1. What is the typical source for the value(s) for the parameter(s) of the application method?
   1. The command line

The following 2 questions are related.

1. Write a simple Java class that will represents a sphere with the following two properties: radius and volume. You may omit the JavaDoc comments. The radius of an instance of a sphere is fixed; it cannot be changed once the instance is created. The radius is given as the parameter to the constructor. Here is the formula for the volume of a sphere: volume = (3/4)π ⋅ r^3

public class Sphere {

private static int RADIUS;

private double volume;

public Sphere(int radius) {

RADIUS = radius;

calcVolume();

}

private void calcVolume() {

volume = (3.0/4.0) \* Math.PI \* (RADIUS \* RADIUS \* RADIUS);

}

public double getVolume() {

return volume;

}

public static void main(String[] args) {

Sphere sphere = new Sphere(5);

System.out.println(sphere.getVolume());

}

}

1. Recalling the categories of properties we saw in the Properties lab, what kind of property is volume. Briefly explain.
   1. I would say that volume in most cases volume would be a “mutator without an accessor” because you turn the volume up or down, but don’t check it (except by maybe turning it down a bit)
      1. classA a is a reference variable.
      2. new classA(); created Object / Instance
      3. a = new classB() is assign object reference to variable
2. Based on the JavaDoc comment, implement the following method, using the methods listed on page 9. Extra-credit for not using the get method of List.

/\*\*

\* Given a List of floats, the posMaxMin method returns the minimum and

\* maximum positive values from the list in a two-element array, with the

\* minimum positive value as the first element and the maximum positive

\* value as the second element. When there are no positive elements in the

\* List, the return array contains two zeroes.

\* @param vals The List to be searched.

\* @return The array with the minimum and maximum positive values of the given List.

\*/

public float[] posMaxMin(java.util.List<Float> vals) {

java.util.Iterator<Float> iter = vals.iterator();

float min = Float.MAX\_VALUE;

float max = 0;

while (iter.hasNext()) {

float current = iter.next();

if (current > 0) {

if (current < min) min = current;

if (current > max) max = current;

}

}

return (max > 0 && min > 0)

? new float[] {max, min}

: new float[] {0,0};

}

The following two questions are related.

1. List the five (5) primary flow-of-control (branching and looping) statements of Java. Use the keyword that starts a statement as the name of the statement.

|  |  |  |
| --- | --- | --- |
| Branching statements | Looping statements | Decision statements |
| break | for | if |
| continue | while | switch |
| return | do |  |
|  |  |  |
|  |  |  |
|  |  |  |

1. Give simple examples of the syntax for three of the statements listed in the previous question. (For extra credit, indicate optional parts or alternate forms of the statements.)

for (String city : cities) {

getOut(city);

}

while (hungry) {

getFood();

}

do {

eat();

sleep();

writeCode();

} while (alive);

1. The **break** statement can be used: (mark all that apply)
   1. **to end processing within an exception handler.**
   2. to interrupt normal program flow to start debugging.
   3. **to prevent “fall through” in a switch statement.**
   4. **to terminate the execution of a while loop.**
   5. to provide time for the user to get a cup of coffee.
2. The **continue** statement can be used: (mark all that apply)
   1. **to end a single iteration within a for loop.**
   2. to resume normal program flow after a break statement.
   3. to provide a no-op statement for a label.
   4. to exit a then or else part in an if statement.
   5. **to jump to the loop test in a do loop.**
3. The **return** statement can be used: (mark all that apply)
   1. to continue program flow back at the point an error occurred.
   2. **to specify a return value for a method.**
   3. to force a line-break in textual output.
   4. **to exit a method.**
   5. to indicate that the user has come back from getting coffee

The following 2 questions are related.

1. Write a method named equalSpace that takes three parameters: an int (n) and two doubles (lo, hi). The method will return an array of n double values. The array will contain equally spaced values between the endpoints, lo and hi. Include the JavaDoc comment. For example, equalSpace(5, 2.0, 8.0) returns the array {2.0, 3.5, 5.0, 6.5, 8.0}.

public double[] equalSpace(int n, double lo, double hi) {

double spacer = (hi - lo) / (n - 1);

double[] nums = new double[n];

nums[0] = lo;

for (int i = 1; i < n; i++) {

nums[i] = lo+=spacer;

}

return nums;

}

1. Assume there is a printArray method that takes one parameter, an array of doubles, and print it to System.out, as follows:

{2.0 : 3.5 : 5.0 : 6.5 : 8.0}

Call the equalSpace and printArray methods to print out the following: {5.0 : 3.0 : 1.0} (Extra credit if you do this in one line of code.)

* // assuming both methods are static
* printArray(equalSpace(3, 5.0, 1.0));

The following three (3) questions are related.

1. Briefly describe the effect of the final modifier on methods.
2. Briefly describe the effect of the static modifier on methods.
3. Can a method be both final and static? Explain
4. In the Tic-Tac-Toe assignment and the ButtonButton demo and lab, you used ActionListener to handle button clicks. Complete the three steps that you need to complete to handle button clicks:
   1. Create a class that
   2. Write a method that
   3. Call the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ method (from the standard library)
5. java.util.Map is a subinterface of java.util.Collection. True or False Explain the reason. (An answer without an explanation will receive zero points.)
6. Briefly describe the two ways the Java keyword super is used in Java source code. (For partial credit, describe the significance of the keyword.)