First Name: _____ **Last Name**: _____

McGill ID: _____ **Section**: _____

Faculty of Science
COMP-202B - Introduction to Computing I (Winter 2010) - All Sections
Midterm Examination

Thursday, March 11, 2010
18:30–20:30

Examiners: Milena Scaccia [Section 1]
Mathieu Petitpas [Sections 2 and 3]

Instructions:

- **DO NOT TURN THIS PAGE UNTIL INSTRUCTED**
- This is a **closed book** midterm 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 this examination paper** and return it. If you need additional space, use page 14 or the booklets supplied and clearly indicate where each question is continued. **In order to receive full marks for a question, you must show all work.**
- This midterm examination has **16** pages including this cover page, and is printed on both sides of the paper. Pages 15-16 contain information about useful classes and methods.

1	2	3	4	Subtotal
/5	/4	/5	/6	/20

5	6	Subtotal
/15	/15	/30

7	Subtotal
/50	/50

Total
/100

Section 1 - Short Questions

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

(a) Compiler

(b) Comment

(c) Loop

(d) Method

(e) `null`

[4] 2. Consider the following statement, where `p`, `q`, `r`, and `e` are all variables of type `boolean`:

```
e = (p && !q) || (q && !r) || !p || r;
```

Complete the following truth table by calculating the value of `e` for each possible combinations of values for `p`, `q`, and `r`, and entering the result in the corresponding row of the truth table.

p	q	r	e
false	false	false	
false	false	true	
false	true	false	
false	true	true	
true	false	false	
true	false	true	
true	true	false	
true	true	true	

[5] 3. What will be displayed to the screen after each of the following code fragments is executed?

(a)
```
int H = 44;
int a = 55;
System.out.println("H" + "a");
System.out.println(H + a);
```

(b)
```
int i1 = 3;
int i2 = 4;
double[] a1 = {5.0, 2.0, 3.5, 4.0, 1.0};
double[] a2 = a1;
a1[0] = a1[0] + a1[1];
a2[1] = i1 / i2;
a2[2] = (int)a2[2] * 3;
a2[3] *= a2[3] + 1.0;
a2[4] = 3.0 * (i1 % i2);
System.out.println("a2[0]: " + a2[0]);
System.out.println("a2[1]: " + a2[1]);
System.out.println("a2[2]: " + a2[2]);
System.out.println("a2[3]: " + a2[3]);
System.out.println("a2[4]: " + a2[4]);
```

(c)
```
String pig = "catch-22";
String dog = "catch-" + 22;
System.out.println("Animals are equal: " + pig == dog);
```

[6] 4. For each of the following conditions, write a **boolean expression** that evaluates to `true` if and only if the condition is true. You are not allowed to use any methods from the Java Platform API (that is, the Java standard library) to answer this question.

(a) The values stored in variables `m` (which specifies the number of a month, where January is month `1`) and `d` (which specifies the number of a day within a month), both of type `int`, represent a statutory holiday in the Winter 2010 semester. There are only two such holidays: March 26 (Good Friday) and March 29 (Easter Monday).

(b) The character stored in variable `c` (of type `char`) is a letter (upper-case or lower-case)

(c) **Exactly two** of the values stored in variables `a`, `b` and `c` (all of type `int`) are odd.

Section 2 - Long Questions

[15] 5. Consider the following program:

```
   public class Mystery {
     public static int mystify(int[] a, int x) {
       int s, e, m, p;

1      s = 0;
2      e = a.length - 1;
3      p = -1;

4      while (s <= e && p == -1) {
5        m = (s + e) / 2;
6        if (a[m] == x) {
7          p = m;
         } else {
8          if (a[m] < x) {
9            s = m + 1;
           } else {
10           e = m - 1;
           }
         }
       }
11     return p;
     }

     public static void main(String[] args) {
12     int[] array = {1, 2, 3, 5, 8, 13, 21, 34, 55, 89};
       int v;

13     v = mystify(array, 13);
14     System.out.println("v: " + v);
     }
   }
```

Trace this program by writing down a list of the statements which will be executed, in the same order as that in which the statements will be executed. Each statement must be included on your list exactly as many times as it is executed. Use the numbers to the left of each statement to indicate which statement is executed.

To do this correctly, write your answer using a 2-column format. For each row, the first column contains the statement number of the statement being executed, and the second column contains the following information, according to the type of the statement:

- If the statement is an assignment statement: the variable whose value is being changed, and its new value.
- If the statement is a control flow construct (`if` or `while`): the value to which the boolean expression controlling it evaluates.
- If the statement is a call to the `print()` or `println()` methods: the `String` which is displayed on the screen as a result of the call.

- If the statement is a call to the `trace()` method: the word "CALL", followed by the name of the method, as well as each of the formal parameters this method accepts and the actual values that are passed to each parameter for this call.
- If the statement is a `return` statement: the word "RETURN", followed by the value to which the expression following the statement evaluates.

As an example, here are the first five lines of the answer (you do not need to write these lines in your solution):

```
12     array ← {1, 2, 3, 5, 8, 13, 21, 34, 55, 89}
13     CALL trace(a ← {1, 2, 3, 5, 8, 13, 21, 34, 55, 89},
          x ← 13)
 1     s ← 0
 2     e ← 9
 3     p ← -1
```

CONTINUE THE TRACE UNTIL THE PROGRAM TERMINATES:

[15] 6. Consider the following program, saved in a file called `PalindromeChecker.java`:

```
1  import java.util.Scanner;
2
3  public class Palindrome {
4    public static boolean isPalindromic(int[] myNumbers) {
5      boolean palindrome;
6      int i;
7
8      if (myNumbers == null) {
9        palindrome = false;
10     } else {
11       i = 0;
12       palindrome = true;
13       while (i < myNumbers.length / 2 && palindrome = true) {
14         if (myNumbers[i] != myNumbers[myNumbers.length - i]) {
15           palindrome = false;
16         } else {
17           i = i + 1;
18         }
19       }
20       return palindrome;
21     }
22   }
23
24   public static void main(String[] args) {
25     Scanner reader = new Scanner(System.in);
26     boolean palindrome = false;
27     int[] array = null;
28     int size = 0;
29
30     System.out.print("Enter the number of values to be stored " +
31       "in the array: ");
32     size = reader.nextInt();
33
34     for (int i = 0; i < array.length; i++) {
35       System.out.print("Enter the value at position " + i + ": ");
36       array[i] = reader.nextInt();
37     }
38
39     if (isPalindromic(array)) {
40       System.out.println("This number sequence is palindromic.");
41     } else {
42       System.out.println("This number sequence is not palindromic.");
43     }
44   }
45 }
```

The above program is designed to determine whether a number sequence stored in an array is palindromic, and display the result. A number sequence is palindromic if it reads the same way forwards and backwards; for example, the sequence stored in the array {`1, 3, 4, 3, 1`} is palindromic.

However, there are **5** errors in the above program. Find all the errors and list them. For each error you list, you **MUST** include the number of the line where the error occurs, the type of error (syntactic or

semantic) and a description of the error. Do not list more than 5 errors, as you will be penalized for every "error" in excess of 5 that you list.

Note that the line numbers to the left of the above program are included solely to make it easier for you to list the line numbers where errors occur; they are not part of the actual program.

LIST THE ERRORS YOU FIND IN THE PROGRAM HERE:

Section 3 - Programming Questions

7. In this question, you will write parts of a program which reads lines of text from the keyboard and inserts them in an array until the user signals that he/she does not want to enter more lines. At this point, the program displays the contents of the array in lexicographical order and terminates.

   The complete program consists of two methods:

   - `insert()`
   - `main()`

   The `insert()` method is declared in a class called `ArrayUtils`, and the `main()` method is declared in a class called `DictionaryCompiler`.

   This question has two parts; in each of these parts, you will write one of the above methods. Note that neither of the parts depends on the successful completion of the other part.

[30] **Part 1**:

Write a class called `ArrayUtils`. This class defines a single `public` and `static` method called `insert()`, which takes as parameters an array of `String`s as well as a single `String`, and returns an array of `String`s. The parameter array contains a number of `String`s exactly equal to its length; in other words, it does not contain any `null` elements. Furthermore, the `String`s it contains appear in lexicographical order.

The method returns a **NEW** array whose length is equal to the length of the parameter array, plus one. The contents of the new array consist of the elements of the parameter array as well as the parameter `String`. All of these `String`s also appear in lexicographical order in the returned array.

For example, suppose the length of the parameter array is `4`, and that the array contains the `String`s `"Alfa"`, `"Bravo"`, `"Delta"`, and `"Echo"`, in this order; also suppose that the parameter `String` is `"Charlie"`. In this case, the array returned by the method will have length `5` and will contain the `String`s `"Alfa"`, `"Bravo"`, `"Charlie"`, `"Delta"`, and `"Echo"`, in this order.

You **MAY** assume that parameter values satisfy all of the following preconditions:

- The array parameter is not a `null` reference.
- The single `String` parameter is not a `null` reference.
- The number of `String`s stored in the parameter array is exactly equal to the length of the array; in other words, the array does not contain any `null` elements.
- The `String`s in the parameter array appear in lexicographical order.

In other words, your method does not have to handle cases where one or more of the above conditions are not satisfied. On the other hand, your method **MUST** handle the case where the length of the parameter array is `0`; in such a case, simply return an array of length `1` containing the parameter `String`.

Hint: Use the fact that the `String`s in the parameter array are already sorted in lexicographical order.

WRITE YOUR `ArrayUtils` CLASS IN THE SPACE BELOW:

[20] **Part 2**:

Write a class called `DictionaryCompiler`, which contains only a `main()` method that does the following:

- Asks the user to enter a `String`, and reads the `String` from the keyboard.
- Inserts the `String` entered by the user in an array containing all `String`s previously entered by the user. The `String` is inserted in such a way that after the insertion has been performed, all the `String`s in the array appear in lexicographical order.
- Repeats the previous two steps until the user enters the empty `String` (`""`). When the user does enter the empty `String`, the program displays all the `String`s stored in the array (in the order in which they appear) in the array, and terminates.

You **MUST** call the `insert()` method you were asked to write in the previous part to insert each `String` entered by the user in the array containing all `String`s entered by the user up to that point. You **MAY** assume that the `insert()` method has been implemented correctly, even if you did not successfully complete the previous part. Remember that the `insert()` method is declared in the `ArrayUtils` class, **NOT** the `DictionaryCompiler` class.

Sample session:

```
Enter a line (empty line to stop): Echo
Enter a line (empty line to stop): Bravo
Enter a line (empty line to stop): Charlie
Enter a line (empty line to stop): Alfa
Enter a line (empty line to stop): Delta
Enter a line (empty line to stop):
The lines you entered:
* Alfa
* Bravo
* Charlie
* Delta
* Echo
```

WRITE YOUR `DictionaryCompiler` CLASS IN THE SPACE BELOW:

YOUR `DictionaryCompiler` 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

- `String` (package `java.lang`) Methods:
  - `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)`: Compares this `String` to `anotherString` lexicographically.
  - `public int compareToIgnoreCase(String anotherString)`: Compares, ignoring case considerations, this `String` to `anotherString` lexicographically.
  - `public char[] toCharArray()`: Converts this `String` to a new character array.
- `Scanner` (package `java.util`) Methods:
  - `public Scanner(InputStream source)`: Constructs a new `Scanner` that produces values scanned from the specified input stream.
  - `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 int nextLong()`: Scans the next token of the input as an `long`.
- `PrintStream` (package `java.io`) Methods:
  - `public void print(boolean b)`: Prints `boolean` value `b`.
  - `public void print(char c)`: Prints `char` value `c`.
  - `public void print(char[] s)`: Prints the array of `char` `s`.
  - `public void print(double d)`: Prints `double` value `d`.
  - `public void print(int i)`: Prints `int` value `i`.
  - `public void print(Object o)`: Prints `Object` `o`.
  - `public void print(String s)`: Prints `String` `s`.
  - `public void println()`: Terminates the current line by writing the line separator string.
  - `public void println(boolean b)`: Prints `boolean` value `b` and then terminates the line.
  - `public void println(char c)`: Prints `char` value `c` and then terminates the line.
  - `public void println(char[] s)`: Prints array of `char` `s` and then terminates the line.
  - `public void println(double d)`: Prints `double` value `d` and then terminates the line.
  - `public void println(int i)`: Prints `int` value `i` and then terminates the line.
  - `public void println(Object o)`: Prints `Object` `o` and then terminates the line.
  - `public void println(String s)`: Prints `String` `s` and then terminates the line.
- `Math` (package `java.lang`) Methods:
  - `public static double pow(double a, double b)`: Returns the value of `a` raised to the power of `b`.
  - `public static double sqrt(double a)`: Returns the correctly rounded positive square root of `double` value `a`.
  - `public static double random()`: Returns a `double` value with a positive sign, greater than or equal to `0.0` and less than `1.0`.
  - `public static double sin(double a)`: Returns the trigonometric sine of angle `a`, where `a` is in radians.
  - `public static double cos(double a)`: Returns the trigonometric cosine of angle `a`, where `a` is in radians.
  - `public static double tan(double a)`: Returns the trigonometric tangent of angle `a`, where `a` is in radians.
  - `public static double toDegrees(double angrad)`: Converts angle `angrad` measured in radians to an approximately equivalent angle measured in degrees.
  - `public static double toRadians(double angdeg)`: Converts angle `angdeg` measured in degrees to an approximately equivalent angle measured in radians.
  - `public static double exp(double a)`: Returns Euler's number e raised to the power of `double` value `a`.

  - `public static double log(double a)`: Returns the natural logarithm (base e) of `double` value `a`.
  - `public static double log10(double a)`: Returns the base 10 logarithm of `double` value `a`.

- `Character` (package `java.lang`) Methods:

  - `public static boolean isDigit(char ch)`: Determines if character `ch` is a digit.
  - `public static int digit(char ch, int radix)`: Returns the numeric value of character `ch` in the radix `radix`, `-1` if `ch` does not represent a digit.
  - `public static char forDigit(int digit, int radix)`: Returns the character representation of `digit` in the radix `radix`.
  - `public static boolean isLetter(char ch)`: Determines if character `ch` is a letter.
  - `public static boolean isLowerCase(char ch)`: Determines if character `ch` is a lowercase character.
  - `public static boolean isUpperCase(char ch)`: Determines if character `ch` is an uppercase character.
  - `public static boolean isWhitespace(char ch)`: Determines if character `ch` is white space according to Java.
  - `public static char toLowerCase(char ch)`: Converts character `ch` to lowercase.
  - `public static char toUpperCase(char ch)`: Converts character `ch` to uppercase.