VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JNANA SANGAMA", BELGAUM – 590014

A Project Report on

"Automatic Detection of Optimal Data Structure in Eclipse"

Submitted in partial fulfillment of the requirements for the award of degree of

Bachelor of Engineering
in
Information Science & Engineering

Submitted by:

SANDEEP K V	**1PI11IS091**
SHAILJA AGARWALA	**1PI11IS097**
SHARATH R	**1PI11IS099**

Under the guidance of

Internal Guide

Dr. Viraj Kumar
Professor,
Department of CSE,
PESIT

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

PES INSTITUTE OF TECHNOLOGY

100 Feet Ring Road, BSK 3rd Stage, Bengaluru – 560085

January 2015 – May 2015

PES INSTITUTE OF TECHNOLOGY

100 Feet Ring Road, B S K 3rd Stage,
Bengaluru-560085

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project work entitled **"Automatic Detection of Optimal Data Structure in Eclipse"** carried out by **Sandeep K V**, bearing USN **1PI11IS091**, **Shailja Agarwala**, bearing USN **1PI11IS097**, **Sharath R**,bearing USN **1PI11IS099**, are bonafide students of **PES INSTITUTE OF TECHNOLOGY**, Bangalore, an autonomous institute, under VTU, in partial fulfillment for the award of degree of **BACHELOR OF ENGINEERING IN INFORMATION SCIENCE & ENGINEERING** of **Visvesvaraya Technological University, Belgaum** during the year **2015**. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the above said degree.

Dr. Viraj Kumar Internal Guide Professor Department OF CSE PESIT	**Dr. Shylaja S S** HOD Department OF ISE PESIT	**Dr. K. S. Sridhar** Principal & Director PESIT

External Viva

Name of the Examiners	**Signature with Date**
1.______________________	__________________
2.______________________	__________________

ACKNOWLEDGEMENT

The satisfaction and the euphoria that accompany the successful completion of our task would be incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crowned our efforts with success.

We express our sincere words of gratitude to our **Chairman, Dr. M R Doreswamy**, for his constant and dedicated support to brighten our career.

We shower sincere words of gratitude to our **Director, Dr. D Jawahar**, for his constant and dedicated support to brighten our career.

We express profound gratitude to our honorable **Principal & Director, Dr. K S Sridhar,** who was a great source of encouragement at all times.

We owe our gratitude to **Head of the Department** of Information Science and Engineering, **Dr. Shylaja S S**, for her kind co-operation and encouragement which helped us in completion of this project.

We take this opportunity to thank our guide, **Dr. Viraj Kumar**, **Professor,** Department of Computer Science and Engineering, who has always been a source of inspiration and encouragement throughout the tenure of the project.

Last but not the least, we wish to place on record our gratitude to **parents** and **friends** who have always been a source of constant moral support and appreciation during the work.

Sandeep KV, Shailja Agarwala, Sharath R
(1PI11IS091, 1PI11IS097, 1PI11IS099)

ABSTRACT

"Bad programmers worry about the code. Good programmers worry about data structures and their relationships." - By Linus Torvalds.

In the above quote, Linus Torvalds claims that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships.

It is true that, data structures and algorithms are the fundamentals of programming. In order to become a good developer it is essential to master the basic data structures and algorithms and learn to apply them in the right way.

The key to a solid foundation in data structures and algorithms is not an exhaustive survey of every conceivable data structure and its sub forms, with memorization of each data structure's Big-O value and amortized cost. Instead, it is the knowledge of when and how to use different data structures and algorithms in a particular code.

Hence, our project aims at finding the optimal data structure automatically given any particular scenario or user program. This will help enhance the knowledge of beginners as well as it will make it easier for teachers to give live examples of which data structure to us.

Index

List of Tables and Figures

1. INTRODUCTION

Data structures and algorithms are the fundamentals of programming. In order to become a good developer it is essential to master the basic data structures and algorithms and learn to apply them in the right way.

The key to a solid foundation in data structures and algorithms is not an exhaustive survey of every conceivable data structure and its sub forms, with memorization of each data structures's Big-O value and amortized cost. Instead, we should learn to:

1) Visualize the data structure. Intuitively understand what the data structure looks like, what it feels like to use it, and how it is structured both in the abstract and physically in your computer's memory. This is the single most important thing to do, and it is useful from the simplest queues and stacks up through the most complicated self-balancing tree. In other words, we need to understand the structure intuitively.

2) Learn when and how to use different data structures and their algorithms in our own code. We should realize we won't be able to master data structures until we are working on a real-world problem and discover that a hash is the solution to our performance woes. But we should focus on learning not the minute details but the practicalities: When do you want a hash? When do you want a tree? When is a min-heap the right solution?

1.1 ALGORITHM COMPLEXITY

Algorithm complexity is a measure which evaluates the order of the count of operations, performed by a given or algorithm as a function of the size of the input data. To put this simpler, complexity is a rough approximation of the number of steps necessary to execute an algorithm. When we evaluate complexity we speak of order of operation count, not of their exact count.

Complexity can be constant, logarithmic, linear, n*log (n), quadratic, cubic, exponential, etc. This is respectively the order of constant, logarithmic, linear and so on, number of steps, are executed to solve a given problem. For simplicity, sometime instead of “algorithms complexity” or just “complexity” we use the term “running time”.

The choice of an appropriate data structure is highly dependable on the specific task. Sometimes data structures have to be combined or we have to use several of them simultaneously.

What data structure should be used mostly depends on the operations being performed on it, so it depends on "what operations should the structure perform efficiently". If we are familiar with the operations, we can easily conform which structure does them most efficiently and at the same time is easy and handy.

However it is not an easy task. In order to efficiently choose an appropriate data structure, we should firstly invent the algorithm, which you are going to implement, and then look for an appropriate data structures for it.

In our project, we implement a program which has inbuilt methods of various data structures. Given any user program, it determines the optimal data structure to use.

1.2 MOTIVATION

Data structures are an important part of any programming language. However learning the optimal one, comes with practice and to write efficient and reliable code, the knowledge of data structures is a must.

Hence, our project aims at finding the optimal data structure automatically given any particular scenario or user program. This will help enhance the knowledge of beginners as well as it will make it easier for teachers to give live examples of which data structure to use where.

1.3 IMPORTANCE OF DATA STRUCTURES IN COMPUTER SCIENCE

Data structure is a particular way of storing and organizing information in a computer so that it can be retrieved and used most productively. Different kinds of data structures are meant for different kinds of applications, and some are highly specialized to specific tasks.

Data structures are important for the following reasons:

1. Data structures are used in almost every program or software system.
2. Specific data structures are essential ingredients of many efficient algorithms, and make possible the management of huge amounts of data, such as large integrated collection of databases.
3. Some programming languages emphasize data structures, rather than algorithms, as the key organizing factor in software design.

As computer scientist, our job is to perform operations on data, we basically perform the following three steps :-

1) Take some input
2) Process it
3) Give back the output.

The input can be in any form, for eg while searching for directions on google maps, you give the starting point and the destination as input to google maps, while logging in to facebook, you give your email and password as input and so on. Similarly, in the third step, the computer application gives us output in some form or the other. To make this process efficient, we need to optimize all the three steps. As you can guess, the most we can optimize is the 2nd step, which is where we have Data structures and algorithms.

Data structures refers to the way we organize information on our computer. With a slight thinking , you can guess that the way we organize information can have a lot of impact on the performance. Take for example, a library. Suppose, you want to have a book on Set Theory from a public library, to do that you have to first go to the maths section, then to set theory section. If these books are not organized in this manner and just distributed randomly then it will be really a cumbersome process to find a book on set theory.

This is the way a librarian organizes his books(data) into a particular form (data structure) to efficiently perform a task(find a book on set theory).

In this manner we computer scientist process and look for the best way we can organize the input taken by user, so it can be better processed.

2. Problem Definition

This project aims at finding the optimal data structure automatically given any particular scenario or user program. The knowledge of when and how to use different data structures and algorithms, in any situation, is still very critical in this age of GHz computations, because of the size of the problems that challenge us today.

In order to efficiently choose an appropriate data structure, we should firstly invent the algorithm, which you are going to implement, and then look for an appropriate data structures for it. understanding what the data structure looks like, what it feels like to use it, and how it is structured both in the abstract and physically in your computer's memory, increases the chances of right data structure being used in the appropriate scenario.

This will help enhance the knowledge of beginners, as well as will make it easier for teachers to give live examples of which data structure to use.

Execution times increases the understanding of the value of asymptotic analysis and its limitations. Since asymptotic analysis in itself cannot give complete reasoning as to why the performance of a particular data structure is the way it is. Important aspects like the computer's cache, memory footprint, garbage collection and other variables come into the picture.

Thus, by using running time in our analysis, we incorporate not only asymptotic factors, but also other factors which should be included none the less.

3. LITERATURE SURVEY

In 2012 **B. Boothe** from Computer Science Department, University of Southern Maine, Portland, ME, USA wrote a paper on **Using Real Execution Timings to Enliven a Data Structures Course**

The problem being discussed in this paper is the online ordered collection problem. The online ordered collection problem is to take N items and insert them into the data structure while maintaining the data structure in sorted order at every insert. The online aspect refers to the possibility of using this data structure to perform a search at any point while it is being built. It incorporates both the cost of creating and growing the data structure as well as the cost of searching it.

The question which this paper tries to address is very simple, which is faster w.r.t the above problem, array list or linked list?

By the traditional data structures knowledge, it seems quite straight forward. Linked lists will be either a little faster or a lot faster than array lists. For justifications they cite the high cost of inserting into the array and the cost of repeatedly growing the array to make it larger.

(1) Using LinkedList to build an ordered collection.

Building O(N)

Searching $O(N^2)$

Inserting O(N)

Total $O(N^2)$

(2) Using an ArrayList to build an ordered collection.

Building and growing	O(N)
Searching	O(N Log N)
Inserting	O(N^2)
--	
Total	O(N^2)

Now, looking at the above asymptotic analysis, they still might stick with Linked List.

Below we have the run time analysis for four different data structures.

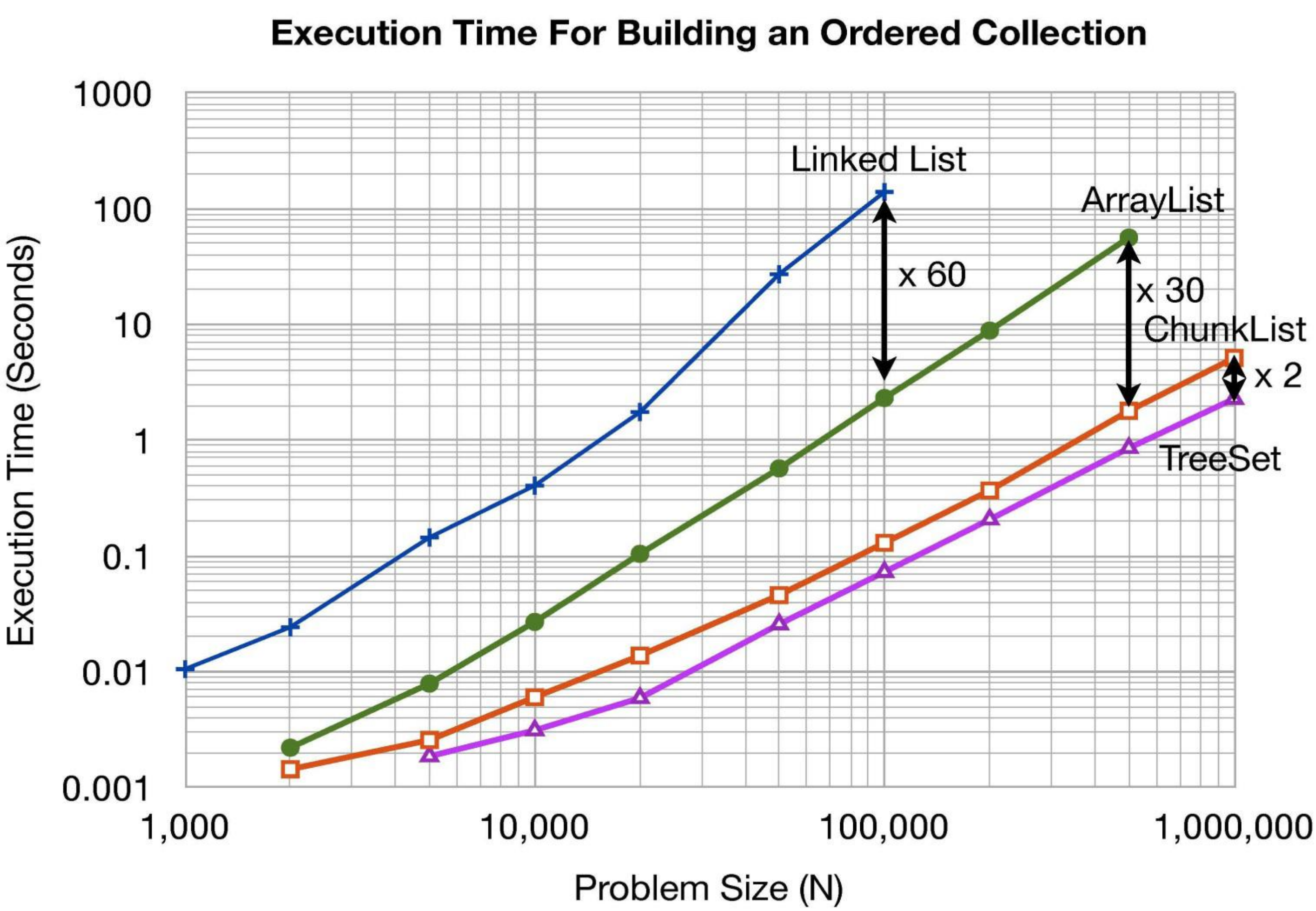


Fig 3.1 Execution time for building an Ordered Collection

Expectation was that they were both O(N2), and thus they would perform similarly. Order analysis ignores constant factors, but a factor of 60 is startlingly large.

Two explanations for the performance difference.

A method call versus a memory access: For the linked list this was the searching time. Each insertion must linearly search down the list to find the insertion point. It involves (N2) method calls to the comparator.

In contrast, for the array list, the (N2) factor was the insertion time incurred as each item was inserted somewhere into the array and everything in the array above the insertion point was moved up to make room for the new item.

This involves N calls to System.arraycopy(). The difference is that arraycopy() is a tuned system routine very efficiently moving a contiguous block of memory.

Memory Footprint: An active memory footprint of 3 times larger for Java's LinkedList than its ArrayList. This means that the linked list is being pushed into lower levels of the memory hierarchy before the array list is.

Furthermore the array copies are nice sequential accesses that work well with caches and pre-fetching compared to the linked list accesses which are jumping all over memory.

In paper [2], Changhee Jung and Silvius Rus has explained the design and evaluation of Brainy, a new program analysis tool that automatically selects the best data structure for a given program on a specific micro architecture .

In this paper, the data structures' interface functions are instrumented to dynamically monitor how the data structure interacts with the application for a given input. The instrumentation records traces of various runtime characteristics including underlying architecture-specific events. These generated traces are analyzed and fed into an offline model constructed using machine learning; this model then selects the best data structure. That is, Brainy exploits runtime feedback of data structures to understand the situation an application runs on, and selects the best data structure for a given application/input/architecture combination based on the constructed model. The empirical evaluation shows that this technique is highly accurate across several real-world applications with various program input sets on two different state-of-the-art microarchitectures. Consequently, Brainy achieved an average performance improvement of 27% and 33% on both microarchitectures, respectively.

DS	Alternate DS	Benefit	Limitation
vector	list	Fast insertion	None
	deque	Fast insertion	None
	set (map)	Fast search	Order-oblivious
	avl_set (avl_map)	Fast search	Order-oblivious
	hash_set (hash_map)	Fast insertion & search	Order-oblivious
list	vector	Fast iteration	None
	deque	Fast iteration	None
	set (map)	Fast search	Order-oblivious
	avl_set (avl_map)	Fast search	Order-oblivious
	hash_set (hash_map)	Fast search	Order-oblivious
set	avl_set	Fast search	None
	vector	Fast iteration	Order-oblivious
	list	Fast insertion & deletion	Order-oblivious
	hash_set	Fast insertion & search	Order-oblivious
map	avl_map	Fast search	None
	hash_map	Fast insertion & search	Order-oblivious

Table 3.1:Data structure replacements considered for each target data structure.

The purpose of this work is to provide a tool that can report the best data structures for different situations due to specific input sets and underlying hardware architecture changes. To keep up with the various behaviors of an application, this work exploits dynamic profiling that utilizes runtime instrumentation. Every interface function of each data structure is instrumented to model how that data structure interacts with the application. The instrumentation code observes how the data structure is used by the application (i.e, software features), and at the same time monitors a set of performance counters (i.e., hardware features) from the underlying architecture. The runtime system maintains the trace information in a context-sensitive manner, i.e., the calling sequences are considered at the data structure's construction time. This helps developers know the location in the source code of the data structures to be replaced.

In paper [3] (CoCo: Sound and Adaptive Replacement of Java Collections.),Guoquing Xu, University of California, Irvine,CA,USA has explained an application-level dynamic optimization technique called CoCo, that exploits algorithmic advantages of Java collections to improve performance. CoCo dynamically identifies optimal Java collection objects and safely performs runtime collection replacement, both using pure Java code. It uses a framework that abstracts container elements to achieve efficiency and that concretizes abstractions to achieve soundness. They implemented part of the Java collection framework as instances of this framework, and developed a static CoCo compiler to generate Java code that performs optimizations.

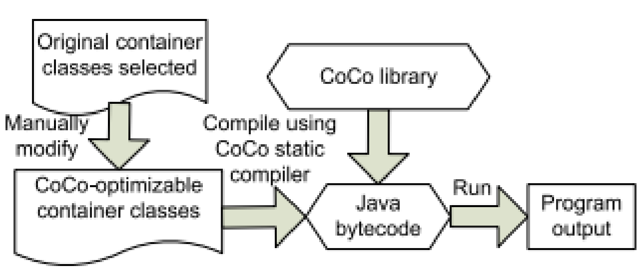


Fig 3.2: Overview of CoCo system

The paper proposes a novel container optimization technique, called CoCo, that is able to determine at run time, for each container instance (e.g., a LinkedList object) used in the program, whether or not there exists another container implementation (e.g., ArrayList) that is more suitable for the execution; and (2) automatically and safely switch to this new container implementation (e.g., replace the old LinkedList object with a new ArrayList object online) for increased efficiency.

CoCo performs same-interface optimizations—given a container class that implements interface I, CoCo looks for potentially switchable candidates only within the types that implement I. Performance gains may be seen sometimes from switching implementations across different interfaces. For example, if it is beneficial to switch from ArrayList to LinkedHashSet,it would be easy to extend CoCo to support multi-interface switches— the developer may need to create a wrapper class that serves as an adapter between interfaces. This class implements APIs of the original interface using methods of the new interface. From a set of all container classes that implement the same interface, it selects those among which the online replacement may result in large performance benefit (at least large enough to offset the replacement overhead). In this paper, they focus on containers that have clear algorithmic advantages (e.g., lower worst-case complexity) over others in certain execution scenarios. For example, switching from a LinkedList to an ArrayList upon experiencing many calls to method get(i) may reduce the complexity of get from O(n) (where n is the size of this List) to O(1). This may have much larger benefit than switching from ArrayList to SingletonList (upon observing there is always one single element)—in this case, no significant algorithmic advantage can be exploited and the benefit resulting from space reduction may not be sufficient to offset the overhead of creating and maintaining multiple containers. For the selected container classes, they first modify them manually to add abstraction-concretization operations. The CoCo static compiler then generates glue code that connects these modified classes, performs run-time profiling, and makes replacement decisions. Next, both the generated glue classes and the modified container classes are compiled into Java bytecode, which is executed to enable optimizations.

4. PROJECT REQUIREMENT SPECIFICATION

4.1 GANTT CHART

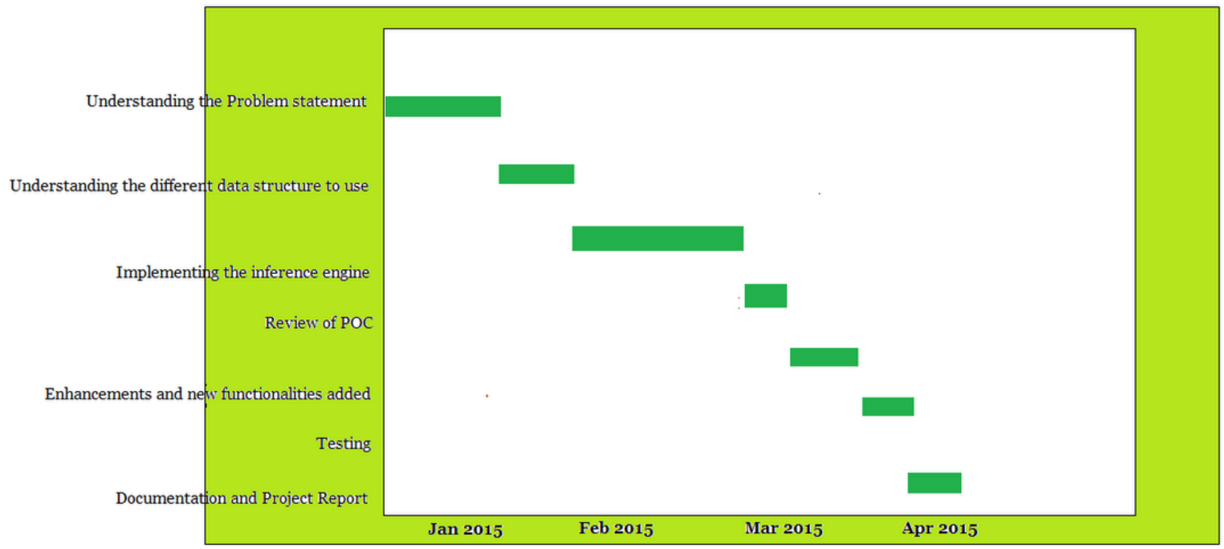


Fig 4.1 Gantt chart

4.2 ACTIVITY DIAGRAM

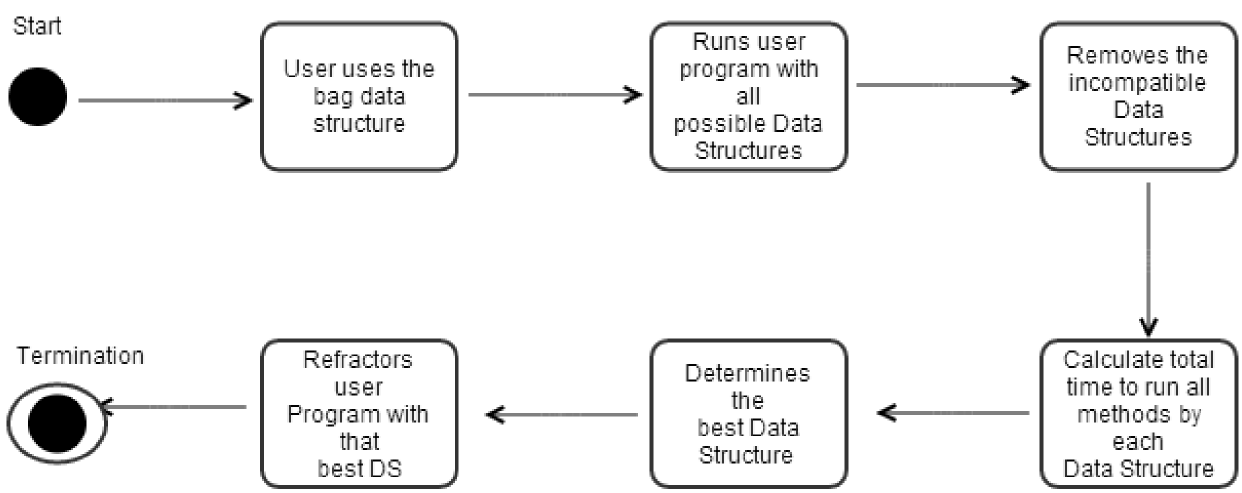


Fig 4.2 Activity Diagram

The above figure gives description about the system architecture which is followed in our project. The figure shows that the user program uses the bag program which internally runs the user program with all possible data structures. The incompatible data structures are removed and the total time taken to run all methods by each data structure is calculated. Finally, the best data structure (the one which least time taken) is chosen and the user program is refactored with the optimal data structure.

4.3 WORKFLOW DIAGRAM

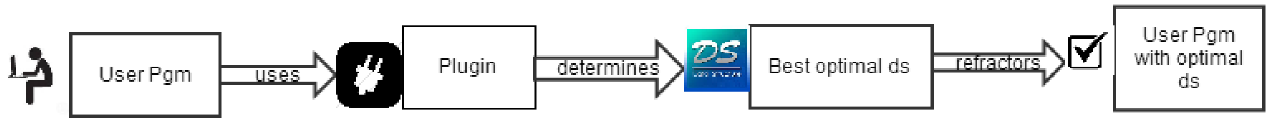


Fig 4.3 Workflow Diagram

5. SYSTEM REQUIREMENT SPECIFICATION

5.1 Hardware Requirements

The hardware requirements of the project are summarized in the following table

Sl No	**Parameter**	**Description**
1	RAM	500MB-1GB
2	Hard Disk	120GB-160GB
3	Operating System	Windows, Linux, MAC

Table 5.1 Hardware Requirements

5.2. Software Requirements

The software requirements is summarized in the following table

Sl No	Parameter Name	Parameter Value
1	Development Language	JAVA
2	Java Development Kit Version	JDK 1.8
3	Java Run Time Environment	JRE 6
4	Integrated Development Environment	Eclipse Luna
5	Jar File	Bag.jar
6	Jar File	BagPair.jar

Table 5.2 Software Requirements

5.3 Non-Functional Requirement

5.3.1 Assumptions and Dependencies

The following assumptions are made while developing the project

1. The User should be familiar with java programming, eclipse ide, and in-built java data structures.
2. The User should have the ability to install plugins into his eclipse ide.
3. The User should use the plugin only on programs that invoke the Bag class. If done on other programs, the results are non-determinate.
4. The User program should terminate smoothly, else the results are non-determinate.

5.3.2 Use Case Diagram

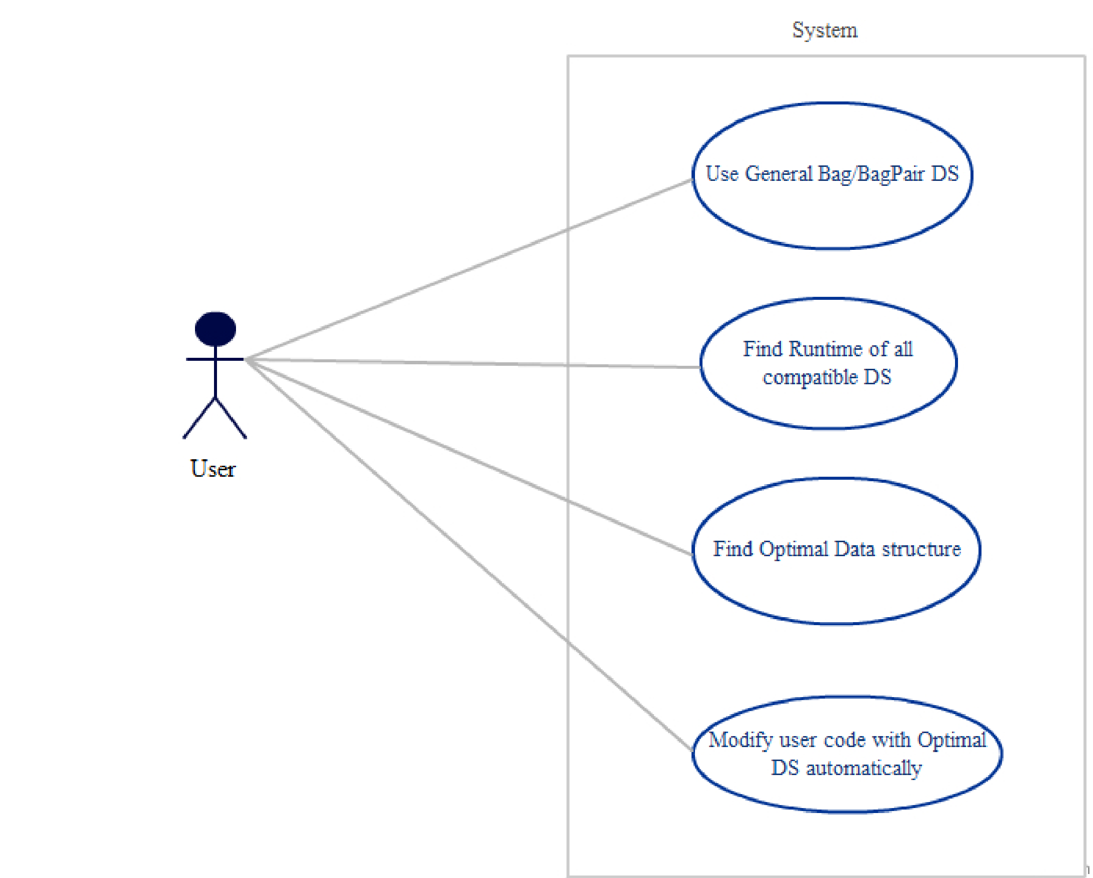


Fig 5.3.2 Use Case Diagram

6. SYSTEM DESIGN

6.1 BLOCK DIAGRAM

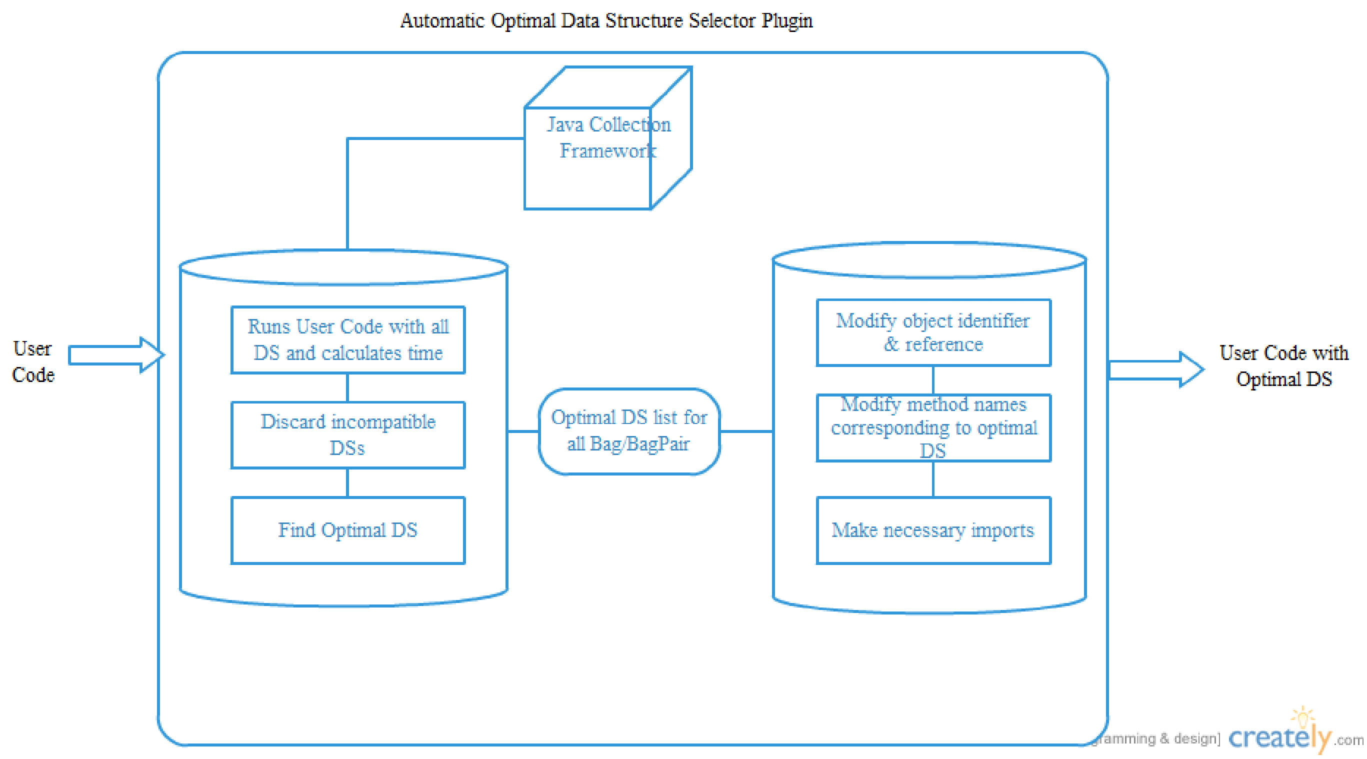


Fig 6.1 Block Diagram

6.2 Implemented Modules

Module 1: Optimal Data Structure Selector

The user code using Bag/BagPair Data Structure is passed to this module where the code is run through different possible data structures of the Java Collection Framework. Total time to run each method by a particular data structure is calculated. The incompatible data structures are discarded. Data structure with minimum runtime is selected as the optimal one.

Module 2: User Code Modifier with Optimal Data Structure

The user code using Bag/BagPair Data Structure is modified by using optimal data structure. First the Object identifiers & Reference names are modified first. Then the method names are modified to match it with the methods of the optimal data structure. Necessary imports of the packages are made.

6.3 DATAFLOW DIAGRAM (LEVEL 0)

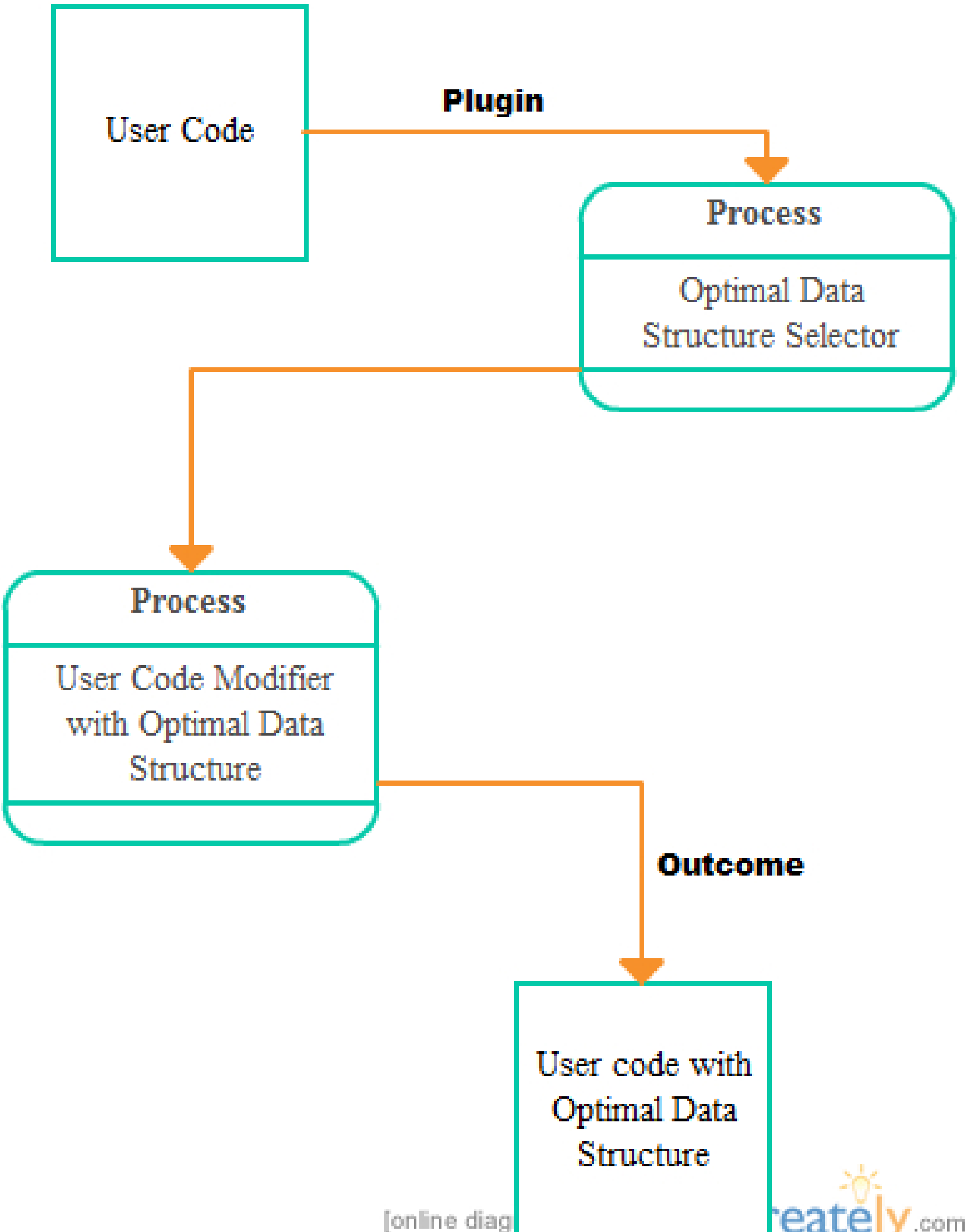


Fig 6.3 Dataflow Diagram

7. DETAILED DESIGN

7.1 Module 1

Module Name: Optimal Data Structure Selector

Input: The user code using Bag/BagPair Data Structure

Description: A set of data structures from the Java Collection Framework is selected. The user code is made to run with all data structures from that set. The incompatible data structures which doesnot support particular method which the user has used are discarded from that set. Time required to run each method by a data structure is calculated. Also total time required to run all the methods by a particular data structure is also computed. Finally the data structure with minimum runtime is selected as the optimal data structure.

Ouput: Optimal datastructures for each Bags or PairBags

PTO

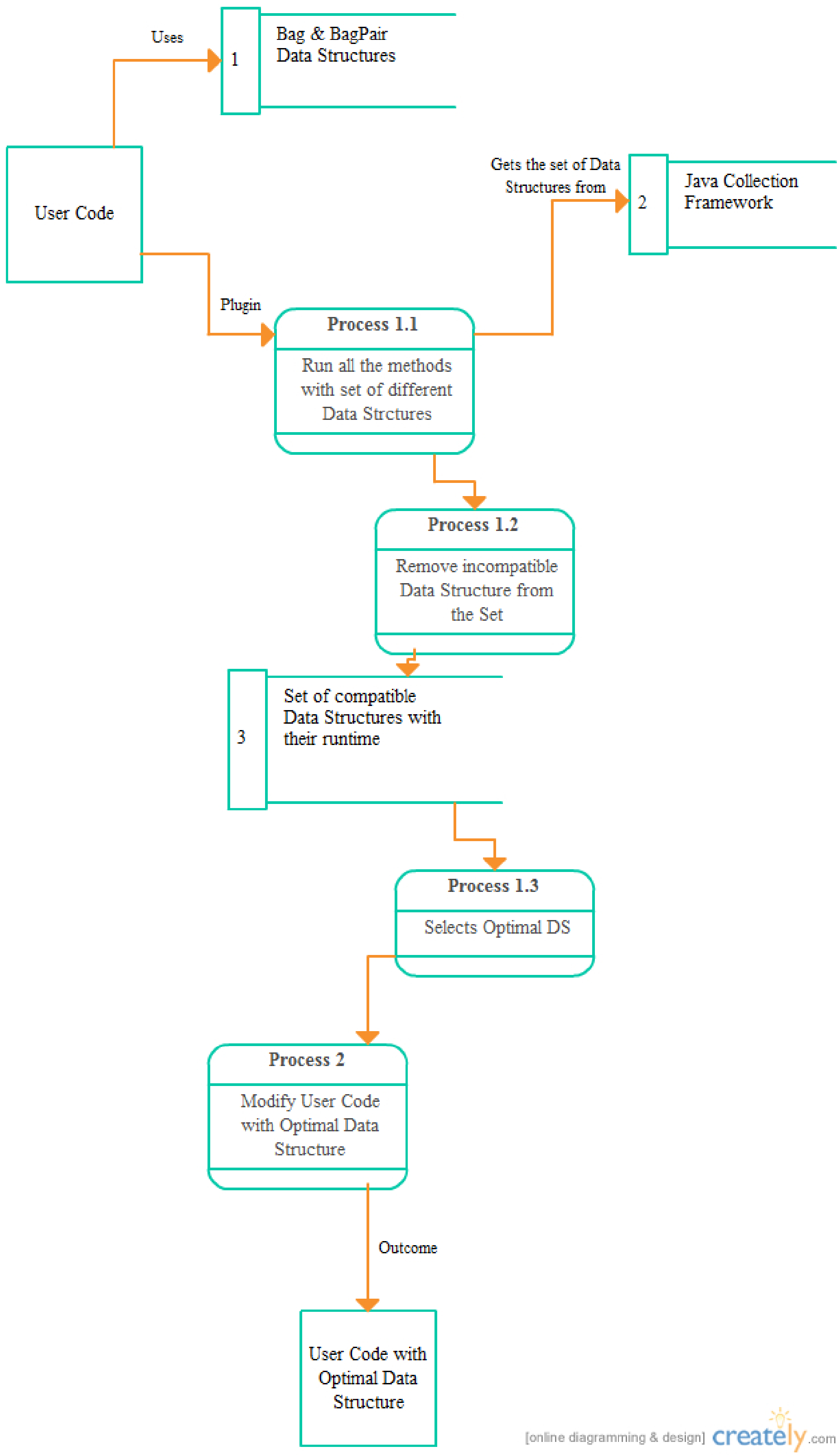


Fig 7.1 Data Flow Diagram (Level 1)

7.2 Module 2

Module Name: User Code Modifier with Optimal Data Structure

Input: The user code using Bag/BagPair Data Structure & List of Optimal DataStructures for each Bag/BagPair

Description: The Bag/BagPair object identifiers & References are modified with Optimal data structure object identifiers & referneces. The method names are also modified to match it with the method names of the optimal datastructure. This is done from a mapping of Bag/BagPair method names to particular datastructure methods. Necessary packages from the java library are imported.

Output: The user code with Optimal Data Structures.

PTO

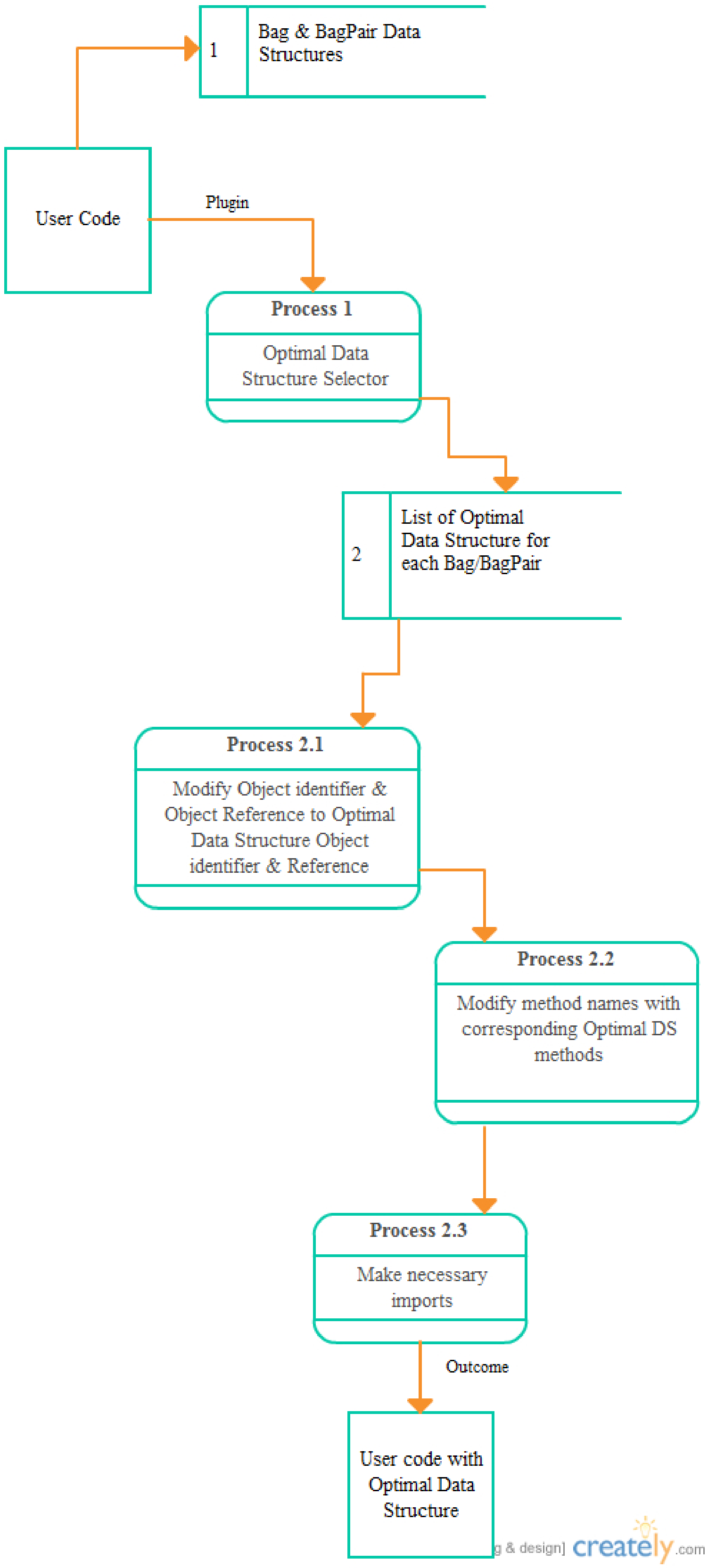


Fig 7.2 Dataflow Diagram (Level 2)

8. IMPLEMENTATION/ PSEUDOCODE

8.1 Pseudo-code

Input: User Program which uses Bag DS

Step 1: Create a hashmap of different data structure objects for each Bag object.

Step 2: Let Name of the DS as the key and initial time 0.0 as value in that hashmap.

Step 3: Run each method with all the DS objects in that hashmap.

Step 4: Each method has proper mapping for every DS and the DS which doesnot provide mapping for that particular method is removed from the hashmap.

Step 5: Calculate time to run the methods in every Data Structures.

Step 6: Calculate total time to run all the methods by each Data Structure.

Step 7: Find the Optimal Data Structure.

Step 8: Find all the object names in the user program.

Step 9: Create mappings of object name & optimal ds object name for that object.

Step 10: Modify object names with corresponding optimal ds object names.

Step 11: Modify Object Reference names with the optimal data structure Class's Reference name.

Step 12: Modify the method names to match it with the methods of optimal data structure class.

Step 13: Import all the required packages.

Output: User Code with Optimal Data Structure.

8.2 IMPLEMENTATION

Bag's Constructor

```
Bag()
    {
        ++num_bags;
        obj_list.add(this);
        bag_id=num_bags;
        ideal_ds=null;
        data_structures = new HashMap<String,Double>();
        stack = new Stack<E>();
        linked_list = new LinkedList<E>();
        array_list = new ArrayList<E>();
        vector = new Vector<E>();
        hash_set = new HashSet<E>();
        tree_set = new TreeSet<E>();
        linked_hash_set = new LinkedHashSet<E>();
        array_deque = new ArrayDeque<E>();
        priority_queue = new PriorityQueue<E>();
        data_structures.put("Stack",new Double(0.00));
        data_structures.put("LinkedList",new Double(0.00));
        data_structures.put("ArrayList",new Double(0.00));
        data_structures.put("Vector",new Double(0.00));
        data_structures.put("HashSet",new Double(0.00));
        data_structures.put("TreeSet",new Double(0.00));
        data_structures.put("LinkedHashSet",new Double(0.00));
        data_structures.put("ArrayDeque",new Double(0.00));
```

```
			data_structures.put("PriorityQueue",new Double(0.00));
		}

/*
		 * This method removes the unwanted ds from the ds list
*/
		public void remove_unwanted_ds(ArrayList<String> l)
		{
			for(int i=0; i<l.size(); ++i)
			{
				System.out.println(l.get(i)+" is removed from the 
data_structure list");
				data_structures.remove(l.get(i));
			}
		}

		/*
		 * Inserts the specified element at the specified position
		 */
		public void insert(int index, E element)
		{
			ArrayList<String> unwanted_ds= new ArrayList<String>();
			for(Object key : data_structures.keySet())
			{
				String ds = (String)key;
				long start,end;
				if(ds.hashCode() == "Stack".hashCode() && 
ds.equals("Stack"))
				{
					// Start the Timer
					start = System.nanoTime();

					// Perform Operation
					stack.insertElementAt(element, index);

					// Stop the Timer
					end = System.nanoTime();

					// Get the time difference
					double time_taken = (end-start)/1000.0;

					// Add the time taken to perform the operation to 
the respective ds
```

```
                data_structures.put(ds,
data_structures.get(key)+time_taken);
            }
            else if(ds.hashCode() == "LinkedList".hashCode() &&
ds.equals("LinkedList"))
            {
                // Start the Timer
                start = System.nanoTime();

                // Perform Operation
                linked_list.add(index, element);

                // Stop the Timer
                end = System.nanoTime();

                // Get the time difference
                double time_taken = (end-start)/1000.0;

                // Add the time taken to perform the operation to
the respective ds
                data_structures.put(ds,
data_structures.get(key)+time_taken);
            }
            else if(ds.hashCode() == "ArrayList".hashCode() &&
ds.equals("ArrayList"))
            {
                // Start the Timer
                start = System.nanoTime();

                // Perform Operation
                array_list.add(index, element);

                // Stop the Timer
                end = System.nanoTime();

                // Get the time difference
                double time_taken = (end-start)/1000.0;

                // Add the time taken to perform the operation to
the respective ds
                data_structures.put(ds,
data_structures.get(key)+time_taken);
            }
            else if(ds.hashCode() == "Vector".hashCode() &&
ds.equals("Vector")
```

```
			{
				// Start the Timer
				start = System.nanoTime();

				// Perform Operation
				vector.insertElementAt(element, index);

				// Stop the Timer
				end = System.nanoTime();

				// Get the time difference
				double time_taken = (end-start)/1000.0;

				// Add the time taken to perform the operation to
the respective ds
				data_structures.put(ds,
data_structures.get(key)+time_taken);
			}

			else
			{
				System.out.println("Does not support insert
operation for "+ds+" datastructure");

				// Add this ds to unwanted list
				unwanted_ds.add(ds);
			}
		}

		// Remove Unwanted ds
		remove_unwanted_ds(unwanted_ds);
	}

	/*
	 * Inserts the specified element at the beginning of this list.
	 */
	public void insertAtFirst(E element)
	{

		ArrayList<String> unwanted_ds= new ArrayList<String>();
		for(Object key : data_structures.keySet())
		{
			String ds = (String)key;
```

```
long start,end;
if(ds.hashCode() == "LinkedList".hashCode() &&
ds.equals("LinkedList"))
{
    // Start the Timer
    start = System.nanoTime();

    // Perform Operation
    linked_list.addFirst(element);

    // Stop the Timer
    end = System.nanoTime();

    // Get the time difference
    double time_taken = (end-start)/1000.0;

    // Add the time taken to perform the operation to
the respective ds
    data_structures.put(ds,
data_structures.get(key)+time_taken);
}
else if(ds.hashCode() == "HashSet".hashCode() &&
ds.equals("HashSet"))
{
    // Start the Timer
    start = System.nanoTime();

    // Perform Operation
    hash_set.add(element);

    // Stop the Timer
    end = System.nanoTime();

    // Get the time difference
    double time_taken = (end-start)/1000.0;

    // Add the time taken to perform the operation to
the respective ds
    data_structures.put(ds,
data_structures.get(key)+time_taken);
}
else if(ds.hashCode() == "TreeSet".hashCode() &&
ds.equals("TreeSet"))
{
    // Start the Timer
```

```
				start = System.nanoTime();

				// Perform Operation
				tree_set.add(element);

				// Stop the Timer
				end = System.nanoTime();

				// Get the time difference
				double time_taken = (end-start)/1000.0;

				// Add the time taken to perform the operation to
the respective ds
				data_structures.put(ds,
data_structures.get(key)+time_taken);
			}
			else if(ds.hashCode() == "LinkedHashSet".hashCode() &&
ds.equals("LinkedHashSet"))
			{
				// Start the Timer
				start = System.nanoTime();

				// Perform Operation
				tree_set.add(element);

				// Stop the Timer
				end = System.nanoTime();

				// Get the time difference
				double time_taken = (end-start)/1000.0;

				// Add the time taken to perform the operation to
the respective ds
				data_structures.put(ds,
data_structures.get(key)+time_taken);
			}

			else if(ds.hashCode() == "ArrayDeque".hashCode() &&
ds.equals("ArrayDeque"))
			{
				// Start the Timer
				start = System.nanoTime();

				// Perform Operation
```

```
				array_deque.addFirst(element);

				// Stop the Timer
				end = System.nanoTime();

				// Get the time difference
				double time_taken = (end-start)/1000.0;

				// Add the time taken to perform the operation to
the respective ds
				data_structures.put(ds,
data_structures.get(key)+time_taken);
			}
			else
			{
				System.out.println("Does not support insertAtFirst
operation for "+ds+" datastructure");

				// Add this ds to unwanted list
				unwanted_ds.add(ds);
			}
		}

		// Remove Unwanted ds
		remove_unwanted_ds(unwanted_ds);

	}

	/*
	 * Appends the specified element to the end of this list.
	 */
	public void insertAtLast(E element)
	{
		ArrayList<String> unwanted_ds= new ArrayList<String>();
		for(Object key : data_structures.keySet())
		{
			String ds = (String)key;
			long start,end;
			if(ds.hashCode() == "Stack".hashCode() &&
ds.equals("Stack"))
			{
				// Start the Timer
				start = System.nanoTime();
```

```
                    // Perform Operation
                    stack.push(element);

                    // Stop the Timer
                    end = System.nanoTime();

                    // Get the time difference
                    double time_taken = (end-start)/1000.0;

                    // Add the time taken to perform the operation to
the respective ds
                    data_structures.put(ds,
data_structures.get(key)+time_taken);
               }
               else if(ds.hashCode() == "LinkedList".hashCode() &&
ds.equals("LinkedList"))
               {
                    // Start the Timer
                    start = System.nanoTime();

                    // Perform Operation
                    linked_list.addLast(element);

                    // Stop the Timer
                    end = System.nanoTime();

                    // Get the time difference
                    double time_taken = (end-start)/1000.0;

                    // Add the time taken to perform the operation to
the respective ds
                    data_structures.put(ds,
data_structures.get(key)+time_taken);

               }
               else if(ds.hashCode() == "ArrayList".hashCode() &&
ds.equals("ArrayList"))
               {
                    // Start the Timer
                    start = System.nanoTime();

                    // Perform Operation
                    array_list.add(element);
```

```
					// Stop the Timer
					end = System.nanoTime();

					// Get the time difference
					double time_taken = (end-start)/1000.0;

					// Add the time taken to perform the operation to
the respective ds
					data_structures.put(ds,
data_structures.get(key)+time_taken);
				}
				else if(ds.hashCode() == "Vector".hashCode() &&
ds.equals("Vector"))
				{
					// Start the Timer
					start = System.nanoTime();

					// Perform Operation
					vector.add(element);

					// Stop the Timer
					end = System.nanoTime();

					// Get the time difference
					double time_taken = (end-start)/1000.0;

					// Add the time taken to perform the operation to
the respective ds
					data_structures.put(ds,
data_structures.get(key)+time_taken);
				}

				else if(ds.hashCode() == "ArrayDeque".hashCode() &&
ds.equals("ArrayDeque"))
				{
					// Start the Timer
					start = System.nanoTime();

					// Perform Operation
					array_deque.addLast(element);

					// Stop the Timer
					end = System.nanoTime();

					// Get the time difference
```

```
                                double time_taken = (end-start)/1000.0;

                                // Add the time taken to perform the operation to
the respective ds
                                data_structures.put(ds,
data_structures.get(key)+time_taken);
                        }
                        else if(ds.hashCode() == "PriorityQueue".hashCode() &&
ds.equals("PriorityQueue"))
                        {
                                // Start the Timer
                                start = System.nanoTime();

                                // Perform Operation
                                priority_queue.add(element);

                                // Stop the Timer
                                end = System.nanoTime();

                                // Get the time difference
                                double time_taken = (end-start)/1000.0;

                                // Add the time taken to perform the operation to
the respective ds
                                data_structures.put(ds,
data_structures.get(key)+time_taken);
                        }
                        else
                        {
                                System.out.println("Does not support insertAtLast
operation for "+ds+" datastructure");

                                // Add this ds to unwanted list
                                unwanted_ds.add(ds);
                        }
                }

                // Remove Unwanted ds
                remove_unwanted_ds(unwanted_ds);
        }

public static String find_modified_method(String ds, String method) {
                String mod_method = method;
```

```
		if(method.hashCode() == "insert".hashCode() &&
method.equals("insert"))
		{
			if(ds.hashCode() == "Stack".hashCode() &&
ds.equals("Stack"))
				mod_method = "insertElementAt";
			else if(ds.hashCode() == "LinkedList".hashCode() &&
ds.equals("LinkedList"))
				mod_method = "add";
			else if(ds.hashCode() == "ArrayList".hashCode() &&
ds.equals("ArrayList"))
				mod_method = "add";
			else if(ds.hashCode() == "Vector".hashCode() &&
ds.equals("Vector"))
				mod_method = "insertElementAt";
		}
		else if(method.hashCode() == "insertAtFirst".hashCode() &&
method.equals("insertAtFirst"))
		{
			if(ds.hashCode() == "LinkedList".hashCode() &&
ds.equals("LinkedList"))
				mod_method = "addFirst";
			else if(ds.hashCode() == "HashSet".hashCode() &&
ds.equals("HashSet"))
				mod_method = "add";
			else if(ds.hashCode() == "TreeSet".hashCode() &&
ds.equals("TreeSet"))
				mod_method = "add";
			else if(ds.hashCode() == "LinkedHashSet".hashCode() &&
ds.equals("LinkedHashSet"))
				mod_method = "add";
			else if(ds.hashCode() == "ArrayDeque".hashCode() &&
ds.equals("ArrayDeque"))
				mod_method = "add";
			else if(ds.hashCode() == "PriorityQueue".hashCode() &&
ds.equals("PriorityQueue"))
				mod_method = "addFirst";
		}
		else if(method.hashCode() == "insertAtLast".hashCode() &&
method.equals("insertAtLast"))
		{
			if(ds.hashCode() == "Stack".hashCode() &&
ds.equals("Stack"))
				mod_method = "push";
```

```
                        else if(ds.hashCode() == "LinkedList".hashCode() &&
ds.equals("LinkedList"))
                                mod_method = "addLast";
                        else if(ds.hashCode() == "ArrayList".hashCode() &&
ds.equals("ArrayList"))
                                mod_method = "add";
                        else if(ds.hashCode() == "Vector".hashCode() &&
ds.equals("Vector"))
                                mod_method = "add";
                        else if(ds.hashCode() == "ArrayDeque".hashCode() &&
ds.equals("ArrayDeque"))
                                mod_method = "addLast";
                        else if(ds.hashCode() == "PriorityQueue".hashCode() &&
ds.equals("PriorityQueue"))
                                mod_method = "add";
                }
                else if(method.hashCode() == "insertAll".hashCode() &&
method.equals("insertAll"))
                {
                        mod_method = "addAll";
                }
                else if(method.hashCode() == "get".hashCode() &&
method.equals("get"))
                {
                        if(ds.hashCode() == "Stack".hashCode() &&
ds.equals("Stack"))
                                mod_method = "get";
                        else if(ds.hashCode() == "LinkedList".hashCode() &&
ds.equals("LinkedList"))
                                mod_method = "get";
                        else if(ds.hashCode() == "ArrayList".hashCode() &&
ds.equals("ArrayList"))
                                mod_method = "get";
                        else if(ds.hashCode() == "Vector".hashCode() &&
ds.equals("Vector"))
                                mod_method = "get";
                }
                else if(method.hashCode() == "getFirst".hashCode() &&
method.equals("getFirst"))
                {
                        if(ds.hashCode() == "LinkedList".hashCode() &&
ds.equals("LinkedList"))
                                mod_method = "getFirst";
                        else if(ds.hashCode() == "Vector".hashCode() &&
ds.equals("Vector"))
```

```
					mod_method = "firstElement";
				else if(ds.hashCode() == "ArrayDeque".hashCode() &&
ds.equals("ArrayDeque"))
					mod_method = "getFirst";
				else if(ds.hashCode() == "PriorityQueue".hashCode() &&
ds.equals("PriorityQueue"))
					mod_method = "peek";
			}
			else if(method.hashCode() == "getLast".hashCode() &&
method.equals("getLast"))
			{
				if(ds.hashCode() == "Stack".hashCode() &&
ds.equals("Stack"))
					mod_method = "peek";
				else if(ds.hashCode() == "LinkedList".hashCode() &&
ds.equals("LinkedList"))
					mod_method = "getLast";
				else if(ds.hashCode() == "Vector".hashCode() &&
ds.equals("Vector"))
					mod_method = "lastElement";
				else if(ds.hashCode() == "ArrayDeque".hashCode() &&
ds.equals("ArrayDeque"))
					mod_method = "getLast";
			}
			else if(method.hashCode() == "lastIndexOf".hashCode() &&
method.equals("lastIndexOf"))
			{
				if(ds.hashCode() == "Stack".hashCode() &&
ds.equals("Stack"))
					mod_method = "lastIndexOf";
				else if(ds.hashCode() == "LinkedList".hashCode() &&
ds.equals("LinkedList"))
					mod_method = "lastIndexOf";
				else if(ds.hashCode() == "ArrayList".hashCode() &&
ds.equals("ArrayList"))
					mod_method = "lastIndexOf";
				else if(ds.hashCode() == "Vector".hashCode() &&
ds.equals("Vector"))
					mod_method = "lastIndexOf";
			}
			else if(method.hashCode() == "indexOf".hashCode() &&
method.equals("indexOf"))
			{
				if(ds.hashCode() == "Stack".hashCode() &&
ds.equals("Stack"))
```

```
                mod_method = "indexOf";
            else if(ds.hashCode() == "LinkedList".hashCode() &&
ds.equals("LinkedList"))
                mod_method = "indexOf";
            else if(ds.hashCode() == "ArrayList".hashCode() &&
ds.equals("ArrayList"))
                mod_method = "indexOf";
            else if(ds.hashCode() == "Vector".hashCode() &&
ds.equals("Vector"))
                mod_method = "indexOf";
        }
        else if(method.hashCode() == "clear".hashCode() &&
method.equals("clear"))
        {
            mod_method = "clear";
        }
        else if(method.hashCode() == "isEmpty".hashCode() &&
method.equals("isEmpty"))
        {
            mod_method = "isEmpty";
        }
        else if(method.hashCode() == "remove".hashCode() &&
method.equals("remove"))
        {
            mod_method = "remove";
        }
        else if(method.hashCode() == "removeAt".hashCode() &&
method.equals("removeAt"))
        {
            if(ds.hashCode() == "Stack".hashCode() &&
ds.equals("Stack"))
                mod_method = "remove";
            else if(ds.hashCode() == "LinkedList".hashCode() &&
ds.equals("LinkedList"))
                mod_method = "remove";
            else if(ds.hashCode() == "ArrayList".hashCode() &&
ds.equals("ArrayList"))
                mod_method = "remove";
            else if(ds.hashCode() == "Vector".hashCode() &&
ds.equals("Vector"))
                mod_method = "remove";
        }
```

```
		else if(method.hashCode() == "removeFirst".hashCode() &&
method.equals("removeFirst"))
		{
			if(ds.hashCode() == "LinkedList".hashCode() &&
ds.equals("LinkedList"))
				mod_method = "removeFirst";
			else if(ds.hashCode() == "ArrayDeque".hashCode() &&
ds.equals("ArrayDeque"))
				mod_method = "removeFirst";
			else if(ds.hashCode() == "PriorityQueue".hashCode() &&
ds.equals("PriorityQueue"))
				mod_method = "remove";
		}
		else if(method.hashCode() == "removeLast".hashCode() &&
method.equals("removeLast"))
		{
			if(ds.hashCode() == "Stack".hashCode() &&
ds.equals("Stack"))
				mod_method = "pop" ;
			else if(ds.hashCode() == "LinkedList".hashCode() &&
ds.equals("LinkedList"))
				mod_method = "removeLast";
			else if(ds.hashCode() == "ArrayDeque".hashCode() &&
ds.equals("ArrayDeque"))
				mod_method = "removeLast";
		}
		else if(method.hashCode() == "removeAll".hashCode() &&
method.equals("removeAll"))
		{
			mod_method = "removeAll";
		}
		else if(method.hashCode() == "replace".hashCode() &&
method.equals("replace"))
		{
			if(ds.hashCode() == "Stack".hashCode() &&
ds.equals("Stack"))
				mod_method = "set";
			else if(ds.hashCode() == "LinkedList".hashCode() &&
ds.equals("LinkedList"))
				mod_method = "set";
			else if(ds.hashCode() == "ArrayList".hashCode() &&
ds.equals("ArrayList"))
				mod_method = "set";
			else if(ds.hashCode() == "Vector".hashCode() &&
ds.equals("Vector"))
```

```
                mod_method = "set";
        }
        else if(method.hashCode() == "sort".hashCode() &&
method.equals("sort"))
        {
            if(ds.hashCode() == "Stack".hashCode() &&
ds.equals("Stack"))
                mod_method = "sort";
            else if(ds.hashCode() == "LinkedList".hashCode() &&
ds.equals("LinkedList"))
                mod_method = "sort";
            else if(ds.hashCode() == "ArrayList".hashCode() &&
ds.equals("ArrayList"))
                mod_method = "sort";
            else if(ds.hashCode() == "Vector".hashCode() &&
ds.equals("Vector"))
                mod_method = "sort";
        }
        else if(method.hashCode() == "subList".hashCode() &&
method.equals("subList"))
        {
            if(ds.hashCode() == "Stack".hashCode() &&
ds.equals("Stack"))
                mod_method = "subList";
            else if(ds.hashCode() == "LinkedList".hashCode() &&
ds.equals("LinkedList"))
                mod_method = "subList";
            else if(ds.hashCode() == "ArrayList".hashCode() &&
ds.equals("ArrayList"))
                mod_method = "subList";
            else if(ds.hashCode() == "Vector".hashCode() &&
ds.equals("Vector"))
                mod_method = "subList";
        }
        else if(method.hashCode() == "size".hashCode() &&
method.equals("size"))
        {
            mod_method = "size";
        }
        else if(method.hashCode() ==
"insertedAtSortedOrder".hashCode() &&
method.equals("insertedAtSortedOrder"))
        {
```

```
                    if(ds.hashCode() == "TreeSet".hashCode() &&
ds.equals("TreeSet"))
                              mod_method = "add";
            }
            return mod_method;
      }

      public void analyze()
      {
            Entry<String, Double> minEntry = null;
            System.out.println("\nAnalysis: Bag"+this.bag_id+"\n-----------
------");
            for (Entry<String, Double> entry : data_structures.entrySet())
            {
                    System.out.printf("Data Structure: %s time_taken: %.2f
milli_sec\n",entry.getKey(),entry.getValue());
                if (minEntry == null ||
entry.getValue().compareTo(minEntry.getValue()) < 0)
                {
                    minEntry = entry;
                }
            }
            ideal_ds = minEntry.getKey();
            optimal_bag_list.put(bag_id, ideal_ds);
      }

      public static void optimize() throws IOException
      {
            for(int i=0;i<obj_list.size();++i)
            {
                    obj_list.get(i).analyze();
            }
            Bag.modify_code();
      }
      /*
       * This method returns array of Strings which contain ideal ds of each
bag
       * Point to be noted is, bagId starts from zero whereas this array index
starts from 1
       * So, while printing to the console, plz print 'Bag'+(index+1)....
       */
      public static String[] printIdealDSList()
      {
            String[] idealDSList = new String[num_bags];
            int i=0;
```

```
		for (Entry<Integer, String> entry :
Bag.optimal_bag_list.entrySet())
		{
			idealDSList[i++] = entry.getValue();
		}
		return idealDSList;
	}
	public static void modify_code() throws IOException
	{
		System.out.println("\nOptimal DS\n----------------");
		for (Entry<Integer, String> entry :
Bag.optimal_bag_list.entrySet())
		{
			System.out.println("Bag-"+entry.getKey()+"--
>"+entry.getValue());
		}

		ds_names = new HashMap<String,String>();
		ds_names.put("Stack","stack");
		ds_names.put("LinkedList", "linked_list");
		ds_names.put("ArrayList", "array_list");
		ds_names.put("Vector", "vector");
		ds_names.put("HashSet", "hash_set");
		ds_names.put("TreeSet", "tree_set");
		ds_names.put("LinkedHashSet", "linked_hashset");
		ds_names.put("ArrayDeque", "array_deque");
		ds_names.put("PriorityQueue", "priority_queue");

		// Path of User file.
		Path path = Paths.get("C:\\Users\\Sandeep K
V\\workspace\\SuperBag\\src\\Test1.java");

		// UTF Charset.
		Charset charset = StandardCharsets.UTF_8;

		// Read contents of User File
		String content = new String(Files.readAllBytes(path), charset);
		in = new Scanner(content);

		// This stores object names of user program
		String obj_names[] = new String[Bag.num_bags];
		int index=0;

		/*
```

```
             * To find Existing Object Names
             */
            while(in.hasNext())
            {
               String line=in.nextLine();
               if(Pattern.compile("\\bBag.*=").matcher(line).find())
               {

obj_names[index++]=line.substring(line.indexOf('>')+1,line.indexOf('=')).tr
im();
               }
            }

            // Object Names of Optimal DS
            String mod_bag_names[] = new String[Bag.num_bags];

            // Reference Names of Optimal DS
            String mod_class_names[] = new String[Bag.num_bags];


            /*
             * To modify existing Objects with optimal Objects
             */
            for(int n=0;n<num_bags;++n)
            {
                  mod_class_names[n] = optimal_bag_list.get(n+1);

                  mod_bag_names[n] =
ds_names.get(optimal_bag_list.get(n+1))+(n+1);
                  //System.out.println(obj_names[n]+"--
"+mod_bag_names[n]+"--"+mod_class_names[n]);
                  content = content.replaceAll("\\b"+obj_names[n]+"\\b",
mod_bag_names[n]);
                  //Files.write(path, content.getBytes(charset));
            }

            /*
             *  To modify Bag Reference with optimal DS reference
             */
            in = new Scanner(content);
            int count = 0;
            while(in.hasNext())
            {
               String line=in.nextLine();
               if(Pattern.compile("\\bBag<\\b").matcher(line).find())
```

```
                {
                    //System.out.println(line);
                    String mod_line =
line.replaceAll("Bag",mod_class_names[count]);

        if(!Pattern.compile("\\bjava.util."+mod_class_names[count]+"\\b").m
atcher(content).find())
                        content="import
java.util."+mod_class_names[count]+";\r\n"+content;
                    //System.out.println(mod_line);
                    content = content.replace(line, mod_line);
                    count++;
                }
            }

            /*
             * To Find Modified Method Names
             */
            for(int i=0 ; i < mod_bag_names.length ; i++)
            {
                in = new Scanner(content);
                while(in.hasNext())
                {
                    String line=in.nextLine();
                    Pattern pattern =
Pattern.compile("\\b"+mod_bag_names[i]+"\\.\\b");
                  Matcher matcher = pattern.matcher(line);
                  while (matcher.find()) {
                      int start = matcher.end();
                      int end = start;
                      for(; line.charAt(end) != '(' ; ++end);

                      // Existing methodname
                      String method_name = line.substring(start, end);

                      // Modified Methodname
                      String mod_method_name =
find_modified_method(mod_class_names[i],method_name);

                      //System.out.println(mod_class_names[i]+"
"+method_name+" "+mod_method_name);

                      String mod_line = line.replace(method_name,
mod_method_name);
```

```
                content = content.replace(line, mod_line);
            }
        }
    }

    /*
     * To remove analyse & modify methods.
     */
    String new_content = "";
    in = new Scanner(content);
    while(in.hasNext())
    {
        String line = in.nextLine();
        if(!Pattern.compile("\\banalyze\\b").matcher(line).find()
&& !Pattern.compile("\\bBag\\b").matcher(line).find())
            new_content+=line+"\r\n";
    }

    // Writes new content to the user file.
    Files.write(path, new_content.getBytes(charset));
}
```

/*

- BagPair's Constructor

/

```
BagPair()
{
    ++num_bags;
    bag_id=num_bags;
    ideal_ds=null;
    data_structures = new HashMap<String,Double>();
    hashmap = new HashMap<K,V>();
    treemap = new TreeMap<K,V>();
    linkedhashmap = new LinkedHashMap<K,V>();
    hashtable = new Hashtable<K,V>();
    data_structures.put("HashMap",new Double(0.00));
    data_structures.put("TreeMap",new Double(0.00));
    data_structures.put("LinkedHashMap",new Double(0.00));
    data_structures.put("Hashtable",new Double(0.00));
    obj_list.add(this);
}
```

9. TESTING

9.1 UNIT TESTING

Test Case No	Module	Test Case Description	Procedure	Expected Output	Result
TC_1	Bag package import option	To check whether Bag package can be imported in user's program	Import Bag package in user's program	Bag package is imported successfully and all the public features of Bag class is available to user's program	Pass
TC_2	Bag Object Creation	To check whether object for Bag class can be created from User program	Create Bag object from user's program Ex: Bag<Integer> = new Bag<Integer>();	Bag object is created & Constructor of Bag class is called	Pass

TC_3	BagPair package import option	To check whether BagPair package can be imported in user's program	Import BagPair package in user's program	BagPair package is imported successfully and all the public features of BagPair class is available to user's program	Pass
TC_4	BagPair Object Creation	To check whether object for BagPair class can be created from User program	Create Bag object from user's program Ex: BagPair<Integer,Integer> = new BagPair<Integer,Integer>();	BagPair object is created & Constructor of BagPair class is called	Pass
TC_5	Bag methods availabilty	To check whether methods of the Bag class are	Call the methods of Bag class from user's method	Methods can be called successfully with no	Pass

		available to user's program		compilation error	
TC_6	BagPair methods availabilty	To check whether methods of the BagPair class are available to user's program	Call the methods of BagPair class from user's method	Methods can be called successfully with no compilation error	Pass
TC_7	To check methods mapping	Check mappings for insert(x,i)	Call insert(x,i)	The method is properly mapped to different data structure's corresponding method	Pass
TC_8	To check methods mapping	Check mappings for insertAtFirst (x)	Call insertAtFirst(x)	The method is properly mapped to different data structure's	Pass

				corresponding method	
TC_9	To check methods mapping	Check mappings for insertAtLast(x)	Call insertAtLast(x)	The method is properly mapped to different data structure's corresponding method	Pass
TC_10	To check methods mapping	Check mappings for insertAll(Collection<? extends E> c)	Call insertAll(Collection<? extends E> c)	The method is properly mapped to different data structure's corresponding method	Pass
TC_11	To check methods mapping	Check mappings for get(index)	Call get(index)	The method is properly mapped to different data	Pass

				structure's corresponding method	
TC_12	To check methods mapping	Check mappings for getFirst()	Call getFirst()	The method is properly mapped to different data structure's corresponding method	Pass
TC_13	To check methods mapping	Check mappings for getLast()	Call getLast()	The method is properly mapped to different data structure's corresponding method	Pass
TC_14	To check methods mapping	Check mappings for indexOf(Object o)	Call indexOf(Object o)	The method is properly mapped to different data	Pass

				structure's corresponding method	
TC_15	To check methods mapping	Check mappings for lastIndexOf(Object o)	Call lastIndexOf (Object o)	The method is properly mapped to different data structure's corresponding method	Pass
TC_16	To check methods mapping	Check mappings for clear()	Call clear()	The method is properly mapped to different data structure's corresponding method	Pass
TC_17	To check methods mapping	Check mappings for isEmpty()	Call isEmpty()	The method is properly mapped to different data	Pass

				structure's corresponding method	
TC_18	To check methods mapping	Check mappings for remove(int index)	Call remove(int index)	The method is properly mapped to different data structure's corresponding method	Pass
TC_19	To check methods mapping	Check mappings for remove(Object o)	Call remove(Object o)	The method is properly mapped to different data structure's corresponding method	Pass
TC_20	To check methods mapping	Check mappings for removeFirst()	Call removeFirst()	The method is properly mapped to different data	Pass

				structure's corresponding method	
TC_21	To check methods mapping	Check mappings for removeLast()	Call removeLast()	The method is properly mapped to different data structure's corresponding method	Pass
TC_22	To check methods mapping	Check mappings for removeAll(Collection<? extends E> c)	Call removeAll(Collection<? extends E> c)	The method is properly mapped to different data structure's corresponding method	Pass
TC_23	To check methods mapping	Check mappings for replace(int index,Object o)	Call replace(int index,Object o)	The method is properly mapped to different data	Pass

				structure's corresponding method	
TC_24	To check methods mapping	Check mappings for sort(Comparator<? super E> c)	Call sort(Comparator<? super E> c)	The method is properly mapped to different data structure's corresponding method	Pass
TC_25	To check methods mapping	Check mappings for subList(int fromIndex, int toIndex)	Call subList(int fromIndex, int toIndex)	The method is properly mapped to different data structure's corresponding method	Pass
TC_26	To check methods mapping	Check mappings for size()	Call size()	The method is properly mapped to different data	Pass

				structure's corresponding method	
TC_27	To check methods mapping	Check mappings for put(K key, V value)	Call put(K key, V value)	The method is properly mapped to different data structure's corresponding method	Pass
TC_28	To check methods mapping	Check mappings for get(Object key)	Call get(Object key)	The method is properly mapped to different data structure's corresponding method	Pass
TC_29	To check methods mapping	Check mappings for containsKey(Object key)	Call containsKey(Object key)	The method is properly mapped to different data	Pass

				structure's corresponding method	
TC_30	To check methods mapping	Check mappings for containsValue(Object value)	Call containsValue(Object value)	The method is properly mapped to different data structure's corresponding method	Pass
TC_31	To check methods mapping	Check mappings for putAll(Map<? extends K,? extends V> m)	Call putAll(Map <? extends K,? extends V> m)	The method is properly mapped to different data structure's corresponding method	Pass
TC_32	Automatic removal of incompatible DS	To check whether a ds is which doesnot support a particular	Call the method	The incompatible ds is removed from DS list	Pass

		function is removed from the ds list			
TC_33	Printing of time taken by each ds	To check if the time taken by compatible ds are displayed	Run the test program	Time taken by each compatible DS are printed on the console	Pass
TC_34	Finding Optimal DS	To check whether the Optimal DS is selected or not	Run the test program	DS with minimum runtime is selected as Optimal DS	Pass
TC_35	Finding existing object names	To find existing object names in user program to replace it	Run the program	Names of user program objects are found	Pass
TC_36	Modifying Object Reference Names	To check whether object Refernce names are modified to	Run the user program	object Refernce names are modified to optimal DS Reference	Pass

		optimal DS Reference name or not		names	
TC_37	Modifying Object names	To check whether object names are modified to optimal DS object names or not	Run the user program	Object names are modified to optimal DS object names	Pass
TC_38	Importing of unavailab le packages	To check whether unavailable packages are imported	Run the user program	Unavailable packages are imported	Pass

Table 9.1 Unit Test Results

9.2 INTEGRATION TESTING

Test Case No	Module	Test Case Description	Procedure	Expected Output	Result
1	Method mapping module integrated with run time calculation module	To check the integration of these modules work or not	Run any method of Bag/BagPair	Method mapping works along with time calculation	Pass
2	Code modifier module integration with package import module	To check the integration of these modules work or not	Run the user program	User code should be modified & necessary packages should be imprted.	Pass
3	Optimal DS selector module integration with code modification module	To check whether the integration of these modules work or not	Run the user program	Optimal DS is determined first & then user code will be modified.	Pass

Table 9.3 Integration Test Results

9.3 SYSTEM TESTING

Test Case No	Module	Test Case Description	Procedure	Expected Output	Result
1	Bag Class System	To check the system works for Bag Class	Run Use program with Bag Class	Optimal Bag DS is selected and code is modified with that DS	Pass
2	BagPair Class System	To check the system works for BagPair Class	Run Use program with BagPair Class	Optimal Bag DS is selected and code is modified with that DS	Pass

Table 9.3 System Test Results

10. RESULTS AND DISCUSSIONS

The project helps in determining the most optimal data structure for a user program. It is easy to use and does not require any extra effort from user. The results are depicted with the examples below:

Example 1:

User program

```
import java.io.IOException;
import java.util.Stack;
public class Test2 {
    public static void main(String[] args) throws IOException
    {
        Bag<Integer> bag1 = new Bag<Integer>();
        for(int i=0;i<100;i++)
        {
            bag1.insertAtLast(new Integer(i));
        }
        int ch = 0;
        for(int i = 0; i < 10000 ; ++i)
        {
            ch = (int)(Math.random() * 1000);
            switch(ch%3)
            {
                case 0 : int index = (int)(Math.random() * bag1.size());
                        if(!bag1.isEmpty())
                            bag1.get(index);
                        break;
                case 1 : if(!bag1.isEmpty())bag1.removeLast();
```

```
                break;
            case 2 : Stack<Integer> st = new Stack<Integer>();
                for(int x = 0; x < 100 ; x++)
                {
                   st.push((int)(Math.random()*1000));
                }
                bag1.insertAll(st);
                break;

         }
      }
      Bag.optimize();
   }

}
```

It is seen that the user program is not easy to comprehend and it is not obvious which data structure is to be used. This project helps in determining the data structure for such programs.

The results obtained suggest that stack the optimal one.

It is explained in the graph below

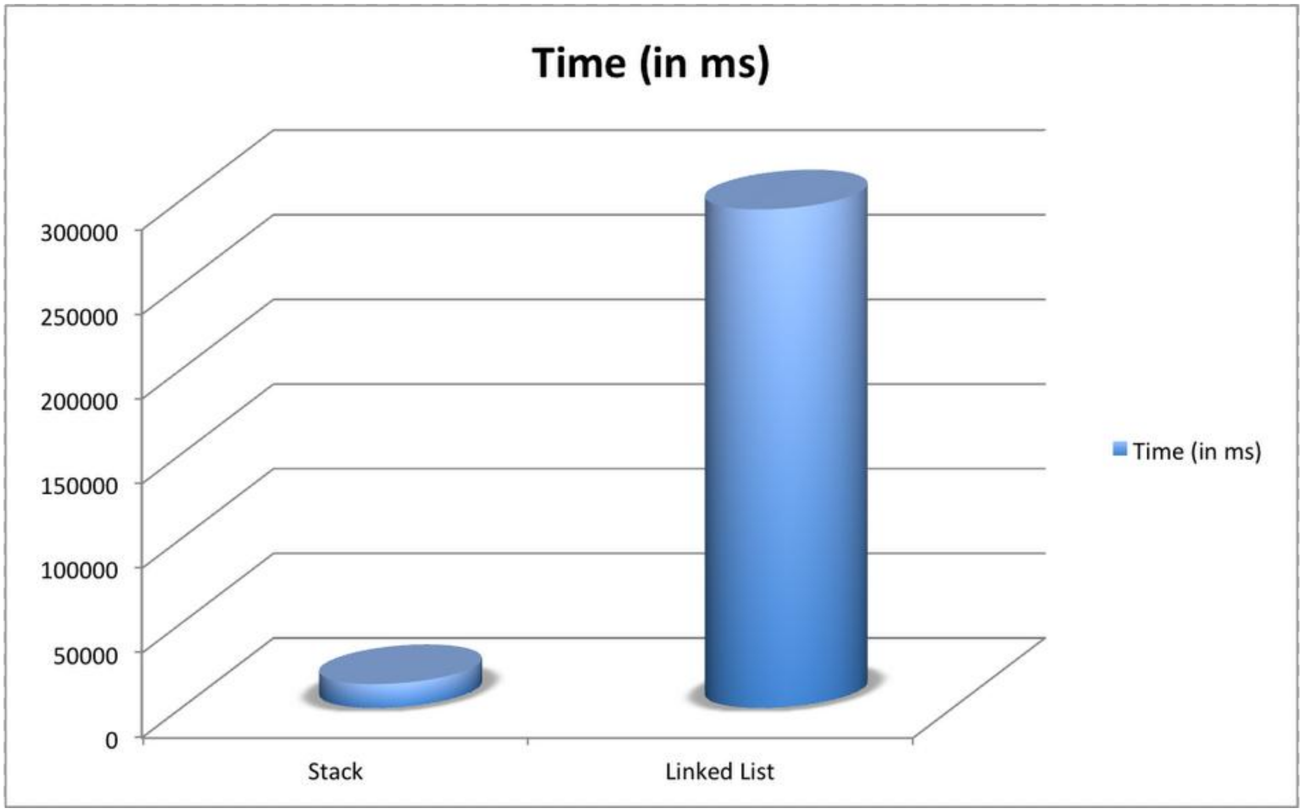


Fig 10.1 Comparison of execution time taken by a stack and linked list by the user code

Example 2:

User program..

```
import java.io.IOException;
public class Test1
{
    public static void main(String[] args) throws IOException
    {
        Bag<Integer> bag1 = new Bag<Integer>();
        int num=0;
```

```
        for(int i=0;i<1000;++i)
        {
            num = (int)(Math.random() * 10); // Generates an integer in the
range 0-9
            if(!bag1.contains(num))  {
                bag1.insertAtFirst(new Integer(num));
            }
        }
        Bag.optimize();
    }
}
```

The following graph explains which is the optimal one for the above user program-

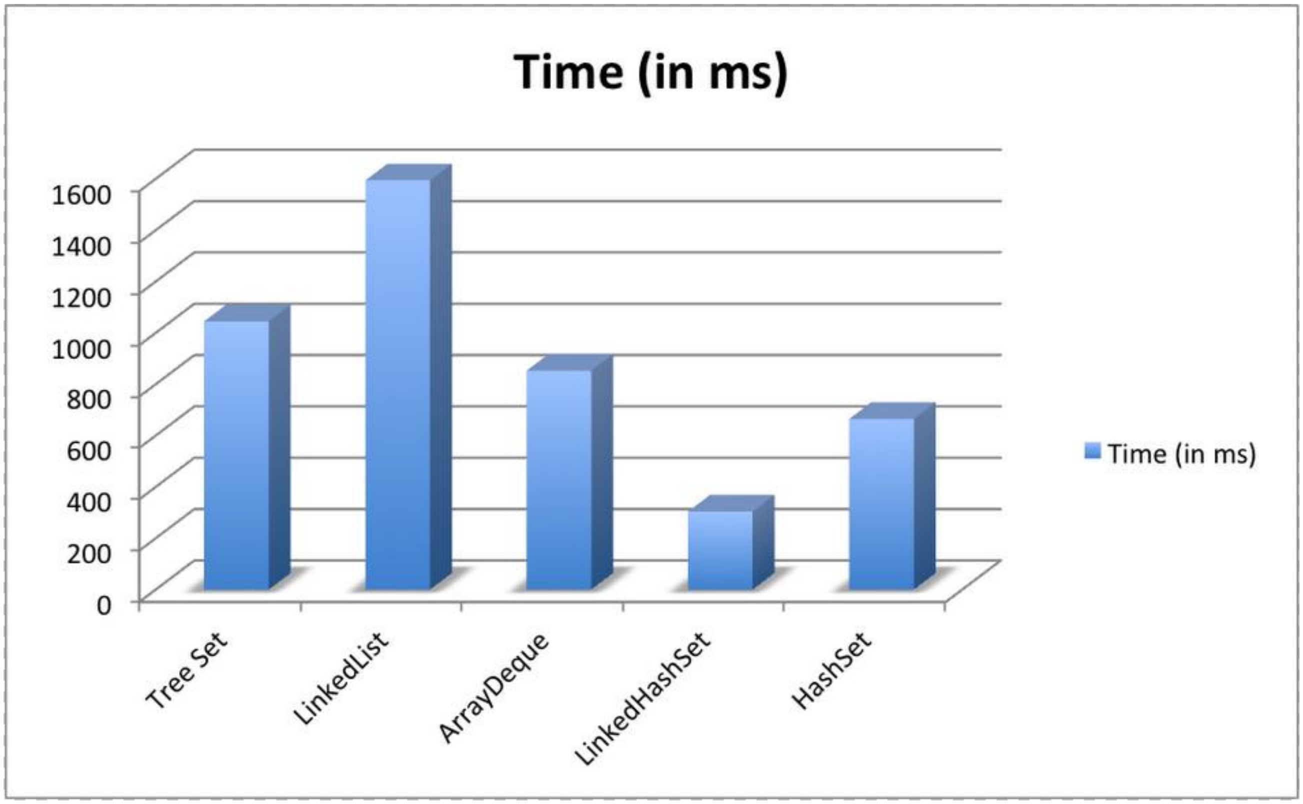
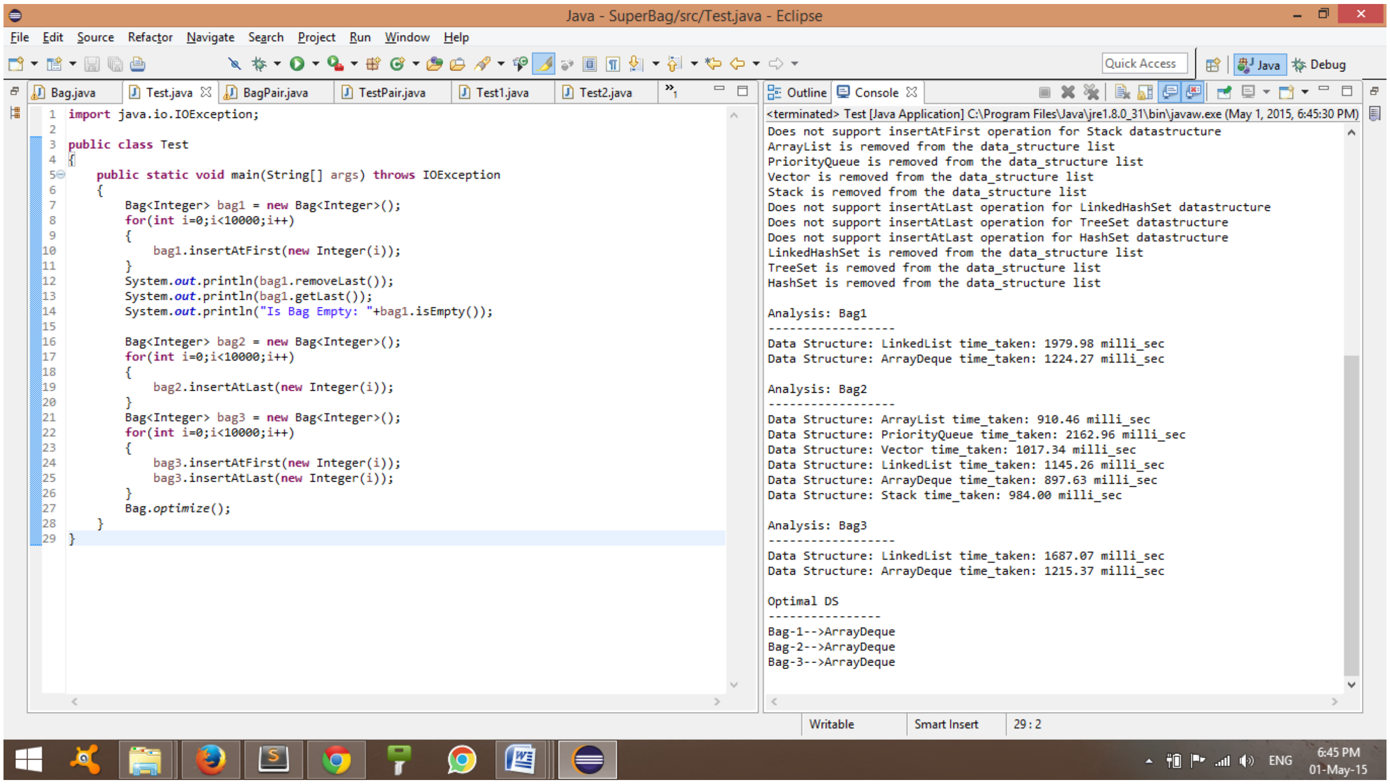
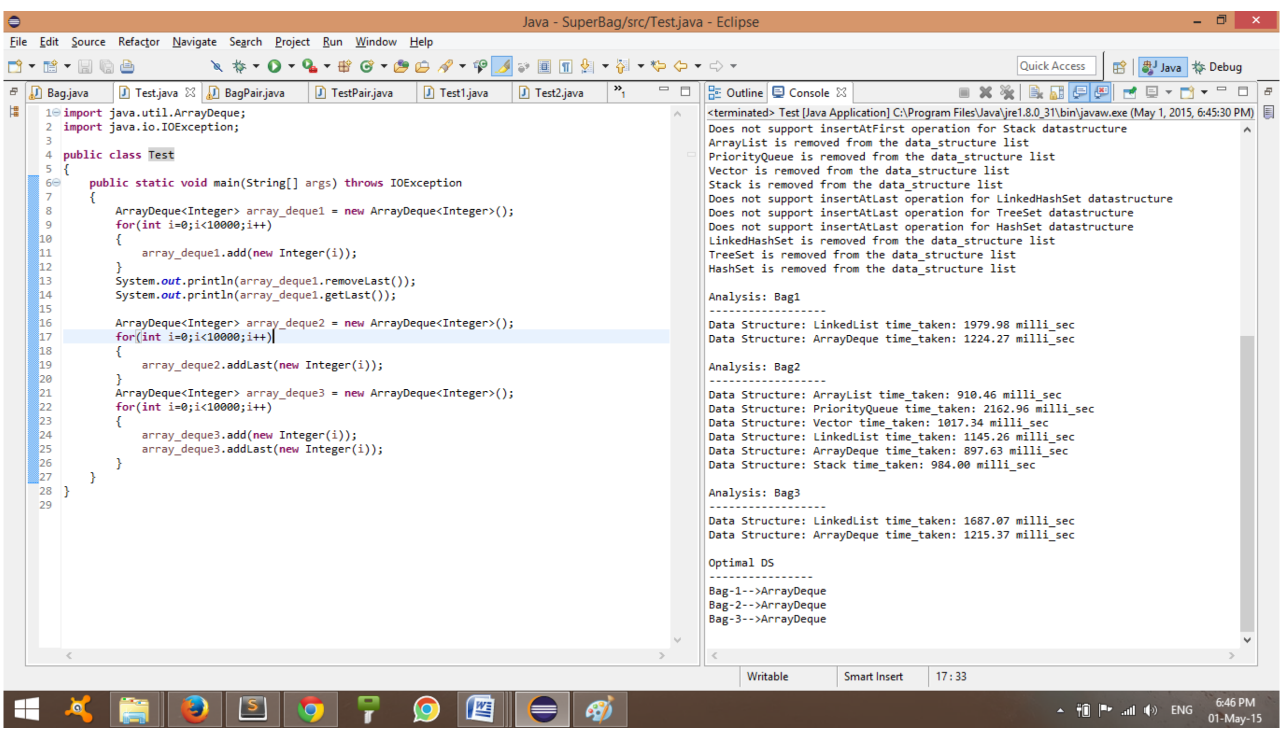


Fig 10.2 Comparison of time taken by various data structures for example 2

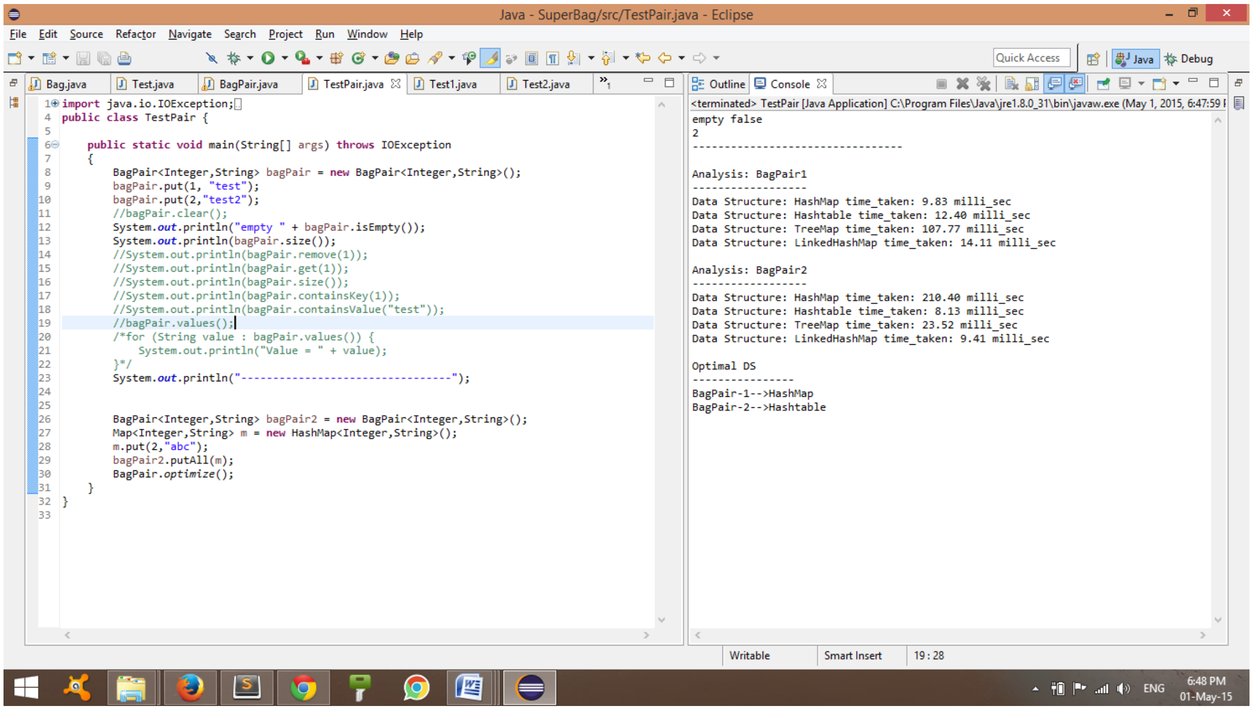
Hence, linkedhashset the optimal datastructure for such cases.

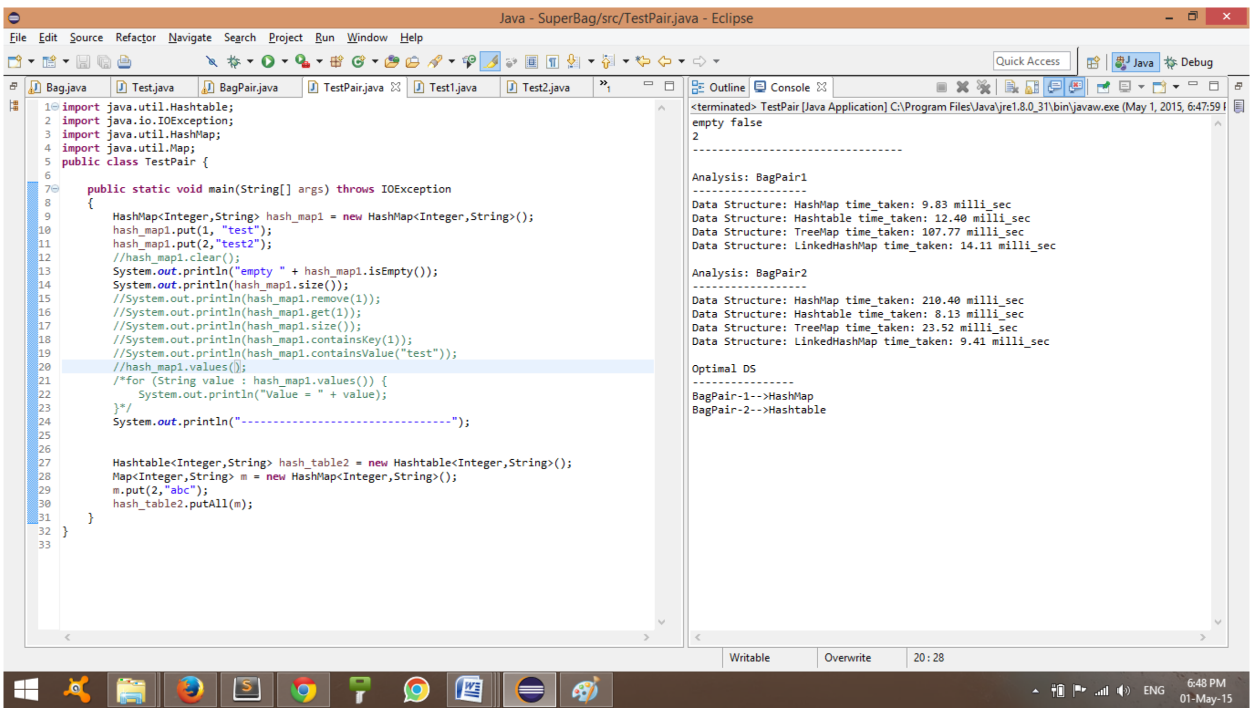
11. SNAPSHOTS





Automatic Detection of Optimal Data Structure in Eclipse





12. CONCLUSION

Thus user can just import our Bag Package, Create Bag objects wherever data structures are needed. User need not have to worry about what data structure to use. The Bag object which the user creates will be automatically converted into optimal data structure objects. Also, the method names are changed according to match it with optimal data structure methods. Hence this eases the user in using data structures effeciently. Also, the beginners can run their program & see which data structure is optimal in different cases. Finally, this can be a really useful tool for a programmer.

13. Further Enhancement

1. To make the plugin to understand Java using refactor API & not just String replacement in code modification.

2. To extend this functionality for different programming languages.

3. To support some more data structures which are not part of the standard collection framework.

14. REFERENCES

14.1 Research Paper References

1. B.Boothe, University of Southern Maine, Portland, ME, USA, 'Using Real Execution Timings to Enliven a Data Structures Course', 2012.
2. Changhee Jung, Georgia Institute of Technology, Atlanta; Silvius Russ, Google Inc, Mountain View, ‘Brainy: effective selection of data structures, 2011
3. Guoqing Xu, University of California, USA, ‘CoCo: sound and adaptive replacement of java collections’, 2013.
4. Shengqian Yang, Ohio State University, USA; Dacong Yan, Ohio State University, USA, ‘Dynamic analysis of inefficiently-used containers’, 2012.
5. Mattias De Wael, Vrije Universisteit Brussel; Wolfgang De Meuter, Vrije Universisteit Brussel, ‘Data interface + algorithms = efficient programs: separating logic from representation to improve performance’, 2014.
6. Minhaz F. Zibran, University of Saskatchewan; Chanchal Kumar Roy, ‘Towards flexible code clone detection, management, and refactoring in IDE’, 2011.
7. Yoshio Kataoka, University of Washington; D Notkin, University of Washington, ‘Automated Support for Program Refactoring using Invariants’, 2001.
8. Soares, G. ; Gheyi, R. ; Serey, D. ; Massoni, T., ‘Making Program Refactoring Safer’, 2006.
9. Zhenchang Xing ; Dept. of Comput. Sci., Alberta Univ., Edmonton, Alta. ; Stroulia, E., ‘Refactoring Practice: How it is and How it Should be Supported - An Eclipse Case Study’, 2004.

14.2 Website References

1. Java™ Platform, Standard Edition 7 API Specification, http://docs.oracle.com/javase/7/docs/api/

2. Oracle Sun Developer Network, “JAVA SE 6 Performance White Paper”,http://java.sun.com/performance/reference/whitepapers/6_performance.html

3. Neil Coffey, 'Memory usage in Java', http://www.javamex.com/tutorials/memory.

4. Lesson: Introduction to Collections, https://docs.oracle.com/javase/tutorial/collections/intro/

5. The Collection Interface, https://docs.oracle.com/javase/tutorial/collections/interfaces/collection.html

6. Collections Framework Overview, http://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html.