CSSE 220 – Object-Oriented Software Development

Exam 1 Topics

The exam has two parts:

- A closed-book, paper-and-pencil section. Topics for this section are drawn *only* from the first two lists below. Questions might be short answer, fill-in-the-blank, multiple-choice, true/false, or problems whose answer is a short code fragment.
- An open-book, on-the-computer section. You write code and turn it in via your individual repository. See the third topic list below.

For the closed-book portion, you can use a single 8.5 inch by 11 inch "cheat sheet" (back and front, with whatever you want on it, handwritten or typed). You may work with others to make your cheat sheet, but you will probably find it most useful if you do much or all of it yourself.

For the open-book portion, you may use any written source, anything you can find on the internet, and your computer. You may not communicate with any human being except your instructor in any way during the exam, however.

Closed-book topics, things you should know:

1. Given a class, identify the:
   a. ***Fields***
   b. ***Constructors***
   c. ***Methods***
   d. ***Type*** of variable *blah*
   e. ***Visibility*** of variable *blah*
   f. ***Return type*** of method *blah*
   g. Number of ***parameters*** of method *blah*, and the types of those parameters

   Identifying elements of a class
2. What style do we use for ***method names***? ***Class names***? Global ***constants***? ***Getter's***? ***Setter's***? ***Predicate*** methods?
3. What is the ***signature*** of a method?
4. What is the ***scope*** of a field? Of a parameter? Of a local variable that is not a parameter?
5. What is the ***lifetime*** of a field? Of a parameter? Of a local variable that is not a parameter?

   Variables: scope, lifetime, type
6. List the seven ***primitive types***.
7. What is the most commonly used ***type for whole numbers***? For ***floating-point numbers***?
8. What is the ***type for a string of characters***, like "When the moon is in the Seventh House"?
9. How do ***object-type variables*** differ from ***primitive-type variables***?
10. True or false: In Java, every variable has a declared type.
11. What keyword indicates that a ***method does not return a value***?

Operators: +, == (and *equals*)

12. The ***plus sign*** (+) has two meanings: what are they? How does the compiler know which to use?
13. Explain the ***difference between the == operator and the* equals *method***. When should you use the == operator, and when the *equals* method?
14. True or false: We can define a method in Java that is not associated with any class.

Static, instance, *this* keyword

15. How does a ***static method*** differ from a non-static method?
16. How does a ***static field*** differ from a non-static field?
17. What is the difference between a ***class*** and an ***instance*** of a class? What word do we use for instances of classes (hint: begins with an "o")?
18. What does the ***final*** keyword mean when applied to a variable, as in the following?

```
final int DEFAULT_RADIUS = 25;
```

19. What is the ***implicit parameter*** in the following? The ***explicit parameters***?

```
father.replace('a', '8');
```

20. In a class definition, what keyword refers to the ***implicit parameter***?
21. What does the keyword ***this*** mean, as in the following?

```
return this.widthAndHeight;
```

22. What does the keyword ***this*** mean, as in the following?

```
world.addBall(this);
```

23. What are two reasons why we use "this" to refer to the implicit argument (as in *this.color*) when implementing a constructor or method?

Constructors

24. What does a ***constructor*** do?
25. How do you ***invoke a constructor***?
26. True or false: In Java, the constructors for a class always have the same name as the class.
27. Explain what space is allocated by the following (hint: space for ***5*** things are allocated):

```
Rectangle box = new Rectangle(5, 10, 20, 30);
```

28. What are the four ***visibility*** levels? For each, what restrictions does it impose?

    (You can skip *protected* and *package* visibility for this exam; we'll cover them later in the course. That leaves two of the four visibility levels – they are …)

Visibility

29. Generally, what ***visibility*** should ***fields*** have? Why?
30. Generally, what ***visibility*** should ***methods*** have? Why?
31. Generally, what ***visibility*** should ***constructors*** have? Why?
32. Generally, what ***visibility*** should ***classes*** have? Why?

Inplements an interface

Extends a class

33. If class B ***implements interface X***, what does that imply?
34. If class B ***extends class Y***, what does that imply?

cast

35. What does the word ***cast*** mean?
36. What is the ***notation for doing a cast***?
37. ***Why do you need a cast*** to assign a *double* value to an *int* variable, but not the other way around? When you do such a cast, what happens to the fractional part of the *double*?

UML class diagrams

38. Give at least two reasons why ***UML class diagrams*** are useful.
39. What is the UML notation for a class? Give an example.
40. What is the UML notation for ***is-a (extends)***?
41. What is the UML notation for ***is-a (implements)***?
42. What is the UML notation for ***has-a***?
43. What is the UML notation for ***depends-on***?

Software engineering processes and principles.

Testing.

44. What is ***encapsulation*** in object-oriented software? What two things are encapsulated inside an object?
45. What is ***version control***? Why is it useful?
46. What is ***developing by using documented stubs***? Why is it useful?
47. What is ***test-first development***? Why is it useful?
48. What is a ***unit test***? Why is unit testing useful?
49. What is ***test coverage***?
50. What is ***regression testing***?
51. What ***annotation*** indicates that a method is a ***JUnit 4 test***?
52. What does the ***@Before*** annotation in a JUnit test mean?
53. What is ***pair programming***? Why is it useful?
54. What is ***iterative enhancement***? Why is it useful?
55. What are ***magic numbers***? Why do you want to avoid them? How do you avoid them?
56. What are ***parallel arrays***? Why is their use generally a bad idea?
57. In Eclipse, what keystroke combination ***formats your code*** according to an accepted style?
58. Define each of the following key concepts in debugging: ***breakpoint***, ***single stepping***, and ***inspecting variables***.

Closed-book topics, things you should be able to do:

Constructing and using objects. Assignment. API's.

1. ***Construct*** an object.
2. ***Use an object***'s methods and/or fields.
3. Use ***assignment***.
4. Read, understand and use the ***API*** (Application Programming Interface) of a class that you have not seen before.
5. Explain the implications of object variables being ***references***. Draw ***box-and-pointer diagrams*** to illustrate assignment involving object variables.
6. Output to the console by using the ***System.out*** object. Do ***formatted output*** with *printf*.
7. Read from the console by using the ***Scanner*** class.

Console input and output

8. Use ***conditional*** statements, including those that use ***comparison operators*** and/or ***boolean operators***. The ***selection (ternary) operator***.
9. Write *for* and *while* ***loops***.
10. Write ***nested loops***.

Conditionals and loops

11. Process a loop using a ***sentinel value***. Use ***break*** and/or ***continue*** in a loop.
12. Use an ***array*** of a ***generic type***: declare, fill, iterate through (***new style*** and old style), get/set elements (by iterating and/or by directly accessing elements).
13. Use an ***ArrayList*** of a ***generic type***: declare, fill, iterate through (***new style*** and old style), get/set elements (by iterating and/or by directly accessing elements).
14. ***Copy an array*** or ArrayList, by using a loop or by calling the appropriate method from the ***System***, ***Arrays*** and ***Collections*** classes.
15. Choose ***whether to use an array or an ArrayList***.
16. Use the ***counting***, ***summing***, **min/max** and ***histogram*** looping patterns. Use the ***loop-and-a-half*** pattern.

Arrays and ArrayList's

17. Use ***wrapper classes*** and ***autoboxing***.
18. Write ***switch*** statements, including those that use an ***enum*** object.
19. Write and use ***enumerated types***.

Implementing classes

20. ***Implement a class***, including implementing an ***interface*** and/or ***extending*** a class. Determine what fields, constructors and methods are necessary to meet the specification, and implement them. (For this exam, you need only a superficial understanding of how to *extend* a class; we will cover that topic in more depth later in the course.)
21. ***Implement a UML class diagram.*** (For this exam, you need only a superficial understanding of how to implement objects that *interact* with each other; will cover that topic in more depth later in the course.)
22. Choose good ***unit tests*** and implement them in JUnit 4.
23. Read a ***stack trace*** and indicate: What statement caused the error? What was the last statement executed in *your* code before the error occurred?

On-the-computer topics: You should be able to do the following:

1. ***Choose and implement JUnit 4 tests*** for a class which is specified either in ordinary language or by a UML class diagram.
2. Write ***Javadoc comments***.
3. ***Implement a class***, including:
    a. Choosing and implementing the necessary fields
    b. Choosing and implementing the necessary constructors
    c. Choosing and implementing the necessary methods
4. Read and ***apply an API***, even one that you have not seen before.
5. Implement ***loops*** and use ***arrays*** and ***ArrayLists***.
6. Implement a program that uses the Swing API to ***display a window and draw/fill shapes and/or text*** on it. Set the color and/or font used in such drawing.
7. Use the ***debugger*** to set a breakpoint; launch a program with the debugger; inspect a variable; and single step over the current line of code.
8. Use ***Subclipse*** to ***checkout*** a project from your individual repository (or other repository as specified); ***commit*** changes to that project; and get ***updates*** to the project from teammates.
9. Use ***good style***. Control-Shift-F covers most (not all) style issues.