A Comparison Between Using the STL list Template and the Java List Collection

The example code here compares how a waiting list might be maintained for a course (as in CPS122 labs)

C++

```
/*
 *   course.h
 *
 *   Declaration for class Course
 *
 *   Copyright (c) 2001, 2003, 2013 - Russell C. Bjork
 */

#include "student.h"
#include <list>
using std::list;

class Course
{
    public:
         ...
         /* Add a student to waiting list */
         void addWaiting(Student * student);

         /* Check to see if student is on waiting list */
         bool isWaiting(Student * student) const;

         /* Remove first student from waiting list */
         Student * removeFirstWaiting();

         /* Remove specific student from waiting list */
         void removeWaiting(Student * student);

         /* Print a report on this course to cout */
         void printReport() const;
     private:
         ...
         typedef list < Student * > WaitingList;
         WaitingList _waiting;
};
```

Java

```
/* Course.java */

package registrationsystem.model;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.Map;
import java.util.TreeMap;

/** Representation for a course
 *
 *  @author Russell C. Bjork
 */
public class Course
{
    ...
    private List<Student> waiting = new LinkedList<Student();
    ...

    /** Add a student to the waiting list for a course
     *
     *  @param student the student to add
     */
    public void addWaiting(Student student)
    {
        waiting.add(student);
    }

    /** Test to see whether a student is on the waiting list
     *
     *  @param student the student to check
     *  @return true if the student is on the waiting list
     */
    public boolean isWaiting(Student student)
    {
        return waiting.contains(student);
    }
```

```
/*
 *   course.cc - Implementation of class Course
 *   Copyright (c) 2001, 2003 - Russell C. Bjork
 */

#include <iostream.h>
using namespace std;
#include "course.h"
...
void Course::addWaiting(Student * student)
{ _waiting.push_back(student); }

bool Course::isWaiting(Student * student) const
{    // We have to do this one the hard way!
     for (WaitingList::const_iterator iter =
            _waiting.begin(); iter != _waiting.end(); iter ++)
          if ((* iter) == student)
               return true;
     return false;
}

Student * Course::removeFirstWaiting()
{
     Student * result = _waiting.front();
     _waiting.pop_front();
     return result;
}

void Course::removeWaiting(Student * student)
{ _waiting.remove(student); }

void Course::printReport() const
{
     ...
     if (_waiting.empty())
          cout << "No one waiting";
     else
     {
          cout << "Waiting:" << endl;
          for (WaitingList::const_iterator iter =
               _waiting.begin();iter != _waiting.end();iter ++)
            cout << (* iter) -> getName() << endl;
     }
     ...
}
```

```
/** Remove the first student from the waiting list when a
  * slot opens up
  *
  *  @return the student that was removed
  */
 public Student removeFirstWaiting() {
     return waiting.remove(0);
 }

 /** Remove a specific student from the waiting list
  *
  *  @param student the student to remove
  */
 public void removeWaiting(Student student)
 {
     waiting.remove(student);
 }

 /** Print a report on the course - enrollments and waiting
  */
 public void printReport()
 {
     ...
     if (waiting.size() == 0)
         System.out.println("No one waiting");
     else
     {
         System.out.println("Waiting:");
         Iterator<Student> waitingIterator =
                                          waiting.iterator();
         while (waitingIterator.hasNext())
             System.out.println((waitingIterator.next()).
                                  getName());
     }
     ...
 }
 ...
}
```