CS 251L
2010.9.3
Review Quiz Key

The purpose of this quiz will be to enable you to know where you stand in preparation for this semester's material. It is important that you have a firm grasp of procedural programming in Java. The quiz will not be graded but it is highly recommended that you complete this over the next two weeks as we review.

1. Enumerate every primitive data type and the number of bytes required by each.

   boolean (1/8), byte (1), char (2), int (4), short (2), long (8), float (4), double (8)

   The actual number of bytes required by booleans, bytes, and shorts may vary depending on the underlying system architecture for performance reasons. Computers process data much faster if the data comes in chunks of the size equal to that of the computer's "word size", which is often now 4 or 8 bytes in modern computers.

2. Enumerate each arithmetic operator.

   ```
   +   -   *   /   %
   ```

   Modulus operator is the "remainder of" operator

3. Enumerate each relational operator.

   ```
   <   <=   >   >=
   ```

4. Show three different ways to increment a variable by one.

   ```
   var = var + 1;
   var += 1;
   var++;
   ```

5. Explain the difference between the following two blocks of code:

   ```
        int i = 0;
        System.out.println(++i);    // Outputs "1"

        int i = 0;
        System.out.println(i++);    // Outputs "0"
   ```

   The first block's prefix increment operator increments the value in the variable `i`, and returns the new value.
   The second block's postfix increment operator increments the value in the variable `i`, but returns the value it had before the increment.

6. What is the result of this expression if a = 10, b = 20, and c = 30? (by hand obviously! ☺)

```
++a % 7 + b * c++ / 10 - a

(((++a) % 7) + ((b * (c++)) / 10)) - a
```

Parentheses above show the operator precedence and associativity for this expression. Once you know the operator precedence you can evaluate the expression.

```
((11 % 7) + ((b * 30) / 10)) - a
(4 + ((20 * 30) / 10)) - a
(4 + 60) - a
64 - a
64 - 11
53
```

Remember that the second reference to `a` will evaluate to 11 instead of 10 because the prefix increment operator executed earlier in the expression actually changes the variable's value.

7. List the resulting values and data types of these expressions:

   10 * 20 = 200, integer
   10.0 * 20 = 200.0, double
   20 / 10 = 2, integer
   10 / 20 = 0, integer*
   10.0 / 20 = 0.5, double*
   20 / 10.0 = 2.0, double

   * When two integers types are divided then "integer division" is performed, which is essentially a truncation of the real mathematical value, since integer data types cannot hold any information past the decimal point.

   Remember that the "smaller" operand in an arithmetic expression is "up cast" to the data type of the "larger" operand. In other words, the integer value 10 in 10 * 20.0 is "up cast" to a double 10.0 before the multiplication occurs.

8. Does Java have a separate language keyword for "else if" ?

   No, Java only has an "if", and an "else". The "else if" concept in Java is implemented by merely placing another "if" in an "else"clause.

```
if(expr) {
} else if(expr) {
}
```

same as

```
if(expr) {
} else {
   if(expr) {
   }
}
```

In the Bash scripting language for Unix systems, this is an "if-else if-else" block:

```
if [ expr ]; then
    echo "msg"
elif [ expr ]; then
    echo "msg2"
else
    echo "msg3"
fi
```

In Perl, this is an "if-else if-else" block:

```
if (expr) {
    print "msg";
} elsif (expr) {
    print "msg2";
} else {
    print "msg3";
}
```

They both have separate language keywords to implement "else if". Visual Basic uses "ElseIf" for its "else if" keyword. Which do you prefer, the simplicity of Java (and C, C++), or the explicitness of these other languages?

9. What are the 3 types of loops?

   counted loops (for), top-tested loops (while), bottom-tested loops (do-while)

10. Convert this loop to a top-tested loop:

```
for(int var = 0; var < 20; var += 2) {
   dbConnection.ping();
   var += 2;
}
```

```
int var = 0;
while(var < 20) {
   dbConnection.ping();
   var += 4;
}
```

11. Write a counted loop to sum every integer divisible by 17 between 100 and 200, inclusive.

```
int sum = 0;
for(int var = 100; var <= 200; var++) {
   if(var % 17 == 0) {
      sum += var;
   }
}
```

12. Write a bottom-tested loop whose body executes the statement `connection.establish();` and then tests the condition `!connection.isAlive();`

```
do {
   connection.establish();
} while(!connection.isAlive());
```

13. Declare an array of type double and of size 20 and initialize its values to the square roots of the numbers between 49 and 68.

```
double[] vals = new double[20];
for(int v = 0; v < 20; v++) {
   vals[v] = Math.sqrt(v + 49);
}

double[] vals = new double[20];
for(int v = 49; v <= 68; v++) {
   vals[v - 49] = Math.sqrt(v);
}
```

14. Identify what the following loop does:

```
float[] prices = // initialize array.
for(int a = 0; a < prices.length - 1; a++) {
   for(int b = a + 1; b < prices.length; b++) {
      if(prices[a] > prices[b]) {
         float temp = prices[a];
         prices[a] = prices[b];
         prices[b] = temp;
      }
   }
}
```

This code implements an "exchange sort". It uses a doubly-nested for loop to scan over the elements in the array, swaps elements that are out of order.

15. Are strings in Java a primitive data type?

No, they are fully-fledged objects in Java. These objects essentially are wrappers around an array of type 'char'. In other programming languages like C and C++, the manipulation of strings using character arrays is commonplace.

16. List the results of these expressions:

`"Paris".charAt(3)`	`'i'`
`"Moscow".substring(0, 2)`	`"Mo"`
`"Berlin".substring(3)`	`"lin"`

```
"Caracas".indexOf("q")              -1

"Manama".replace('a', 'o')          "Monomo"

"Buenos Aires".lastIndexOf('e')     10
```

17. What is the output of the following code?

```
String s = "Canada";
String s2 = new String("Canada");

System.out.println(s == s2);
System.out.println(s.equals(s2));
```

This code outputs false, then true. Since the “new” operator was used to construct the second string, it will explicitly get different memory than the first string. Therefore, then equality operator, which tests the identity of objects, will fail since the two are not the same object. However, the equals operator for a string object actually compares the characters in the two strings so it passes.

18. Write a method that, given the two floating-point lengths of the legs of a right triangle, returns the hypotenuse of this triangle.

```
public static double hyp(double leg1, double leg2) {
   return Math.sqrt(leg1 * leg1 + leg2 * leg2);
}
```

19. Write four methods that each take in an array of integers. These methods will return:
    a. The maximum value of the array

```
public int max(int[] arr) {
   int max = Integer.MIN_VALUE;
   for(int a = 0; a < arr.length; a++) {
      if(arr[a] > max) {
         max = arr[a];
      }
   }
   return max;
}

public int max(int[] arr) {
   Arrays.sort(arr);
   return arr[arr.length - 1];
}
```

The only problem with the second solution is that the Arrays.sort method actually modifies the array, leaving the array modified upon returning from the method, which is almost always undesirable when writing a method that should just inspect the array for the maximum value.

    b. The mean value of the array. Values of -1 in the array do not affect the mean.

```
public double meanIgnoreNegOne(int[] arr) {
   int numNonNegOne = 0;
   int sumNonNegOne = 0;
   for(int a = 0; a < arr.length; a++) {
      if(arr[a] != -1) {
         numNonNegOne++;
         sumNonNegOne += arr[a];
      }
   }
   return (double) sumNonNegOne / numNonNegOne;
}
```

On the last line you must cast one of the integer variables to a double so that floating-point arithmetic is performed in the calculation. Otherwise, you will not get as exact an answer.

c. The median value of the array. Assume you have access to this self-explanatory method:

```
     private int[] sort(int[] array);
```

```
public int median(int[] arr) {
   int[] sorted = sort(arr);
   if(sorted.length % 2 == 1) {
      return sorted[sorted.length / 2];
   }
   int half = sorted.length / 2;
   return (sorted[half - 1] + sorted[half]) / 2.0;
}
```

When calculating the median of a set of numbers, it matters whether you have an even or odd number of numbers in the set. For an odd number of numbers, you just return the middle value, but for an even number of numbers, you need to take the average of the two middle numbers.

d. The mode of the array (a thinker…).

See associated Java code.