Student Name: _____ **Student Number:** _____

Section: _____

Faculty of Science
Final Exam

COMP-202A - Introduction to Computing I (Fall 2007)

Friday, December 14, 2007
14:00–17:00

Examiners: Mathieu Petitpas [Section 1]
Prof. Clark Verbrugge [Section 2]
Ladan Mahabadi [Section 3]
Associate Examiner: Prof. Laurie Hendren

Instructions:

- **DO NOT TURN THIS PAGE UNTIL INSTRUCTED**
- This is a **closed book** exam.
- **Calculators** are allowed (though you should not need one).
- **Translation dictionaries** are allowed (human languages only).
- You **must return the exam paper.**
- **Answer all questions on the exam paper;** if you need additional space, use the last page or the booklets supplied and clearly indicate where each question is continued. **In order to receive full marks, you must show all work.**
- This exam has **22** pages including this cover page. Note that for your convenience the last page has a generic list of useful classes and methods that may be helpful in programming questions.

1	2	3	4	5	6	7	8	9	SubTotal
/2	/2	/2	/3	/4	/1	/3	/2	/3	/22

10	11	12	13	14	15	16	17	SubTotal
/4	/4	/6	/5	/5	/5	/4	/5	/38

18	19	20	SubTotal
/12	/15	/13	/40

Total
/100

Section 1 - Short Answer Questions

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. **[2]**

   (a) Primitive types and reference types, with regards to how they are stored in memory.

   (b) Classes and objects.

2. The following question deals with exceptions. **[2]**

   (a) In one or two sentences, explain the difference between checked and unchecked exceptions. BE BRIEF; overly long answers will be grounds for mark deductions.

   (b) List one unchecked exception type and one checked exception type defined in the Java platform.

3. There is a problem in this code fragment; explain what it is and how to fix it. **[2]**

```
int i = 0, sum = 1;
do {
   boolean more;
   i++;
   sum *= i;
   if (i > 6)
      more = false;
   else
      more = true;
} while(more);
System.out.println(sum);
```

4. Consider the following array declaration: **[3]**

```
int[] anArray = new int[n];
```

Suppose that `n` has already been assigned some non-negative input value and the programmer would like to apply the method `f(int)` to each element of the array. Give two ways of doing so, one based on a `for` loop, and one based on a `for-each` loop.

5. Consider the following program: **[4]**

```
public class TrickOverload {
    public int foo(int a, int b) {
        System.out.println("add(int, int)");
        return a + b;
    }

    public double foo(double a, double b) {
        System.out.println("add(double, double)");
        return a * b;
    }

    public double foo(double a, int b) {
        System.out.println("add(double, int)");
        return a - b;
    }

    public static void main(String[] args) {
        TrickOverload load;
        int i1;
        int i2;
        double d;

        load = new TrickOverload();
        i1 = 2;
        i2 = 3;
        d = 4;

        System.out.println("The result is: " + load.foo(i1 * i2, d));
    }
}
```

The program compiles without error and terminates normally when it is executed; what does it display?

6. In one or two sentences, explain the purpose of the base case in a recursive method. BE BRIEF; overly long answers will be grounds for mark deductions. **[1]**

7. The following code does not compile for some reason. Explain what the problem is and show two different ways of modifying the code to fix it. **[3]**

```
public readSerialNumber(java.io.File someFile) {
   java.util.Scanner scan = new java.util.Scanner(someFile);
   String internalCode = "";
   for (int i = 0; i < 3; i++) {
      internalCode += scan.next();
   }
   return internalCode;
}
```

8. List two differences between constructors and other methods of a class. **[2]**

9. A student writes the following code, intending it to display every other character in a given `String` starting from the first `char`. Sadly, while it compiles, it does not actually work. Explain what the problem(s) is (are) and rewrite the loop to work as intended. **[3]**

```
String input = "CAAOAAMAAAP";
for (int i = 0; i < input.length(); i++)
{
    System.out.print(input.charAt(i));
    input = input.substring(i+2);
}
System.out.println();
```

Section 2 - Long Answer Questions

10. Consider the following program: **[4]**

```
public class Scope {
    private double d;

    public Scope(double d) {
        this.d = d;
    }

    public double foo(int i) {
        if (i < 0) {
            double d = i * i * i;
            System.out.println("d (foo) == " + d);
        } else {
            d = i * i;
            System.out.println("d (foo) == " + d);
        }
        return d;
    }

    public static void main(String[] args) {
        Scope s;
        double d;

        s = new Scope(2.0);
        d = 5.0;

        System.out.println("foo(-2) == " + s.foo(-2));
        System.out.println("d (main) == " + d);
    }
}
```

What will be displayed to the screen when this program is executed?

11. The following program will not compile; the compiler apparently complains about some sort of "typing" or "casting" problems. **[4]**

```
import java.util.*;
public class CigarettePack {
   ArrayList cigs;

   public CigarettePack() {
      cigs = new ArrayList();
   }
   public void addCigarette(Cigarette c) {
      cigs.add(c);
   }
   public Cigarette getCigarette() {
      Cigarette c;
      if (cigs.size() > 0)
         c = cigs.remove(0);
      else
         c = null;
      return c;
   }
   public String toString() {
      String result = "pack of [";
      for (int i=0; i < cigs.size(); i++) {
         Cigarette c = cigs.get(i);
         result += ((i==0) ? "" : ",") + c.toString();
      }
      return result + "]";
   }
}
```

(a) Describe how to fix the program so it works without resorting to generics.

(b) Describe how to fix the program so it uses generics and does not require or contain any casting.

12. On a cold, freshly-snowed winter day, Professor Merlin is walking home. As he's approaching his house, he estimates that he is n steps away from his door steps and tries to calculate what his last (n^{th}) step would be (left or right) if he were to start with his right foot. Write a `main()` method that prompts the user for n and then proceeds with listing `Right, Left, Right, Left, ...` for n steps and then displays that last step. For instance, if $n = 4$, then your `main()` method should display: **[6]**

```
Right, Left, Right, Left
Left foot takes the last step!
```

13. Consider the following program. Which of the lines A through I below have correct syntax and will compile? **[5]**

```
public class Foo {
   private int x;
   private static int y;

   public Foo(int z) { x = z; }

   public static int m1(int x) {
      Foo f = new Foo(1);
      y = y+x;
      System.out.println(m2());     /* Line A */
      System.out.println(f.m2());   /* Line B */
      System.out.println(Foo.m2()); /* Line C */
      return y;
   }
   public int m2() {
      Foo f = new Foo(2);
      x = m3(0);                    /* Line D */
      x += f.m3(3);                 /* Line E */
      x += Foo.m3(1729);            /* Line F */
      return y+x;
   }
   public int m3(int y) {
      return y+x;
   }
   public static void main(String[] args) {
      Foo f = new Foo(0);
      m1(0);                        /* Line G */
      f.m1(3);                      /* Line H */
      Foo.m1(1729);                 /* Line I */
   }
}
```

14. Write the body of a method with the following header: **[5]**

```
public static boolean alternates(int[] array)
```

This method returns `true` if the values in `array` alternate between negative and non-negative numbers, and `false` otherwise. That is, a negative number must occur immediately before and immediately after each positive number in `array`, and a positive number must occur immediately before and immediately after each number in the `array`. The first value in `array` may be either a positive or negative value. You should consider that the values in an array of size `1` or less alternate, and you should consider `0` to be a positive number.

Examples:

- if `array` is {`-4, 7, -9, 2, -3, 10, -3`}, the method returns `true`.
- if `array` is {`6, -2, -9, 4, -11`}, the method returns `false` because the value which occurs immediately after `-2` is not positive.

15. What is the control flow of the following code? Indicate the order of statements from the statement numbers given; if statements are not in sequence explain each change. **[5]**

```
/*    */   try {
/*    */      try {
/*  1 */         int i = 0,j=7;
/*  2 */         java.util.Random random = null;
/*  3 */         int[] ids = new int[10];
/*  4 */         System.out.println("Starting");
/*  5 */         System.out.println(random.nextInt(j/i+ids[10]));
/*  6 */         System.out.println("Done..?");
/*    */      } catch(ArithmeticException ae) {
/*  7 */         System.out.println("That can't happen!");
/*    */      } catch(ArrayIndexOutOfBoundsException aie2) {
/*  8 */         System.out.println("How could this be?");
/*    */      } finally {
/*  9 */         System.out.println("Are we done here?");
/*    */      }
/* 10 */      System.out.println("I think it's done, maybe");
/*    */   } catch(ArrayIndexOutOfBoundsException aie1) {
/* 11 */      System.out.println("This is impossible surely!");
/*    */   } catch(NullPointerException npe) {
/* 12 */      System.out.println("I, make a mistake?");
/*    */   } finally {
/* 13 */      System.out.println("Are we done now?");
/*    */   }
/* 14 */   System.out.println("Are we printed out?");
```

16. Professor Fibo would like to simulate the shape of snail shells. To do so, he draws two small squares of size 1 next to each other. On top of both of these, he draws a square whose side is of length 2 $(= 1 + 1)$. Then, he draws a new square that is adjacent to one of the unit squares and the square of side 2, giving it a side length of 3. He continues adding squares around the perimeter such that each new square has a side which is as long as the sum of the previous two squares' sides. Finally, he draws a quarter of a circle in each square as shown in Figure 1 to form the snail shell. **[4]**

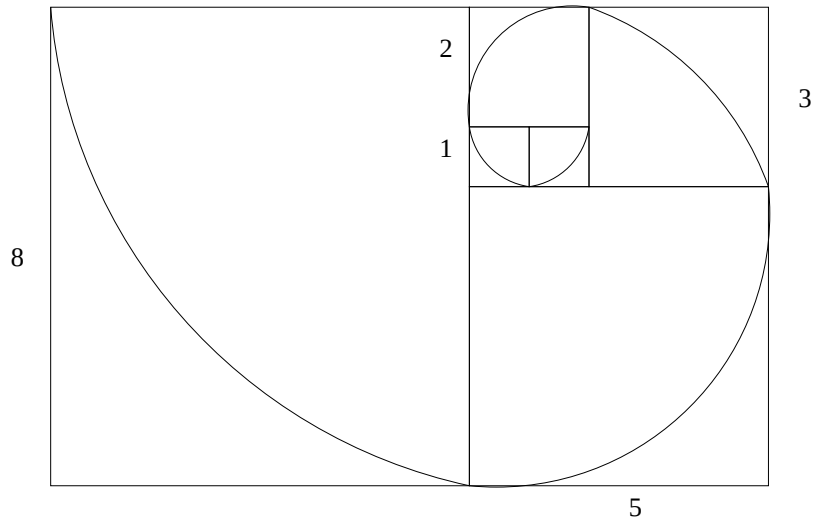

Figure 1: Snail shell

The sides of the generated squares form the following sequence: $1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$. More generally, note that each number in the sequence is generated by adding the two previous values in the sequence. Namely:

$F(0) = 1$
$F(1) = 1$
$F(n) = F(n-1) + F(n-2)$, for all $n > 1$

Help Professor Fibo with his task by implementing a method called `generateFibo()` that takes a non-negative integer `n` as input, and uses a **RECURSIVE** algorithm to display the first `n` values in the above sequence.

Your code should be contained in a single method, with the following header:

```
public int generateFibo(int n)
```

17. Describe the output of the following program assuming the program is invoked as: **[5]**

```
java MysteryProgram 11
```

```
public class MysteryProgram {

   public static void main(String[] args) {
      int x = Integer.parseInt(args[0]);
      String s = (x <= 0) ? "0" : strangeMethod(x);
      System.out.println(s);
   }

   public static String strangeMethod(int x) {
     if (x == 0)
        return "";
     return strangeMethod(x / 2) + ((x % 2 == 0) ? "0" : "1");
   }
}
```

Section 3 - Programming Questions

18. Mismatched brackets are a common problem in programming. Many programming environments (IDEs) highlight these kinds of errors automatically, but in less nice environments, separate tools must be developed to check for common errors. **[12]**

    Write a program `CheckBrackets` that receives as command-line input the name of a text file to process. For example, your program would be invoked by typing:

    ```
    java CheckBrackets SomeOtherProgram.java
    ```

    in order to check the brackets in the file `SomeOtherProgram.java`. Your program should read through the file and ensure each '{' character is matched by a corresponding '}' character, and that each '(' character is matched by a corresponding ')' character. It should then print a message indicating how many brackets are unmatched for each type. For example, suppose the file `SomeOtherProgram.java` contains this:

    ```
    public class SomeOtherProgram {
       public void foo() {
          boolean b = false;
          for (int i = 0; i < 7; i++) {
             if (!b {      // notice missing right bracket
                System.out.println("i = " + i);
                b = true;
             // notice missing ending brace
          }
       }
    // notice missing ending brace
    ```

    Then running your program and its output should look like so:

    ```
    $ java CheckBrackets SomeOtherProgram.java
    unmatched (: 1
    unmatched {: 2
    ```

    Note that you need to verify that there enough of each kind of left bracket to match the number of right brackets (and vice-versa). In particular, you do not need to check that the two kinds of brackets are properly nested, nor do you need to verify that the code between matching brackets makes any sense. You can assume that no comments or `String` constants contain brackets of any form.

USE THIS PAGE IF YOU NEED ADDITIONAL SPACE FOR YOUR PROGRAM:

19. Internally, the `.class` files store method signatures in `String`s using a special code. The list of argument types and then the return type are encoded; each basic type is abbreviated as a letter, one of {Z, B, C, S, I, J, F, D} for `boolean`s, `byte`s, `char`s, `short`s, `int`s, `long`s, `float`s, and `double`s respectively, and the syntax "L*fullclassname*;" is used for specifying objects. For example, the following two methods: **[15]**

```
long bar(int index,String s,boolean b)
String foo(java.util.Random,long q,byte c,boolean b)
```

would have respective internal signatures:

```
(ILjava.lang.String;Z)J
(Ljava.util.Random;JBZ)Ljava.lang.String;
```

where the part in brackets are the arguments, in the first case an `int`, a `String` and a `boolean`, and in the second, a `Random` object, a `long`, a `byte`, and a `boolean`. The rest of the `String` after the brackets indicates the return type in the same encoding. In the first case, 'J' is used to indicate that a `long` value is returned, and in the second case, a `String`.

Write a method called `encode()` which receives as input a `String` representing a method header (as in the example above: one line, no body, no access modifiers, no exceptions declared) and which emits a `String` representing the encoded internal signature. For example, invoking your method like:

```
encode("String encode(String s)");
```

should produce the `String` result:

```
(Ljava.lang.String;)Ljava.lang.String;
```

You can assume any non-primitive types are either in the `java.lang` package, or fully-qualified names (complete package name already specified). You can further assume no arrays are used, either as arguments or return types, and that the input method declaration does not have a return type of `void`. You may use Java library methods, as well as write any private helper methods you find convenient.

Hint: Notice that in the input method declaration, the return type is first, and separated from the method name by a single space character. After the method name (which, like all identifiers may not contain whitespace characters), the formal parameter list has parameters separated by commas, and for each parameter, its type is separated from its name by a single space. A fully-qualified object type will include at least one period ('`.`') character, while one that is not fully-qualified will have no period characters. You can use these facts to come up with a way of extracting the list of parameter types and return type and producing the correct output.

USE THIS PAGE IF YOU NEED ADDITIONAL SPACE FOR YOUR PROGRAM:

20. At MVP University, each student registered in the Introduction to Computing I course is assigned to one TA for the duration of the course. Each student goes to the tutorial sessions offered by the TA he or she has been assigned to (every TA offers every tutorial session), and all of his or her assignments are graded by that TA as well. **[13]**

    For this question, you will write a part of a program that instructors who teach Introduction to Computing I at MVP University can use to randomly assign students to tutorial groups. As well as a class containing `main()` to drive the whole program, it consists of two primary classes:

    - A class called `TutorialGroup`; objects which belong to this class represent a tutorial group led by a TA and to which students can be assigned.
    - A class called `AllocationScheme`; objects which belong to this class assign students to tutorial groups when their `allocate()` method is called.

    The `TutorialGroup` class has already been written, and has the following functionality:

```
public class TutorialGroup {

    /* Creates an empty tutorial group led by the given TA. */
    public TutorialGroup(String TA) { /* Body */ }

    /* Returns the TA leading this tutorial group. */
    public String getTA() { /* Body */ }

    /* Returns the student at the specified index. */
    public String getStudent(int index) { /* Body */ }

    /* Returns the total number of students in this group. */
    public int getNumberStudents() { /* Body */ }

    /* Inserts the given student into the tutorial group. */
    public void add(String studentInfo) { /* Body */ }
}
```

    For this question, you must write the definition of the `AllocationScheme` class. This class class defines only one instance method, called `allocate()`. The `allocate()` method takes as its parameters an `ArrayList` of `String`s called `TAs`, and another `ArrayList` of `Strings` called `students`. `TAs` contains the names of all TAs who will lead tutorial sessions, and `students` contains the identifying information for all students registered in the course.

    The `allocate()` method returns an `ArrayList` of `TutorialGroup` objects. This list **MUST** contain exactly as many `TutorialGroup` objects as there are TA names in `TAs`; moreover, all the students represented by `String`s in `students` **MUST** have been `add()`ed to exactly one of the `TutorialGroup` objects in the returned `ArrayList`, and the students assigned to each `TutorialGroup` **MUST** be selected at random. Note that you **MUST NOT** change the parameter data; when the method returns, the input `ArrayList`s **MUST** contain the same elements in the same order as before the method was called.

    Your solution **MUST** ensure an even allocation, so each TA has as close to the same number of students in their `TutorialGroup` as possible (given that the number of students may or may not be a multiple of the number of TAs).

USE THIS PAGE IF YOU NEED ADDITIONAL SPACE FOR YOUR PROGRAM:

Total Marks for Section 3: **40**

Total Marks: **100**

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

LAST PAGE: SUMMARY OF METHODS FOR YOUR CONVENIENCE:

- java.lang.String

```
public int length();
public char charAt(int);
public boolean equals(java.lang.Object);
public boolean equalsIgnoreCase(java.lang.String);
public int compareTo(java.lang.String);
public boolean startsWith(java.lang.String);
public boolean endsWith(java.lang.String);
public int indexOf(int);
public int indexOf(int, int);
public int indexOf(java.lang.String);
public int indexOf(java.lang.String, int);
public java.lang.String substring(int);
public java.lang.String substring(int, int);
public java.lang.String replace(char, char);
public java.lang.String replaceFirst(java.lang.String, java.lang.String);
public java.lang.String replaceAll(java.lang.String, java.lang.String);
public java.lang.String[] split(java.lang.String);
public java.lang.String toLowerCase();
public java.lang.String toUpperCase();
public java.lang.String trim();
public java.lang.String toString();
```

- java.util.Scanner

```
public void close();
public java.util.Scanner useDelimiter(java.lang.String);
public boolean hasNext();
public java.lang.String next();
public boolean hasNextLine();
public java.lang.String nextLine();
public java.lang.String findInLine(java.lang.String);
public boolean nextBoolean();
public byte nextByte();
public short nextShort();
public int nextInt();
public long nextLong();
public float nextFloat();
public double nextDouble();
```

- java.util.ArrayList

```
public int size();
public boolean isEmpty();
public boolean contains(java.lang.Object);
public int indexOf(java.lang.Object);
public int lastIndexOf(java.lang.Object);
public java.lang.Object[] toArray();
public java.lang.Object get(int);
public java.lang.Object set(int, java.lang.Object);
public boolean add(java.lang.Object);
public void add(int, java.lang.Object);
public java.lang.Object remove(int);
public boolean remove(java.lang.Object);
public void clear();
```