PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

St. George Campus

May EXAMINATIONS

CSC 207H1S
Duration — 3 hours

Examination Aids: Any handwritten or printed materials. No electronic aids.

Student Number: _____

Last (Family) Name(s): _____

First (Given) Name(s): _____

---

*Do **not** turn this page until you have received the signal to start.*
(In the meantime, please fill out the identification section above, and read the instructions below *carefully*.)

This final examination consists of 7 questions on 18 pages (including this one). *When you receive the signal to start, please make sure that your copy of the examination is complete.*
There are two mostly-blank pages at the end of the exam that you may use if you run out of space.
Comments are not necessary unless specifically indicated in the question, you may abbreviate `System.out.println` as `S.o.p`, and you may assume that any necessary imports have been done.

Good Luck!

MARKING GUIDE

# 1:	_____/10
# 2:	_____/ 6
# 3:	_____/14
# 4:	_____/14
# 5:	_____/ 7
# 6:	_____/12
# 7:	_____/17
BONUS MARKS:	_____/ 2
TOTAL:	_____/80

Question 1. [10 MARKS]

In E4 you wrote code to handle article information, include a title, URL, and description, all embedded in XML. The next two pages contain a parser for a different file format; see the comment for class `FSA` for details.

Draw the finite state automaton that corresponds to the code:

```
public class Record {
    private String title;
    private String url;
    private String description = "";

    public void setTitle(String t) {  title = t;  }
    public void setURL(String u) {  url = u;  }
    public void addToDescription(String line) {  description = description + line + " ";  }

    public String toString() {  return title + "\n" + url + "\n" + description.trim();  }
}

/**
 * A parser for webpage records, where each webpage record has this format:
 *
 * TITLE: [title of webpage]
 * URL:  [url of webpage]
 * DESCRIPTION:
 * [first line of description]
 * [second line of description]
 * ...
 *
 * Each description will have at least one and possibly many lines of text,
 * none of which will start with "TITLE:", "URL:", or "DESCRIPTION:".
 */
public class FSA {
    // The states.
    final static int START = 0;                     final static int SEEN_DESCRIPTION = 4;
    final static int SEEN_TITLE = 1;                final static int ERROR = 5;
    final static int SEEN_URL = 2;                  final static int END = 6;
    final static int SEEN_DESCRIPTION_HEAD = 3;

    public static List<Record> recordList;
    private static Record currentRecord;
    private static int state;

    private static boolean isTitle(String line) {  return line.startsWith("TITLE: ");  }

    private static boolean isURL(String line) {  return line.startsWith("URL: ");  }

    private static boolean isDescriptionHead(String line) {  return line.startsWith("DESCRIPTION:");  }

    private static boolean isDescription(String line) {
        return !isDescriptionHead(line) && !isURL(line) && !isTitle(line);
    }

    private static void handleTitle(String line) {
        if (state == START || state == SEEN_DESCRIPTION) {
            if (currentRecord != null) {
                recordList.add(currentRecord);
            }
            currentRecord = new Record();
            currentRecord.setTitle(line.substring(7));
            state = SEEN_TITLE;
        } else {
            state = ERROR;
        }
    }
```

```
    private static void handleURL(String line) {
        if (state == SEEN_TITLE) {
            currentRecord.setURL(line.substring(5));
            state = SEEN_URL;
        } else {
            state = ERROR;
        }
    }

    private static void handleDescription(String line) {
        if (state == SEEN_DESCRIPTION_HEAD || state == SEEN_DESCRIPTION) {
            currentRecord.addToDescription(line);
            state = SEEN_DESCRIPTION;
        } else {
            state = ERROR;
        }
    }

    private static void handleDescriptionHead(String line) {
        if (state == SEEN_URL) {
            state = SEEN_DESCRIPTION_HEAD;
        } else {
            state = ERROR;
        }
    }

    /**
     * Parse the contents of fileName, which contains records as described
     * in the class description.
     */
    public static void parse(String fileName) throws IOException {
        BufferedReader br = new BufferedReader(new FileReader(fileName));
        String line = null;
        recordList = new ArrayList<Record>();
        state = START;
        while ((line = br.readLine()) != null && state != ERROR) {
            if (isTitle(line)) {
                handleTitle(line);
            } else if (isURL(line)) {
                handleURL(line);
            } else if (isDescriptionHead(line)) {
                handleDescriptionHead(line);
            } else if (isDescription(line)) {
                handleDescription(line);
            }
        }

        if (state == SEEN_DESCRIPTION) {
            recordList.add(currentRecord);
        } else {
            System.err.println("Oops, error on line " + line);
        }
    }
}
```

Question 2. [6 MARKS]

Below are some Unix commands in which some parts of the commands have been deleted and replaced with a question mark (`?`). (Notice that `eben:~>` is the prompt, and the stuff between the `:` and the `>` is the current directory. Also, the lines have been numbered for easier reference.)

```
1:  eben:~> mkdir testdir
2:  eben:~> chmod 000 testdir
3:  eben:~> cd testdir
4:  testdir: Permission denied.
5:  eben:~> chmod u+? testdir
6:  eben:~> cd testdir
7:  eben:~/testdir> ls
8:  ls: .: Permission denied
9:  eben:~/testdir> chmod u+? ~/testdir
10: eben:~/testdir> ls
11: eben:~/testdir>
```

Part (a) [1 MARK]

When the `cd` command is run in line 3, it returns "permission denied". Why?

Part (b) [1 MARK]

What is the full command that was run in line 5? Replace the question mark with the appropriate character.

Part (c) [1 MARK]

When the `ls` command is run in line 7, it returns "permission denied". Why?

Part (d) [1 MARK]

What is the full command that was run in line 9? Replace the question mark with the appropriate character.

Part (e) [1 MARK]

After all the commands have been executed, who can access the directory `testdir`?

Part (f) [1 MARK]

How many files does `testdir` contain?

Question 3. [14 MARKS]

Part (a) [7 MARKS] Complete the following Python method:

```
def equal(list1, list2):
    '''Return true if list1 and list2 contain exactly the same elements in
    the same order, including the contents of any nested lists.'''
```

Part (b) [7 MARKS]

In Python, `map` is a function that takes another function and a list as parameters and applies the function to every element in the list. In this part you will work with mapping functions that work on trees instead of lists.

Consider this node class for a binary tree:

```
class Node:
    def __init__(self, value, left=None, right=None):
        self.value = value
        self.left = left
        self.right = right
```

The next function calls a helper function `map_all_helper`. Write that helper function.

```
def map_all(f, tree):
    '''Apply function f to the values of every Node in tree in preorder order and return the
    results in a list.'''

    result = [] # The list of results.
    map_all_helper(f, tree, result)
    return result
```

Your answer:

How would you modify your helper function so that it only applied `f` to only the leaves, rather than the whole tree?

Question 4. [14 MARKS]

Part (a) [7 MARKS]

The code on the next page contains a method called `sort` that uses a modified version of insertion sort to sort an array of strings representing Roman numerals in the range 1 to 10. The code is not very testable. Refactor the sort routine so that it is more testable—a call to `sortBetter` will have the same result as a call to the original `sort` method, except that `sortBetter` will use your refactored code, including any new methods that you create during the refactoring.

The Roman numerals, from 1 to 10, are: I, II, III, IV, V, VI, VII, VIII, IX, X.

Part (b) [3 MARKS]

The original `sort` method contains a bug. (It isn't in the sorting code, it's in the Roman numeral-to-`int` conversion.) Find the bug and briefly describe how you would fix it in your refactored code:

Briefly discuss whether this is any easier than it would have been to fix it in the original method:

Part (c) [4 MARKS]

Below, fill in the table with a good set of test fixtures for testing both `sortBetter` and the helper methods that you wrote. Give the best three fixtures that you can think of for each method you are testing.

Notice the style for writing the arrays for arguments and expected results. Notice also that you don't have to use quotes. (If you wrote these as JUnit tests, you of course would need to use proper Java, but for this question we want to see only the fixures.)

Method to test	Argument	Expected result
`sortBetter`	`[X, I]`	`[I, X]`

```
import java.util.*;

public class RomanNumeralSort {
    public static void sort(String[] a) {
        for (int i = 1; i < a.length; i++) {
            int j = i;

            // Remember the one at index 1.
            String curr = a[i];
            int currInt = 0;
            if (curr.equals("X")) {
                currInt = 10;
            } else if (curr.startsWith("V")) {
                currInt = 4 + curr.length();
            } else if (curr.endsWith("V")) {
                currInt = 4;
            } else {
                currInt = curr.length();
            }

            String prev = a[j - 1];
            int prevInt = 0;
            if (prev.equals("X")) {
                prevInt = 10;
            } else if (prev.startsWith("V")) {
                prevInt = 4 + prev.length();
            } else if (prev.endsWith("V")) {
                prevInt = 4;
            } else {
                prevInt = prev.length();
            }

            // Find out where to insert curr, shuffling up the items as we go.
            while ((j > 0) && (prevInt > currInt)) {
                a[j] = a[j - 1];
                j--;
                if (j > 0) {
                    prev = a[j - 1];
                    if (prev.equals("X")) {
                        prevInt = 10;
                    } else if (prev.startsWith("V")) {
                        prevInt = 4 + prev.length();
                    } else if (prev.endsWith("V")) {
                        prevInt = 4;
                    } else {
                        prevInt = prev.length();
                    }
                }
            }

            a[j] = curr;
        }
    }
```

```
public static void betterSort(String[] a) {
```

Question 5. [7 MARKS]

Complete method `map` below so that it implements the `map` functionality.

Hint: `Method` objects define a method `invoke(Object obj, Object[])`. The first parameter specifies the object from which the method will be invoked, and the second parameter specifies the list of parameters that will be passed to the `Method` being invoked.

Do not bother handling exceptions, and assume all necessary imports have been done.

```
/**
 * Apply m with arguments args to each element in list, returning the results in another list.
 *
 * @param  m     the method that will be applied to each element in list.
 * @param  args  the arguments to give to m when it is applied.
 * @param  list  the list of objects to which the specified method will be applied.
 * @return the list of results.
 */
public static List map(Method m, Object[] args, List list) {




}
```

Question 6. [12 MARKS]

Part (a) [4 MARKS]

There are lots of different spelling conventions for the English language, and there is no easy way to generalize those differences so that short, simple programs can be written to convert text from one spelling convention to another. However, for this question we will assume that text that follows the American convention can be converted to text that follows the Canadian convention simply by replacing the last two letters of any word ending in *or* with *our*. For instance, *honor* would become *honour*.

Write a Python regular expression that converts American spelling to Canadian spelling, as described above.

Part (b) [4 MARKS]

If two words end in the same few letters, they often rhyme with each other. For this problem, if the ending words on two consecutive lines of text rhyme with each other, then the lines are said to rhyme.

Write a Python regular expression that matches a single string containing a newline-separated pair of lines that rhyme. Assume that two words rhyme with each other if at least the last three characters of the last word of each line are identical.

Part (c) [4 MARKS]

Given this string:

```
a 123 c 123456789 c 123 d
```

Consider two regular expressions `(a.*c).*d` and `(a.*?c).*d`. Briefly discuss whether the characters that the groups `(a.*c)` and `(a.*?c)` match are the same.

Question 7. [17 MARKS]

Many websites, such as `tribe.net` and `orkut.com`, maintain social networks of friends. This question has you analyze data that might be stored by such a website. This will have you read and understand an XML file, write a class describing the information contained in that XML file, and then write a method in Java doing something interesting with that class.

Here is an XML file describing people and their friendships with each other. The file contains all the tags and attributes that you will need to handle in this question; there will be no other kinds of tags or attributes. Notice that all the person tags have the same format.

```
<?xml version="1.0" ?>
<people>
   <person personID="1" name="John Lennon">
      <friends>
         <friend personID="2">
         <friend personID="3">
         <friend personID="4">
      </friends>
   </person>
   <person personID="2" name="Paul McCartney">
      <friends>
         <friend personID="1">
         <friend personID="3">
         <friend personID="4">
      </friends>
   </person>
   <person personID="3" name="Luke Skywalker">
      <friends>
         <friend personID="2">
         <friend personID="4">
      </friends>
   </person>
   <person personID="4" name="Princess Leia">
      <friends>
         <friend personID="3">
      </friends>
   </person>
</people>
```

Part (a) [3 MARKS]

Design class `Person`; be sure to include all relevant information contained in the XML document as instance variables. To save you time, don't write getters and setters: just assume that getters and setters exist for all instance variables.

```
public class Person {




}
```

Part (b) [7 MARKS]

Write a `DomVisitor` subclass, including any necessary instance variables, that creates two `java.util.HashMap` objects. One of the `HashMap`s, `idToPerson`, will associate integers representing `personID`'s to the corresponding `Person` objects. The second `HashMap`, `idToFriendIds`, will associate integers representing `personID`'s to corresponding lists of friends, represented as `java.util.LinkedList<Integer>` objects containing the `personId` of each friend.

```
public class PersonVisitor extends DomVisitor {

    private HashMap<Integer, Person> idToPerson;
    private HashMap<Integer, LinkedList<Integer>> idToFriendIds;
```

Bonus. [2 MARKS]

Why did we ask for two `HashMap`s in the DomVisitor subclass, instead of asking for a single `HashMap` that associates `Person` objects with a `List<Person>` representing the associated friends?

Part (c) [7 MARKS]

The XML file obviously contains lists of people and their friends. Below, we add two methods (one of which is a helper method that you will write) to the `PersonVisitor` class in order to recursively compute a person's *circle* of friends. The helper method has these parameters:

- An `int` representing the `PersonID` of the person whose circle of friends is to be computed.
- A `Set<Integer>` representing the friends found so far for the specified `PersonID`.

The method returns a `java.util.LinkedList<Person>` containing the names of everyone in the circle of friends of the person whose name is provided as a parameter. A person's circle of friends includes anyone listed as one of their direct friends, as well as anyone who is in the circle of friends of one of their direct friends.

You are free to write any necessary helper methods. Comments are not necessary. Remember, these methods are to become a part of the `PersonVisitor` class, so you should use `idToPerson` and `idToFriendsIds` to gather the necessary information.

```
/** The top-level method for computing a circle of friends */
public Set<Integer> friendCircle(int pId) {
  Set result = new HashSet<Integer>();
  friendCircle(pId, result);
  return result;
}

/**The helper method. Please fill in the body */
public void friendCircle(int pId, Set<Integer> found) {









}
```

*[This is a "blank" page. Use the space below for rough work. This page will **not** be marked, unless you clearly indicate the part of your work that you want us to mark.]*

*[This is a "blank" page. Use the space below for rough work. This page will **not** be marked, unless you clearly indicate the part of your work that you want us to mark.]*

*[This is a "blank" page. Use the space below for rough work. This page will **not** be marked, unless you clearly indicate the part of your work that you want us to mark.]*

Total Marks = 80