UNIVERSITY OF WISCONSIN-LA CROSSE
Department of Computer Science

CS 120 | **Software Design I** | **Fall 2016**
Final Exam | **21 December 2016**

- **Do not turn the page until instructed to do so.**
- **This booklet contains 11 pages including the cover page.**
- **This is a closed-book exam. All you need is the exam and a writing utensil.**
- **You have exactly 120 minutes.**
- **The maximum possible score is 80 points.**

PROBLEM	SCORE
1	
2	
3	
4	
5	
6	
7	
8	
TOTAL	

NAME: _____

1. **(10 pts.) TRUE/FALSE.**

   Label each statement true or false. **You do not need to explain your answers.**

   a. The following contains an error:

   ```
   double[][] dubs = new double[10][10];
   dubs[1][2] = 3;
   ```

   b. The following contains an error:

   ```
   double[][] dubs = new double[10][10];
   dubs[1.0][2.0] = 3.0;
   ```

   c. The following loop runs exactly 4 times:

   ```
   for ( int i = 1; i <= 4; i++ )
       System.out.print( i + " " );
   ```

   d. The following loop runs exactly 4 times:

   ```
   for ( int i = 0; i != 4; i++ )
       System.out.print( i + " " );
   ```

   e. The following loop runs exactly 4 times:

   ```
   for ( int i = 0; i < 4.5; i++ )
       System.out.print( i + " " );
   ```

   f. Any variable declared inside a loop is always local to that loop, and cannot be accessed outside the loop.

   g. After running the following code, the variable `length == 0`.

   ```
   String s;
   int length = s.length();
   ```

   h. A Java class can only `extend` a single class at one time.

   i. A Java class can only `implement` a single interface at one time.

   j. Everybody likes a free point on an exam!

2. **(10 pts.) SHORT ANSWER.**

i. (5 pts.) When we over-ride a method, the new version ***must have*** four things in common with the original:

i. _____

ii. _____

iii. _____

iv. _____

One thing that ***can be different*** between the two versions of the method is:

v. _____.

ii. (3 pts.) When we `extend` a parent class, the child class has direct access to every method and global instance variable of the parent that has access type (circle all that apply):

i. `public`

ii. `private`

iii. `protected`

iii. (2 pts.) Suppose we have a class, `Driver`, with code using two arrays and an object of type `Converter`:

```
double[] nums = { 1.2, 3.4, 5.6, 8.9 };
Converter con = new Converter();
int[][] ints = con.convert( nums, nums.length );
```

Without knowing what the `convert()` method does, we do know what its method signature (i.e., its first line) must be. What will it look like, exactly?

3. **(10 pts.) CODE EVALUATION, I.**

For each of the following, give the output of the code.

```
char[] blizzard = {'Z', 'Z', 'Y', 'X', 'W'};
System.out.println( blizzard.length );
System.out.println( blizzard[1] != blizzard[2] );
System.out.println( blizzard[3] >= blizzard[3] );
System.out.println( "COLD" + blizzard[2] + blizzard[2 - 1] );
System.out.println( blizzard[blizzard.length - 1] );
```

```
int base = 2;
while( base < 6 )
{
    int pow = base;
    for( int e = 2; e != 4; e++ )
    {
        pow = pow * pow;
        System.out.println( pow + ", " + e );
    }
    base = base + 3;
}
```

4. **(5 pts.) CODE EVALUATION, II.**

Suppose we have the following two class definitions:

```
public class ClassA {
    protected int x;

    public ClassA( int xin ) {
        x = xin;
    }

    public int getValue() {
        return x;
    }
}

public class ClassB extends ClassA {
    private int y;

    public ClassB( int z, int yin ) {
        super(z);
        y = yin;
    }

    public int getY() {
        return y;
    }

    public int getValue() {
        return x + y;
    }
}
```

Suppose now we create three variables as follows:

```
ClassA a = new ClassA( 1 );
ClassB b = new ClassB( 2, 3 );
ClassA c = b;
```

Circle the line(s) among the three immediately below that will **not** compile properly, based on the code above that creates the variables `a`, `b`, and `c`.

```
int u = a.getY();

int v = b.getY();

int w = c.getY();
```

After the following, what are the values of the variables `x`, `y`, and `z`, based on the code above that creates the variables `a`, `b`, and `c`?

```
int x = a.getValue();
int y = b.getValue();
int z = c.getValue();
```

5. **(10 pts.) CODE COMPLETION, I.**

Write a method called `sumPositive()` below. This method should:

- Take a 2-dimensional array of integers as input. This array ***can be ragged*** (with different numbers of columns per row).
- Return an integer array; each element of the array should consist of the sum of values that are greater than 0 in the corresponding input array row.

Thus, if we have the following code:

```
int[][] nums = { { -1, 2, 3 }, { 2, -4, 3, 1 }, { -3, -4 } };
int[] out = sumPositive( nums );
```

then when we are done, we will have the array `out == { 5, 6, 0 }`.

For full points, your code must use **nested loops**, each of which is actually used to generate the output. (You may use whatever types of loops you choose.)

---

6. **(10 pts.) CODE COMPLETION, II.**

Complete the class below as follows:

- Add a `void` method called `setMileage()` that takes in two `double` values, one to be assigned to the `distance` variable, and one to be assigned to the `gas` variable. The method should check its input parameters. A valid parameter for `distance` must be ***greater than or equal to*** `0`, and a valid parameter for `gas` must be ***greater than*** `0`. The method must assign proper values to `gasIsValid` and `distanceIsValid` to reflect whether or not the input parameters were in fact valid. If ***both*** parameters are valid, then the method should assign these values to the proper instance variables; otherwise the input values should be ignored.
- Add a method called `isValid()` that returns **true** if and only if both of the boolean instance variables is **true**.

```
public class TripMileage
{
    private double distance, gas;
    private boolean gasIsValid, distanceIsValid;









}
```

7. **(10 pts.) CODE COMPLETION, III.**

Fill in the class below. It should have two methods.

- The first, `countDollars()`, should take in a `String` as input, and return the number of dollar signs (`$`) in it.
- The second, `highCount()`, should take in two `String` objects and a single character as input, and return the string that has the highest number of occurrences of that character. Thus, calling `highCount( "test", "tattle", 't' )` would return `"tattle"`, since it has three occurrences of `'t'`, whereas `"test"` has only two. In the event of a tie, it should return the ***first*** input string, so `highCount( "test", "tutor", 't' )` would return `"test"`.

```
public class Question7
{









}
```

8. **(15 pts.) CODE COMPLETION, IV.**

On the next page, there is code for a `Driver` class. It contains a method that you need to complete (instructions for doing so are in the comments within the code). Doing that will also involve writing another class, `BetterWindow`.

(a) The `BetterWindow` class should extend the `Window` class (which has the class diagram shown at the bottom of this page). The `Window` class has the usual methods for changing size, location, etc. It has a constructor that takes a title as input; calling that constructor will create a window with the given title displayed at the top.

(b) The `BetterWindow` constructor should also take a title as input and display it (in the format described in part **(d)**, below). The window should always start off with width and height both 250. Thus, the following creates a (250×250) window with title `"My window"`:

```
BetterWindow win = new BetterWindow( "My window" );
```

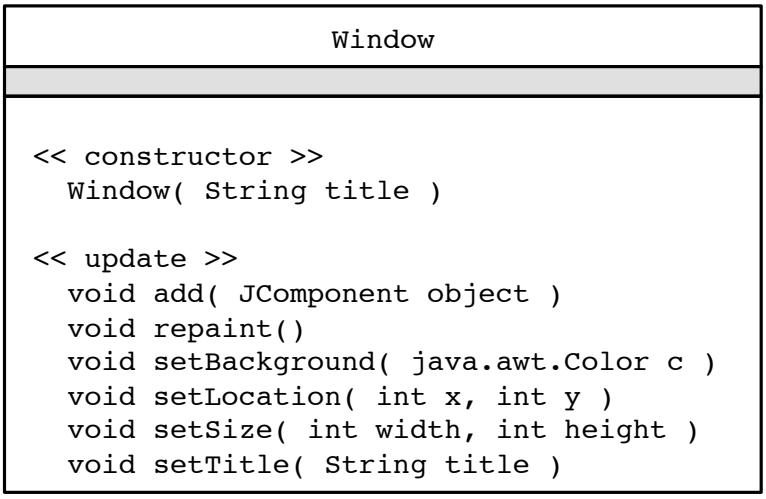
(c) The new class should over-ride the `setSize()` method. The new version should only allow width or height of the window to be between `250` and `500` pixels, inclusive. If either value is set too low, then the method should simply use the smallest possible value, and if too high, use the highest. Thus, the following will produce a window that is actually (250 × 500) pixels in size:

```
win.setSize( 100, 800 );
```

(d) The new class should over-ride the `setTitle()` method as well. The new version should ensure that the window title is in ***sentence case***: that is, the first letter of the title should be capitalized, and any remaining letters are in lower-case. (If the title starts with or contains other characters that are not letters, that is OK.) Thus, the following will produce a window that actually has title `"This is a window"`:

```
win.setTitle( "this IS A WINdow" );
```

(e) You should also fill in the rest of the `Driver` class; the comments in that class indicate what should be added there. (***Note***: there are **TWO** places where code needs to be added to the `Driver` class.)



```
public class Driver
{
    // [ONE] Declare global array variable here.




    public static void main( String[] args )
    {
        Driver d = new Driver();
        d.makeWindows();
    }

    // [TWO] Complete the method, to fill a global array of 10 things.
    // Things with an even array index (0, 2, ...) should
    // be normal Window objects, and things with an odd index
    // should be BetterWindow objects.
    //
    // Each should have title "Window X" (where X is its index in the array).
    // Each should be of size (500 x 500), at location (20,20).
    private void makeWindows()
    {









    }
}
```

```
// Write the BetterWindow class here.
```