

1. Introduction

1.1 About Project

The main aim of this project is we can easily write a java program compile it and debug in on-line. The client machine doesn't have java development kit .The client machine is only connected to the server having java compiler ,so server executes the java code produces the error message to the appropriate client machine.

In this project is also creates a security editor. This editor performs Encryption and decryption of the file. These processes are performed using RSA Algorithms. There is lot of security algorithms, but RSA algorithm is most efficient to encrypt and decrypt the file.

In this project it is used to view all type of java API .It is very useful for writing the java program easily, for example if any error in the format of API there is a possibility to view API thr The main objective of the project is to visualize the things that goes under each and every phase of a compiler. With the help of this product the users can easily understand the compiler. In this project it visualizes the java programs.

In the first phase, it visualizes the tokens that are identified from the uploaded java program done by the user. In the second phase, it generates the parse trees both in top-down and bottom-up view based on the tokens. In the third phase, it shows the semantic view that concentrates on syntax checking and type-casting operation.

In the fourth phase, it generates the intermediate code based on the syntax trees. In the fifth phase, it removes the unnecessary temporary variables from the intermediate code. Finally, the output will be displayed for the given program.

own through this module.

2. Project Analysis

2.1 Purpose of the Project:

The main purpose of this project is to help the users to know the things that goes under each and every phase of a compiler. To make them understand about complete process of compilation, to provide them about knowledge of programming languages implementation and their dependencies with system architecture.

The phases of a compiler are described as follows:

1.Lexical Analysis

This is alternatively known as scanning or tokenization. Lexical Analysis is roughly the equivalent of splitting ordinary text written in a natural language (e.g. English) into a sequence of words and punctuation symbols.

The purpose of lexical analysis is to convert a source program expressed as characters (arranged on lines) into a sequence of valid tokens. Note that this is not the same as a valid sequence of tokens.

Token:

1. In computing, a token is a categorized block of text, usually consisting of indivisible characters known as lexemes.
2. A lexical analyzer initially reads in lexemes and categorizes them according to function, giving the3. A token can look like anything: English, gibberish symbols, anything; it just needs to be a useful part of the structured text.
4. Tokens are frequently defined by regular expressions, which are understood by a lexical analyser such as lex.
5. The lexical analyser reads in a stream of lexemes and categorises them into tokens. This is called "tokenizing." If the lexer finds an invalid token, it will report an error.

6. Following tokenizing is parsing. From there, the interpreted data may be loaded into data structures, for general use, interpretation, or compiling.

Consider the following table:

Lexeme	Token
Sum	IDENT
=	ASSIGN_OP
3	NUMBER
+	ADD_OP
2	NUMBER
;	SEMICOLON

Consider a text describing a calculation:

"46 - Number of (cows);"

The lexemes here might be: "46", "-", "number of", "(", "cows", ")", and ";".

The lexical analyser will denote lexemes 4 and 6 as 'number'

and - as character, and 'number of ' as a separate token. Even the lexeme ';' in some languages (such as C) has some special meaning.

7. The whitespace lexemes are sometimes ignored later by the syntax analyser. A token doesn't need to be valid, in order to be recognized as a token. "Cows" may be nonsense to the language, "number of" may be nonsense. But they are tokens none the less, in this example.

2.Syntax Analysis:

This is alternatively known as parsing. It is roughly the equivalent of checking that some ordinary text written in a natural language (e.g. English) is grammatically correct (without worrying about meaning).

The purpose of syntax analysis or parsing is to check that we have a valid sequence of tokens. Note that this sequence need not be meaningful; as far as syntax goes, a phrase such as "true + 3" is valid but it doesn't make any sense in most programming languages.

Recursive Descent Parsing

A *recursive descent parser* is a top-down parser built from a set of mutually-recursive procedures (or a non-recursive equivalent) where each such procedure usually implements one of the production rules of the grammar. Thus the structure of the resulting program closely mirrors that of the grammar it recognizes.

A predictive parser is a recursive descent parser with no backup. Predictive parsing is possible only for the class of LL(k) grammars, which are the context-free grammars for which there exists some positive integer k that allows a recursive descent parser to decide which production to use by examining only the next k tokens of input. (The LL(k) grammars therefore exclude all ambiguous grammars, as well as all grammars that contain left recursion. Any context-free grammar can be transformed into an equivalent grammar that has no left recursion, but removal of left recursion does not always yield an LL(k) grammar.) A predictive parser runs in linear time.

Recursive descent with backup is a technique that determines which production to use by trying each production in turn. Recursive descent with backup is not limited to LL(k) grammars, but is not guaranteed to terminate unless the grammar is LL(k). Even when they terminate, parsers that use recursive descent with backup may require exponential time.

Although predictive parsers are widely used, programmers often prefer to create LR or LALR parsers via parser generators without transforming the grammar into LL(k) form.

Some authors define the recursive descent parsers as the predictive parsers. Other authors use the term more broadly, to include backed-up recursive descent.[citation needed]

Operator Precedence Parsing

Operator precedence parsing is used in shift-reduce parsing. *operator grammar* No production has two nonterminal symbols in sequence on the right hand side. An operator grammar can be parsed using shift-reduce parsing and precedence relations between terminal symbols to find handles. This strategy is known as operator precedence.

Using the generated Token manager

In order to use the generated token manager, an instance of it has to be created. When doing so, the constructor expects a Reader as the source of the input data Once created, the token manager object can be used to get tokens via the

Token *ParserName*TokenManager.getNextToken() throws ParseError;

Each Token object as returned by the getNextToken() method. Such a Token object provides a field *kind* which identifies the token (*ParserNameConstants.java* defines the corresponding constants). It also contains the field *image*, which just holds the matched input data as a String.

3.Semantic Analysis:

This is roughly the equivalent of checking that some ordinary text written in a natural language (e.g. English) actually means something (whether or not that is what it was intended to mean).

The purpose of semantic analysis is to check that we have a meaningful sequence of tokens. Note that a sequence can be meaningful without being correct; in most programming languages, the phrase "x + 1" would be considered to be a meaningful arithmetic expression. However, if the programmer really meant to write "x - 1", then it is not correct.

4.Code Optimization:

Intermediate Representation

The form of the internal representation among different compilers varies widely. If the back end is called as a subroutine by the front end then the intermediate representation is likely to be some form of annotated parse tree, possibly with supplementary tables. If the back end operates as a separate program then the intermediate representation is likely to be some low-level pseudo assembly language or some register transfer language (it could be just numbers, but debugging is easier if it is human-readable).

Optimization

Optimization within a compiler is concerned with improving in some way the generated object code while ensuring the result is identical. Technically, a better name for this chapter might be "Improvement", since compilers only attempt to improve the operations the programmer has requested.

Optimizations fall into three categories:

- Speed; improving the runtime performance of the generated object code. This is the most common optimization
- Space; reducing the size of the generated object code
- Safety; reducing the possibility of data structures becoming corrupted (for example, ensuring that an illegal array element is not written to)

Unfortunately, many "speed" optimizations make the code larger, and many "space" optimizations make the code slower -- this is known as the space-time tradeoff.

Common Optimization Algorithms

Common optimization algorithms deal with specific cases:

1. Dead Code Elimination

2. Common Sub-expression Elimination
3. Copy Propagation
4. Code Motion
5. Induction Variable Elimination
6. Reduction In Strength
7. Function Chunking

Dead Code Elimination

Dead code elimination is a size optimization (although it also produces some speed improvement) that aims to remove logically impossible statements from the generated object code. Dead code is code which will never execute, regardless of input

Consider the following program:

```
a = 5
if (a != 5) {
// Some complicated calculation } ...
```

It is obvious that the complicated calculation will never be performed; since the last value assigned to `a` before the `if` statement is a constant, we can calculate the result at compile-time. simple substitution of arguments produces `if (5 != 5)`, which is false. Since the body of an `if(false)` statement will never execute - it is *dead code* we can rewrite the code:

```
a = 5
// Some statements
```

The algorithm was used to identify and remove sections of dead code

Common Sub-expression Elimination

Common sub-expression elimination is a speed optimization that aims to reduce unnecessary recalculation by identifying, through code-flow, expressions (or parts of expressions) which will evaluate to the same value: *the recomputation of an expression can be avoided if the expression*

has previously been computed and the values of the operands have not changed since the previous computation.

Consider the following program:

```
a = b + c
d = e + f
g = b + c
```

In the above example, the first and last statement's right hand side are identical and the value of the operands do not change between the two statements; thus this expression can be considered as having a *common sub-expression*.

The common sub-expression can be avoided by storing its value in a temporary variable which can cache its result.

After applying this Common Sub-expression Elimination technique the program becomes:

```
t0 = b + c
a = t0
d = e + f
g = t0
```

Thus in the last statement the re-computation of the expression $b + c$ is avoided.

Code Motion

This optimization technique mainly deals to reduce¹ the number of source code lines in the program. For example, consider the code below:

```
for (i = 0; i < n; ++i) {
    x = y + z;
    a[i] = 6 * i + x * x;
}
```


The calculations $x = y + z$ and $x * x$ can be moved outside the loop since within they are loop invariant - they do not change over the iterations of the loop - so our optimized code will be something like this:

```
x = y + z;
t1 = x * x;
for (i = 0; i < n; ++i) {
a[i] = 6 * i + t1;
}
```

This code can be optimized further. For example, strength reduction could remove the two multiplications inside the loop ($6*i$ and $a[i]$).

Induction Variable Elimination

Function Chunking

Function chunking is a compiler optimization for improving code locality. Profiling information is used to move rarely executed code outside of the main function body. This allows for memory pages with rarely executed code to be swapped out.

5.Code Generation

1. Actually, A compiler usually is designed to output an executable program that will allow the user to run your program, and to be directly run by the processor, without having an intermediary interpreter such as in the interpretation process. For your program to be run by the processor however, you will need to transform the instructions in your specific programming language into assembler code, which is then sent to an assembler tool to create object code, which is then linked together with specific libraries to create your executable code.

2. For now, we only really need to worry about transforming the instructions into assembler code. If you intend your programs to run on the x86 architecture, you need to be familiar with x86 assembler code, and so on.

3. Code generation occurs after semantic analysis is done, which gives us enough information to generate more primitive concrete code. The general idea behind code generation is decompose the tree structure of the syntax tree into a sequence of instructions, whatever an instruction set is.

4. In this stage, since we are done with the semantic program, we are not interested in the syntactic and semantic structure of programs but in the order of executions of instructions.

5. Sometimes it may be beneficial to output some sort of intermediate code is often produced before generating actual machine code.

The benefits of this are:-

1. It is easier to generate more abstract code, not bothering too much about things like register allocations,
2. Optimization independent to machine architecture can be done and
3. Compiler bugs can be spotted more easily.

However, it may be simpler for the program to output assembler code directly, but lose the above advantages.

The three address format is used to represent the intermediate code. The format is useful because it is analogous to actual machine instructions in some architectures and, more importantly, allows us to easily change the execution order of instructions, which is an huge advantage over stack-based intermediate code like the byte code of Java.

Although is not a complex problem to reuse names after they have already been used, it is actually beneficial to allocate a new name every time one is needed because it allows us to form a call graph and optimize easily .

The three address code, as the name suggests, consist of three address and opcode, which tells what kind of operation is meant to be done.

For example, an expression $(a + b) * 3$ can be transformed into:

temp1 := a + b; temp2 := temp1 * 3

In the first line, temp1, a and b are addresses and + is an opcode, and the second line is similar to the first one. Unlike load-store machines, it is unnecessary to load variables to registers and store them back. You see why the three address code is easy to handle.

Choosing portable, flexible and expressive instructions is critical; Not having enough instructions can complicate generated code with the combination of several instructions to achieve one operation and having too much may obviously make maintenance more daunting task. Probably the best way to do this is to examine existing machine code. It is more straightforward to transform code close to underlying machine code than abstract one.

Expression

Algebraic expressions can be translated into the three address code in a very straightforward manner. This can be done rather recursively as follows: Assume two expressions left and right with an operation op-code, then the results should be:

code for left

code for right

temp = place for left + place for right

2.2 Existing System:

At present , Students prefer text books to gain knowledge about the Compiler but they don't extract the exact things from the text book as the human nature is that if we see the actual process with our eyes we will not forget it in our life time.

2.2.1 Problems in Existing System:

- i. Takes time to understand the compiler process.
- ii. User cannot understand completely about the compiler process.

2.2.2 Proposed system:

By using this system there are following advantages:

1. By using this system user can easily understand the compiler.
2. User can save his/her time by using this system.

3. Requirement Analysis

3.1 Purpose and Scope:

The main purpose of this project is to help the users to know the things that goes under each and every phase of a compiler. To make them understand about complete process of compilation, to provide them about knowledge of programming languages implementation and their dependencies with system architecture.

1. Any compiler has some essential requirements, which are perhaps more stringent than for most programs:

a)Any valid program must be translated correctly, i.e. no incorrect translation is allowed.

b)Any invalid program must be rejected and not translated.

2. There will inevitably be some valid programs which can't be translated due to their size or complexity in relation to the hardware available.

for example, problems due to memory size.

3. The compiler may also have some fixed-size tables which place limits on what can be compiled (some language definitions place explicit lower bounds on the sizes of certain tables, to ensure that programs of reasonable size/complexity can be compiled).

4. There are also some desirable requirements, some of which may be mutually exclusive:

a)Errors should be reported in terms of the source language or program.

b)The position at which an error was detected should be indicated; if the actual error probably occurred somewhat earlier then some indication of possible cause(s) should also be provided.

c)Compilation should be fast.

d)The translated program should be fast

5. If the source language has some national or international standard:

a) Ideally the entire standard should be implemented.

b) Any restrictions or limits should be well and clearly documented.

6. If extensions to the standard have been implemented:

a) These extensions should be documented as such.

b) There should be some way of turning these extensions off.

7. There are also some possibly controversial requirements to consider:

a) Errors detected when the translated program is running should still be reported in relation to the original source program e.g. line number.

b) Errors detected when the translated program is running should include division by 0, running out of memory, use of an array subscript/index which is too big or too small, attempted use of an undefined variable, incorrect use of pointers, etc.

3.2 Users of the System:

a. Student

b. Faculty

c. Programmer

d. Software Engineer

4. Specific Requirements

4.1 Functional and Non- Functional Requirements :

Functional Requirements :

1. Accept the source code as an input.
2. Translate the input file into lexemes.
3. Provides Visualization for the translated lexemes.
4. Construct Parse trees for the lexemes.
5. Provide Visualization for the Parse trees.
6. Create a stream of simple instructions from the parse trees.
7. Provides Visualization for the instructions formed.
8. Optimal code generation from the intermediate code generated.
9. Provide Visualization for the optimal code.
10. Object code generation for the Optimized code.
11. Provide Visualization for the object code.
12. Keeps track of names used by the program in symbol table.
13. Error flaw detection is done in the error-handler.

Non- Functional Requirements:

- 1 .Secure access of confidential data(user's details).SSL can be used.
2. 24x7 availability.
3. Better component design to get better performance at peak time.
4. Flexible service based architecture will be highly desirable for future for extension.

4.2 User Interface Requirements:

- A. Professional look and feel
- B. Use of AJAX atleast with all registration forms
- C. Use of Graphical tool to show strategic data to admin
- D. Reports exportable in .XLS, .PDF or any other desirable format.

4.3 Proposed System Architecture :

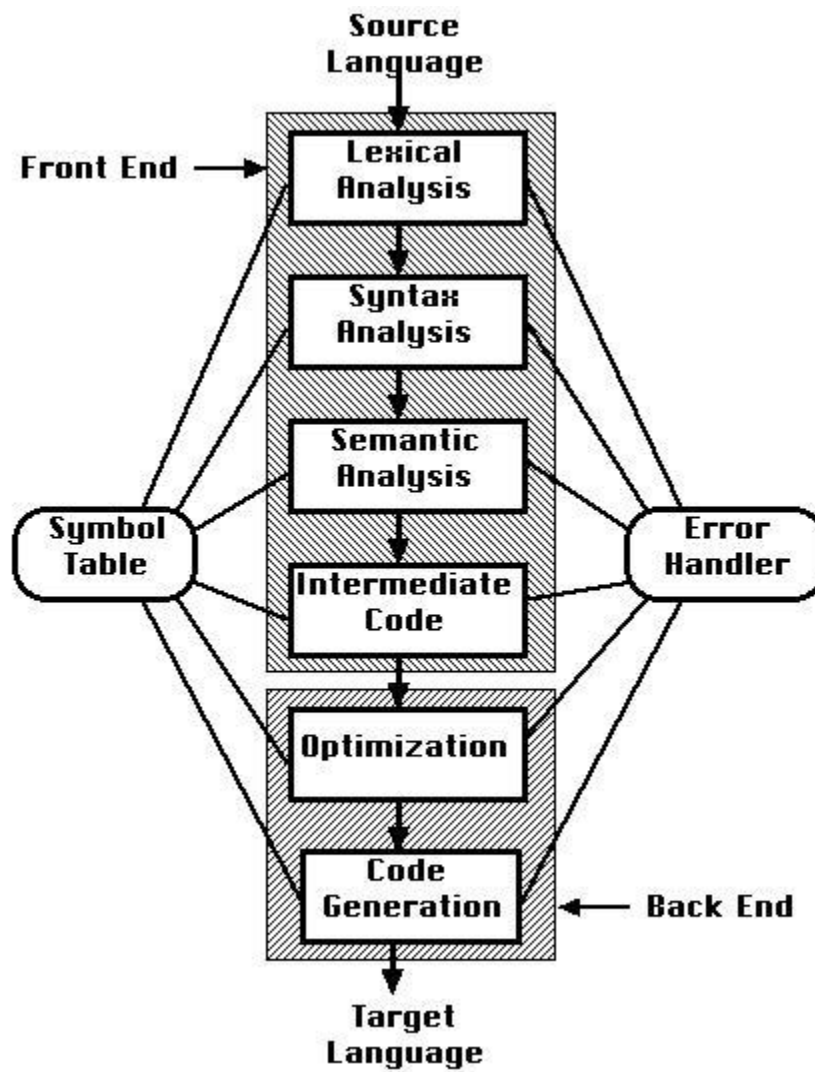


Figure1: Architecture Diagram

5. System Requirements

5.1 Technologies Used:

- 1. Core Java**
- 2. Java Database Connectivity (JDBC)**
- 3. Unified Modeling Language (UML)**

5.2 Tools Used

- 1. Netbeans**
- 2. Oracle 10G Database**

5.3 Software Overview

5.3.1 Netbeans

NetBeans Platform Features

Why would you use the NetBeans Platform? What does the NetBeans Platform give you? Many out-of-the-box components and much else besides.

The main reusable features and components comprising the NetBeans Platform are outlined below.

Module System

The modular nature of a NetBeans Platform application gives you the power to meet complex requirements by combining several small, simple, and easily tested modules encapsulating coarsely-grained application features.

Powerful versioning support helps give you confidence that your modules will work together, while strict control over the public APIs your modules expose will help you create a more flexible application that's easier to maintain.

Since your application can use either standard NetBeans Platform modules or OSGi bundles, you'll be able to integrate third-party modules or develop your own.

Lifecycle Management

Just as application servers, such as GlassFish or WebLogic, provide lifecycle services to web applications, the NetBeans runtime container provide lifecycle services to Java desktop applications.

Application servers understand how to compose web modules, EJB modules, and related artifacts, into a single web application. In a comparable manner, the NetBeans runtime container understands how to compose NetBeans modules into a single Java desktop application.

There is no need to write a main method for your application because the NetBeans Platform already

contains one. Also, support is provided for persisting user settings across restart of the application, such as, by default, the size and positions of the windows in the application.

Pluggability, Service Infrastructure, and File System

End users of the application benefit from pluggable applications because these enable them to install modules into their running applications.

NetBeans modules can be installed, uninstalled, activated, and deactivated at runtime, thanks to the runtime container.

The NetBeans Platform provides an infrastructure for registering and retrieving service implementations, enabling you to minimize direct dependencies between individual modules and enabling a loosely coupled architecture (high cohesion and low coupling).

The NetBeans Platform provides a virtual file system, which is a hierarchical registry for storing user settings, comparable to the Windows Registry on Microsoft Windows systems. It also includes a unified API providing stream-oriented access to flat and hierarchical structures, such as disk-based files on local or remote servers, memory-based files, and even XML documents.

Window System, Standardized UI Toolkit, and Advanced Data-Oriented Components

Most serious applications need more than one window. Coding good interaction between multiple windows is not a trivial task. The NetBeans window system lets you maximize/minimize, dock/undock, and drag-and-drop windows, without you providing any code at all.

Swing and JavaFX are the standard UI toolkits on the Java desktop and can be used throughout the NetBeans Platform. Related benefits include the ability to change the look and feel easily via "Look and Feel" support in Swing and CSS integration in JavaFX, as well as the portability of GUI components across all operating systems and the easy incorporation of many free and commercial third-party Swing and JavaFX components.

With the NetBeans Platform you're not constrained by one of the typical pain points in Swing: the JTree model is completely different to the JList model, even though they present the same data. Switching between them means rewriting the model. The NetBeans Nodes API provides a generic model for

presenting your data. The NetBeans Explorer & Property Sheet API provides several advanced Swing components for displaying nodes.

In addition to a window system, the NetBeans Platform provides many other UI-related components, such as a property sheet, a palette, complex Swing components for presenting data, a Plugin Manager, and an Output window.

Miscellaneous Features, Documentation, and Tooling Support

The NetBeans IDE, which is the software development kit (SDK) of the NetBeans Platform, provides many templates and tools, such as the award winning Matisse GUI Builder that enables you to very easily design your application's layout.

The NetBeans Platform exposes a rich set of APIs, which are tried, tested, and continually being improved.

The community is helpful and diverse, while a vast library of blogs, books, tutorials, and training materials are continually being developed and updated in multiple languages by many different people around the world.

5.3.2 Oracle Database 10g

What's New in PL/SQL in Oracle Database 10g?

As is the case with every new release, Oracle Database 10g introduces some new PL/SQL language features and some new supplied PL/SQL packages. These are listed briefly at the end of this page.

However, the big news for PL/SQL in this release is the dramatic increase in runtime performance from transparent changes. Oracle Database 10g brings a new PL/SQL compiler and a newly tuned PL/SQL execution environment. Additionally, the system for the native compilation of PL/SQL has been substantially improved. As a result, users can expect that

Computationally intensive PL/SQL programs compiled under Oracle Database 10g will run, on average, twice as fast as they did under Oracle9i Database Release 2.

They will run three times as fast as they did under Oracle8 Database.

These factors are so large that they might seem — literally — incredible. Lest you doubt them, we provide extensive corroborative collateral:

PL/SQL Just Got Faster explains the workings of the PL/SQL compiler and runtime system and shows how major improvements on this scale are indeed possible.

PL/SQL Performance Measurement Harness describes a performance experiment whose conclusion is the large factors quoted above. We've provided a downloadable kit to enable you to repeat the experiment yourself.

Freedom, Order, and PL/SQL Optimization , intended for professional PL/SQL programmers, explores the use and behavior of the new compiler.

PL/SQL Performance — Debunking the Myths , again intended for readers who work at the PL/SQL codeface, re-examines some old notions about PL/SQL performance.

PL/SQL Just Got Faster

This whitepaper is derived from the OracleWorld 2003 presentation with the same title by Oracle Corporation's PL/SQL Team. It describes the radical changes that were made to the PL/SQL compilation and runtime systems moving from Oracle9i Database Release 2 to Oracle Database 10g. It also explains how some of the improvements to the runtime system from this long term project have been "leaked" into earlier Oracle versions starting with Oracle8i Database.

Download the whitepaper

PL/SQL Performance Measurement Harness

This paper describes an experiment, using a large set of representative test programs, which measures the effect of the changes described in the PL/SQL Just Got Faster companion paper and presents the experimental results and their analysis.

Everything required to enable the reader to repeat the experiment and to verify the conclusions has been packaged for download from this site. The paper serves also as the instruction manual for the downloaded kit. It shows how you can easily include your own test programs in the experiment.

[Download the whitepaper](#)

[Download the complete software kit \(6040 Kb\)](#)

Freedom, Order, and PL/SQL Optimization

To achieve its speedups, the Oracle Database 10g PL/SQL compiler reorganizes the original source code somewhat (as all optimizing compilers do). The optimizations apply both to interpreted and to natively compiled PL/SQL programs. These rearrangements may cause “inessential” changes in the detailed behavior of a PL/SQL program.

This paper describes exactly what changes may be made and why; it is a careful explanation of parts of the PL/SQL definition which have been only lightly described previously. Those who design or program PL/SQL applications and who generally are familiar with the PL/SQL reference materials will benefit from reading this paper as will those who are responsible for porting existing applications from earlier Oracle Database releases. The material is technical, but every effort is made to keep the presentation clear enough so that all those interested in PL/SQL may learn from it.

[Download the whitepaper](#)

PL/SQL Performance — Debunking the Myths

The PL/SQL improvements in Oracle Database 10g change the way programmers should think about traditional hand optimizations. The paper examines some old notions about PL/SQL performance and shows that some no longer hold while others do continue to be sound. Further notions come under scrutiny which have never been sound but which mysteriously have been popularly held.

The paper describes an experimental investigation whose results speak for themselves. However, the interpretation does take advantage of the special understanding of the internals of PL/SQL brought by the team at Oracle Headquarters that implements the language. The results and their expert interpretation show that, in Oracle Database 10g, the PL/SQL programmer can now concentrate on clarity and correctness and — more than ever before — delegate the job of generating efficient runtime code to the compiler and the execution environment.

[Download the whitepaper](#)

[Download the presentation slides](#)

Summary of New PL/SQL Language Features

Oracle Database 10g introduces support for these new language features:

the `binary_float` and `binary_double` datatypes (the IEEE datatypes).

the `regexp_like`, `regexp_instr`, `regexp_substr` and `regexp_replace` builtins to support regular expression manipulation with standard POSIX syntax.

multiset operations on nested table instances supporting operations like equals, union, intersect, except, member, and so on.

the user-defined quote character.

indices of and values of syntax for `forall`.

the distinction between `binary_integer` and `pls_integer` vanishes.

All these features except the user-defined quote character — a convenience feature for the programmer — are performance features and, of course, they all have efficient implementations. For example, the IEEE datatypes enjoy the benefit of machine arithmetic for mathematical real numbers. An appropriate algorithmic task that is expressed in PL/SQL using any of these new features will run very much faster than one that avoided them. The PL/SQL Just Got Faster whitepaper reports improvement factors ranging between four and thirteen for test programs which depend heavily on number arithmetic and which were reimplemented using the new `binary_double` datatype.

In the list of new language features, all but the `forall` enhancement and the change concerning PL/SQL integers are, formally speaking, new features in SQL. PL/SQL has an obligation to support all such SQL features in a PL/SQL context (for example, in PL/SQL assignment statements). This is a unique strength of PL/SQL and is closely related to the fact that it shares the same datatype system as SQL. The PL/SQL Development Team works to enable users to be able to use SQL and PL/SQL seamlessly together.

The new PL/SQL compiler also introduces support for compiler warnings. There are various categories of warning (severe, performance, and informational). Each category and each individual warning can be independently enabled, disabled, or treated as a compilation error.

Summary of New PL/SQL Supplied Packages

PL/SQL packages are used to extend the functionality of the Oracle Database when SQL cannot be extended for the purpose. As such, they implement a vast range of functionality. As is normal when discussing new PL/SQL features, we restrict ourselves in this section to just those packages which can be considered to augment the PL/SQL language itself.

`Utl_Mail`. This new package makes it possible for a PL/SQL programmer to send programmatically composed emails. It requires only the normal mental model of a user of a GUI email client rather than an understanding of the underlying protocol (SMTP) features. This distinguishes it from `Utl_Smtp` which was introduced in Oracle8i Database. `Utl_Smtp` requires that the programmer understands the details of the SMTP protocol. `Utl_Mail` is much simpler to use because it supports just a limited, but very common, subset of the functionality that `Utl_Smtp` provides.

Utl_Compress. This new package delivers the familiar functionality of the zip and unzip utilities in a PL/SQL environment. It lets you compress and uncompress a raw or blob bytestream and guarantees return of original bytestream after a round trip.

Dbms_Warning. This allows the PL/SQL programmer fine grained control over which categories of warning and which individual warnings to disable, to enable, or to treat as errors. Its expected use is at the start and end of installation scripts so that each script may run in its intended regime without affecting the regime of subsequent scripts.

6. System Design

6.1 Behavioral Diagrams :

6.1.1 Use Case Diagrams :

Use Case : User actions

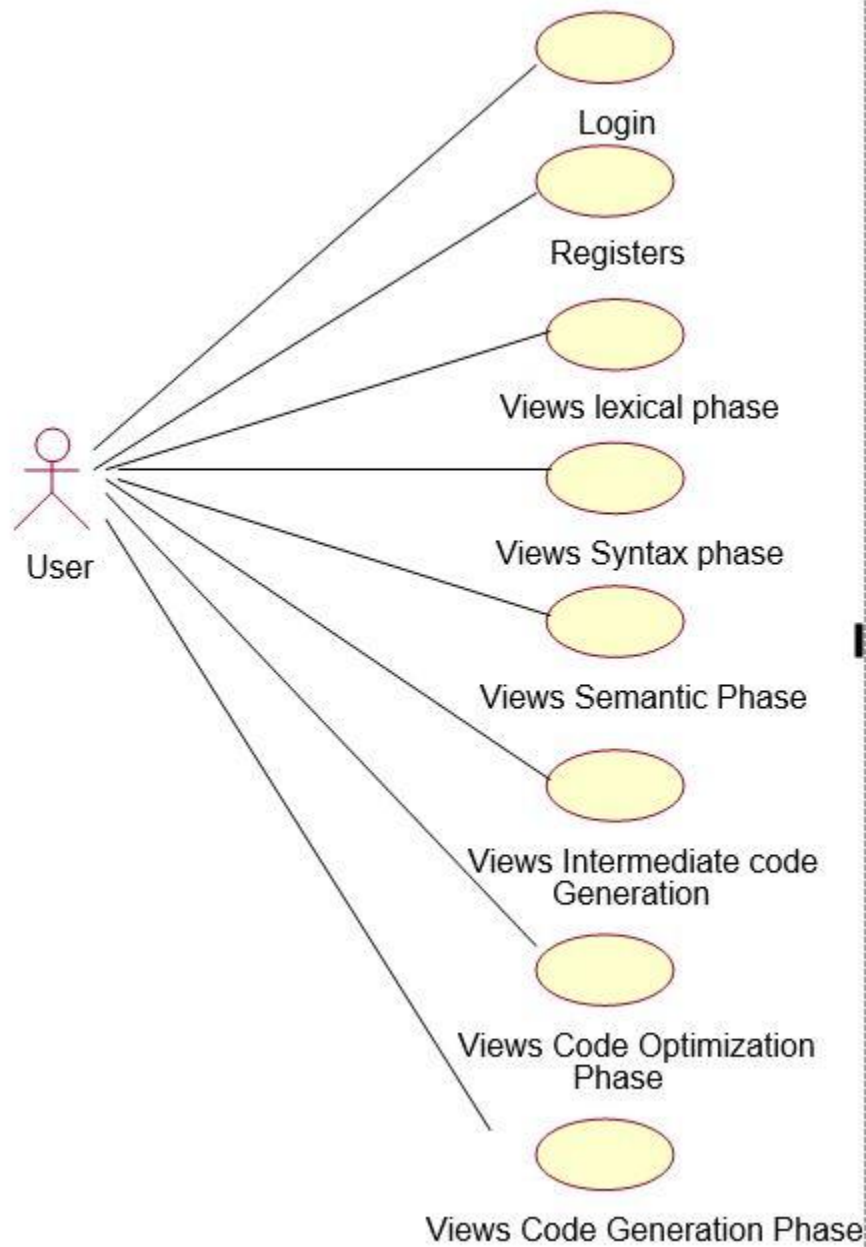


Fig 2 : use case of the system

Description:

Name of the Use Case: Login

Description: Every user of the product must be login to view the compiler process .

Pre-condition: Each user must have a valid user id and password.

Post condition: Uploading file page will be displayed.

Flow of events:

- Invoke the Welcome page of System.
- Enter the valid User ID and Password.
- Click on Sign In button to access Uploading page.

Alternative Flow of Events:

- If the user is new Click on Registration Link.

Name of The Use Case: Uploading a file

Description: The user can upload a java file

Pre-condition: The User must be logged into the system.

Post condition: views all phases.

Flow of events:

- Login to the Home Page.
- Uploads the file.

6.1.2 Class Diagrams:

For User:

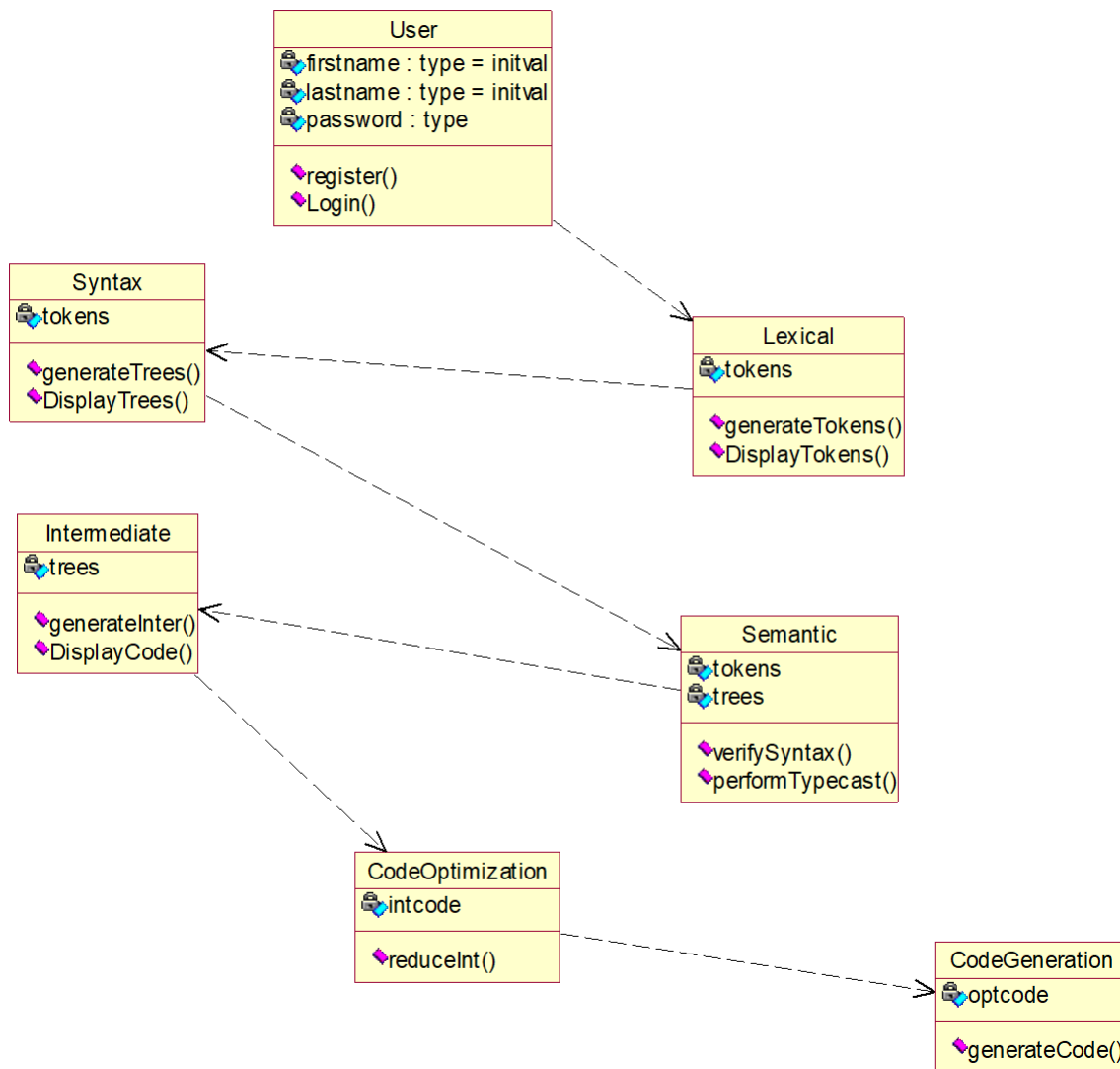


Fig:3 :Class Diagram for Visualization of Compiler Phases

6.1.3 Sequence Diagrams:

A sequence diagram represents the sequence and the interactions of given case or scenario. A sequence diagram shows an interaction arranged in a time sequence.

Login and Registration:

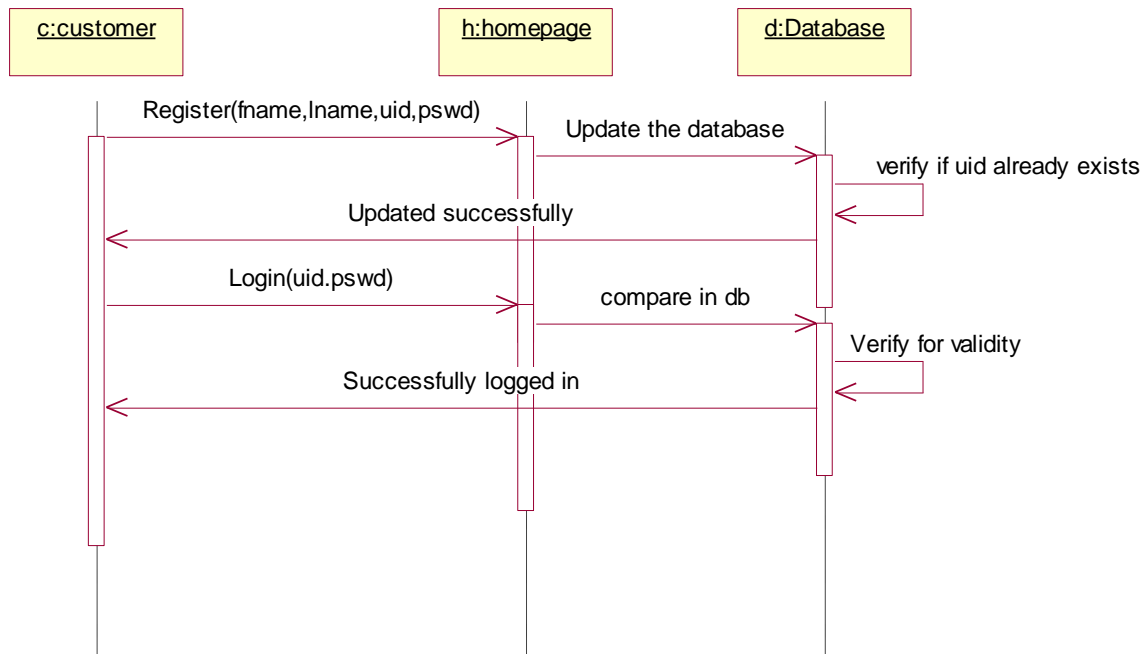


FIGURE 4: Sequence Diagram for Login and Registration

Description:

1. Customer enters userid and password in the Login screen.
2. login screen get details from database and verifies these details in database.
- 3: If user is valid then he can view homepage.
4. If he is not valid, then he has to register.
5. Successful registration will be displayed.

Sequence diagram for Uploading a file:

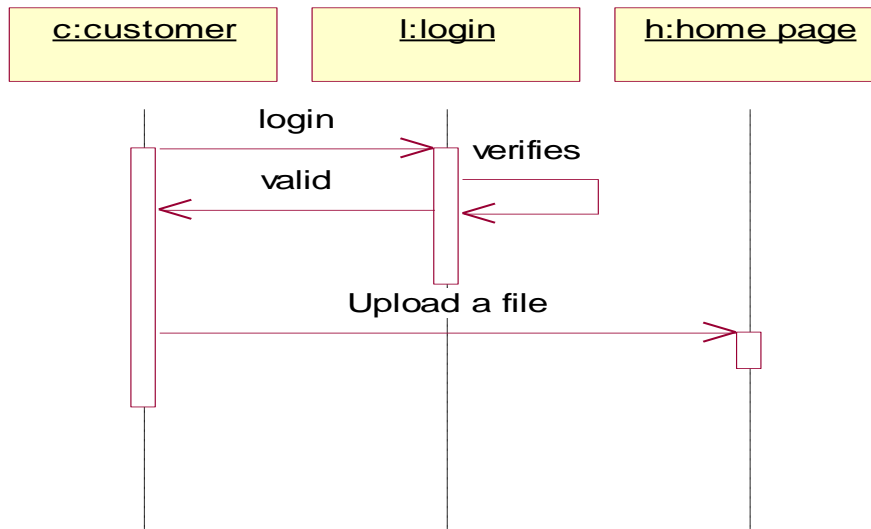


FIGURE 5: Sequence Diagram for Updating a file

Sequence diagram for Lexical phase

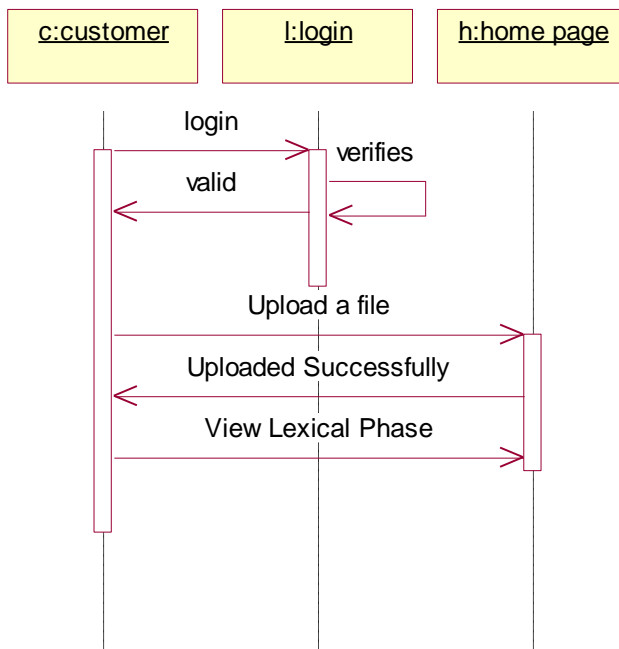


FIGURE 6: Sequence Diagram for Lexical phase

Sequence diagram for Syntax Phase

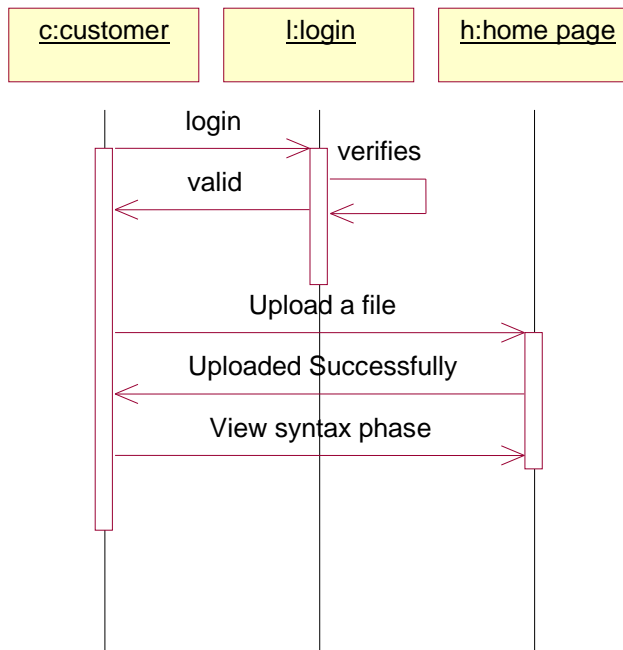


FIGURE 7:Sequence Diagram for Syntax phase

Sequence diagram for Semantic phase

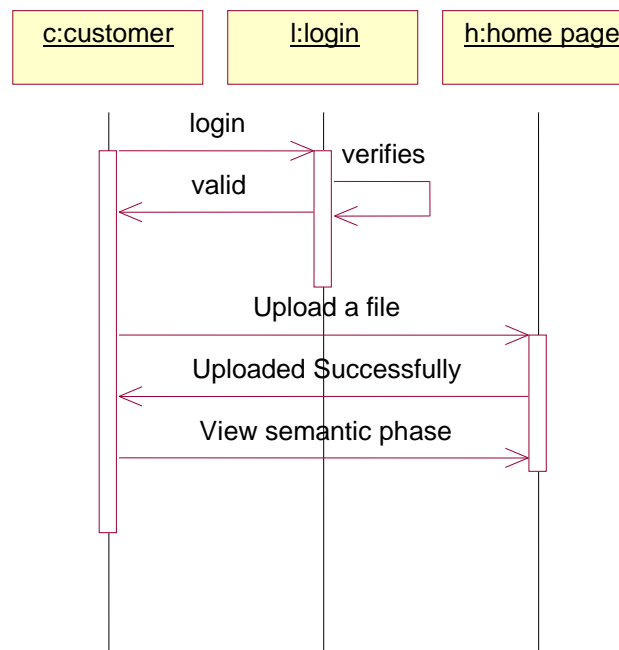


FIGURE8:Sequence diagram for Semantic phase

Sequence diagram for Intermediate code Phase

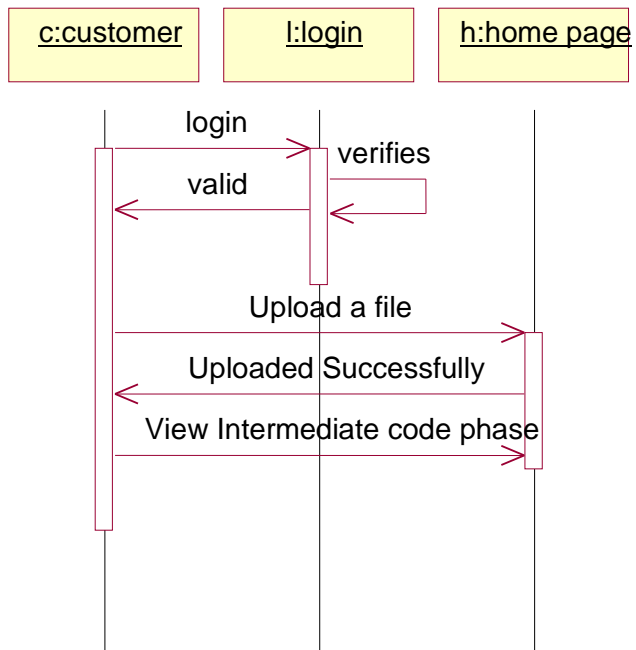


FIGURE 9: Sequence Diagram for Intermediate code

Sequence diagram for Code Optimization Phase

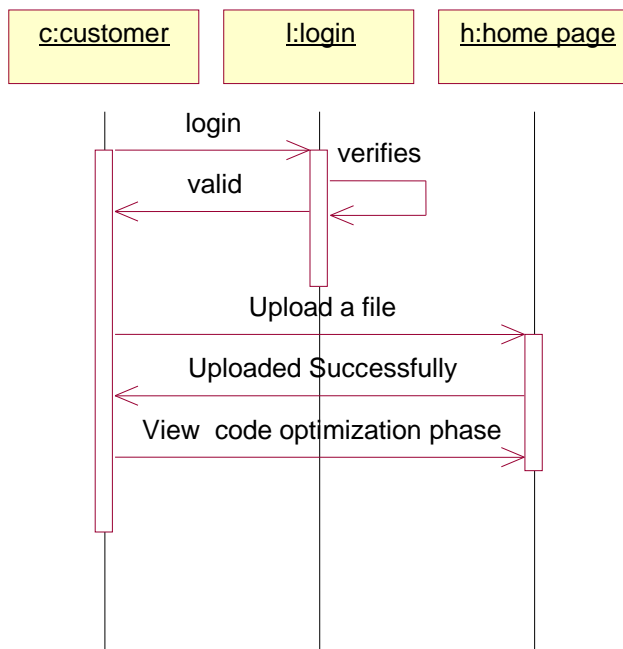


FIGURE 10: Sequence Diagram for Code Optimization

Sequence diagram for Code Generation Phase

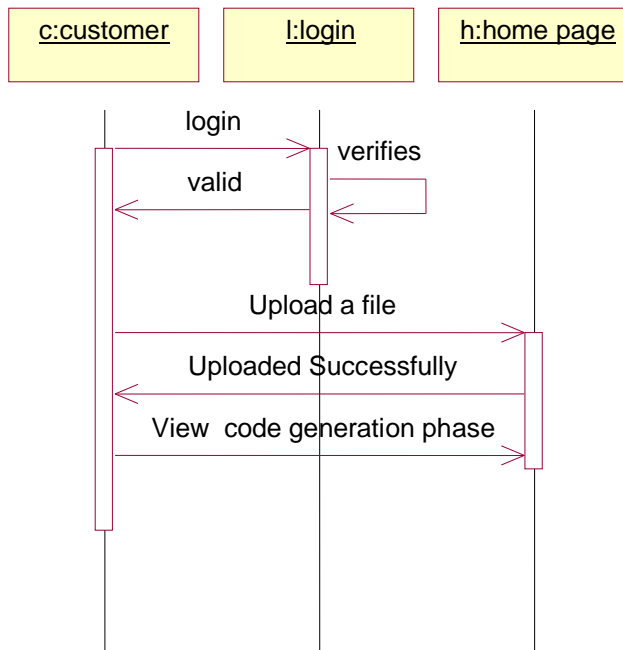


FIGURE 11: Sequence Diagram for Login and Registration

6.1.4 Collaboration Diagrams :

This diagram is an interaction diagram that stresses or emphasizes the structural organization of the objects that send and receive messages. It shows a set of objects, links between objects and messages send and received by those objects. There are used to illustrate the dynamic views of a system.

Collaboration Diagram for Login and Registration:

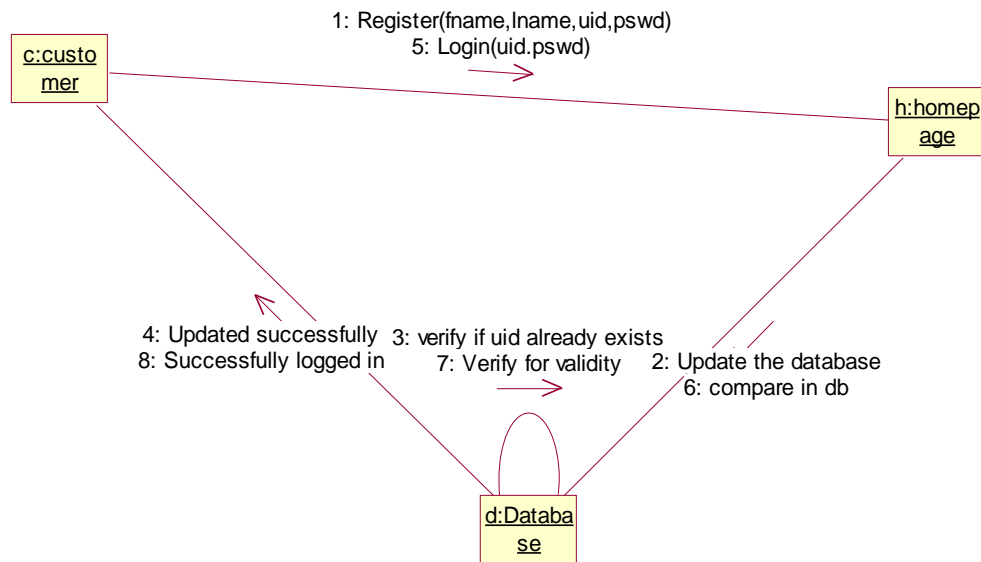


FIGURE 12: Collaboration Diagram for Login and Registration.

Collaboration Diagram for Lexical Phase:

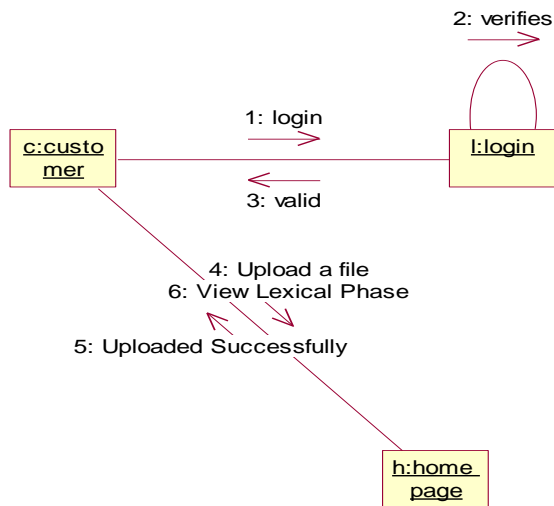


FIGURE 13: Collaboration Diagram for Lexical phase.

Collaboration diagram for Syntax Phase

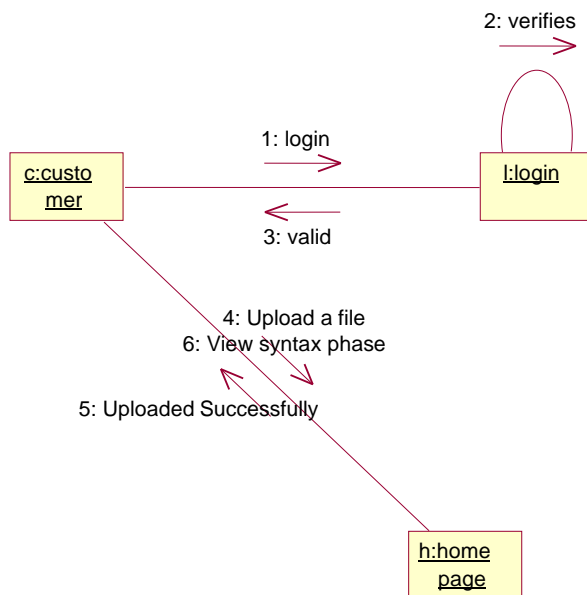


FIGURE 14: Collaboration Diagram for Syntax phase.

Collaboration diagram for Semantic Phase

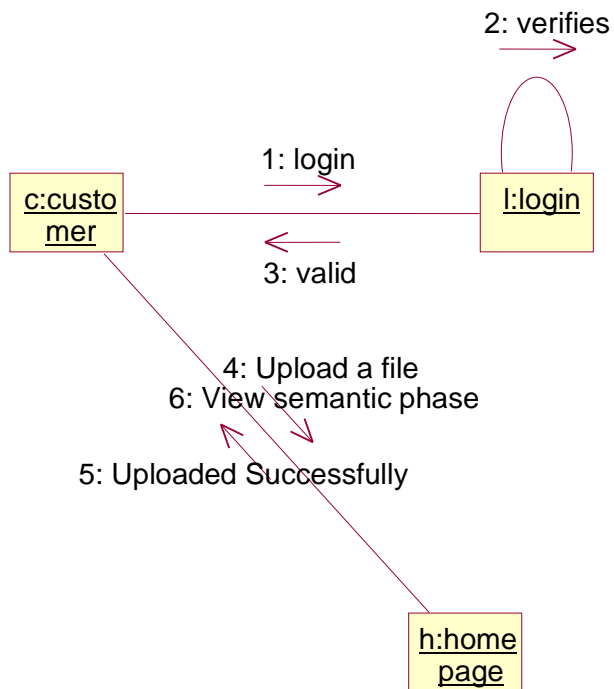
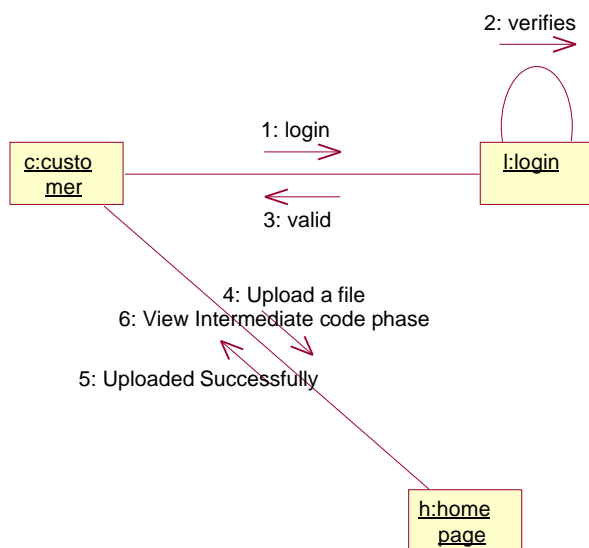


FIGURE 15: Collaboration Diagram for Semantic phase



Collaboration diagram for Code Optimization phase

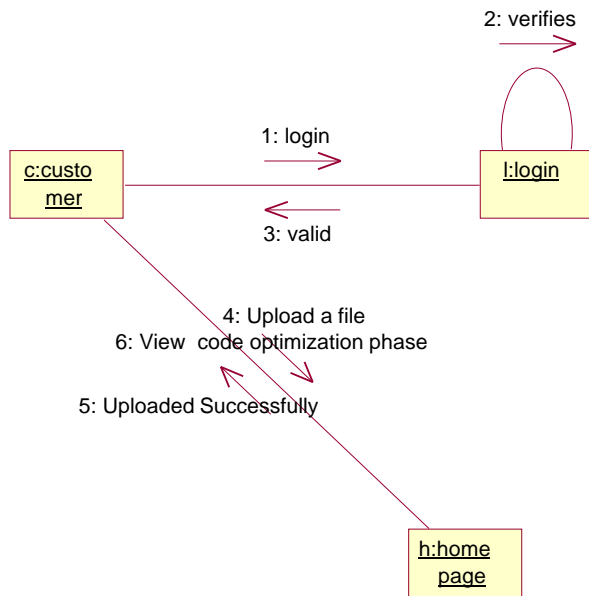
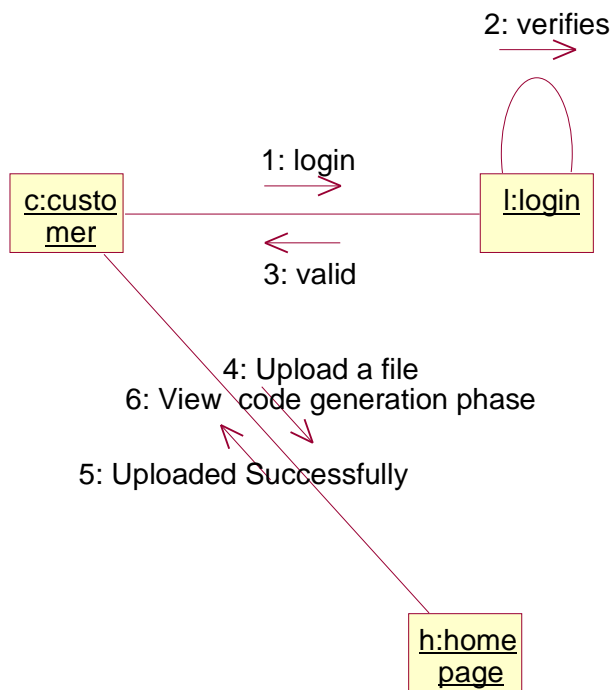


FIGURE 17: Collaboration Diagram for Code optimization
Collaboration diagram for Code Generation Phase



6.1.5 State Chart Diagrams :

State chart diagram shows a state machine consisting of states, transitions and activities these illustrates the dynamic view of a system. They focus on the event ordered.

Login :

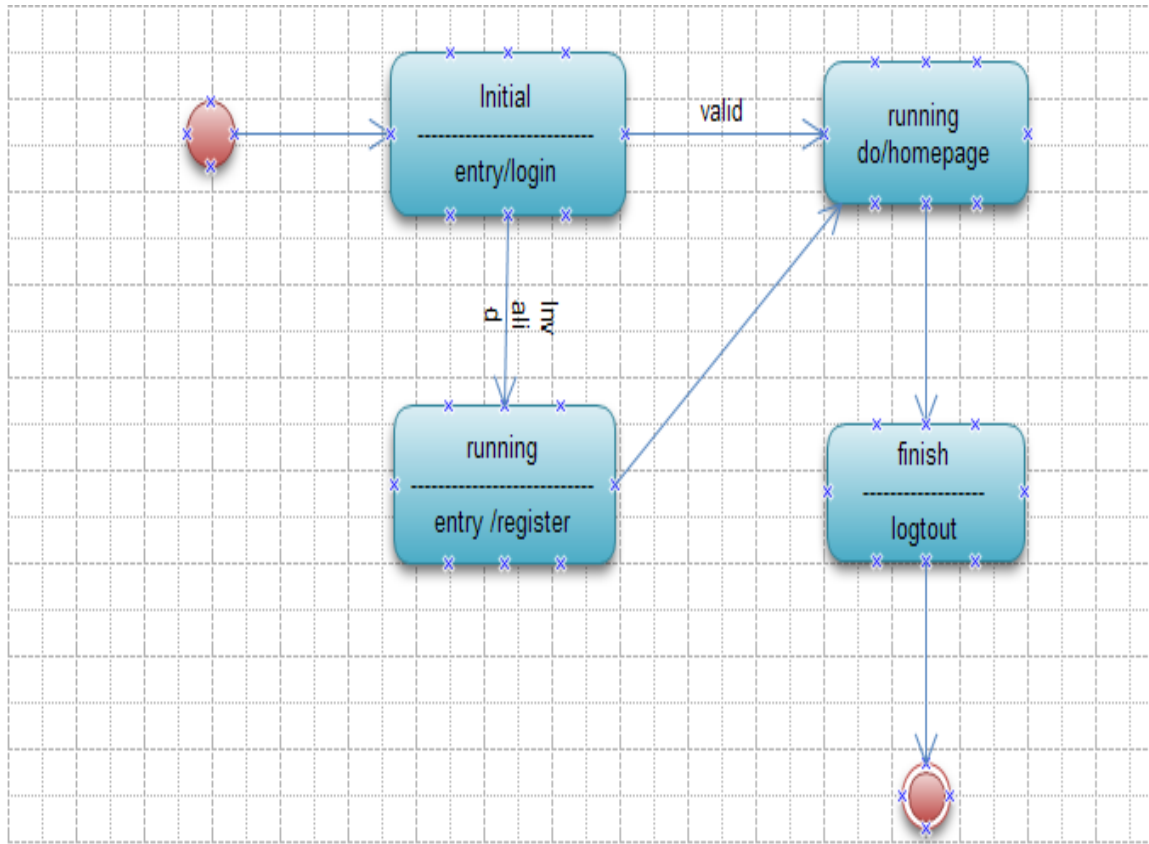


FIGURE 19: State Chart Diagram for Login.

7. System Specific Modules

1. Login and Registration:

In this module a customer first login in to the system if he is already registered then he is authenticated user. Suppose a customer is not authenticated then he is first registered into system.

2. Uploading file:

In this module a customer can upload the file so that he/she can view all the phases of a java compiler.

3. Lexical Phase:

In this module the tokens will be identified from the uploaded java file and are displayed to the user.

4. Syntax Phase:

In this module parse trees were constructed from the identified tokens and are displayed to user in both top-down and bottom-up order.

5. Semantic Phase:

In this module it verifies whether the syntax is right or not and performs type-casting operation..

6. Intermediate Code Generation:

In this module it displays the intermediate code for the uploaded java file.

7. Code Optimization:

In this module it removes the unnecessary temporary variables from the intermediate code and displays it to the user.

8. Code generation:

In this module it displays the actual output of the program.

7.1 System Evolution

System to be changed:

Generally, Students prefer text books to gain knowledge about the Compiler but they don't extract the exact things from the text book as the human nature is that if we see the actual process with our eyes we will not forget it in our life time.

Change Proposals:

The change proposal to the existing system is that exists today, where user can save time and easily understand what the compiler process is by using this system.

Change Analysis:

Here student can upload any java program and view the outputs of each and every phase of a compiler.

System understanding:

Complete understanding of the system that is to be generated i.e. a brief study of the requirements and Designing the system that is to be developed

System Validation:

Validation can be fined in many ways, but a simple definition is that validation succeeds when software functions in a manner that can be reasonably expected by the customer, i.e. the customer expected output i.e. fulfilling all the customer specified requirements.

Modified System:

The modified system from the existing system is the Visualization of compiler phases. This is very effective to students .

8. Testing

8.1 Functional Test cases:

Table 1. Functional Test cases for Visualization of compiler phases.

8.2 Integration Test cases :

TEST CASE ID	DESCRIPTION	TEST STEPS	EXPECTED RESULT	ACTUAL RESULT	REMARKS	DEFECT ID
Upload_2	Verify the outputs of every phase.	Run reg.java and verify userid and password	Upload page of user should be displayed.			
		Click on upload button.	Verify whether the file is uploaded successfully.			
Phases_7		Click on submit button.	Verify whether Lexical phase output is obtained for the given program or not.			
		click on next button.	Verify whether Syntax phase			

			output is obtained or not.			
--	--	--	-------------------------------	--	--	--

	Verify reports and graphs based on items sold.	Click on the next button	Verify whether semantic phase output is obtained or not.			
		Click on next button.	Verify whether Intermediate code is obtained or not.			
		Click on next button.	Verify whether Code Optimizatio n phase is obtained or not.			
		Click on the next button	Verify whether Final output is obtained or not.			

Table .2. Integration Test cases for Visualization of Compiler Phases.

8.3 Database Tables

ORACLE Database Express Edition

User: SYSTEM

Home > Object Browser

Tables

login

LOGIN

Table

Data

Indexes

Model

Constraints

Grants

Statistics

UI Defaults

Triggers

Dependencies

SQL

Add Column

Modify Column

Rename Column

Drop Column

Rename


Copy

Drop

Truncate

Create Lookup Table

Column Name	Data Type	Nullable	Default	Primary Key
EMAIL1	VARCHAR2(40)	No	-	1
NAME1	VARCHAR2(40)	Yes	-	-
PASS1	VARCHAR2(40)	Yes	-	-
MOB1	NUMBER	Yes	-	-
DESC1	VARCHAR2(250)	Yes	-	-
1 - 5				

EDIT	EMAIL1	NAME1	PASS1	MOB1	DESC1
	aamir2786@gmail.com	mohd aamir	9358464830	8307417070	-
row(s) 1 - 1 of 1					

Tables

java

JAVAPROGRAMS1

JAVAPROGRAMS1

Table

Data

Indexes

Model

Constraints

Grants

Statistics

UI Defaults

Triggers

Dependencies

SQL

Add Column

Modify Column

Rename Column

Drop Column

Rename


Copy

Drop

Truncate

Create Lookup Table

Column Name	Data Type	Nullable	Default	Primary Key
EMAIL1	VARCHAR2(30)	Yes	-	-
PROGNAME	VARCHAR2(30)	Yes	-	-
PROG1	CLOB	Yes	-	-
1 - 3				

EDIT	EMAIL1	PROGNAME	PROG1
	aamir2786@gmail.com	AAMIR	class AAMIR { public static void main(String args[]) { int x, y, z; x = 20; y = 20; z = x + y; System.out.println("Sum of x and y = "+z); } }
row(s) 1 - 1 of 1			

9 Output Screens



YOU HAVE TO LOGIN FIRST

Login Here

Enter Emaild Or password for login

New Account Create

Welcome New User ,Please Enter Your Details Here

Login Here

Enter EmailId Or password for login

Signin

New Account Create

Welcome New User ,Please Enter Your Details Here



Login Here

Enter EmailId Or password for login

New Account Create


Welcome New User ,Please Enter Your Details Here

Write Here

Enter Class Name

ShowCode RunCode Download save

```
class AAMIR
{
    public static void main(String args[])
    {
        int x, y, z;
        x = 10;
        y = 20;
        z = x + y;
        System.out.println("Sum of x and y = "+z);
    }
}
```

 Rectangular Snip

Output

Sum of x and y = 30

Contact Info

Lorem ipsum dolor sit amet, consectetur adipiscing elit



Phone

+91-705-524-3882



Address

Latin literature from 45 BC

Send a Message

Lorem ipsum dolor sit amet, consectetur adipiscing elit

10 Conclusion And Features

The project is based on “Text Editor”. This software firm deals in developing software for its clients.

Text Editor: - A text editor is a type of program used for editing plain text files. A plain text file is represented and edited by showing all the characters as they are present in the file. The only characters usable for 'mark-up' are the control characters of the used character set; in practice this is newline, tab and form etc.

As it is a competitive world and very fast world, every thing in the universes is to be internet. In this internet world all the things are on-line. So we created software called “On-line java compiler with security editor”.

Text Book References:

- GRADY BOOCH, IVAR JACOBSON, JAMES RUMBAUGH, Unified Modeling Language , 2nd edition , 2004.
- H.M. DEITEL & P.J . DEITEL , JAVA How to Program ,6th edition , 2005.
- ROGER S. PRESSMAN ,Software Engineering , A Practitioner's Approach , 6th edition , 2005.
- HERBERT SCHILDT , The Complete Reference JAVA , 5th edition, 2005.

Web References:

- www.ScienceDirect.com
- www.w3schools.com
www.compilerworld.com