Java - First Impressions for a C++ Programmer

Steven Zeil

October 5, 2013

Contents

1 Program Structure 2

2 Program Structure == File Structure 5

3 Swimming in Pointers 24

4 Exceptions 26

5 Corresponding Data Structures 30

Moving from C++ to Java

- First, the good news...
  - You already know most of the language
    - Syntax is largely the same as C++
    - Semantics are similar
- Then, the bad news...
  - The "standard libraries" of Java and C++ are very, very different

. . . . . .

I'm not going to lecture on the basics

- You already know most of them
- Readings from the course Outline page take you to CS382 material
  - do the readings
  - and the labs
- In this lesson, I'll highlight the differences between Java and C++ that you will need to get started
- Next lesson, I'll begin focusing on OO issues in Java

. . . . . .

1 Program Structure

Hello Java

The traditional starting point:

```
public class HelloWorld {

  public static void main (String[] argv)
  {
    System.out.println ("Hello, World!");
  }
}
```

- Why is `main()` inside a class?

  - Because Java has no standalone functions.
  - *All* functions must be inside a class.

- Any class with a "public static void" main function taking an array of *String*s cane be executed.

. . . . . .

Class Syntax

C++

```
class MailingList {
private:
   struct Node {
      Contact data;
      Node* next;
   };
   Node* first;
public:
   MailingList()
   {first = 0;}
     ⋮
};
```

Java

```
public class MailingList {
   private class Node {
      Contact data;
      Node next;
   }
   private Node first;
public MailingList()
  {first = null;}
     ⋮
}
```

- `public` and `private` are labels for each declaration, not names of "regions"
- No trailing semi-colon

. . . . . .

Packages

A Java *package* is like a C++ *namespace*:

- A container of classes and other, smaller packages/namespaces

C++

```
namespace myUtilities {
  class Randomizer {
    ⋮
  };
};
```

Java

```
package myUtilities;
class Randomizer {
   ⋮
}
```

- Becomes part of the *fully qualified name* of a class

C++

```
myUtilities::Randomizer r;
```

Java

```
myUtilities.Randomizer r;
```

. . . . . .

We Can Have Short-Cuts

C++

```
using myUtilities::Randomizer
   ⋮
Randomizer r;
```

Java

```
import myUtilities.Randomizer;
   ⋮
Randomizer r;
```

. . . . . .

We Can Have Shorter Short-Cuts

C++

```
using namespace myUtilities;
   ⋮
Randomizer r;
```

Java

```
import myUtilities.*;
   ⋮
Randomizer r;
```

. . . . . .

Cultural Difference

- C++ programmers rarely invent their own namespaces
  - And use `using` to circumvent `std::`
- Java programmers frequently invent packages
  - Often multiple packages in a single project
  - Leaving things in the untitled default package is considered the sign of a beginner

. . . . . .

2 Program Structure == File Structure

Class == File

- In C++, I can put a class *MailingList* in `student.h` and `automobile.cpp` if I want
- Not so in Java:

  A class named "Foo" must be placed in a file named `Foo.java`

  – And upper/lower case do count!

.....

Classes are not Split into Two Files

- C++ distinguishes between class
  – declarations: usually placed in a header (.h file),

```
#ifndef MAILINGLIST_H
#define MAILINGLIST_H

#include <iostream>
#include <string>

#include "contact.h"

/**
   A collection of names and addresses
*/

class MailingList
{
public:
  MailingList();
  MailingList(const MailingList&);
  ~MailingList();

  const MailingList& operator= (const MailingList&);

  // Add a new contact to the list
  void addContact (const Contact& contact);
```

```
  // Remove one matching contact
  void removeContact (const Contact&);
  void removeContact (const Name&);

  // Find and retrieve contacts
  bool contains (const Name& name) const;
  Contact getContact (const Name& name) const;


  // combine two mailing lists
  void merge (const MailingList& otherList);

  // How many contacts in list?
  int size() const;


  bool operator== (const MailingList& right) const;
  bool operator< (const MailingList& right) const;

private:

  struct ML_Node {
    Contact contact;
    ML_Node* next;

    ML_Node (const Contact& c, ML_Node* nxt)
      : contact(c), next(nxt)
    {}
  };

  int theSize;
  ML_Node* first;
  ML_Node* last;

  // helper functions
  void clear();
  void remove (ML_Node* previous, ML_Node* current);

  friend std::ostream& operator<< (std::ostream& out, const MailingList& addr);
```


```
};

// print list, sorted by Contact
std::ostream& operator<< (std::ostream& out, const MailingList& list);


#endif
```

and

– definitions: usually placed in a compilation unit (.cpp file)

```
#include <cassert>
#include <iostream>
#include <string>
#include <utility>

#include "mailinglist.h"


using namespace std;
using namespace rel_ops;



MailingList::MailingList()
  : first(NULL), last(NULL), theSize(0)
{}

MailingList::MailingList(const MailingList& ml)
  : first(NULL), last(NULL), theSize(0)
{
  for (ML_Node* current = ml.first; current != NULL;
       current = current->next)
    addContact(current->contact);
}


MailingList::~MailingList()
{
  clear();
}
```

```
const MailingList& MailingList::operator= (const MailingList& ml)
{
  if (this != &ml)
    {
      clear();
      for (ML_Node* current = ml.first; current != NULL;
       current = current->next)
    addContact(current->contact);
    }
  return *this;
}


// Add a new contact to the list
void MailingList::addContact (const Contact& contact)
{
  if (first == NULL)
    { // add to empty list
      first = last = new ML_Node(contact, NULL);
      theSize = 1;
    }
  else if (contact > last->contact)
    { // add to end of non-empty list
      last->next = new ML_Node(contact, NULL);
      last = last->next;
      ++theSize;
    }
  else if (contact < first->contact)
    { // add to front of non-empty list
      first = new ML_Node(contact, first);
      ++theSize;
    }
  else
    { // search for place to insert
      ML_Node* previous = first;
      ML_Node* current = first->next;
      assert (current != NULL);
      while (contact < current->contact)
    {
```

```
      previous = current;
      current = current->next;
      assert (current != NULL);
    }
      previous->next = new ML_Node(contact, current);
      ++theSize;
    }
}


// Remove one matching contact
void MailingList::removeContact (const Contact& contact)
{
  ML_Node* previous = NULL;
  ML_Node* current = first;
  while (current != NULL && contact > current->contact)
    {
      previous = current;
      current = current->next;
    }
  if (current != NULL && contact == current->contact)
    remove (previous, current);
}


void MailingList::removeContact (const Name& name)
{
  ML_Node* previous = NULL;
  ML_Node* current = first;
  while (current != NULL
     && name > current->contact.getName())
    {
      previous = current;
      current = current->next;
    }
  if (current != NULL
     && name == current->contact.getName())
    remove (previous, current);
}
```

```
// Find and retrieve contacts
bool MailingList::contains (const Name& name) const
{
  ML_Node* current = first;
  while (current != NULL
     && name > current->contact.getName())
    {
      previous = current;
      current = current->next;
    }
  return (current != NULL
      && name == current->contact.getName());
}


Contact MailingList::getContact (const Name& name) const
{
  ML_Node* current = first;
  while (current != NULL
     && name > current->contact.getName())
    {
      previous = current;
      current = current->next;
    }
  if (current != NULL
      && name == current->contact.getName())
    return current->contact;
  else
    return Contact();
}




// combine two mailing lists
void MailingList::merge (const MailingList& anotherList)
{
```

```
// For a quick merge, we will loop around, checking the
// first item in each list, and always copying the smaller
// of the two items into result
MailingList result;
ML_Node* thisList = first;
const ML_Node* otherList = anotherList.first;
while (thisList != NULL and otherList != NULL)
  {
    if (thisList->contact < otherList->contact)
  {
    result.addContact(thisList->contact);
    thisList = thisList->next;
  }
    else
  {
    result.addContact(otherList->contact);
    otherList = otherList->next;
  }
  }
// Now, one of the two lists has been entirely copied.
// The other might still have stuff to copy. So we just copy
// any remaining items from the two lists. Note that one of these
// two loops will execute zero times.
while (thisList != NULL)
  {
    result.addContact(thisList->contact);
    thisList = thisList->next;
  }
while (otherList != NULL)
  {
    result.addContact(otherList->contact);
    otherList = otherList->next;
  }
// Now result contains the merged list. Transfer that into this list.
clear();
first = result.first;
last = result.last;
theSize = result.theSize;
result.first = result.last = NULL;
result.theSize = 0;
```


```
}

// How many contacts in list?
int MailingList::size() const
{
  return theSize;
}


bool MailingList::operator== (const MailingList& right) const
{
  if (theSize != right.theSize) // (easy test first!)
    return false;
  else
    {
      const ML_Node* thisList = first;
      const ML_Node* otherList = right.first;
      while (thisList != NULL)
    {
      if (thisList->contact != otherList->contact)
        return false;
      thisList = thisList->next;
      otherList = otherList->next;
    }
      return true;
    }
}


bool MailingList::operator< (const MailingList& right) const
{
  if (theSize < right.theSize)
    return true;
  else
    {
      const ML_Node* thisList = first;
      const ML_Node* otherList = right.first;
      while (thisList != NULL)
    {
      if (thisList->contact < otherList->contact)
```

```
        return true;
      else if (thisList->contact > otherList->contact)
        return false;
      thisList = thisList->next;
      otherList = otherList->next;
    }
      return false;
    }
}


// helper functions
void MailingList::clear()
{
  ML_Node* current = first;
  while (current != NULL)
    {
      ML_Node* next = current->next;
      delete current;
      current = next;
    }
  first = last = NULL;
  theSize = 0;
}


void MailingList::remove (MailingList::ML_Node* previous,
       MailingList::ML_Node* current)
{
  if (previous == NULL)
    { // remove front of list
      first = current->next;
      if (last == current)
    last = NULL;
      delete current;
    }
  else if (current == last)
    { // remove end of list
      last = previous;
      last->next = NULL;
```

```
      delete current;
    }
  else
    { // remove interior node
      previous->next = current->next;
      delete current;
    }
  --theSize;
}


// print list, sorted by Contact
std::ostream& operator<< (std::ostream& out, const MailingList& list)
{
  MailingList::ML_Node* current = list.first;
  while (current != NULL)
    {
      out << current->contact << "\n";
      current = current->next;
    }
  out << flush;
  return out;
}


book1.setTitle(''bogus title'');
assert (book1.getTitle() == ''bogus title'');

book2 = book1;
assert (book1 == book2);
book1.setTitle(''bogus title 2'');
assert (! (book1  == book2));



catalog.add(book1);
assert (catalog.firstBook() == book1);>
```

```
string s1, s2;
cin >> s1 >> s2;
if (s1 < s2)       ''abc'' < ''def''
                       ''abc'' < ''abcd''

    x y

Exactly one of the following is true for any x and y
    x == y
    x < y
    y < x

 namespace std{

   namespace relops {
template <class T>
bool operator!= (T left, T right)
{
  return !(left == right);
}


template <class T>
bool operator> (T left, T right)
{
  return (right < left);
}



     using namespace std::relops;
```

- Java puts the entire class into one file

```
package mailinglist;

/**
 * A collection of names and addresses
 */
public class MailingList implements Cloneable {
```

```
/**
 * Create an empty mailing list
 *
 */
public MailingList() {
 first = null;
 last = null;
 theSize = 0;
}

/**
 *  Add a new contact to the list
 *  @param contact new contact to add
 */
public void addContact(Contact contact) {
 if (first == null) {
 // add to empty list
 first = last = new ML_Node(contact, null);
 theSize = 1;
 } else if (contact.compareTo(last.contact) > 0) {
 // add to end of non-empty list
 last.next = new ML_Node(contact, null);
 last = last.next;
 ++theSize;
 } else if (contact.compareTo(first.contact) < 0) {
 // add to front of non-empty list
 first = new ML_Node(contact, first);
 ++theSize;
 } else {
 // search for place to insert
 ML_Node previous = first;
 ML_Node current = first.next;
 assert (current != null);
 while (contact.compareTo(current.contact) < 0) {
 previous = current;
 current = current.next;
 assert (current != null);
 }
 previous.next = new ML_Node(contact, current);
 ++theSize;
```


```
 }
}

/**
 *  Remove one matching contact
 *  @param c remove a contact equal to c
 */
public void removeContact(Contact c) {
 ML_Node previous = null;
 ML_Node current = first;
 while (current != null && c.compareTo(current.contact) > 0) {
 previous = current;
 current = current.next;
 }
 if (current != null && c.equals(current.contact))
 remove(previous, current);
}

/**
 * Remove a contact with the indicated name
 * @param name name of contact to remove
 */
public void removeContact(String name) {
 ML_Node previous = null;
 ML_Node current = first;
 while (current != null && name.compareTo(current.contact.getName()) > 0) {
 previous = current;
 current = current.next;
 }
 if (current != null && name == current.contact.getName())
 remove(previous, current);

}

/**
 * Search for contacts
 * @param name name to search for
 * @return true if a contact with an equal name exists
 */
public boolean contains(String name) {
```

```
 ML_Node current = first;
 while (current != null && name.compareTo(current.contact.getName()) > 0) {
 current = current.next;
 }
 return (current != null && name == current.contact.getName());
}

/**
 * Search for contacts
 * @param name name to search for
 * @return contact with that name if found, null if not found
 */
public Contact getContact(String name) {
 ML_Node current = first;
 while (current != null && name.compareTo(current.contact.getName()) > 0) {
 current = current.next;
 }
 if (current != null && name == current.contact.getName())
 return current.contact;
 else
 return null;
}

/**
 * combine two mailing lists
 *
 */
public void merge(MailingList anotherList) {
 // For a quick merge, we will loop around, checking the
 // first item in each list, and always copying the smaller
 // of the two items into result
 MailingList result = new MailingList();
 ML_Node thisList = first;
 ML_Node otherList = anotherList.first;
 while (thisList != null && otherList != null) {
 if (thisList.contact.compareTo(otherList.contact) < 0) {
 result.addContact(thisList.contact);
 thisList = thisList.next;
 } else {
 result.addContact(otherList.contact);
```

```
 otherList = otherList.next;
 }
 }
 // Now, one of the two lists has been entirely copied.
 // The other might still have stuff to copy. So we just copy
 // any remaining items from the two lists. Note that one of these
 // two loops will execute zero times.
 while (thisList != null) {
 result.addContact(thisList.contact);
 thisList = thisList.next;
 }
 while (otherList != null) {
 result.addContact(otherList.contact);
 otherList = otherList.next;
 }
 // Now result contains the merged list. Transfer that into this list.
 first = result.first;
 last = result.last;
 theSize = result.theSize;
}

/**
 * How many contacts in list?
 */
public int size() {
 return theSize;
}

/**
 * Return true if mailing lists contain equal contacts
 */
public boolean equals(Object anotherList) {
 MailingList right = (MailingList) anotherList;
 if (theSize != right.theSize) // (easy test first!)
 return false;
 else {
 ML_Node thisList = first;
 ML_Node otherList = right.first;
 while (thisList != null) {
 if (!thisList.contact.equals(otherList.contact))
```


```
 return false;
 thisList = thisList.next;
 otherList = otherList.next;
 }
 return true;
 }
}

public int hashCode() {
 int hash = 0;
 ML_Node current = first;
 while (current != null) {
 hash = 3 * hash + current.hashCode();
 current = current.next;
 }
 return hash;
}

public String toString() {
 StringBuffer buf = new StringBuffer("{");
 ML_Node current = first;
 while (current != null) {
 buf.append(current.toString());
 current = current.next;
 if (current != null)
 buf.append("\n");
 }
 buf.append("}");
 return buf.toString();
}

/**
 * Deep copy of contacts
 */
public Object clone() {
 MailingList result = new MailingList();
 ML_Node current = first;
 while (current != null) {
 result.addContact((Contact) current.contact.clone());
 current = current.next;
```

```
     }
     return result;
    }

    private class ML_Node {
     public Contact contact;

     public ML_Node next;

     public ML_Node(Contact c, ML_Node nxt) {
     contact = c;
     next = nxt;
     }
    }

    private int theSize;

    private ML_Node first;

    private ML_Node last;

    // helper functions
    private void remove(ML_Node previous, ML_Node current) {
     if (previous == null) {
     // remove front of list
     first = current.next;
     if (last == current)
     last = null;
     } else if (current == last) {
     // remove end of list
     last = previous;
     last.next = null;
     } else {
     // remove interior node
     previous.next = current.next;
     }
     --theSize;
    }

}
```

- Function bodies are written immediately after their declaration
- To C++ programmers, these look like *inline* functions

. . . . . .

Package == Directory

- In C++, we can put our files into any directory we want, as long as we give the appropriate paths in our compilation commands
- In Java, all items that belong in package `packageName` must be stored in a directory filenamepackageName/

. . . . . .

Packaging and Compiling 1

Suppose we are building a Java project in `~/jproject`.
If we have a class

```
public class HelloWorld {

  public static void main (String[] argv)
  {
    System.out.println ("Hello, World!");
  }
}
```

- It would be stored in `~/jproject/HelloWorld,java`.
- The commands to compile and run this would be

```
cd ~/jproject
javac HelloWorld.java
java HelloWorld
```

. . . . . .

Packaging and Compiling 2

Suppose we are building a Java project in `~/jproject`.
If we have a class

```
package Foo;

public class HelloWorld {

  public static void main (String[] argv)
  {
    System.out.println ("Hello, World!");
  }
}
```

- It would be stored in `~/jproject/Foo/HelloWorld,java`.
- The commands to compile and run this would be

```
cd ~/jproject
javac Foo/HelloWorld.java
java Foo.HelloWorld
```

.....

Packaging and Compiling 3

Suppose we are building a Java project in `~/jproject`.
If we have a class

```
package Foo.Bar;

public class HelloWorld {

  public static void main (String[] argv)
  {
    System.out.println ("Hello, World!");
  }
}
```

- It would be stored in `~/jproject/Foo/Bar/HelloWorld,java`.
- The commands to compile and run this would be

```
cd ~/jproject
javac Foo/Bar/HelloWorld.java
java Foo.Bar.HelloWorld
```

.....

Is Java Just Trying to Be Annoying?

Actually, no.

- To compile a C++ program, we have to give explicit paths to *each* compilation unit in our compilation commands and we need to `#include` each header, giving the correct path to it.
- The Java compiler finds the source code of our other classes that we use by looking at
  - the fully qualified name (e.g., `MyUtilities.Randomizer`, or
  - our `import` statements
- It just follows the package/class names to find the directory and file in which the rest of our code is located.
- At run time, the loader acts similarly to find our compiled code.

. . . . . .

3 Swimming in Pointers

Primitives are Familiar

- *int, long, float, double*
  - *boolean*, not "bool"
  - *char* is 16 bits, not 8, to permit the use of Unicode
- Variables of these types behave as you would expect

```
int x = 2;
int y = x;
++x;
System.out.println ("x=" + x + " y=" + y);
```

prints "x=3 y=2"

. . . . . .

Everything Else is a Pointer

```
void foo(java.awt.Point p) {
  p.x = 1;
  java.awt.Point w = p;
  w.x = 2;
  System.out.println ("p.x=" + p.x + " w.x=" + w.x);
}
```

prints “p.x=2 w.x=2”

- Why did `p.x` change value?
  - Because *p* and *w* are references (pointers), so

    ```
    java.awt.Point w = p;
    ```

    causes *w* to point to the same value that *p* does.

. . . . . .

Lots of Allocation

Because all new class variables are really pointers, all new class values have to be created on the heap:

C++

```
Point p (1,2);
```

Java

```
Point p = new Point(1,2);
```

. . . . . .

Arrays of Pointers

C++ programmers need to be particularly careful dealing with arrays:

C++

```
int a[10];
Point* p = new Point[10];
```

Java

```
int[] b = new int[10];
Point q = new Point[10];
for (int i = 0;
     i < q.length; ++i)
  q[i] = new Point();
```

Without the loop, *q* would actually be an array of null pointers.

. . . . . .

Because there are so many pointers

- Sharing in Java is much more common than copying
  - If you do need a distinct copy, use the `clone()` function
    - We’ll see later that this is a “standard” function on all Java objects
- It’s a good thing that Java features automatic garbage collection!

. . . . . .

Beware of ==

- The == operator works like you would expect on primitives

```
int x = 23;
int y = 23;
if (x == y)
   System.out.println ("Of course!");
```

- But for class objects, == is comparing addresses:

```
Point p = new Point(1,2);
Point q = new Point(1,2);
if (p == q)
   System.out.println ("Not gonna happen");
else
   System.out.println ("Surprise!");
```

. . . . . .

equals

To compare objects to see if they have the same *contents*, use the `equals` function:

```
Point p = new Point(1,2);
Point q = new Point(1,2);
if (p.equals(q))
   System.out.println ("That's better.");
else
   System.out.println ("Pay no attention...");
```

. . . . . .

4 Exceptions

Exceptions

An *exception* is a run-time error signal.

- It may be signalled (*thrown*) by the underlying runtime system...
  - or by programmer-suppled code
- Programs can *catch* exceptions and *handle* them ...
  - or let them go and allow the program to abort

. . . . . .

Playing Catch

Try compiling and (if successful), running each of the following:

```
import java.io.*;

/**
   Demo of a program that may throw exceptions.
   @param argv The name of a file to open for input
*/
public class OpenFile1 {

  /**
     Attempt to open a file
  */
  static void openFile (String fileName) {
    FileReader reader = new FileReader(fileName);
  }

  /**
     Attempt to open the file whose name is given in
     the first command line parmaeter
  */
  public static void main (String[] argv) {
    String fileName = argv[1];
    openFile (fileName);
  }
}
```

```
import java.io.*;

/**
   Demo of a program that may throw exceptions.
   @param argv The name of a file to open for input
*/
public class OpenFile2 {

  /**
     Attempt to open a file
  */
  static void openFile (String fileName)
    throws java.io.FileNotFoundException
```

```
  {
    FileReader reader = new FileReader(fileName);
  }

  /**
     Attempt to open the file whose name is given in
     the first command line parmaeter
  */
  public static void main (String[] argv) {
    String fileName = argv[0];
    openFile (fileName);
  }
}
```

```
import java.io.*;

/**
   Demo of a program that may throw exceptions.
   @param argv The name of a file to open for input
*/
public class OpenFile3 {

  /**
     Attempt to open a file
  */
  static void openFile (String fileName)
    throws java.io.FileNotFoundException
  {
    FileReader reader = new FileReader(fileName);
  }

  /**
     Attempt to open the file whose name is given in
     the first command line parmaeter
  */
  public static void main (String[] argv) {
      try {
      openFile (argv[0]);
      }
      catch (java.io.FileNotFoundException ex)
```

```
      {
          System.err.println ("Something is wrong with the file: " + ex);
      }
      System.out.println ("All done");
  }
}
```

using both the names of exiting and non-exiting files, or no name at all.

. . . . . .

Unchecked Exceptions

Exceptions come in two main kinds: checked and unchecked

- *unchecked* exceptions could arise almost anywhere
  - e.g., `NullPointerException`, `ArrayIndexOutOfBoundsException`
- functions need not declare that they might throw these

. . . . . .

Checked Exceptions

checked exceptions are more specialized

- include all programmer-defined exceptions
- functions *must declare* if they can throw these

. . . . . .

Another Cultural Difference

- C++ actually has exceptions as well.
  - Same `throw` and `try` - `catch` syntax
  - Can't declare what exceptions a function is known to throw
- But they are used much more widely in Java

. . . . . .

5 Corresponding Data Structures

The Java API

The Java API is huge, but well documented

- Focus initially on packages `java.lang`, `java.io`, and `java.util`
- BTW, You can use the **javadoc** to generate the same kind of documentation for your own code. Example

.....

Strings

C++ *std::string* : Java *java.lang.String*

- Strings in Java are *immutable*
  - You cannot change the contents of a string value
  - But you can compute a slightly different value and make a string variable point to the different value

C++

```
string s = ...;
s[s.size()/2] = 'b';
s = s + s;
```

Java

```
String s = ...;
s = s.substring(0,s.length()/2)
  + 'b'
  +  s.substring(s.length()/2+1);
s = s + s;
```

.....

If You Need to Change a Java String

use a *StringBuilder*

```
StringBuilder sb = new StringBuilder();
String line = input.readline();
while (line != null) {
  sb.append(line);
  sb.append("\n");
  line = input.readline();
}
String allTheText = sb.toString();
```

.....

vector : ArrayList

C++

```
vector<string> v;
v.push_back("foo");
cout << v[0] << endl
```

Java

```
ArrayList<String> v = new ArrayList<String>();
v.add("foo");
System.out.println (v.get(0));
```

.....

list : LinkedList

C++

```
list<string> L;
L.push_back("foo");
cout << L.front() << endl
```

Java

```
LinkedList<String> L = new LinkedList<String>();
L.add("foo");
System.out.println (L.getFirst());
```

.....

set : HashSet

C++

```
set<string> s;
s.insert("foo");
cout << s.count("foo") << endl
```

Java

```
HashSet<String> S = new HashSet<String>();
S.add("foo");
System.out.println ("" + S.contains("foo"));
```

.....

map : HashMap

C++

```
map<string,int> zip;
zip["ODU"] = 23529;
cout << zip["ODU"] << endl
```

Java

```
HashMap<String,Integer> zip
      = new HashMap<String,Integer>();
zip.put("ODU", 23529);
System.out.println (zip.get("ODU"));
```

.....