Your name: _____

CSSE 221 – Introduction to Software Development Honors

Exam 1

Closed-book problems: You can use a single 8.5 inch by 11 inch "cheat sheet" (back and front, with whatever you want on it). Nothing else. When you are done with this section, turn it in and get the rest of the test.

Each of the closed-book problems are 1 point except as noted. There is a total of 40 points in this section.

1. What keyword indicates that a ***method does not return a value***? _____

2. What keyword indicates that ***a variable cannot change its value from its initial value***? _____

3. (2 points) Give four examples of ***primitive types***.

4. (2 points) What is the ***implicit parameter*** in the following? The ***explicit parameters***?

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

5. (2 points) In a class definition, what keyword refers to the ***implicit parameter***? _____

6. (2 points) Explain what space is allocated by the following:

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

7. Generally, what ***visibility*** should ***fields*** have?
8. What ***visibility*** should a ***method*** have if you want it visible to objects of ANY class?

9. (2 points) If class B ***implements interface X***, what does that imply?

10. (2 points) If class B ***extends class Y***, what does that imply?

11. (5 points) Consider the Eye class, part of which is shown here:

```
public class Eye extends JPanel implements MouseMotionListener {
   protected EyeBall eyeBall;
   protected Color eyeColor;
   protected int eyeRadius;

   public Eye() {
         this(JavaEyes.DEFAULT_EYE_COLOR, JavaEyes.DEFAULT_EYEBALL_COLOR);
   }

   public Eye(Color eyeColor, Color eyeBallColor) {
         this.eyeColor = eyeColor;
         ...
   }


   public void mouseMoved(MouseEvent event) {
         Point mousePoint = new Point(event.getX(), event.getY());
         this.look(mousePoint);
   }

   protected void look(Point mousePoint) {
         int eyeX = this.getX() + this.eyeRadius; // Center of eye,
         int eyeY = this.getY() + this.eyeRadius; // relative to the frame
               ...
   }

   @Override
   public void paintComponent(Graphics graphics) {
         super.paintComponent(graphics);
         ...
   }
}
```

a. How many fields are there, and what are their names:

b. How many constructors are there?

c. How many methods are there and what are their names?

d. What is the name of one of the parameters (any one you pick is fine)?

e. What is the name of one of the local variables that is not a parameter (any one you pick is fine)?

12. (2 points) On the previous page, the *eyeColor* variable is NOT *static*. Suppose that we made it *static* – what would be the effect of that?

13. What does it mean for a method to ***override*** another method?

14. What does it mean for a method to be ***overloaded***?

15. What class is at the top of the ***inheritance hierarchy***? That is, what class does *every* class extend?

16. Does Java allow ***multiple inheritance***?

17. (2 points) Every object has a ***declared type***, and every non-null object has an ***actual type***. Give an example that shows the difference.

18. (2 points) What is the ***notation for doing a cast*** of the expression *x * 6.4* to type *int*?

19. (2 points) Give two reasons why ***UML class diagrams*** are useful.

20. What is the UML notation for ***is-a (extends)***?
21. What is the UML notation for ***is-a (implements)***?
22. What is the UML notation for ***has-a***?

23. What is ***cohesion*** in class design?

24. Do you want ***high cohesion*** or ***low cohesion*** in the classes that you design? (Circle your choice.)
25. What is ***coupling*** in class design?

26. Do you want ***high coupling*** or ***low coupling*** in the classes that you design? (Circle your choice.)

Your name: _____

Open-book problems: You may use any non-human source, e.g. your computer, the Internet, your text, the course web site, and your notes.

1. (10 points) Write statements that implement the following method per its description, using an explicit loop (no fair using the *Arrays.max* method):

```
// Returns the largest number in the given array.
// Assume that the array contains at least one number.
private Double largest(Double[] numbers) {




}
```

2. (10 points) Write statements that constructs an ArrayList of String's and initializes the ArrayList to contain 100 String's, each of which is "Hello".

On-the-computer problem

1. (40 points) Go to the SVN Repository Perspective and Checkout your ***Exam1*** project from your individual repository.

   The project has the *StringTransformable* interface that you used in WordGames.

   Use the following steps to implement the ***Quitter*** class described below, much as you did for the WordGames project:

   - Step 1: Create the class and add **documented stubs** for the constructors and methods of the class.
   - Step 2: Write **JUnit 4 tests** for the *transform* method of the class.
     - Thorough unit-testing might require a dozen or so tests per class, but in order to keep this project from being too much work, you should have **TWO well-chosen tests for this class**.
     - **Your Quitter code must have good style and documentation**, but your JUnit test class need NOT have any documentation and will NOT be graded on style. That is, I'll grade the quality of your two tests, but nothing else in your JUnit test class.
   - Step 3: Implement the class, as follows:
     a. Implement the code for the class.
     b. Run your JUnit tests for the class.
     c. Correct your code and/or unit tests as needed, and modify your documented stubs as needed.
   - Step 4: When your class is correct, commit your project.
     - Make sure that your commit has the *src* folder checked.
     - As always, include an appropriate comment, e.g. "Quitter finished" when you finish the *Quitter* class.

   A Quitter has a positive whole number (I'll call it *n*). For the first *n* String's that a Quitter is given, it returns its given String with " **- I am on it!**" appended. (That is a single space, a hyphen, another space, the words *I am on it* and an exclamation point.) After that, the Quitter simply returns its given String unchanged.

   If the Quitter is not told its number, it should use 1 as its number (so that it quits appending " - I am on it!" after having done the appending only once).

   For example, one Quitter might have 3 as its number. Given the following String's (in the order listed), it acts as indicated:
   - Given "Hello", returns "Hello - I am on it!"
   - Given "Hi there", returns "Hi there - I am on it!"
   - Given "Oops!", returns "Oops! - I am on it!"
   - Given "OK, stop fooling around", returns "OK, stop fooling around"
   - Given "Hello", returns "Hello"

   Just ask if you are not sure what a Quitter does. Grading rubric:

   - Correctness: 20 points (10 points for right constructor(s), 10 points for *transform* is correct)
   - Good JUnit tests: 10 points
   - Uses good style (Quitter class only, style is not graded in your JUnit class): 5 points
   - Has appropriate documentation (Quitter class only, no need for any documentation in your JUnit class): 5 points