

Standard Template Library and the Java Collections Classes

Both C++ and Java have libraries that let us implement common data structures. C++ has STL, the Standard Template Library, and Java has the Collections classes. For high-level applications it is relatively rare to build your own linked list, hash table, binary search tree, etc. Instead these are already implemented for us through these classes! Nevertheless the occasion does arise to construct your own class or to modify the class, so it is important to know how the lower-level algorithms work.

The easiest way to demonstrate the classes is through code, so let's go straight to some examples!

C++ Standard Template Library

Documentation for the library is here: https://www.sgi.com/tech/stl/table_of_contents.html

Here we will just give a few examples for a couple of the classes in this library. You've already been using one of the classes in this library, the string class, which abstracts away the messiness of C-style strings terminated by a null character. As the name implies, the library uses templates. This allows you to substitute the data type of your choice into the class.

Vector

The vector class is like a cross between an array and a linked list. You can add items dynamically to the vector and access it like an array. Here is an example:

```
#include <iostream>
#include <vector>
using namespace std;

int main( )
{
    vector<int> v;
    cout << "Enter a list of positive numbers.\n"
         << "Place a negative number at the end.\n";

    int next;
    cin >> next;
    while (next > 0)
    {
        v.push_back(next);
        cout << next << " added. ";
        cout << "v.size( ) = " << v.size( ) << endl;
        cin >> next;
    }

    cout << "You entered:\n";
```

```

for (unsigned int i = 0; i < v.size( ); i++)
    cout << v[i] << " ";
cout << endl;

return 0;
}

```

We can use an **iterator** to access items in the STL collection. An iterator lets us move back and forth between items in the collection.

```

//Program to demonstrate STL iterators.
#include <iostream>
#include <vector>
using std::cout;
using std::endl;
using std::vector;
using std::vector<int>::iterator;

int main( )
{
    vector<int> container;

    for (int i = 1; i <= 4; i++)
        container.push_back(i);

    cout << "Here is what is in the container:\n";
    iterator p;
    for (p = container.begin( ); p != container.end( ); p++)
        cout << *p << " ";
    cout << endl;

    cout << "Setting entries to 0:\n";
    for (p = container.begin( ); p != container.end( ); p++)
        *p = 0;
    cout << "Container now contains:\n";

    for (p = container.begin( ); p != container.end( ); p++)
        cout << *p << " ";
    cout << endl;

    return 0;
}

```

The new C++11 ranged-based for loop and the “auto” type makes it easy to loop through vectors. This version outputs the numbers in the vector.

```

vector<int> vec;
vec.push_back( 10 );

```

```
vec.push_back( 20 );
```

```
for (int i : vec )  
{  
    cout << i;  
}
```

This next version shows how we can edit the items in the vector by specifying a reference. The following increments the numbers in the vector.

```
vector<int> vec;  
vec.push_back( 10 );  
vec.push_back( 20 );
```

```
for (int &i : vec )  
{  
    i++;  
}
```

The auto keyword figures out the proper data type based on the type of the expression on the right hand side of the assignment. It is often useful when we have complex data types in the vector. In our case it doesn't save too much:

```
vector<int> vec;  
vec.push_back( 10 );  
vec.push_back( 20 );
```

```
for (auto i : vec )  
{  
    cout << i;  
}
```

Some other functions we can use on a vector:

- c.insert(iterator, element)
 - Insert element before the iterator
- c.clear()
 - Remove all elements
- c.front()
 - Return a reference to the front item
- c.erase(iterator)
 - Remove the element at location Iterator

The Standard Template Library also has a list that is similar to a vector, but without the random access. It is basically a linked list.

Stack

The stack class behaves just like the stack data structure we discussed previously. Here is an example:

```
//Program to demonstrate use of the stack template class from the STL.
```

```
#include <iostream>
```

```
#include <stack>
```

```
using std::cin;
```

```
using std::cout;
```

```
using std::endl;
```

```
using std::stack;
```

```
int main( )
```

```
{
```

```
    stack<char> s;
```

```
    cout << "Enter a line of text:\n";
```

```
    char next;
```

```
    cin.get(next);
```

```
    while (next != '\n')
```

```
    {
```

```
        s.push(next);
```

```
        cin.get(next);
```

```
    }
```

```
    cout << "Written backward that is:\n";
```

```
    while ( ! s.empty( ) )
```

```
    {
```

```
        cout << s.top( );
```

```
        s.pop( );
```

```
    }
```

```
    cout << endl;
```

```
    return 0;
```

```
}
```

The key functions are push, pop, top, and empty.

Set

The set template class is, in some sense, the simplest container you can imagine. It stores elements without repetition. The first insertion places an element in the set. Additional insertions after the first have no effect, so that no element appears more than once. Each element is its own key. Basically, you just add or delete elements and ask if an element is in the set or not. Like all STL classes, the set template class was written with efficiency as a goal. To work efficiently, a set object stores its values in sorted order.

```
//Program to demonstrate use of the set template class.
```

```
#include <iostream>
#include <set>
using std::cout;
using std::endl;
using std::set;

int main()
{
    set<char> s;

    s.insert('A');
    s.insert('D');
    s.insert('D');
    s.insert('C');
    s.insert('C');
    s.insert('B');

    cout << "The set contains:\n";
    for (auto element : s)
        cout << element << endl;
    cout << endl;

    cout << "Set contains 'C': ";
    if (s.find('C') == s.end())
        cout << " no " << endl;
    else
        cout << " yes " << endl;

    cout << "Removing C.\n";
    s.erase('C');
    for (auto element : s)
        cout << element << endl;
    cout << endl;

    cout << "Set contains 'C': ";
    if (s.find('C') == s.end())
        cout << " no " << endl;
    else
        cout << " yes " << endl;
    return 0;
}
```

Map

A map is as an associative array. It could be implemented in a balanced tree, a hash table, etc. A traditional array maps from a numerical index to a value. For example, `a[10]=5` would store the number 5 at index 10. An associative array allows you to define your own indices using the data type of your choice. For example, `numberMap["c++"]=5` would associate the integer 5 with the string "c++". For convenience, the `[]` square bracket operator is defined to allow you to use an array-like notation to access a map, although you can also use the `insert` or `find` methods if you want.

Like a set object, a map object stores its elements sorted by its key values. You can specify the ordering on keys as a third entry in the angular brackets, `<>`. If you do not specify an ordering, a default ordering is used.

The easiest way to add and retrieve data from a map is to use the `[]` operator. Given a map object `m`, the expression `m[key]` will return a reference to the data element associated with `key`. If no entry exists in the map for `key` then a new entry will be created with the default value for the data element. This can be used to add a new item to the map or to replace an existing entry. For example, the statement `m[key] = newData;` will create a new association between `key` and `newData`. Note that care must be taken to ensure that map entries are not created by mistake. For example, if you execute the statement `val = m[key];` with the intention of retrieving the value associated with `key` but mistakenly enter a value for `key` that is not already in the map, then a new entry will be made for `key` with the default value and assigned into `val`.

```
#include <map>

map<string, string> planets;

planets["Mercury"] = "Hot planet";
planets["Venus"] = "Atmosphere of sulfuric acid";
planets["Earth"] = "Home";
planets["Mars"] = "The Red Planet";
planets["Jupiter"] = "Largest planet in our solar system";
planets["Saturn"] = "Has rings";
planets["Uranus"] = "Tilts on its side";
planets["Neptune"] = "1500 mile per hour winds";
planets["Pluto"] = "Dwarf planet";

cout << "Entry for Mercury - " << planets["Mercury"]
    << endl << endl;

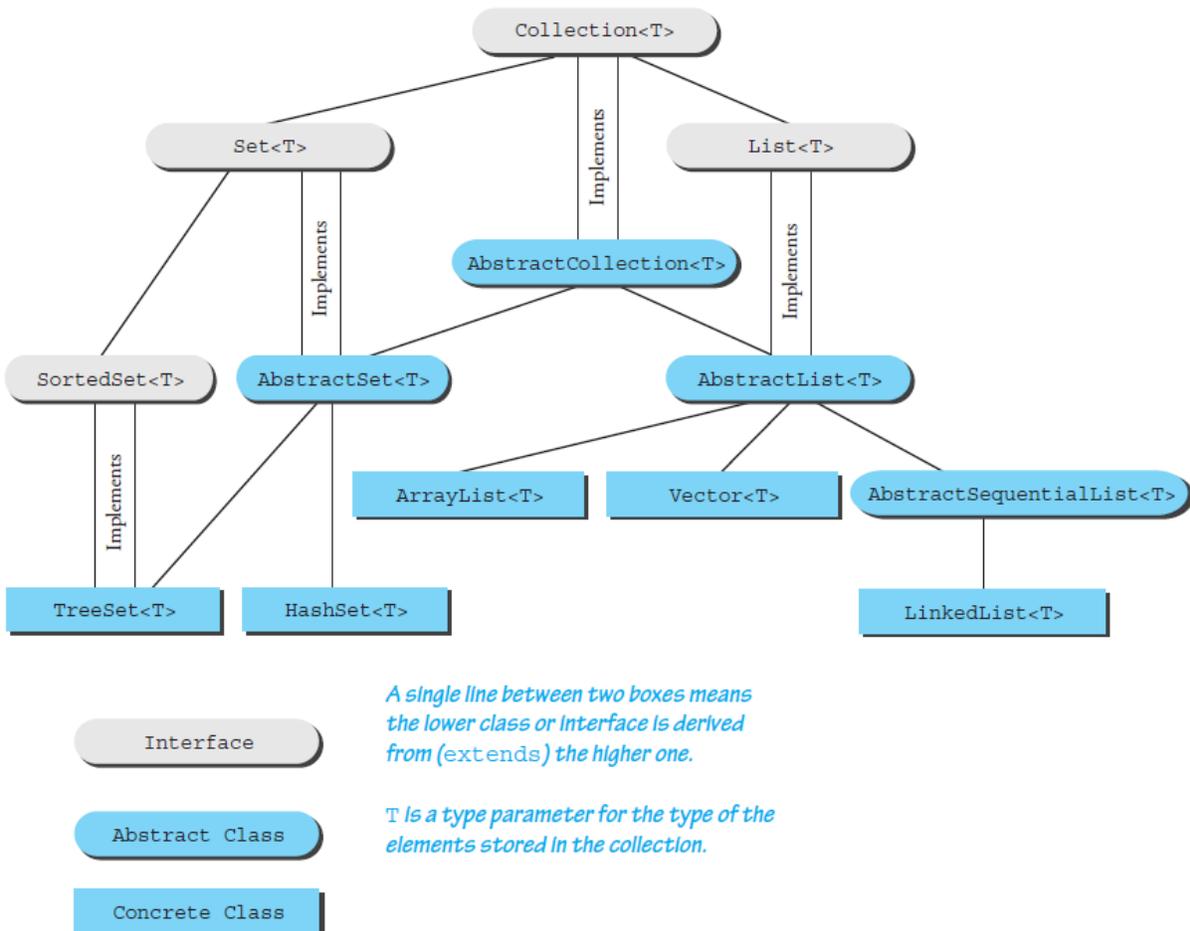
if (planets.find("Mercury") != planets.end())
    cout << "Mercury is in the map." << endl;
if (planets.find("Ceres") == planets.end())
    cout << "Ceres is not in the map." << endl << endl;

// Delete Pluto
planets.erase("Pluto");

// Iterator outputs planets in order sorted by key
cout << "Iterating through all planets: " << endl;
for (auto item : planets)
{
    cout << item.first << " - " << item.second << endl; // Key and Value
}
```

Java Collections

Java has a collections framework that is similar to the Standard Template Library, but does make inheritance and polymorphism a larger component of the library. There are several classes in the library; here is an overview:



There are some methods that every class implements in the Collections framework. This is kind of nice because you can get used to using methods like `size()` or `toArray()` or `contains()` and apply them to any collection.

<code>public boolean isEmpty()</code>	True if the calling object is empty
<code>public boolean contains(Object target)</code>	True if the calling object contains target
<code>public int size()</code>	Number of elements in the calling object
<code>public boolean equals(Object other)</code>	Our standard equality method
<code>public Object[] toArray()</code>	Return items in the collection as an array

ArrayList

Let's start with the ArrayList. Java has a vector class like C++, but in most cases we will use the ArrayList instead. It is pretty straightforward to use:

To access the arraylist code we can import the class via:

```
import java.util.ArrayList;
```

To create a variable of type ArrayList that is a list of type DataType use the following:

```
ArrayList<DataType> varName = new ArrayList<DataType>();
```

If you know how many items the arraylist will likely need, execution can be a bit faster by specifying an initial capacity.

Here is a basic example:

```
import java.util.ArrayList;
public class Test
{
    public static void main(String[] args)
    {
        ArrayList<String> stringList = new ArrayList<String>();

        stringList.add("foo");
        stringList.add("bar");
        stringList.add("zot");
        stringList.add("bah");

        System.out.println("Size of List: " + stringList.size());
        System.out.println("Contents:");
        for (int i = 0; i < stringList.size(); i++)
        {
            System.out.println("At index " + i + " value=" +
                stringList.get(i));
        }

        stringList.remove("bar");
        System.out.println("Contents after remove bar:");
        for (int i = 0; i < stringList.size(); i++)
        {
            System.out.println("At index " + i + " value=" +
                stringList.get(i));
        }

        stringList.remove(1);
        System.out.println("Contents after remove 1:");
        for (int i = 0; i < stringList.size(); i++)
        {
            System.out.println("At index " + i + " value=" +
                stringList.get(i));
        }
    }
}
```

```

stringList.add(1, "meh");
System.out.println("Contents after add 1:");
for (int i = 0; i < stringList.size(); i++)
{
    System.out.println("At index " + i + " value=" +
        stringList.get(i));
}

stringList.set(1, "bar");
System.out.println("Contents after set 1:");
for (int i = 0; i < stringList.size(); i++)
{
    System.out.println("At index " + i + " value=" +
        stringList.get(i));
}

System.out.println("Index of bar:");
System.out.println(stringList.indexOf("bar"));

System.out.println("Index of zzz:");
System.out.println(stringList.indexOf("zzz"));
}
}

```

Here is another simple example. Let's say we would like to make an arraylist of integers. To drive home the point that the arraylist only works with objects, we'll make our own Integer class (but we could have used the built-in Integer class too).

```

import java.util.ArrayList;

// Simple integer class with just two constructors.
// A more robust version would make the int m_value private with
// accessor methods instead

public class MyIntClass
{
    public int value;

    public MyIntClass()
    {
        value = 0;
    }
    public MyIntClass(int i)
    {
        value = i;
    }
}

// This class illustrates common uses of the arraylist
public class ArrayL
{
    public static void main(String[] args)
    {
        ArrayList<MyIntClass> v = new ArrayList<MyIntClass>();
    }
}

```

```

        int i;

        // First add 4 numbers to the arraylist
        for (i=0; i<4; i++)
        {
            v.add(new MyIntClass(i));
        }
        // Print out size of the arraylist, should be 4
        System.out.println("Size: " + v.size() + "\n");

        // Print arraylist
        System.out.println("Original arraylist:");
        PrintArraylist(v);

        // Remove the second element
        v.remove(2);
        // Print out the elements again
        System.out.println("After remove:");
        PrintArraylist(v);

        // Insert the number 100 at position 1
        v.add(1, new MyIntClass(100));
        // Print out the elements again
        System.out.println("After insert:");
        PrintArraylist(v);
    }

    // This method prints out the elements in the arraylist
    public static void PrintArraylist(ArrayList<MyIntClass> v)
    {
        MyIntClass temp;
        int i;

        for (i=0; i<v.size(); i++)
        {
            temp = v.get(i);
            System.out.println(temp.m_value);
        }
        System.out.println();
    }
}

```

Let's try using the indexOf method:

```

// Index of -1

System.out.println("Position of 1");

System.out.println(v.indexOf(new MyIntClass(1)));

```

Adding this gives an output of:

```

Position of 1
-1

```

This is not right, the list has a node with the value of 1 in position 2. We are getting -1 back because the `indexOf` method invokes the `equals` method for the `BaseType` object to determine which item in the `arraylist` matches the target. However, we didn't define one ourselves, so our program is really using the `equals` method defined for `Object` (which is inherited automatically for any object we make). However, this definition of `equals` only checks to see if the addresses of the objects are identical, which they are not in this case. What we really need to do is see if the integer values are the same by defining our own `equals` method. Add to the `MyIntClass` object:

```
public boolean equals(Object otherIntObject)
{
    MyIntClass otherInt = (MyIntClass) otherIntObject;
    if (this.m_value == otherInt.m_value)
        return true;
    return false;
}
```

If we run the program now, it will output:

```
Position of 1
2
```

A few comments of note: First, we had to define `equals` with an input parameter of type `Object`. This is because this is how the method is defined in the `Object` class which we need to override (and is invoked by the `indexOf` method). This means we have to typecast the object back to our class of `MyIntClass`. Once this is done we can compare the ints.

For Each Loop

Iterating through an `arraylist` is such a common occurrence that Java includes a special type of loop specifically for this purpose. Called the for-each loop, it allows you to easily iterate through every item in the `arraylist`. The syntax looks like this:

```
for (BaseType varName : ArrayListVariable)
{
    Statement with varName
}
```

Here is a modified version of the `PrintArrayList` method from the previous example, but using the for each loop:

```
// This method prints out the elements in the arraylist
public static void PrintArrayList(ArrayList<MyIntClass> v)
{
    for (MyIntClass intObj : v)
    {
        System.out.println(intObj.m_value);
    }
    System.out.println();
}
```

HashSet

The HashSet class is an implementation of the Set interface based on a hash table. Here we jump straight to some sample code:

```
import java.util.HashSet;
import java.util.Iterator;

public class HashSetDemo
{
    private static void outputSet(HashSet<String> set)
    {
        for (String s : set)
            System.out.print(s + " ");
        System.out.println();
    }

    public static void main(String[] args)
    {
        HashSet<String> round = new HashSet<String>( );
        HashSet<String> green = new HashSet<String>( );

        // Add some data to each set
        round.add("peas");
        round.add("ball");
        round.add("pie");
        round.add("grapes");

        green.add("peas");
        green.add("grapes");
        green.add("garden hose");
        green.add("grass");

        System.out.println("Contents of set round: ");
        outputSet(round);
        System.out.println("\nContents of set green: ");
        outputSet(green);

        System.out.println("\nball in set 'round'? " +
            round.contains("ball"));
        System.out.println("ball in set 'green'? " +
            green.contains("ball"));

        System.out.println("\nball and peas in same set? " +
            ((round.contains("ball") &&
              (round.contains("peas")))) ||
            (green.contains("ball") &&
```

```

        (green.contains("peas"))));
    System.out.println("pie and grass in same set? " +
        ((round.contains("pie") &&
        (round.contains("grass")) ||
        (green.contains("pie") &&
        (green.contains("grass"))));

    // To union two sets we use the addAll method.
    HashSet<String> setUnion = new HashSet<String>(round);
    round.addAll(green);
    System.out.println("\nUnion of green and round:");
    outputSet(setUnion);

    // To intersect two sets we use the removeAll method.
    HashSet<String> setInter = new HashSet<String>(round);
    setInter.removeAll(green);
    System.out.println("\nIntersection of green and round:");
    outputSet(setInter);
    System.out.println();
}
}

```

It is important to note that if you intend to use the `HashSet<T>` class with your own class as the parameterized type `T`, then your class must override the following methods:

```

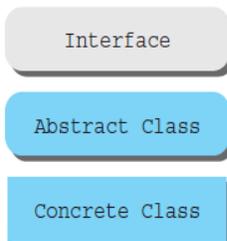
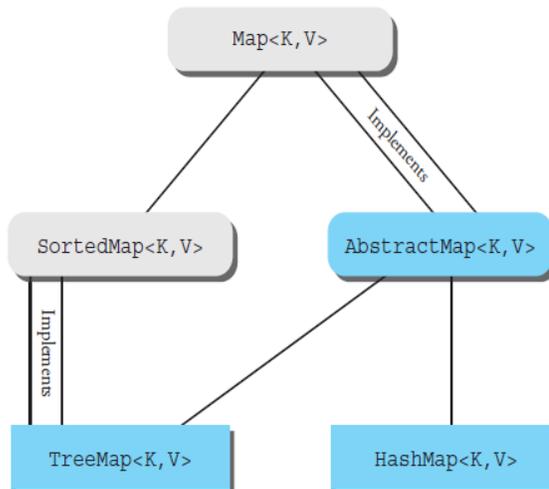
    public int hashCode();

    public boolean equals(Object obj);

```

Why? `hashCode` is needed for the hash function to work well. The `equals` method is used to determine if two objects in the set are the same.

Map and HashMap



A single line between two boxes means the lower class or interface is derived from (extends) the higher one.

`K` and `V` are type parameters for the type of the keys and elements stored in the map.

The map framework works the same way as a map in C++ STL. It lets you map one type onto another. There are several implementations of a map, including a `TreeMap` or `LinkedHashMap` or a `HashMap`. If you use a `HashMap`, you must make sure the item being mapped overrides the `hashCode()` and `equals()` methods as described above for `HashSet`.

```
import java.util.HashMap;
import java.util.Scanner;
public class HashMapDemo
{
    public static void main(String[] args)
    {
        // First create a hashmap with an initial size of 10 and
        // the default load factor
        HashMap<Integer,String> employees = new HashMap<Integer,String>(10);

        // Add several employees objects to the map using
        // their id as the key
        employees.put(10, "Joe");
        employees.put(49, "Andy");
    }
}
```

```

employees.put(91, "Greg");
employees.put(70, "Kiki");
employees.put(99, "Antoinette");
System.out.print("Added Joe, Andy, Greg, Kiki, ");
System.out.println("and Antoinette to the map.");

// Output everything in the map
System.out.println("The map contains:");
for (Integer key : employees.keySet())
    System.out.println(key + " : " + employees.get(key));
System.out.println();

// Ask the user to type a name. If found in the map,
// print it out.
Scanner keyboard = new Scanner(System.in);
int id;
do
{
    System.out.print("\nEnter an id to look up in the map. ");
    System.out.println("Enter -1 to quit.");
    id = keyboard.nextInt();
    if (employees.containsKey(id))
    {
        String e = employees.get(id);
        System.out.println("ID found: " + e.toString());
    }
    else if (id != -1)
        System.out.println("ID not found.");
} while (id != -1);
}
}

```