Architecture for Encapsulation of "Collection Framework Classes" in JAVA for Reducing the Complexity

Dr. Mohd Ashraf
Department of Computer Science & Engineering
Maulana Azad National Urdu University
Hyderabad, Telengana
Email Id: ashraf.saifee@gmail.com

Abstract—A Collection Framework classes introduced in the Java core library after the release of JDK version-1.2. Collection framework classes were built up by Joshua Bloch. Although, Collection framework Classes are widely practiced in today's environment to develop Java application. Java application programs suffer from a major problem of complexity and un-scalability. In this paper, the author proposed a design of the framework which encapsulates Java collection framework classes of complex hierarchy into a single class that is based on some certain parameters (e.g. data structure used, uniqueness, synchronization). The proposed framework can greatly lessen the efforts required to learn and use of complex hierarchy of Collection Framework. Further, the proposed framework (Architecture) uses the identical method signature as the original Collection Framework, which makes it easy to understand for those who earlier use Collection Framework for data storage. Thus, the users can easily shift to the new framework.

Keywords—*Object, Class, Encapsulation, Java Collection Framework, Java Library Class, Data Structure, Decorator pattern, Predator System.*

*****

I. Introduction

Reusability has always been a major research area in software engineering. A big bit of reuse based API has been grown in Java in order to ease the work of software engineers in developing software. Using this API, software engineers can focus on meeting customer demands rather than coding the elementary services required by the software. Most often developers are needed to store data in a number of data structures [16] such as arrays, linked lists, queues, stacks and hash table on the basis of the unique characteristics of the stored data [8]. Initially, implementation of Java contained only a few Classes catering to data store needs such as HashTable, Vector etc. These were difficult to operate and did not cater to whole data storage demands. A major breakthrough in this country was established by Doug Lea's Collection Package and ObjectSpace Generic Collection, Library [5, 8, 9, and 11]. This was uniform with the C++ Standard Tag Library. Using the ideas of Doug Lea's Collection Package Joshua Bloch designed "Java Collection Framework" which was included in the core library of Java i.e. Rt. jar file from JDK 1.2 release [5, 8, 9, 11]. Java Collection Framework still enjoys a singular position in today's technological environment. After many changes were made to the Collection framework such as including "generic concept" which introduced compile time type checking [15, 13].

Although the Collection framework is a full-fledged package catering to whole data storage demands, its complex hierarchy has led many researchers to knock it as un-scalable and inflexible library which is difficult to determine and practice [2]. It contains a number of classes and interfaces which are often difficult to return. To overcome these limitations, we have prepared an approach for preparing a property based framework that can internally use an appropriate data structure based on the attributes set by the user.

In section II, Author reported researches in the field of making a property based Collection framework are described. Furthermore, in section III, the existing approaches have been described briefly. In section IV, the proposed architecture is identified with the help of algorithm, flow chart.

II. Literature Survey

In the yesteryear, much work in this direction has been acted to prove a property based collection framework. Below are only some of them:

A. *Predator System*

Predator based system [1] emphasizes the dynamic creation of data structures based on the features specified by the user (e.g. If the user specifies that he/she wants the elements should be inserted in random order, then internally a Hash Table should be used as the data structure). For designing such a careful system design and implementation of each element is needed. The internal details of implementation of data structures based on features should be abstracted using interfaces. Interfaces should have three basic properties: High level abstraction, Standardized Interface and Layered design.

B. *Good Framework /Library-a brief outline*

Extensive research has been reported along the characteristics of good collection framework [3]. The extensive work/idea required to prepare a good framework is a vital matter. Designing proper abstractions for every functionality is a decisive job. Java Collection Framework suffers from a major issue of its complexity which has rendered it inflexible, unwieldy and difficult to understand and use. In [3] brief study on features of every component of Collection Framework has been produced (e.g. The hash table class supports random order insertion of elements. Here "random order" is a property of HashTable class). It resolves with a need for putting through a framework based on characteristics.

B. *Decorator-Patter Based Framework*

The decorator design pattern allows new features to be added by modifying the existing methods, rather than adding new methods. This can be beneficial to the programmers as it will not be hard for them to shift to this newfangled technology. In [6] a framework has been proposed combining the decorator pattern with proxy design and Template method adding prefix and suffix operation to previous implementations. Prefix proxy operations alters the pre-conditions and postfix proxy operations alters the post-conditions (e.g. the pre-conditions for add() is to check the type of collection in which data is to be stored and post-condition is to check whether the specified collection contains the required data). The main idea behind this theoretical explanation is that "anytime a feature can be added to a collection data structure and then could be dispatched".

In the next section Collection Framework, which is widely used today in industries is explained in detail followed by its limitations.

III. Existing Approach

In the present scenario the "Collection Framework" included in the core library of Java (Rt. jar) enjoys an excellent situation in the marketplace. It consists of a number of classes and interfaces [5,8,9,11] developed to fit each and every demand of data memory. The Hierarchical representation of the diverse classes and interfaces is shown in Fig.1. Each year in the framework has some properties that outline the course of study from other divisions.

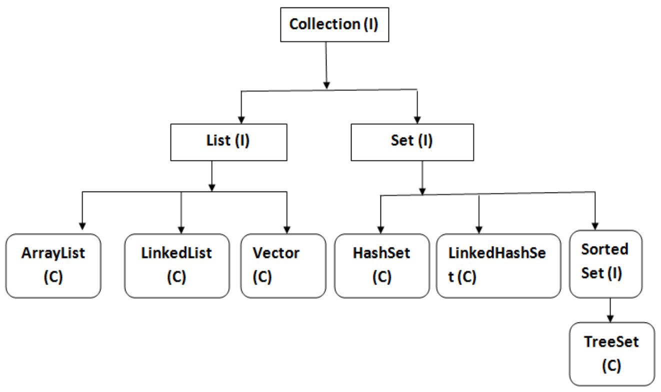


Fig.1. Collection Framework Hierarchy. Here (I) represents Interface and (C) represents Class

Collection Framework [5, 8, 9, 11] has a number of elements in the form of classes and interfaces. At the pinnacle of the hierarchy is the Collection interface which outlines the basic operations needed for every storage class (say adding of elements to a collection). At the next level of the hierarchy is the *List* and *Set* interfaces.

List interface is used to store elements in *the insertion order* and support duplicity. It has 3 subclasses namely *ArrayList* is supporting insertion order Asynchronized storage, Vector supports insertion order synchronized storage, LinkedList is supporting insertion order synchronized storage. Both ArrayList and Vector internally uses an array data structure while LinkedList uses linked list as data structure.

Set interface is used to store elements *uniquely* in *ordered* or *unordered* manner. It has 3 subclasses namely *HashSet* for *random* order insertion *Asynchronized* storage, *LinkedHashSet* for *insertion* order *Asynchronized* storage, TreeSet for storing elements in sorted order *asynchronously*.

Generics, Autoboxing/unboxing and *Enhanced for loop* [5] features were added to Java Collection Framework in JDK-5 and subsequent versions.

IV. Pitfalls of Existing Approach

In Collection Framework has a number of classes and interfaces which are frequently hard to recall. The criticism of Collection framework is as follows:

- Domain-specific libraries like Collection Framework serve efficiently for building software. Each Component of these libraries implements a unique set of features (HashSet class of the Collection framework serves for storing elements in random order say). The Combination of these elements forms a vast library which is un-scalable [2]. An extended study of these libraries is required in order to employ even a single component of the library.
- A good framework is that which make the task of the programmer easier. Java Collection Framework despite of its endless advantages suffers a major drawback due to its complex power structure. It is clunky, inflexible and difficult to learn and practice [1].

To overcome these restrictions, we have planned the architecture for developing a property based framework that can internally use an appropriate data structure on the attributes set by the user.

V. Proposed Approach

The principal idea behind the framework [18] is "users will define properties and internally the framework will determine on the Collection Library class to be applied"

A. *Framework Structure*

Counting at the major drawback of Collection Framework, i.e. complexity and difficulty in learning and also not forgetting about its capability to do all data storage demands, the framework will use the Collection Library only but will not employ it immediately. It will require the user to set certain properties that are:

- Uniqueness: whether data to be stored in collection in unique way or whether duplicity is allowed.
- Data Structure usage: whether to store elements in arrays, linked lists, binary tree, hash table or doubly linked list.
- Ordering of storage: ordering can be random order, insertion order, or sorted order
- Synchronization: whether the Collection library to use is synchronized or unsynchronized.

Based on these properties [11] the framework will internally decide on the Collection, Library class to be used for storage.

Our framework also supports the generic nature of collection library which was introduced to Collection Framework from

JDK version 5 onwards.

A classification has been made in Table 1 according to the properties of each of the Components of Collection Library.

TABLE I
UNITS FOR MAGNETIC PROPERTIES

Class/ Interface	**Data structure usage**	**Unique/ duplicate**	**Order of traversal**	**Synchronized/Asynchronized**
List (I)	Array or Linked List	Duplicity allowed	Insertion Order	Asynchronized
Set (I)	Hash Table/Binary Tree/Doubly Linked List	Unique elements	Random/ sorted/ insertion ordered	Asynchronized
Sorted Set (I)	Binary Tree	Unique elements	Sorted order	Asynchronized
Array List (C)	Array	Duplicity allowed	Insertion order	Asynchronized
Vector (C)	Array	Duplicity allowed	Insertion order	Synchronized
LinkedList (C)	Linked List	Duplicity allowed	Insertion order	Asynchronized
Hash Set (C)	Hash Table	Unique element	Random order	Asynchronized
LinkedHash Set (C)	Doubly Linked List	Unique element	Insertion order	Asynchronized
TreeSet (C)	Binary Tree	Unique element	Sorted order	Asynchronized

The table classifies all the major components of Collection Framework based on the properties possessed by them. Here (I) represents interface and (C) represents class that implements the interfaces.

B. *Algorithm*

Our framework design follows the following algorithm internally to use properties as a base for selecting an appropriate Collection Library class.

Step1:[Specification of Properties] Properties are specified in an XML file [17] namely "Collection.xml". This is done at user side. The xml file starts with the root tag as <Colmer>. The nested tag <Collection> describes a single collection. The nested tags of <Collection> and their allowed values are as follows:

```
<Collection-Name>Collection_name (any user defined name)</Collection-Name>
<Data-Structure>linked list/array/binary tree/doubly linked list/hash table</Data-Structure>
<Unique>duplicate/unique</Unique>
<Order>insertion order/random order/sorted order</Order>
<Synchronize>asynchronize/synchronize</Synchronize>
```

It is to be noted that the user can select any combination of properties, but only few maps to a valid Collection Framework class. So one has to be careful while choosing a combination of properties. The framework will take into account all the properties mentioned by user and then decide a Collection class to be used internally. If a combination do not map to any of the Collection classes, then an error will be thrown.

Step2: [Storing Properties in a properties file] In the main program user need to create the object of XMLInitiator class and call its init() method which will read [19] the *Collection.xml* file [14] and store the properties in the ../produced/*Collection_name.Properties* file. In addition to storage of properties the Framework internally uses these properties and decide on the Collection, Library to be used and store it in the same file. The keys in the properties file and their associated allowed values are as follows:

Data-Structure= linked list/array/binary tree/doubly linked list/hash table
Order= insertion order/random order/sorted order
Collection-name= Collection_name (any user defined name)
Synchronize=asynchronize/synchronize
Unique=duplicate/unique
Collection-used= LinkedList/ArrayList/HashSet/Vector/TreeSet/LinkedHashSet

Step3: [Making the object of Colmer class] in the main program user need to attain the object of Colmer class. Colmer<T> class act as an intermediary between the class that perform actual storage of factors in the particular Collection Library class and the main course of study. Colmer<T> class reads the ../produced/Collection_name.Properties file. Utilizing the properties file it says the value corresponding to the Collection-used key and directs the storage to a grade that does actual storage. E.g. if the value corresponding to *the Collection-used* is *LinkedList* then Colmer<T> class will create the object of ColmerLinkedList<T> and directs the actual storage to this class. The constructors of Colmer class are as follows:

```
public Colmer(String col_name)
public Colmer(String col_name,int minCapacity)
public Colmer(String col_copy, Colmer<T> c)
```

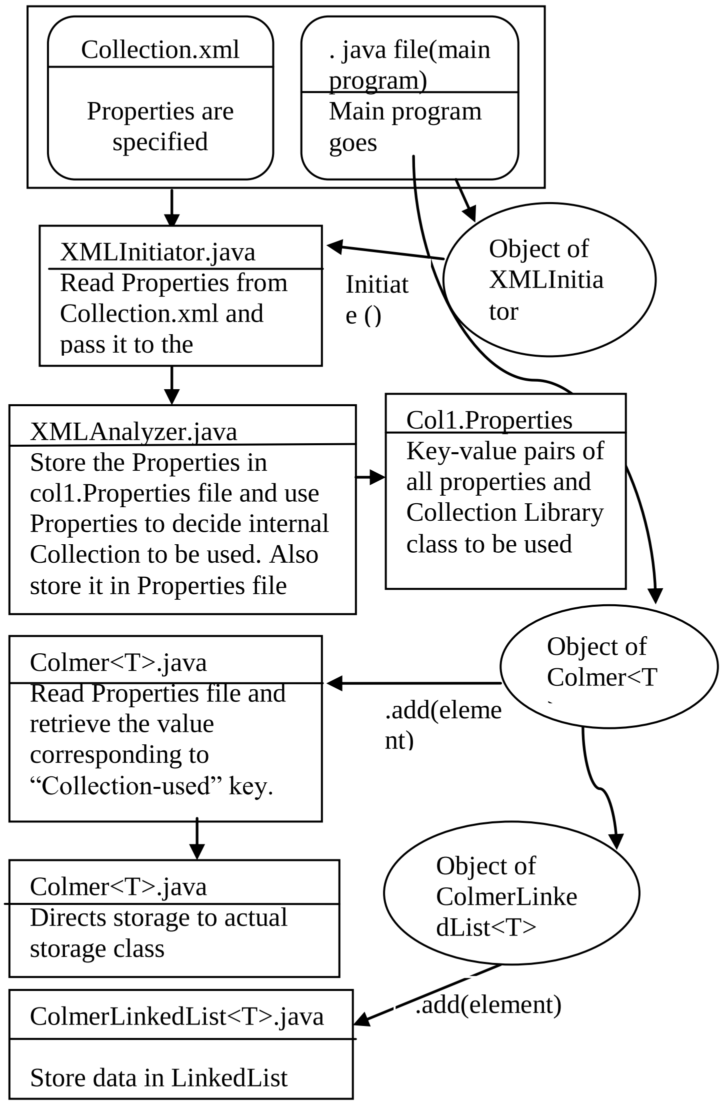
Here *col_name* is the name of the Collection specified by the user in the Collection.xml file. If there is no Collection-name same as the name passed by user then an error is displayed "no entry found in xml for collection specified". minCap is the minimum capacity of the data structure required.

Step4: [Making a call to any of the method] using the Colmer<T> class object the user can make the call to any of the methods that aid in storage, deletion or retrieval of elements. The methods accept the same syntax as in the original Collection Framework, which facilitate the task of programmer as he do not have to read entirely new set of methods. For e.g.

to add an element to the collection syntax is *add(T element)* where T is the type variable. This is same as the *Classical Collection Framework*.

C. *Flow Diagram Representation*

Below is the flow chart representation of the proposed framework.

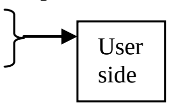

Collection.xml
Properties are specified
. java file(main program)
Main program goes
XMLInitiator.java
Read Properties from Collection.xml and pass it to the
Initiate ()
Object of XMLInitiator
XMLAnalyzer.java
Store the Properties in col1.Properties file and use Properties to decide internal Collection to be used. Also store it in Properties file
Col1.Properties
Key-value pairs of all properties and Collection Library class to be used
Object of Colmer<T>
Colmer<T>.java
Read Properties file and retrieve the value corresponding to "Collection-used" key.
.add(element)
Colmer<T>.java
Directs storage to actual storage class
Object of ColmerLinkedList<T>
ColmerLinkedList<T>.java
Store data in LinkedList
.add(element)

It is to be noted that col1.properties are being exactly an example of Properties file. Col1 is the user defined name specified in the Collection.xml file.

D. *Component Specification*

There are a number of components in the proposed framework which require an in depth explanation. Our framework consists of 6 major components namely:

- Collection.xml file
- .java file
- XMLInitiator.java
- XMLAnalyzer.java



- Colmer.java
- Actual Storage classes

I. *Collection.xml file*

Attributes are defined in an XML file [17] namely "Collection.xml". This is performed at the user side. The xml file starts with the root tag as <Colmer>. The nested tag <Collection> specify describes a single collection. An e.g. Collection.xml file is as follows

```
<Colmer>
<Collection>
<Collection-Name>col1</Collection-Name>
<Data-Structure>linked list</Data-Structure>
<Unique>duplicate</Unique>
<Order>insertion order</Order>
<Synchronize>asynchronize</Synchronize>
</Collection>
```

II. *.java files*

The main program of user is specified in these files. There may be only a single Java file or many Java files which are at user discretion. The .java file needs to [5,8,9,11] import *Java. util. **, *calmer. xmlreader. XMLInititor* and *calmer. Mains. colmer* libraries. An e.g. .java file is as follows:

```
public class A{

public static void main(String args[]){
XMLReader xr=new XMLReader();
xr.initiate();
Colmer<Integer> c=new Colmer<Integer>("col1");
c.add(1);
```

III. *XMLInitiator.java*

The function of XMLInitiator.java is to read the Collection.xml [19] file [14]. After recording the properties are passed to XMLAnalyzer.java:

```
public class XMLInitiator {
public void initiate(){
reading xml file and passing properties to XMLAnalyzer
analyze(list of properties as parameters)
}
}
```

IV. *XMLAnalyzer.java*

The function of XMLAnalyzer.java is to study the attributes and decide the Collection, Library class to be habituated. The properties along with the collection library used are stored in the col1.properties file [12] (col1 is user defined name in Collection.xml file). After reading the properties are passed to XMLAnalyzer.java:

```
public class XMLAnalyzer {
public void analyze(list of properties as parameters){
//to do task here
}
}
```

V. *Colmer.java*

Colmer<T> is a generic [13] class. Here T is the type parameter. It acts as the intermediary between the main user program and the actual storage classes. It reads the properties file and directs the storage to required storage class. It uses three constructors as:

public Colmer(String col_name) constructor creates a collection whose properties are specified in the col_name.properties class.

public Colmer(String col_name, int minCapacity) constructor creates a collection with minimum capacity as minCapacity and whose properties are stored in col_name,properties file

public Colmer(String col_copy, Colmer<T> c) constructor copies the Collection c to the new Collection whose properties are stored in col_copy.properties file. Here the type mismatch between the target collection and source collection will lead to type mismatch error.

```
public Colmer(String col_name){//(1)
//to do task here
}
public Colmer(String col_name, int minCapacity){//(2)
//to do task here
}
public Colmer(String col_copy, Colmer<T> c){//(3)
//to do task here
}
 // methods same as original Collection framework
e.g. public boolean add(T element){}
```

VI. *Actual storage classes*

Actual storage classes are contained in colmer.storageclasses package. Their name, constructor and methods will vary according to the internal Collection Library. All the methods are same as classic collection Library classes. These are also generic [13] classes. For e.g. if collection to be used is linked list, then internally ColmerLinkedList<T> class will be used.

```
public class ColmerLinkedList<T> {

LinkedList<T> l;
public ColmerLinkedList(){ // constructor
      l=new LinkedList<T>();// creation of Object of classic
                                          linked list class
}
// methods same as classic LinkedList class
e.g.
public Boolean add(T element){
// to do task here
}
```

VI. Conclusions

A framework which utilizes the actual Collection library, since it is a full-fledged library to cater all needs of storage and also eliminating its limitation of being complex can be developed using the above devised algorithm and flow diagram.

The proposed framework has not included Map based collections in its scope which paves way for further research in this area.

Acknowledgment

I would like to take this opportunity to express my profound gratitude and deep regard to Dr. Mohd Ashraf Saifi, for his exemplary guidance, valuable feedback and constant encouragement throughout the duration of my research. His valuable suggestions were of immense help throughout my work. His perspective criticism kept me working to make this research in a much better way. Working under him was an extremely knowledgeable experience for me.

References

[1] Lars Heinemann, “Effective and Efficient Reuse with Software Libraries”, Technische Universitat Munchen, July 2012

[2] Jeff Thomas, Don Batory and Vivek Singhal, “Scalable Approach to Software Library”, 6th annual workshop on software reuse, November 1993

[3] Virginia Niculescu, Dana Lupasa and Radu Lupasa, “Issues in Collection Framework Design”, Studia Univ. Babes, Informatica, Volume LVII, November 4 2012

[4] J. Bloch, The Java Tutorial. Trail: http://docs.oracle.com/javase/tutorial/collection.

[5] Sun Microsystems. Collections Framework Overview. [Online]. Viewed 2010 July 23. Available: http://java.sun.com/javase/6/docs/technotes/guides/collections/overview.html.

[6] Virginia Niculescu and Dana Lupasa, “A Decorator based Design for Collections, Volume LVIII, November 3, 2013.

[7] Joshua Bloch – Effective Java from the source; Second Edition; Sun Microsystems, May 2008.

[8] Guoqing Xu, “Finding Reusable Data Structures”, ACM Digital Library, 2012.

[9] Doug Lee, “Overview of Collection Package”, 1999.

[10] Herbert Schildt– Java: The Complete Reference, Seventh Edition, McGraw Hill Professional, 01-Dec-2006.

[11] Java Collection Framework-Wikipedia: Available: http://en.wikipedia.org/wiki/Java_collections_framework.

[12] Sun Microsystems. Java. util. Properties. [Online]. Viewed 2014. Available: https://docs.oracle.com/javase/7/docs/api/java/util/package-summary.html.

[13] Sun Microsystems. Java Generics. [Online]. Viewed 2014. Available: https://docs.oracle.com/javase/tutorial/java/generics/.

[14] Sun Microsystems. Reading XML Data into DOM. [Online]. Viewed 2014. Available: https://docs.oracle.com/javase/tutorial/jaxp/dom/readingXML.html.

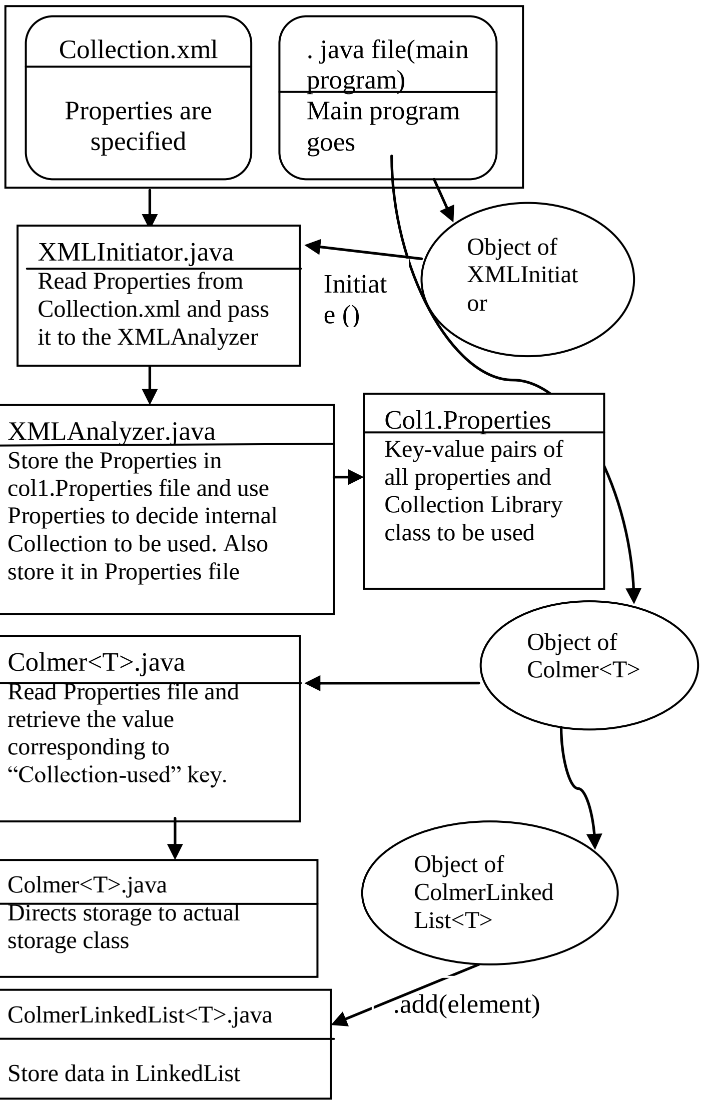
[15] Maurice Naftalin, Philip Wadler– Java Generics and Collection; Orielly, 2006.

[16] Nell Dale, Daniel Joyce, Chip Weems– Object Oriented Data Structures using Java; Third edition, 2011.

[17] Williamson- Xml: The Complete Reference; Tata McGraw-Hill Education, 2001.

[18] Jaroslav Tulach- Practical API Design: Confessions of a Java Framework Architect; Apress, 2008.

[19] Sun Microsystems. Reading, writing and creating files. [Online]. Viewed 2014.
Available:
https://docs.oracle.com/javase/tutorial/essential/io/file.html.
[20] E Balaguruswamy- Programming with Java: A Primer; Tata McGraw-Hill Education, 2009.


Collection.xml
Properties are specified
. java file(main program)
Main program goes
XMLInitiator.java
Read Properties from Collection.xml and pass it to the XMLAnalyzer
Initiate ()
Object of XMLInitiator
XMLAnalyzer.java
Store the Properties in col1.Properties file and use Properties to decide internal Collection to be used. Also store it in Properties file
Col1.Properties
Key-value pairs of all properties and Collection Library class to be used
Object of Colmer<T>
Colmer<T>.java
Read Properties file and retrieve the value corresponding to "Collection-used" key.
Colmer<T>.java
Directs storage to actual storage class
Object of ColmerLinkedList<T>
.add(element)
ColmerLinkedList<T>.java
Store data in LinkedList