UNIVERSITY OF WISCONSIN-LA CROSSE
Department of Computer Science

CS 120 **Software Design I** **Spring 2017**
Final Exam (Practice) **11 May 2017**

- **Do not turn the page until instructed to do so.**
- **This booklet contains 14 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	10
2	10
3	10
4	5
5	10
6	10
7	10
8	10
9	5
TOTAL	80

ANSWER KEY

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

   For each of the following, indicate whether the statement is true or false.
   You do not need to explain your answers (although I sometimes do so here).

   a. Anywhere an object of class `C` can be used, a call to a non-`void` method that returns something of type `C` can be used.

      True

   b. A variable is always local to the scope in which it is instantiated.

      False

   c. A variable is always local to the scope in which it is declared.

      True

   d. If we nest one loop inside the other, then both loops must be of the same type (i.e., they must both be `for`-loops or both `while`-loops).

      False

   e. Every result that can be achieved using a `while` loop can be achieved using a `for` loop.

      True

   f. When we override a method, the new version must have the same output `return` type as the original.

      True: this is necessary so that the returned values can be used in the same contexts.

   g. The following contains an error:

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

      False: since `int` literal values are widened automatically to `double` values, we can assign an `int` to an index of an array of `double` type.

   h. The following contains an error:

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

      True: array indices are always integers, no matter what sort of data the array holds.

   i. The following loop runs exactly 4 times:

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

      True

   j. The following loop runs exactly 4 times:

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

      True

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

a. (3 pts.) In Java, expressions are *converted automatically* from a **less precise (or *narrower*)** type to a **more precise (or *wider*)** type, but not vice-versa. For example, an expression that evaluates to `int` type **will** convert automatically to `double` type.

b. (3 pts.) The special Java keyword `this`, when used within the code of a class `C`, always refers to **the current instance of the class (the one whose method or variable is being used)**. The special keyword `super`, when used in a class `C`, always refers to **the parent class instance of the current object**. In particular, the method call `super()` always refers to **the parent class constructor**.

c. (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`

d. (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?

Answer: `public int[][] convert( double[] nums, int length )`

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

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

```
double[] d1 = { 1.0, 2.0, 3.0, 4.0, 5.0 };
double[] d2 = new double[d1.length * 2];
for ( int i = 0; i < d1.length; i++ )
{
    d2[i] = d1[i];
    d2[d2.length - i - 1] = d2[i];
}

for ( int i = 0; i < d2.length; i++ )
    System.out.print( d2[i] + " " );
```

Answer: this code inserts the values from `d1` into `d2`, front-to-back and then back-to-front, and prints them out as follows:

```
1.0 2.0 3.0 4.0 5.0 5.0 4.0 3.0 2.0 1.0
```

```
String[] words = { "Every", "good", "bird", "deserves", "feeding" };
for ( int w = 0; w < words.length; w++ )
{
    System.out.print( words[w] + " >=" );
    for ( int j = 0; j < words.length; j++ )
        if ( words[j].length() <= words[w].length() )
            System.out.print( " " + words[j]);
    System.out.println();
}
```

Answer: this code prints out each element of `words`, followed by all of those elements that are of length *less than or equal to* that of the first element:

```
Every >= Every good bird
good >= good bird
bird >= good bird
deserves >= Every good bird deserves feeding
feeding >= Every good bird feeding
```

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

Suppose we have the following two class definitions:

```
public class ClassA {
    protected int num1, num2;

    public ClassA( int i1, int i2 ) {
        num1 = i1;
        num2 = i2;
    }

    public boolean larger() {
        return ( num2 > num1 );
    }
}

(public class ClassB extends ClassA {
    private int num3;

    public ClassB( int i1, int i2, int i3 ) {
        super( i1, i2 );
        num3 = i3;
    }

    public boolean larger() {
        boolean large1 = super.larger();
        boolean large2 = num3 > num2 ;
        return ( large1 && large2 ) ;
    }
}
```

For each of the `println()` statements below, give the output:

```
        ClassA a1 = new ClassA( 0, 0 );
        ClassA a2 = new ClassA( 0, 5 );
        ClassB b1 = new ClassB( 0, 0, 5 );
        ClassB b2 = new ClassB( 0, 6, 9 );
        ClassB b3 = new ClassB( 0, 6, 6 );

        System.out.println( a1.larger() );     \\ ANSWER:  false
        System.out.println( a2.larger() );     \\ ANSWER:  true
        System.out.println( b1.larger() );     \\ ANSWER:  false
        System.out.println( b2.larger() );     \\ ANSWER:  true
        System.out.println( b3.larger() );     \\ ANSWER:  false
```

(For `ClassA`, the method returns `true` if and only if the second input is strictly larger than the first; for `ClassB`, the over-ridden version returns `true` if and only if ***both*** the first is strictly larger than the second, ***and*** the third is strictly larger than the second.)

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

Fill in the `main()` method in the class below so that when it runs it prints output (using `System.out.println()`) that looks like this:

```
0 1 2 3 4 5
1 2 3 4 5 0
2 3 4 5 0 1
3 4 5 0 1 2
4 5 0 1 2 3
5 0 1 2 3 4
```

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.)

Answer: This pattern, which "rotates" the values, is most easily solved using the remainder function:

```
public class Main
{
    public static void main( String[] args )
    {
        for ( int i = 0; i < 6; i++ )
        {
            int num = i;
            for ( int j = 0; j < 6; j++ )
            {
                System.out.print( num + " " );
                num = ( num + 1 ) % 6;
            }
            System.out.println();
        }
    }
}
```

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

Fill in the class below. Add the `sumArray()` method that has been called from the class constructor. When run, this method should act as follows:

- If the arrays are of ***identical*** length, then it should return a new array of the ***same*** length, where each element is the sum of the elements at the same index in the input arrays. So, in the first call below, it would sum the first elements of the inputs, then their second elements, and so on, and output array `out1` would look like:

  `{ 2, 3, 5, 6, 8 }`

- If the arrays are of ***different*** lengths, then it will return an ***empty*** array, containing no data. (This is what would be returned for the second call, so `out2` would be empty.)

```
public class Main
{
     public Main()
     {
        int[] arr1 = { 1, 2, 3, 4, 5 };
        int[] arr2 = { 1, 1, 2, 2, 3 };
        int[] arr3 = { 1, 2 };

        int[] out1 = sumArrays( arr1, arr2 );
        int[] out2 = sumArrays( arr2, arr3 );
     }

     private int[] sumArrays( int[] a1, int[] a2 )
     {
         int[] output = new int[0];

         if ( a1.length == a2.length )
         {
             output = new int[a1.length];
             for ( int i = 0; i < output.length; i++ )
                 output[i] = a1[i] + a2[i];
         }
         return output;
     }
}
```

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

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

- The first, called `concatenate()`, should take in two `String` objects as input, and return a single `String` composed of the first input, followed immediately by the second input (with no spaces). Thus, calling `concatenate( "A", "B" )` will return `"AB"`.
- The second, called `remove()`, should take in a `String` and a single character, and return the result of removing every occurrence of that exact character from the `String`. Thus, calling `remove( "Test", 't' )` will return `"Tes"` (only removing the lower-case `'t'`), and calling `remove( "Test", 'x' )` will return `"Test"`, since it does not contain `'x'`.

```
public class Driver
{
    public String concatenate( String str1, String str2 )
    {
        String concat = str1 + str2;
        return concat;
    }

    public String remove( String str, char c )
    {
        String rem = "";
        for ( int i = 0; i < str.length(); i++ )
        {
            char current = str.charAt( i );
            if ( current != c )
            {
                rem = rem + current;
            }
        }
        return rem;
    }
}
```

8. **(10 pts.) CODE COMPLETION, IV.**

On the next page, there is code for a `Main` class. It uses a class called `ArrayWorker`, which you must write yourself. The class you write should work as follows:

(a) It should have a constructor method that takes in an array of `String` data, and saves a reference to that array, for use in other methods in the class.

(b) It should have a method called `getLongest()` that returns the longest `String` in the original input array. If there is a tie between two words of the same length, it should return the ***first one*** (i.e., the one occurring at the smallest index in the array). If the array is empty, then it should return an ***empty*** `String`.

(c) It should have a method called `wordLengths()` that returns an array of integer values; each value will be the length of the corresponding `String` in the original input array. If that array was empty, the array returned by the method will also be empty.

You can find the `Main` code that your new class must work with, along with the output you would see by running the `main()` method, on the next page; write your new class on the page after that.

```
public class Main
{
    public static void main( String[] args )
    {
        String[] words1 = { "This", "is", "a", "test" };
        ArrayWorker worker1 = new ArrayWorker( words1 );
        String longest1 = worker1.getLongest();
        System.out.println( "Longest word is: " + longest1 );
        int[] wordLengths1 = worker1.getLengths();
        System.out.print( "Lengths are: " );
        for ( int i = 0; i < wordLengths1.length; i++ )
        {
            System.out.print( wordLengths1[i] + " " );
        }
        System.out.println();
        System.out.println();

        String[] words2 = new String[0];
        ArrayWorker worker2 = new ArrayWorker( words2 );
        String longest2 = worker2.getLongest();
        System.out.println( "Longest word is: " + longest2 );
        int[] wordLengths2 = worker2.getLengths();
        System.out.print( "Lengths are: " );
        for ( int i = 0; i < wordLengths2.length; i++ )
        {
            System.out.print( wordLengths2[i] + " " );
        }
        System.out.println();
        System.out.println();
    }
}
```

Sample output: when run, the above code produces the following:

```
Longest word is: This
Lengths are: 4 2 1 4

Longest word is:
Lengths are:
```

```
public class ArrayWorker
{
    private String[] strings;

    /**
     * Class constructor; input array of data is saved for later use.
     */
    public ArrayWorker( String[] words )
    {
        strings = words;
    }

    /**
     * Returns longest String in array; returns empty String if array is empty.
     */
    public String getLongest()
    {
        if ( strings.length == 0 )
        {
            return "";
        }

        String longest = strings[0];
        for ( int i = 1; i < strings.length; i++ )
        {
            if ( strings[i].length() > longest.length() )
            {
                longest = strings[i];
            }
        }
        return longest;
    }

    /**
     * Returns array containing lengths of Strings in original array; returns
     * empty array if original is empty.
     */
    public int[] getLengths()
    {
        int[] lengths = new int[strings.length];
        for ( int i = 0; i < strings.length; i++ )
        {
            lengths[i] = strings[i].length();
        }
        return lengths;
    }
}
```

9. **(5 pts.) CODE COMPLETION, V.**

Below, there is code for a `Driver` class. It uses a class called `ColoredOval`, which you must write yourself. The class you write should work as follows:

(a) It should extend the basic `Oval` class.

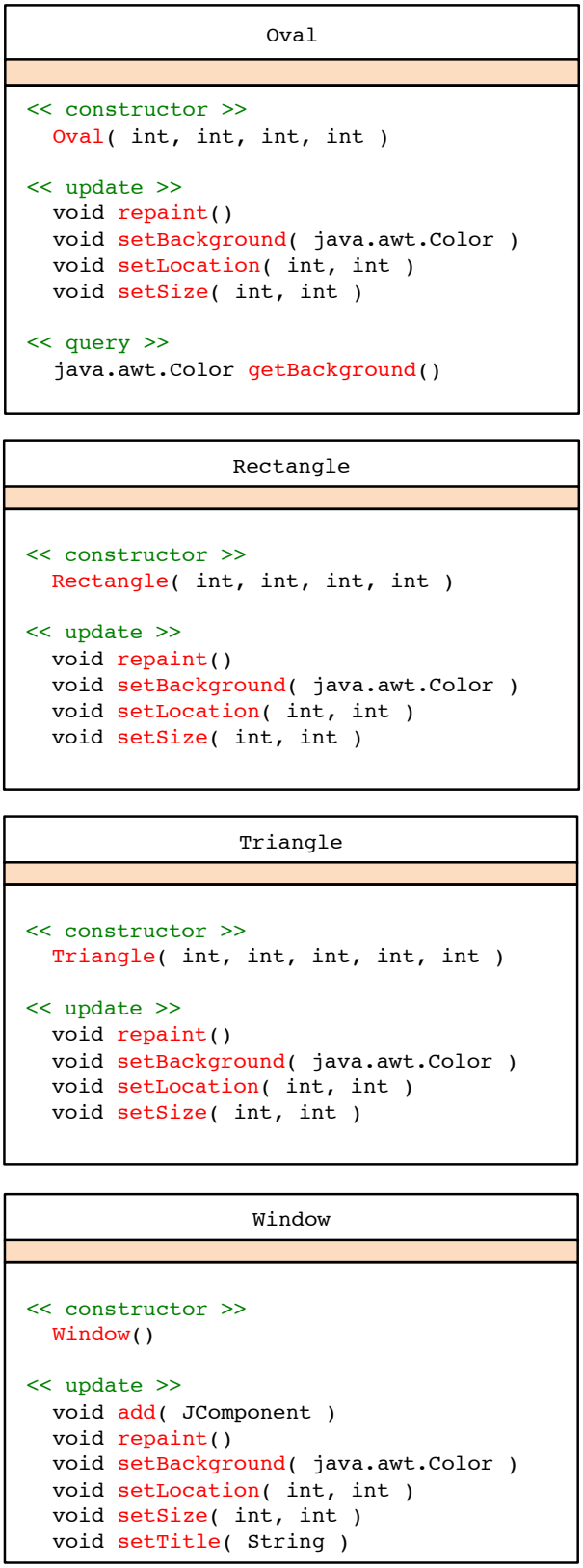
(b) It should have a constructor that takes in its (x, y) location, its *width* and *height* in pixels, and a `Color`. It then creates a graphical oval object with the given location and size, colored the input `Color`. Thus, when the `Driver()` constructor finishes running, the window will contain a red circular object.

Note: class diagrams for graphical classes appear on the last page of the exam.

```
import java.awt.Color;
import javax.swing.JFrame;

public class Driver
{
    private ColoredOval colored;

    public Driver()
    {
        JFrame window = new JFrame( "Question 9" );
        window.setBounds( 100, 100, 500, 500 );
        window.setLayout( null );
        window.getContentPane().setBackground( Color.white );
        window.setVisible( true );

        colored = new ColoredOval( 200, 200, 100, 100, Color.red );
        window.add( colored );
    }
}
```

```
import java.awt.Color;

public class ColoredOval extends Oval
{
    /**
     * Constructs shape object;  sets usual bounds parameters,
     * while also setting color to given input value.
     */
    public ColoredOval( int x, int y, int w, int h, Color c )
    {
        super( x, y, w, h );
        setBackground( c );
    }
}
```

Note: you could also use `super.setBackground( c )` in the last line of the constructor.


Oval
<< constructor >>
Oval(int, int, int, int)
<< update >>
void repaint()
void setBackground(java.awt.Color)
void setLocation(int, int)
void setSize(int, int)
<< query >>
java.awt.Color getBackground()
Rectangle
<< constructor >>
Rectangle(int, int, int, int)
<< update >>
void repaint()
void setBackground(java.awt.Color)
void setLocation(int, int)
void setSize(int, int)
Triangle
<< constructor >>
Triangle(int, int, int, int, int)
<< update >>
void repaint()
void setBackground(java.awt.Color)
void setLocation(int, int)
void setSize(int, int)
Window
<< constructor >>
Window()
<< update >>
void add(JComponent)
void repaint()
void setBackground(java.awt.Color)
void setLocation(int, int)
void setSize(int, int)
void setTitle(String)