UNCA CSCI 202
Final Exam 2013
7 May, 2013

This is a closed book and closed notes exam. It is a 150-minute exam to be given from 11:30 AM to 2:00 PM. Calculators, PDA's, cell phones, and any other electronic or communication devices may not be used during this exam.

Name: _____

If you want partial credit for imperfect answers, explain the reason for your answer!

Problem 1 (7 points)
Complete the `manyMe` method so that it creates and returns a Java `Deque` containing 2013 elements. Your name should be stored in all 2013 elements of the deque. The `LinkedList` is an implementation of the `Deque` interface, and the `Deque` interface has a `push` method.

```
public static Deque<String> manyMe() {




}
```

Problem 2 (4 points)
Suppose there is a class `Match4Exam` with methods having the following headers:

```
public        int    fancyOpA (int x, String y) ;
public        double fancyOpB (int x, double y) ;
public static double fancyOpC (int x, int y) ;
```

Also, assume that `fancyX` and `fancyY` are objects of type `Match4Exam`.

Write Java statements to do each of the following:

Invoke `fancyOpA`
Invoke `fancyOpB`
Invoke `fancyOpC`

Problem 3 (4 points)
Suppose `dq` is an empty `Deque<Integer>`. In the six boxes before, draw `dq` as it would appear after *each* of the following six Java expressions are executed in order.

- `dq.push(5)`
- `dq.push(3)`
- `dq.pop()`
- `dq.push(2)`
- `dq.peek()`
- `dq.pop()`

Problem 4 (4 points)
Insertion sort and selection sort are both $O(n^2)$ algorithms. Quicksort is a faster sorting algorithm with an average running time of $O(n \log n)$.

In the three subproblems given below, always start with the following unsorted array.

344	181	255	431	107	446	331	182

Here is what the array should contain when fully sorted.

107	181	182	255	331	344	431	446

Suppose insertion sort is about halfway though the sorting task, place some numbers in the boxes below to illustrate how the values might be stored in the array.

Suppose selection sort is about halfway though the sorting task, place some numbers in the boxes below to illustrate how the values might be stored in the array.

Suppose quicksort has performed one partition of the array using 331 as a pivot, place some numbers in the boxes below to show how the values might be stored in the array.

Problem 5 (15 points)
The table below contains two columns. The leftmost column is a Java expression. In the rightmost column write the value of the expression as a Java literal. If the value is a `double` or a `boolean` or a `String` be sure to use the Java syntax for writing `double`s or `boolean`s or `String`s. Some of these are tricky, but none require complex arithmetic.

In answering these problems, assume that the following variables have been defined.

```
int   a = 15 ;
int   b = 20 ;
```

`202 / 10 * 2`	
`202 % 10 * 2`	
`(int) 1.5 * 2 / 10`	
`(int) (1.5 * 2) / 10`	
`(double) 1 / 2`	
`7 / 15`	
`7 % 15`	
`"abc" + (int) 3.5`	
`("abc" + 3) + 5`	
`"abc" + (3 + 5)`	
`"abc".length()`	
`7 < 8 || !(7 < 8)`	
`a++`	
`--b`	
`-b`	

Problem 6 (6 points)
Circle all the Java tokens in the following assignment statement:

```
int z = x.toString() + "DOG" + (int) Math.PI + v[0] ;
```

Now circle all the subexpressions of the following logical expression:

```
x + y * ( z + 5 ) > 13 && a <= 0
```

Problem 7 (10 points)
Suppose you are asked to write a Java class within the `edu.unca.nm.nm222` package. What would be the first line of Java within your program?

If your program needs to import the `java.net.Socket` class, what would be the second line of Java within your program?

If your program is to implement a Java class called `ProblemX` that could be accessed from classes outside of the `edu.unca.nm.nm222` package, what would be the Java class header for your program?

What is name the file in which you must store your `ProblemX` class? You do not need to give the directory part of the name, just the part after the slash ("directory separator" in Java).

How would you declare a member variable of type `double` called `point` within your `ProblemX` class so that `point` can be accessed ***only*** by methods of the `ProblemX` class?

Write an accessor ("getter") method for the `point` member that can be invoked from outside of the `ProblemX` class.

Write a mutator ("setter") method for the `point` member that can be invoked from outside of the `ProblemX` class.

If `p` is an object of type `ProblemX`, write a very short section of Java code that uses your accessor and mutator methods to double the number stored in the `point` member of `p`.

Problem 8 (6 points)
Write a `static` ***recursive*** method `logOf2` in Java to compute discrete logarithms of 2 using the following recursive definition:

$\texttt{logOf2}(n) = 0$, if n **≤** 1

$\texttt{logOf2}(n) = 1 + \texttt{logOf2}(n / 2)$, if n **>** 1

Start with the following method header:

```
public static int logOf2(int n) {
```

Problem 9 (7 points)
Write a `static` *iterative* method `logOf2` in Java to compute discrete logarithms of 2 as defined above. I suggest you create a variable `k`, initialized to 1, and count the number of times you must double `k` until it is larger than `n`. Again, start with the following header:

```
public static int logOf2(int n) {
```

Problem 10 (6 points)
Write a Python function `logOf2` to compute discrete logarithms of 2 as defined in the previous problems. Your Python function can be either iterative or recursive. (I think that recursive will be much easier.) Start with the following Python function definition header:

```
def logOf2(n):
```

Problem 11 (6 points)

Here is a class representing a six-sided die.

```
public class Die {
    private int die ;
    public Die() {
        die = 1 ;
    }
    public int getValue() {
        return die ;
    }
    public void setValue(int value)
                        throws IllegalArgumentException {
        if (value < 1 || 6 < value) {
            throw new IllegalArgumentException() ;
        }
        die = value ;
    }
}
```

Here is Java's `Comparable` interface without all that package baggage:

```
public interface Comparable<T> {
    public int compareTo(T o) ;
}
```

According to the `Comparable` Javadoc, the `compareTo` method returns "a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object". (Return a positive number if the object is greater than the parameter to the method, *etc*.)

In the remainder of this page, write a complete Java class called `ComparableDie` that extends the class `Die` and implements the interface `Comparable`. The implementation of `compareTo` is similar to the implementations of the following method in Homework 11.

```
public double distance(Point1080 p) ;
```

and the many `Compare` methods of Homework 13. This can be done in six to ten lines of Java.

Problem 12 (25 points)
Within the Java `Arrays` class, `java.util.Arrays`, there are several useful methods for sorting arrays. The description of one of these methods, `sort`, is given below:

`public static void sort(int[] a)`

Sorts the specified array into ascending numerical order.
Parameters:
`a` – the array to be sorted

The following two pages contain the outline of a Java program. The comments within the program specify incomplete code blocks, identified by letters. Your task is to fill in the code blocks. Additional information about these blocks is given below.

A) Start your program with the statements needed to create a public class `PrintMedian` within the `edu.unca.cs.csci202` package. Your program should also import the `java.util.Arrays` and the `java.util.Scanner` classes. [2 points]
B) Provide the header for a private static method called `readNumbers` that receives a `Scanner` as its only parameter and returns an array of integers. The `Scanner` parameter **must** be named `numbIn`. [1 point]
C) Create an array of integers of size `retSize`. The method `readNumbers` already contains code for setting the variable `retSize` by reading an integer using the `Scanner` variable `numbIn`. [1 point]
D) Use `numbIn` to read `retSize` numbers into your newly created array. You'll need a loop to do this. Note that all I/O is being done within a `try` statement. [4 points]
E) Return the newly created array. [1 point]
F) If any exception occurs while the numbers are being read, `readNumbers` should return `null`. Provide the exception handler for doing this. [3 points]
G) Provide the header for a private static method called `getMedian` that receives an array of integers as its only parameter and returns the median of that array of integers as a double. The integer parameter **must** be named `numbers`. [1 point]
H) The program already contains a statement to create a copy of `numbers` called `cNumbers`. Add a statement to sort `cNumbers`. This is a one-liner. [2 points]
I) The median of a sample of numbers is the middle element when the numbers are listed in sorted order. If `cNumbers` has an odd number of elements, return the one in the middle. If `cNumbers` has an even number of elements, return the average (which need not be an integer) of the two in the middle. Assume that `cNumbers` has at least one element. [4 points]
J) Now it's time to complete the `main` method. [6 points]
   - Start by calling `readNumbers`, with the `Scanner` variable `stdin`, to read the samples.
   - If `readNumbers` returns `null`, do nothing more.
   - If `readNumbers` returns an array of integers;
     - First call `getMedian` to obtain the median value of the array
     - Then print all the numbers of the array that are greater than the median.

For example, if input number sequence is

```
4    202   181   107   182
```

then `readNumbers` should return [202, 181, 107, 182], `getMedian` should return 181.5, and the numbers 202 and 182 should be printed.

```
// Part A:  class header and supporting statements
         ____________________________________________

         ____________________________________________

         ____________________________________________

         ____________________________________________
// Part B:  method header for readNumbers
//          The input parameter must be named numbIn
//          Method must return an array of integers

_______________ readNumbers ____________________________ {

    try {
         int retSize = numbIn.nextInt() ;
         if (retSize <= 0) {
              return null ;
         } //    this ends the if, but not the try
// Part C:  Create an integer array of size retSize

         ____________________________________________ ;

// Part D:  Read retSize integers into your array
         ____________________________________________

         ____________________________________________

         ____________________________________________

         ____________________________________________
// Part E:  Return the array created in Part C
         ____________________________________________
// Part F:  Add an exception handler to return null
//          if any exception was raised in the try
    } catch(__________________) {

         ____________________________________________

         ____________________________________________

    }
} // End of readNumlines
```

```
// Part G:  method header for getMedian
//          The input parameter must be named numbers
//          Method must return a double

_________________ getMedian _______________________________ {

// Clone numbers into a new array
   int[] cNumbers = numbers.clone() ;
// Part H:  Sort cNumbers (use Arrays sort method)

    ____________________________________________

// Part I:  Return the median (see previous page)

    ____________________________________________

    ____________________________________________

    ____________________________________________

    ____________________________________________

    ____________________________________________

    ____________________________________________

}

// Part J:  Write the main routine

public static void main(String[] args) {

     Scanner stdin = new Scanner(System.in) ;

     ____________________________________________

     ____________________________________________

     ____________________________________________

     ____________________________________________

     ____________________________________________

     ____________________________________________

     ____________________________________________

     ____________________________________________

     ____________________________________________

}  // end of main
}  // end of class
```