Alisa Ivanova BS19-06

Differential equations

Computational Practicum

Contents:

1. General information & UML-diagrams
2. Process description
3. Exact Solution
4. Implementation of Exact solution
5. Implementation of the Euler method, LTE&GTE
6. Implementation of the Improved Euler method, LTE&GTE
7. Implementation of the Runge-Kutta method, LTE&GTE
8. Comparing all the plots

1. General Information

Given the initial value problem with the ODE of the first order and some interval:

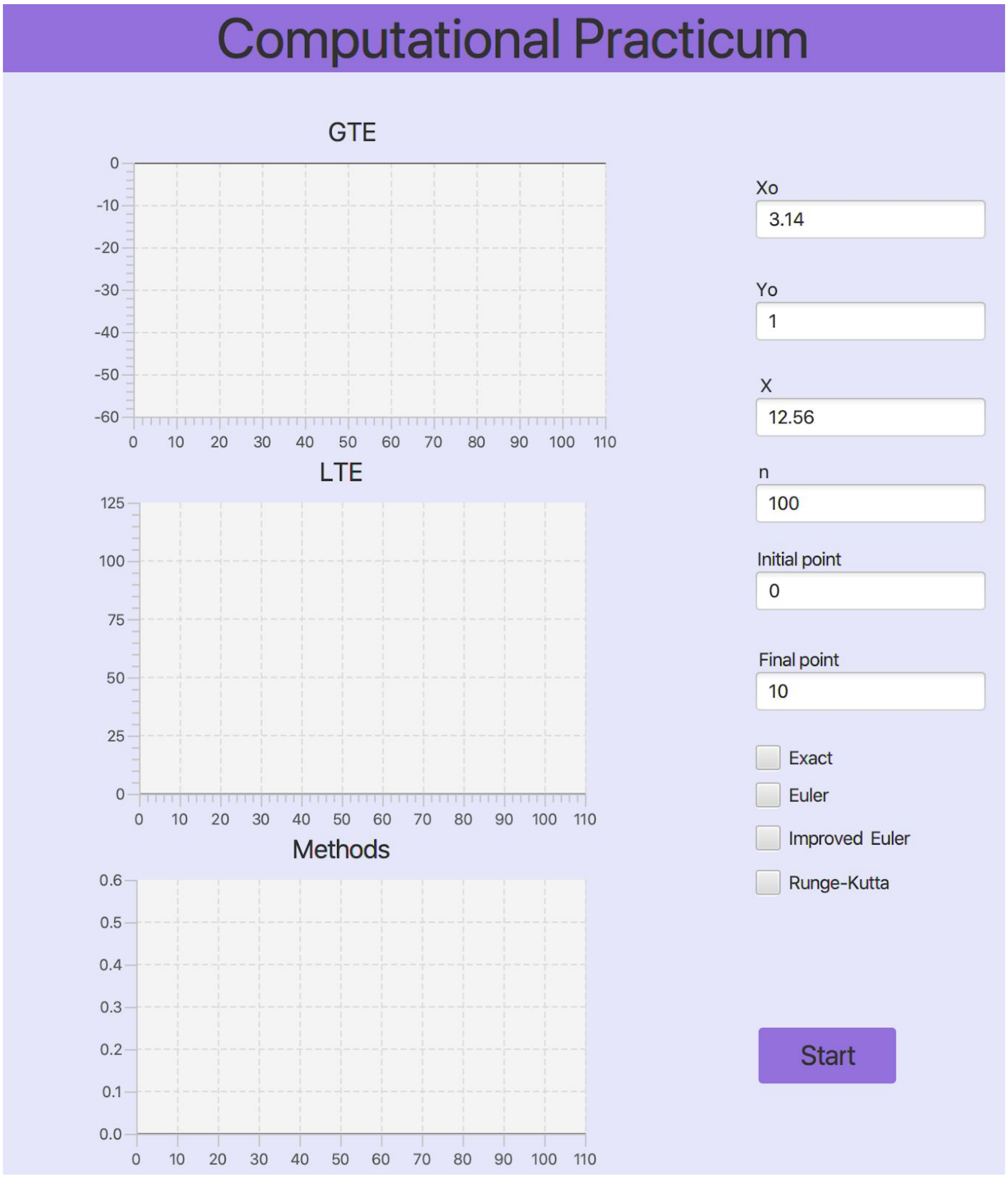
$$\begin{cases} y' = f(x, y) \\ y(x_0) = y_0 \\ x \in (x_0, X) \end{cases}$$

In my case (variant 5):

$$y' = f(x, y) = y/x + x \cos x$$
$$y(\pi) = 1$$
$$x \in (\pi, 4\pi)$$

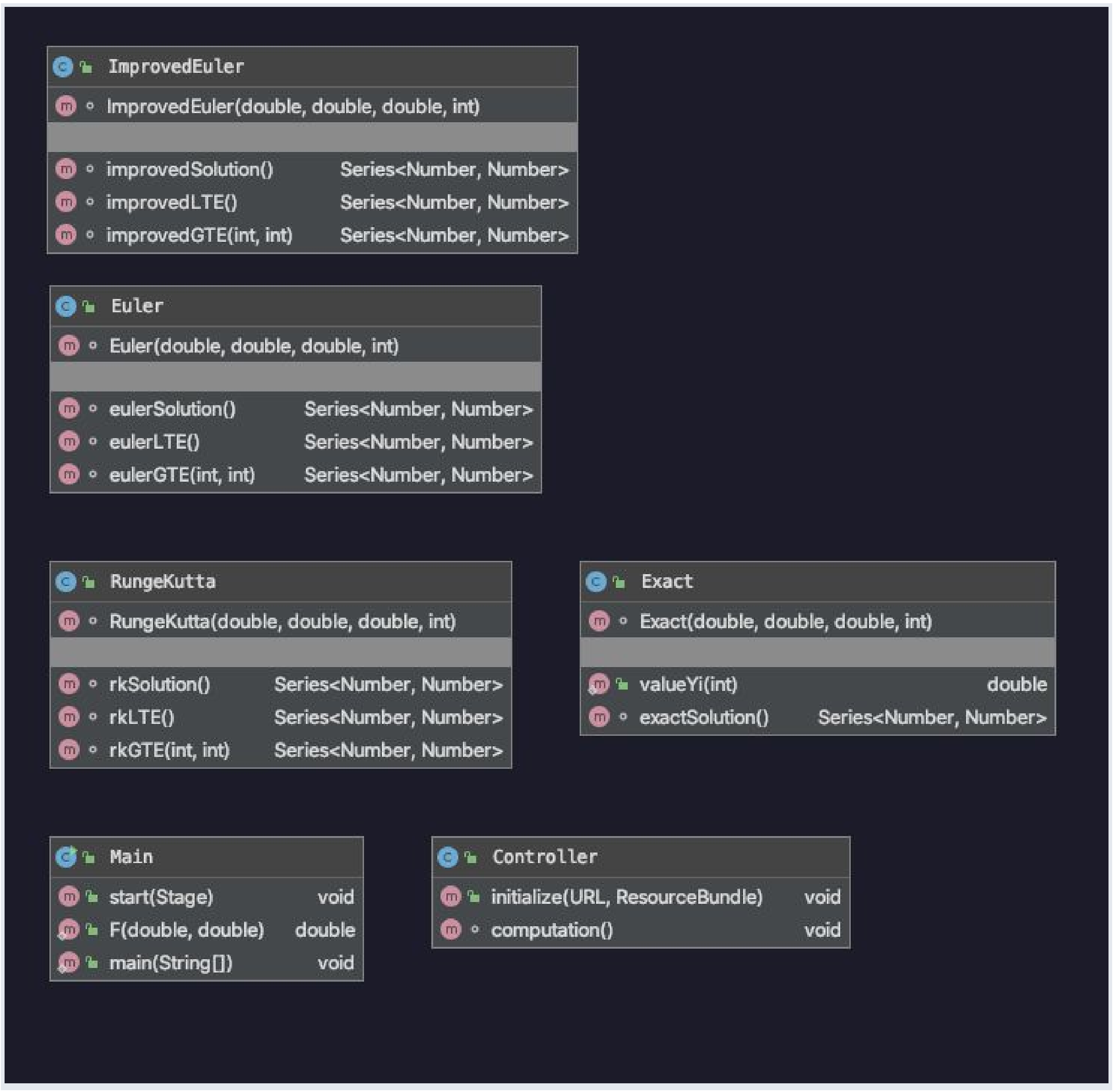
For the computational part of the practicum, I used Java programming language. For creating GUI I used JavaFX library with SceneBuilder application.

The main window:



There are 3 2-axis LineCharts in the left part and 6 input text fields (that provide the possibility to change all the input fields), 4 checkboxes and button at right part.
Input text fields: x0 (initial value of x), y0 (initial value of y), X (right endpoint of the interval), n(number of steps), initial point, final point (left and right boundaries of integration steps).
Checkboxes: Exact plot, Euler plot, Improved Euler plot, Runge-Kutta plot
Button: *Start* button (that initiate all computations)

UML-diagrams of classes



```
ImprovedEuler
ImprovedEuler(double, double, double, int)

improvedSolution()      Series<Number, Number>
improvedLTE()           Series<Number, Number>
improvedGTE(int, int)   Series<Number, Number>

Euler
Euler(double, double, double, int)

eulerSolution()         Series<Number, Number>
eulerLTE()              Series<Number, Number>
eulerGTE(int, int)      Series<Number, Number>

RungeKutta
RungeKutta(double, double, double, int)

rkSolution()            Series<Number, Number>
rkLTE()                 Series<Number, Number>
rkGTE(int, int)         Series<Number, Number>

Exact
Exact(double, double, double, int)

valueYi(int)                              double
exactSolution()           Series<Number, Number>

Main
start(Stage)          void
F(double, double)   double
main(String[])        void

Controller
initialize(URL, ResourceBundle)   void
computation()                     void
```

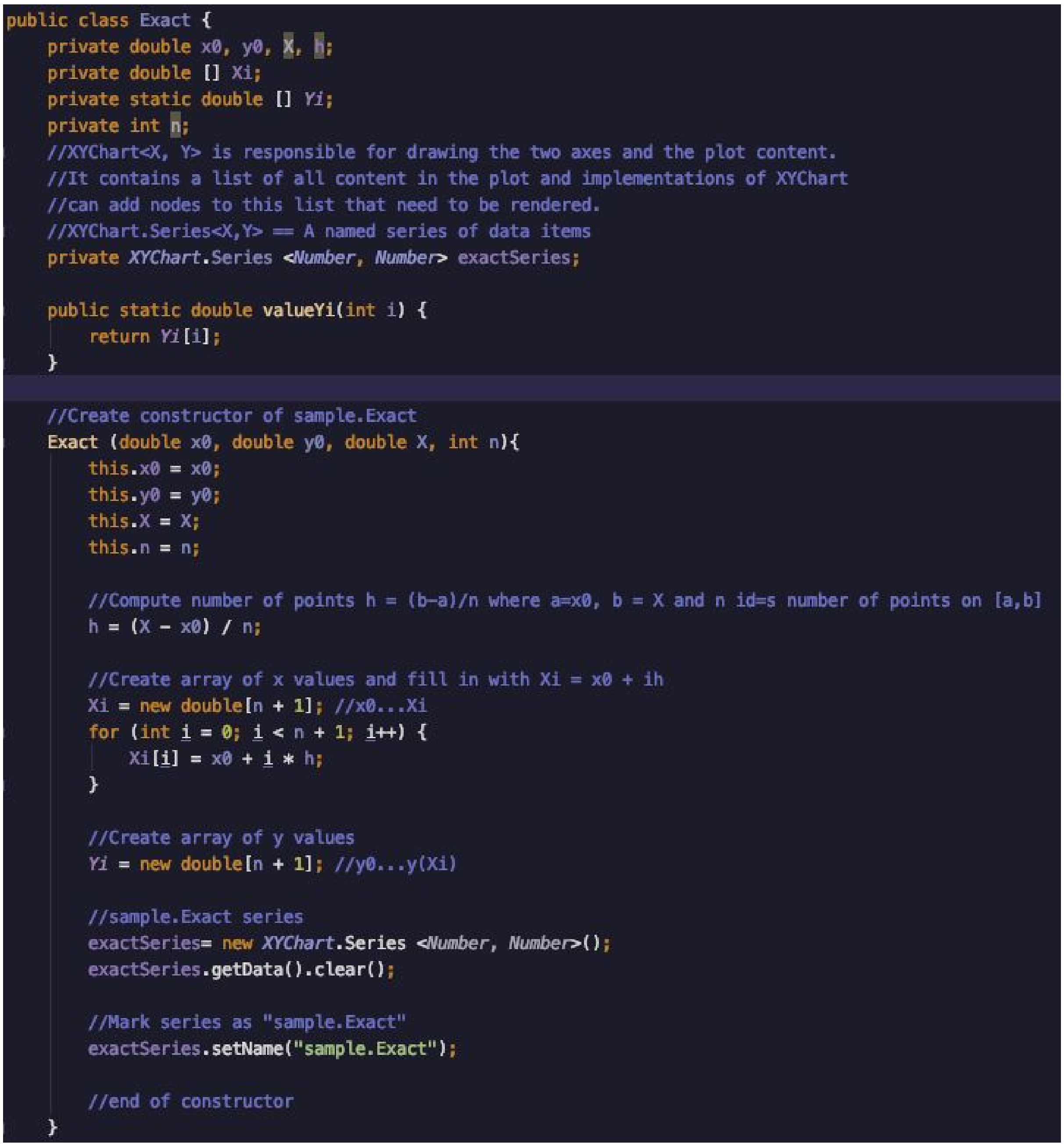
On the diagrams have shown that classes have ***constructors*** and ***methods*** (will be described in implementation parts of the report)

Constructors

Constructors contain:

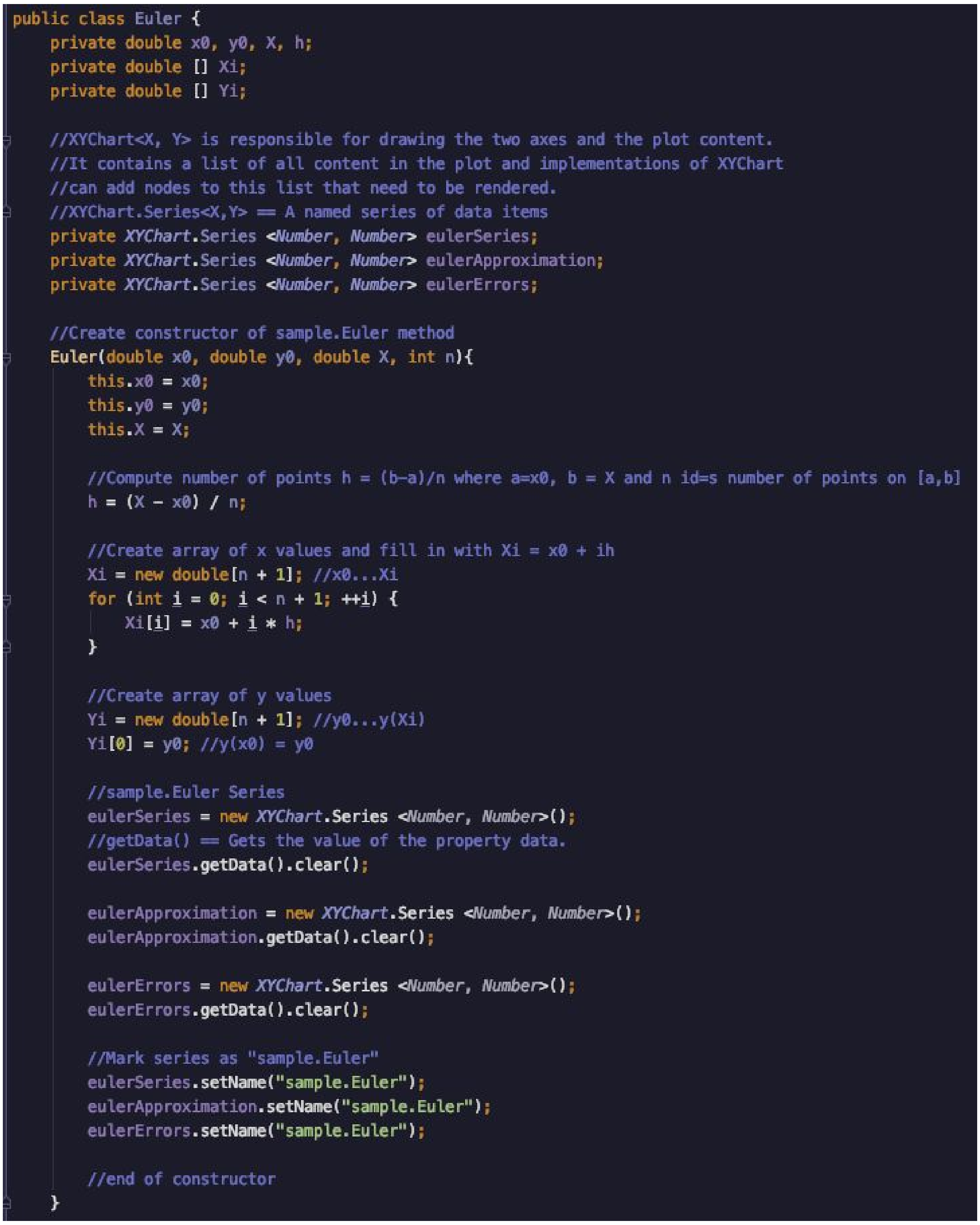
- Computation of h value
- Creation of arrays of Xi and Yi
- Creation of Series for methods/errors/approximation
- Setter for series

1. Exact class



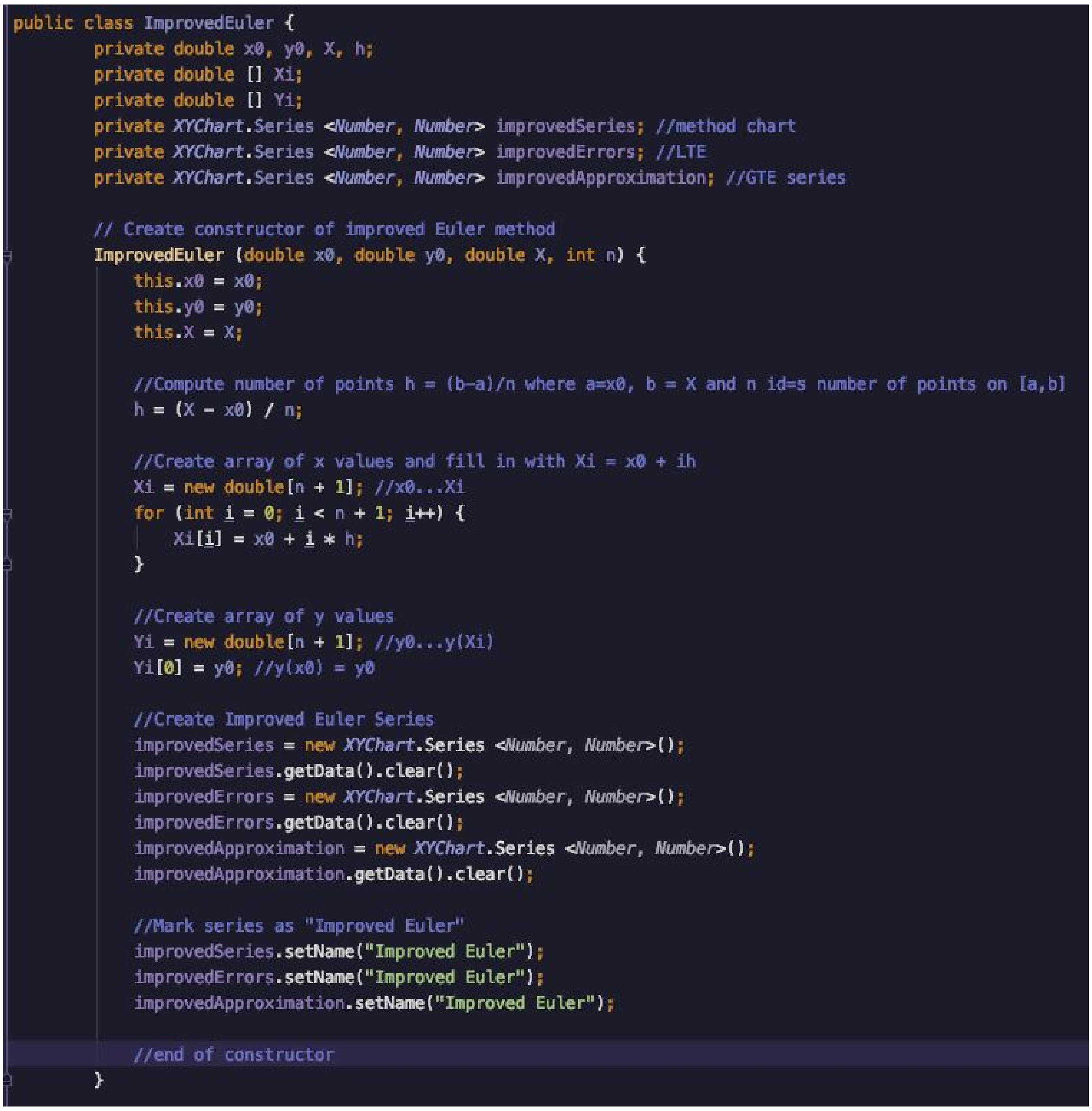
```java
public class Exact {
    private double x0, y0, X, h;
    private double [] Xi;
    private static double [] Yi;
    private int n;
    //XYChart<X, Y> is responsible for drawing the two axes and the plot content.
    //It contains a list of all content in the plot and implementations of XYChart
    //can add nodes to this list that need to be rendered.
    //XYChart.Series<X,Y> == A named series of data items
    private XYChart.Series <Number, Number> exactSeries;

    public static double valueYi(int i) {
        return Yi[i];
    }

    //Create constructor of sample.Exact
    Exact (double x0, double y0, double X, int n){
        this.x0 = x0;
        this.y0 = y0;
        this.X = X;
        this.n = n;

        //Compute number of points h = (b-a)/n where a=x0, b = X and n id=s number of points on [a,b]
        h = (X - x0) / n;

        //Create array of x values and fill in with Xi = x0 + ih
        Xi = new double[n + 1]; //x0...Xi
        for (int i = 0; i < n + 1; i++) {
            Xi[i] = x0 + i * h;
        }

        //Create array of y values
        Yi = new double[n + 1]; //y0...y(Xi)

        //sample.Exact series
        exactSeries= new XYChart.Series <Number, Number>();
        exactSeries.getData().clear();

        //Mark series as "sample.Exact"
        exactSeries.setName("sample.Exact");

        //end of constructor
    }
```

2. Euler class



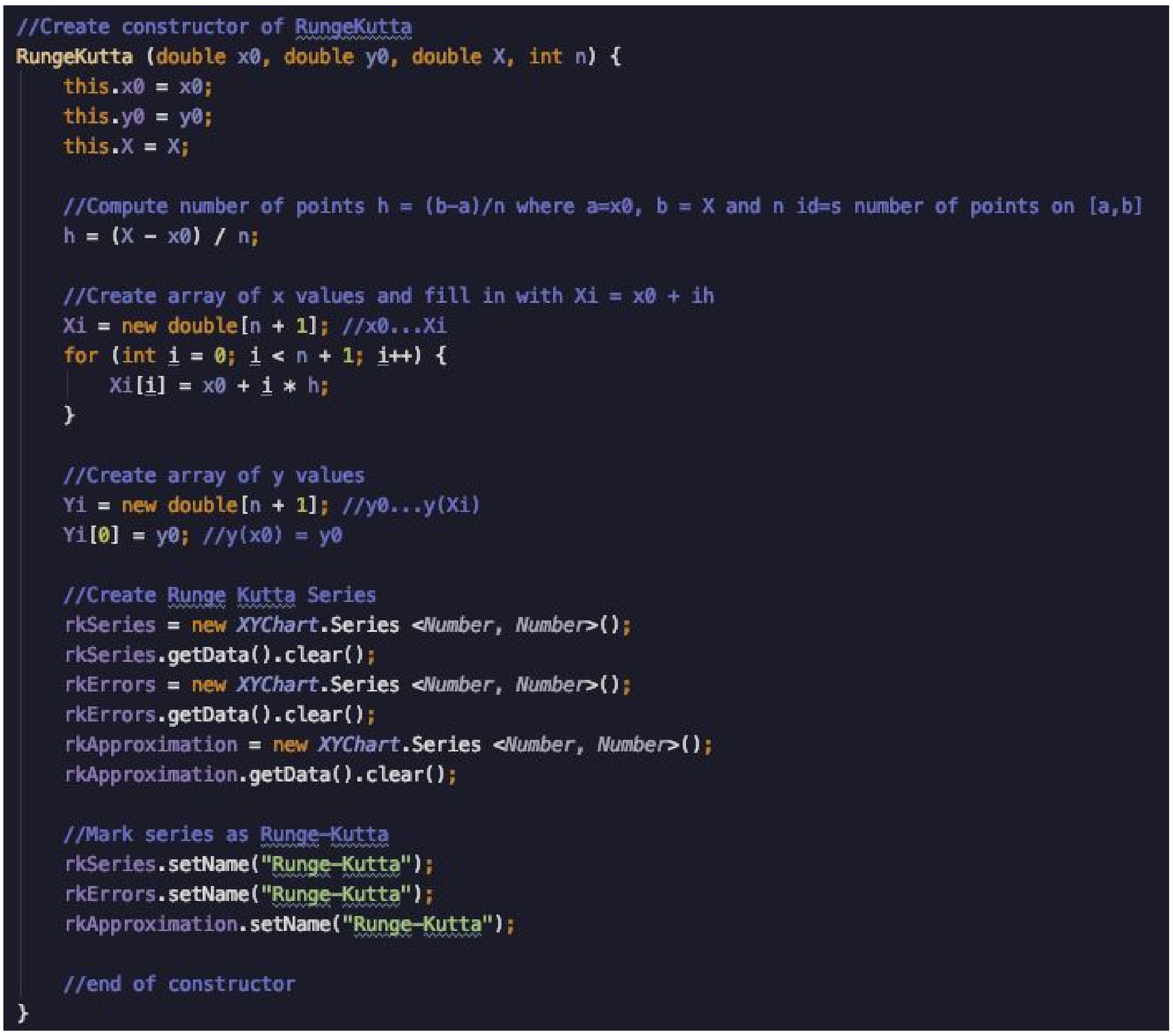
```
public class Euler {
    private double x0, y0, X, h;
    private double [] Xi;
    private double [] Yi;

    //XYChart<X, Y> is responsible for drawing the two axes and the plot content.
    //It contains a list of all content in the plot and implementations of XYChart
    //can add nodes to this list that need to be rendered.
    //XYChart.Series<X,Y> == A named series of data items
    private XYChart.Series <Number, Number> eulerSeries;
    private XYChart.Series <Number, Number> eulerApproximation;
    private XYChart.Series <Number, Number> eulerErrors;

    //Create constructor of sample.Euler method
    Euler(double x0, double y0, double X, int n){
        this.x0 = x0;
        this.y0 = y0;
        this.X = X;

        //Compute number of points h = (b-a)/n where a=x0, b = X and n id=s number of points on [a,b]
        h = (X - x0) / n;

        //Create array of x values and fill in with Xi = x0 + ih
        Xi = new double[n + 1]; //x0...Xi
        for (int i = 0; i < n + 1; ++i) {
            Xi[i] = x0 + i * h;
        }

        //Create array of y values
        Yi = new double[n + 1]; //y0...y(Xi)
        Yi[0] = y0; //y(x0) = y0

        //sample.Euler Series
        eulerSeries = new XYChart.Series <Number, Number>();
        //getData() == Gets the value of the property data.
        eulerSeries.getData().clear();

        eulerApproximation = new XYChart.Series <Number, Number>();
        eulerApproximation.getData().clear();

        eulerErrors = new XYChart.Series <Number, Number>();
        eulerErrors.getData().clear();

        //Mark series as "sample.Euler"
        eulerSeries.setName("sample.Euler");
        eulerApproximation.setName("sample.Euler");
        eulerErrors.setName("sample.Euler");

        //end of constructor
    }
```

3. ImprovedEuler class



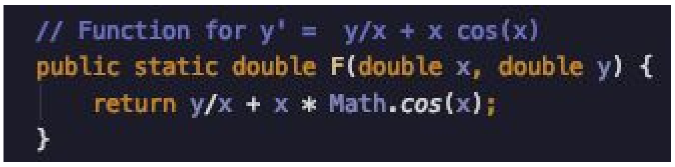
```java
public class ImprovedEuler {
        private double x0, y0, X, h;
        private double [] Xi;
        private double [] Yi;
        private XYChart.Series <Number, Number> improvedSeries; //method chart
        private XYChart.Series <Number, Number> improvedErrors; //LTE
        private XYChart.Series <Number, Number> improvedApproximation; //GTE series

        // Create constructor of improved Euler method
        ImprovedEuler (double x0, double y0, double X, int n) {
            this.x0 = x0;
            this.y0 = y0;
            this.X = X;

            //Compute number of points h = (b-a)/n where a=x0, b = X and n id=s number of points on [a,b]
            h = (X - x0) / n;

            //Create array of x values and fill in with Xi = x0 + ih
            Xi = new double[n + 1]; //x0...Xi
            for (int i = 0; i < n + 1; i++) {
                Xi[i] = x0 + i * h;
            }

            //Create array of y values
            Yi = new double[n + 1]; //y0...y(Xi)
            Yi[0] = y0; //y(x0) = y0

            //Create Improved Euler Series
            improvedSeries = new XYChart.Series <Number, Number>();
            improvedSeries.getData().clear();
            improvedErrors = new XYChart.Series <Number, Number>();
            improvedErrors.getData().clear();
            improvedApproximation = new XYChart.Series <Number, Number>();
            improvedApproximation.getData().clear();

            //Mark series as "Improved Euler"
            improvedSeries.setName("Improved Euler");
            improvedErrors.setName("Improved Euler");
            improvedApproximation.setName("Improved Euler");

            //end of constructor
        }
```

4. RungeKutta class



```java
//Create constructor of RungeKutta
RungeKutta (double x0, double y0, double X, int n) {
    this.x0 = x0;
    this.y0 = y0;
    this.X = X;

    //Compute number of points h = (b-a)/n where a=x0, b = X and n id=s number of points on [a,b]
    h = (X - x0) / n;

    //Create array of x values and fill in with Xi = x0 + ih
    Xi = new double[n + 1]; //x0...Xi
    for (int i = 0; i < n + 1; i++) {
        Xi[i] = x0 + i * h;
    }

    //Create array of y values
    Yi = new double[n + 1]; //y0...y(Xi)
    Yi[0] = y0; //y(x0) = y0

    //Create Runge Kutta Series
    rkSeries = new XYChart.Series <Number, Number>();
    rkSeries.getData().clear();
    rkErrors = new XYChart.Series <Number, Number>();
    rkErrors.getData().clear();
    rkApproximation = new XYChart.Series <Number, Number>();
    rkApproximation.getData().clear();

    //Mark series as Runge-Kutta
    rkSeries.setName("Runge-Kutta");
    rkErrors.setName("Runge-Kutta");
    rkApproximation.setName("Runge-Kutta");

    //end of constructor
}
```

About ***Main.java*** class:

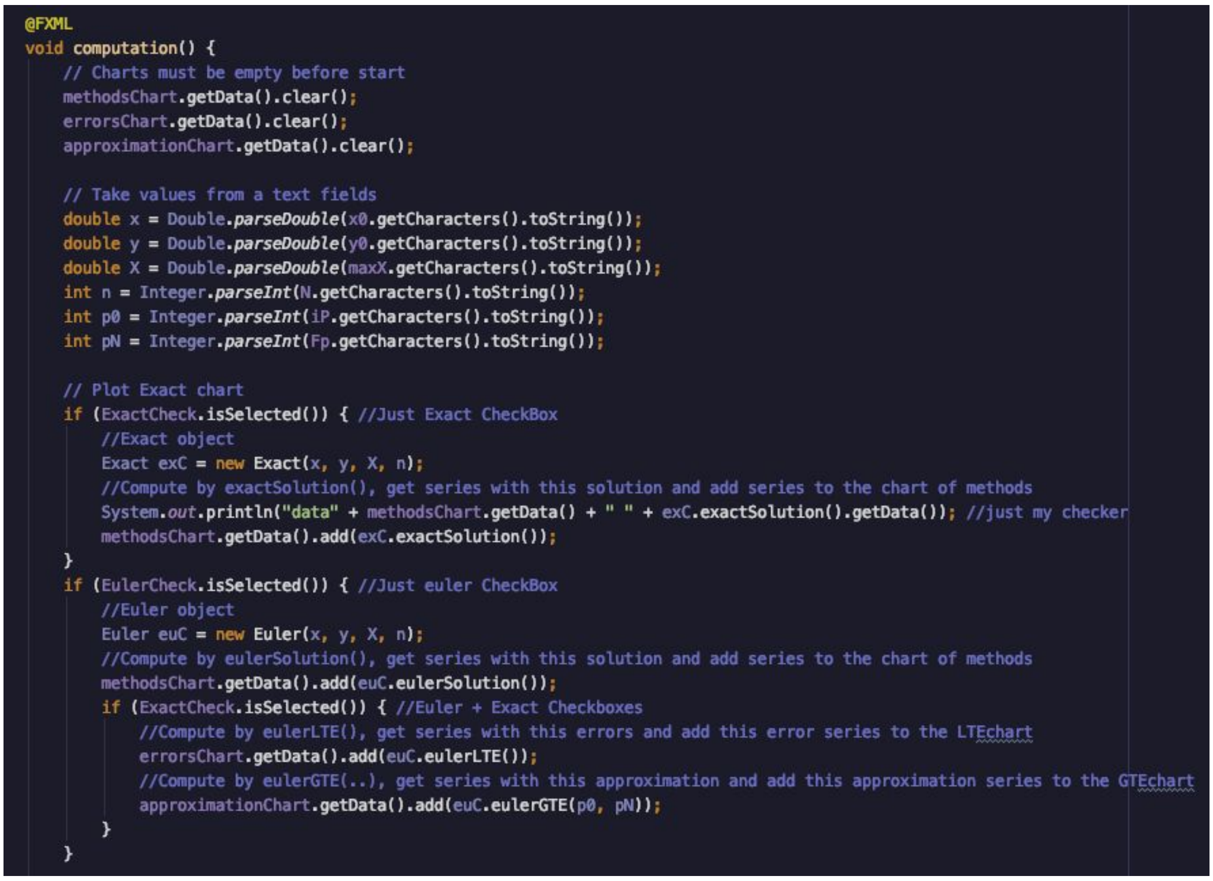


```java
// Function for y' =  y/x + x cos(x)
public static double F(double x, double y) {
    return y/x + x * Math.cos(x);
}
```

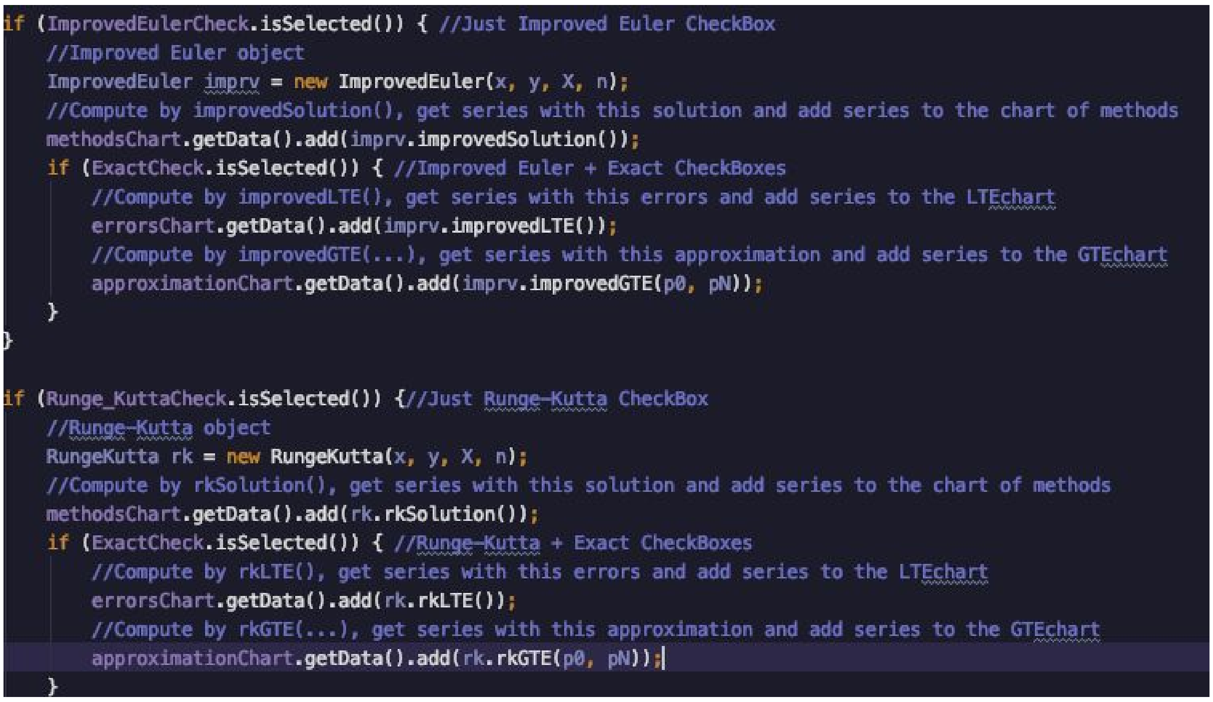
We use function ***F*** from ***Main*** to compute the derivative.

2. Process description

1. Run ***Main.java***
2. In the main window fill ***the textfields*** with corresponding values
3. Choose ***checkboxes*** that you need to see plots
4. Press the “Start” button
   This button will call the function ***computation()*** from the class ***Controller***
   All ***computation()*** function properties are shown in code ***//comments***



```java
@FXML
void computation() {
    // Charts must be empty before start
    methodsChart.getData().clear();
    errorsChart.getData().clear();
    approximationChart.getData().clear();

    // Take values from a text fields
    double x = Double.parseDouble(x0.getCharacters().toString());
    double y = Double.parseDouble(y0.getCharacters().toString());
    double X = Double.parseDouble(maxX.getCharacters().toString());
    int n = Integer.parseInt(N.getCharacters().toString());
    int p0 = Integer.parseInt(iP.getCharacters().toString());
    int pN = Integer.parseInt(Fp.getCharacters().toString());

    // Plot Exact chart
    if (ExactCheck.isSelected()) { //Just Exact CheckBox
        //Exact object
        Exact exC = new Exact(x, y, X, n);
        //Compute by exactSolution(), get series with this solution and add series to the chart of methods
        System.out.println("data" + methodsChart.getData() + " " + exC.exactSolution().getData()); //just my checker
        methodsChart.getData().add(exC.exactSolution());
    }
    if (EulerCheck.isSelected()) { //Just euler CheckBox
        //Euler object
        Euler euC = new Euler(x, y, X, n);
        //Compute by eulerSolution(), get series with this solution and add series to the chart of methods
        methodsChart.getData().add(euC.eulerSolution());
        if (ExactCheck.isSelected()) { //Euler + Exact Checkboxes
            //Compute by eulerLTE(), get series with this errors and add this error series to the LTEchart
            errorsChart.getData().add(euC.eulerLTE());
            //Compute by eulerGTE(..), get series with this approximation and add this approximation series to the GTEchart
            approximationChart.getData().add(euC.eulerGTE(p0, pN));
        }
    }
```



```java
if (ImprovedEulerCheck.isSelected()) { //Just Improved Euler CheckBox
    //Improved Euler object
    ImprovedEuler imprv = new ImprovedEuler(x, y, X, n);
    //Compute by improvedSolution(), get series with this solution and add series to the chart of methods
    methodsChart.getData().add(imprv.improvedSolution());
    if (ExactCheck.isSelected()) { //Improved Euler + Exact CheckBoxes
        //Compute by improvedLTE(), get series with this errors and add series to the LTEchart
        errorsChart.getData().add(imprv.improvedLTE());
        //Compute by improvedGTE(...), get series with this approximation and add series to the GTEchart
        approximationChart.getData().add(imprv.improvedGTE(p0, pN));
    }
}

if (Runge_KuttaCheck.isSelected()) {//Just Runge-Kutta CheckBox
    //Runge-Kutta object
    RungeKutta rk = new RungeKutta(x, y, X, n);
    //Compute by rkSolution(), get series with this solution and add series to the chart of methods
    methodsChart.getData().add(rk.rkSolution());
    if (ExactCheck.isSelected()) { //Runge-Kutta + Exact CheckBoxes
        //Compute by rkLTE(), get series with this errors and add series to the LTEchart
        errorsChart.getData().add(rk.rkLTE());
        //Compute by rkGTE(...), get series with this approximation and add series to the GTEchart
        approximationChart.getData().add(rk.rkGTE(p0, pN));
    }
```

3. Exact solution

The initial value problems is given: $y' = y/x + x\cos x,\ y(\pi) = 1$
Rewrite: $y' - y/x = x\cos x$
It is a first-order, non-homogeneous linear ordinary differential equation.
Solve the *complementary* equation:

$$y'(comp) - y(comp)/x = 0$$
$$dy/y(comp) = dx/x$$
$$y(comp) = Cx$$

Find the solution of the initial equation $y = C(x)y(comp)$:

$$[C(x)y(comp)]' - C(x)y(comp)=x\cos x$$
$$C'(x)x + C(x) - [C(x)x]/x = x\cos x$$
$$C'(x)x + C(x) - C(x) = x\cos x$$
$$C'(x)x = x\cos x$$
$$C'(x) = \cos x$$
$$C(x) = \sin x + C0$$

So, we get:

$$y = (\sin x + C0)x$$
$$y = x\sin x + C0x$$

Rewrite for initial values y0 and x0:

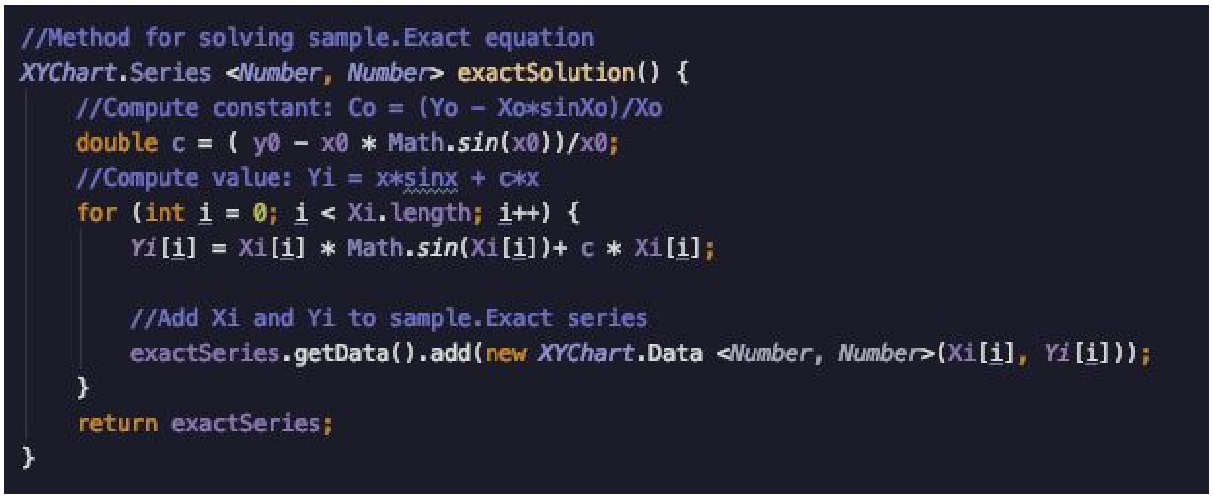
$$y0 = x0\sin(x0) + C0(x0)$$
$$C0 = [y0 - x0\sin(x0)] / x0$$
$$y = x\sin x + [(y0-x0\sin(x0)/x0]x$$

For our IVP exact solution is:

$$y = x\sin x + (1/\pi)x$$

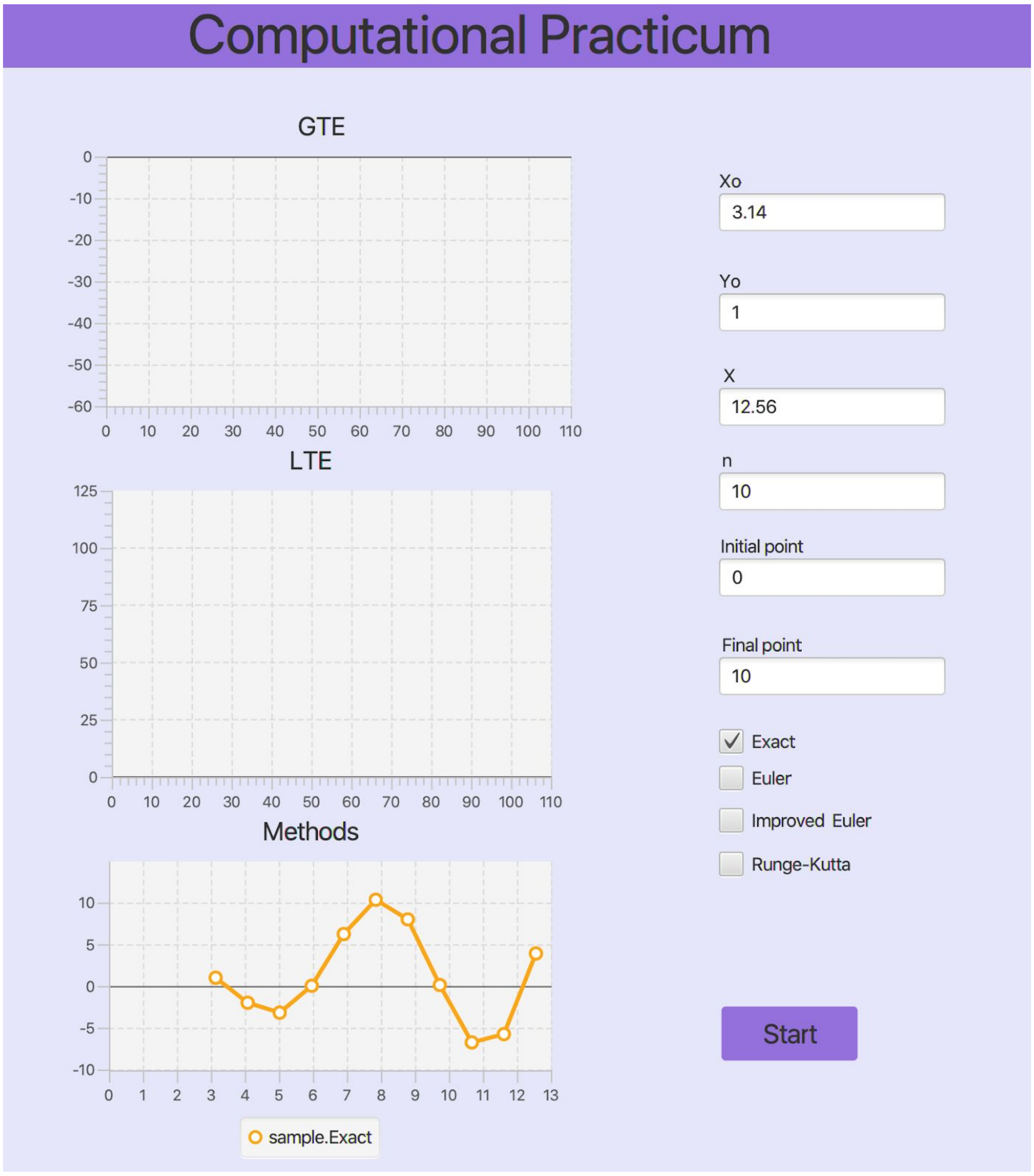
4. Implementation of Exact solution

Function ***exactSolution()*** from class ***Exact***:
All the properties of ***exactSolution()*** function in ***//comments***



```
//Method for solving sample.Exact equation
XYChart.Series <Number, Number> exactSolution() {
    //Compute constant: Co = (Yo - Xo*sinXo)/Xo
    double c = ( y0 - x0 * Math.sin(x0))/x0;
    //Compute value: Yi = x*sinx + c*x
    for (int i = 0; i < Xi.length; i++) {
        Yi[i] = Xi[i] * Math.sin(Xi[i])+ c * Xi[i];

        //Add Xi and Yi to sample.Exact series
        exactSeries.getData().add(new XYChart.Data <Number, Number>(Xi[i], Yi[i]));
    }
    return exactSeries;
}
```

The plot of Exact method:



Computational Practicum

Xo: 3.14

Yo: 1

X: 12.56

n: 10

Initial point: 0

Final point: 10

[✓] Exact

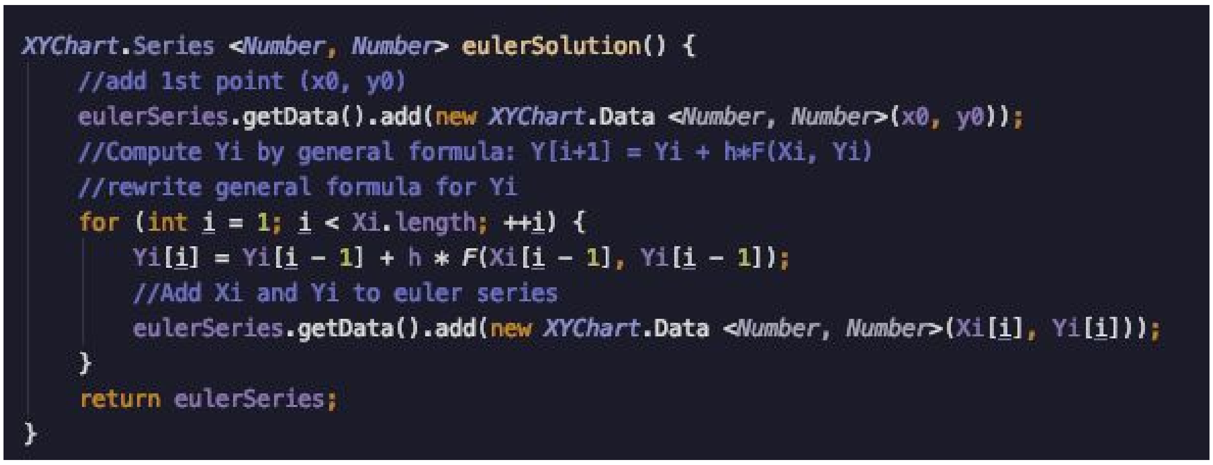
[ ] Euler

[ ] Improved Euler

[ ] Runge-Kutta

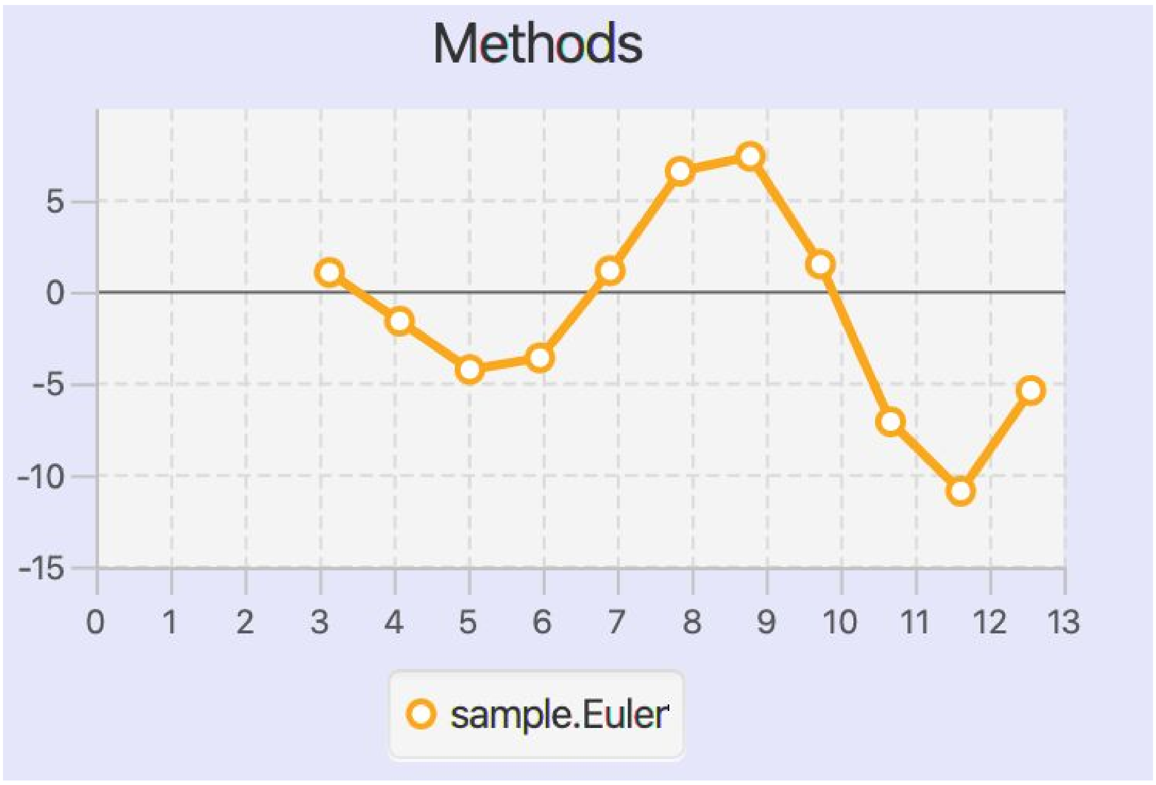
5. Implementation of the Euler method, LTE&GTE

Class ***Euler*** contains the implementation of the Euler method by function ***eulerSolution()***
All the properties of ***eulerSolution()*** function in ***//comments***



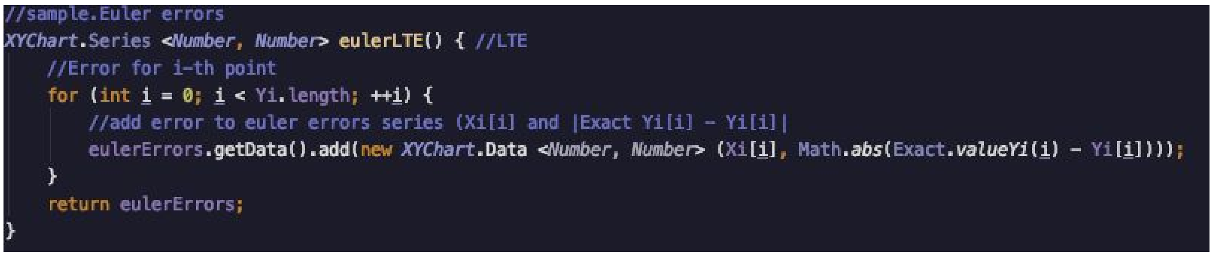
```
XYChart.Series <Number, Number> eulerSolution() {
    //add 1st point (x0, y0)
    eulerSeries.getData().add(new XYChart.Data <Number, Number>(x0, y0));
    //Compute Yi by general formula: Y[i+1] = Yi + h*F(Xi, Yi)
    //rewrite general formula for Yi
    for (int i = 1; i < Xi.length; ++i) {
        Yi[i] = Yi[i - 1] + h * F(Xi[i - 1], Yi[i - 1]);
        //Add Xi and Yi to euler series
        eulerSeries.getData().add(new XYChart.Data <Number, Number>(Xi[i], Yi[i]));
    }
    return eulerSeries;
}
```

The plot of the Euler method:

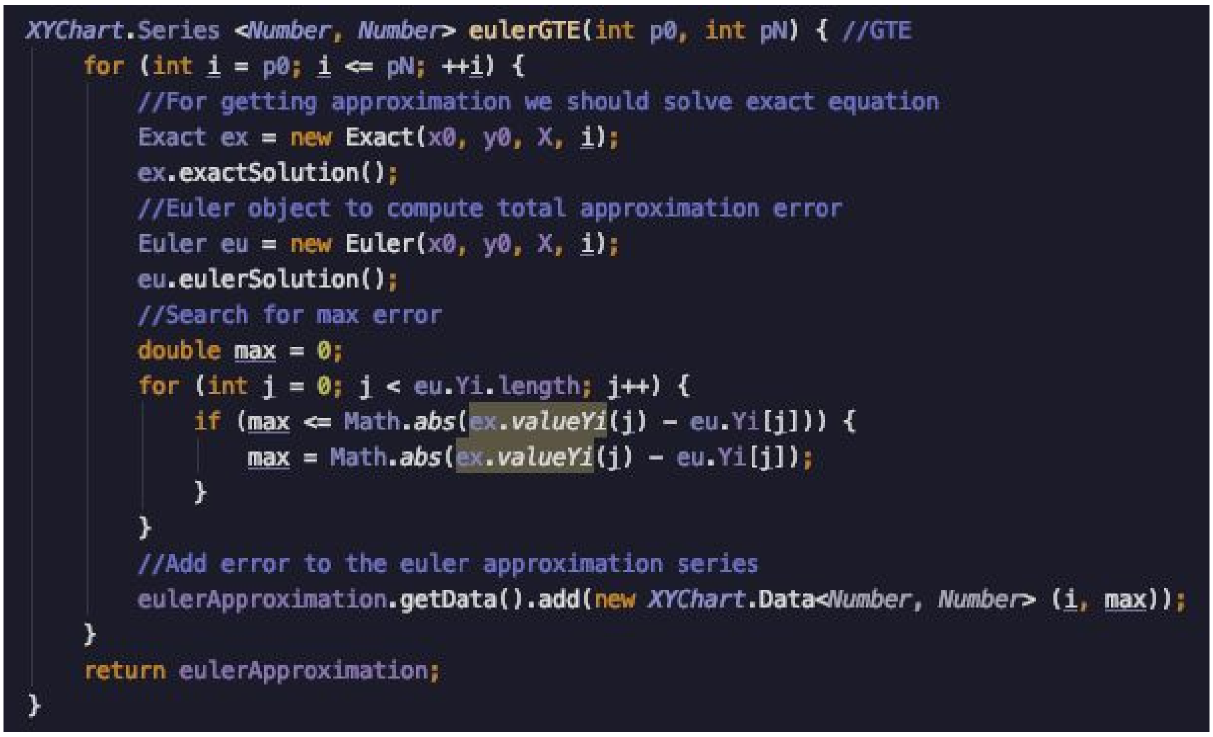


LTE&GTE:

Class ***Euler*** contains the functions ***eulerLTE()*** that compute Local Truncation Error and ***eulerGTE()*** that compute Global Truncation Error for Euler method. All the properties of ***eulerLTE()*** and ***eulerGTE()*** function in ***//comments***

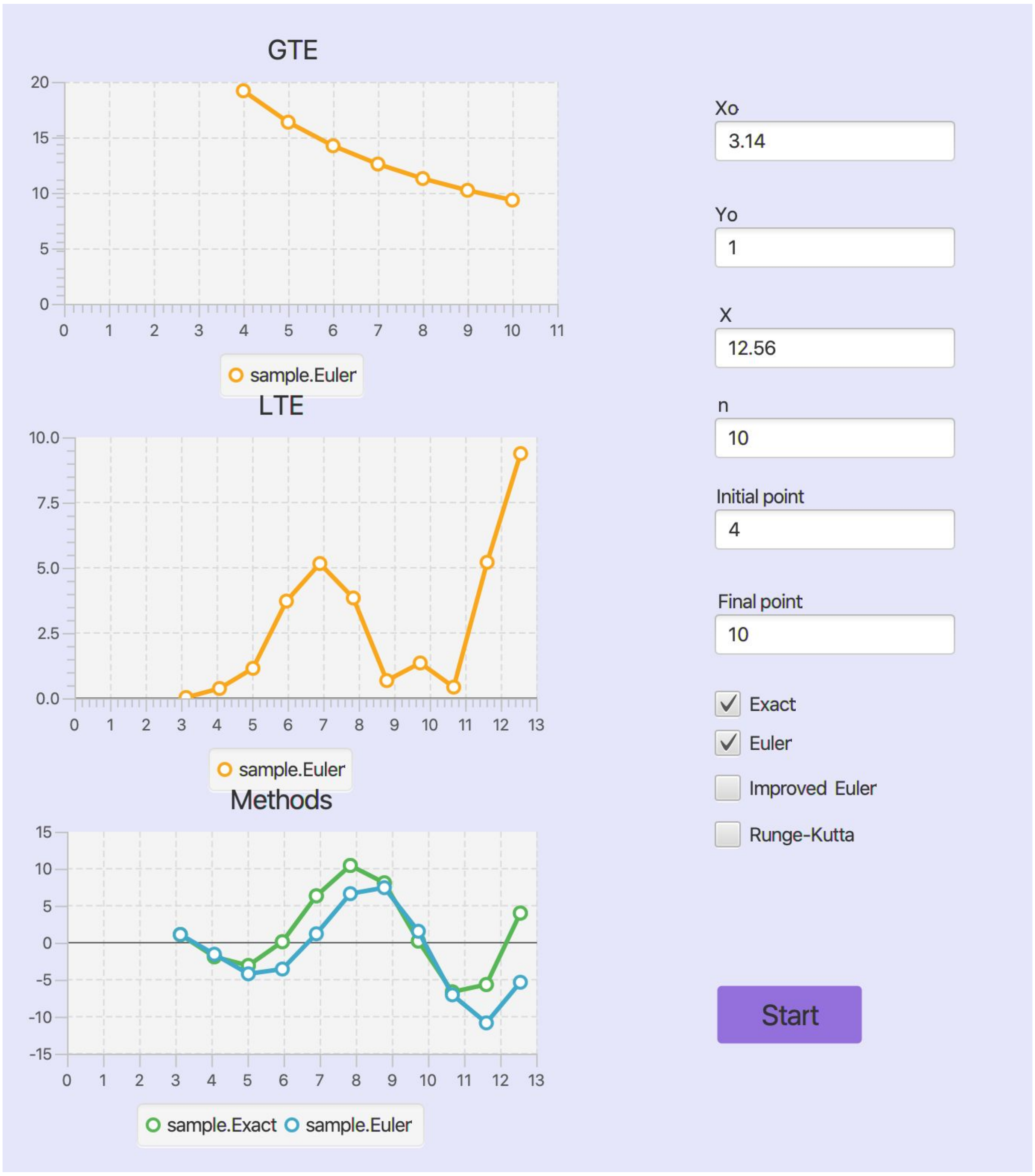


```java
//sample.Euler errors
XYChart.Series <Number, Number> eulerLTE() { //LTE
    //Error for i-th point
    for (int i = 0; i < Yi.length; ++i) {
        //add error to euler errors series (Xi[i] and |Exact Yi[i] - Yi[i]|
        eulerErrors.getData().add(new XYChart.Data <Number, Number> (Xi[i], Math.abs(Exact.valueYi(i) - Yi[i])));
    }
    return eulerErrors;
}
```



```java
XYChart.Series <Number, Number> eulerGTE(int p0, int pN) { //GTE
    for (int i = p0; i <= pN; ++i) {
        //For getting approximation we should solve exact equation
        Exact ex = new Exact(x0, y0, X, i);
        ex.exactSolution();
        //Euler object to compute total approximation error
        Euler eu = new Euler(x0, y0, X, i);
        eu.eulerSolution();
        //Search for max error
        double max = 0;
        for (int j = 0; j < eu.Yi.length; j++) {
            if (max <= Math.abs(ex.valueYi(j) - eu.Yi[j])) {
                max = Math.abs(ex.valueYi(j) - eu.Yi[j]);
            }
        }
        //Add error to the euler approximation series
        eulerApproximation.getData().add(new XYChart.Data<Number, Number> (i, max));
    }
    return eulerApproximation;
}
```

```
The plots of the Euler LTE & GTE:
```



Xo: 3.14

Yo: 1

X: 12.56

n: 10

Initial point: 4

Final point: 10

[✓ Exact]

[✓ Euler]

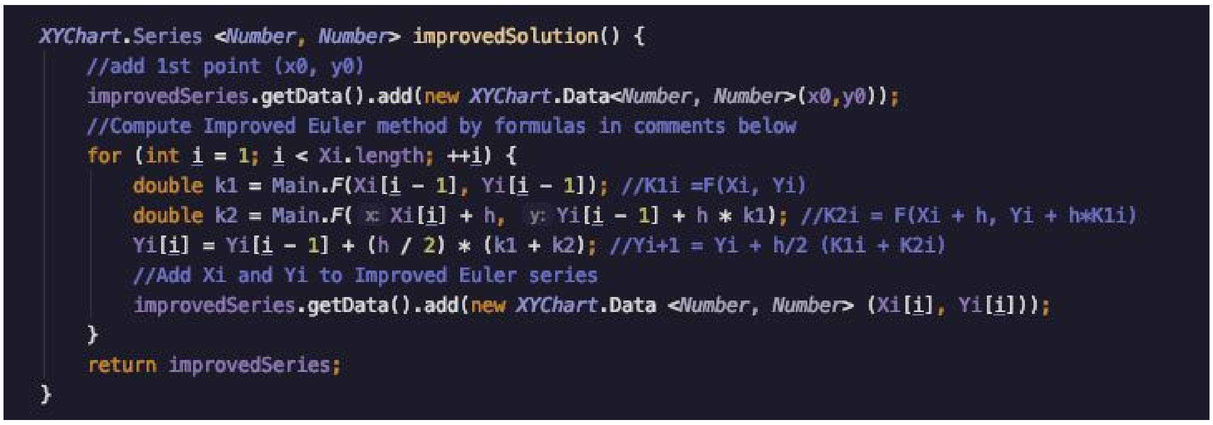
Improved Euler

Runge-Kutta

6. Implementation of the Improved Euler method, LTE&GTE

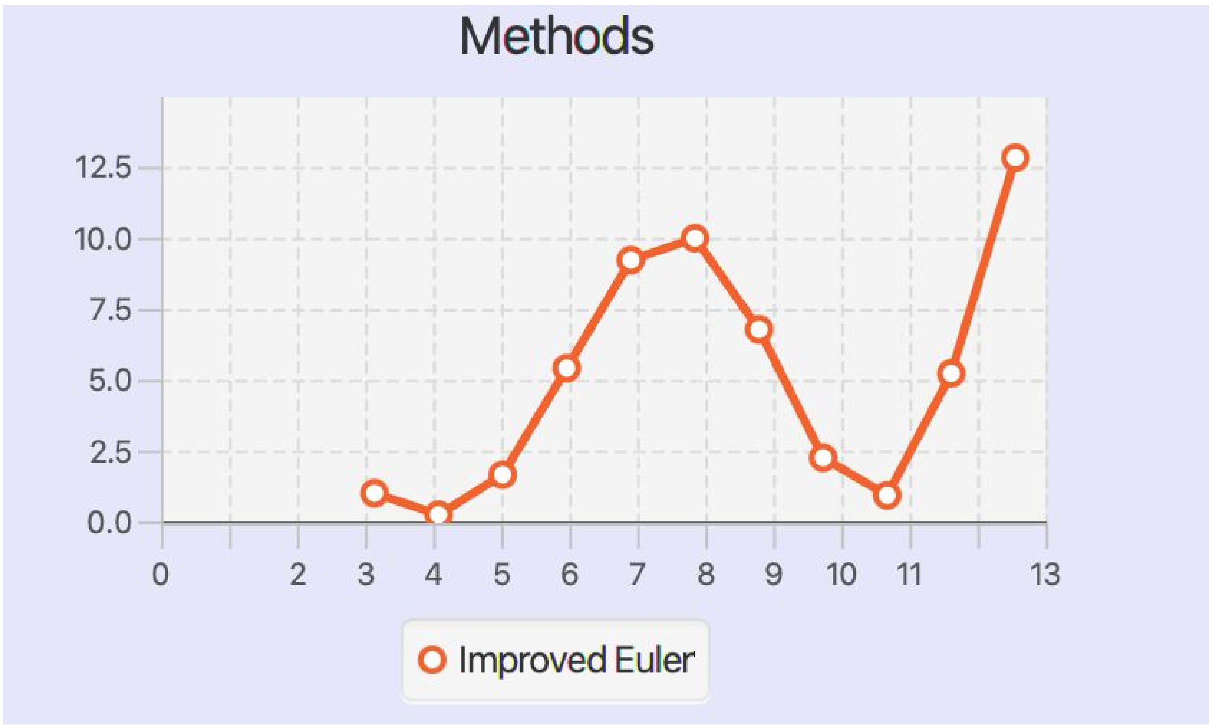
Class **Improved*Euler*** contains the implementation of the Improved Euler method by function ***improvedSolution()***

All the properties of ***improvedSolution()*** function in ***//comments***



```java
XYChart.Series <Number, Number> improvedSolution() {
    //add 1st point (x0, y0)
    improvedSeries.getData().add(new XYChart.Data<Number, Number>(x0,y0));
    //Compute Improved Euler method by formulas in comments below
    for (int i = 1; i < Xi.length; ++i) {
        double k1 = Main.F(Xi[i - 1], Yi[i - 1]); //K1i =F(Xi, Yi)
        double k2 = Main.F( x: Xi[i] + h,  y: Yi[i - 1] + h * k1); //K2i = F(Xi + h, Yi + h*K1i)
        Yi[i] = Yi[i - 1] + (h / 2) * (k1 + k2); //Yi+1 = Yi + h/2 (K1i + K2i)
        //Add Xi and Yi to Improved Euler series
        improvedSeries.getData().add(new XYChart.Data <Number, Number> (Xi[i], Yi[i]));
    }
    return improvedSeries;
}
```

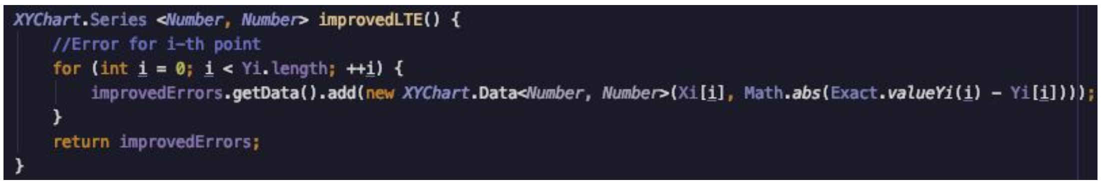
The plot of the Improved Euler method:



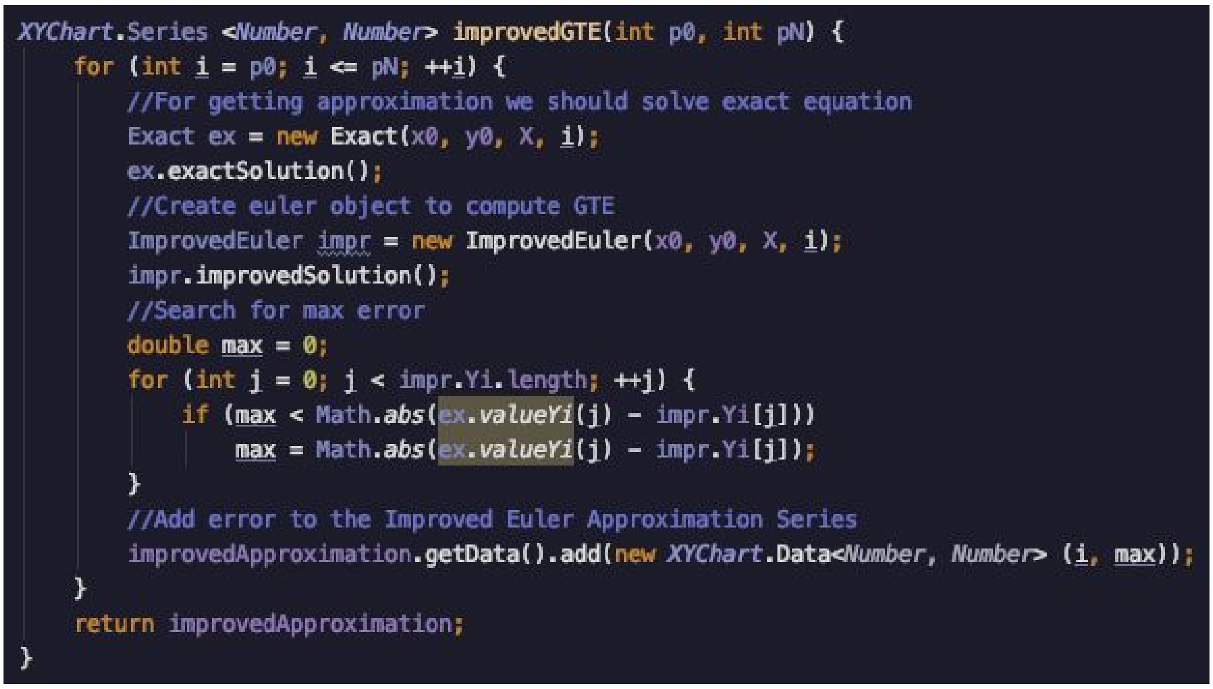
LTE&GTE:

Class ***ImprovedEuler*** contains the functions ***improvedLTE()*** that compute Local Truncation Error and ***improvedGTE()*** that compute Global Truncation Error for Improved Euler method.

All the properties of ***improvedLTE()*** and ***improvedGTE()*** function in ***//comments***

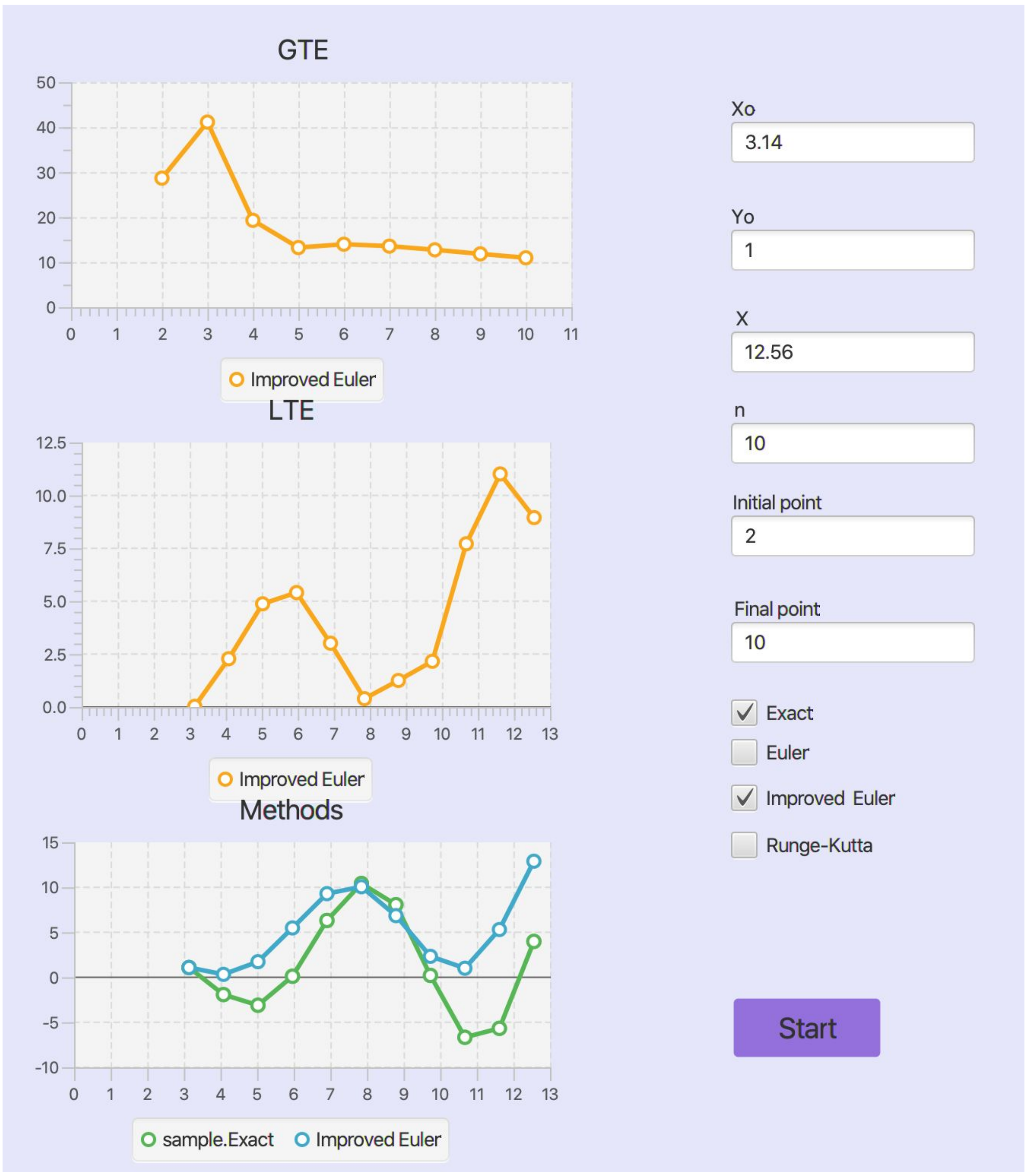


```java
XYChart.Series <Number, Number> improvedLTE() {
    //Error for i-th point
    for (int i = 0; i < Yi.length; ++i) {
        improvedErrors.getData().add(new XYChart.Data<Number, Number>(Xi[i], Math.abs(Exact.valueYi(i) - Yi[i])));
    }
    return improvedErrors;
}
```



```
XYChart.Series <Number, Number> improvedGTE(int p0, int pN) {
    for (int i = p0; i <= pN; ++i) {
        //For getting approximation we should solve exact equation
        Exact ex = new Exact(x0, y0, X, i);
        ex.exactSolution();
        //Create euler object to compute GTE
        ImprovedEuler impr = new ImprovedEuler(x0, y0, X, i);
        impr.improvedSolution();
        //Search for max error
        double max = 0;
        for (int j = 0; j < impr.Yi.length; ++j) {
            if (max < Math.abs(ex.valueYi(j) - impr.Yi[j]))
                max = Math.abs(ex.valueYi(j) - impr.Yi[j]);
        }
        //Add error to the Improved Euler Approximation Series
        improvedApproximation.getData().add(new XYChart.Data<Number, Number> (i, max));
    }
    return improvedApproximation;
}
```

```
The plots of the Improved Euler LTE & GTE:
```



Xo: 3.14

Yo: 1

X: 12.56

n: 10

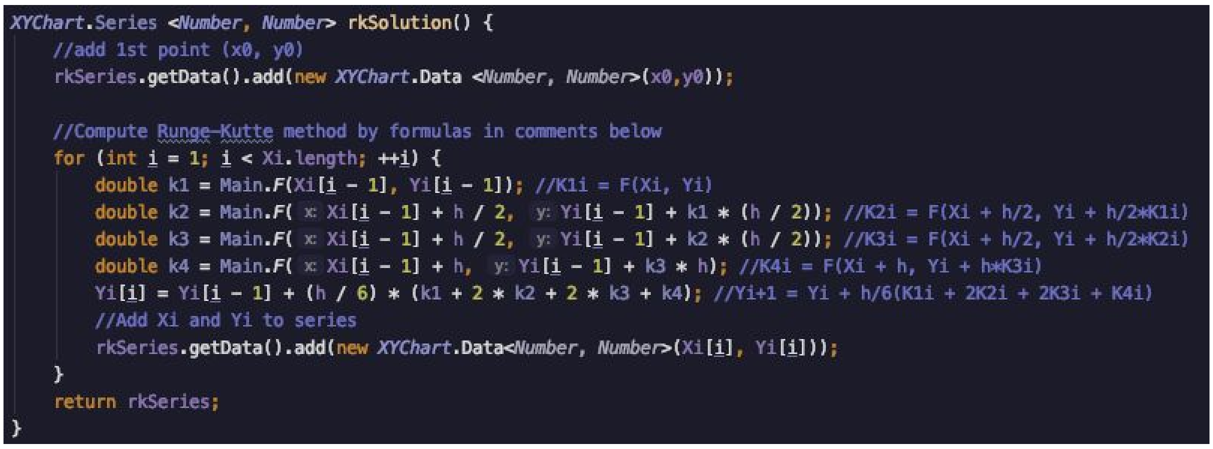
Initial point: 2

Final point: 10

- [✓] Exact
- [ ] Euler
- [✓] Improved Euler
- [ ] Runge-Kutta

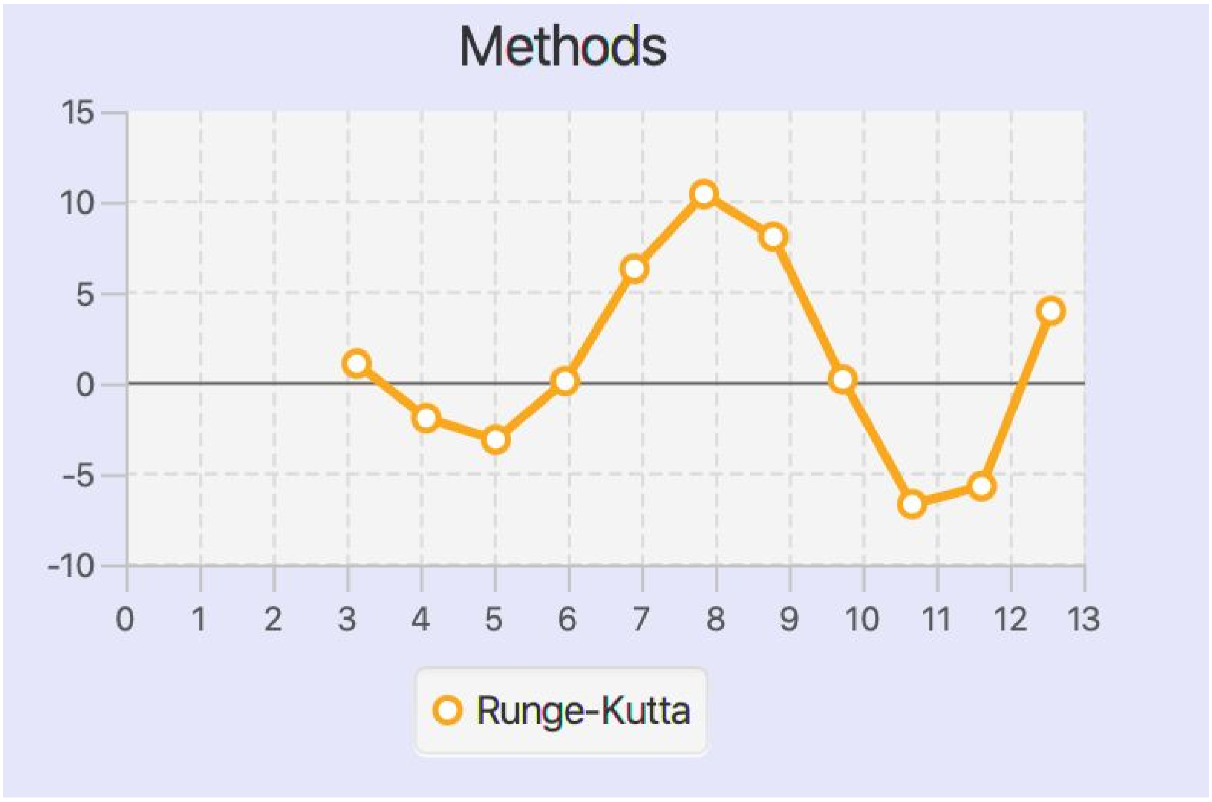
7. Implementation of the Runge-Kutta method, LTE&GTE

Class **RungeKutta*Euler*** contains the implementation of the Runge-Kutta method by function ***rkSolution()***
All the properties of ***rkSolution()*** function in ***//comments***



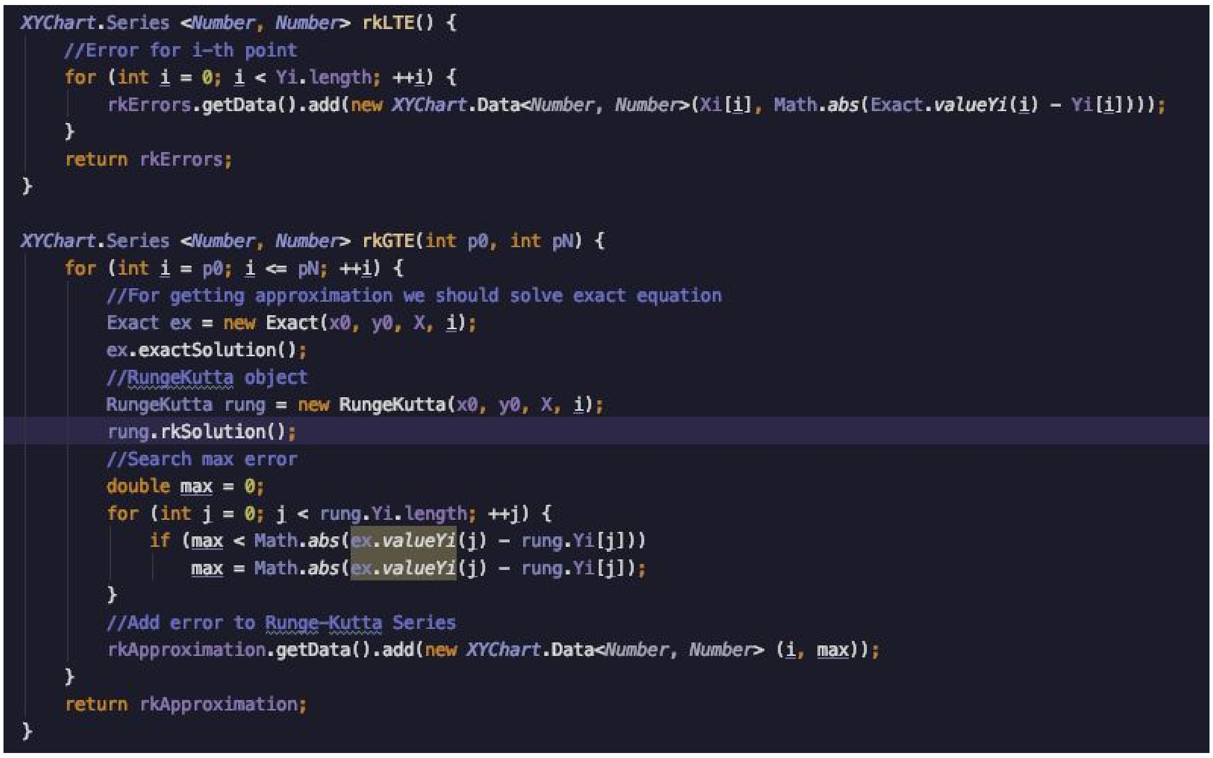
```java
XYChart.Series <Number, Number> rkSolution() {
    //add 1st point (x0, y0)
    rkSeries.getData().add(new XYChart.Data <Number, Number>(x0,y0));

    //Compute Runge-Kutte method by formulas in comments below
    for (int i = 1; i < Xi.length; ++i) {
        double k1 = Main.F(Xi[i - 1], Yi[i - 1]); //K1i = F(Xi, Yi)
        double k2 = Main.F( x: Xi[i - 1] + h / 2,  y: Yi[i - 1] + k1 * (h / 2)); //K2i = F(Xi + h/2, Yi + h/2*K1i)
        double k3 = Main.F( x: Xi[i - 1] + h / 2,  y: Yi[i - 1] + k2 * (h / 2)); //K3i = F(Xi + h/2, Yi + h/2*K2i)
        double k4 = Main.F( x: Xi[i - 1] + h,  y: Yi[i - 1] + k3 * h); //K4i = F(Xi + h, Yi + h*K3i)
        Yi[i] = Yi[i - 1] + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4); //Yi+1 = Yi + h/6(K1i + 2K2i + 2K3i + K4i)
        //Add Xi and Yi to series
        rkSeries.getData().add(new XYChart.Data<Number, Number>(Xi[i], Yi[i]));
    }
    return rkSeries;
}
```

The plot of the Runge-Kutta method:



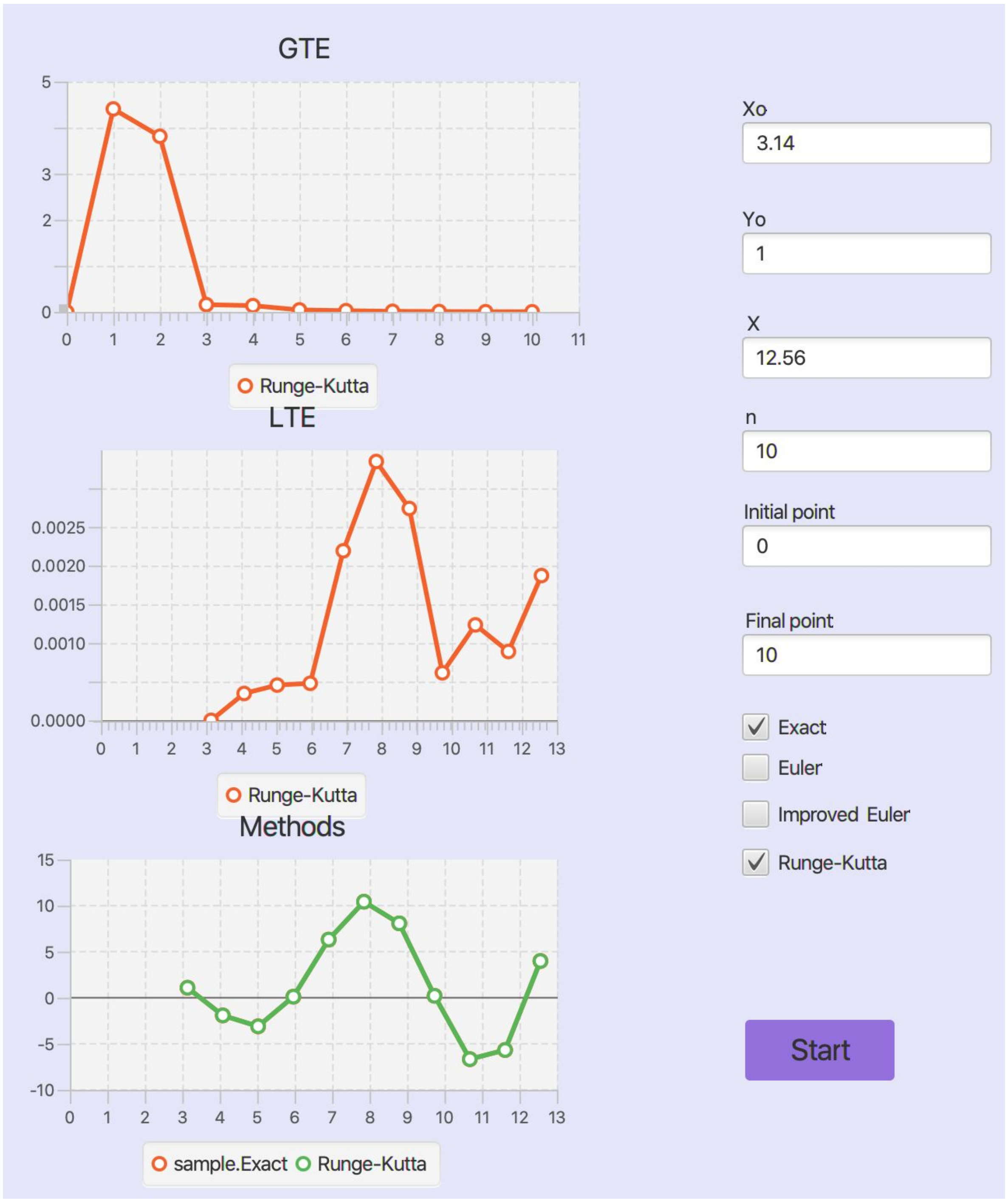
LTE&GTE:

Class ***ImprovedEuler*** contains the functions ***rkLTE()*** that compute Local Truncation Error and ***rkGTE()*** that compute Global Truncation Error for Runge-Kutta method. All the properties of ***improvedLTE()*** and ***improvedGTE()*** function in ***//comments***



```
XYChart.Series <Number, Number> rkLTE() {
    //Error for i-th point
    for (int i = 0; i < Yi.length; ++i) {
        rkErrors.getData().add(new XYChart.Data<Number, Number>(Xi[i], Math.abs(Exact.valueYi(i) - Yi[i])));
    }
    return rkErrors;
}

XYChart.Series <Number, Number> rkGTE(int p0, int pN) {
    for (int i = p0; i <= pN; ++i) {
        //For getting approximation we should solve exact equation
        Exact ex = new Exact(x0, y0, X, i);
        ex.exactSolution();
        //RungeKutta object
        RungeKutta rung = new RungeKutta(x0, y0, X, i);
        rung.rkSolution();
        //Search max error
        double max = 0;
        for (int j = 0; j < rung.Yi.length; ++j) {
            if (max < Math.abs(ex.valueYi(j) - rung.Yi[j]))
                max = Math.abs(ex.valueYi(j) - rung.Yi[j]);
        }
        //Add error to Runge-Kutta Series
        rkApproximation.getData().add(new XYChart.Data<Number, Number> (i, max));
    }
    return rkApproximation;
}
```

The plots of the Runge-Kutta LTE & GTE:



Xo
3.14

Yo
1

X
12.56

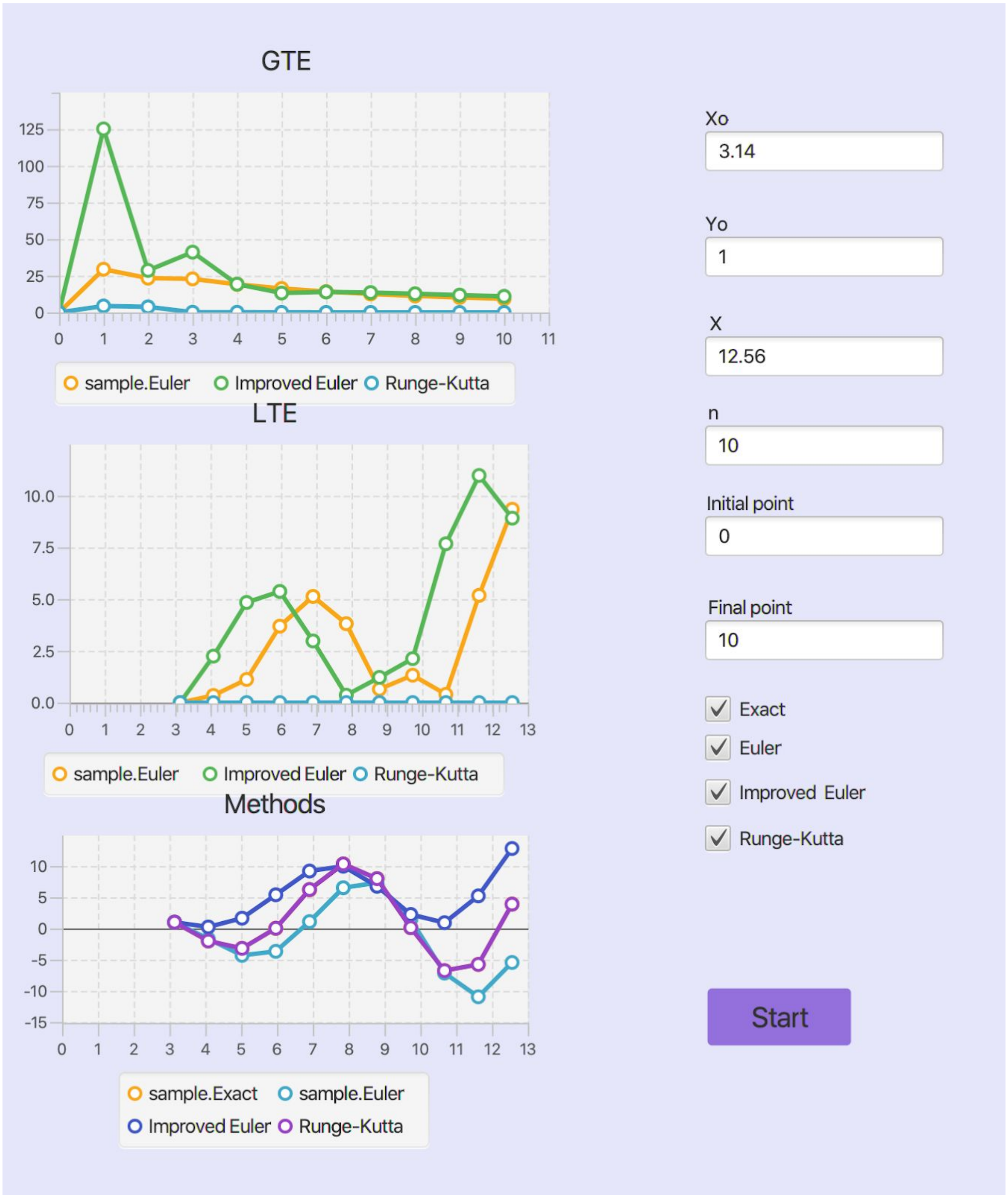
n
10

Initial point
0

Final point
10

✓ Exact
Euler
Improved Euler
✓ Runge-Kutta

8. Comparing all the plots



Xo: 3.14

Yo: 1

X: 12.56

n: 10

Initial point: 0

Final point: 10

✓ Exact

✓ Euler

✓ Improved Euler

✓ Runge-Kutta