```java
package ProblemSet1;
import java.util.Random;
import java.util.Scanner;
public class Problem1 {
    private static MyListNode head;
    private static MyListNode tail;
    public static void main (String args[] ) {
        MyListNode2d();
    }
    public static double MyListNode2d() {
        Random random = new Random();
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter the amount of points you would like to create:");
        int n = keyboard.nextInt();

        for(int i = 0; i <= n; i++) {
            push(random.nextDouble(), random.nextDouble());
        }
        MyListNode printer = head;
        while(printer.getNext() != null) {
            printer = printer.getNext();
        }
        MyListNode[] cV = new MyListNode[4]; //Closest Values
        double distance =0;
        double smallestDistance = 1;
        cV[0] = head;
        cV[1] = head.getNext();
        while(cV[0].getNext() != null) {
            while (cV[1].getNext()!= null){
                cV[0].setX(cV[0].getX());
                distance = Math.pow((cV[0].getX()-cV[1].getX()), 2)+Math.pow((cV[0].getY()-cV[1].getY()),2);
                distance = Math.sqrt(Math.abs(distance));
                if(distance < smallestDistance) {
                    smallestDistance = distance;
                    cV[2] = cV[0];
                    cV[3] = cV[1];
                }
                cV[1] = cV[1].getNext();
            }
            cV[0] = cV[0].getNext();
            if(cV[0].getNext()!=null) {cV[1] = cV[0].getNext();}
        }
        StdOut.printf("\nThe two closest points to each other after rounding are (%.2f, %.2f) and (%.2f, %.2f).", cV[2].getX(), cV[2].getY(), cV[3].getX(), cV[3].getY());
        StdOut.printf("\nTheir true values are: (%f, %f) and (%f, %f).", cV[2].getX(), cV[2].getY(), cV[3].getX(), cV[3].getY());
        StdOut.printf("\nThe distance between these two points is: %f", smallestDistance);
        return 0.0;
    }
    public static void push(double x, double y) {
        if (isEmpty()) {
            head = new MyListNode(x, y, null);
            tail = head;
        }else {
            MyListNode oldHead = head;
            head = new MyListNode(x,y, null);
            head.next = oldHead;
        }
    }
    public static boolean isEmpty() {return head == null;}
}
```

```
Enter the amount of points you would like to create:
12

The two closest points to each other after rounding are (0.52, 0.35) and (0.47, 0.35).
Their true values are: (0.515226, 0.349177) and (0.469080, 0.351764).
The distance between these two points is: 0.046218
```

```
Enter the amount of points you would like to create:
30

The two closest points to each other after rounding are (0.77, 0.49) and (0.76, 0.50).
Their true values are: (0.766777, 0.486039) and (0.760733, 0.497217).
The distance between these two points is: 0.012708
```

```
Enter the amount of points you would like to create:
53

The two closest points to each other after rounding are (0.80, 0.37) and (0.78, 0.37).
Their true values are: (0.795624, 0.369738) and (0.784127, 0.365243).
The distance between these two points is: 0.012344
```

```java
package ProblemSet1;
import java.util.Scanner;

public class Problem2 {
    private static Node head;
    private static Node tail;
    public static void main (String args[] ) {
        StdOut.print(Parentheses());
    }
    public static boolean Parentheses() {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter the string you would like to check to see if it is balanced:");
        String holder = keyboard.nextLine();
        for(int i = 0; i < holder.length(); i++) {
            if(holder.charAt(i) != '}' && holder.charAt(i) != ')' && holder.charAt(i) != ']') {
                push(holder.charAt(i));
            }else {
                if(holder.charAt(i) == '}') {
                    if(head.getChar() != '{') {return false;}else {pop();}
                }else if(holder.charAt(i) == ']') {
                    if(head.getChar() != '[') {return false;}else {pop();}
                }else if(holder.charAt(i) == ')') {
                    if(head.getChar() != '(') {return false;}else {pop();}
                }

            }
        }
        return true;
    }
```

```
Enter the string you would like to check to see if it is balanced:
[()]{}{[()()]()}
true
```

```
Enter the string you would like to check to see if it is balanced:
[(])
false
```

```
Enter the string you would like to check to see if it is balanced:
{([](({})[])}
false
```

Problem 3: Exercise 1.4.6: Give the order of growth (Big-O notation) as a function of `n` of the running times of each of the following code fragments. Please justify your reasoning.

```
a.    int sum = 0;
for (int k = n; k > 0; k /= 2)
  for (int i = 0; i < k; i++)
    sum++;
```

k = 5; k = 2; 3nlg(n)
sum = 5; sum = 7;

The growth order for this code fragment is 3nlg(n). The reason I say this is because the function will increment by this much between each step. I figured this out mathematically by entering numbers and finding the values that they grew to.

```
b.    int sum = 0;
for (int i = 1; i < n; i *= 2)
  for (int j = 0; j < i; j++)
    sum++;
```

```
i = 1;     i = 2;     i = 3;     i = 4;                2n
sum = 1;   sum = 3;   sum = 6;   sum = 10;
```

The growth order for this code fragment is 2n. I say this because the sum is growing in increments of double what the n value is.

```
c.    int sum = 0;
for (int i = 1; i < n; i *= 2)
  for (int j = 0; j < n; j++)
    sum++;
```

```
i = 1;     i = 2;     i = 3;     i = 4;     i = 5;     n^2
sum = 5;   sum = 10;  sum = 15;  sum = 20;  sum = 25;
```

The growth order for this code fragment is n^2. I say this because the final value ends up being the square of the value you put in when all of the code has been gone through.

Problem 4: Creative Problem 1.4.24: *Throwing eggs from a building!* Suppose that you have an n-story building and plenty of eggs. Suppose also that an egg is broken if it is thrown from floor F or higher and intact otherwise. First, devise a strategy to determine the value of F that uses O(lg n) throws (and so breaks only O(lg n) eggs). Then, find a way to reduce the number of eggs broken to O(lg F). *Note: lg n is shorthand for* $\log_2 n$.

To find the value of F I would use a binary search method. I would take the max height and the minimum height and I would find the point directly in the middle of them. I would then drop the egg from there. If it breaks, I would move down to halfway between the midpoint and the ground. Otherwise, I would move up into the middle between the midpoint and the max height. Then after each egg drop I would move either up or down depending on if the egg breaks or not and eventually I will find the exact height at which F is defined. The time complexity of this method is lg(n).

If you were to use an insertion sort, you would be able to find the value of F faster. If you were to start at a certain height and increment the same amount every time, dropping at egg at each of these stops. You would then do a bucket sort in the smaller range and your time complexity will be lower.