Report. Homework 1

Variant 1. Library for solving equations by numerical methods. EquationSolver library was created to support solving equations by numeric methods. The interface of simple operation:

```
public interface Operation {
    // get value of variable and return the result
    public double solve(double var);
    // return derivative that is a derivation of current operation
    public Operation derivate();
    // return a string that describes operation
    public String description();
}
```

Library support addiction, subtracting, multiplying, division, power function, constants and variable. All operations get two operand in its initialization, constant get double value.

Interface for all methods:

```
public interface Solver {
    // get equation and return approximate root when this equation is equal to zero
    public double findRoot(Operation equation);
}
```

Such methods are supported: dichotomy method, chord method, Newton method.

A program was created to demonstrate library features. On next page you can see images with code and program output:

```java
public class Main {

    private static double epsilon = 0.00001;

    public static void main(String[] args) {
        double epsilon = 0.00001;
        System.out.println("Check dichotomy method.");
        check(getFirstEquation(), new DichotomySolver( leftBorder: 0,  rightBorder: 2, epsilon));
        System.out.println("Check chord method.");
        check(getSecondEquation(), new ChordSolver( firstCoord: 8,  secondCoord: 3, epsilon));
        System.out.println("Check Newnton method.");
        check(getThirdEquation(), new NewtonSolver( firstCoord: 0.6, epsilon));
    }

    private static void check(Operation equation, Solver solverMethod) {
        System.out.println("Equation: " + equation.description());
        double root = solverMethod.findRoot(equation);
        System.out.println("Root between 0 and 2 is: " + root);
        System.out.println(equation.description() + " ~= " +  equation.solve(root) + " when x = " + root);
    }

    // 4 - e^x - 2x^2
    private static Operation getFirstEquation() {
        Operation first = new Constant( number: 4);
        Operation second = new Power(new Constant(Math.exp(1)), new Variable());
        Operation third = new Multiply(new Constant( number: 2), new Power(new Variable(), new Constant( number: 2)));
        return new Minus(new Minus(first, second), third);
    }

    // x^3 - 18x - 83
    private static Operation getSecondEquation() {
        Operation first = new Power(new Variable(), new Constant( number: 3));
        Operation second = new Multiply(new Constant( number: 18), new Variable());
        Operation third = new Constant( number: 83);
        return new Minus(new Minus(first, second), third);
    }

    // e^x + 3x - 4
    private static Operation getThirdEquation() {
        Operation first = new Power(new Constant(Math.exp(1)), new Variable());
        Operation second = new Multiply(new Constant( number: 3), new Variable());
        Operation third = new Constant( number: 4);
        return new Minus(new Add(first, second), third);
    }
}
```

```
Check dichotomy method.
Equation: ((4.0 - 2.718281828459045^(x)) - 2.0 * x^(2.0))
Root between 0 and 2 is: 0.8867683410644531
((4.0 - 2.718281828459045^(x)) - 2.0 * x^(2.0)) ~= 1.0973681433767979E-5 when x = 0.8867683410644531
Check chord method.
Equation: ((x^(3.0) - 18.0 * x) - 83.0)
Root between 0 and 2 is: 5.70511579682103
((x^(3.0) - 18.0 * x) - 83.0) ~= 3.7803317809448345E-8 when x = 5.70511579682103
Check Newnton method.
Equation: ((2.718281828459045^(x) + 3.0 * x) - 4.0)
Root between 0 and 2 is: 0.677208331718951
((2.718281828459045^(x) + 3.0 * x) - 4.0) ~= 6.750155989720952E-14 when x = 0.677208331718951

Process finished with exit code 0
```

Oleksii Andriushchenko, AI-1