COMP20012 Tutorial 1: Collections – The questions

Happy New Year. For some of you, this tutorial takes place before many lectures have taken place for COMP20012, so it does not depend on any material from the course itself. We have based it on material already covered in previous programming courses, for which you have notes. There is also useful background material in the course textbook, Weiss, and copious material on the web e.g. `http://java.sun.com/docs/books/tutorial/collections/`.

1. `java.util` provides the Collections Framework, a set of interfaces for working with groups of objects. Read your previous notes on this topic and explain to your tutor the main features of the Collections Framework.

2. A *stack* is a collection of items that has the property that *the last item in is the first item out* (LIFO), so the operation *add* adds an item onto the stack and *remove* takes off the item added most recently. The operations *add* and *remove* for stacks are usually called *push* and *pop*.

   A common text processing problem is that of bracket-matching. For example the string `"[(z + b) + (c * d)]"` has matching brackets, but `"[(z + b] + (c * d))"` does not, even though there are an appropriate number of opening and closing brackets of each type, because bracket pairs are not properly nested.

   Describe an algorithm which uses a stack to determine whether a given string has matching brackets. You do not need to produce `Java` code, just a description of the algorithm.

3. Explain to your tutor how `Iterator`s are used. Why is a separate `Iterator` object needed, rather than using a class or instance variable of the `Collection` of interest?

   Write static methods to print out the items in any `Collection` in each of the following cases:-

   a) in the default order given by an `Iterator`

   b) in reverse order. (Note: you can not use a `ListIterator`, as this is not available for all `Collection` objects.)

   c) in reverse order, but without using another intermediate `Collection` (Hint: recursion)

COMP20012 Tutorial 1: Collections – Tutors notes

This tutorial is supposed to be revision of first year Java material, but your students may be a bit rusty in this area.

Graham

Aim: This sheet is a revision for the students and focuses on some of the topics which will be evident in this course unit. It covers abstract types as the Collection class in Java and a range of data structures which may be used as representations of collections. It also acts as a revision of the construction of iterative and recursive routines, including the relationship between recursion and stacks.

1. A tutorial on the Collections Framework can be found at `http://java.sun.com/docs/books/tutorial/collections/`.

   The students have had a brief introduction to Collections in previous Java courses. The main points are that the Collections interface provides a uniform interface to various types of collection, such as Vector, List, Stack, Set etc. The interface itself can be found at `http://java.sun.com/javase/6/docs/api/java/util/Collection.html`. The methods in the interface are used to update and query the underlying objects. There is much more to say, but this will be covered later in the course.

2. No Java is required here. The algorithm is the standard one whereby the string is examined character by character (you might ask how to do this in Java; using a `String` and the `charAt` method is the obvious way), opening brackets of various types are pushed on to a stack as they occur and when a closing bracket is found, the top of the stack examined to see if it has the same type (or false returned if the stack is empty). If so, it is popped and we continue; if not, the brackets don't match. If anything is left on the stack at the end, then we have unmatched brackets.

   if they find this easy you might want to discuss approaches to implementation. For completeness, here is my version of the code

```
public static boolean bracketMatch(String str)
{
  Stack<Character> openers = new Stack <Character>();

  for (int i = 0 ; i < str.length(); i++)
    {
      char thisChar = str.charAt(i);
      if (isOpenBracket(thisChar))
         openers.push(new Character(thisChar));
      else if (isClosedBracket(thisChar))
        {
          if (openers.empty())
             return false;
          char topChar = ( openers.peek()).charValue();
          if (bracketsMatch(topChar,thisChar))
             openers.pop();
          else
             return false;
```

```
        }
    }

  return (openers.empty());
}

private static boolean isOpenBracket(char c) {
  return ( c == '(' || c == '[' || c == '{');
}

private static boolean isClosedBracket(char c) {
  return ( c == ')' || c == ']' || c == '}');
}

private static boolean bracketsMatch(char c1, char c2) {
  return (
          c1 == '(' && c2 == ')' ||
          c1 == '[' && c2 == ']' ||
          c1 == '{' && c2 == '}'
          );
}
```

3. `Iterator`s are used to successively access all the members of a collection. Each `Collection` provides an `iterator` method to return such an object. The `Iterator` interface ensures the existence of methods `next()` and `hasNext()` to access and query the iterator. Using a separate iterator object means that the collection can be iterated over several times independently, with the iterator keeping a record of the state of each iteration.

   The code for the methods is given below. The only slightly tricky point is the use of an auxiliary (private) method to do the recursion.

```
public static <T> void printCollection( Collection<T> col )
{
  Iterator<T> itr = col.iterator();
  while (itr.hasNext())
    {
      System.out.println(itr.next());
    }
}

public static <T> void printCollectionRev( Collection<T> col )
{
  Iterator<T> itr = col.iterator();
  Stack<T> s = new Stack<T>();

  while (itr.hasNext()) {
    s.push(itr.next());
  }

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

```
    System.out.println(s.pop());
  }
}


public static <T> void printCollectionRevRec( Collection<T> col )
{
  Iterator<T> itr = col.iterator();
  printCollectionRevRecAux(itr);
}

private static <T> void printCollectionRevRecAux( Iterator<T> itr )
{
  if (itr.hasNext()) {
    T n = itr.next();
    printCollectionRevRecAux(itr);
    System.out.println(n);
  }
}
```