ECE 522 – Software Construction, Verification and Evolution

Objectives:

- Getting familiar with Java collections framework
- Getting familiar with skip lists
- Compile and run the code using Apache ANT

1. Introduction

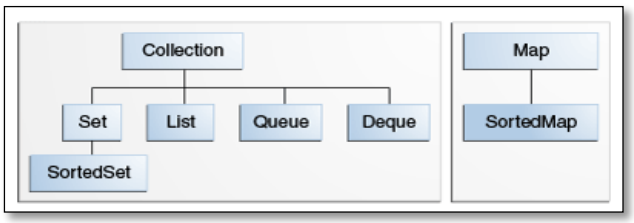
Welcome to the fifth ECE 522 lab. This lab contains 2 deliverables, which you should demo to the lab instructor before the end lab session.

1.1 Collections

A *collection* (sometimes called a container) is simply an object that groups multiple elements into a single unit. Collections are used to store, retrieve, manipulate, and communicate aggregate data. Typically, they represent data items that form a natural group, such as a poker hand (a collection of cards), a mail folder (a collection of letters), or a telephone directory (a mapping of names to phone numbers).
A *collections framework* is a unified architecture for representing and manipulating collections. All collections frameworks contain the following:

- **Interfaces:** These are abstract data types that represent collections. Interfaces allow collections to be manipulated independently of the details of their representation. In object-oriented languages, interfaces generally form a hierarchy.
- **Implementations:** These are the concrete implementations of the collection interfaces. In essence, they are reusable data structures.
- **Algorithms:** These are the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces. The algorithms are said to be *polymorphic*: that is, the same method can be used on many different implementations of the appropriate collection interface. In essence, algorithms are reusable functionality.

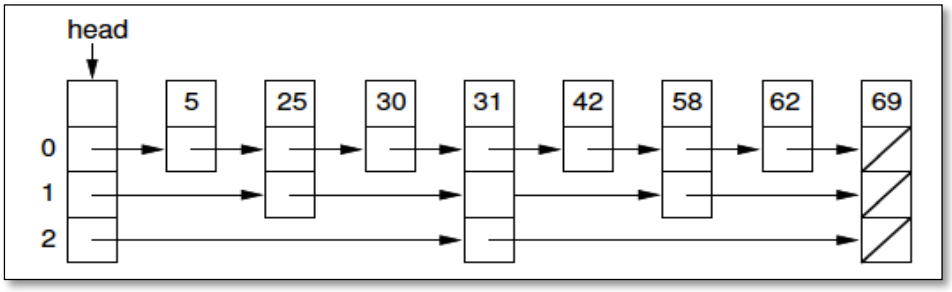
The *core collection interfaces* encapsulate different types of collections, which are shown in the figure below. These interfaces allow collections to be manipulated independently of the details of their representation. Core collection interfaces are the foundation of the Java Collections Framework. Note that all the core collection interfaces are ***generic***. Note that all the core collection interfaces are generic.



Core Collection Interfaces

1.2 Skip Lists

Skip lists are designed to overcome a basic limitation of array-based and linked lists: either search or update operations require linear time. The skip list is an example of a probabilistic data structure, because it makes some of its decisions at random. Traditionally balanced trees have been used to efficiently implement *Set* and *HashMap* style data structures. Balanced tree algorithms are characterized by the need to continually rebalance the tree as operations are performed. This is necessary in order to assure good performance.
Skip list provides an alternative to the balanced tree structure. The benefit of the skip list is that the algorithms for search, insertion and deletion are rather simple. This provides improved constant time overhead as compared to balanced trees. The algorithms involved are also relatively easy to understand.
In a singly linked list, each node contains one pointer to the next node in the list. A node in a skip list contains an array of pointers. The size of this array is chosen at random at the time when the node is created. The size of this array determines the level of the node. For example, a level 3 node has 4 forward pointers, indexed from 0 to 3. The pointer at index 0 (called the level 0 forward pointer) always points to the immediate next node in the list. The other pointers however point to nodes further down the list, and as such if followed allow portions of the list to be skipped over. A level i pointer points to the next node in the list that has level $\geq i$. Following figure shows an overview of a skip list.



A Skip List with header node named **head** (This list stores digits)

1.3 Apache ANT

A defined build process is one of the most necessary tools in the software development process ensuring that the software you produce is built to the required specifications. You should establish, document, and automate the exact series of steps required to correctly build your product.
A defined build process helps close the gap between the development, integration, test and production environments. A build process alone will speed the migration of software from one environment to another. It also removes many issues related to compilation, library paths, or properties that cost many projects considerable time and money.
From this, ANT (originally an acronym for Another Neat Tool), a Java-based build tool with special support for Java programming language, has emerged. It helps programmers to build complex applications and place the files in the desired location with less effort. ANT executes different tasks using Java classes and XML-based configuration files.
An ANT build file comes in the form of an XML document; all that is required is a simple text editor to edit an ANT build file. The ANT installation comes with a JAXP-compliant XML parser, this means that the installation of an external XML parser is not necessary.

The parser checks that the syntax of your ANT file is correct in a similar fashion to `javac` which checks that the syntax of your Java code is correct.

Project Homepage	http://ant.apache.org/
Project Documentation	http://ant.apache.org/manual/
ANT Online Tutorial	http://www.visualbuilder.com/java/ant/tutorial/

2. Deliverable 1 – Skip List

For this lab assignment, you need to explore the Java Collections Framework to implement a skip list class.

```
class SkipList<Key extends Comparable<? super Key>, Value> {
    public void insert(Key k, Value v) {
        // TODO: insert a new node
    }
    public void remove(Key k) {
        // TODO: remove a node according to the key
    }
    public Value find(Key k) {
        // TODO: find a node's value according to the key
        return null;
    }
    public void clear() {
        // TODO: clear the skip list
    }
    public int size() {
        // TODO: get the size of the skip list
        return -1;
    }
    public String toString() {
        // TODO: convert skip list to a string
        return null;
    }
}
```

DEMO this deliverable to the lab instructor.

3. Deliverable 2- ANT Build File

In this step, you need to compile and run your code, using ANT.

Step 1:

Copy and paste your source code (all java files) into `"src"` folder in your home directory.

Step 2:

Consider a sample ANT build file (build.xml). Put it in the same directory with your `"src"` folder.

Look at the ANT build file. If you focus on each of the `target` tags, you can see how in an ANT build file the targets are strung together by a series of depends attributes. We can see that compile is dependent on clean, jar is dependent on compile, and that run is dependent on jar. What does this mean? Well, if we tell ANT to make the target run, then it will automatically run its dependent tasks.

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="compile" name="lab5">
    <target name="compile"
            description="Compile the code into classes">
        <mkdir dir="build/classes"/>
        <javac destdir="build/classes" srcdir="src"/>
    </target>
    <target name="jar" depends="compile"
            description="Bundle classes into a single JAR file">
        <mkdir dir="build/jar"/>
        <jar basedir="build/classes" destfile="build/jar/skiplist.jar">
            <manifest>
                <attribute name="Main-Class" value="Skipmain"/>
            </manifest>
        </jar>
    </target>
    <target name="run" depends="jar" description="Execute skiplist">
        <java fork="true" jar="build/jar/skiplist.jar"/>
    </target>
</project>
```

Step 3:
Since I now know about ANT, what do I need to do to allow the following commands to complete successfully?

```
$ ant -projecthelp
$ ant compile
$ ant jar
$ ant run
```

ANT can dramatically reduce the overhead related to compiling, archiving and executing your software projects.

Step 4:
Change the default target for this build file. Change the `default="compile"` to `default="run"` on line 2 of the build.xml file. Save your changes, then run:

```
$ ant run
```

Now by default, when you execute ANT, it will automatically compile, build a jar file and execute the code!

Step 5:
Now since it is trivial to run our application, replace the one of the names in the dictionary. You can test your modifications by simply running the application:

```
$ ant run
```

DEMO this deliverable to the lab instructor.

Good Luck