Trace the following Bowling Score algorithm (taken for Section 2.9 in the textbook). Use the setup shown below the algorithm.

```
 1 SET totalScore TO 0
 2 SET count TO 0
 3 PRINT "Enter score (-1 to quit): "
 4 INPUT score
 5 WHILE score IS NOT EQUAL TO -1
 6   SET totalScore TO totalScore + score
 7   SET count TO count + 1
 8   PRINT "Enter score (-1 to quit): "
 9   INPUT score
10 SET avg TO totalScore / count
11 PRINT "Average score is " avg
```

Trace setup:

```
Input
=====
95
105
-1
```

line#	score	totalScore	count	avg	output

Complete the program below. See the comments for what the program should do.

```
import java.util.Scanner;

/**
 * The following program asks the user to enter the price of a hamburger
 * and displays the number of pennies required to purchase the burger.
 *
 * Example program interaction:
 *
 *        Enter the price of a hamburger: $1.29
 *        It would take 129 pennies to purchase that burger.
 */
public class Quiz3 {

  public static void main(String[] args) {

    Scanner in = new Scanner(System.in);
    System.out.print("Enter the price of a hamburger: $");









  }
}
```

Trace the following Java program using the long form tracing technique described in the textbook and lecture.

```
1  public static void main(String[] args)
2  {
3    System.out.println("Enter as many scores as desired.  Signal you're done with -1");
4    Scanner in = new Scanner(System.in);
5    int score = in.nextInt();
6    int count = 0;
7    double totalScore = 0.0;
8    while(score != -1) {
9      count++;
10     totalScore += score;
11     score = in.nextInt();
12   }
13   if(count>0) {
14     System.out.println("The average of the scores is: " + totalScore/count);
15   } else {
16     System.out.println("Ugh. The universe appears to be winning.");
17   }
18 }
```

Trace setup:

```
Input
=====
90
110
-1
85
```

line#	score	totalScore	count	output

Consider the following code:

```
{
  Quiz6 obj = new Quiz6();
  obj.setName("Jon Dough");
  System.out.println(obj.getName());
}
```

Complete the Quiz6 class so that the code above works as expected.

```
public class Quiz6 {

  private String name;

}
```

Draw a memory diagram (similar to the one drawn in class) that shows the all of the objects and any local variables for the main method and the subtract method from the Complex class.

```
public static void main(String[] args) {
  Complex c1 = new Complex();
  Complex c2 = new Complex();
  Complex c3 = c1.subtract(c2);
}
```

and from the Complex class:

```
  /**
   * This subtracts a complex number from a complex number
   * @param minus The complex number to be subtracted
   * @return Complex
   */
  public Complex subtract(Complex minus){
    Complex difference = new Complex();
    difference.setReal(this.real - minus.getReal());
    difference.setImaginary(this.imag - minus.getImaginary());
    return difference;
  }
```

True/False

1. In the interest of encapsulation, use local variables instead of attributes/fields whenever possible.
2. Since some of your preliminary code might change in the course of development, you should not test the code until everything is done.
3. When testing a program, it's important to not have any preconceived expectations of what your output should look like.
4. The top-down design methodology is good because it keeps everyone focused on a common goal.
5. The top-down design methodology is good because it avoids "reinventing the wheel."
6. The top-down design methodology is good because it keeps management informed.
7. You can drive any class from a `main` method within that class.
8. You can have multiple classes with a `main` method and, when your program starts, can select which class' `main` method should be used to run your program.
9. Writing a helper method is a good idea if you have a long and complicated method and would like to partition it into several smaller modules.
10. Writing a helper method is a good idea if your class contains two or more methods where some of the code is the same in both methods.

Write the code need to generate the memory map drawn on the board.