**Birla Institute of Technology & Science – Pilani**
**Second Semester 2016-17**

**Date: 18th April, 2017 (Tuesday)**

**ONLINE TEST (OPEN BOOK)**

| | | |
|---|---|---|
| **Course Name** | **:** | **Object Oriented Programming** |
| **Course No** | **:** | **CS F213** |
| **MM** | **:** | **90 Marks** |
| **Time** | **:** | **8:00 P.M – 11:00 P.M [180 Minutes]** |

**[Expected Time: 150 Minutes (60 Minutes for Understanding + 120 Minutes for Coding)]**

**Important Instructions:**

1. You are given an executable java file named Online.java. Write the whole java code in this file only. Rename the file as P<Your Idno>.java. For example, if your id no is 2015A7PS123P then name of file should be P2015A7PS123.java.
2. The examination is of 90 marks. Out of these 90 marks, 75 marks are for coding and 15 marks are for running and execution of the code.
3. Save your work regularly and periodically. All machines are on backup but still you are advised to save your work. You are responsible for the safekeeping of your code. No query regarding loss of code by any means will be considered later on.
4. IT IS ADVISED TO FIRST READ AND UNDERSTAND THE WHOLE PROBLEM CAREFULLY AND THEN START WRITING THE CODE.

## Problem Description

The problem is regarding the computation of a root of a polynomial equation by using two famous root finding methods namely “Bisection Method’ and ‘Regula Falsi Method”.

A polynomial equation f(x) for variable ‘x’ is defined as follows

$$F(x) = a_0 . x^0 + a_1 . x^1 + a_2 . x^2 + \ldots\ldots\ldots\ldots + a_n . x^n$$

Where $a_0$, $a_1$, ……., $a_n$ are known as coefficients and ‘n’ is known as the degree of a polynomial. The root of the equation is that value ‘x’ where value of F(x) = 0.

You are given a the whole code of two classes namely ‘Polynomial’ and ‘PolynomialList’ which encapsulates the features of a given polynomial and list of polynomials. Skeleton code for these classes is as given [For More Details Please Look in The Online.java Files]:

```
class Polynomial
{
        private  ArrayList<Integer>      coefficients;      // Coefficients of Polynomial
        private  int                     degree;            // Degree of Polynomial
        private  Random                  r;                 // A Random Number Generator Variable
        private  double                  rootValue;         // Root Value of the Polynomial
        private  boolean          isRootValueComputed; // Flag To Check if Root Exists or Not

        // Constructor Method
        Polynomial(int degree)
        {
        /* This Constructor Method generates a polynomial equation of suitable degree by generating
           various random numbers */
        }
```

```
        // Accessor Methods
        public   ArrayList<Integer>        getCoefficients()                  { return coefficients;}
        public   int                       getDegree()                        { return degree; }
        public   double                    getRootValue()                     { return rootValue;}
        public   boolean          getIsRootValueComputed()       { return isRootValueComputed; }

        // Method to Compute the Value of a Polynomial at 'x'
        public   double           computeValueAt(double x) {      }
        // ToString() Method
        public   String   toString()        {         }
        // Mutator Method for Updating root value
        public   void     setRootValue(double value)      {         }
        // Mutator Method for Updating root value flag
        public   void     setIsRootValueComputed(boolean value)         {         }
}// End of class Polynomial

class PolynomialList
{
        private  ArrayList<Polynomial>   polynomialList; //        Polynomial List
        private  Polynomial                        currentPolynomial;
        //       The above field represents the currently selected polynomial from the Polynomial List

        // Constructor Method
        PolynomialList(){ }

        // Accessor Method for Polynomial List
        public   ArrayList<Polynomial> getPolynomialList()        { }

        // Accessor Method For currentPolynomial
        public   Polynomial                getCurrentPolynomial()   { }

        // Mutator Method For currentPolynomial
        public   void  setCurrentPolynomial(Polynomial p)         {}

        // Accessor Method to Retrieve a Polynomial From a Given Index
        public   Polynomial       getPolynomialAtIndex(int index) { }

        // Method to Add a Polynomial in Polynomial List
        public   void     addPolynomial(Polynomial p)    {}

        // The Following Method Initializes Polynomial List by Randomly Creating 10 Polynomials
        public   void     intializePolynomialList(int no)    {}
        }
        // Method to display the Created Polynomials on System.out
        public   void     displayPolynomialList() {}
}// End of class
```

The PolynomialList class defines a special instance field named ‘currentPolynomial’ whose value is set to one of the polynomials in the list and this value is selected in sequence. Both the root finding threads (BisectionThread and RegulaFalsi Thread Explained Later) computes the root value of this ‘currentPolynomial’.

The pseudo code description for the for the Bisection Root Finding Method is given below

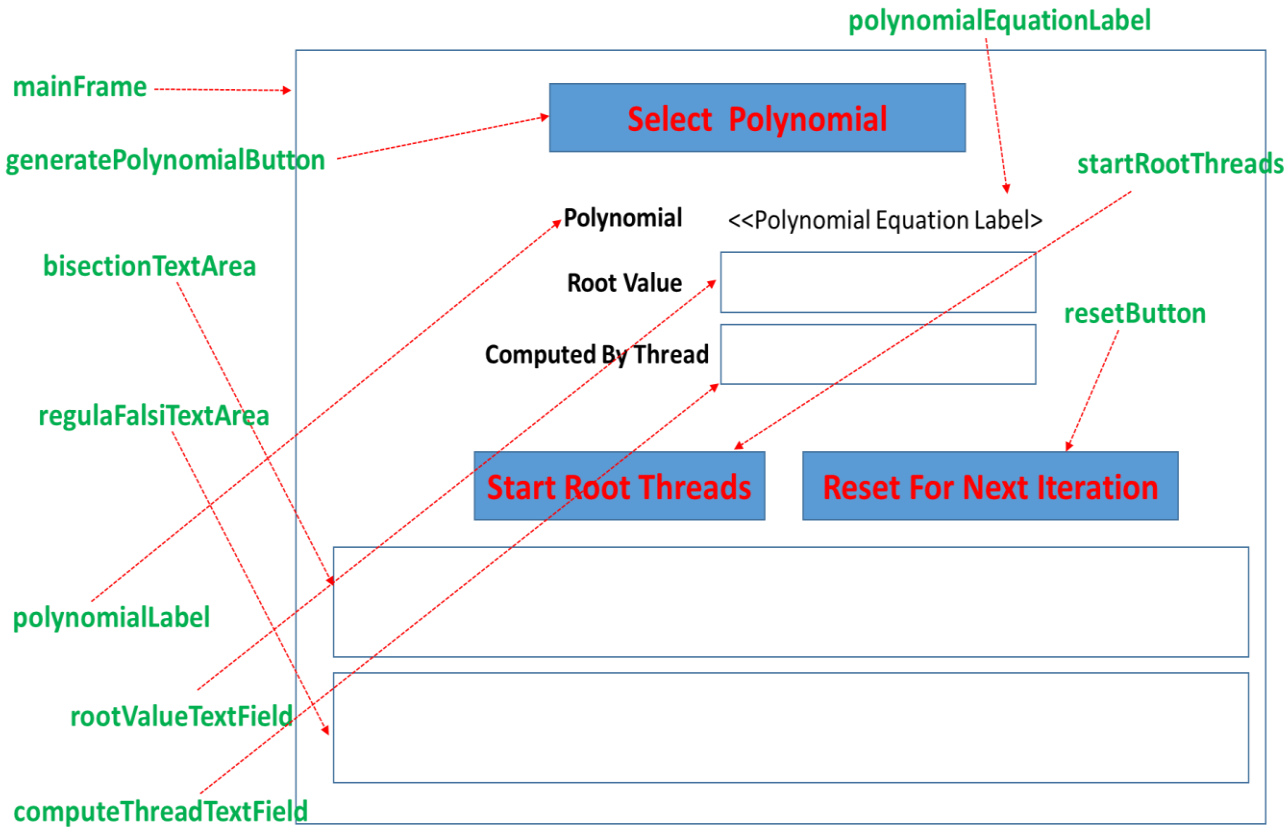
1. For a given polynomial f(x) , select two values ‘a’ and ‘b’ for variable ‘x’ such that f(a) * f(b) < 0
2. Compute a mid point value c = (a+b) / 2
3. IF | f(c) | < 0.0001 OR | b – a | < 0.0001 then ‘c’ is root and stop the procedure

4. IF f(a) * f(c) < 0 Then b = c ELSE a = c and Repeat Steps 2 to 4
5. IF steps 2 to 4 have been repeated for 20 times and condition of Step 3 is not satisfied Then exit displaying " Method Does not Converge"
6. END

The pseudo code description for the for the Regula Falsi Root Finding Method is given below

1. For a given polynomial f(x) , select two values 'a' and 'b' for variable 'x' such that f(a) * f(b) < 0
2. Compute a Value c = [ a * f(b) + b * f(a) ] / [ f(b) – f(a)]
3. IF | f(c) | < 0.0001 OR then 'c' is root and stop the procedure
4. IF f(b) * f(c) < 0 Then a = c ELSE b = c and Repeat Steps 2 to 4
5. IF steps 2 to 4 have been repeated for 20 times and condition of Step 3 is not satisfied Then exit displaying " Method Does not Converge"
6. END

You are given two incomplete class named 'BisectionMethod' and 'RegulaFalsiThread'. You have to complete their run methods by incorporating the logic mentioned above. The execution of the given file displays frame window as shown on the next page. [Note: Green Color Text shown in the Figure is the actual name of Variables used in the given Java File].




The driver method uses a polynomial list variable 'pList' which is populated with 10 different polynomials.

When ‘Select Polynomial’ button is pressed the following tasks are carried out.

1. A polynomial from a polynomial list is selected in sequence and made the currentPolynomial and equation kount is incremented and if all the polynomials have been selected then exit the program.
2. Displays the string form of ‘currentPolynomial’ over frame [Via polynomialEquationLabel]
3. Enables the ‘Start Root Threads’ button and disables itself

When ‘Start Root Threads’ button is pressed the following tasks are carried out.

1. If any of the threads is in waiting state then they are notified
2. If any of the threads is in suspended state then they are resumed
3. Disables itself

When ‘Reset For Next Iteration’ button is pressed the following tasks are carried out.

1. currentPolynomial of pList is set to null and display equation is removed
2. Text Field Values are Reset
3. Enables ‘Select Polynomial’ button and Disables itself

**Working of BisectionThread and RegulaFalsiThread**

The BisectionThread computes the root of the ‘currentPolynomial’ of pList using Bisection Method. Both the threads enters into wait() state during their time slice if the value of currentPolynomial is null. Each root thread uses two initial guess values during the first iteration and these guess values have to supplied from System.in. Each thread perform its work for one iteration and then enters into wait() state by notifying the other thread. Each thread will proceed with the iteration only if the root value of the currentPolynomial has not been computed so far. If value has been computed by opposite thread then it suspends itself and wait for the resumption for the next polynomial.

Each root thread tries to find the root in maximum of 20 iterations. If in 20 iterations it is able to find out the root value then it updates the attributes of the currentPolynomial and suspends itself and waits for its resumption for the next selected polynomial.

If any thread is not able to find out the root is 20 iterations then it suspends itself by printing the trace in its respective text area field. [For more details Please Look at the demo Video and Sample Executable Trace File]. Each root thread runs for only one iteration at a time and then enters into waiting state by notifying the opposite root thread. Whether any root thread is able to find out root value or not in its allotted time slice, it will enables the resetButton just before suspending itself.

[Note: Between successive root finding iterations for the currentPolynomial, the roots threads enters into waiting state by notifying the opposite thread. However, when the root value is computed then the threads suspends for the currentPolynomial and waits resumption for the next polynomial].

**What You have to do?**

1. Complete the Thread Classes
2. Complete the ActionsListeners for buttons.

**Note: Write the code as per the execution trace. Marking Scheme is based upon execution trace of each thread.**