What is your name?: ________________________ (0 points)

There are two sections:

I. True/False 40 points; (2 points each, 20 questions)
II. Problems 60 points; (6 points each, 10 questions)

100 points total

This test is worth 20% of your final grade. This test is open book and open notes. Use the back of the sheet if you need more room to write, although you shouldn’t need to. State your assumptions wherever it makes a difference. You have 60 minutes.

I. True False: (2 pts. each)

1. In the classic article "GoTo Statement Considered Harmful," Dijkstra explains that our powers to visualize processes evolving in space are relatively poorly developed.

2. Consider Pascal, with nested subprogram declarations. The static chain length and local offsets must be computed dynamically, since the calling sequence may vary.

3. Every user-defined class automatically has toString() and equals() methods that can be used.

4. The Object class clone() method will work correctly if the object being cloned does not itself have any object references.

5. When you override a base class method, the access specifier in the derived class must be at least as restrictive as in the base class. No, it must be no more restrictive.

6. An abstract method may or may not have a body.

7. When moving an object from one type of container class to another, it is important to cast the object into the correct type. That is not necessary, since they are stored as objects.

8. Swing components can handle running on multiple platforms, and as such are called heavy-weight components. They’re called lightweight components. AWT components are heavy-weight.

9. Items such as buttons can be added either directly to a Frame or to the contentPane within the Frame. They can be added to the Frame directly.

10. To use a sports analogy, event handling in Java is more like a baseball game than a soccer or basketball game. Everything is setup, waiting for an event (the pitch).

11. The same event handler can be registered more than once for the same object, such as a button.

12. An inner class has access to its corresponding outer class public variables, but not the outer class private variables.

13. A public inner class can be instantiated from outside the outer class. Using the Outerclass#Innerclass notation.
14. If the same event handler will be used in many places, it is best not to make it an anonymous inner class. By naming it, the event handler can be defined once and reused.

15. An anonymous inner class used to implement an event handler can refer to an object that has been declared but not defined until later in the code. (See question 10 in the Problems section)

16. When implementing an Interface, you must implement all the methods in that Interface.

17. A class can be its own event handler. It must implement the event-handler methods

18. The Swing `paintComponent()` method should not be called directly. Instead call `repaint()`

19. The `update()` method should be overridden when implementing double-buffering, because otherwise the superclass version will repaint the background, causing flickering.

20. There is no simple way to extend an instantiable class (e.g. `Point`) and add an aspect (e.g. color) while preserving the `equals` contract. See the Joshua Bloch material for explanation.

II. Problems: (6 pts. each) Briefly answer each of the following in the space provided.

1. Consider the program outline shown at left below. Using the blocks shown at right below, draw the stack using a single block for each stack frame corresponding to the calling sequence:

```
main -> bigsub -> sub1 -> sub3 -> sub1 -> bigsub -> sub2
```

For each stack frame show only the static links. Assume the call to `main` uses the stack.
2. What problems would arise in Java if suddenly the mechanism implementing the dispatch vector stopped working?

   Polymorphism would no longer work. Methods would need to be bound to a class statically.

3. What are the 3 main uses of Interfaces?
   a) Simulate multiple Inheritance
   b) Sharing Constants
   c) Type Tag denoting that a class is serializable
   d) Mark how clone should be used to provide a deep copy

4. How is the association made between a GUI button and the code that should be executed when the button is pressed?

   The code to be executed must be registered as an event-handler for the button. This creates a reference to the handler on the object's listenerList.

5. Is there any way in Java to implement multiple-inheritance, aside from using Interfaces? Explain why or why not.

   Yes, using an inner class. The outer class can extend class A, and then an inner class can extend class B, implementing multiple inheritance.
6. Assume you have a GUI with multiple buttons. Pressing each button will change the GUI background color to the color that corresponds to that button. I.e. pressing one of the buttons might always make the background blue, while pressing another of the buttons might make it red. What are two ways to implement this that we discussed in class?

a) Check the identity of the sender within the event-handler code, using a decision statement (if, switch) to choose between alternative colors to set the background.

b) Have the event handler be a separate class, with an instance variable to represent the color. When creating the event handler instance for each button, set the appropriate color. The event handler then uses this value to set the background.

7. List the four methods you should always implement in one of your own classes if you are planning on using the Collections classes.

a) toString()

b) equals()

c) hashCode()

d) compareTo()

8. For each of the four methods you listed in the problem above, tell what you would not be able to do if it was missing from your class.

a) toString() - without it we cannot print meaningful information using System.out.println

b) equals() - without it Set collections can’t eliminate duplicates

c) hashCode() - without it Collections classes using a hashable cannot eliminate duplicates, since hashCode() is called first to determine the location at which to subsequently call equals().

d) compareTo() - without it (and the Comparable Interface declaration) we can’t sort objects, giving them an order.
9. What is the difference between putting code in the exception-handling **finally** block and simply listing the code after the **try-catch** block?

The code in the **finally** block will always be executed.

Code after the try-catch block may not be executed after an exception occurs.

10. Consider the following code:

```java
public class myFrame extends JFrame {
    private JList firstJList;
    private JList secondJList;
    private JButton theJButton;
    private final String listNames[] = {"One", "Two", "Three", "Four");
    ...

    public myFrame() // constructor
    {
        ...
        firstJList = new JList(listNames);
        theJButton = new JButton("Press Here"); // create button
        theJButton.addActionListener(
            new ActionListener()
            {
                // handle button event
                public void actionPerformed(ActionEvent event)
                {
                    secondJList.setListData(firstJList.getSelectedValues());
                }
            }
        );
        getContentPane().add(copyJButton);

        secondJList = new JList(); // definition
    } // end myFrame constructor

} // end class myFrame
```

In particular consider the highlighted code using **secondJList**. Will this code work correctly? Explain why or why not.

This does work because the code at \(\Box\) will execute after the code at \(\Box\). The code at \(\Box\) is event-handler code, and will only execute after everything has been set up and the corresponding event occurs.