

C Sc 335 Section 11- or 12- April **ANSWERS** Low and High Design Considerations

- 1) **b** When a method name does not reveal its purpose, change the name of the method. Use the refactoring "**Rename Method**" to change `empty` to `isEmpty` (unless `empty` does really *empty* the container). BTW: The C++ STL has many collection classes (`stack`, `queue`, `list`) where `empty` does not mean empty, it returns true if the stack `isEmpty`. Also, `pop` can modify and return as long as it is documented. You could argue that `pop` should only modify, and some people do.
- 2) **b** Whenever there is a public field, make it private and provide accessors. This prevents other object from modifying a field, which would be the worst form of coupling. Use the "**Encapsulate Field**" refactoring (Eclipse has this refactoring)
- 3) **a** The while loop is difficult to read, trace, and understand. And it is longer. The recursive solution can be easier to understand. Use the refactoring **Replace iteration with Recursion** to make more readable code. BTW: Worried about all of those method calls at runtime? A compiler can make the code to the left run as fast as the code to the right. And you should wait until you profile the code to see if this is a bottleneck
- 4) **b** When you have a complicated conditional (if-then-else) statement, use the refactoring "**Decompose Conditional**". Extract methods from the Boolean condition
- 4B) **Yes** Apply "**Rename Method**" and have the method return the logical negation, then apply "**Reverse Conditional**"
- 5) **a** Whenever a method returns an object that needs to be downcasted by its callers, use the "**Encapsulate Downcast**" refactoring to move the downcast to within the method.
- 5b) Use generics so you do not have to cast while providing type safety to avoid the wrong types from being added to or returned from a collection. With generics you get a compiletime error rather than a class cast exception at runtime.
- 6) **b** The design on the left has one class doing work that should be done by two. Use the "**Extract Class**" refactoring to create a new class and move the relevant fields and methods from the old class into the new class.
- 7) **b** If you have a code fragment that can be grouped together, use **Extract Method** to turn the fragment into a method whose name explains the purpose of the method.
- 8) **b** When a method is not used by any other class, use the refactoring "**Hide Method**" to make the method private. Why: Do not clutter the public interface with methods the user does not need or is not going to be interested in.
- 9) **b** Magic numbers often don't reveal their meaning very well (we usually know 3.14, but what is 0.045 (BTW it is the VA state sales tax, I think) If you have a literal number with a particular meaning, create a constant, name it after the meaning, and replace the number with it. Apply "Replace Magic Number with Symbolic Constant".
- 10) **b** Class with low coupling reduce the risk of cascading changes when a change is made to one.
- 11) **Neither** Use whatever coding style your team, project or instructor demand
- 12) **a** Because the code uses two indirect objects to get the tail wagging
- 13) When a method send a message to one of its parameters
- 14) An instance variable (there are 4 possible answers to this question)
- 15) Rigidity: When one change requires many other changes
- 16) Strategy is Open-Closed Principle because implementing a new Strategy does not require a change to the classes that depend upon it.
- 17) To allow for continued use of polymorphism. It is often used in conjunction with other design patterns.