Software Engineering, Quiz 2, November 13, 2019

Name: solutions

Signature: solutions

Instructions: Please fill out your name and signature in the spaces provided above. Read the questions carefully. Some of the questions ask for multiple responses. After the test has begun do not ask to leave for any reason except if you plan to submit your paper. Answers that are not readable will result in lost points. Good luck.

All figures referenced in individual problems appear at the *end* of the test.

1. How many views are proposed in the $4+1$ view model of software architecture? What are they?

   There are 5 views in Kruchten's 4+1 view model: logical, process, development, physical, and scenario. 1/5 for each view.

2. In the $4+1$ model, what view is the best place to include UML class diagrams? What view is most appropriate for UML activity diagrams?

   For UML class diagrams the logical view. For UML activity diagrams the process view.

3. What is "waste" in the context of lean software development? Give 2 common kinds of waste in software.

   Waste is any activity/work that does not add customer value or add knowledge or contribute to learning about how to deliver value.

4. What does a *value stream map* show? Draw an example.

   A value stream map shows when time was wasted in a project. Slide set se_5_15 contains an illustration.

5. Name one example of a *code smell* and provide a brief description of it.

   We discussed 10 code smells in the lecture. See slide set se_5_21 for definitions.

   - *Mysterious name*
   - *Duplicated Code*
   - *Long Method/Function*
   - *Long Parameter Lists*
   - *Global Data/Mutable Data*
   - *Divergent Change*
   - *Feature Envy*
   - *Data Clumps*

- *Primitive Obsession*
- *Comments**

6. What terms are used to differentiate type systems that do type checking at compile-time versus run-time?

   static versus *dynamic*

7. Where is it appropriate to call the `wait()` method on an object? What will this method do?

   It is appropriate to call `wait()` in a synchronized method or a synchronized block.

8. Which interface from the Java concurrency utilities does the class `newCachedThreadPool()` implement?

   `newCachedThreadPool()` implements the `ExecutorService` interface.

9. In the context of concurrent programming, what is *deadlock*?

   Deadlock occurs when two threads are blocked, each waiting on a lock held by the other.

10. For Java Threads, what is the primary difference between the *new* state and the *runnable* state?

    The *new* state is the state of the thread just after it is created. The *runnable* state indicates that the thread has been queued for potential execution.

11. A `BiPredicate` is a functional interface defined in Java. It is a version of a predicate that takes two arguments. The declaration is shown below. Write a `BiPredicate` that tests if the first characters of two strings are the same.

```
@FunctionalInterface public class BiPredicate<T, U> {
   boolean test(T t, U u);
}
```

```
BiPredicate<String, String> biPredicate = (s1,s2) -> s1.substring(0, 1).equals(s2.substring(0, 1));
```

12. Consider the lambda expression below. Rewrite this assignment using a *method reference*.

```
Supplier<LocalDate> supplier = () -> LocalDate.now();
```

```
Supplier<LocalDate> supplier = LocalDate::now;
```

13. Suppose you are designing a web application and want to render some of the page with a script. What is the problem with the following example? Show how to resolve the problem

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Example</title>
<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>
<script>
$("#mydiv").html("Hello, world");
</script>
</head>
<body>
<h2>Web Application</h2>

<div id="mydiv"></div>
</body>
</html>
```

    The problem is updating the html of the div. Since the script is in the `head` element, the selection may run before the div has actually been loaded into the DOM. There are two reasonable solutions: move the script to the end of `body` or wrap the script in jQuery `ready()` function.

14. What will be output to the console from the following Javascript code?

```
function Person() {
  this.name = function () {return "default";};
}

var person1 = new Person();

Person.prototype.name = function () { return "empty"; };

var person2 = new Person();

console.log(person1.name());
console.log(person2.name());
```

Since the name function is defined within the constructor, both calls to the `name()` method will result in “default”.

15. In Javascript, when calling `Promise` as a constructor, what should be passed as the argument? Give a detailed description of the components of the argument.

    When we creating `Promise` in this way, the constructor argument should be a function with two arguments for resolve and reject.

16. The `fetch` API is a newer interface for fetching resources included in some modern browsers. An example of how the basic API is called is shown below. The `fetch` method returns a Javascript Promise. Assume that response resolves to an object with a `text()` method that also returns a promise (for the text of the response). Show how to chain together a call the `fetch` call and a subsequent call to `text()` on the result using the promise API.

```
response.then(function (res) {
    return res.text();
}).then(function (t) {
    console.log(t + "!");
});
```

17. Refactor the following Javascript code using *Inline Function* to remove the call to `gatherCustomerData`.

```
function reportLines(aCustomer) {
  const lines = [];
  gatherCustomerData(lines, aCustomer);
  return lines;
}
function gatherCustomerData(out, aCustomer) {
  out.push(["name", aCustomer.name]);
  out.push(["location", aCustomer.location]);
}
```

To inline function we replace a function call with an inline block of code. Since the function has arguments we have to be careful of variable names.

```
function reportLines(aCustomer) {
  const lines = [];
  lines.push(["name", aCustomer.name]);
  lines.push(["location", aCustomer.location]);
  return lines;
}
```

18. Write a simple program using lambda expressions and streams that processes a stream of strings (see below) and does the following: the stream of strings should be transformed into a stream of string arrays. The first element of the string array should be the original string and the second element of the string array should be the first letter of the string (see comments in code). Also, you should not need looping, only stream methods and lambda expressions.

```
public static void main(String args[]) {
    Stream<String> s = Stream.of("Alice","Bob","Charlie");
    // your code here
    // Convert to Stream of String[]
    // {Alice, Bob, Charlie}
    // becomes
    // {[Alice, A], [Bob, B], [Charlie, C]}
}
```

One possiblity

```
public static void main(String args[]) {
    Stream<String> s = Stream.of("Alice","Bob","Charlie");

    s.map(e -> new String[] {e.substring(0, 1), e});
}
```

19. Consider the class `VarianceExample` below. Find all mistakes, these may be errors at either compile or at run time.
    Errors are indicated with comments

```
public class VarianceExample {

    public static void main(String[] args) {
        List<Cat> catList = new ArrayList<Cat>();
        List<Object> objectList = new ArrayList<Object>();

        catList.add(new Cat());

        getCovariantList(catList);
        getContravariantList(objectList);

        getCovariantList(objectList); // compile error
        getContravariantList(catList); // compile error

        List<? extends Animal> animalList = catList;

        Animal[] arr = new Cat[10];
        arr[0] = new Animal(); // runtime error
        arr[1] = new Cat();

        for (int n = 0; n < 10; n++) {
            System.out.println(arr[n]);
        }
    }

    static void getCovariantList(List<? extends Animal> myList) {
        myList.add(new Cat()); // compile error
        Animal a = myList.get(0);
    }

    static void getContravariantList(List<? super Animal> myList) {
        myList.add(new Cat());
        Animal a = myList.get(0); // compile error
    }
}

class Animal {}
class Cat extends Animal {}
```

20. What is a reasonable lower bound for the running time of the following Java program? Explain why.

```
public class MyClass {
    public static void main(String[] args) {
        ExecutorService service = Executors.newFixedThreadPool(2);

        for (int j = 0; j < 30; j++) {
            service.submit(new MyTask(j));
        }

        service.shutdown();
    }
}

class MyTask implements Runnable {
    public int i;

    public MyTask(int i) {
        this.i = i;
    }

    @Override
    public void run() {
        try {
            Thread.sleep(1000);
        } catch (Exception e) {
            //
        }
        System.out.println(System.nanoTime());
    }

}
```

A reasonable lower bound for the running time is 15 seconds. Each task takes about 1 second to run, and there are 30 tasks total. The thread pool is fixed with two threads, which means that it can run at most two tasks concurrently.