CHAPTER V
IMPLEMENTATION AND TESTING

5.1 Implementation

The first step that user have to do is to input the equation and the precission of the root which user want. The precission is the range between 0 to the result when the root become answer of the equation. To get the value of precision and the function as follows :

```
public void actionPerformed(ActionEvent e){
    try{
        fungsi = func.getText();
        marg = margin.getText();
        func2.setText(fungsi);
        GAProject myGa = new GAProject();
        myGa.fungsi = this.fungsi;
        myGa.setMargin(marg);
        myGa.cetakData();
        myGa.consoleToTxt();
        FileReader reader = new FileReader("log.txt");
        BufferedReader br  = new BufferedReader(reader);
        output.read(br,null);
        br.close();
        output.requestFocus();
        root.setText(myGa.rootSimple);
        index.setText(myGa.in);
        fitness.setText(myGa.fittest);
        totalFit.setText(myGa.totFitness);
    } catch(Exception e2){}

    //output.setText(out);
}
```

5.1.1 GUI method to catch equation and root precision

And then the function is stored to variable fungsi and precision is stored into variable margin. Variable fungsi will be used to determine and arrange the objective function and precission or variable margin will be criterion that application use to determine the solution as well.

```
private void setFungsi(String f){
fungsi = f;
}
```

5.1.2 Storing equation into variable

```
public void setMargin(String ma){
    margin = Double.parseDouble(ma);
}
```

5.1.3 Storing root precision into variable

After get the equation the application process the equation become to objective function that will used to determine the fitness value of the individual. In the process to make the equation become objective function the application has to through 3 steps.

The first step is split function to some parts that separated by matemathics operand(+,-). For example : **f(x)=3x²+2x-1** splited into 3 parts. Part 1 : **3x²**, part 2 : **2x** , part 3 : **-1**

```
private String[] splitFungsi(String fungsi){
    String mod = fungsi.replaceAll("\\-","+-");
    String[] splited = mod.split("\\+");
    return splited;
}
```

5.1.4 Split equation code.

The second step is to get coefficient of each part. The coefficient is very important to obejctive function. In this case operand mathematics will not affect to the equation because negative or positive value will be the coefficient sign. For example : **f(x)= 3x²+2x-1** the coefficient is (3,2,-1).

```
private String getKoefisien(String kof,String var){
    if(kof.equals("x")) return "1";
    if(kof.contains("-x")) return "-1";
    String[] splited2=kof.split(var);
    if(splited2[0].equals("")){splited2[0]="1";
        return splited2[0];}
    else return splited2[0];

}
```

5.1.5 Get coefficien code

The third step is getting the degree of variable of each part. For example : **f(x)= $3x^2$+2x-1.** The degree is(2,1,0). The application get degree by find the number after variable(x). If the system find only variable without following number then the system decide the degree is 1 and if the system don't find the variable then the system determine that the degree is 0.

```
private String getDegree(String deg,String var){
    String[] splited3 = deg.split("\\^");
    int i=splited3.length;
    if(i==1){
       if(splited3[0].contains(var)){
       return "1";}

        else return "0";
    }
    if(i>1) return splited3[1];
    else return "0";

}
```

5.1.6 Get degree from equation.

After the system collect the coefficient and the degree from the equation the system will start to intialize the population. The population will be initialized by random number generator. Random number generator use library from java(java.util.random) and initial gaussian number for individual

value. The individual stored to linked list.

```
private double RandomGaussianGenerator(double stdDev, double Mean){
    java.util.Random rand = new java.util.Random();
    return ((rand.nextGaussian()*stdDev)+Mean);
}
```

5.1.7 Random Gaussian Number

The system will encode the individual into the 64-bit binary code to prepare the crossover and mutation process in the application. The reason why the application encode the individual into 64-bit binary code is because double is long number. 64-bit Binary for double can be divided into three important parts. The first part is the first number of 64-bit binary code. It determine the sign of the number. If the first number of 64-bit binary code is 0 then the value is positive and if the first number is 1 then the value is negative. And the second important part is the number 2-12 of 64-bit binary. It determine the exponent of the value. The third part is 13-64 number. It is called mantis and it determine the number of value.

```
private String doubleToBinary(double value){

    int krg=64;
    int tot=0;
    String nol="";
    String biner = Long.toBinaryString(Double.doubleToRawLongBits(value));
    if(biner.length()<64){
        //tot=krg-biner.length();
        biner ="00"+biner;
    }

    return biner;

}
```

5.1.8 Binary Encoding

After the population initialized and encoded the system will start to arrange the objective function. Objective function is consist of coefficient and degree that got from splited equation. After objective function has arranged. The system will send the individual(gaussian value) to objective function and make the result of each individual.

```
private double ObjectiveFunction(String[] koefisien, String[] pangkat,double kromosom){
    double hasil=0;
    double [] kof=new double[koefisien.length];
    double [] deg=new double[koefisien.length];

    for(int i=0;i<koefisien.length;i++){
          kof[i]=Double.valueOf(koefisien[i]);
          deg[i]=Double.valueOf(pangkat[i]);
    }

    for(int i=0;i<koefisien.length;i++){
        hasil +=kof[i]*(Math.pow(kromosom,deg[i]));
    }
    return hasil;
}
```

5.1.9 Objective function method

Fitness value of each individual is determined by the value of the result from objective function. The application use the formula **fitness = 10/result of objective function** to get number of fitness value of each individual.

```
private double getFitness(double HasilObj,int irasional){
    if(irasional==1){
    if(HasilObj < 0) HasilObj = HasilObj * -1;
    double fittest=(10/HasilObj);
    return fittest;
    }
    else{
    if(HasilObj < 0) HasilObj = HasilObj * -1;
        if(HasilObj>maxi) {maxi=HasilObj;}
        double fittest = maxi-HasilObj;
        return fittest;
    }
}
```

5.1.10 Get fitness method

After the system get the fitness value of the each individual. The system will select the individual who has the best or highest fitness value. The individual who has the highest score is the best solution of 1 population.

The mutation process start for each individual. The system has determined the mutation rate. De Jong said for wide-range problem the parameter for mutation rate is 0.01. The mutation process is randoming the point that will mutate. Randoming point is made by random function from java. The individual which sent into mutate method had been formed as a 64-bit binary code. For example : The random number is 26.

Before mutation :

1110100100011100111111011101000111101001000111001111110111010001

After Mutation :

1110100100011100111111011001000111101001000111001111110111010001

And this is the mutation method :

```
private String mutate(double indiv){
    java.util.Random rd = new java.util.Random();
    String bits= doubleToBinary(indiv);
    StringBuffer bf = new StringBuffer(bits);
    for(int i=0;i<bits.length();i++){
        if(rd.nextFloat() < mutationRate){
            if(bits.charAt(i) == '1'){
                bf.setCharAt(i,'0');
                return bf.toString();
                }
                else {
                    bf.setCharAt(i,'1');
                    return bf.toString();
                }
            }
        }
        return bf.toString();
}
```

5.1.11 Mutation Method

After the mutation process was done. The individual who has best score selected to crossover with other individual. The system use one-point crossover to implemented into crossover process. The point of crossover is determined by random number. For example :

The random point : 32

Individual 1 :

11101001000111001111110111010001 11101001000111001111110111010001

Individual 2:

10010100100011100111111011101000 11110100100011100111111011101101

Offspring 1 :

11101001000111001111110111010001 11110100100011100111111011101101

Offspring 2:

10010100100011100111111011101000 11101001000111001111110111010001

Here is the crossover method :

```
private String[] crossOver(double indiv1,double indiv2){
    String indivBits1 = doubleToBinary(indiv1);
    String indivBits2 = doubleToBinary(indiv2);

    String[] newIndiv = new String[2];
    String nIndiv1 = "";
    String nIndiv2 = "";
    java.util.Random rd = new java.util.Random();
    int crossPoint = rd.nextInt(indivBits1.length());
    String a=indivBits1.substring(0,crossPoint);
    String b=indivBits1.substring((crossPoint),indivBits1.length());

    String c=indivBits2.substring(0,crossPoint);
    String d=indivBits2.substring((crossPoint),indivBits2.length());

    nIndiv1 = a+d;
    nIndiv2 = b+c;

    newIndiv[0] = nIndiv1;
    newIndiv[1] =nIndiv2;
    return newIndiv;
}
```

5.1.12 CrossOver method

After Crossover process is done the system will determine the best individual through the evaluation process. And the system will intialize the population until the precision of the root is fulfilled.

5.2 Testing

The Project will try to find the best root for a equation using genetic algorithm. This test consist of 10 equation and various precision value. These are the result table :

Figure 5.2 Testing table.

No	Equation	Precision	Root	Conclusion
1	x^3-2x-1	0.1	-0.6134745	The system only make 1 iteration because the precision value is to big and the equation is too simple. The system needs 0.0 second.
2	x^3-2x-1	0.01	-1.0018219	The system makes 4 iteration. The system takes a litle bit longer time to find the root.0.1
3	x^3-2x-1	0.001	-1.0007121	The system makes 10 iteration. The system needs 0.8 second to find the root.
4	x^3-2x-1	0.0001	-0.9999642	The system needs 8 second to execute and make 500 iteration due to highest precision in this testing.
5	$5x^5$-$2x^4$-$3x^2$-2x+2	0.1	0.9852316	The system make 1 iteration because the precision smaller bu the equation more complex than number 1
6	$5x^5$-$2x^4$-$3x^2$-2x+2	0.01	1.0038712	The system takes longer time and make more iteration when precision number is changed to be smaller. Smaller = more accurate
7	$5x^5$-$2x^4$-$3x^2$-2x+2	0.001	1.004453	The system makes 47 iteration and need still less than 1 second to find

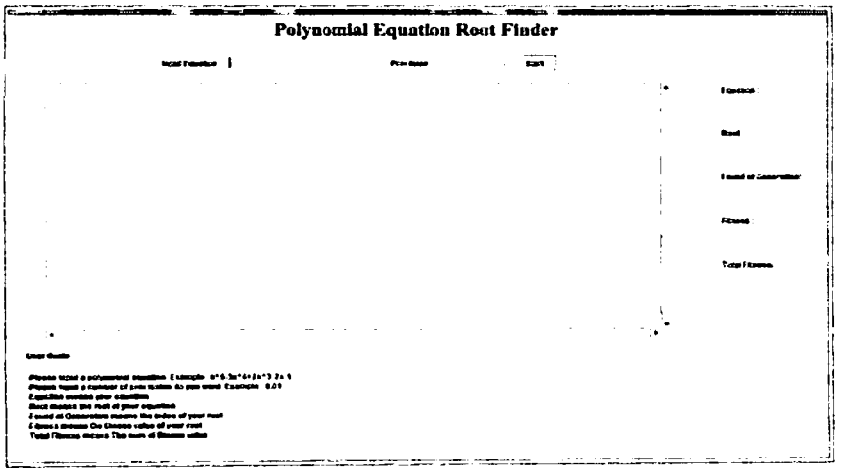
				the root.
8	$5x^5-2x^4-3x^2-2x+2$	0.0001	-	The system request time out for complex equation and the highest accuracy in this testing.
9	3x^7-2x^5+3x^4-5x^2+2x-1	0.1	1.0015643	Because the number of accuracy is small. The system juts make 2 iteration and execute less than 1 second.
10	3x^7-2x^5+3x^4-5x^2+2x-1	0.01	1.0001576	The system makes 28 iteration because of higher number of accuracy with same equation. Still takes less than 1 second to execute.
11	3x^7-2x^5+3x^4-5x^2+2x-1	0.001	1.0000162	The system makes 304 iteration because of higher number of accuracy and takes 3 second to execute it.
12	3x^7-2x^5+3x^4-5x^2+2x-1	0.0001	-	The system request time out due to more complex equation and highest precision in this testing.
13	x^10-2x^8+3x^5-5x^3+2x^2-1	0.1	-1.2640261	The system takes short time during and only make 1 iteration due to small precision number. Exceuted in less than 1 second.
14	x^10-2x^8+3x^5-5x^3+2x^2-1	0.01	-0.4963420	The system makes 9 iteration and takes less than 1 second to execute.
15	x^10-2x^8+3x^5-5x^3+2x^2-1	0.001	-0.4948898	The system need 1 second to execute and takes 400 iteration to find root.
16	x^10-2x^8+3x^5-5x^3+2x^2-1	0.0001	-	The system request time out due to more complex equation and highest precision in this testing.
17	6x^12-2x^8+3x^5-5x^3+2x^2-1	0.1	-0.4783443	The system needs 2 iteration and takes less than 1 second to find the root due to small precision number
18	6x^12-2x^8+3x^5-5x^3+2x^2-1	0.01	-0.4951658	The system needs 14 iteration and takes time less than 1 second due to more complex equation and

				higher precision number
19	6x^12-2x^8+3x^5-5x^3+2x^2-1	0.001		The system takes 502 iteration and takes 8 second to fin the root of equation.
20	6x^10-2x^8+3x^5-5x^3+2x^2-1	0.0001		Due to the complexities and the height of precision number. The system is takes too long time and decide to stop the iteration.

5.3 Main Interface Window

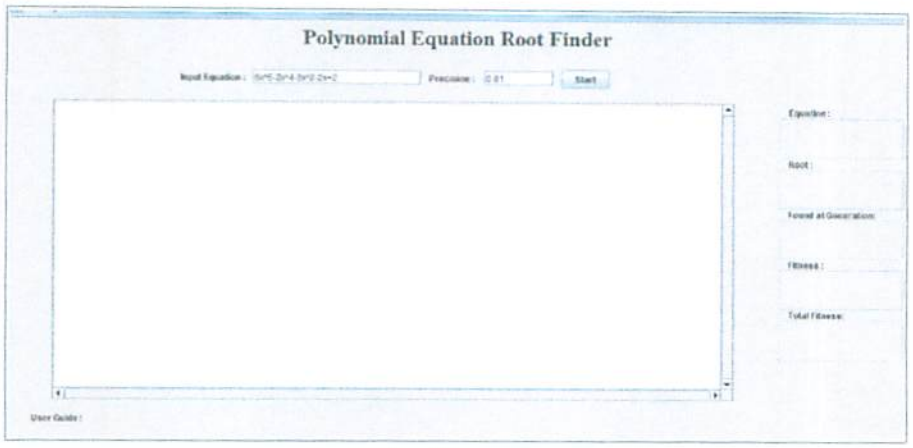
This is the main interface of this application. The interface or GUI class was build using java GUI. The interface use 3 layout manager on three panel and content area. They are Box Layout,Flow Layout and Box Layout. The interface can be separated into 4 important part.

1. Title Panel is for the information or name of this application.
2. Input or user panel for user give equation and accuracy number that user want and there is a button to start the application.
3. Display area is for user can look the information of the application result in finding root of polynomial equation with high-level accuracy.
4. User guide is for help or inform user about this application and how to use this application.



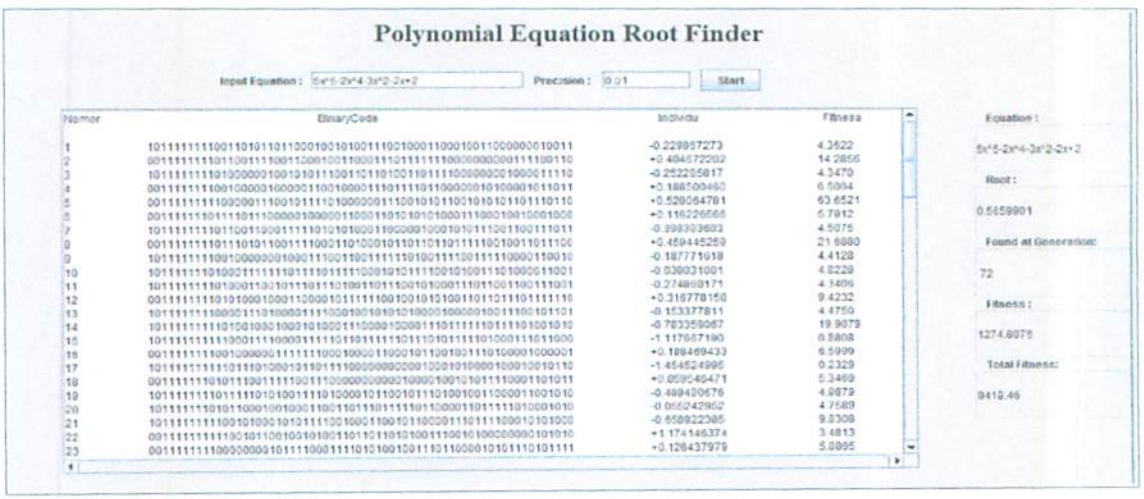
5.3.1 Main Interface Window

The first step user input the polynomial equation into form and fill the precision form with number that user want. The application give the information how to use and inform the user what is texts and form on the application.



5.3.2 User fill the equation and precision form

After user fill the equation form and precision form. User will press the start button to start process of finding root. If the root is found the information text field will be field with some information.



5.3.3 Application shows the result and information.

Application shows the information and result to the user. Right side of the program shows the result and information that application found. Equation text field shows the equation that user input. The result or root is the number in the root text field. It is a complex number because the application need to make sure how precise the root. And then the "Found at Generation" explain the index of the solution/root of equation. Fitness text field explain fitness value of solution or root. It can compared to other individual. Total fitness is the sum of the fitness value of all individual. The answer is on the root text field.