CS 501 Final Exam Review

1. Method overloading
2. Javadoc
3. Exceptions
    a. Program output when an uncaught exception is thrown
    b. Stack trace
    c. Try block
    d. Catch block
    e. Finally block
4. Values vs references
    a. Understand the difference in assigning variables
    b. Understand the difference in parameter passing
    c. Understand why an “equals” method is necessary
5. Arrays
6. Using Generic classes
7. ArrayList
    a. How to declare and initialize
    b. Element must be a class type, not a primitive type (instead use wrapper classes)
    c. Methods
        i. add(E element)
        ii. get(int index)
        iii. set(int index, E element)
        iv. remove(int index)
        v. size()
        vi. isEmpty()
    d. Understand usage in simple programs as in Chapter 12 (Listing 12-1 and 12-2)
8. IO Streams
9. Checked vs Unchecked exceptions
10. Class Hierarchies
    a. Creating a subclass of a superclass
    b. The keyword “extends”
    c. The keyword “super”
        i. Use in subclass constructor
        ii. Use in subclass methods
    d. “protected” vs “private” data fields
    e. Is-a versus Has-a relationships for code re-use
    f. Assigning superclass variable to subclass instance is valid
    g. Assigning subclass variable to superclass instance requires explicit cast operator and may throw ClassCastException
11. Polymorphism
    a. Method overriding versus overloading

  - i. To override a method, the subclass method signature (method name and parameter types) must match the superclass method signature exactly
  - ii. Use “@Override” annotation to guarantee an override is valid
- b. Invoking a method on a superclass variable assigned to a subclass instance
  - i. The method must be declared (not necessarily implemented) in the superclass, else compile-time error
  - ii. If the method is overridden in the subclass, then the subclass implementation will run, else the superclass implementation will run
- c. Interfaces
  - i. How to define
  - ii. Can only have abstract methods (no body)
  - iii. All methods are automatically public, abstract
  - iv. All fields are automatically public, static, and final
  - v. Cannot be instantiated
  - vi. Define subclass by using the keyword “implements” unless the subclass is itself an interface, in which case you use the keyword “extends”
  - vii. Allows multiple inheritance
  - viii. Does not allow code to be re-used
- d. Abstract classes
  - i. How to define
  - ii. Cannot be instantiated
  - iii. Can have abstract methods (no body)
  - iv. Can implement some methods
  - v. Define subclass by using the keyword “extends”
  - vi. Does not allow multiple inheritance
  - vii. Allows code to be re-used
- e. Concrete class (also called Actual class)
  - i. Implements all its methods
  - ii. Can be instantiated
  - iii. Define subclass by using the keyword “extends”
  - iv. Does not allow multiple inheritance
  - v. Allows code to be re-used
- f. The Object class
  - i. Every class has Object as a superclass
  - ii. Most classes should override the “toString” and “equals” methods
  - iii. The “getClass” method is useful for determining if two object instances are the same subclass
- g. Code to an interface
  - i. Motivation based on encapsulation and information hiding
  - ii. Ideally, public methods (which specify your API) should be defined in terms of interfaces rather than concrete classes
  - iii. This hides the details of your implementation from the user, which gives you the flexibility to change your implementation in the future

iv. However, an interface should be viewed as a contract between you and your user that changes as little as possible over time

12. Big-O Notation
    a. Time complexity means the same thing as time performance
    b. Intuitive definition of Big-O
        i. Big-O ignores added constants
        ii. Big-O ignores constant multiples
        iii. Big-O ignores small values of n
    c. Know how to identify the time complexity of common loop structures
    d. Know how to distinguish good time complexity from bad
        i. Best: O(1)
        ii. Good: O(log n)
        iii. Acceptable: O(n^k) for some constant k (polynomial time)
        iv. Unacceptible: O(2^n) (exponential time)
        v. Even worse: O(n!) (factorial time)
13. Searching
    a. Linear Search
        i. Time complexity: O(n)
        ii. Simple code
    b. Binary Search
        i. Requires a sorted input array
        ii. Time complexity: O(log n)
        iii. More complex code
14. Sorting
    a. Selection Sort
        i. Time complexity: O(n^2)
        ii. Simple code
        iii. Sorts in-place
    b. Insertion Sort
        i. Time complexity: O(n^2)
        ii. More complex code
        iii. Sorts in-place
        iv. Very fast when the input array is partially sorted.
    c. Merge Sort
        i. Time complexity: O(n log n)
        ii. Most complex code (uses recursion)
        iii. Requires extra storage space
15. Recursion
16. GUI