I. Fill in the Blank (30 pts)

1. **Encapsulation** is a binding together of data and the code which operates on the data.

2. We use the method `repaint()` to request that the paint or paintComponent method be called on a Component as soon as possible.

3. If a class is **abstract**, its definition is incomplete, and we cannot create an instance of it.

4. If a class definition is declared **final**, we cannot create a subclass of it.

5. The components in the java.awt package are considered **heavyweight** components since they rely heavily on the underlying platform.

6. We use the method `getStateChange()` to determine if an ItemEvent was caused by a selection or deselection of a JCheckBox, JRadioButton, or JComboBox.

7. The principle of **loose coupling** says that we should not write class definitions in such a way that changing the internal implementation will affect how outside classes use them.

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

9. We use the `split` method in the String class to produce an array of tokens separated by a delimiter.

10. **Autoboxing** allows a primitive to be automatically converted to its corresponding wrapper type.

11. Casting changes the **compile—timetype** of a reference.

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

13. If a local variable in a method has the same name as an instance variable in the class definition containing the method, then the local variable is said to **shadow** the instance variable.
14. **Method Overriding** involves changing the implementation of an instance method we inherit from a superclass.

15. The principle of **cohesion** says that we shouldn’t include extra things in our class definition that don’t pertain to the problem we are trying to solve.

16. A String is an example of an **immutable** object since, once its created, its contents can’t be changed.

17. The method **itemStateChanged** is found in the ItemListener interface.

18. We use the method **drawOval** to draw a circle or ellipse using a Graphics object.

19. If A extends B, we call A a **subclass** of B.

20. In order to include a “ in a String literal, we must **escape** the character.

21. The variables defined in an interface are implicitly public, **static**, and final.

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

23. We use the method **getSource()** to get a reference to the component that caused an event to occur.

24. Class methods are **hidden** instead of overridden in a subclass.

25. If we do not include a constructor in our class definition, then the compiler supplies a **default constructor**.

26. We use the method **setDefaultCloseOperation** on a JFrame so that when we click on the X in the upper-right corner of the JFrame, our program will exit and close.

27. If an entity is declared **protected**, it can be accessed from within the same class, the same package, and any subclass.

28. If C extends B, we call B a **superclass** of C.

29. A **wrapper class** is a class which can be used to encapsulate a corresponding primitive type so that it can be used in situations where an object is required.

30. If a variable or method defined in a class definition is not declared static, it is known as an **instance** variable or method.
II. Matching (10 pts)

31. protected \textbf{N} \quad \text{A. constructors, initializers, and instance methods.}

32. addActionListener \textbf{H} \quad \text{B. A Java keyword that allows a variable to be accessed from the same class and the same package only.}

33. “is a” \textbf{M} \quad \text{C. s.charAt(s.length());}

34. Things not inherited by subclasses \textbf{X} \quad \text{D. private variables and methods, constructors, and static variables and methods.}

35. The last valid position in a String, s \textbf{G} \quad \text{E. s.charAt(s.length()-1);} \quad \text{F. Is used to retrieve the value of private members of a class definition.}

36. Mutator method \textbf{I} \quad \text{G. s.charAt(s.length()-1);}

37. FlowLayout \textbf{O} \quad \text{H. Is used to register a listener on a JButton so that a button click is detected.}

38. Compile-time type of a reference \textbf{L} \quad \text{I. Is used to change the value of private members of a class definition.}

39. == \textbf{W} \quad \text{J. When used in a regular expression will match one or more occurrences of a character.}

40. . \textbf{Q} \quad \text{K. When comparing two object references will return true only if both references refer to objects with the same contents.}

\text{L. Determines which methods are accessible to a reference.}

\text{M. A relationship that exists between A and B if A extends B.}

\text{N. A Java keyword that allows a variable to accessed from the same class, the same package, or any subclass.}

\text{O. A layout manager that will lay out components from left to right.}

\text{P. A layout manager that will lay out components in a rectangular grid.}

\text{Q. When used in a regular expression will match any single character except a newline.}

\text{R. s.charAt(s.size()-1);}

\text{S. A relationship that exists between A and B if B extends A.}

\text{T. A Java keyword that allows a variable to be accessed nowhere.}

\text{U. Is used to register a listener so that a change to a JRadioButton is detected in such a way that we can determine whether it was selected or deselected.}

\text{V. When used in a regular expression will only match digits.}

\text{W. When comparing two object references will return true only if both references refer to the same object.}

\text{X. private variables and methods, constructors, and initializers.}

\text{Y. Determines the instance method that will be called at run-time.}
III. Short Answer (20 pts)

41. (2 pts) Suppose we have a class definition containing a class variable named num. Suppose we create an array, the type of whose elements is the class definition, of size 8,454 and initialize each of the instances. How many copies of the variable num are stored in memory?

1

42. (3 pts) Suppose a String object referred to as s contains the following characters: floccinaucinihilipilification Show one line of code that will print the position of the second occurrence of the character ‘a’ without using any actual numbers other than the number 1.

    System.out.println(s.indexOf('a',s.indexOf('a')+1));

43. (2 pts) Will the following produce a compile-time error? Explain.

    Double d = 5;

There will be a compile-time error. 5 will be autoboxed into an Integer, but there is no relationship between Integer and Double.

44. (2 pts) Would it make sense to declare a class both final and abstract? Explain.

No it doesn’t make sense. We can only create an instance of an abstract class by creating a concrete subclass, but if the class is declared final, we can’t create a subclass of it.
45. (3 pts) Suppose we override the paintComponent method in a class definition. How would we obtain a reference to a FontMetrics object and use that object to horizontally center the String “Hello” on the container at the vertical position 55?

```
FontMetrics metrics = g.getFontMetrics();
g.drawString("Hello", getWidth()/2-metrics.stringWidth("Hello")/2,55);
```

46. (2 pts) Suppose we have a reference called button to a JButton object. Suppose the label on the JButton is a number. How can we in one line change the number on the button to its quotient when the number is divided by 50?

```
button.setText(String.valueOf(Integer.parseInt(button.getText()) / 50));
```

47. (3 pts) Suppose we have the following class definitions.

```
class A extends B
class B extends C
```

Which two of the following declarations will cause a compile-time error? Assume that there are no constructors defined in classes A, B, or C. Explain the compile-time errors.

- A a = new A(); **No compile-time error**
- B b = new A(); **No compile-time error because A “is a” B**
- C c = new A(); **No compile-time error because A “is a” C**
- A a1 = new B(); **Compile-time error because B is not a A**
- B b1 = new C(); **Compile-time error because C is not a B**
- C c1 = new B(); **No compile-time error because B “is a” C**

48. (3 pts) Suppose A extends B and B extends C. In the following code, will there be a compile-time error or a run-time error? Explain.

```
B b = new B();
Object o = (Object)b;
A a = (A)o;
C c = (C)o;
```

**There will not be a compile-time error because each cast is to a superclass or subclass type. However, there will be a run-time error because the object that is created is of type B. So when we try to cast the compile-time type of the reference to A, there will be a ClassCastException.**
IV. Discussion (10 pts)

49. (2 pts) What are two different ways of creating an “is a” relationship?

**By extending a class and by implementing an interface**

50. (4 pts) Suppose we want to use the instanceof operator. Assuming that the use of instanceof is of the form 
\[ a \text{ instanceof } B \]
where \( a \) is an object reference, and \( B \) is a class definition, under what circumstances will this use of instanceof produce a compile-time error?

**If a cast \((B)a\) is not allowed at compile-time, then we will get a compile-time error using instanceof**

51. (4 pts) Suppose that a class definition includes the phrase “implements ActionListener” in its header. Explain the contract that exists between the class definition and the interface. In particular, identify what the class definition is required to do and the benefit the class receives for doing it.

**The class definition agrees to implement all methods listed in the interface. In exchange, the class definition has the “is a” relationship with the interface and an instance of the class can be used anywhere an object reference whose type is the interface is expected.**
V. Problem Solving and Coding (30 pts)

52. (5 pts) What is printed when Question52 is executed? Explain the output.

```java
public class Question52 {
    public static void main(String[] args) {
        A a = new A();
        B b = new A();
        a.print();
        System.out.println(a.isPrimitiveRoot());
        System.out.println(a.num);
        b.print();
        System.out.println(b.isPrimitiveRoot());
        System.out.println(b.num);
    }
}
```

A
true
1
B
true
2

53. (5 pts) Suppose we have the following declarations. Are there any syntax errors when we compile Test.java? If not, what is printed? Explain the output.

```java
public interface Feasible {
    public void feaseIt(Object o);
    public boolean canBeFeased();
}

class B {
    protected int num = 2;
    public static void print() { System.out.println("B"); }
    public boolean isPrimitiveRoot() { return(false); }
}

class A extends B {
    protected int num = 1;
    public static void print() { System.out.println("A"); }
    public boolean isPrimitiveRoot() { return(true); }
}

class Test implements Feasible {
    public void feaseIt() {
        System.out.println("We feased it");
    }
    public boolean canBeFeased() {
        return(true);
    }
    public static void main(String[] args) {
        Test test = new Test();
        test.feaseIt();
    }
}
```

There will be a compile-time error when Test.java is compiled. Test.java inherits the method public void feaseIt(Object o) from the interface, but doesn’t give it an implementation.
54. (5 pts) Consider the following code. Will there be a compile-time error? If so, what is it? If there is a compile-time error, what line of code could you remove to remove it? What is printed at runtime in that case? If there was no compile-time error to begin with, what is printed at runtime?

```java
public class Question54 {
    private int num;
    public Question54(int num) {
        this.num = num;
    }
    public boolean equals(Object question54) {
        return (num == question54.num);
    }
    public static void main(String[] args) {
        Question54 question541 = new Question54(1);
        Question54[] question54 = new Question54[5];
        for (int counter = 0; counter < question54.length; counter++)
            System.out.println(question54[counter].equals(question541));
    }
}
```

Yes, there will be a compile-time error. In the equals method, the type of the parameter is Object so even though the type of the object we send to the method is Question54, the type of the parameter is Object so we cannot access any variable not accessible in Object. If we remove the line `return (num == question54.num);` , then the program will compile successfully. In that case, there will be an error at runtime since each element of the question54 array is initialized to null so we cannot call any methods with those references.
55. (7.5 pts) Fill in the following method so that a square of width 70 is drawn centered on the Container containing this paint method with a circle filled in blue inside the square and lines drawn in black from opposite corners of the entire Container. Also draw a quadrilateral (4-sided polygon) in yellow without using the drawLine method whose vertices are the midpoints of the outer edges of the Container.

```java
import java.awt.*;
import javax.swing.*;
public class Question55 extends JFrame {
    public Question55(String title) {
        super(title);
    }
    public void paint(Graphics g) {
        super.paint(g);
        g.drawRect(getWidth()/2-35,getWidth()/2-35,70,70);
        g.setColor(Color.blue);
        g.fillOval(getWidth()/2-35,getWidth()/2-35,70,70);
        g.setColor(Color.black);
        g.drawLine(0,0,getWidth(),getHeight());
        g.drawLine(getWidth(),0,0,getHeight());
        g.setColor(Color.yellow);
        int[] xpoints = getWidth()/2,0,getWidth()/2,getWidth();
        int[] ypoints = 0,getHeight()/2,getHeight(),getHeight()/2;
        g.drawPolygon(xpoints,ypoints,4);
    }
    public static void main(String[] args) {
        Question55 question55 = new Question55("Question 55");
        question55.setSize(500,500);
        question55.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        question55.setLocationRelativeTo(null);
        question55.setVisible(true);
    }
}
```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class DrawStringAndButton extends JFrame implements ActionListener {
    private JButton button;
    private boolean draw;

    public DrawStringAndButton(String title) {
        super(title);
        setLayout(new FlowLayout());
        button = new JButton("Press Me");
        button.addActionListener(this);
        add(button);
    }

    public void paint(Graphics g) {
        super.paint(g);
        if (draw) {
            FontMetrics metrics = g.getFontMetrics();
            int width = metrics.stringWidth("Hello");
            g.drawString("Hello",getWidth()/2-width/2,3*getHeight()/4);
        }
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == button) {
            draw = true;
            repaint();
        }
    }

    public static void main(String[] args) {
        DrawStringAndButton drawStringAndButton = new DrawStringAndButton("Draw String and Button");
        drawStringAndButton.setSize(500,500);
        drawStringAndButton.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        drawStringAndButton.setLocationRelativeTo(null);
        drawStringAndButton.setVisible(true);
    }
}