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

McGill ID: ______________________________ **Section**: ______

Faculty of Science
COMP-202A - Introduction to Computing I (Fall 2009) - All Sections
Midterm Examination

Tuesday, November 3, 2009
18:30–20:30

Examiners: Mathieu Petitpas [Sections 1 and 3]
Kamal Zellag [Section 2]

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 18 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 **20** pages including this cover page, and is printed on both sides of the paper. Pages 19-20 contain information about useful classes and methods.

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

5	**6**	**Subtotal**
/15	**/15**	**/30**

7	**8**	**Subtotal**
/15	**/35**	**/50**

Total
/100

Section 1 - Short Questions

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

(a) An `if` statement and a `while` statement

(b) The `&&` operator and the `||` operator

(c) Formal parameters and actual parameters

[6] 2. What will be displayed to the screen after each of the following code fragments is executed? For full marks, describe **BRIEFLY** with a few words and a value what happens at intermediate steps.

(a)
```
int a = 5;
int b = a % 2;
double c = a / 10;
double d = b + 2.5;
double e = (int)d * 3;
System.out.println("c == " + c + ", e == " + e);
```

(b)
```
int[] a1 = {5, 10, 20, 0};
int[] a2 = a1;
a2[0] = 3;
boolean b = a1[1] + a1[2] == a1[0] + a1[3];
System.out.println(a1[1] + a1[2] + " == " + a1[0] + a1[3] + ": " +
  b);
```

[4] 3. The following program prints a shape to the screen using the character `*`. Describe what shape it is and what its dimensions are. You can also draw the program's exact output for full marks.

```
public class MysteryShape {
  public static void main(String[] args) {
    final int SIZE = 15;
    final int HALF_SIZE = SIZE / 2;
    int side;

    for (int i = 0; i < SIZE; i++) {
      if (i <= HALF_SIZE) {
        side = i;
      } else {
        side = SIZE - (i + 1);
      }

      for (int j = 0; j < SIZE; j++) {
        if (HALF_SIZE - side <= j && j <= HALF_SIZE + side) {
          System.out.print("*");
        } else {
          System.out.print(" ");
        }
      }
      System.out.println();
    }
  }
}
```

[4] 4. Consider the `swap()` method, declared in the following `Swapper` class:

```
public class Swapper {
  public static void swap(int a, int b) {
    int temp;

    temp = a;
    a = b;
    b = temp;
  }

  public static void main(String[] args) {
    int v1 = 3;
    int v2 = 4;

    System.out.println("Swapping...");
    swap(v1, v2);
    System.out.println("v1's new value: " + v1);
    System.out.println("v2's new value: " + v2);
  }
}
```

The above `swap()` method above is intended to swap the values of the actual parameters passed to the method. For example, suppose the `swap()` method works as intended, and the `main()` method is executed; after the call to the `swap()` method, the value stored in variable `v1` should be `4` (that is, the value that was stored in variable `v2` before the call to the `swap()` method), while the value stored in variable `v2` should be `3` (that is, the value that was stored in variable `v1` before the call to the `swap()` method).

Does the above `swap()` method work as it is intended to? That is, will it effectively swap the values of the actual parameters? **BRIEFLY** explain your answer.

Total marks for Section 1: **20**

Section 2 - Long Questions

[15] 5. Consider the following program:

```
      public class TraceMe {
        public static int trace(int[] a) {
          int m, c, mc, i;

1         m = a[0];
2         c = 1;
3         mc = 1;

4         i = 1;
5         while (i < a.length) {
6           if (a[i] == a[i - 1]) {
7             c = c + 1;
8             if (c > mc) {
9               m = a[i];
10              mc = c;
              }
            } else {
11              c = 1;
            }
12          i = i + 1;
          }
13        return m;
        }

        public static void main(String[] args) {
14        int[] array = {2, 3, 3};
          int v;

15        v = trace(array);
16        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()` method: 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):

```
14     array ← {2, 3, 3}
15     CALL trace(a ← {2, 3, 3})
 1     m ← 2
 2     c ← 1
 3     mc ← 1
```

CONTINUE THE TRACE UNTIL THE PROGRAM TERMINATES:

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

```
1  import java.util.Scanner;
2
3  public class Minimum {
4    public static int minimum(int[] a) {
5      int minSoFar;
6
7      minSoFar = a[0];
8      for (int i = 1; i <= a.length; i++) {
9        if (a[i] > minSoFar) {
10         minSoFar = a[i];
11       }
12     }
13     return minsoFar;
14   }
15
16   public static void main(String[] foo) {
17     Scanner keyboard = new Scanner(System.in);
18     int[] array;
19     int size;
20     int min;
21
22     System.out.print("Enter the number of values to be stored " +
23       "in the array: ");
24     size = keyboard.nextInt();
25
26     array = new int(size);
27     for (int i = 0; i < array.length; i++) {
28       System.out.print("Enter the value at position " + i + ": ");
29       array[i] = keyboard.nextInt()
30     }
31
32     min = minimum(array);
33     System.out.println("The minimum value in the array is: " + min);
34   }
35 }
```

The above program is designed to determine the minimum value in an array whose values are entered by the user (the number of values in the array is intended to be equal to the size of the array, which is also entered by the user), and display this minimum value. 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 help you 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:

Total marks for Section 2: **30**

Section 3 - Programming Questions

[15] 7. Suppose that on the first day of each month, you save a fixed amount of money a_m in a savings account with annual interest rate r_y. You can calculate the amount of money in your savings account at the beginning of a given month using the following formula:

$$a_i = a_{i-1} \cdot \left(1 + \frac{r_m}{100}\right) + a_m$$

where

- a_i is the amount of money in the account at the beginning of the current month
- a_{i-1} is the amount of money in the account at the beginning of the previous month
- r_m is the monthly interest rate; it is equal to $r_y/12$, where r_y is the yearly interest rate
- a_m is the amount being deposited in the savings account each month

Note that the initial amount in the account, denoted a_1, is equal to the amount being deposited in the savings account each month (that is, a_m).

For example, suppose that you decide to save $100.00 each month into a savings account with a yearly interest rate of 6% (the monthly interest rate is therefore 0.5%).

- At the beginning of the first month, you deposit $100.00 in the account.
- At the beginning of the second month, there will be $100.00 · (1 + 0.005) + $100.00 = $200.50 in the account.
- At the beginning of the third month, there will be $200.50 · (1 + 0.005) + $100.00 = $301.5025 in the account.

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

- Ask the user to enter a **yearly** interest rate, an amount to be deposited in the saving accounts each month, as well as a number of months, and read all these values from the keyboard. The yearly interest rate and the amount being deposited in the savings account are real numbers, while the number of months is an integer. Note that the user specifies the yearly interest rate as a percent; in other words, to specify a yearly interest rate of 6%, the user will enter `6.0` or simply `6`, **NOT** `0.06`.
- Calculate the amount in the savings account after the number of months entered by the user has passed.
- Display this value, with an appropriate explanatory message. This value does not have to be formatted in any particular fashion; in particular, the number of significant digits displayed does not matter.

Sample session:

```
Enter the yearly interest rate: 6.0
Enter the amount to be deposited each month: 100.0
Enter the number of months: 3
After 3 months, there will be $301.5025 in the savings account
```

Your program **MAY** assume that the values entered by the user are of the correct type and that they are in the proper range; in other words, your program does not have to handle cases where a value entered by the user is invalid in any way (such as a negative interest rate or number of months).

WRITE YOUR `SavingsCalculator` CLASS IN THE SPACE BELOW:

8. In this question, you will write parts of a program which reads lines of text from the keyboard, encrypts these lines of text, and displays the encrypted lines.

   The complete program consists of three methods:

   - `getPosition()`
   - `encrypt()`
   - `main()`

   The `getPosition()` and `encrypt()` method are declared in a class called `Cipher`, and the `main()` method is declared in a class called `CipherTest`.

   This question has three parts; in each of these parts, you will write one of the above three methods. Note that none of the parts depend on the successful completion of any of the other parts.

[5] **Part 1**:

Write a `public` and `static` called `getPosition()`, which takes as its only parameter a `char` representing an upper-case letter, and returns an `int` representing the position of this letter in the English alphabet. Note that for the purposes of this question, the first letter of the alphabet has position `0`, and the positions of the other letters are adjusted accordingly; that is, the letter `'A'` has position `0`, the letter `'B'` has position `1`, and so on.

You **MAY** assume that the `char` value this method accepts as parameter represents an upper-case letter; in other words, your method does not have to handle the case where the `char` value this method accepts as parameter does **NOT** represent an upper-case letter.

Hint: Use the properties of the character codes for each letter to avoid writing 26 `if` statements. This method is actually much, much shorter than the space allocated for it suggests.

WRITE YOUR `getPosition()` METHOD IN THE SPACE BELOW:

YOUR `getPosition()` METHOD CONTINUED:

[15] **Part 2**:

A monoalphabetic substitution cipher is a simple but insecure scheme used to encrypt messages. In such a cipher, for each possible symbol in the original message (called the *plaintext*), each occurrence of this symbol is replaced by another symbol in the encrypted message (called the *ciphertext*). A special value called the *key* determines which ciphertext symbol replaces each possible plaintext symbol.

Write a `public` and `static` method called `encrypt()`, which takes as parameters two arrays of `char`, and returns an array of `char`. The first array this method takes as parameter represents the plaintext, and the second array this method takes as parameter represents the key. For each character in the plaintext, this method will generate the corresponding ciphertext character based on the key and store this ciphertext character in a new array of `char` representing the ciphertext. The length of the ciphertext will be equal to the length of the plaintext. Once every character in the plaintext has been encrypted, the method will return the ciphertext.

This method will only encrypt characters representing letters; characters which occur in the plaintext and which are not letters will not be encrypted, and will appear as-is in the ciphertext. To determine which ciphertext letter corresponds to a plaintext letter, determine the position of the plaintext letter in the alphabet, and retrieve the letter at the resulting position in the key. For example, each occurrence of the character `'A'` in the plaintext will be replaced by the character at position `0` in the key, each occurrence of the character `'B'` in the plaintext will be replaced by the character at position `1` in the key, and so on. Lower-case letters occuring in the plaintext will be converted to uppercase before being encrypted.

The ciphertext characters will be stored in the ciphertext in the same order as the corresponding plaintext characters were stored in the plaintext. In other words, if a character is stored at position i in the plaintext, the corresponding ciphertext character will be stored at position i in the ciphertext; this is true of all characters in the plaintext.

For example, the plaintext:

```
"Hello World!"
```

when encrypted with the key

```
"BCDEFGHIJKLMNOPQRSTUVWXYZA"
```

will produce the following ciphertext:

```
"IFMMP XPSME!"
```

You **MAY** assume that the length of the array representing the key is exactly 26, and that this array contains each upper-case letter **exactly** once.

You **MUST** call the `getPosition()` method you were asked to write in the previous part to calculate the position of a letter in the alphabet. You **MAY** assume that the `getPosition()` method has been implemented correctly, even if you did not successfully complete the previous part.

Hint: Use the methods of the `Character` class to verify whether a character represents a letter, determine the case of a letter, or to convert a letter to its uppercase or lowercase equivalent. See the Summary of Java Standard Library Methods at the end of this examination for more details on the methods of the `Character` class.

WRITE YOUR `encrypt()` METHOD IN THE SPACE BELOW:

[15] **Part 3**:

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

- Ask the user to enter the plaintext.
- Ask the user to enter the key; you **MAY** assume that the key entered by the user is 26 characters long, and contains each upper-case letter exactly once.
- Encrypt the plaintext using the key, using the encryption scheme described in the previous part.
- Display the resulting ciphertext.
- Ask whether to encrypt another plaintext using another key. If the user answers `"y"`, `"Y"`, `"yes"`, `"YES"`, or any other meaningful combination of these letters such as `"YeS"`, then the program will repeat the steps listed above, otherwise it will terminate.

You **MUST** call the `encrypt()` method you were asked to write in the previous part to encrypt the plaintext entered by the user using the key entered by the user. You **MAY** assume that the `encrypt()` method has been implemented correctly, even if you did not successfully complete the previous part. Remember that the `encrypt()` method is declared in the `Cipher` class, **NOT** the `CipherTest` class.

Hints:

- Read the plaintext and key entered by the user as `String`s, and convert these `Strings` to arrays of `char` using the `toCharArray()` method. You can call this method as follows:
  ```
  a = s.toCharArray();
  ```
  where `s` is the name of the `String` variable containing the `String` you want to convert, and `a` is the name of a variable of type `char[]` in which you want to store the array resulting from the conversion.
- You can display the contents of an array of `char` by passing the array variable to the `print()` or `println()` method. For example, the statement
  ```
  System.out.println(charArray);
  ```
  where `charArray` is an array of `char` whose contents is {`'H', 'i', '!'`}, will result in the following being displayed:
  ```
  Hi!
  ```
  In other words, you do **NOT** have to display the contents of an array of `char` one character at a time using a loop.

WRITE YOUR `CipherTest` CLASS IN THE SPACE BELOW:

YOUR `CipherTest` CLASS CONTINUED:

Total marks for Section 3: $\overline{}$ **50**

Total marks: $\overline{}$ **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 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.
- `PrintStream` (package `java.io`) Methods:
  - `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.
- `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:
  - `static boolean isLetter(char ch)`: Determines if character `ch` is a letter.
  - `static boolean isLowerCase(char ch)`: Determines if character `ch` is a lowercase character.
  - `static boolean isUpperCase(char ch)`: Determines if character `ch` is an uppercase character.
  - `static char toLowerCase(char ch)`: Converts character `ch` to lowercase.
  - `static char toUpperCase(char ch)`: Converts character `ch` to uppercase.