First Name: ______________________________ **Last Name**: ______________________________

McGill ID: ______________________________ **Section**: ______

Faculty of Science
COMP-202A - Introduction to Computing I (Fall 2008)
Final Examination

Thursday, December 11, 2008
14:00–17:00

Examiners:
Mathieu Petitpas [Section 1]
Prof. Paul Kry [Section 2]
Gregory B. Prokopski [Section 3]

Instructions:

- **DO NOT TURN THIS PAGE UNTIL INSTRUCTED**
- This is a **closed book** examination; notes, slides, textbooks, and other forms of documentation are **not** allowed.
- **Non-programmable calculators** are allowed (though you should not need one).
- **Computers, PDAs, cell phones, and other electronic devices** are **not** allowed.
- Answer all questions on the examination paper; if you need additional space, use page 16 or the booklets supplied and clearly indicate where each question is continued. **In order to receive full marks, you must show all work.**
- This final examination has **19** pages including this cover page. **Pages 17-19 can be detached from the rest of the examination for easy reference and do not have to be returned.**

1	2	Subtotal
/10	/5	/15

3	4	5	Subtotal
/10	/10	/15	/35

6	7	8	Subtotal
/20	/15	/15	/50

Total
/100

Section 1 - Short Questions

[10] 1. In one or two sentences, explain the differences between the concepts in each of the following pairs. BE BRIEF; overly long answers will be grounds for mark deductions.

(a) `throw` and `throws`

(b) `while` and `do while`

(c) local variables and instance variables

(d) iteration and recursion

(e) classes and objects

[5] 2. What does the following program display?

```
public class X {
    private char x = 'x';
    public X() {
      System.out.println("A x=" + x + "\t this.x=" + this.x );
    }
    public void x() {
      System.out.println("B x=" + x + "\t this.x=" + this.x );
    }
    public void x( char x )   {
      System.out.println("C x=" + x + "\t this.x=" + this.x );
    }
    public void x( long x )   {
      System.out.println("D x=" + x + "\t this.x=" + this.x );
    }
    public void x( double x ) {
      System.out.println("E x=" + x + "\t this.x=" + this.x );
    }
    public void x( String x ) {
      System.out.println("F x=" + x + "\t this.x=" + this.x );
    }

    public static void main( String[] args ) {
        try {
            X x = new X();
            x.x( );
            x.x( 1 );
            x.x( 2.0f );
            x.x( '3' );
            x.x( 4L );
            x.x( 5.0 );
            x.x( "6" );
        } catch ( Exception x ) {
            System.err.println("Avenge my death!");
        }
    }
}
```

YOUR ANSWER CONTINUED:

Total marks for Section 1: **15**

Section 2 - Long Questions

[10] 3. The *look-and-say sequence* is the sequence of integers beginning as follows:

```
1, 11, 21, 1211, 111221, 312211, 13112221, 1113213211, ...
```

To generate a member of the sequence from the previous member, read off the digits of the previous member, counting the number of digits in groups of the same digit. For example:

- 1 is read off as “one one” or 11.
- 11 is read off as “two ones” or 21.
- 21 is read off as “one two, then one one” or 1211.
- 1211 is read off as “one one, then one two, then two ones” or 111221.
- 111221 is read off as “three ones, then two twos, then one one” or 312211.

Write a program that prints the first 15 terms of this sequence. *Hint:* use a `String` to store each term, starting with the `String "1"`.

YOUR ANSWER CONTINUED:

[10] 4. Write a method with the following header:

```
public static int sumOfSquares(int n)
```

This method computes the square of each of the first `n` integers and adds them. That is, if the value of `n` is 3, the method should return $1^2 + 2^2 + 3^2$, or 14. You can assume that the value of `n` will be a positive non-zero integer. You must use recursion to get full marks; however, a correct iterative solution will still be worth 6 marks. Do not use the method `Math.pow()` in your answer.

[15] 5. What is displayed when the `main()` method of class `Confusion` is executed? Carefully track the state of memory in the space provided, then clearly indicate the program output below.

```
public class Thingy {
  private int i, j;
  private static int k = 0;

  public Thingy(int i) {
    this.i = i;
    k = k + 1;
    this.j = k;
  }
  public void setI(int i) {
    this.i = i;
  }
  public String toString() {
    return this.i + " (" + this.j + ")";
  }
}

public class Confusion {
  public static int confuse( int i1, int i2, Thingy t1, Thingy t2,
                             int[][] m1, int[][] m2, Thingy[] ta ) {
    i1 = 10;
    i2 = 20;
    t1.setI(30);
    t2 = new Thingy(40);
    m1 = new int[2][1];
    m1[0][0] = 50;
    m1[1][0] = 60;
    m2[0][0] = 70;
    m2[1][0] = 80;
    m2[1] = m2[0];
    ta[0].setI(90);
    ta[1] = new Thingy(100);
    return i2;
  }

  public static void main(String[] args) {
    int v1 = 1, v2 = 2;
    Thingy myT1 = new Thingy(3), myT2 = new Thingy(4);
    int[][] myM1 = { {5}, {6} }, myM2 = { {7}, {8} };
    Thingy[] thingyArray = { new Thingy(9), new Thingy(10) };

    v2 = confuse( v1, v2, myT1, myT2, myM1, myM2, thingyArray );

    System.out.println( v1 + ", " + v2 );
    System.out.println( myT1 + ", " + myT2 );
    System.out.println("{ {" + myM1[0][0] + "}, {" + myM1[1][0] + "} }, "
                  + "{ {" + myM2[0][0] + "}, {" + myM2[1][0] + "} }" );
    System.out.println( thingyArray[0] + ", " + thingyArray[1] );
  }
}
```

USE THE SPACE BELOW TO TRACK THE STATE OF DIFFERENT VARIABLES IN MEMORY:

USE THE SPACE BELOW TO CLEARLY INDICATE THE PROGRAM OUTPUT:

Total marks for Section 2: $\overline{\textbf{35}}$

Section 3 - Programming Questions

This section involves classes which are part of a program that reads files containing numeric course evaluation results, and writes a summary of the evaluation results to another file. The complete program involves five classes, and your task will be to write three of these five classes:

- `ResultLoader`
- `ReportWriter`
- `GenerateReport`

Complete details for these classes will be provided in the relevant questions. The other two classes **have already been implemented**, and **you can use them** in the classes that you write. They are `Course` and `Instructor`. **Complete details for these classes**, such as the type of objects they represent, the methods they define, what these methods do, what parameters they take, what values they return, and so on, **are specified on the last page of this examination.**

[20] 6. Write the `ResultLoader` class. This class defines only one method, with the following header:

```
public static void load(Instructor instructor, String fileName)
```

This method opens the file whose name is given by `fileName`. This file contains evaluation results for a course offered during a given semester. The `load()` method reads these results from the file, and creates a `Course` object which it adds to the `Instructor` object provided in the method parameters. Note that you must also compute the average response for each question and set these averages with the appropriate method of the `Course` object you created. Be sure to close the file once all the information has been read.

A file containing evaluation results has the following format:

- The course code is on the first line.
- The date when the course was offered on the next line.
- Each of the remaining lines in the file contains one student's answers to each question on the evaluation questionnaire. The number of answers will be `Course.NUMBER_QUESTIONS`. Each of these answers is an integer value, and is separated by one or more white space characters.

For example, a file containing the evaluation results for a course whose code is COMP-202, offered in the fall term of 2008, taken by 2 students, and where `Course.NUMBER_QUESTIONS` is equal to 4, would look like this:

```
COMP-202
September 2008
5 4 5 3
4 5 3 3
```

In the example above, the averages for the 4 questions are 4.5, 4.5, 4, and 3. Note again that the actual number of questions is specified in the constant `Course.NUMBER_QUESTIONS`.

You may assume that the file specified by `fileName` follows the above format exactly; in other words, the file does not contain any formatting errors.

Remember that when opening and reading from the file, `IOException`s could occur. Your `load()` method must propagate these exceptions to the method which called `load()` (this might involve making a slight change to the method header).

WRITE YOUR `ResultLoader` CLASS IN THE SPACE BELOW:

[15] 7. Write the `ReportWriter` class. This class defines only one method, with the following header:

```
public static void writeReport(Instructor instructor, String fileName)
```

This method opens a file for writing, whose name is given by `fileName`, generates a summary of the evaluation results for courses taught by the specified `instructor`, and closes the file. The format of the report is as follows:

- The first line contains `"Evaluation results for: "`, followed by the instructor's name.
- If the instructor has not taught any courses, `"- No results found"` is written to the file.
- Otherwise, for each course the instructor has taught, the evaluation results are displayed. All the information about a course is displayed on the same line, with the information about each course being displayed on separate lines.
  - First, `"- "` is written to the file.
  - Then, the course code is printed, followed by `" ("`, the date string for when the course was taught, and then `"): "`.
  - Finally, the average score for each question is printed, rounded to exactly two decimal spaces, and separated by one space.

For example, suppose instructor X taught COMP-250 during the winter semester of 2008, and COMP-202 during the fall semester starting of 2008. The average evaluation scores for COMP-250 were 4.5, 4, 4, and 4.5, while the average evaluation scores for COMP-202 were 4.5, 4.5, 4, and 3. Therefore, the report written to the file would look like this:

```
Evaluation results for: X
- COMP-250 (2008-01): 4.50 4.00 4.00 4.50
- COMP-202 (2008-09): 4.50 4.50 4.00 3.00
```

The order in which the courses appear in the report does not matter.

Remember that when opening and writing to the file, `IOExceptions` could be thrown. In such circumstances, your `writeReport()` method must propagate these exceptions to the method that called `writeReport()` (this might involve making a slight change to the method header).

YOUR `ReportWriter` CLASS CONTINUED:

[15] 8. Write the `GenerateReport` class, with a `main()` method, which does the following:

- The method first checks the number of command-line arguments. If there are less than 2 command-line arguments, the `main()` method should display the following error message to the standard **error** stream and terminate:

```
Usage: java GenerateReport <name> <outfile>
  [<file1> <file2> ...]
```

- The method then loads the evaluation results stored in the files whose names are given by each command-line argument after the second (the third, the fourth, ..., and the last). If an error occurs while reading one of the files, your `main()` method should display the following error message to the standard **error** stream:

```
Could not read from file <file>
```

  Here, `<file>` is the file that could not be read. However, if your program fails to read from one of the files, it should still attempt to read from the other files.

- The method finally writes a summary of the evaluation results that you loaded in the previous step. The summary should be written to the file whose name is given by the second command-line argument, and the name of the instructor on the generated report is given by the first command-line argument. Furthermore, the format of this summary should be the one described in Question 7. If an error occurs while writing the summary, your `main()` method should display the following error message to the standard **error** stream:

```
Could not write to file <file>
```

  Here, `<file>` is the file to which the report could not be written. Your program should then terminate.

You should use the `ResultLoader` and `ReportWriter` classes written in the previous two questions to write the `main()` method required for this question. You may assume that both classes have been implemented correctly, even if you did not successfully complete the two previous questions.

YOUR `GenerateReport` CLASS CONTINUED:

Total marks for Section 3: **50**

Total marks: **100**

USE THIS PAGE IF YOU NEED ADDITIONAL SPACE. CLEARLY INDICATE WHICH QUESTION(S) YOU ARE ANSWERING HERE.

SUMMARY OF JAVA STANDARD LIBRARY METHODS FOR SELECTED CLASSES
(DETACH THESE PAGES FROM THE REST OF THE EXAMINATION FOR EASY REFERENCE)

- `String` (package `java.lang`) Methods:
  - `public int length()`: Returns the length of this `String`.
  - `public char charAt(int index)`: Returns the `char` value at the specified `index`.
  - `public boolean equals(Object anObject)`: Compares this `String` to `anObject`
  - `public boolean equalsIgnoreCase(String anotherString)`: Compares, ignoring case considerations, this `String` to `anotherString`.
  - `public int compareTo(String anotherString)`: Lexicographic comparison to `anotherString`.
  - `public boolean startsWith(String prefix)`: Tests if this `String` starts with the specified `prefix`.
  - `public boolean endsWith(String suffix)`: Tests if this `String` ends with the specified `suffix`.
  - `public int indexOf(int ch)`: Returns the index within this `String` of the first occurrence of character `ch`, `-1` if it does not occur.
  - `public int indexOf(int ch, int fromIndex)`: Returns the index within this `String` of the first occurrence of character `ch`, starting the search at position `fromIndex`; returns `-1` if `ch` does not occur in this `String`.
  - `public int indexOf(String str)`: Returns the index within this `String` of the first occurrence of substring `str`, `-1` if it does not occur.
  - `public int indexOf(String str, int fromIndex)`: Returns the index within this `String` of the first occurrence of substring `str`, starting at position `fromIndex`; returns `-1` if `str` does not occur in this `String`.
  - `public String substring(int beginIndex)`: Returns a new `String` that is a substring of this `String`, composed of the characters starting at position `beginIndex` (inclusive).
  - `public String substring(int beginIndex, int endIndex)`: Returns a new `String` that is a substring of this `String`, composed of the characters starting at position `beginIndex` (inclusive), and ending at position `endIndex` (exclusive).
  - `public String replace(char oldChar, char newChar)`: Returns a new `String` resulting from replacing all occurrences of `oldChar` in this `String` with `newChar`.
  - `public String toLowerCase()`: Returns a new `String` consisting of all the characters in this `String` converted to lower case.
  - `public String toUpperCase()`: Returns a new `String` consisting of all the characters in this `String` converted to upper case.
  - `public String trim()`: Returns a copy of this `String`, with leading and trailing whitespace omitted.
- `Scanner` (package `java.util`) Methods:
  - `public Scanner(File source)`: Constructs a new `Scanner` that produces values scanned from the specified file.
  - `public Scanner(InputStream source)`: Constructs a new `Scanner` that produces values scanned from the specified input stream.
  - `public Scanner(String source)`: Constructs a new `Scanner` that produces values scanned from the specified string.
  - `public void close()`: Closes this `Scanner`
  - `public boolean hasNext()`: Returns `true` if this `Scanner` has another token in its input.
  - `public boolean hasNextDouble()`: Returns `true` if the next token in this `Scanner`'s input can be interpreted as a `double` value using the `nextDouble()` method.
  - `public boolean hasNextInt()`: Returns `true` if the next token in this `Scanner`'s input can be interpreted as an `int` value using the `nextInt()` method.
  - `public boolean hasNextLine()`: Returns `true` if there is another line in the input of this `Scanner`
  - `public boolean hasNextLong()`: Returns `true` if the next token in this `Scanner`'s input can be interpreted as a `long` value using the `nextLong()` method.
  - `public String next()`: Finds and returns the next complete token from this `Scanner`.
  - `public double nextDouble()`: Scans the next token of the input as a `double`.
  - `public int nextInt()`: Scans the next token of the input as an `int`.
  - `public String nextLine()`: Advances this `Scanner` past the current line and returns the input read.
  - `public long nextLong()`: Scans the next token of the input as a `long`.

- `ArrayList<E>` (package `java.util`) Methods:
  - `public int size()`: Returns the number of elements in this list.
  - `public boolean isEmpty()`: Returns `true` if this list contains no elements.
  - `public boolean contains(Object o)`: Returns `true` if this list contains element `o`.
  - `public int indexOf(Object o)`: Returns the index of the first occurrence of element `o` in this list, or -1 if this list does not contain this element.
  - `public Object[] toArray()`: Returns an array containing all of the elements in this list in proper sequence (from first to last element).
  - `public E get(int index)`: Returns the element at position `index` in this list.
  - `public E set(int index, E element)`: Replaces the element at the position `index` in this list with the specified `element`.
  - `public boolean add(E e)`: Appends the specified `element` to the end of this list.
  - `public void add(int index, E element)`: Inserts the specified element at the position `index` in this list.
  - `public E remove(int index)`: Removes the element at position `index` in this list.
  - `public boolean remove(Object o)`: Removes the first occurrence of the specified element `o` from this list, if it is present.
  - `public void clear()`: Removes all of the elements from this list.
- `PrintStream` (package `java.io`) Methods:
  - `public PrintStream(File file)`: Creates a new `PrintStream` with the specified `file`.
  - `public PrintStream(String fileName)`: Creates a new `PrintStream`, with the specified `fileName`.
  - `public print(boolean b)`: Prints `boolean` value `b`.
  - `public print(char c)`: Prints `char` value `c`.
  - `public print(double d)`: Prints `double` value `d`.
  - `public print(int i)`: Prints `int` value `i`.
  - `public print(Object o)`: Prints `Object` `o`.
  - `public print(String s)`: Prints `String` `s`.
  - `public println()`: Terminates the current line by writing the line separator string.
  - `public println(boolean b)`: Prints `boolean` value `b` and then terminates the line.
  - `public println(char c)`: Prints `char` value `c` and then terminates the line.
  - `public println(double d)`: Prints `double` value `d` and then terminates the line.
  - `public println(int i)`: Prints `int` value `i` and then terminates the line.
  - `public println(Object o)`: Prints `Object` `o` and then terminates the line.
  - `public println(String s)`: Prints `String` `s` and then terminates the line.
- `DecimalFormat` (package `java.text`) Methods:
  - `public DecimalFormat()`: Creates a `DecimalFormat` using the default pattern and symbols for the default locale.
  - `public DecimalFormat(String pattern)`: Creates a `DecimalFormat` using the given pattern and the symbols for the default locale.
  - `public String format(double number)`: Formats the number with in the pattern of this `DecimalFormat` object.
- `File` (package `java.io`) Methods:
  - `public File( String pathname )`: Creates a `File` representing the file at the given `pathname`.

DESCRIPTIONS OF CLASSES PROVIDED FOR QUESTIONS 6, 7, AND 8
(DETACH THIS PAGE FROM THE REST OF THE EXAMINATION FOR EASY REFERENCE)

- `Course`: Objects which belong to the this class contain full evaluation results for one course offered by an instructor during a semester.

  Fields:

  – `public static final int NUMBER_QUESTIONS`: The number of questions on the evaluation questionnaire.

  Methods:

  – `public Course(Instructor instructor, String code, String date)`: Creates a new `Course` object containing evaluation results for the course with the specified course `code` offered during the semester specified by `date` and taught by the given `instructor`. `Course` objects instantiated using this constructor initially contain no evaluation results.
  – `public String getCourseCode()`: Returns the course code for this `Course`.
  – `public String getDate()`: Returns the date string that specifies when the `Course` was offered.
  – `public void setAverage(int question, double average)`: Sets the average score for the specified `question`. This method assumes that `questionNumber` is greater than or equal to `1`, and less than or equal to `Course.NUMBER_QUESTIONS`.
  – `public double getAverage(int question)`: Returns the average score for the specified `question`. This method assumes that `questionNumber` is greater than or equal to `1`, and less than or equal to `Course.NUMBER_QUESTIONS`.

- `Instructor`: Objects which belong to this class contain information about an instructor and the `Course`s he/she has taught in the past.

  Methods:

  – `public Instructor(String name)`: Initializes a new `Instructor` object that will contain evaluation results for all the courses taught by the instructor whose name is `name`. `Instructor` objects instantiated with this constructor are initially not associated with any `Course` objects.
  – `public String getName()`: Returns the name of this `Instructor`.
  – `public void addCourse(Course course)`: Adds `course` to the list of courses taught by this `Instructor`.
  – `public ArrayList<Course> getAllCourses()`: Returns an `ArrayList` containing all `Course` objects representing courses taught by this `Instructor`.