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

McGill ID: _____ **Section**: _____

Faculty of Science
COMP-202B - Introduction to Computing I (Summer 2011) - All Sections
Midterm Examination

Tuesday, June 14, 2011
14:30-16:30

Examiners: Daniel Pomerantz

Instructions:

- **DO NOT TURN THIS PAGE UNTIL INSTRUCTED**
- This is a **closed book** examination; only a letter-sized (8.5" by 11") **crib sheet** is permitted. This crib sheet can be single or double-sided; it can be handwritten or typed. Non-electronic translation dictionaries are permitted, but instructors and invigilators reserve the right to inspect them at any time during the examination.
- Besides the above, only writing implements (pens, pencils, erasers, pencil sharpeners, etc.) are allowed. The possession of any other tools or devices is prohibited.
- Answer **all** questions **on this examination paper** and return it. **If you need additional space**, use pages 14-15 or the booklets supplied upon request and clearly indicate where each question is continued. **In order to receive full marks for a question, you must show all work** unless otherwise stated.
- This examination has **16** pages including this cover page, and is printed on both sides of the paper. On page 16, you will find information about **useful classes and methods**. You may detach this page from the examination if you wish.
- The exam is graded out of 100 points, but you can get up to 110 points.

1	2	3	4	5	Subtotal
/14	/5	/9	/4	/14	/40

6	7	8	9	10	11	Subtotal	Total
/10	/14	/10	/10	/10	/10	/60	/100

Section 1 - Expressions and Tracing Code

1. (14points) For each of the following Java expressions, write the *type* of the expression as well as the resulting value. An example is provided:

	Java Expression	Type	Value
Example 1:	`11.0 + 11.0`	double	22.0
Example 2:	`11 + 11`	int	22
Example 3:	`"2" + "2"`	String	“22”

	Java Expression	Type	Value
A	`(double)(1 / 2)`	double	0
B	`5 + 2 * 6`	17	int
C	`(double) 1 / 2`	double	.5
D	`5.0 > 3-2 || !(3 < 4)`	boolean	true
E	`"3" + " is " + 1 + 2`	String	“3 is 13”
F	`(int) (1.0 / 2.0)`	int	0
G	`(10 % 4) * 5 / 2`	int	5

2. (5points) If `x` and `y` and `z` are each booleans, write a boolean expression that will have the value `true` whenever exactly one of x,y, and z are true and false otherwise.

```
(x && !y && !z) || (!x && y && !z ) || (!x && !y && !z)
```

3. (9points) In the table below, state **how many lines** are printed to the screen by each of the following code fragments.

```
This is an
example
of what we mean by 3 lines printed to the screen.
```

If the call to `println()` is inside an **infinite** loop, write ∞. You do **not** have to show your work.

YOUR ANSWERS:

(a)	(b)	(c)
2	10	infinite

(a)

```
1  int num = 2;
2  for (int i = num; i < num * 3; i++) {
3    System.out.println("Hello.");
4    i = i + num;
5  }
```

(b)

```
1  for(int i = 0; i < 5; i++)
2    for(int j = 0; j < i; j++)
3      System.out.println("*");
```

(c)

```
1  double i = 1;
2  int x;
3  while(i < 2){
4    System.out.println(i);
5     x = i;
6     i = x / 2 + 1;
7  }
```

4. (4points) In the following, you will be presented with several variable declarations. For each variable declared, you should decide whether the value stored in the variable is an address or not.

   If the variable stores an address (i.e. a reference to data), then circle YES. If the variable does not, circle NO.

   Make sure it is clear which choice you are circling. Unclear answers will not be given credit.

	Stores an address?
`int x;`	NO
`double y;`	NO
`int[] foo;`	YES
`String bar;`	YES
`String[]`	YES

5. (14points) For the next question, consider the class `ReferenceTest` defined below. In the main method, an array is created in the first line. You will be asked several questions about the array referenced by coolArray. (You should look at the questions first before reading every line of code).

   Here is the class:

```
1  public class ReferenceTest
2  {
3    public static void main(String[] args)
4    {
5      int[] coolArray = {1, 2, 3, 4, 5};
6      int[] secondArray = makeCopy(coolArray);
7      int[] thirdArray = coolArray;
8      methodOne(coolArray);
9      System.out.println(methodTwo(coolArray));
10     methodThree(secondArray);
11     thirdArray = new int[10];
12     String s = methodFour(thirdArray);
13     methodFive(coolArray);
14     coolArray = methodSix();
15   }
16
17   public static int[] makeCopy(int[] original)
18   {
19     int[] newArray = new int[original.length];
20     for (int i=0; i < original.length; i++)
21     {
22       newArray[i] = original[i];
23     }
24
25     return newArray;
26   }
27
28   public static void methodOne(int[] original)
29   {
30     original[0] = 2;
31   }
32
33   public static int methodTwo(int[] original)
34   {
35     original = new int[3];
36     original[1] = 4;
37     return original[1];
38   }
39
40   public static double methodThree(int[] original)
41   {
42     original[2] = 10;
43     return 0.0;
44   }
45
46   public static String methodFour(int[] original)
47   {
48     for (int i=0; i < original.length; i++)
49     {
50       original[i] = 0;
51     }
52
53     return "";
54   }
55
56   public static void methodFive(int[] original)
57   {
58     int[] second = original;
59     second[0] = 100;
60   }
61
62   public static int[] methodSix()
```

```
63    {
64      int[] returnArray = new int[2];
65      returnArray[0] = 1;
66      returnArray[2] = 2;
67      return returnArray;
68    }
69  }
```

Remember that the variable `coolArray` stores the address of an array. For each of the following, write the values stored in the array at that address *after* the completion of the method call. These all refer to the calls in the main method above.

Note that if you make a mistake on an early part of this question, the remaining questions will be graded as if the previous part had been right. In other words, if you say the values of the array are {1,2,3,4,5} but it should be {5,2,3,4,5} and at the next step the first value is increased by 1, then you will get full marks for writing {2,2,3,4,5} and no marks for {6,2,3,4,5}

	Contents of array referred to by coolArray
`makeCopy(coolArray);`	1,2,3,4,5
`methodOne(coolArray);`	1,2,3,4,5
`methodTwo(thirdArray);`	2,2,3,4,5
`methodThree(secondArray);`	2,2,3,4,5
`methodFour(thirdArray)`	2,2,3,4,5
`methodFive(coolArray)`	100,2,3,4,5
`methodSix()`	1,2

Programming Questions

Answer the following questions.
Even if you are short of time, do not leave any questions blank. You will get points for things such as class headers, method headers, etc if you write them. But if you leave the question blank you will get a 0 for the question.
If a question asks you to write a method and you are unable to do so successfully, *you still can and should call the method as if it does what is specified*
If you need more space, you may write on the extra pages at the end or in an exam booklet, but make sure it is clear where everything fits together.

Array manipulation

6. (10points) Write a method `find` that takes as input an `int[]` that is sorted in ascending order (smallest value at the front) and an `int` and returns a boolean. The method should return true if the array contains the int and false otherwise.

```
public static boolean insert(int[] array, int target) {
for (int i=0; i < array.length; i++) {
if (array[i] == target) {
return true;
}
}

return false;
}
```

7. (14points) Write a method `insert` which takes as input an `int[]` that is sorted in ascending order and an `int`. The method should return a new array which consists of the original array plus the new int inserted into the correct location of the array so that the array remains sorted.

   For example, if the old array has the values {-1,4,8,9} and the new int is 5, your method should return the array {-1,4,5,8,9}

   Note: You may *NOT* assume that the original array is non-empty, but you may assume it is not null.

   If the value is already in the array, then you should insert it a second time. For example, if the old array has values {1,2,5} and the new int is 5, then the result should be {1,2,5,5}

```
public static int[] insert(int[] array, int newValue) {
if (array.length == 0) {
int[] newArray = new int[1];
newArray[0] = newValue;
return newArray;
}

int i; //declared outside the loop so it stays in scope
for (i=0; i < array.length; i++)
{
if (array[i] > newValue) {
break; //we should insert at location i
}
}

int arrayWithNewValue = new int[array.length + 1];

for (int j=0; j < i; j++ ) {
arrayWithNewValue[j] = array[j];
}

arrayWithNewValue[i] = newValue;

for (int j = i; j < array.length; j++) {
arrayWithNewValue[j+1] = array[j];
}

return arrayWithNewValue;
}
```

Approximating sin(x) better

In the following methods, you may not use any method defined in the Math library
Remember that one way to approximate the mathematical function $sin(x)$ is via the Taylor series expansion:

$$sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + ...$$

In this question, you will write a series of methods to approximate this. You are not allowed to use any of the functions defined in the math library for this question.

8. (10points) Write a method `power` which takes as input 2 non-negative numbers `x` and `y` and returns the result of x^y as an int. Remember that whenever x is equal to 0, $x^y = 0$ regardless of the value of y. You may assume that neither x nor y is negative.

```
//note: returning double or int was okay for this question for full marks
public static double power(double x, int y) {
     if (y == 0) {
      return 1;
     }

     double returned = 1;

     for (int i=0; i < y; i++) {
      returned *= x;
     }

     return returned;
}
```

9. (10points) Write a method `factorial` which takes as input 1 non-negative `int n` and returns the result of $n!$ as an int. Remember that:

$$0! = 0$$

$$1! = 1$$

$$2! = 2 * 1$$

$$3! = 3 * 2 * 1$$

$$...$$

$$n! = n * (n-1) * (n-2) * ... * 3 * 2 * 1$$

You may assume that `n` is not negative.

```
public static int factorial(int n) {
    int accumulation = 1;
    for (int i=2; i <= n; i++) {
        accumulation = accumulation * i;
    }

    return accumulation;
```

10. (10points) Write a method `calculateTerm` which takes as input an `int n` and a double `x` and outputs the n^{th} *non-zero* term of the sequence using the following formula:

$$calculateTerm(n, x) = \frac{-1^n}{(2n+1)!}x^{2n+1}$$

Note that this means the 0th term will be x, the 1st term will be $-x^3/6$, the 2nd term will be $x^5/120$, etc

Remember that you can use the methods `factorial` and `power` and assume that they work correctly regardless of if they actually do.

```
public static double calculateTerm(int n, double x) {
    if (n \% 2 == 1) {
         //you had to cast if you made the method power return an int in the previous
         //otherwise you have an integer division issue
     return -1 * power(x, 2*n + 1) / factorial(2*n+1);
    }

    else {
        return power(x, 2*n+1) / factorial(2*n+1);
    }
}
```

11. (10points) Write a method `approximateSinBetter` which takes as input a `double x` and approximates the value of `sin(x)` by adding the first 10 non-zero terms of sin(x).

```
public static double approximateSinBetter(double x) {
double sum = 0;

for (int i=1; i <= 10; i++) {
sum += calculateTerm(i, x)
}

return sum;
}
```

USE THIS PAGE IF YOU NEED ADDITIONAL SPACE.

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

USE THIS PAGE IF YOU NEED EVEN MORE 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; returns a negative value if this `String` occurs before `anotherString`, a positive value if this `String` occurs after `anotherString`, and `0` if both `String`s are equal.
  - `public int compareToIgnoreCase(String anotherString)`: Compares, ignoring case considerations, this `String` to `anotherString` lexicographically; returns a negative value if this `String` occurs before `anotherString`, a positive value if this `String` occurs after `anotherString`, and `0` if both `String`s are equal.
  - `public char[] toCharArray()`: Converts this `String` to a new character array.
- `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`.