For char data type, the field can contain only 1 character
    - For numeric data types (int, float, double, etc), the field can contain only numeric values.
    - For all fields, blank field validations have to be done.

3.6 Special Processing Notes

3.6.1 Assumptions

- The remote system will not compile the programs that contain looping statements.
- The remote system will not compile the programs that contain goto statements.
- The remote system will not compile the programs that contain multiple statements in single line.
- The remote system will not compile the programs that contain Structures.

3.6.2 Limits

This project is limited to three programming languages, i.e, c, c++ and java. And to run a complete user application, the executables of all the related files (to the project) should have their executables present in the same folder.

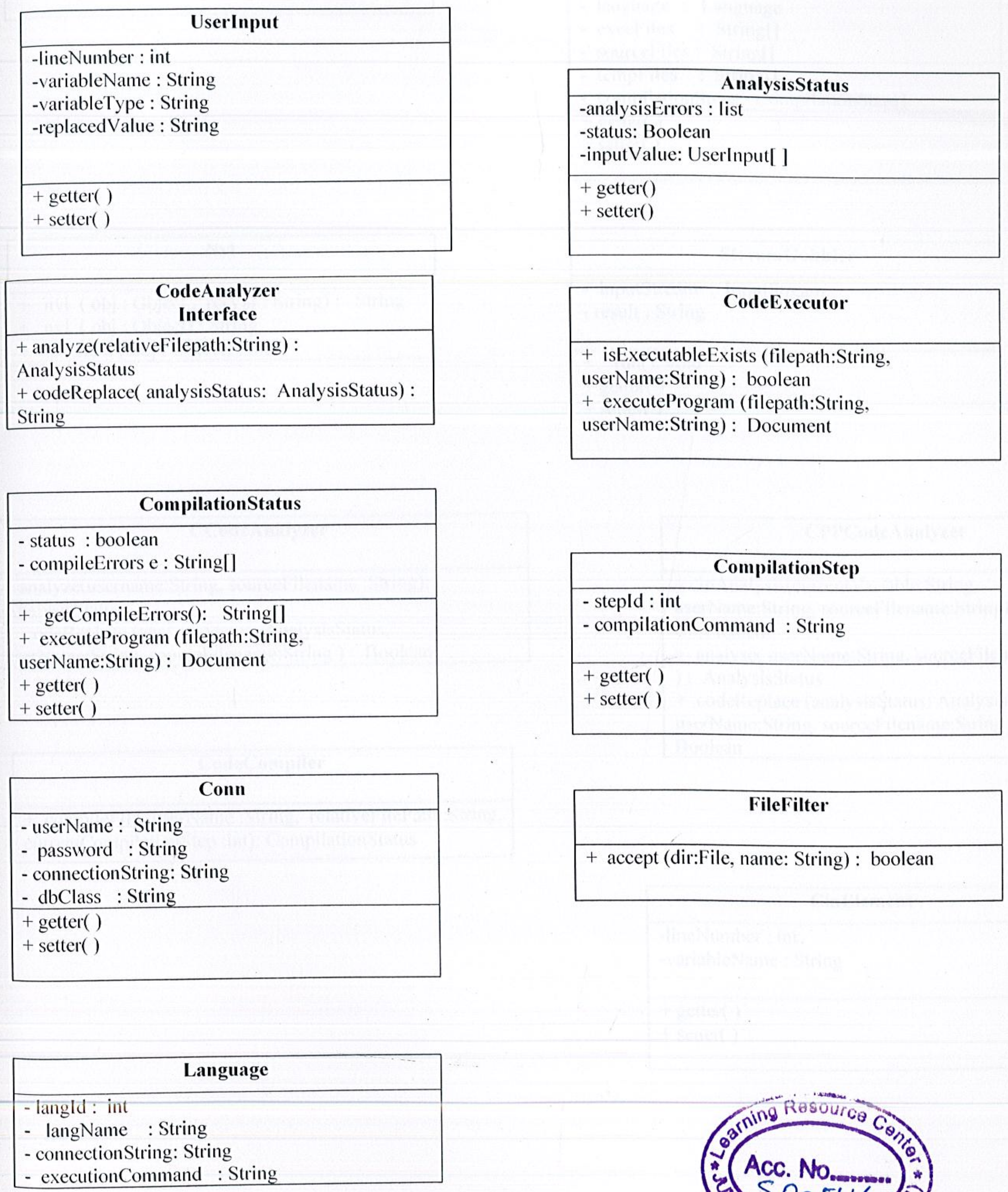
3.6.3 Exception Handling

Exception handling is done if:

- The source code contains loops.
- The source code contains goto statements.
- The source code contains structures.
- The source code contains multiple statements separated by semicolon (;).

CHAPTER 4

CLASS DIAGRAMS



```
UserInput
-lineNumber : int
-variableName : String
-variableType : String
-replacedValue : String
+ getter( )
+ setter( )
```

```
AnalysisStatus
-analysisErrors : list
-status: Boolean
-inputValue: UserInput[ ]
+ getter()
+ setter()
```

```
CodeAnalyzer
Interface
+ analyze(relativeFilepath:String) :
AnalysisStatus
+ codeReplace( analysisStatus:  AnalysisStatus) :
String
```

```
CodeExecutor
+  isExecutableExists (filepath:String,
userName:String) :  boolean
+  executeProgram (filepath:String,
userName:String) :  Document
```

```
CompilationStatus
- status  : boolean
- compileErrors e : String[]
+  getCompileErrors():   String[]
+  executeProgram (filepath:String,
userName:String) :  Document
+ getter( )
+ setter( )
```

```
CompilationStep
- stepId : int
- compilationCommand  : String
+ getter( )
+ setter( )
```

```
Conn
- userName :  String
-  password   : String
- connectionString: String
-  dbClass   : String
+ getter( )
+ setter( )
```

```
FileFilter
+  accept (dir:File, name: String) :  boolean
```

```
Language
- langId :  int
-   langName    : String
- connectionString: String
-  executionCommand   : String
```

+ getter( )
+ setter( )

LanguageConfig

- language : Language
- execFiles : String[]
- sourceFiles : String[]
- tempFiles : String[]
- compilationSteps: CompilationStep[]

+ getter( )
+ setter( )

Nvl

+ nvl (obj : Object , retVal : String) : String
+ nvl (obj : Object) : String
+ parseInt (obj : Object) : int

StreamGobbler

- inputStream : InputStream
- result : String

+ run(): void
+ getter( )
+ setter( )

CCodeAnalyzer

+ analyze(username:String, sourceFilename :String): AnalysisStatus
+ codeReplace (analysisStatus: AnalysisStatus, userName:String, sourceFilename:String) : Boolean

CPPCodeAnalyzer

+ cinAnalysis(currentVariable:String, userName:String, sourceFilename:String) : CinElement
+ analyse(userName:String, sourceFilename:String) : AnalysisStatus
+ codeReplace (analysisStatus: AnalysisStatus, userName:String, sourceFilename:String) : Boolean

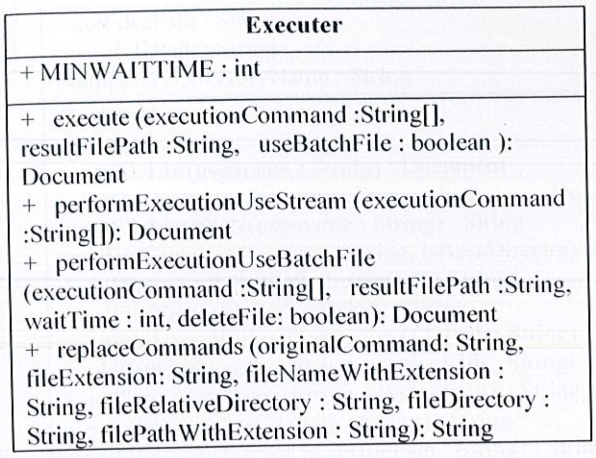
CodeCompiler

+ compileFile (userName :String, relativeFilePath :String, currentCompilationStep :int): CompilationStatus

CinElement

-lineNumber : int
-variableName : String

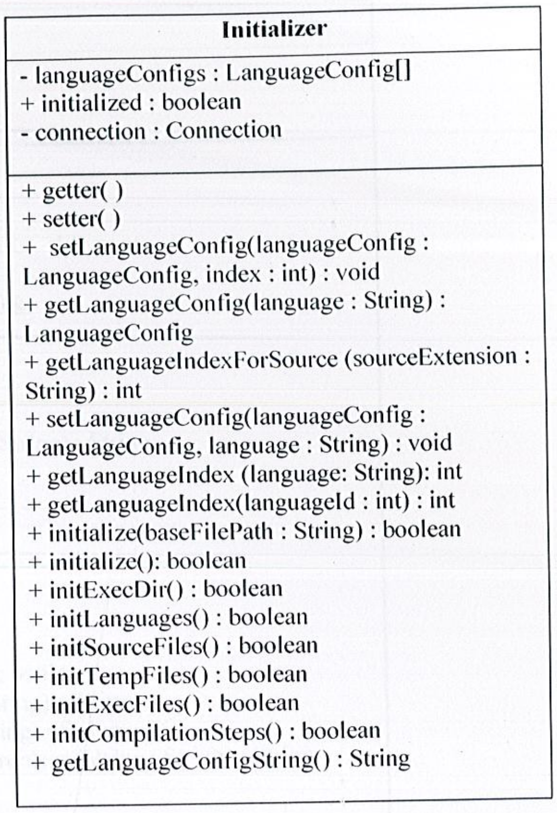
+ getter( )
+ setter( )



Executer

+ MINWAITTIME : int

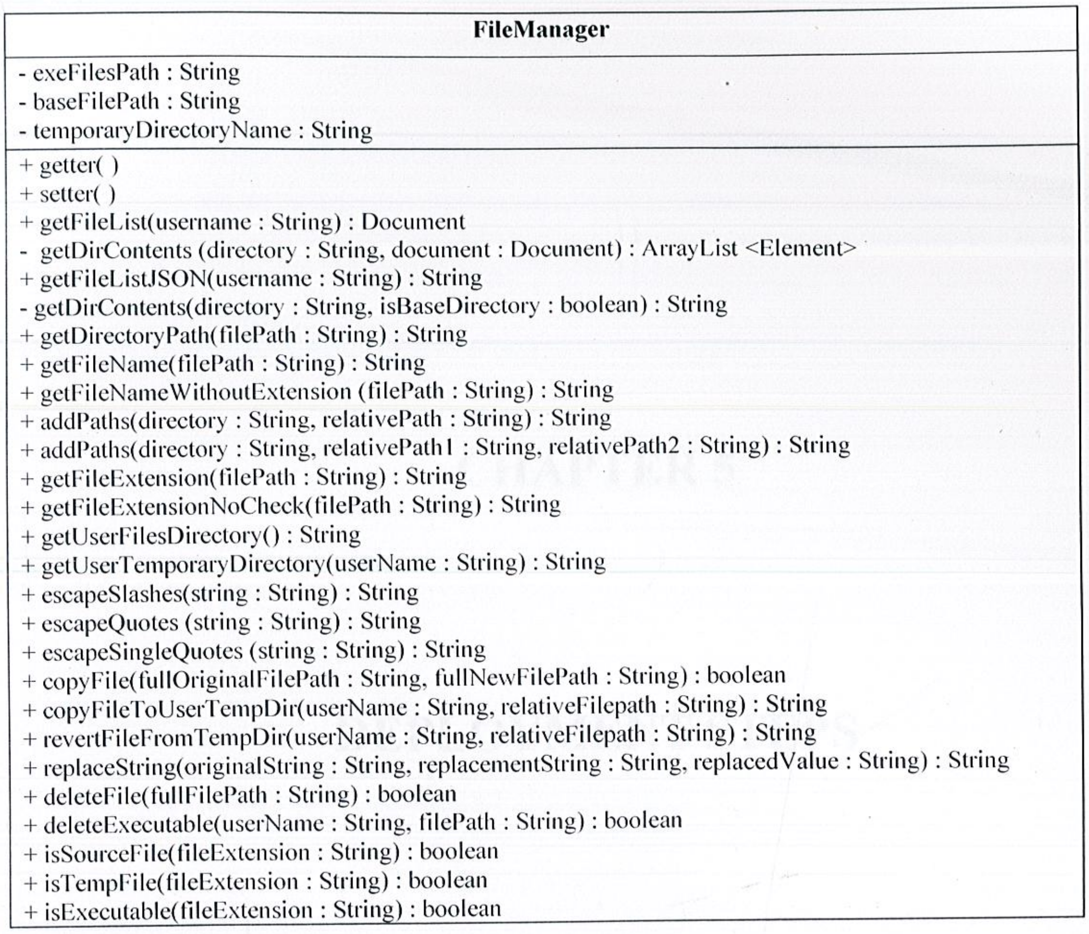
+ execute (executionCommand :String[], resultFilePath :String, useBatchFile : boolean): Document
+ performExecutionUseStream (executionCommand :String[]): Document
+ performExecutionUseBatchFile (executionCommand :String[], resultFilePath :String, waitTime : int, deleteFile: boolean): Document
+ replaceCommands (originalCommand: String, fileExtension: String, fileNameWithExtension : String, fileRelativeDirectory : String, fileDirectory : String, filePathWithExtension : String): String



Initializer

- languageConfigs : LanguageConfig[]
+ initialized : boolean
- connection : Connection

+ getter( )
+ setter( )
+ setLanguageConfig(languageConfig : LanguageConfig, index : int) : void
+ getLanguageConfig(language : String) : LanguageConfig
+ getLanguageIndexForSource (sourceExtension : String) : int
+ setLanguageConfig(languageConfig : LanguageConfig, language : String) : void
+ getLanguageIndex (language: String): int
+ getLanguageIndex(languageId : int) : int
+ initialize(baseFilePath : String) : boolean
+ initialize(): boolean
+ initExecDir() : boolean
+ initLanguages() : boolean
+ initSourceFiles() : boolean
+ initTempFiles() : boolean
+ initExecFiles() : boolean
+ initCompilationSteps() : boolean
+ getLanguageConfigString() : String



FileManager

```
- exeFilesPath : String
- baseFilePath : String
- temporaryDirectoryName : String
+ getter( )
+ setter( )
+ getFileList(username : String) : Document
-  getDirContents (directory : String, document : Document) : ArrayList <Element>
+ getFileListJSON(username : String) : String
- getDirContents(directory : String, isBaseDirectory : boolean) : String
+ getDirectoryPath(filePath : String) : String
+ getFileName(filePath : String) : String
+ getFileNameWithoutExtension (filePath : String) : String
+ addPaths(directory : String, relativePath : String) : String
+ addPaths(directory : String, relativePath1 : String, relativePath2 : String) : String
+ getFileExtension(filePath : String) : String
+ getFileExtensionNoCheck(filePath : String) : String
+ getUserFilesDirectory() : String
+ getUserTemporaryDirectory(userName : String) : String
+ escapeSlashes(string : String) : String
+ escapeQuotes (string : String) : String
+ escapeSingleQuotes (string : String) : String
+ copyFile(fullOriginalFilePath : String, fullNewFilePath : String) : boolean
+ copyFileToUserTempDir(userName : String, relativeFilepath : String) : String
+ revertFileFromTempDir(userName : String, relativeFilepath : String) : String
+ replaceString(originalString : String, replacementString : String, replacedValue : String) : String
+ deleteFile(fullFilePath : String) : boolean
+ deleteExecutable(userName : String, filePath : String) : boolean
+ isSourceFile(fileExtension : String) : boolean
+ isTempFile(fileExtension : String) : boolean
+ isExecutable(fileExtension : String) : boolean
```

CHAPTER 5

DEPLOYMENT STEPS

5.1 Deploying to server

- **Deployment to Apache Tomcat v5.5 on Eclipse IDE**
  - Open Eclipse
  - Go to File - > Import.
  - Select Web - > WAR File
  - Click Next.
  - Click browse against WAR File to select the Remote Compiler WAR file.
  - Click New against Target Runtime and configure a new Apache Tomcat 5.5 server.
  - In select JARs, keep all JAR files unchecked.
  - Click Finish.
  - Go to Run - > Run As - > Run on Server.
  - Access Remote Compiler through web browser.

5.2 Environment Setup for Visual C++ Compilers

5.2.1 Microsoft Visual Studio 2005

- Go to your Visual Studio installation directory, usually *C:\Program Files\Microsoft Visual Studio 8*
- Go to Common Tools folder generally in *Common7\Tools*
- Right click on VCVARS32.BAT and select Edit.
- Now go to *Start - > Control Panel - > System*
- Go to **Advanced** Tab and click on **Environment Variables.**
- Now, you'll need to add environment variables for current user as given in **VCVARS32.BAT** file.
- Corresponding to each **Set variableName = variableValue** command in batch file, Add a new system variable for current user. (Note that Click on "New" button below "User variables for <<CurrentUser>>" instead of "New" button below "System variables". Make sure to replace all %VAR% in variables values with actual value for that variable as shown in list of variables e.g. If command is **"Set Abc = %windir%\system32"** and value of Windir in system variables is **C:\Windows**, the actual value to be set for environment variable Abc will be **C:\Windows\system32.** Similarly, if command is **Set Def = C:\Program Files\Adobe; C:\Program Files\Java; %Def** and variable Def already exists in list of variables and has value **C:\Program Files\Java; C:\Program Files\Infosys**, the actual value to be set for environment variable Def will be **C:\Program Files;C:\Program Files\Adobe; C:\Program Files\Java;**. Also, if there are some statements like **if "%OS%" == "Windows_NT" Set VarName=VarValue1** and **if "%OS%" == "" Set VarName=VarValue2**, use *VarValue1* as the value for *VarName*
- Typical system variables and their values for VC2005 are:
  - DevEnvDir - **C:\Program Files\Microsoft Visual Studio 8\Common7\IDE**

- FrameworkDir - **C:\WINDOWS\Microsoft.NET\Framework**
- FrameworkSDKDir - **C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0**
- FrameworkVersion - **v2.0.50727**
- INCLUDE - **C:\Program Files\Microsoft Visual Studio 8\VC\ATLMFC\INCLUDE;C:\Program Files\Microsoft Visual Studio 8\VC\INCLUDE;C:\Program Files\Microsoft Visual Studio 8\VC\PlatformSDK\include;C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\include;**
- LIB - **C:\Program Files\Microsoft Visual Studio 8\VC\ATLMFC\LIB;C:\Program Files\Microsoft Visual Studio 8\VC\LIB;C:\Program Files\Microsoft Visual Studio 8\VC\PlatformSDK\lib;C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\lib;**
- PATH - **C:\Program Files\Microsoft Visual Studio 8\Common7\IDE;C:\Program Files\Microsoft Visual Studio 8\VC\BIN;C:\Program Files\Microsoft Visual Studio 8\Common7\Tools;C:\Program Files\Microsoft Visual Studio 8\Common7\Tools\bin;C:\Program Files\Microsoft Visual Studio 8\VC\PlatformSDK\bin;C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\bin;C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727;C:\Program Files\Microsoft Visual Studio 8\VC\VCPackages;**
- VCINSTALLDIR - **C:\Program Files\Microsoft Visual Studio 8\VC**
- VSINSTALLDIR - **C:\Program Files\Microsoft Visual Studio 8**
- LIBPATH - **C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727; C:\Program Files\Microsoft Visual Studio 8\VC\ATLMFC\LIB**

5.2.2 Microsoft Visual Studio 98

- Go to your Visual C++ installation directory, usually *C:\Program Files\Microsoft Visual Studio\VC98\*
- Go to *bin* folder
- Right click on VCVARS32.BAT and select Edit.
- Now go to *Start - > Control Panel - > System*
- Go to **Advanced** Tab and click on **Environment Variables.**
- Now, you'll need to add environment variables for current user as given in **VCVARS32.BAT** file.
- Corresponding to each **Set variableName = variableValue** command in batch file, Add a new system variable for current user. (Note that Click on "New" button below "User variables for <<CurrentUser>>" instead of "New" button below "System variables". Make sure to replace all %VAR% in variables values with actual value for that variable as shown in list of variables e.g. If command is **"Set Abc = %windir%\system32"** and value of Windir in system variables is **C:\Windows**, the actual value to be set for environment variable Abc will be **C:\Windows\system32.** Similarly, if command is **Set Def = C:\Program Files\Adobe; C:\Program Files\Java; %Def%** and variable Def already exists in list of variables and has value **C:\Program Files\Java; C:\Program Files\Infosys**, the actual value to be set for

environment variable Def will be **C:\Program Files;C:\Program Files\Adobe; C:\Program Files\Java;**. Also, if there are some statements like **if "%OS%" == "Windows_NT" Set VarName=VarValue1** and **if "%OS%" == "" Set VarName=VarValue2**, use *VarValue1* as the value for *VarName*

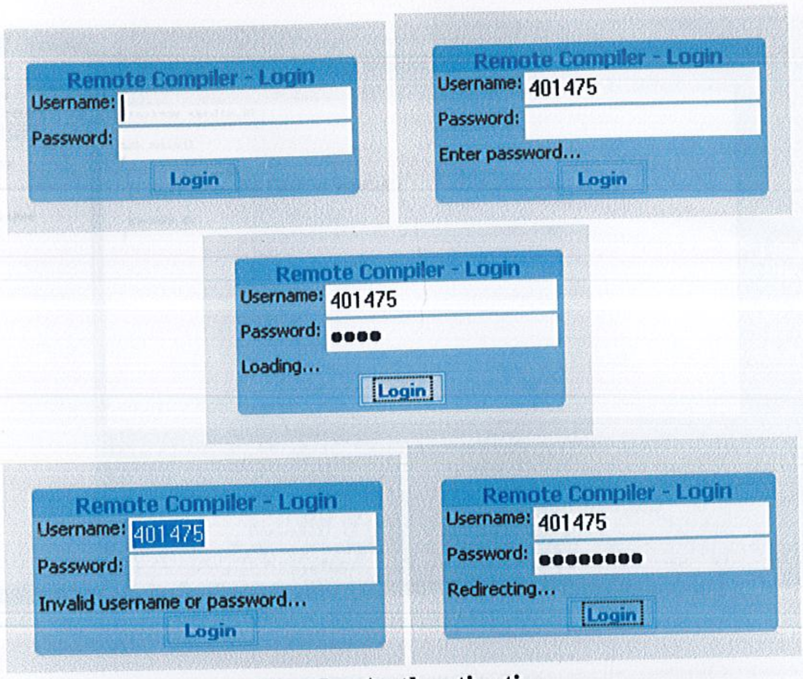
- Typical system variables and their values for VC98 are:
  - VSCommonDir - **C:\PROGRA~1\MICROS~3\Common**
  - MSDevDir - **C:\PROGRA~1\MICROS~3\Common\msdev98**
  - MSVCDir - **C:\PROGRA~1\MICROS~3\VC98**
  - VcOsDir=**WINNT**
  - Path- **C:\PROGRA~1\MICROS~3\Common\msdev98\BIN;C:\PROGRA~1\MICROS~3\VC98\BIN;C:\PROGRA~1\MICROS~3\Common\TOOLS\WINNT; C:\PROGRA~1\MICROS~3\Common\TOOLS;**
  - Include- **C:\PROGRA~1\MICROS~3\VC98\ATL\INCLUDE;C:\PROGRA~1\MICROS~3\VC98\INCLUDE;C:\PROGRA~1\MICROS~3\VC98\MFC\INCLUDE;**
  - Lib- **C:\PROGRA~1\MICROS~3\VC98\LIB;C:\PROGRA~1\MICROS~3\VC98\MFC\LIB;**

5.3 Installation verification

Restart the machine and run the application server. Verify Visual studio configuration by going to *Start - > Run*, type *cmd*, Press *OK*. Type cl.exe and press enter. If the output is similar to: **Microsoft (R) 32-bit C/C++ Optimizing compiler**, your installation was successful.

CHAPTER 6

SCREENSHOTS


Remote Compiler - Login
Username:
Password:

Remote Compiler - Login
Username: 401475
Password:
Enter password...

Remote Compiler - Login
Username: 401475
Password: ●●●●
Loading...

Remote Compiler - Login
Username: 401475
Password:
Invalid username or password...

Remote Compiler - Login
Username: 401475
Password: ●●●●●●●●●
Redirecting...

Fig. 5 – Authentication

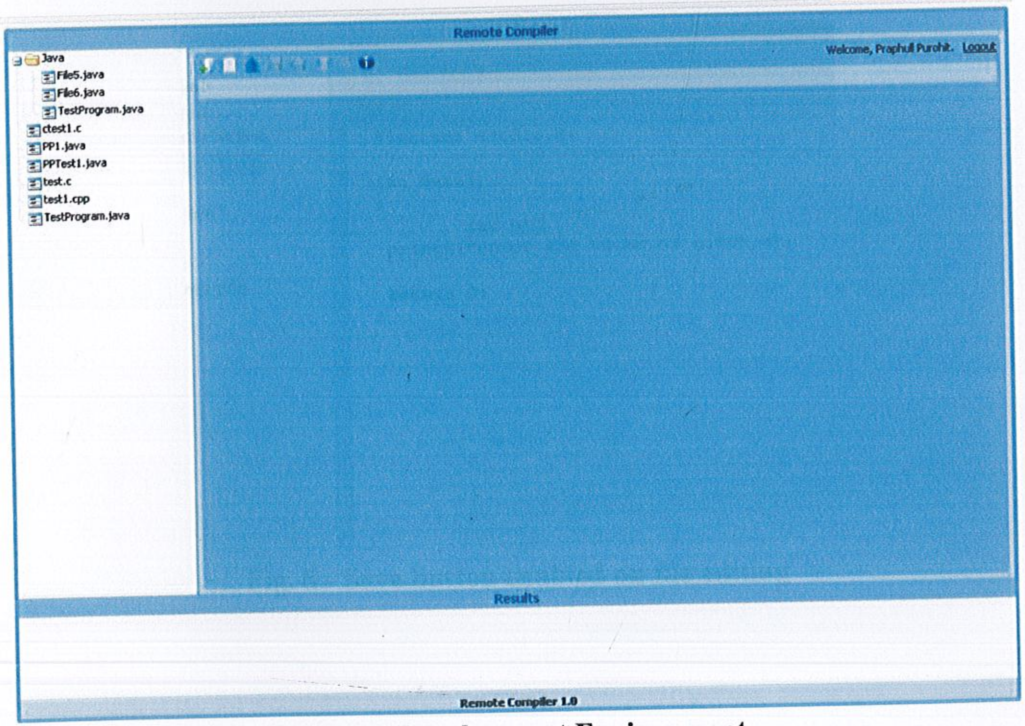
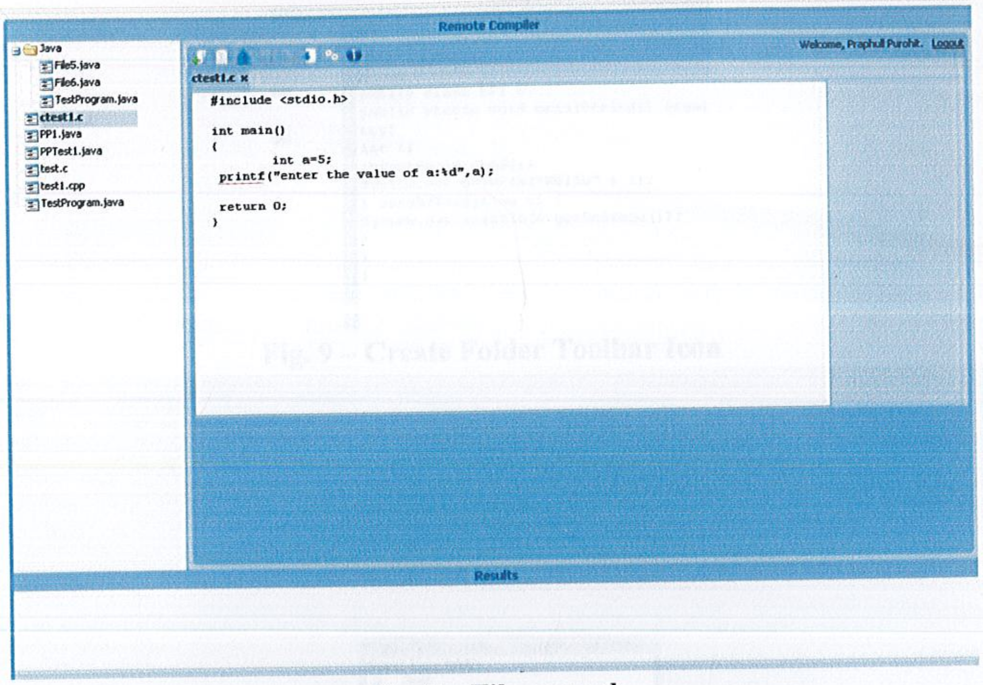
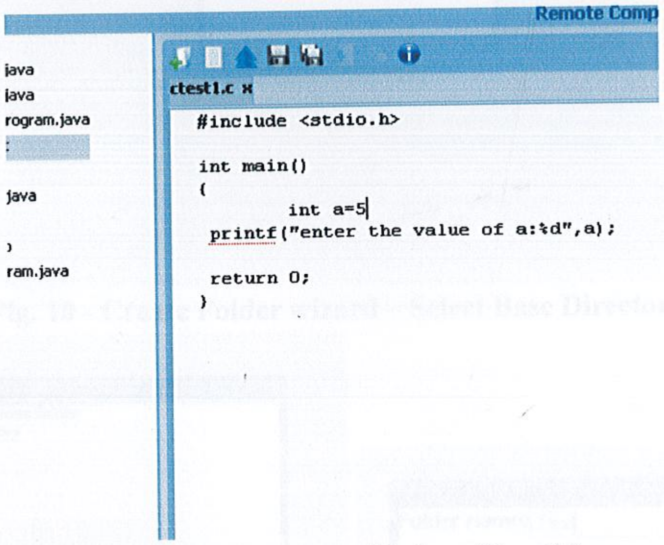


Fig. 6 – Development Environment



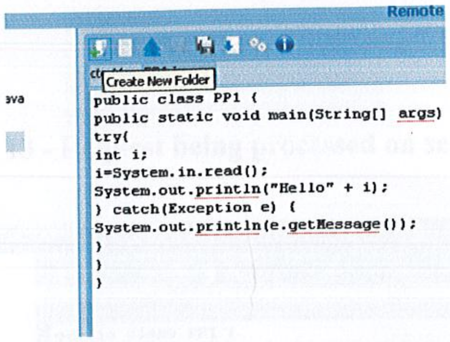
```
#include <stdio.h>

int main()
{
        int a=5;
 printf("enter the value of a:%d",a);

 return 0;
}
```

Fig. 7 – File opened



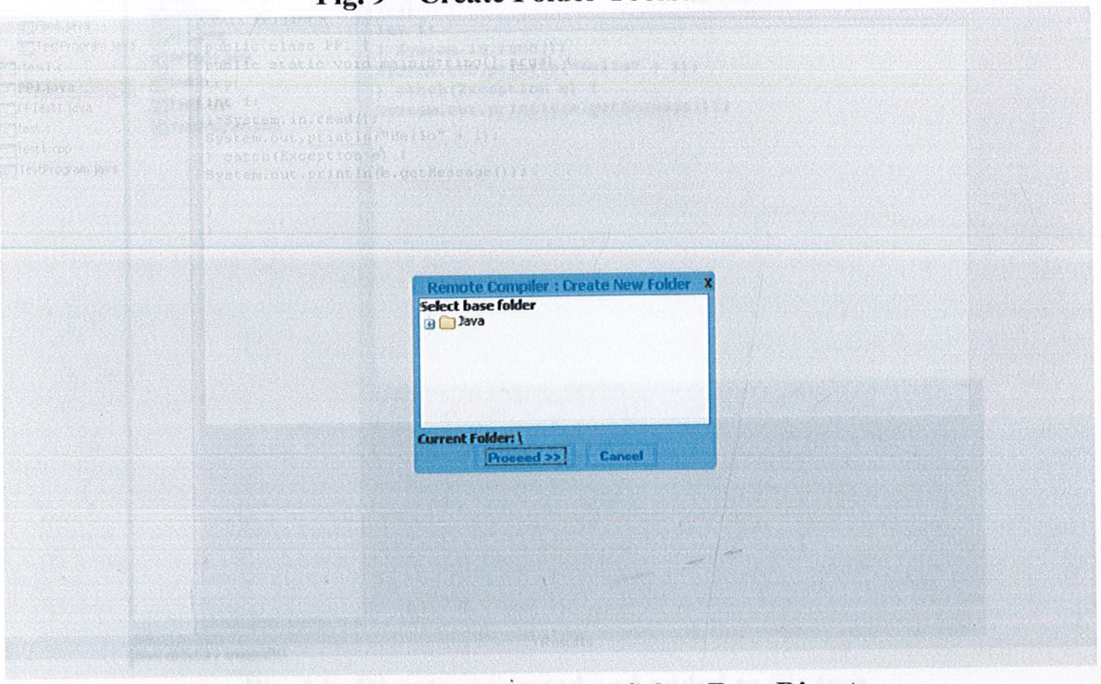
```
#include <stdio.h>

int main()
{
        int a=5
 printf("enter the value of a:%d",a);

 return 0;
}
```

Fig. 8 - Save button enabled on file editing



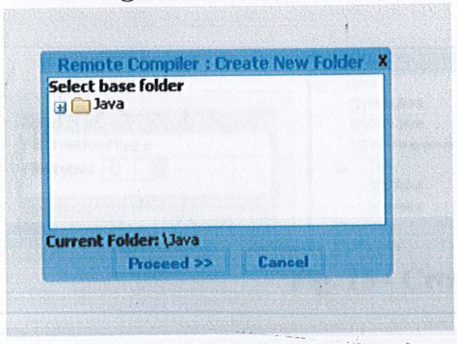
```
public class PP1 {
public static void main(String[] args)
try{
int i;
i=System.in.read();
System.out.println("Hello" + i);
} catch(Exception e) {
System.out.println(e.getMessage());
}
}
}
```

Fig. 9 – Create Folder Toolbar Icon

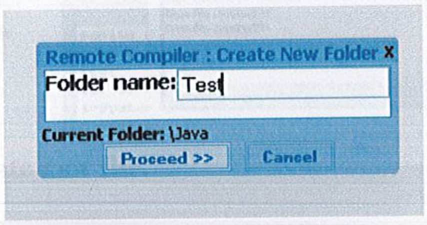


Current Folder: \

Fig. 10 - Create Folder wizard – Select Base Directory



Current Folder: \Java



Folder name: Test

Current Folder: \Java

Fig. 11 - Create Folder wizard – Base Directory Selected

Fig. 12 - Create Folder wizard – Enter new folder name

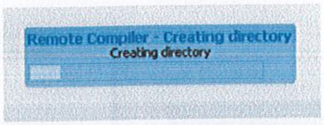
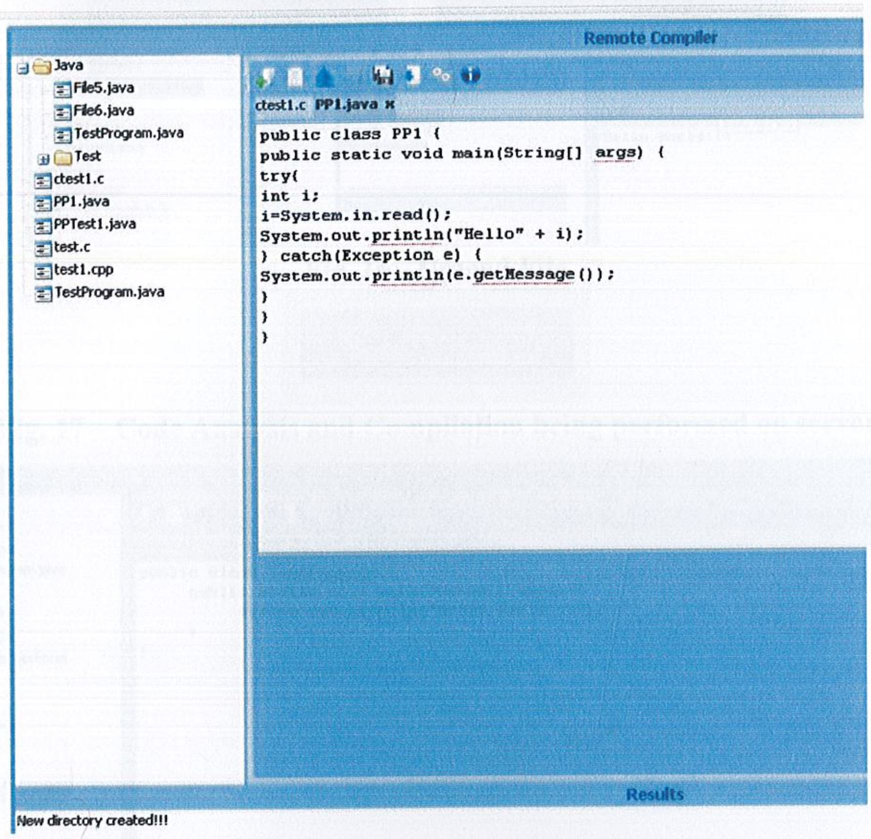


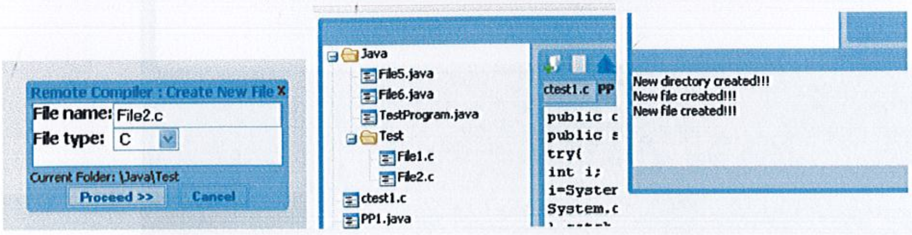
Fig. 13 - Request being processed on server



```
public class PP1 {
public static void main(String[] args) {
try{
int i;
i=System.in.read();
System.out.println("Hello" + i);
} catch(Exception e) {
System.out.println(e.getMessage());
}
}
}
```

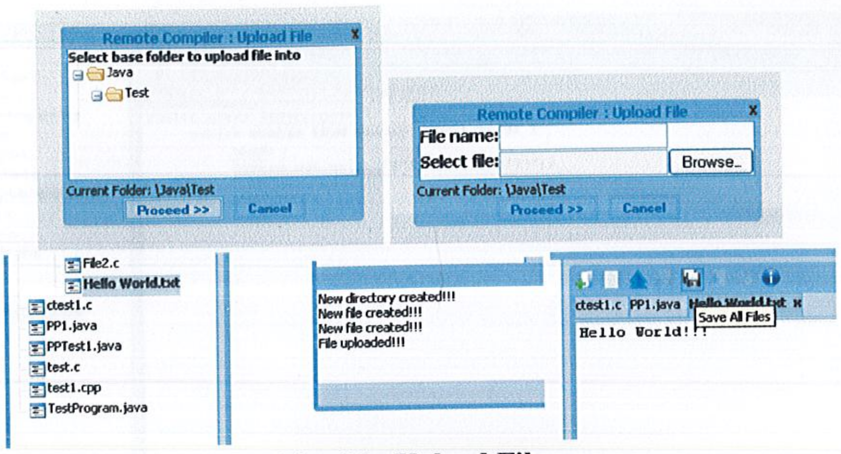
```
New directory created!!!
```

Fig. 14 - Directory created and added to file tree



```
New directory created!!!
New file created!!!
New file created!!!
```

Fig. 15 - Create new file.



```
New directory created!!!
New file created!!!
New file created!!!
File uploaded!!!
```

```
Hello World!!!
```

Fig. 16 - Upload File

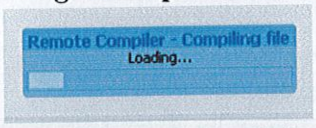
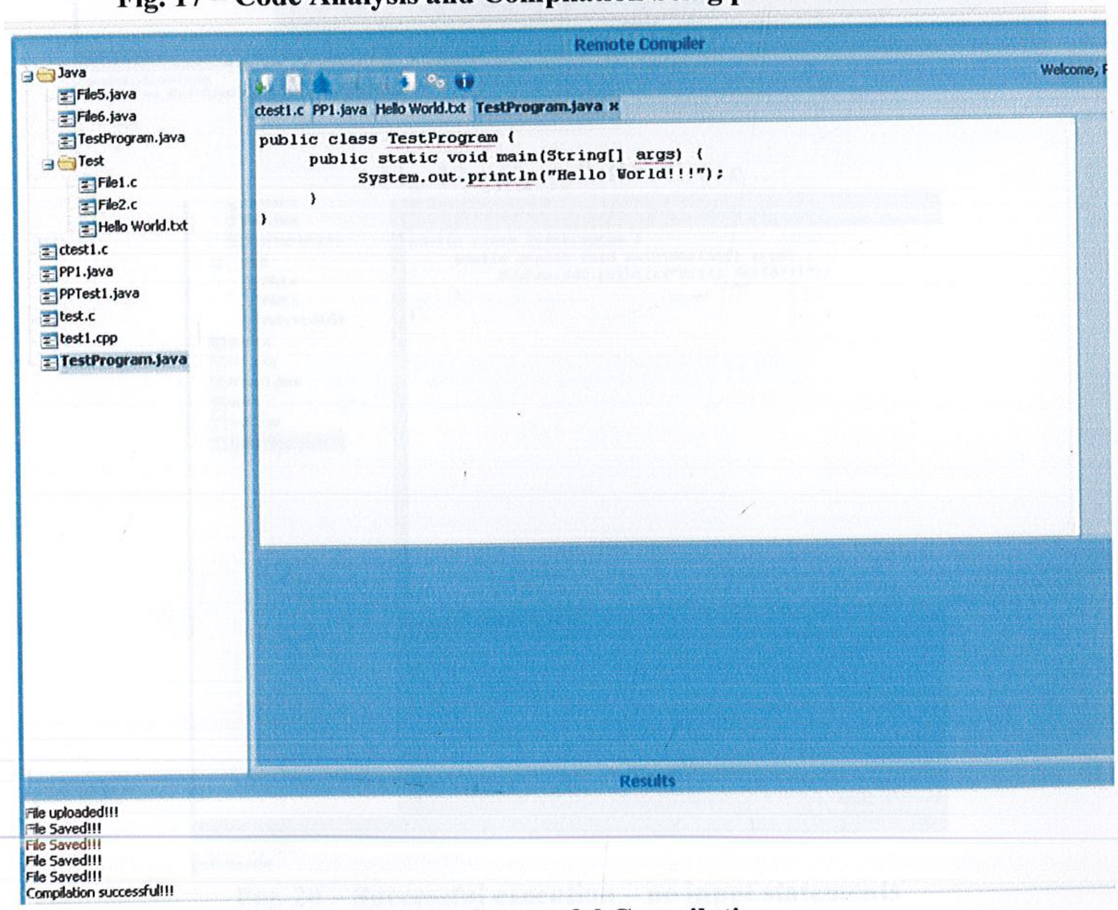


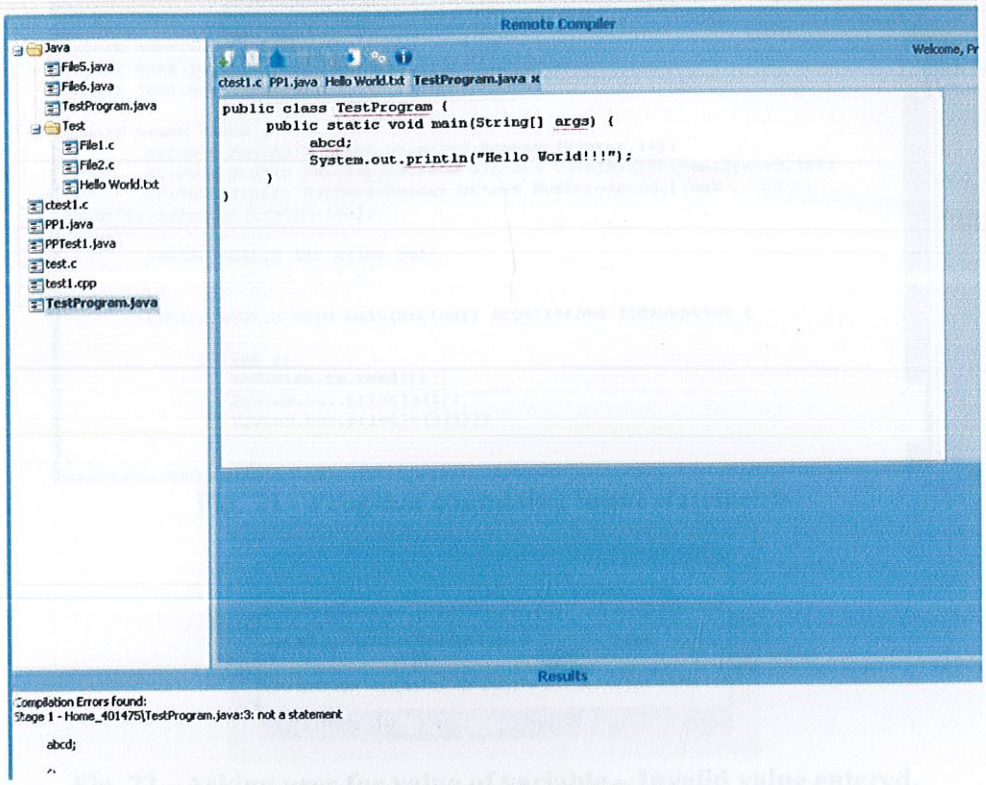
Fig. 17 – Code Analysis and Compilation being performed on server



```
public class TestProgram {
      public static void main(String[] args) {
            System.out.println("Hello World!!!");
      }
}
```

```
File uploaded!!!
File Saved!!!
File Saved!!!
File Saved!!!
File Saved!!!
Compilation successful!!!
```

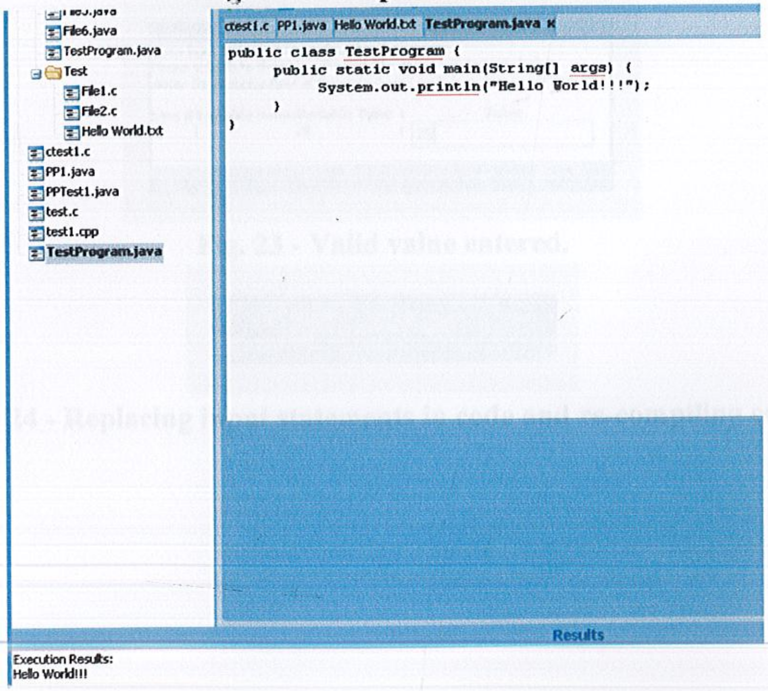
Fig. 18 – Successful Compilation.



```
public class TestProgram {
        public static void main(String[] args) {
                abcd;
                System.out.println("Hello World!!!");
        }
}
```

```
Compilation Errors found:
Stage 1 - Home_401475\TestProgram.java:3: not a statement

        abcd;

        ^
```

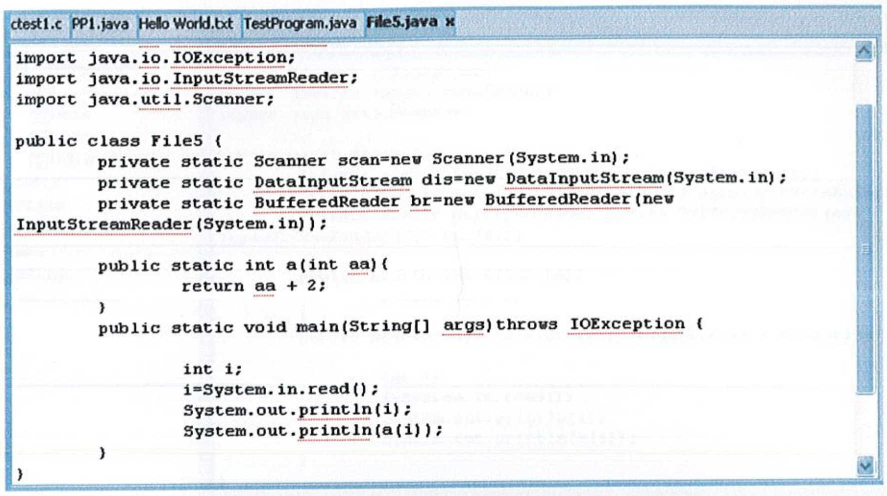
Fig. 19 – Compilation Error



```
public class TestProgram {
        public static void main(String[] args) {
                System.out.println("Hello World!!!");
        }
}
```

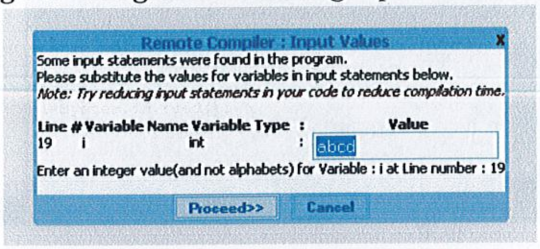
```
Execution Results:
Hello World!!!
```

Fig. 20 – Successful execution – no input statements


ctest1.c PP1.java Hello World.txt TestProgram.java File5.java x

```
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Scanner;

public class File5 {
        private static Scanner scan=new Scanner(System.in);
        private static DataInputStream dis=new DataInputStream(System.in);
        private static BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));

        public static int a(int aa){
                return aa + 2;
        }
        public static void main(String[] args)throws IOException {

                int i;
                i=System.in.read();
                System.out.println(i);
                System.out.println(a(i));
        }
}
```

Fig. 21 - Program containing input statements

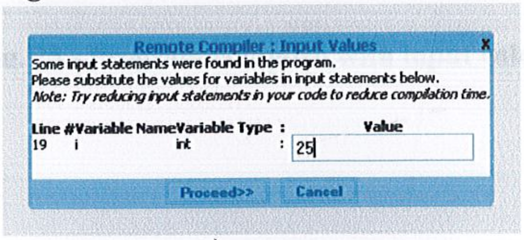


Some input statements were found in the program.
Please substitute the values for variables in input statements below.
Note: Try reducing input statements in your code to reduce compilation time.

Line #	Variable Name	Variable Type	:	Value
19	i	int	:	abcd

Enter an integer value(and not alphabets) for Variable : i at Line number : 19

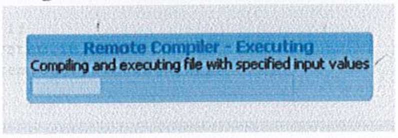
Fig. 22 - Asking user for value of variable – Invalid value entered.



Some input statements were found in the program.
Please substitute the values for variables in input statements below.
Note: Try reducing input statements in your code to reduce compilation time.

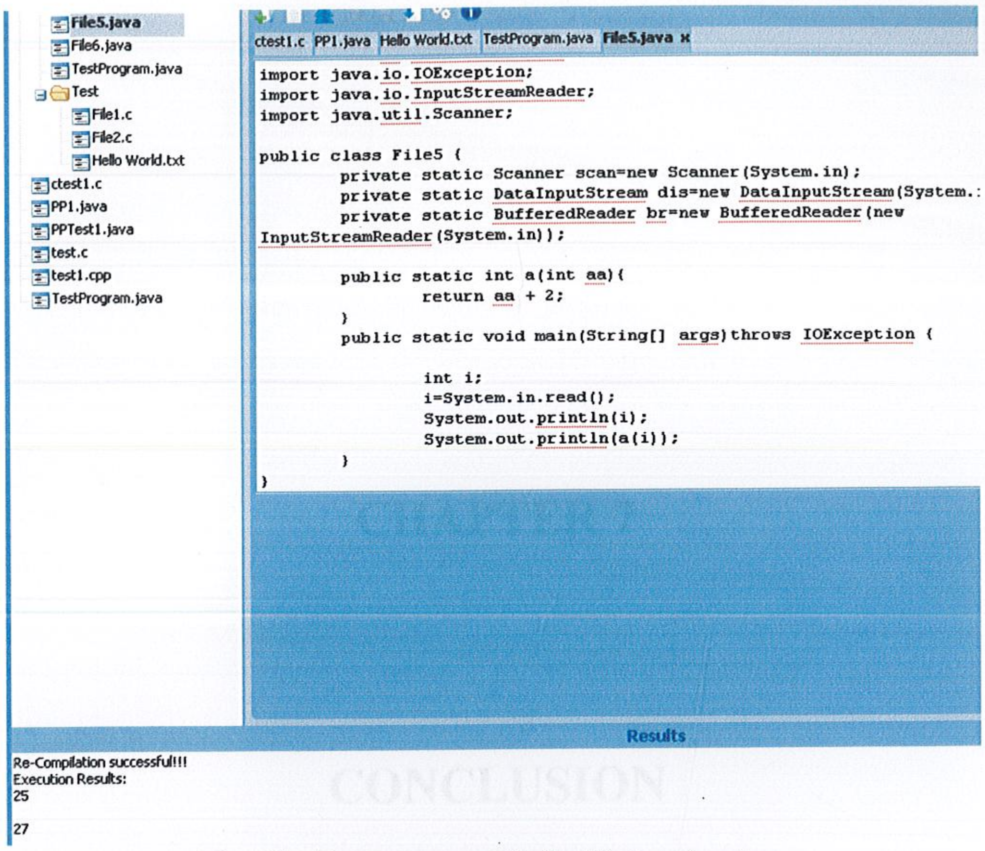
Line #	Variable Name	Variable Type	:	Value
19	i	int	:	25

Fig. 23 - Valid value entered.



Compiling and executing file with specified input values

Fig. 24 - Replacing input statements in code and re-compiling code.



```
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Scanner;

public class File5 {
        private static Scanner scan=new Scanner(System.in);
        private static DataInputStream dis=new DataInputStream(System.:
        private static BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));

        public static int a(int aa){
                return aa + 2;
        }
        public static void main(String[] args)throws IOException {

                int i;
                i=System.in.read();
                System.out.println(i);
                System.out.println(a(i));
        }
}
```

```
Re-Compilation successful!!!
Execution Results:
25

27
```

Fig. 25 - Program compiled with input value.

CHAPTER 7

CONCLUSION

In this project, the authenticated user writes or uploads a source code, in any programming language like c, c++ or java. Then the user sends this code to the remote compiler for compilation. If any errors exist in compilation, they are displayed to the user; otherwise the respective executable file is created. A user is allowed to execute a source code only when it has been compiled. On execution, the respective output (errors or results) are displayed to the user.

Behind all this, when the user sends a code for compilation, that code is analysed and the user is prompted for input, if any input statement exists. The assignment statements replace the input statements. And then the code is compiled. In this process certain temporary files are created that are deleted after the source code has been compiled.

Also, the user is allowed to view his own directory structure in the tree form and also he is allowed to create his own directories within. The user is also allowed to run an entire user application provided all the required files have their respective executables.

BIBLIOGRAPHY

- Java 2 Complete Reference
- http://extjs.com/
- http://ww.prototypejs.org/
- http://commons.apache.org/io/
- http://commons.apache.org/fileupload/
- When Runtime.exec() won't – http://www.javaworld.com/javaworld/jw-12-2000/jw-1229-traps.html
- http://www.ibm.com/developerworks/java/
- http://www.ibm.com/developerworks/xml/