I. Fill in the Blank (30 pts)

1. Most of the components in the java.awt package are **heavyweight** components which means they rely a good deal on the underlying platform.

2. **Polymorphism** refers to a dynamic binding that occurs at runtime between a method call and its correct implementation.

3. A **package** in Java is a collection of related classes.

4. Most of the components in the javax.swing package are **lightweight** components which means that they do not rely a good deal on the underlying platform.

5. An inner class that cannot define a constructor is called an **anonymous** inner class.

6. We use the keyword **synchronized** to define code that will allow a thread to obtain a lock on an object.

7. In order to detect and act on a button click, we implement the **ActionListener** interface.

8. **Autoboxing** is a feature of Java that allows a primitive type to be automatically wrapped into its corresponding wrapper class.

9. The **signature** of a method is a combination of its name and parameter list.

10. If a method is inherited and its implementation changes, but its signature stays the same, we call this **methodoverriding**.

11. **InformationHiding** is the ability to restrict the access to or the visibility of data.

12. If a class is declared final, then it cannot include the keyword **extends** in its definition.

13. Java supports **singleinheritance** which means that we can only list one class definition in an extends clause.
14. We can use the method `getStateChange()` to determine whether an ItemEvent was caused by selecting or deselecting an item.

15. In order to register a listener on a button when we want to detect and act on button clicks, we use the method `addActionListener`.

16. The wrapper class for the int data type is `Integer`.

17. We can use the method `getSource()` to get a reference to the component that caused an event to occur.

18. The method `getCodeBase()` determines the URL where an applet’s .class file is found.

19. **Casting** changes the reference type of an identifier to a compatible reference type.

20. Every field that we declare in an interface is implicitly public, static, and final.

21. **Covariant returns** is a new feature of Java that allows us to replace the return type on an overridden method with a subclass of the return type of the original method.

22. **Completeness** is a property of good class design that says a class definition should completely solve the problem it was intended to solve.

23. A **abstract** class contains at least one method without an implementation.

24. The method `run()` is specified in the Runnable interface.

25. If class A extends B, then we call A a **subclass** of B.

26. The **leading** of printed text is the space between lines.

27. A thread calls the method `notifyAll()` to inform all waiting threads to get ready to run.

28. The **FlowLayout** layout manager arranges components on the GUI from left to right.

29. **Number** is the superclass of all the wrapper classes for the numeric primitive types.

30. If an entity is declared **protected**, then it is visible from within the same class, from within the same package, and from within subclasses.
II. Matching (10 pts)

31. super V  A. The class that is automatically extended if a class definition doesn’t have an extends clause.

32. repaint() L  B. A property of good class design that says we shouldn’t add extra features to a class definition that aren’t related to the problem it was intended to solve.

33. public void actionPerformed(ActionEvent e) Q  C. A nested class that is declared static.

34. public String toString(String string) G  D. Is generated by the compiler for a class definition when it contains no other constructors.

35. private H  E. The method specified in the ItemListener interface.

36. inner class I  F. A Java keyword that allows a class definition to refer to itself.

37. getDocumentBase() P  G. An example of method overloading.

38. default constructor D  H. An access control keyword that means an entity is only visible from within the same class.

39. cohesion B  I. An access control keyword that means an entity is visible from anywhere.

40. Object A  J. A nested class that is not declared static.

K. The only class in Java that has no subclasses.

L. A method that should be called on a component to request it be redrawn.

M. A constructor that a compiler calls when it is tired of waiting for a programmer to call one.

N. The method specified in the Runnable interface.

O. A property of good class design that says the internal activity of one class should not be affected by another class.

P. A method that returns a reference to the URL where the .html file that loaded an applet is located.

Q. The method specified in the ActionListener interface.

R. A method that is guaranteed to always result in a call to paint.

S. An access control keyword that means an entity is only visible from within the same class and within the same package.

T. A method that returns a reference to the URL where the .class file of an applet is located.

U. An example of method overriding.

V. A Java keyword that allows a class definition to refer to its immediate superclass.
III. **Short Answer** (20 pts)

41. (2 pts) What are two things not inherited by a subclass?

   **constructors and private members**

42. (2 pts) If we have a call to a superclass constructor from a subclass constructor, where must that call be?

   **the first executable line in the subclass constructor**

43. (2 pts) Given a logical group of JRadioButtons with the first button selected, how many ItemEvents are generated by selecting another button?

   2

44. (2 pts) If we have a KeyListener attached to a GUI, and we also are listening for button clicks, what method must we call on the GUI after a button click so that KeyEvents will be heard?

   **requestFocus**

45. (2 pts) Suppose we have one currently executing thread running and we have another class definition that defines a Thread. If we call the run() method of that class definition, how many executing Threads will there be?

   1

46. (2 pts) Suppose we have a class definition called MovingButtons, and within that class definition, we have an inner class defined. If MovingButtons defines an int named num, and the inner class defines an int named num, from within the inner class how could we refer to the variable named num associated with the outer class?

   **MovingButtons.this.num**

47. (2 pts) Suppose we have the following param tag in the HTML code that loads an applet.

   ```html
   <param name="number" value="5">
   ```

   Within the applet, how could we retrieve the value associated with the parameter?

   ```java
   getParameter("name");
   ```

48. (2 pts) Will the following assignment compile correctly? Why or why not?

   ```java
   Double d = 1;
   ```

   **No, it will not. int is autoboxed into an Integer, but Integer is not a subclass of Double**

49. (2 pts) Does the statement import java.awt.* import the interface that we use to listen for button clicks?

   **No**

50. (2 pts) Is the following possible? Why or why not?

    Create an anonymous inner class and define an int in its constructor.

    **It is not possible. An anonymous inner class can’t have a constructor.**
IV. Discussion (10 pts)

51. (3 pts) What is the start() method used for in an applet?

It is used to inform the applet to start running. It is executed each time the page containing the applet is loaded.

52. (3 pts) What is a reason for making a class abstract?

To specify behaviors that must be implemented in subclasses.

53. (4 pts) Why can’t an abstract class be declared final?

The only way we can get an instance of an abstract class is to create a concrete subclass and define the methods that don’t have implementations. If we declare the class final, we can’t create a subclass.
V. Problem Solving and Coding (30 pts)

54. (5 pts) What will be the result of attempting to compile and execute the following program?

```java
public class Question54 implements Interface1 {
    public static void main(String[] args) {
        System.out.println("Hello");
    }
}

interface Interface1 {
    public void print();
}
```

We will get a compile-time error because we don't fulfill our obligation in the interface contract. If a class definition implements an interface, it must give implementations to the methods specified in the interface.

55. (5 pts) What is printed by the following code?

```java
public class Question55 {
    public void print() { System.out.println("Print"); }
    public static void print1() { System.out.println("Print 1"); }
}

class Test extends Question55 {
    public void print() { System.out.println("Print in Test"); }
    public static void print1() { System.out.println("Print 1 in Test"); }
    public static void main(String[] args) {
        Question55 question55 = new Question55();
        Question55 question551 = new Test();
        question55.print();
        question55.print1();
        question551.print();
        question551.print1();
    }
}
```

Print
Print 1
Print in Test
Print 1
56. (5 pts) Will the following code compile and execute correctly? Why or why not?

```java
public class Question56 {
    protected int num;

    public Question56(int num) {
        this.num = num;
    }
}

class Test extends Question56 {

    public static void main(String[] args) {
        Test test = new Test();
        System.out.println(test.num);
    }
}
```

No, it will not. A default constructor will be created for Test which will attempt to call the no-arg constructor of Question56. Since there isn’t one defined, but there is a constructor defined, a default constructor will not be created for Question56. So there will be a compile-time error.

Choose two of the following programming problems. Each will be worth 7.5 points. If you solve more than two, correct solutions will count as extra credit.

57. Write the class definition of an applet that will place a grid of 100 buttons with 5 rows and 20 columns on the applet.

```java
import java.awt.*;
import javax.swing.*;

public class Question57 extends JApplet {

    public void init() {
        setLayout(new GridLayout(5,20));
        for (int counter=0;counter<100;counter++)
            add(new JButton());
    }
}
```
58. Write a class called Bubba that specifies two different methods, one called print that accepts an int and one called print that accepts a double. The return type of both methods should be void. Neither method should have an implementation. Create two subclasses, one called Goober that will override the print method that accepts an int and prints 1, and another called Gomer that will override the print method that accepts a double and prints 1.0/0. Make sure that you include the correct keywords so that the class definitions will compile.

```java
public abstract class Bubba {
    public abstract void print(int i);
    public abstract void print(double d);
}

class Goober extends Bubba {
    public void print(int i) {
        System.out.println(1);
    }
    public void print(double d) {
    }
}

class Gomer extends Bubba {
    public void print(int i) {
    }
    public void print(double d) {
        System.out.println(1.0/0);
    }
}
```

59. Write the class definition of an applet that will read the value of a parameter named message from the HTML code that loads the applet and will center that String in the applet.

```java
import java.awt.*;
import javax.swing.*;

public class Question59 extends JApplet {
    private String message;
    public void init() {
        message = getParameter("message");
    }
    public void paint(Graphics g) {
        super.paint(g);
        FontMetrics metrics = g.getFontMetrics();
        int width = metrics.stringWidth(message);
        int height = metrics.getAscent()/2;
        g.drawString(message, getWidth()/2-width/2, getHeight()/2-height/2);
    }
}
```
import java.awt.*;
import javax.swing.*;

public class Question60 extends JFrame {
  public Question60(String title) {
    super(title);
  }

  public void paint(Graphics g) {
    super.paint(g);
    // Head
    g.drawOval(getWidth()/2-50,getHeight()/2-50,100,100);
    // Body
    g.drawLine(getWidth()/2,getHeight()/2+50,getWidth()/2,getHeight()/2+200);
    // Arms
    g.drawLine(getWidth()/2,getHeight()/2+100,getWidth()/2-50,getHeight()/2+150);
    g.drawLine(getWidth()/2,getHeight()/2+100,getWidth()/2+50,getHeight()/2+150);
    // Legs
    g.drawLine(getWidth()/2,getHeight()/2+200,getWidth()/2-50,getHeight()/2+240);
    g.drawLine(getWidth()/2,getHeight()/2+200,getWidth()/2+50,getHeight()/2+240);
  }

  public static void main(String[] args) {
    Question60 question60 = new Question60("Question 60");
    question60.setSize(500,500);
    question60.setLocationRelativeTo(null);
    question60.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    question60.setVisible(true);
  }
}