# CSE 331 19sp Final Exam 6/11/19

Name ____________________________________________ UW ID# ______________

There are 12 questions worth a total of 125 points. Please budget your time so you get to all of the questions. Keep your answers brief and to the point.

The exam is closed book, closed notes, closed electronics, closed telepathy, etc.

Many of the questions have short solutions, even if the question is somewhat long. Don't be alarmed.

If you don't remember the exact syntax of some command or the format of a command's output, make the best attempt you can. We will make allowances when grading.

**Please do NOT remove any pages from the middle of the exam this time.** There are extra copies at the end of the exam of the full pages of code that you can detach and reference during the exam if you want.

There is an additional blank page with extra space for your answers if you need more room after all the questions but before the detachable pages.

Relax, you are here to learn.

Please wait to turn the page until everyone is told to begin.

Score _________________ / 125

| | |
|---|---|
| 1. ______ / 20 | 8. ______ / 10 |
| 2. ______ / 12 | 9. ______ / 7 |
| 3. ______ / 12 | 10. ______ / 8 |
| 4. ______ / 14 | 11. ______ / 5 |
| 5. ______ / 14 | 12. ______ / 1 |
| 6. ______ / 10 | |
| 7. ______ / 12 | |

# CSE 331 19sp Final Exam 6/11/19

**Question 1.** (20 points) Correct code via proofs. Recall that Java supports multi-dimensional arrays, but they are different than the ones found in C and other languages. A 2-D Java array is really an array of arrays. The top-level array has one entry for each row pointing to the array containing the elements of that row, and each row can have a different length. So, for example, if we declare `int a2d[][] = {{1,2,3}, {4}, {}, {5,6,7,8}};` we have an array with 4 rows. The first row has three elements (1, 2, 3), the second row has one element (1), the third row is empty, and the fourth row has four elements. An individual element of an array can be referenced using the notation `a[r][c]`, and `a[r]` by itself references the array that is stored in row `r`. The expression `a.length` returns the number of rows in array `a`, and `a[r].length` returns the number of elements in row `r`.

(a) (12 points) The following method is alleged to return the total number of elements in a 2-D Java array. Provide a suitable loop invariant `inv` and assertions to prove that it returns the correct result.

```
public static int numElts(int[][] lst) {

   { pre: ______________________________________________ }
   int num = 0;
   int i = 0;

   { inv: ______________________________________________ }
   while (i != lst.length) {

      { ____________________________________________ }
      num += lst[i].length;

      { ____________________________________________ }
      i++;

      { ____________________________________________ }
   }

   { post: _____________________________________________ }

   return num;
}
```

(continued on next page)

**Question 1. (cont.)** (b) (8 points) The following method, `flatten`, returns all of the elements of a 2-D array in a single 1-D array. For example, the result of `flatten` on our example 2-D array containing `{{1,2,3}, {4}, {}, {5,6,7,8}}` is the 1-D array whose contents are `{1,2,3,4,5,6,7,8}`. Your only job in this part of the question is to provide a suitable loop invariant `inv` to complete the proof. All of the rest of the code and proof are provided for you. In the postcondition we write `a[0..i-1]` to mean the subsequence of `a` containing `a[0]` up to and including `a[i-1]`. This is not legal Java code, but is helpful for writing assertions in the proof.

Write the appropriate loop invariant in the blank line right before the start of the loop.

```
public static int[] flatten(int[][] lst) {
  { pre: lst != null && for all 0<=i<lst.length, lst[i] != null }
  int[] res = new int[numElts(lst)];
  int iOuter = 0;
  int iInner = 0;
  int iRes = 0;



  { inv: ________________________________________________ }
  while (iRes != res.length) {
    if (iInner != lst[iOuter].length) {
      res[iRes] = lst[iOuter][iInner];
      iInner++;
      iRes++;
    } else {
      iOuter++;
      iInner = 0;
    }
  }
  { post: for all i, j, res[numElts(lst[0..i - 1]) + j] = lst[i][j] }
  return res;
}
```

Specifications and debugging. The following code implements a very simple queue ADT. In the usual CSE 331 exam style, it is woefully lacking in various forms of documentation, it may not be correct (i.e., it may be buggy), and there may be other problems (or maybe it actually does work). Please **leave this page in the exam.** An extra copy of this page is included at the end of the exam that you can remove for convenience while working.

```
// a finite queue of characters
// add() can only be called a finite number of times before the
// queue is "used up", no matter how many times get() is called
public class CharQueue {
  // instance variables
  private char[] chars;
  private int current;  // current position
  private int end;      // end of data

  // construct new CharQueue with given capacity
  public CharQueue(int n) {
    chars = new char[n];
    current = 0;
    end = 0;
  }

  // append new character to queue if room.
  // return true if character appended, otherwise return false.
  public boolean add(char ch) {
    if (end < chars.length) {
      chars[end] = ch;
      end = end + 1;
      return true;
    } else {
      return false;
    }
  }

  // return oldest character in the queue and remove it
  // throw NoSuchElementException if queue contains no characters
  public char get() {
    if (current <= end) {
      char result = chars[current];
      current++;
      return result;
    } else {
      throw new NoSuchElementException();
    }
  }
}
```

**Do not remove this page from the exam,** but feel free to tear off the copy at the end of the exam. Continue with questions about this code on the next page.

**Question 2.** (12 points) Class specification. This class is lacking the appropriate CSE331-style documentation. For this question, supply a proper abstract description of the class, a rep invariant, and an abstraction function. You should base your specifications on the intended behavior inferred from the original code and comments.

(a) (4 points) Give an appropriate abstract description of the class that should appear in the JavaDoc comment right before the first line of the class.

```
/**
 *
 *
 *
 *
 *
 *
 *
 *
 *
 */
public class CharQueue { ... }
```

(b) (5 points) Give a suitable representation invariant for this class. You should use the existing instance variables that are already present in the code (i.e., you should use the rep that is already there).

(c) (3 points) Give a suitable abstraction function for this class. Your answer should use information from your answers to parts (a) and (b) of the question as needed.

**Question 3.** (12 points, 6 each) Method specifications. Give appropriate CSE331-style JavaDoc specifications for methods `add` and `get`. For CSE331 custom tags like @requires (or any others) you are free to write @requires or @spec.requires. You should base your specifications on the intended behavior inferred from the original code and comments.

```
/**
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 */
public boolean add(char ch) { ... }

/**
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 */
public char get() { ... }
```

**Question 4.** (14 points) There's a bug in the code!! (well, at least one) Using the systematic debugging strategies discussed in CSE 331 we want to diagnose and fix it. Answer the following questions about the bug. If there's more than one bug, pick a particular one and answer the questions for it.

(a) (4 points) Symptom and hypothesis. What is the observed failure and what is your hypothesis about what the defect is and where is it located?

(b) (6 points) Write a JUnit test that demonstrates the presence of the defect by failing when executed with the original code.

**Question 4. (cont.)** (c) (4 points) Describe how to repair the defect to fix the problem. After the repair, the test from part (b) should succeed. You can either give a very precise description of what to fix in the original code, or rewrite the original code for the modified method(s) below showing how to repair the defect. Be sure it is clear what you have changed. If there is more than one possible way to fix the problem, pick the most reasonable one.

It's summer and it's time for picnics! But since it's a CSE 331 picnic, we have to write some code to keep track of the food. Here is the code. Please **leave this page in the exam.** An extra copy of this page is included at the end of the exam that you can remove for convenience while working. Answer questions about this code on the next few pages.

```
/** food for picnics */
abstract class Food {
  /** request n servings of this Food item */
  abstract public void order(int n);
}

class Hotdog extends Food {
  public void order(int n)
                         { System.out.println(n + " hotdogs"); }
}

class Dessert extends Food {
  public void order(int n)
                         { System.out.println(n + " desserts"); }
  public void order()    { this.order(1); }
}

class Icecream extends Dessert {
  public void order()      { this.order(2); }
  public void order(int n) { System.out.println(n + " scoops"); }
}

class Cake extends Dessert {
  public void order()
                  { this.order(2); System.out.println("cake"); }
}

class ChocolateCake extends Cake {
  public void order(int n)
                   { System.out.println(n + " chocolate cake"); }
}
```

**Do not remove this page from the exam,** but feel free to tear off the copy at the end of the exam. Continue with questions about this code on the next page.

**Question 5.** (14 points, 2 points each). Here are several groups of statements that might be found in a program that uses the classes on the previous page. For each group, if the statements compile and execute successfully without any errors, write the output that is produced. If there is an error, explain in a sentence what is wrong.

(a)
```
Food meat = new Hotdog();
meat.order(4);
```

(b)
```
Dessert yum = new ChocolateCake();
yum.order();
```

(c)
```
Cake yummy = new ChocolateCake();
yummy.order(2);
```

(d)
```
Food strawberry = new Icecream();
strawberry.order();
```

(e)
```
Icecream vanilla = new Icecream();
vanilla.order();
```

(f)
```
Dessert chocolate = new Icecream();
chocolate.order();
```

(g)
```
Food lemon = new Cake();
lemon.order(3);
```

**Question 6.** (10 points, 1 each) And now for the dreaded generics question. ☺ Using the `Food` class hierarchy from the previous pages, assume we have the following variables:

```
Object obj; Food food; Hotdog hot; Dessert des;
Icecream ice; Cake cake; ChocolateCake choc;

List<? extends Dessert> extd;
List<? extends Cake> extc;
List<? super Cake> supc;
```

For each of the following, circle OK if the statement has correct Java types and will compile without type-checking errors; circle ERROR if there is some sort of type error.

OK ERROR `extd.add(des);`

OK ERROR `extd.add(cake);`

OK ERROR `supc.add(cake);`

OK ERROR `supc.add(choc);`

OK ERROR `supc.add(des);`

OK ERROR `cake = extc.get(1);`

OK ERROR `cake = supc.get(1);`

OK ERROR `choc = extc.get(1);`

OK ERROR `choc = supc.get(1);`

OK ERROR `food = supc.get(1);`

**Question 7**. (12 points) A little bit of React. We have discovered the following fragment of a React web app. `App` is the top-level component when the page is rendered. This code runs without errors when the page is rendered.

```
class App extends Component {
    constructor(props) {
        super(props);
        this.state = {
            favorite: "Raspberry"
        };
    }
    render() {
        return (
            <div>
                <Basket fruit={this.state.favorite} />
                <Basket fruit={"Apple"} veggie={"Broccoli"} />
            </div>
        );
    }
}
export default App;

class Basket extends Component {
    constructor(props) {
        super(props);
        console.log(this.props);
        this.state = {
            fruit: "Blueberry"
        };
    }
    render() {
        return (
            <p>Basket of: {this.props.fruit}</p>
        );
    }
}
export default Basket;
```

(a) (6 points) What output is written to the console log when this code is executed?

(b) (6 points) What html code is generated when this code is executed?

**Question 8.** (10 points, 2 each) Design patterns. In the projects we built this quarter we wound up using a lot of classic design patterns and strategies, even though we didn't make a big point at the time about which ones we were using and where. But now, looking back, it's interesting to identify where the patterns were used. Here is a list of the main patterns and design organizations (like MVC, which is not, strictly speaking, a pattern but is a key idea). We didn't use all of them but we did use several.

Adapter, Builder, Composite, Decorator, Dependency Injection, Factory method, Factory class/object (often called Abstract Factory), Iterator, Intern, Interpreter, Model-View-Controller (MVC), Observer, Procedural, Prototype, Proxy, Singleton, Strategy, Visitor

For each of the situations below, identify the most specific pattern or design principle used in that situation. You only need to write the pattern name, you do not need to explain further. If there is more than one possible answer, pick the one that is the best match.

(a) Registering a function to be called when a user event happens (like a click on a GUI button)

(b) Separating the main graph data and algorithms that process it from user interface classes and objects.

(c) Isolating (separating) the search algorithms for the graph (Dijkstra's, and breadth-first search/BFS) from the basic classes that contained the graph data (nodes and edges).

(d) Creating the Java "bean" parser by calling successive methods to add details and adjust the created object step-by-step rather than having a single constructor do all the work with one constructor invocation.

(e) Create an object by calling a single method to return the appropriate object rather than using the java `new` operator.

$$\begin{tikzcd}
 & & A \arrow[dll] \arrow[d] \arrow[drr] & & \\
B \arrow[dr] & & C \arrow[dl] \arrow[dr] & & D \arrow[dd] \\
 & E & & F \arrow[dr] & \\
 & & & & G
\end{tikzcd}$$

**Question 9.** (7 points) System integration. When building a large system, there are two common strategies for the order in which to implement, combine, and test the different parts of the system: top-down and bottom-up. These two strategies have different characteristics and strengths. Please keep your answers short. You are welcome to use the diagram to the right in your explanations if you like; it is mainly intended as a reminder of the topic from the lecture slides.

(a) (2 points) Describe **one major** (not minor) **advantage of** the **top-down** strategy compared to bottom-up.

(b) (2 points) Describe **one major** (not minor) **advantage of** the **bottom-up** strategy compared to top-down.

(c) (3 points) Assuming a project is an ordinary one, not relying on unexpected technological innovations or other unusual breakthrough, what is often the best overall strategy for building and integrating the parts: top-down, bottom-up, or some combination of the two? Give a brief justification for your answer.

A couple of short questions about types to wrap up.

**Question 10.** (8 points). In the Java language, type `String` is a Java subtype of `Object`. It also is true that the array type `String[]` is a Java subtype of `Object[]`.

(a) (5 points) From what we know about correct (true) specification subtyping, is `String[]` actually a true specification subtype of `Object[]`? Give a brief justification for your answer, either explaining why this subtyping relationship is correct, or give an example showing how this allows programs to typecheck with no errors at compile time and yet have a type error during execution.

(b) (3 points) Why did the Java designers decide to specify that `String[]` is a Java subtype of `Object[]`? (You can explain this in terms of your answer to part (a) if that is useful.)

**Question 11.** (5 points) Some generic puzzles. In Java a method can have generic type parameters even if it is declared in a class that is not itself generic. For instance, the following method definition and declarations are legal:

```
public class C {
  static <T> void func(T param, T[] array) {
    T var;
    T[] vec;
    // code omitted
    ...
  }
```

But it is not allowed to create objects or arrays of a generic type like `T` inside such a function. For instance, both of the following two lines of code are rejected at compile time if they are added to the above method:

```
    var = new T();
    vec = new T[10];
```

Why is this not possible? What is the technical reason that neither of these statements are allowed?

**Question 12.** (1 free point) (All reasonable answers receive the point. All answers are reasonable as long as there is an answer. ☺ )

Draw a picture of something that you plan to do this summer!

*Congratulations from the CSE 331 staff!*
*Have a great summer and see you in the fall!!*

**Additional space for answers if needed. Please indicate clearly which questions you are answering here, and also be sure to indicate on the original page that the rest of the answer can be found here.**