1. **Polymorphism** is a dynamic binding that occurs at runtime between a method call and its correct implementation.

2. **Method Overriding** involves redefining the implementation of a method which is inherited.

3. If we don’t use an extends clause in a class definition, then the class **Object** is automatically extended.

4. A **String literal** is a group of characters surrounded by double quotes.

5. A **deprecated feature** is a feature of an evolving language like Java that may or may not be supported in future releases.

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

7. The parameters in the call to a method are called **actual** parameters.

8. We can allow an applet to run in its own thread by implementing the **Runnable** interface which contains the method **public void run()**.

9. The method **getParameter** can be used within an applet to retrieve values sent in a param tag in the HTML Code that loaded the applet.

10. True/False **False** We can create a subclass of any class definition in Java.

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

12. If a class is **abstract**, then we can’t create instances of it with a constructor.

13. We know that we are creating an instance variable in a class definition if we don’t see the keyword **static** in its declaration.

14. **Inheritance** involves creating a new class definition by absorbing the characteristics of an existing class definition and adding new functionality.
15. If a variable is declared **private**, then it can only be accessed from within the class definition in which it is declared.

16. If a method does not return a value, its return type should be **void**.

17. The method `countTokens()` determines the number of tokens which have not yet been read from a `StringTokenizer`.

18. The value of “Hello”.indexOf(‘a’) is **-1**.

19. True/False **False** The operator `==` when used between two object references determines whether or not the content of the objects referred to by those references are the same.

20. Primitive data types are sent to methods by **value** which means that a copy of the primitive data type is made and there is no change to the original.

21. A **paradigm** is a style of programming.

22. An instance of a class refers to its immediate superclass with the keyword **super**.

23. A String object is **immutable** which means that once it is created, it can’t be changed.

24. The parameters in the declaration of a method are called its **formal** parameters.

25. If public class A extends B, then B is called a **superclass** of A.

26. If a local variable has the same name as an instance variable, then that local variable is said to **shadow** the instance variable.

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

28. The operator **instanceof** determines whether or not an object reference is an instance of a class definition.

29. In order to get a String representation of an object, we can call the method **toString** which is inherited by all classes.
II. Matching (10 pts) Please choose the most appropriate choice for each term from the candidates on the right.

30. public void change(String s) B
   A. A concept supported by Java which allows us to directly extend multiple single classes.

31. protected L
   B. Sending a String reference to this method could never change the String object referred to by the reference.

32. repaint() W
   C. A concept supported by Java which only allows us to directly extend one other class.

33. length() M
   D. An unnecessary method call which could be eliminated by simply calling every instance variable of an applet.

34. A “is a” B U
   E. An example of method overriding.

35. this N
   F. Allows variables to be accessed from within the same class or the same package.

36. “Hello”.substring(0,6) J
   G. An example of a compile-time error and not a run-time error.

37. final V
   H. An example of method overloading.

38. Single Inheritance C
   I. Allows variables to only be accessed from within the class definition in which they are defined.

39. public String toString(String s) in a class H
   J. An example of a run-time error and not a compile-time error.

K. A keyword which will create a static instance variable that can’t be changed unless we create a subvariable of it.

L. Allows variables to be accessed from within the same class, the same package, and in any subclass.

M. A method that determines how many characters are in a String.

N. A reserved word that allows an instance of a class to refer to itself.

O. A method that will automatically change every String reference sent to it.

P. Allows variables to be accessed from any class.

Q. A relationship that allows us to treat objects created from class B as if they are objects created from class A.

R. A pronoun that is used to refer to the instance variables of a subclass.

S. A field associated with an array that determines how many elements are in it.

T. An example of nothing.

U. A relationship that allows us to treat objects created from class A as if they are objects created from class B.

V. A keyword which allows us to prevent a user from making a subclass of a class definition or to prevent a subclass from overriding a method it inherits.

W. A request made to the context of a component like an applet for it’s paint method to be called as soon as possible.
III. Short Answer (30 pts)

40. (4 pts) Suppose we have the following lines in a main method. What is printed when the program is interpreted and why?

```java
String[] words = {"Hello", "Hello"};
System.out.println(words[0] == words[1]);
```

true will be printed. Because we are creating an array where both entries are the same String literal, only one String object is created in memory, so both elements of the array words point to the same object in memory.

41. (4 pts) What are two ways of converting an object reference to a String?

- By calling the toString method on the object reference
- By concatenating the object reference with an empty String
- By using the valueOf method in the String class

42. (4 pts) Suppose we have the following class header.

```java
public class MyClass extends JApplet implements Runnable, ActionListener
```

Name at least 5 “is a” relationships that exist when an object of class MyClass is created.

- MyClass “is a” JApplet
- MyClass “is a” Runnable
- MyClass “is a” ActionListener
- MyClass “is a” Object
- JApplet “is a” Object

43. (4 pts) Suppose we have a JButton called button within an applet. Give two lines of code which will cause the button’s width to be doubled, its height to be halved, and it’s location changed so that it’s lower-right corner is at the lower right-corner of the applet. Assume this code will be called from within a method of the applet.

```java
button.setSize(button.getSize().width*2, button.getSize().height/2);
button.setLocation(getWidth()-button.getSize().width, getHeight()-button.getSize().height);
```
44. (4 pts) Suppose we create the following array.

    int[][] numbers = new int[3][3];

Which of the following lines will not produce a syntax error when compiled? Explain.

   a) System.out.println(numbers.length);
   b) System.out.println(numbers[0].length);
   c) System.out.println(numbers[0][0].length);

   a) is okay because numbers is a reference to an array and length can be called on an array reference
   b) is okay because numbers[0] is a reference to an array and length can be called on an array reference
   c) is not correct because numbers[0][0] is an int

45. (4 pts) Would it make sense for a class definition to be declared both final and contain an abstract method? If so, how could you go about giving an implementation to the abstract method? Explain.

   It would not make sense to make a class final and abstract. The point of creating an abstract class is so that you can make subclasses of it and implement its abstract methods. But we can't create a subclass of a final class.

46. (4 pts) Suppose a devious instructor asked you to perform the following calculations. Initialize an int variable called result to be the result of “Hello”.indexOf(‘o’). Then multiply the current value of result by the number of primitive data types in Java. Replace result with this product. After you have done this, subtract the number of bytes used to store an int in Java from result, replace result with this difference, and then find the quotient when you divide result by the number of bytes used to store a double in Java. Replace result with this quotient. Add to result the number of methods in a class definition that can have the same signature. Then add this to the product of the number of bytes used to store a char in Java and the number of parameters required in an applet tag. Replace result with this sum. What is printed by the following line when it is interpreted?

   “Interesting Question”.substring(result);

   int result = 4
   result = result * 8 = 32
   result = result - 4 = 28
   result = result / 8 = 3
   result = result + 1 + 2*3 = 10
   “Interesting Question”.substring(10) = “g Question” although nothing is printed because the output isn’t captured

47. (1 pt) If s refers to a String object, what line of code would retrieve the last character in the String object?

   s.charAt(s.length()-1)

48. (1 pt) Would the following cause a compile-time error, run-time error, or neither? Explain.

   String[] tokens = “1;2;3;4;5”.split(“;”);
tokens[1000000] = tokens[31/19].length()+tokens[-200*37/21].length()-40+tokens.length*-1 + “”;
System.out.println(tokens[-13]);

   Since objects aren’t built until run-time, the indices aren’t checked. So there aren’t any syntax errors. But there are run-time errors
IV. Problem Solving and Coding (30 pts)

49. (5 pts) Will there be a syntax error when the following class definition is compiled? If so, what is the problem?

```java
public class Question49 {
    private int number;
    public int number() {
        return (number);
    }
    public boolean equals(Object o) {
        return (number == o.number());
    }
    public void main(String[] args) {
        Question49 question49 = new Question49(1);
        System.out.println(question49.equals(new Question49(1)));
    }
}
```

A syntax error will be reported because there are not enough parentheses in the return statement of the method. If there were a correct number of parentheses, then a syntax error will be reported since there is no constructor of Question49 that accepts an int. If there were, then there would be another syntax error because when a Question49 object is sent to the method equals, it is promoted to be an Object reference, and we can’t call the method number() because it isn’t defined in Object.

50. (5 pts) Is there a syntax error when the following code is compiled? If not, is there a run-time error? Explain.

```java
public class Question50 implements Runnable {
    private int number;
    public Question50(int number) {
        this.number = number;
    }
    public static void main(String[] args) {
        Question50 question50 = new Question50(1);
        System.out.println(question50);
    }
}
```

There is a syntax error because we don’t implement the method run that is specified in the Runnable interface. If we defined the class to be abstract, we can’t create an instance of it with a constructor.
51. (5 pts) Consider the following class definition. Is there a syntax error? Will there be a run-time error? If so, what is the run-time error? If not, what is printed when the class definition is interpreted?

```java
public class Question51 {
    public static void main(String[] args) {
        StringTokenizer tempST = new StringTokenizer(“1;2;3;4;5;6”, “;“);
        int sum = 0;
        for (int counter=0;counter<tempST.countTokens();counter++)
            sum += Integer.parseInt(tempST.nextToken());
        System.out.println(“1;2;3;4;5;6” .substring(sum));
    }
}
```

Recall that `countTokens()` decreases each time we call `nextToken()`.

<table>
<thead>
<tr>
<th>counter</th>
<th><code>nextToken()</code></th>
<th><code>countTokens()</code></th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>“1”</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>“2”</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>“3”</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Since counter is no longer less than `countTokens`, the while loop stops, and the value of `sum` is 6. “1;2;3;4;5;6”.substring(6) = “4;5;6”;

52. (7 pts) Recall that in the `String` class there is a method called `substring` that accepts two int parameters. Suppose that you are asked to write that method as part of the original `String` class. Show the implementation of that method where the only instance method you are allowed to use in the `String` class is `charAt`.

```java
public String substring(int start, int end) {
    String output = “”;
    for (int counter=start;counter<end;counter++)
        output += charAt(counter);
    return(output);
}
```
53. (8 pts) Write a class definition called Grid. The Grid class contains as an instance variable an array of ints of size 25. The constructor of the Grid class, should instantiate the array of ints and fill it with random integers between 0 and 99. You should include one instance method called add whose return type is Grid, and whose parameter list consists of an object of type Grid. The method should return a Grid object where the elements of its int array are the sums of the corresponding elements of the input Grid and the Grid that calls the method. (For two points extra credit include a toString() method in which the elements of the array are displayed in 5 rows and columns.)

```java
public class Grid {
    private int[] numbers;
    public Grid() {
        numbers = new int[25];
        for (int counter=0; counter<numbers.length; counter++)
            numbers[counter] = (int)(100*Math.random());
    }
    public int[] numbers() {
        return(numbers);
    }
    public void setNumbers(int[] numbers) {
        this.numbers = numbers;
    }
    public Grid add(Grid g) {
        int[] array = g.numbers();
        int[] output = new int[25];
        for (int counter=0; counter<25; counter++)
            output[counter] = array[counter] + numbers[counter];
        Grid sum = new Grid();
        sum.setNumbers(output);
        return(sum);
    }
    public String toString() {
        String output = "";
        for (int counter=0; counter<numbers.length; counter++) {
            System.out.print(numbers[counter] + " ");
            if ((counter > 0) && (counter % 5 == 0))
                System.out.println("\n");
        }
    }
}
```