Computational assignment

Solution:

$y' = 2\sqrt{y} + 2y$ IVP: $y(0)=1$

$\frac{dy}{dx} = 2\sqrt{y} + 2y$

$\int \frac{dy}{2\sqrt{y}+2y} = \int dx$

$\int \frac{dy}{2\sqrt{y}(1+\sqrt{y})} = x$

Замена: $t = 1+\sqrt{y}$

$dt = \frac{dy}{2\sqrt{y}}$

$dy = dt \cdot 2\sqrt{y}$

$x = \int \frac{dt \cdot \cancel{2\sqrt{y}}}{t \cdot \cancel{2\sqrt{y}}}$

$x = \ln(1+\sqrt{y}) + C$

$\ln(1+\sqrt{y}) = x - C$

$1+\sqrt{y} = e^{x-C}$

$\sqrt{y} = e^{x-C} - 1$

$\sqrt{y} = Ce^{x} - 1$

$y = (Ce^{x}-1)^2$

$C = \frac{\sqrt{y}+1}{e^{x}}$ Ограничение: $y \geq 0$

$y(0)=1$ тогда $C = \frac{2}{1} = 2$

т.е. $y = (2e^{x}-1)^2$

Проверка решения:

$y' = 2(\sqrt{y}+y)$

$y = (Ce^{x}-1)^2 = C^2e^{2x} - 2Ce^{x} + 1$

$y' = 2C^2e^{2x} - 2Ce^{x}$

$2C^2e^{2x} - 2Ce^{x} = 2((Ce^{x}-1) + C^2e^{2x} - 2Ce^{x} + 1)$

$2C^2e^{2x} - 2Ce^{x} = 2(C^2e^{2x} - Ce^{x})$

$\cancel{2C^2e^{2x}} - \cancel{2Ce^{x}} = \cancel{2C^2e^{2x}} - \cancel{2Ce^{x}}$

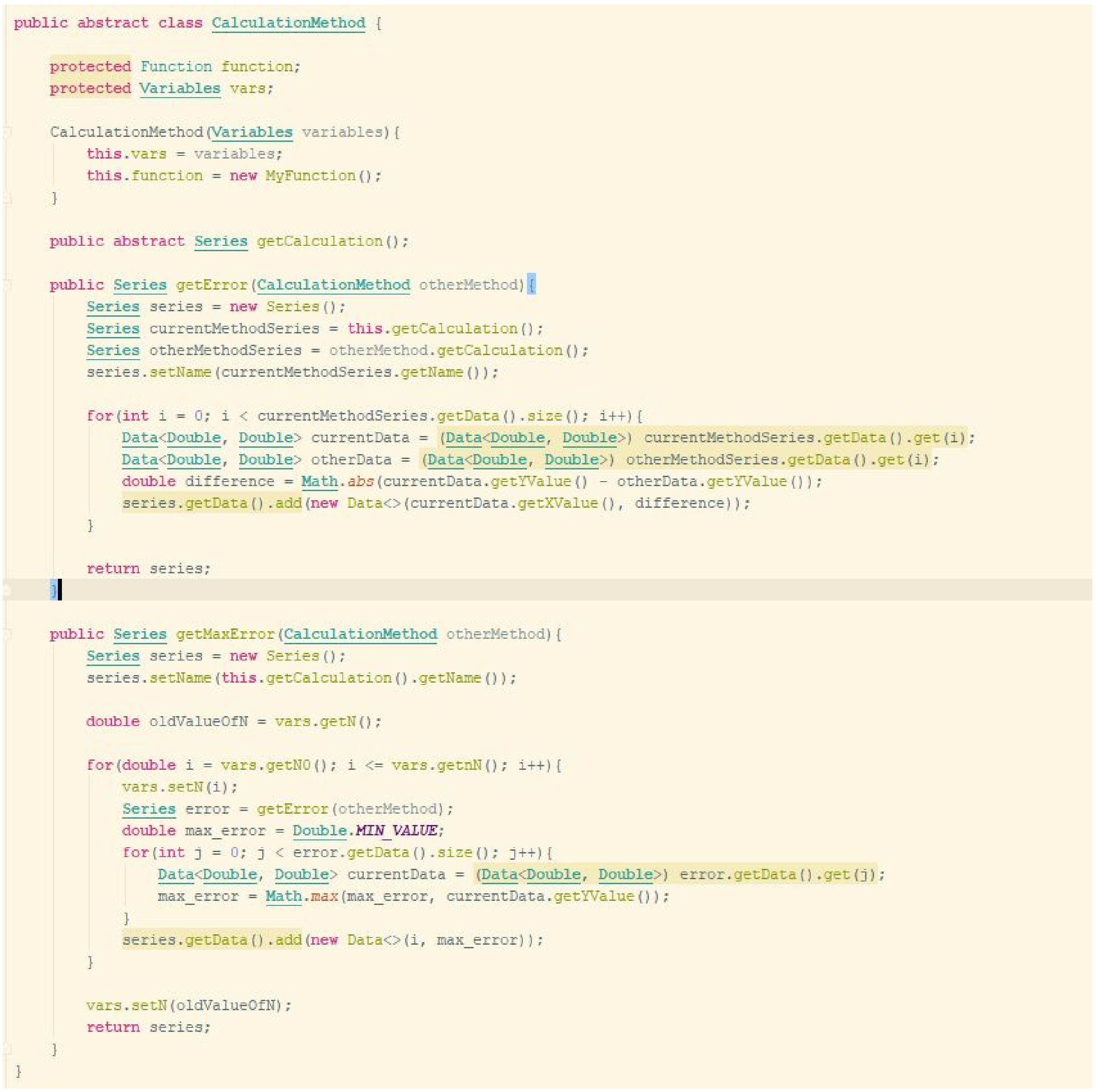
$0=0$

ч.т.д.

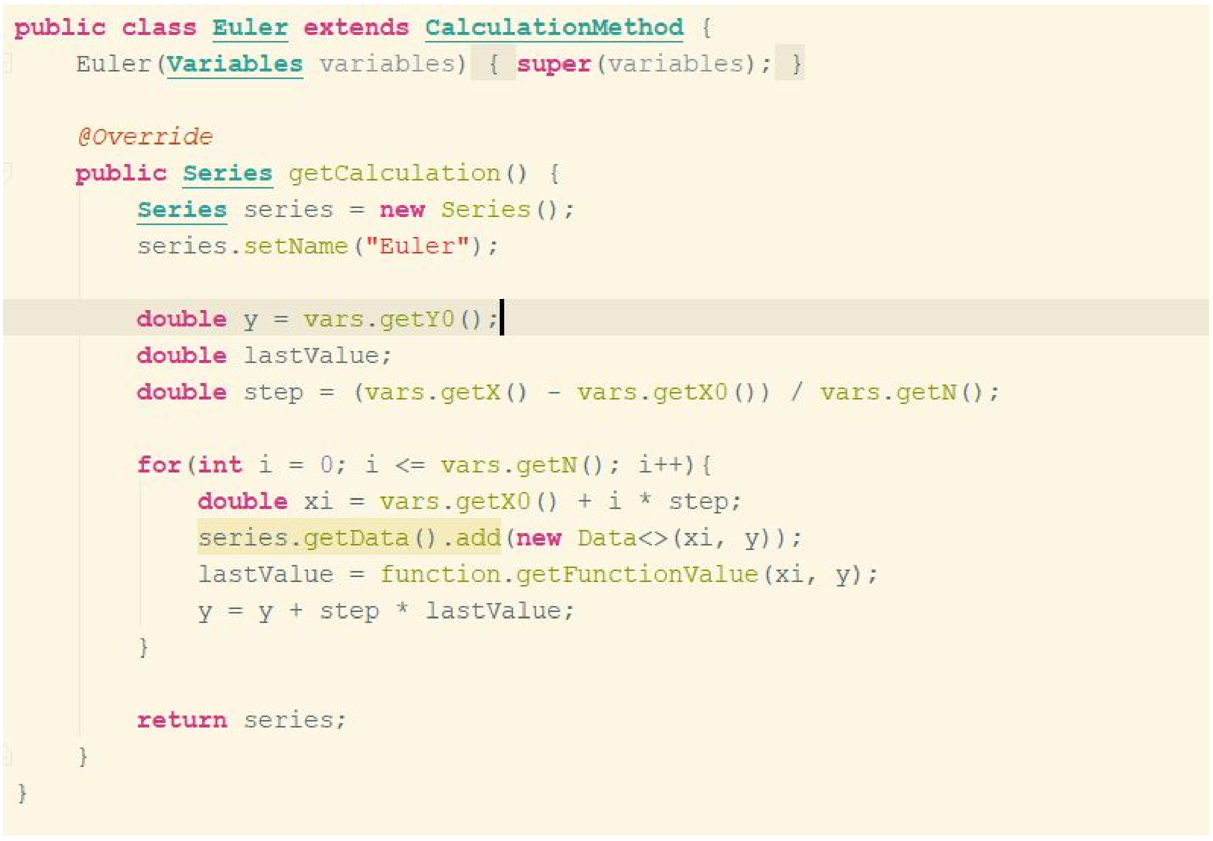
значит решение верное

Source code:

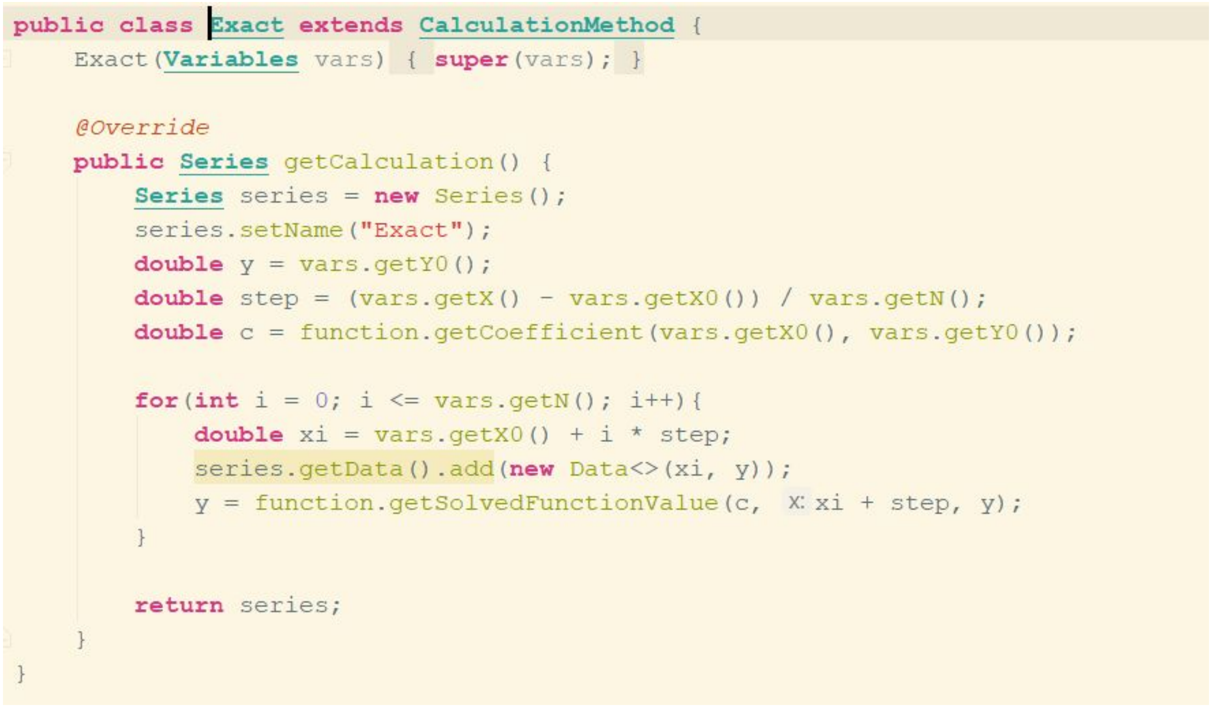
Calculation method class is base class for calculation methods (Exact, Euler etc.). It has methods getError() and getMaxError() which are the same for all calculation methods and has abstract method getCalculation(), which every calculation method should realize by itself.



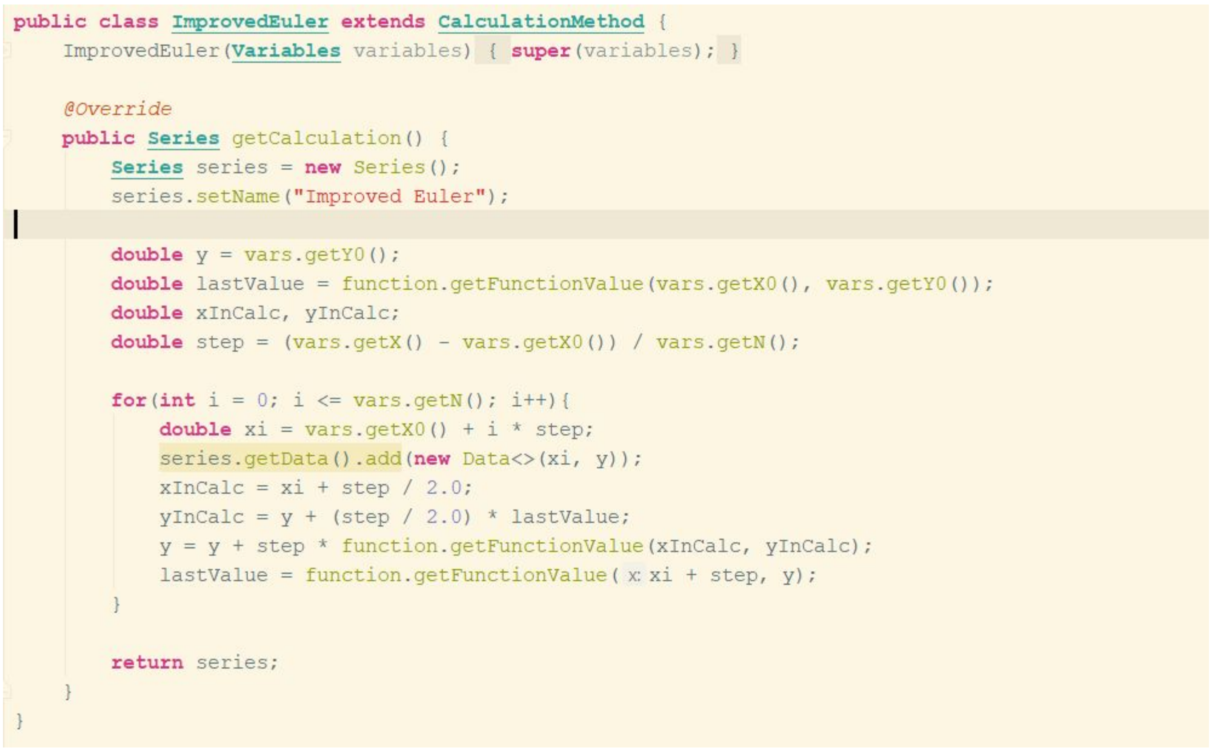
```java
public abstract class CalculationMethod {

    protected Function function;
    protected Variables vars;

    CalculationMethod(Variables variables){
        this.vars = variables;
        this.function = new MyFunction();
    }

    public abstract Series getCalculation();

    public Series getError(CalculationMethod otherMethod){
        Series series = new Series();
        Series currentMethodSeries = this.getCalculation();
        Series otherMethodSeries = otherMethod.getCalculation();
        series.setName(currentMethodSeries.getName());

        for(int i = 0; i < currentMethodSeries.getData().size(); i++){
            Data<Double, Double> currentData = (Data<Double, Double>) currentMethodSeries.getData().get(i);
            Data<Double, Double> otherData = (Data<Double, Double>) otherMethodSeries.getData().get(i);
            double difference = Math.abs(currentData.getYValue() - otherData.getYValue());
            series.getData().add(new Data<>(currentData.getXValue(), difference));
        }

        return series;
    }

    public Series getMaxError(CalculationMethod otherMethod){
        Series series = new Series();
        series.setName(this.getCalculation().getName());

        double oldValueOfN = vars.getN();

        for(double i = vars.getN0(); i <= vars.getnN(); i++){
            vars.setN(i);
            Series error = getError(otherMethod);
            double max_error = Double.MIN_VALUE;
            for(int j = 0; j < error.getData().size(); j++){
                Data<Double, Double> currentData = (Data<Double, Double>) error.getData().get(j);
                max_error = Math.max(max_error, currentData.getYValue());
            }
            series.getData().add(new Data<>(i, max_error));
        }

        vars.setN(oldValueOfN);
        return series;
    }
}
```



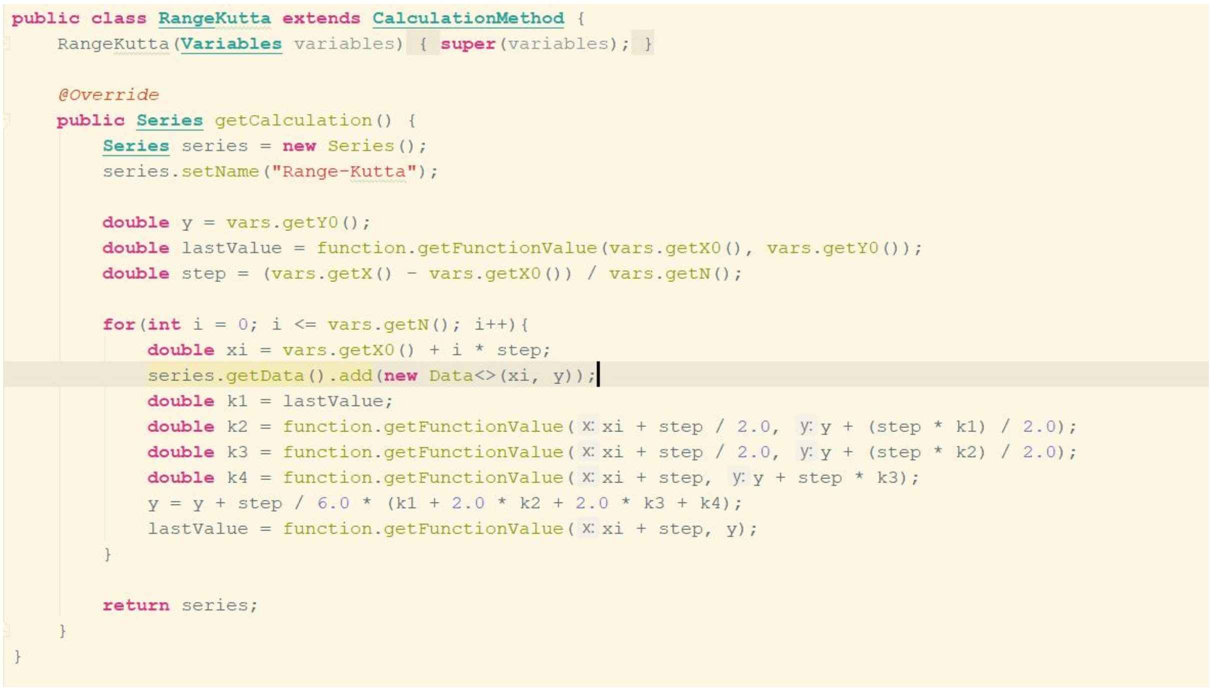
```java
public class Euler extends CalculationMethod {
    Euler(Variables variables) { super(variables); }

    @Override
    public Series getCalculation() {
        Series series = new Series();
        series.setName("Euler");

        double y = vars.getY0();
        double lastValue;
        double step = (vars.getX() - vars.getX0()) / vars.getN();

        for(int i = 0; i <= vars.getN(); i++){
            double xi = vars.getX0() + i * step;
            series.getData().add(new Data<>(xi, y));
            lastValue = function.getFunctionValue(xi, y);
            y = y + step * lastValue;
        }

        return series;
    }
}
```



```java
public class Exact extends CalculationMethod {
    Exact(Variables vars) { super(vars); }

    @Override
    public Series getCalculation() {
        Series series = new Series();
        series.setName("Exact");
        double y = vars.getY0();
        double step = (vars.getX() - vars.getX0()) / vars.getN();
        double c = function.getCoefficient(vars.getX0(), vars.getY0());

        for(int i = 0; i <= vars.getN(); i++){
            double xi = vars.getX0() + i * step;
            series.getData().add(new Data<>(xi, y));
            y = function.getSolvedFunctionValue(c,  x: xi + step, y);
        }

        return series;
    }
}
```

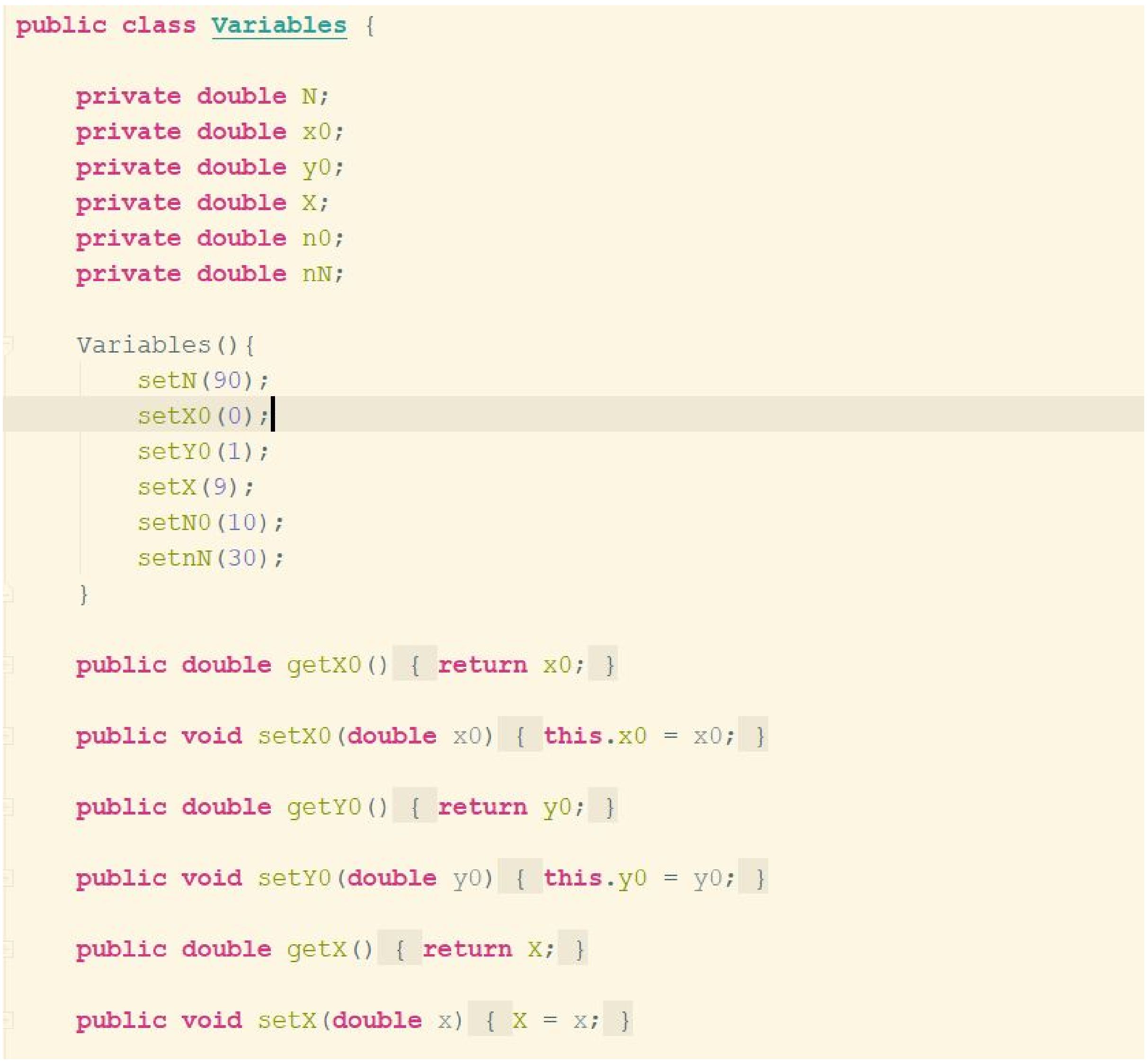


```java
public class ImprovedEuler extends CalculationMethod {
    ImprovedEuler(Variables variables) { super(variables); }

    @Override
    public Series getCalculation() {
        Series series = new Series();
        series.setName("Improved Euler");

        double y = vars.getY0();
        double lastValue = function.getFunctionValue(vars.getX0(), vars.getY0());
        double xInCalc, yInCalc;
        double step = (vars.getX() - vars.getX0()) / vars.getN();

        for(int i = 0; i <= vars.getN(); i++){
            double xi = vars.getX0() + i * step;
            series.getData().add(new Data<>(xi, y));
            xInCalc = xi + step / 2.0;
            yInCalc = y + (step / 2.0) * lastValue;
            y = y + step * function.getFunctionValue(xInCalc, yInCalc);
            lastValue = function.getFunctionValue( x: xi + step, y);
        }

        return series;
    }
}
```



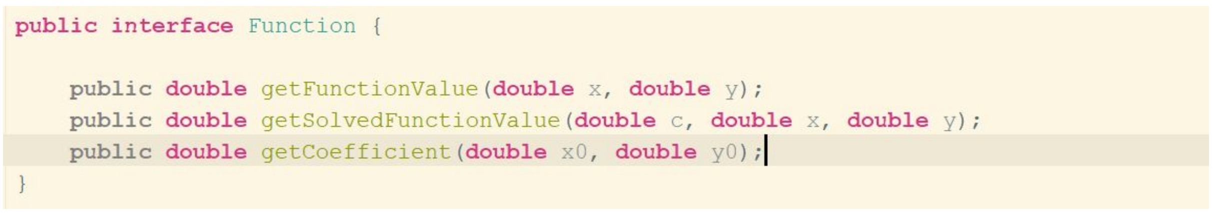
```java
public class RangeKutta extends CalculationMethod {
    RangeKutta(Variables variables) { super(variables); }

    @Override
    public Series getCalculation() {
        Series series = new Series();
        series.setName("Range-Kutta");

        double y = vars.getY0();
        double lastValue = function.getFunctionValue(vars.getX0(), vars.getY0());
        double step = (vars.getX() - vars.getX0()) / vars.getN();

        for(int i = 0; i <= vars.getN(); i++){
            double xi = vars.getX0() + i * step;
            series.getData().add(new Data<>(xi, y));
            double k1 = lastValue;
            double k2 = function.getFunctionValue( x: xi + step / 2.0,  y: y + (step * k1) / 2.0);
            double k3 = function.getFunctionValue( x: xi + step / 2.0,  y: y + (step * k2) / 2.0);
            double k4 = function.getFunctionValue( x: xi + step,  y: y + step * k3);
            y = y + step / 6.0 * (k1 + 2.0 * k2 + 2.0 * k3 + k4);
            lastValue = function.getFunctionValue( x: xi + step, y);
        }

        return series;
    }
}
```

This is class for storing, changing and getting value of variables.



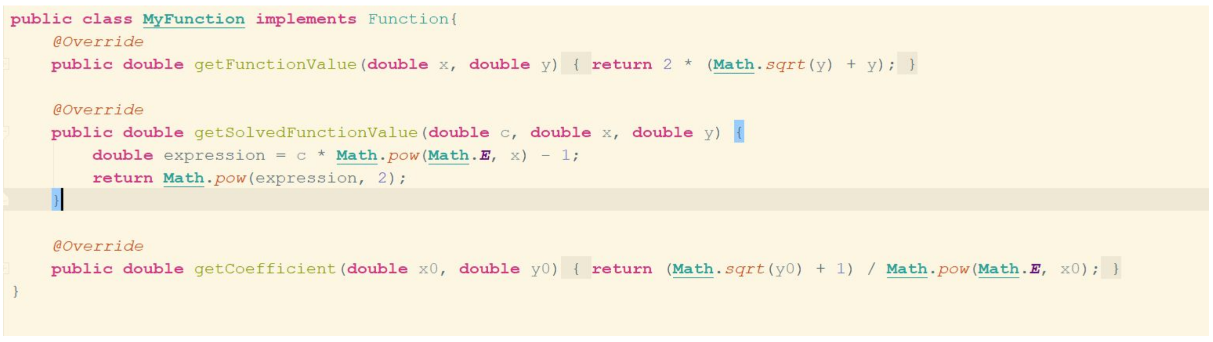
```
public class Variables {

    private double N;
    private double x0;
    private double y0;
    private double X;
    private double n0;
    private double nN;

    Variables(){
        setN(90);
        setX0(0);
        setY0(1);
        setX(9);
        setN0(10);
        setnN(30);
    }

    public double getX0() { return x0; }

    public void setX0(double x0) { this.x0 = x0; }

    public double getY0() { return y0; }

    public void setY0(double y0) { this.y0 = y0; }

    public double getX() { return X; }

    public void setX(double x) { X = x; }
```

This is interface for function and if I want to add new function, I just need to change 2 lines of code and realize it's methods.



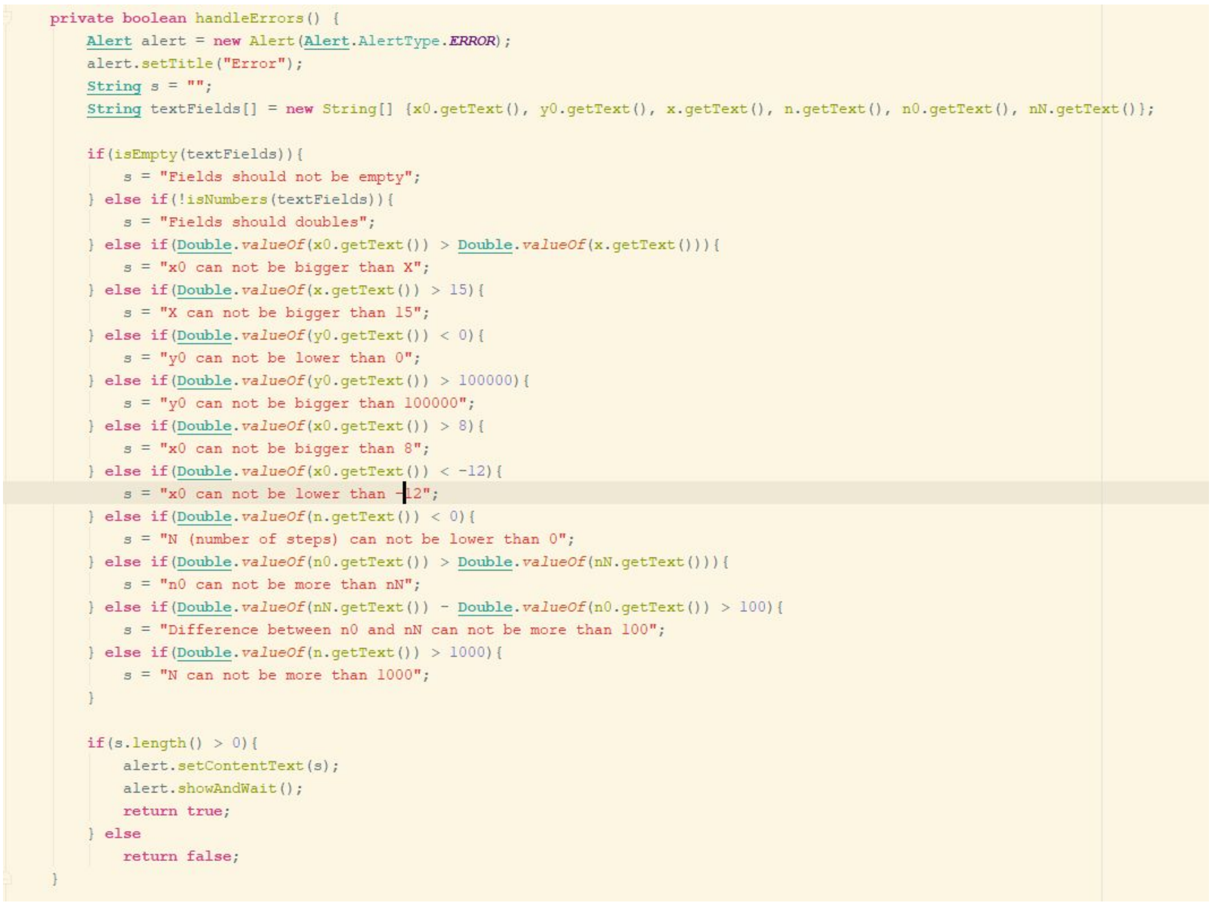
```
public interface Function {

    public double getFunctionValue(double x, double y);
    public double getSolvedFunctionValue(double c, double x, double y);
    public double getCoefficient(double x0, double y0);
}
```

This is my class, which realize function interface.



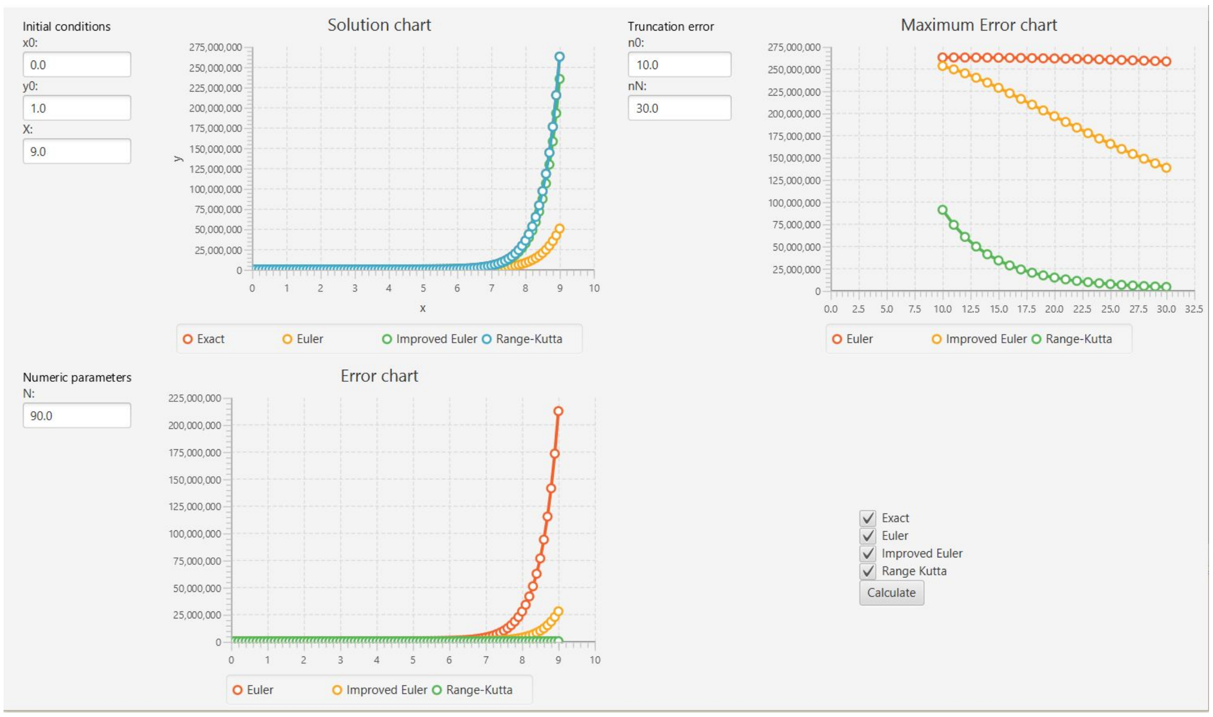
```java
public class MyFunction implements Function{
    @Override
    public double getFunctionValue(double x, double y) { return 2 * (Math.sqrt(y) + y); }

    @Override
    public double getSolvedFunctionValue(double c, double x, double y) {
        double expression = c * Math.pow(Math.E, x) - 1;
        return Math.pow(expression, 2);
    }

    @Override
    public double getCoefficient(double x0, double y0) { return (Math.sqrt(y0) + 1) / Math.pow(Math.E, x0); }
}
```

This method for handling errors. In my function y should be more than 0, also fields can not be empty, they should contain only numbers, number of steps should be more than 0, n0 can not be more than nN. Also I add limitation to x0, because otherwise computer can not compute it values and program will crash. Limitations for x0: -12 < x0 < 8, for y0: 0 < y0 < 100000, for X: X < 15, for N: 0 < N < 100, for n0 and nN: (nN - n0) < 100.



```java
    private boolean handleErrors() {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Error");
        String s = "";
        String textFields[] = new String[] {x0.getText(), y0.getText(), x.getText(), n.getText(), n0.getText(), nN.getText()};

        if(isEmpty(textFields)){
            s = "Fields should not be empty";
        } else if(!isNumbers(textFields)){
            s = "Fields should doubles";
        } else if(Double.valueOf(x0.getText()) > Double.valueOf(x.getText())){
            s = "x0 can not be bigger than X";
        } else if(Double.valueOf(x.getText()) > 15){
            s = "X can not be bigger than 15";
        } else if(Double.valueOf(y0.getText()) < 0){
            s = "y0 can not be lower than 0";
        } else if(Double.valueOf(y0.getText()) > 100000){
            s = "y0 can not be bigger than 100000";
        } else if(Double.valueOf(x0.getText()) > 8){
            s = "x0 can not be bigger than 8";
        } else if(Double.valueOf(x0.getText()) < -12){
            s = "x0 can not be lower than -12";
        } else if(Double.valueOf(n.getText()) < 0){
            s = "N (number of steps) can not be lower than 0";
        } else if(Double.valueOf(n0.getText()) > Double.valueOf(nN.getText())){
            s = "n0 can not be more than nN";
        } else if(Double.valueOf(nN.getText()) - Double.valueOf(n0.getText()) > 100){
            s = "Difference between n0 and nN can not be more than 100";
        } else if(Double.valueOf(n.getText()) > 1000){
            s = "N can not be more than 1000";
        }

        if(s.length() > 0){
            alert.setContentText(s);
            alert.showAndWait();
            return true;
        } else
            return false;
    }
```

Screenshots:


Initial conditions

x0: 0.0

y0: 1.0

X: 9.0

Numeric parameters

N: 90.0

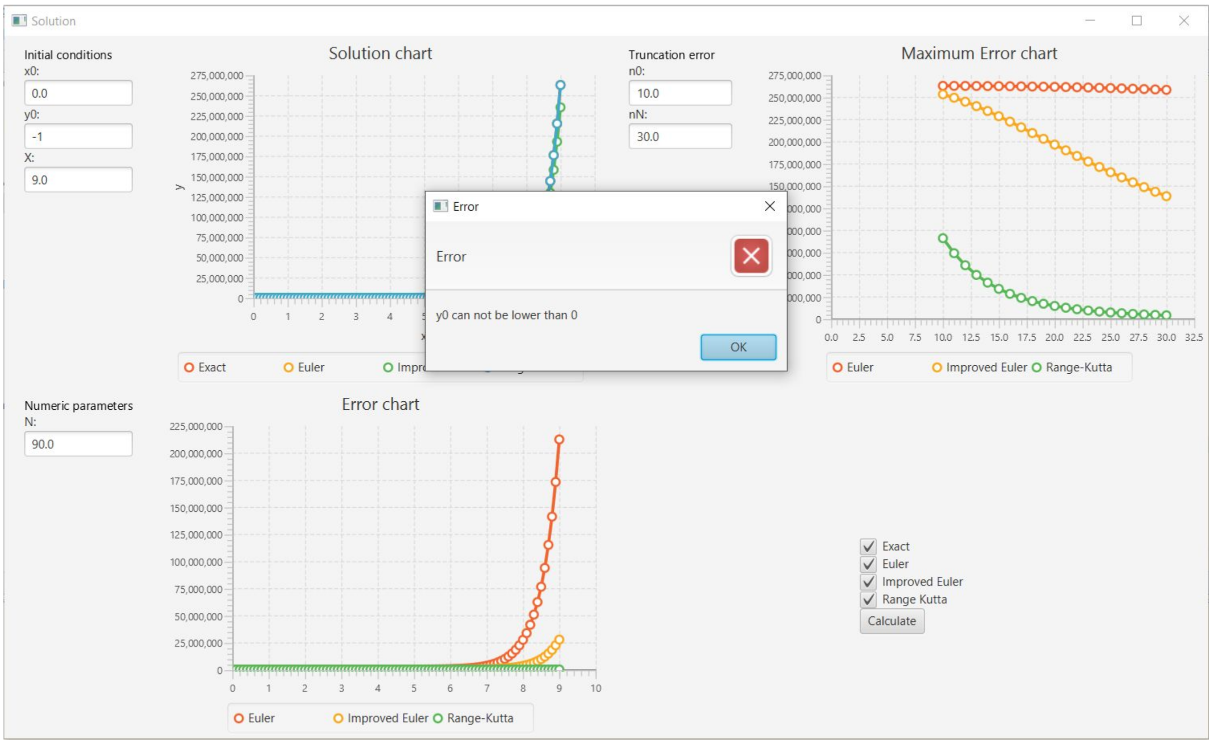
Truncation error

n0: 10.0

nN: 30.0

☑ Exact
☑ Euler
☑ Improved Euler
☑ Range Kutta

Calculate


Solution

Initial conditions

x0: 0.0

y0: -1

X: 9.0

Numeric parameters

N: 90.0

Truncation error

n0: 10.0

nN: 30.0

Error

Error

y0 can not be lower than 0

OK

☑ Exact
☑ Euler
☑ Improved Euler
☑ Range Kutta

Calculate