**CS 4321 - Video Lecture Expectations**

**Video P4L5: Software Refactoring (40 minutes)**

Watch video P4L5 on Udacity. Answer the questions below and submit on Blazeview (HW- P4L5). Instructions:

* Do not remove the questions.
* You can provide the answer(s) where the blank is, but preserve the underline (or use a different color for the answers)
* Or, you can provide the answers below the questions. For example, you could type: Answer: x, y, z.

**Questions to be answered**

1. In software, a \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_is the indication that there might be something wrong with the code that might call for the application of refactoring.
2. A refactored program has an improved \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_but with the same \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
3. The goal of refactoring is to transform an existing program to be more \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, and \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
4. We can “guarantee” that a refactoring is behavior preserving by \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.
5. Testing can only show the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ of defects but cannot demonstrate their \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.
6. Three reasons why code needs to be refactored: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.
7. The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ refactoring should be considered when a subclass becomes too similar to its superclass and might not be adding much value to the system. In this case, it is a good idea to merge the classes together.
8. The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ refactoring replaces multiple conditionals that achieve the same results with a method that combines the conditionals.
9. If a conditional is too complex, it may tell you \_\_\_\_ happens but obscure \_\_\_\_ it happens.
10. The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ refactoring uses suitably named methods to evaluate conditionals and to implement the body of the THEN and ELSE parts of the conditional.
11. The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ refactoring should be considered when we have a class that does the work of more than one class, with many methods and many instance variables.
12. The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ refactoring should be considered when a class is not doing much in which case we merge the class into another class.
13. The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ refactoring should be considered when you have a cohesive code fragment in a large method.
14. The extract method refactoring cannot be used directly when the method has more than 1 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.
15. The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ refactoring is appropriate when: (a) you have a cohesive code fragment in a large method, (b) duplicate code in two or more methods, (c) when a method is highly coupled with a class other than the one where it is defined.
16. List three refactoring risks: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
17. List three costs of refactoring: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
18. List three reasons not to refactor: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
19. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ are usually not bugs and don’t prevent the program from functioning. They indicate weaknesses in the design that might cause problems during maintenance.
20. The duplicated code bad smell indicates that the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ refactoring should be used.
21. The long method bad smell indicates that the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ or the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (or both) refactoring should be used.
22. The large class bad smell indicates that the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ refactoring should be used.
23. The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ bad smell occurs when you make a change to a method and end of having to make many small changes in different methods/classes.
24. The shotgun surgery bad smell indicates that either of these two refactorings might be applicable: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ or \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.
25. The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ bad smell is used to describe the situation where a method in class 1 uses the fields/methods in class 2 extensively to do some computation rather than asking class 2 to do the computation itself.

A brief discussion of feature envy: <https://goo.gl/gHyFcL>