CS163 Computer Science II Summer 2021

Practice Midterm

(v1 4 June 2021)

Questions 9, 10 and 11 are due for credit Wednesday, 16 June at the ***beginning*** of class.

1). Below are two *broken* implementations of `compareTo()` for `Date163.`

(a) For each example below, speculate as to what the programmer was thinking when he wrote the code. (In other words, how could an intelligent programmer make such a mistake?)

(b) For each example below, write a JUnit test demonstrating the bug.

(c) Fix the code.

```
int compareToBroken(SimpleDate other)
{
	if (this.equals(other)) { return 0;}
	if (this.year < other.year || this.month < other.month ||
		this.day < other.day) {
		return -1;
	} else {
		return 1;
	}
}

private int compareToBroken2(SimpleDate other)
{
	if (this.year < other.year) {
		return -1;
	}
	if (this.year > other.year) {
		return 1;
	}
	// At this point, we can assume that this.year == other.year
	if (this.month < other.month) {
		return -1;
	}
	if (this.month > other.month) {
		return 1;
	}

	// At this point, we can assume that this.month == other.month
	if (this.month < other.month) {
		return -1;
	}
	if (this.month > other.month) {
		return 1;
	} else {
		return 0;
	}
}
```

2). In `compareToBroken2()`, would replacing "`if (this.year > other.year)`" with "`else if (this.year > other.year)`" change the behavior of the code? If so, explain why. If not, explain why using "`else if`" instead of "`if`" could be considered better coding style.

3). Think about code-reuse.

(a) Write an `equals()` method for `SimpleDate` that makes use of `compareTo`.
(b) Write a `compareTo()` method that makes use of `equals()`.
(c) Which do you think is a better design? Why?

4). You are considering adding the method `Date163 getNextDay()` to your `Date163` class. (Notice that `getNextDay()` does not modify the `Date163` object.)

(a) Write a set of black-box tests.
(b) Write the code.
(c) Add any necessary white-box tests based on your code.
(d) Show that each line of code is covered by some test.

5). Write a method `Player whoWon(Player[][] board)` that returns which player (if any) has won the game of tic-tac-toe represented by the two-dimensional array board. Player is an `enum` with the values: `NONE`, `P1`, and `P2`. Write the code as if the board could be any square matrix (as opposed to the traditional 3x3 matrix).

6). Write the method described below:

```
public static boolean isSubMatrixConstant(int[][] matrix,
                                           int start_row,
                                           int start_column,
                                           int height,
                                           int width)
```

Determines whether every value in the specified sub-matrix is identical.

Parameters:

`matrix` - the matrix being inspected.
`start_row` - the row in the original matrix where the sub matrix begins.
`start_column` - the column in the original matrix where the sub matrix begins.
`height` - the height of the *sub* matrix.
`width` - the width of the *sub* matrix.

Returns:

`true` if all values in the sub-matrix are identical, `false` otherwise.

For example, calling `isSubMatrixConstant(input, 1, 2, 2, 3)` will return `true`.

4	7	3	2	4
1	6	2	2	2
2	1	2	2	2
4	5	1	4	3
4	2	5	1	4

7). Consider the broken code below.

- What is the result of calling `arrayReverseBroken({1, 2, 3, 4, 5}).`
- What are the bugs (there are two), and how do you fix them?

```
public static void arrayReverseBroken(int[] array)
{
      for (int i = 0; i < array.length; i++) {
            int opposite = array.length - i - 1;
            array[opposite] = array[i];
            array[i] = array[opposite];
      }
}
```

8). Consider the code below:

```
 public class StaticExample2 {
   public static int factor = 2;
   private int a;

   public StaticExample2(int a) {
      this.a = a;
   }

   public int calculate(int input) {
      return (a + input) * factor;
   }

   public void setFactor(int newFactor) {
      factor = newFactor;
   }

   public static void main(String[] args) {
      StaticExample2 obj1 = new StaticExample2(10);
      StaticExample2 obj2 = new StaticExample2(15);

      System.out.println("Line 1: " + obj1.calculate(4));
      System.out.println("Line 2: " + obj2.calculate(4));
      StaticExample2.factor = 3;
      System.out.println("Line 3: " + obj1.calculate(4));
      System.out.println("Line 4: " + obj2.calculate(4));
      obj1.setFactor(-1);
      System.out.println("Line 5: " + obj1.calculate(4));
      System.out.println("Line 6: " + obj2.calculate(4));
   }
}
```

(A) The instance method `calculate` accesses the static variable `factor`. Is this legal in Java?
(B) The instance method `setFactor` modifies the static variable `factor`. Is this legal in Java?
(C) If both highlighted statements are legal, then write the output generated by the `main` method. If they are not legal, then clearly explain why not. In particular, explain why it doesn't make sense for an instance method to access and/or modify a static variable. (Don't just say "Java doesn't allow it.")

```
public class SampleClass {
  public static SampleClass staticMethod(int parameter ) {
  …
  }

  public int[] instanceMethod(int input1,  String input2) {
  …
  }
 }
```

9). Considering the code above

a) Write a snippet of code that calls `staticMethod`. Assume your code is *outside* `SampleClass`.

b) Write a snippet of code that calls `instanceMethod`. Assume your code is *outside* `SampleClass1`.

10). Consider the partial code below:

```
interface ButtonHandler {
  public void handlePush(SimpleButton b);
}

class CounterHandler implements ButtonHandler {
  public void handlePush(SimpleButton b);
}

class MessageHandler implements ButtonHandler {
  public void handlePush(SimpleButton b);
}

public class SimpleButton {
   public void addHandler(ButtonHandler handler) {}

   public static void main(String[] args) {
      SimpleButton b1 = new SimpleButton();

      b1.addHandler(new CounterHandler());
      b1.addHandler(new ButtonHandler());

      MessageHandler mh = new MessageHandler();
      b1.addHandler(mh);

      ButtonHandler bh = new MessageHandler();
      b1.addHandler(bh);

   }
}
```

a) Is the highlighted line of code legal? Explain why or why not.

b) What other lines of code in main are not legal? Why not?

11). Draw the two diagrams described in the comments below and predict the output of the method `practiceReferenceParameter()`. Your diagrams must include *all* variables, not just those currently in scope.

```
public class Counter
{
   private int value;

   public Counter(int init) {
      value = init;
   }

   public String toString() {
      return value + "";
   }

   public int getValue() {
      return value;
   }

   public void update() {
      value++;
   }
}

public static void swap(Counter c1, Counter c2) {
   c1.update();
   c2.update();

   // TO DO: Draw a diagram showing all variables, objects, and references
   // at this point in the code.

   Counter temp = c1;
   c1 = c2;
   c2 = temp;

   // TO DO: Draw a diagram showing all variables, objects, and references
   // at this point in the code.

   temp.update();
}

public static void practiceReferenceParameter() {
   Counter a = new Counter(10);
   Counter b = new Counter(20);

   swap(a, b);
   System.out.println("Line 1: " + a + " " + b);
   a.update();
   b.update();
   System.out.println("Line 2: " + a + " " + b);

   swap(b, a);
   System.out.println("Line 3: " + a + " " + b);
}
```

12). Draw the three diagrams described in the comments below and predict the output of the method `practiceArrayProblem()`.

```
public static void practiceArrayProblem() {
   Counter[] arrayA = { new Counter(0), new Counter(10),
                        new Counter(20), new Counter(30) };
   Counter[] arrayB = new Counter[arrayA.length];
   Counter[] arrayC = new Counter[arrayB.length];

   System.out.println("Line 1: " + arrayA[1] + " " + arrayB[1] + " " + arrayC[1]);
   arrayB = arrayA;
   for (int x = 0; x < arrayB.length; x++) {
      arrayC[x] = new Counter(arrayB[x].getValue());
   }

   // TO DO: Draw a diagram showing all variables, objects, and references
   // at this point in the code.

   System.out.println("Line 2: " + arrayA[1] + " " + arrayB[1] + " " + arrayC[1]);

   arrayA[1].update();
   arrayB[1].update();
   arrayC[1].update();
   System.out.println("Line 3: " + arrayA[1] + " " + arrayB[1] + " " + arrayC[1]);

   System.out.println("Line 4: " + arrayA[2] + " " + arrayB[2] + " " + arrayC[2]);
   arrayA[2] = arrayC[2];

   // TO DO: Draw a diagram showing all variables, objects, and references
   // at this point in the code.

   arrayA[2].update();
   arrayB[2].update();
   arrayC[2].update();
   System.out.println("Line 5: " + arrayA[2] + " " + arrayB[2] + " " + arrayC[2]);

   System.out.println("Line 6: " + arrayA[3] + " " + arrayB[3] + " "+ arrayC[3]);
   arrayA[3] = new Counter(100);
   arrayB[3] = new Counter(200);
   arrayC[3] = new Counter(300);

   // TO DO: Draw a diagram showing all variables, objects, and references
   // at this point in the code.

   arrayA[3].update();
   arrayB[3].update();
   arrayC[3].update();
   System.out.println("Line 7: " + arrayA[3] + " " + arrayB[3] + " "+ arrayC[3]);

}
```

13). The code below is broken. The intent of the method `badTrim` is to take an array of `String`s, and shorten the array so that it doesn't contain any `null` elements.

(1) Explain what is wrong with the code below. (There is more than one problem.)

(2) Write a method that will trim an array as desired. To do this, you may need to modify method's signature. If you do modify the signature, clearly explain why modification is necessary.

```
public static void badTrim(String[] array1) {
      // Count number of non-null elements;
      int nonNull = 0;
      for (String s : array1) {
            if (s != null) {
                  nonNull++;
            }
      }

      // now make an array to hold the non null Strings.
      String[] newArray = new String[nonNull];
      for (int x = 0; x < array1.length; x++) {
            if (array1[x] != null) {
                  newArray[x] = array1[x];
            }
      }

      // make array1 the new array without nulls.
      array1 = newArray;
}
```