How to get ECJ running on your computer?

This tutorial was written using following technologies:

1. Mac OS X El Capitan
2. Eclipse Luna/Mars/Neon for Java Developers
3. Java SE JDK 6.0/7.0/8.0
4. ECJ23

As well I would like to mention that this approach might not be an excellent way, but it totally works and you can explore beyond it if you wish.

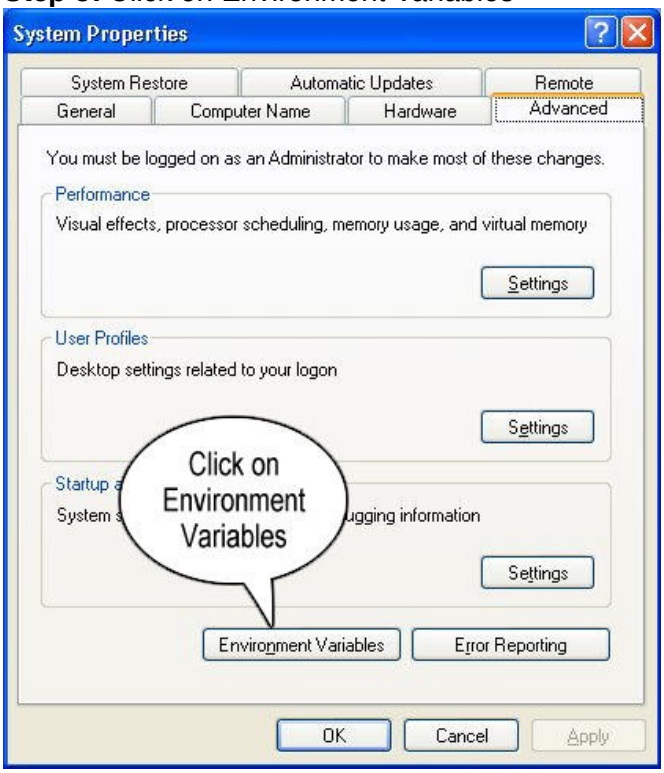
First thing is to download everything we need: Eclipse, Java SE JDK, and ECJ:

1. https://eclipse.org/downloads/ - Be sure to download proper Eclipse version based on your OS and bit rate. Install Eclipse on your machine.
2. http://www.oracle.com/technetwork/java/javase/downloads/index.html - Click download button under JDK label. It will bring you to the latest available JDK page, accept terms and download it. Install the JDK after that. It should be as simple as double-click and then go through some steps.
3. https://cs.gmu.edu/~eclab/projects/ecj/ - Simply download the jar file. Under Download ECJ section there is ecj.23.jar link you can click and download will start.

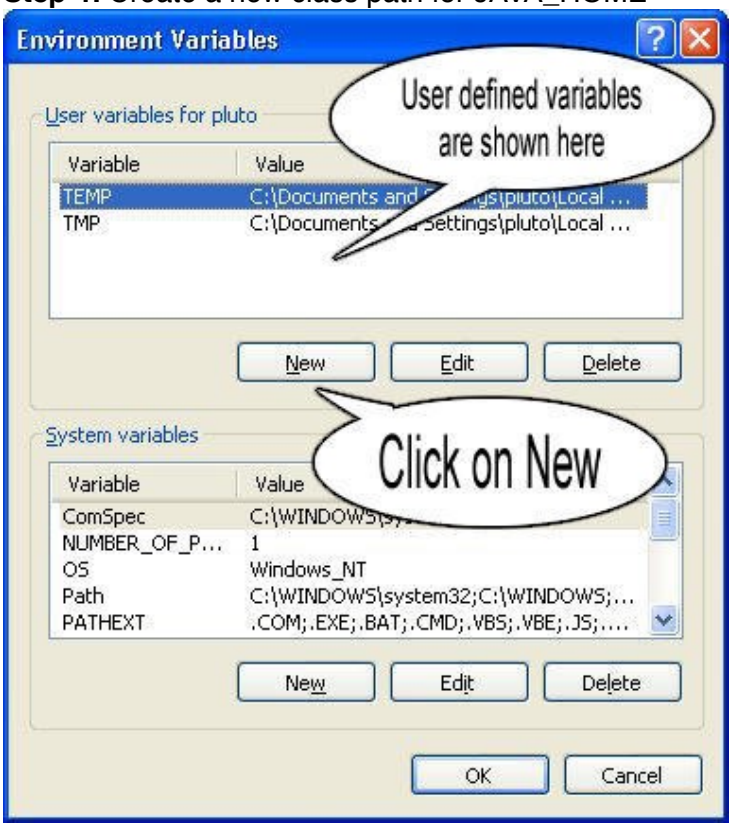
If you use windows, you might need to add java to your PATH. Here is a great link which could guide you through: http://stackoverflow.com/questions/1672281/environment-variables-for-java-installation. Just in case the link will not be available to you, I'll copy the top answer here, but I don't get any credit for it:

Step 1 : Right Click on My Computer and click on properties.
Step 2 : Click on Advanced tab

Step 3: Click on Environment Variables



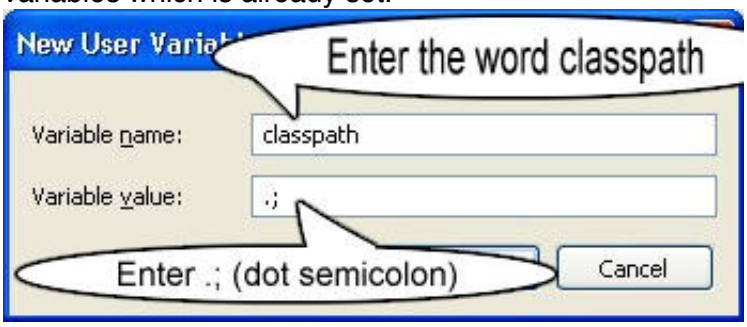
Step 4: Create a new class path for JAVA_HOME



Step 5: Enter the Variable name as JAVA_HOME and the value to your jdk bin path i.e.: C:\Programfiles\Java\jdk-1.6\bin - where this root is where you installed JDK

Step 6 : Now select Path in System Variables and press Edit. Add the following `.;C:\Programfiles\Java\jdk-1.6\bin` in the value field - where this root is where you installed JDK

NOTE Make sure you start with **.;** in the Value so that it doesn't corrupt the other environment variables which is already set.



Step 7 : Your are done setting up your environment variables for your Java , In order to test it go to command prompt and type

java

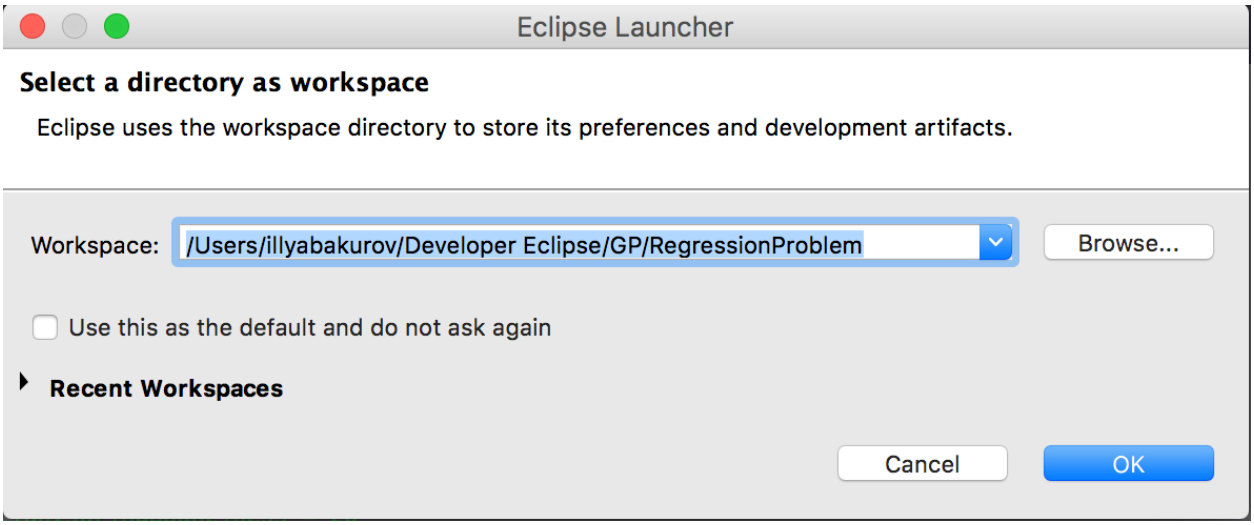
In order make sure whether compiler is setup Type in cmd

javac

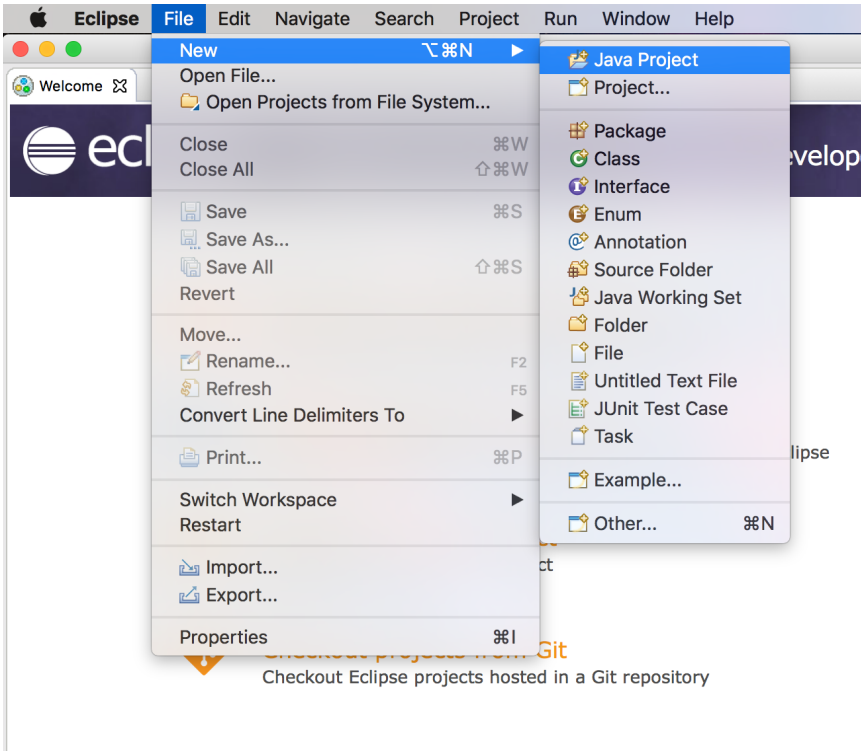
Mac OS users don't need to do anything, because PATH variable is set up for them while installing Java JDK.

Ok, once everything is on our computer, we can get ECJ running. Too make it more interesting let's take regression problem as our task, and we will try to run it.

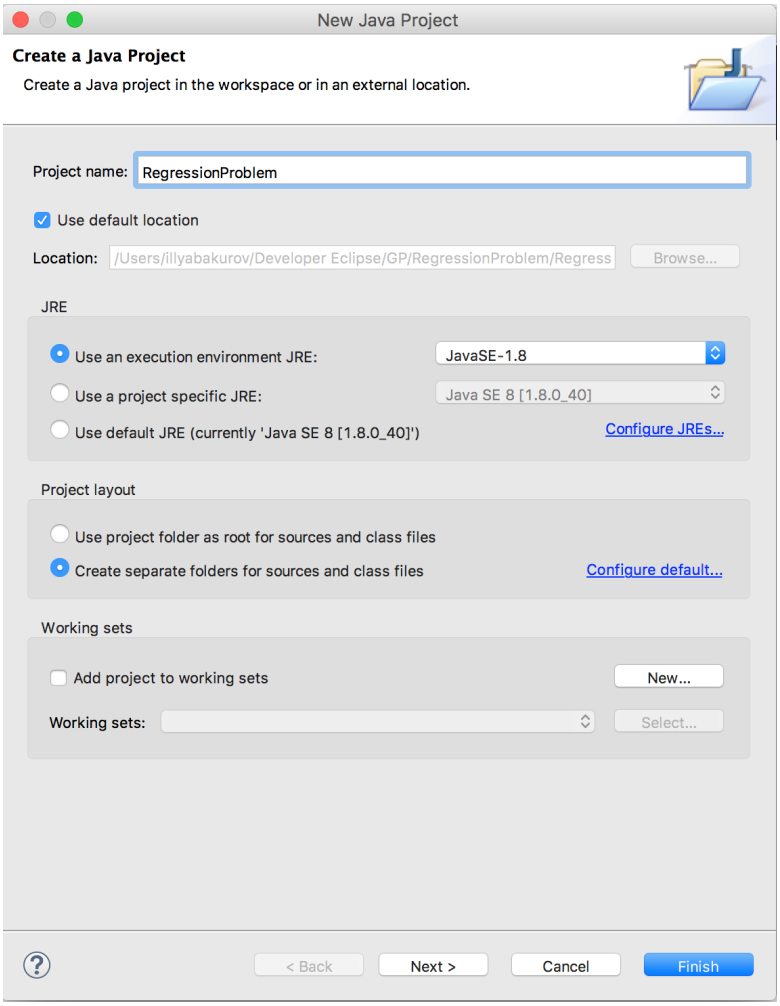
1. Open an Eclipse and choose destination folder for your workspace.



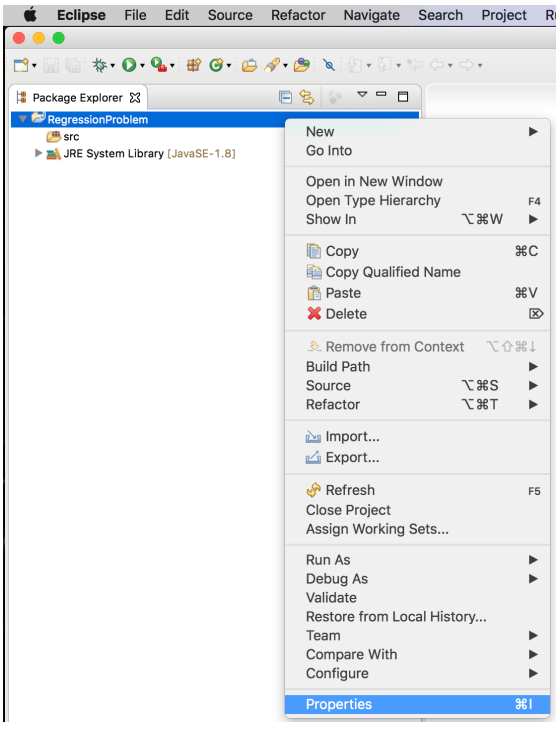
2. Create new Java Project.



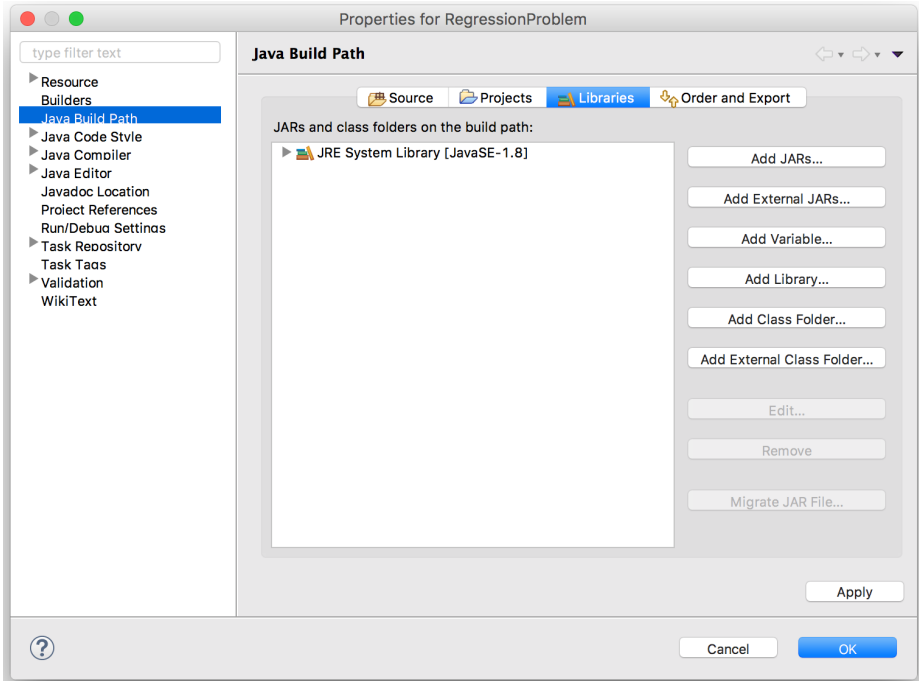
3. Give it a name, I chose RegressionProblem. And Press Finish.



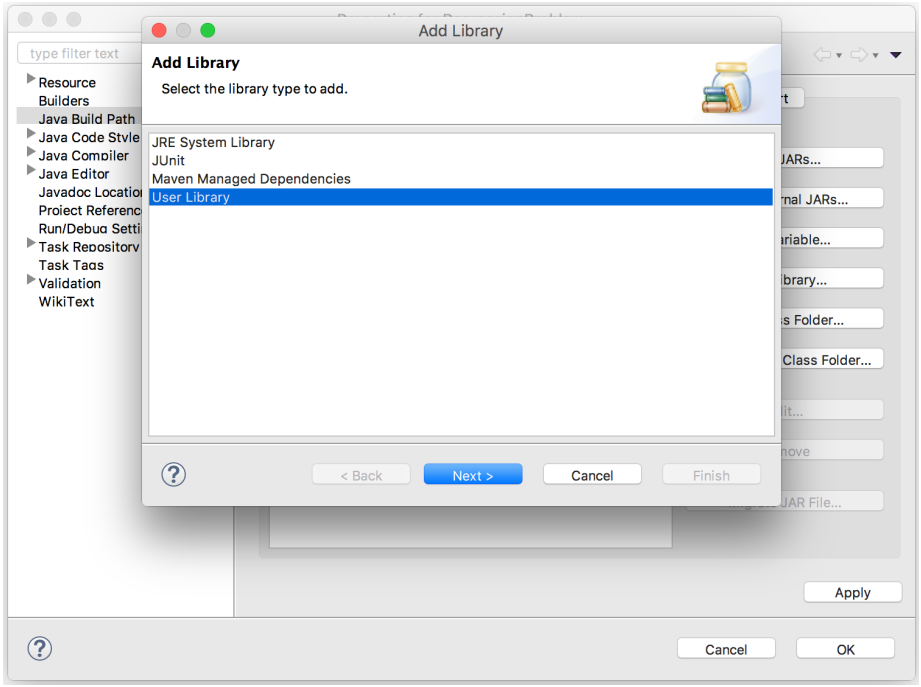
4. Go into properties of The Project. Right click the project in Project Navigator.



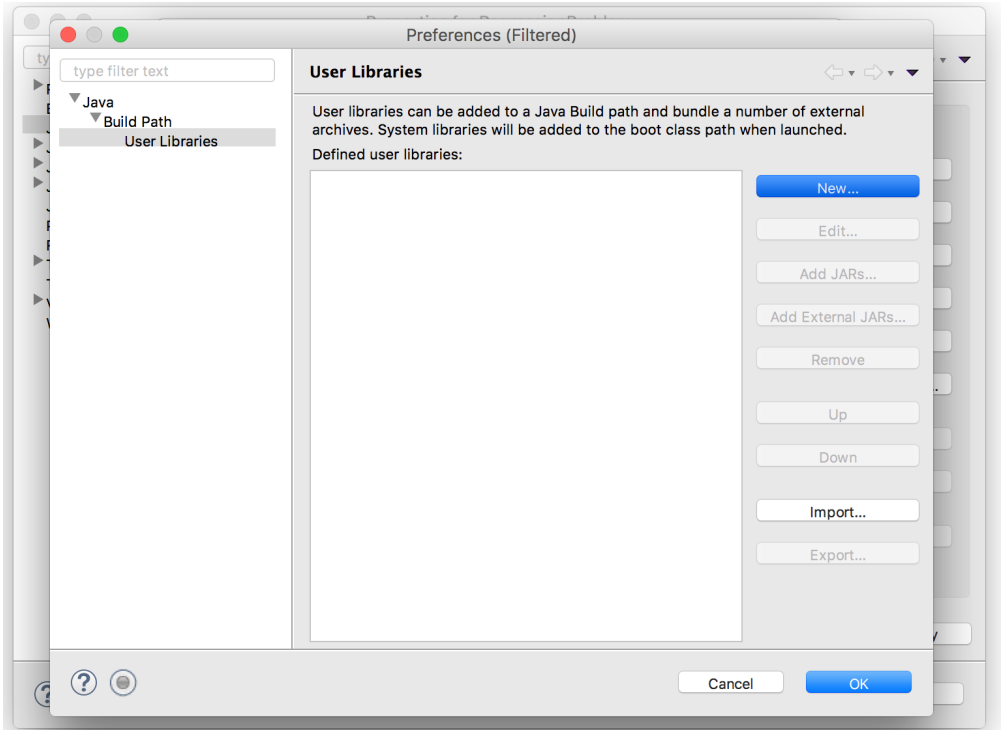
5. Choose Java Build Path section on the left column and Libraries on the top segment chooser.



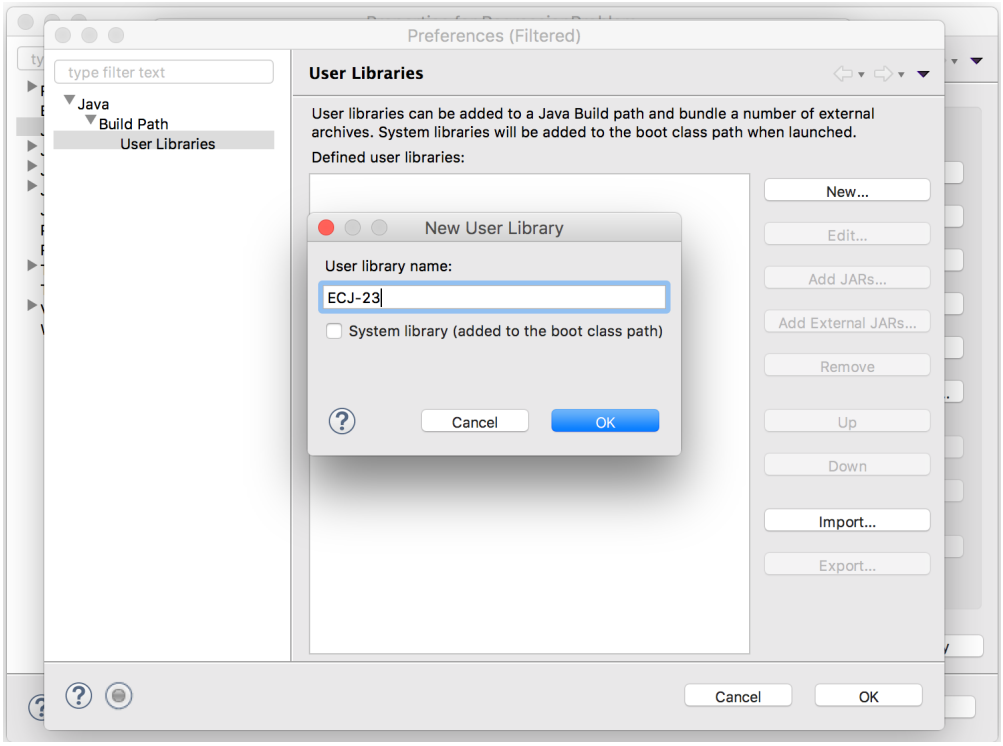
6. On the right column with actions choose Add Library… And on the pop up screen choose User Library. Press Next.



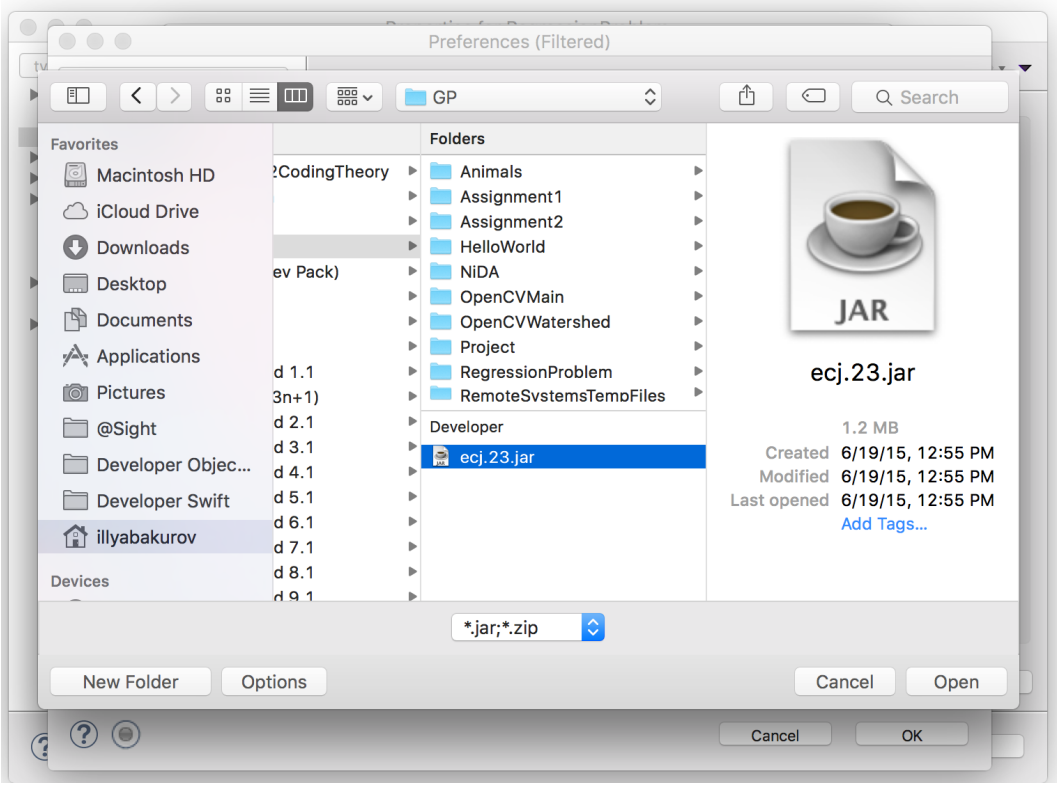
7. Select User Libraries… on the right side if it is not selected for you. And press New..



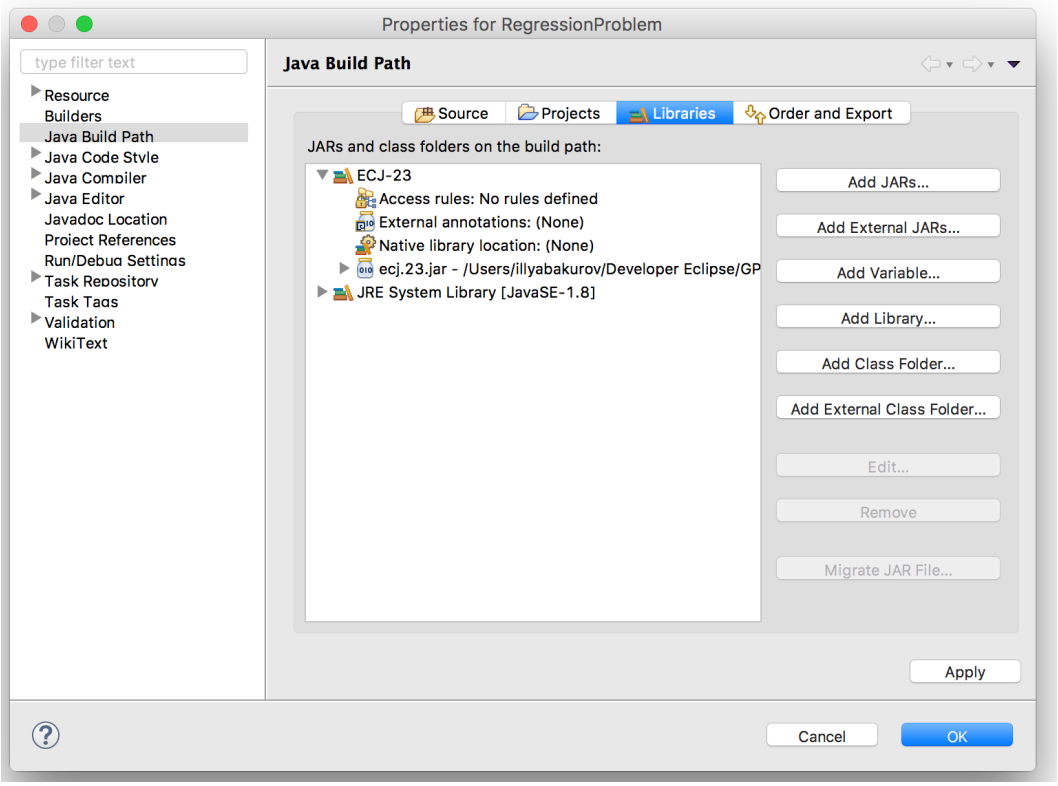
8. Choose any name you want, for me the most which makes sense is ECJ-23. Press OK.



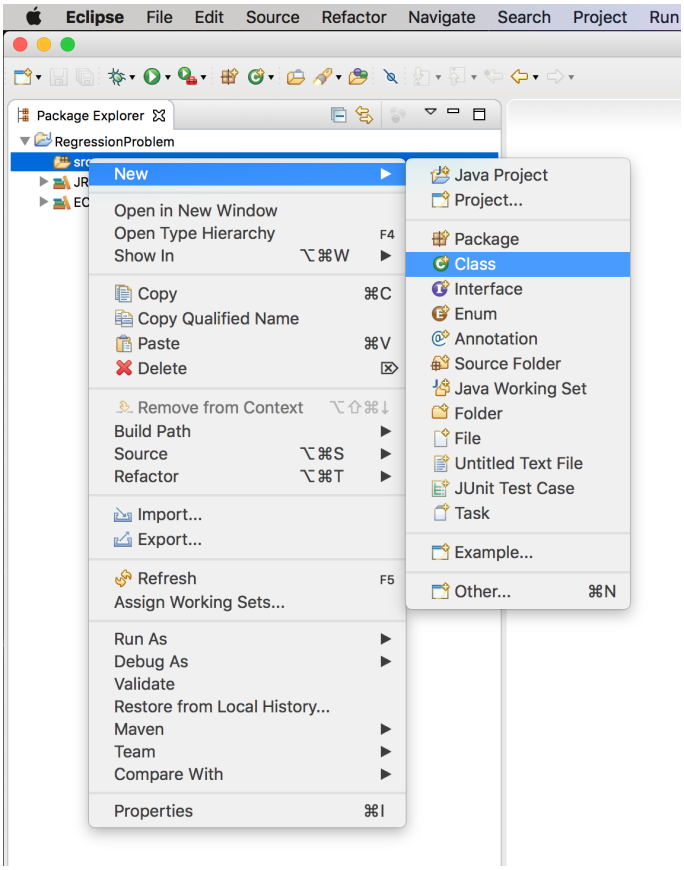
9. You can see that more options on the right side is available now. Choose Add External Jars… And navigate to your downloaded ecj.23.jar file. In my case it looks like the following. Submit selection with pressing Open. **Note**: Put ecj.23.jar to some folder on your computer, so that you don't touch it anytime or change its destination.



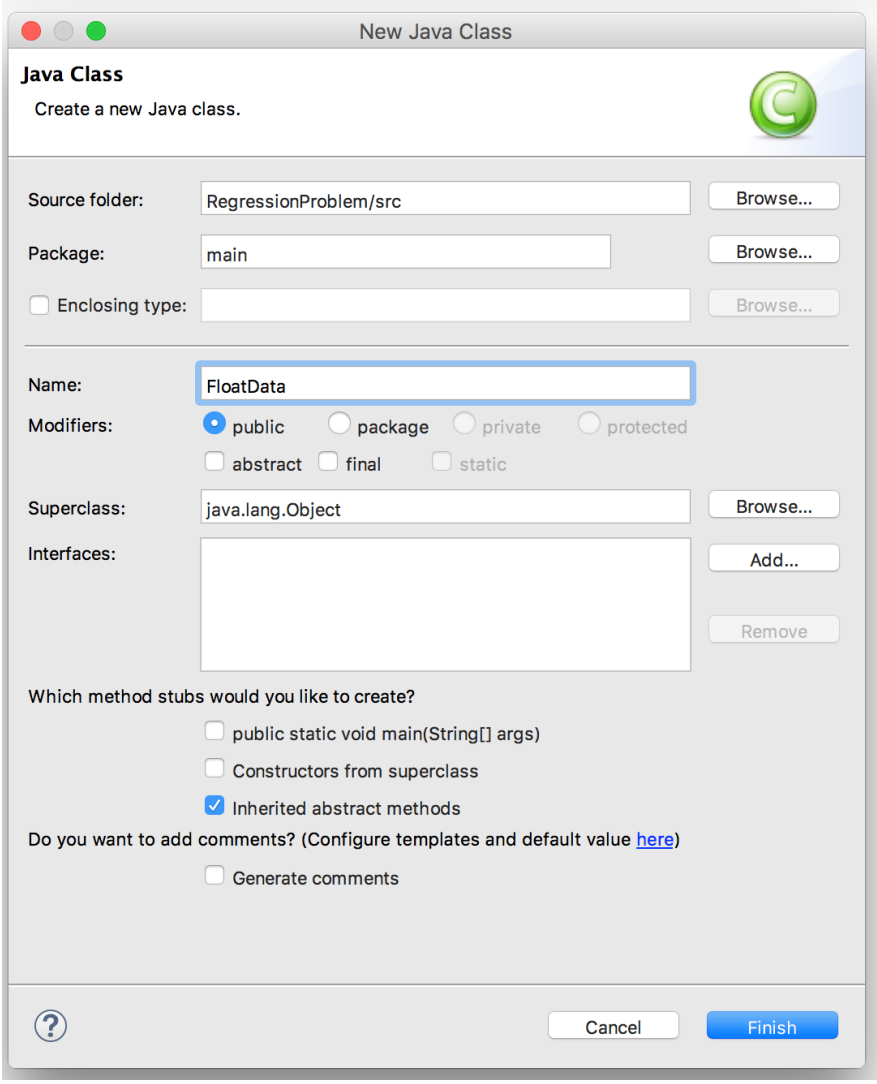
10. Now you are all set here. Press OK and then Finish. You should see something like this:



11. If under any circumstances you change the destination of ecj.23.jar inside your computer, you will need to go to your User Libraries in Eclipse and change the path to the jar file under ECJ-23 Library. You also could simply Add External JARs… from the screen in step 5 or 10, but the way we did it, will give you a chance to add ECJ library to your Java projects faster and easier, because next time you press Add Library… you will see ECJ-23 ready to be added. No hustle navigating jar file on your computer in the future.
12. Simply press OK. ECJ is connected to your Java Project now. So how we can get use of it?
13. At this point I would recommend go over this tutorial: https://cs.gmu.edu/~eclab/projects/ecj/docs/tutorials/tutorial4/ and just have a sneak peek of what is ECJ about. It will be confusing, no doubts here. But it will be easier for us to see what is done and why it is done so with Regression Problem example.
14. Ok in GP we have data types, language with our functions and terminals, tree generation and fitness evaluation (tree evaluation and fitness assignment). Tree generation and tree evaluation is what ECJ takes care of. We are in charge of setting everything else. We need to tell ECJ what data types we have, what is our language consists of and how those functions perform, and, of course, we need to provide ECJ with fitness evaluation formulas.
15. Let's start from data types. For Regression Formula we will use Float data. Go ahead and create the FloatData class file. Right click on src folder inside our project. New… -> Class



16. Type the name of the class. I chose FloatData. And you are also encouraged to create the package for that, to make your project structure more accurate. I chose main as a package name.



17. File should consist of:

```
package main;
import ec.gp.GPData;

@SuppressWarnings("serial")
public class FloatData extends GPData {
	public float x;

	public void copyTo(final GPData gpd) {
		((FloatData)gpd).x = x;
	}
}
```

Let's cover what's happening here. We import ECJ's GPData class. This is the base class for all Data Types which we want to use with of ECJ. So our FloatData extends GPData. We have our float variable *x* to hold the value of the terminal of Float type. And we also implement GPData's method copyTo, so ECJ could use it in tree evaluation. That's it, that's how we instantiated Float data type in ECJ.

18. Now we can get to instantiating our Problem class. Such class extends GPProblem and implements SimpleProblemForm. That way we can override ECJ's methods and receive results of its execution and work on those results - apply fitness evaluation.
19. I cannot simply paste the full code here, because it is a lot. But you can totally have a look at it on this gitHub repository: https://github.com/illyaBakurov/RegressionProblemOnGPWithECJ . Let's have a look at RegressionProblem.java file.

```
public static final String P_DATA = "data";
```

This variable is must be there, because ECJ needs it. Honestly, I just believed the ECJ tutorial under the link I've provided in step 13 and let things go as they do.
Other than that variable, we have three more.

```
public float currentX;
public ArrayList<Float> inputX = new ArrayList<Float>();
public ArrayList<Float> inputY = new ArrayList<Float>();
```

The first one will be needed to assign actual value of the *x* on each step of our GP process. This variable will be taken as a terminal Value on tree evaluation step by ECJ.
Two others are arrays with all our input values for Regression problem. The easiest example for this problem could be following:

x	y
2	4
3	9
4	16
5	25

You can easily see the dependency between *x* and *y* and based on previous experience and knowledge easily answer that this could be written as $y = x^2$. But now we want the program which only has input data to figure this formula by its own. So *inputX* and *inputY* will store those values. It is good idea to organize your program, so it could read values from .txt file, that way it is easier to differentiate the code and the values and make it more reusable.
Here is a simple function to read values from file into two arrays we have here:

```
private void readFromFile() {
        Scanner scan;
        File file = new File("src/packageName/fileName.txt");
        try {
            scan = new Scanner(file);
            int indexOfNumberInFile = 0;
            while(scan.hasNextFloat())
            {
                float floatNumber = scan.nextFloat();
```

```
                if (indexOfNumberInFile % 2 == 0) {
                      inputX.add(floatNumber);
                } else {
                      inputY.add(floatNumber);
                }
                indexOfNumberInFile++;
            }

        } catch (FileNotFoundException e1) {
                e1.printStackTrace();
        }
    }
```

And .txt file has super simple structure:

2 4
3 9
4 16
5 25

and so on. No commas, however if you want to have float values in your file, you should check on what your system's sign for decimal number wether it is ',' or '.' and use it inside the .txt file.

20. Ok the next good question would be when can you read those values, so ECJ could use them? ECJ has the method setup, which gets called before any GP stuff gets involved. That's where you should instantiate all of your data. And that's how it looks like for our problem:

```
 public void setup(final EvolutionState state,
                        final Parameter base) {
        // very important, remember this
        super.setup(state,base);

        // verify our input is the right class (or subclasses from it)
        if (!(input instanceof FloatData)) {
            state.output.fatal("GPData class must subclass from " +
FloatData.class,
                  base.push(P_DATA), null);
        }

        readFromFile();
    }
```

21. Finally this class will handle fitness evaluation of the GP results. ECJ provide us with the handy function called 'evaluate' and it will be called for each individual in population. That's what it will look like of your problem:

```
public void evaluate(final EvolutionState state,
            final Individual ind,
            final int subpopulation,
            final int threadnum) {
           if (!ind.evaluated) {  // don't bother reevaluating
```

```
                FloatData input = (FloatData)(this.input);

                int hits = 0;
                float sum = 0;
                float expectedResult;
                float result;
                for (int i = 0; i < inputX.size(); i++) {
                     currentX = inputX.get(i);
                    expectedResult = inputY.get(i);
                    ((GPIndividual)ind).trees[0].child.eval(
                        state,threadnum,input,stack,
((GPIndividual)ind),this);

                    result = Math.abs(expectedResult - input.x);
                    if (result <= 0.01) hits++;
                    sum += result;
                }

                // the fitness better be KozaFitness!
                KozaFitness f = ((KozaFitness)ind.fitness);
                f.setStandardizedFitness(state, sum);
                f.hits = hits;
                ind.evaluated = true;
            }
        }
```

If individual was already evaluated before, we don't need to worry about it, we already has the result for it. Otherwise, we want to evaluate the tree. Implementation of this function strongly connected to the fitness formula you are using to assess individuals. This example shows the simplest one with sum-of-hits - how many times individual gave us the right answer. However almost all of evaluate methods will have for-loop on the input array, to evaluate individual based on all of our data. we take *x* input and *y* input, and we ask ECJ to evaluate tree, ECJ will use *currentX* value inside of the tree evaluation, when the terminal was generated as the node of the tree. Finally, we calculate the difference between the result and expectedResult, and because we use Float data, we expect some computational oscillation on 0.01. We assign evaluation of this individual as KozaFitness, so that ECJ knows how to generate next generation based on individual fitnesses. That's it, that would be the whole Problem class.

22. Now there are still quite a few things to do. We need to define Language for our GP. Each separate function is defined in separate class. So it is good idea to create 'functions' package and store them all there, because we can have a lot. I will give an example on three different things here: arithmetical function, Ephemeral and terminal.

I. Functions

```
package main

import main.FloatData;
import ec.EvolutionState;
import ec.Problem;
import ec.gp.ADFStack;
import ec.gp.GPData;
import ec.gp.GPIndividual;
import ec.gp.GPNode;

@SuppressWarnings("serial")
public class Add extends GPNode {

      public String toString() { return "+"; }

      public int expectedChildren() { return 2; }

      public void eval(final EvolutionState state,
                  final int thread,
                  final GPData input,
                  final ADFStack stack,
                  final GPIndividual individual,
                  final Problem problem) {
            float result;
            FloatData rd = ((FloatData)(input));


children[0].eval(state,thread,input,stack,individual,problem);
            result = rd.x;


children[1].eval(state,thread,input,stack,individual,problem);
            rd.x = result + rd.x;
      }
}
```

Here we define the class for function '+'. We provide toString function, so it will be nicely and descriptively printed into our logs. We return the number of values this function expects to receive. Let's say if we had 'sin', we would return 1. And finally method 'eval' which ECJ calls when it evaluates tree. When we have two or more inputs in the function, we need to store each value of the branch under new variables. *rd* is the variable ECJ will save the result of evaluating the branch of the node in. We need to evaluate the left branch beneath this node and save the result of it in *result* variables and then we evaluate the right branch of this node, and we store

the final result back in *rd* variable, so ECJ could use it on the node above. The only things in other functions would change is the Class name, toString, expectedChildren and eval function (the way we read the results and perform actions on them)

II. Ephemeral

Ephemeral is the terminal, but the one which randomizes its value and never change it again.

```
package functions;

import java.util.Random;
import main.FloatData;
import ec.EvolutionState;
import ec.Problem;
import ec.gp.ADFStack;
import ec.gp.GPData;
import ec.gp.GPIndividual;
import ec.gp.GPNode;

@SuppressWarnings("serial")
public class Ephemeral extends GPNode {

      FloatData rd = new FloatData();

      public Ephemeral() {
            Random rand = new Random();
      rd.x = Math.round(rand.nextFloat());
      }

      public String toString() { return rd != null ?
String.valueOf(rd.x) : "n"; }

    public int expectedChildren() { return 0; }

    @Override
    public void eval(final EvolutionState state,
            final int thread,
            final GPData input,
            final ADFStack stack,
            final GPIndividual individual,
            final Problem problem) {
      ((FloatData)(input)).x = rd.x;
    }
}
```

It's not much different from function class, so I'll just leave it here for your reference. I use constructor of the class, to instantiate it once, and then just assign it's value to input of ECJ.

III. Terminal

Terminal is the node in the tree which will take the value of the *currentX* from problem class and return it to the node above.

```
package functions;

import main.FloatData;
import main.RegressionProblem;
import ec.EvolutionState;
import ec.Problem;
import ec.gp.ADFStack;
import ec.gp.GPData;
import ec.gp.GPIndividual;
import ec.gp.GPNode;

@SuppressWarnings("serial")
public class X extends GPNode {

      public String toString() { return "x"; }

    public int expectedChildren() { return 0; }

    @Override
    public void eval(final EvolutionState state,
            final int thread,
            final GPData input,
            final ADFStack stack,
            final GPIndividual individual,
            final Problem problem) {
      FloatData rd = ((FloatData)(input));
      rd.x = ((RegressionProblem)problem).currentX;
    }
}
```

It is simple and still looks similar to the function class.

Another thing to mention would be that all of those language classes extend GPNode, because that what they are, they are simple nodes in the tree, which ECJ connects with each other and then evaluate step by step, returning the final value based on input value to the evaluate method in problem class.

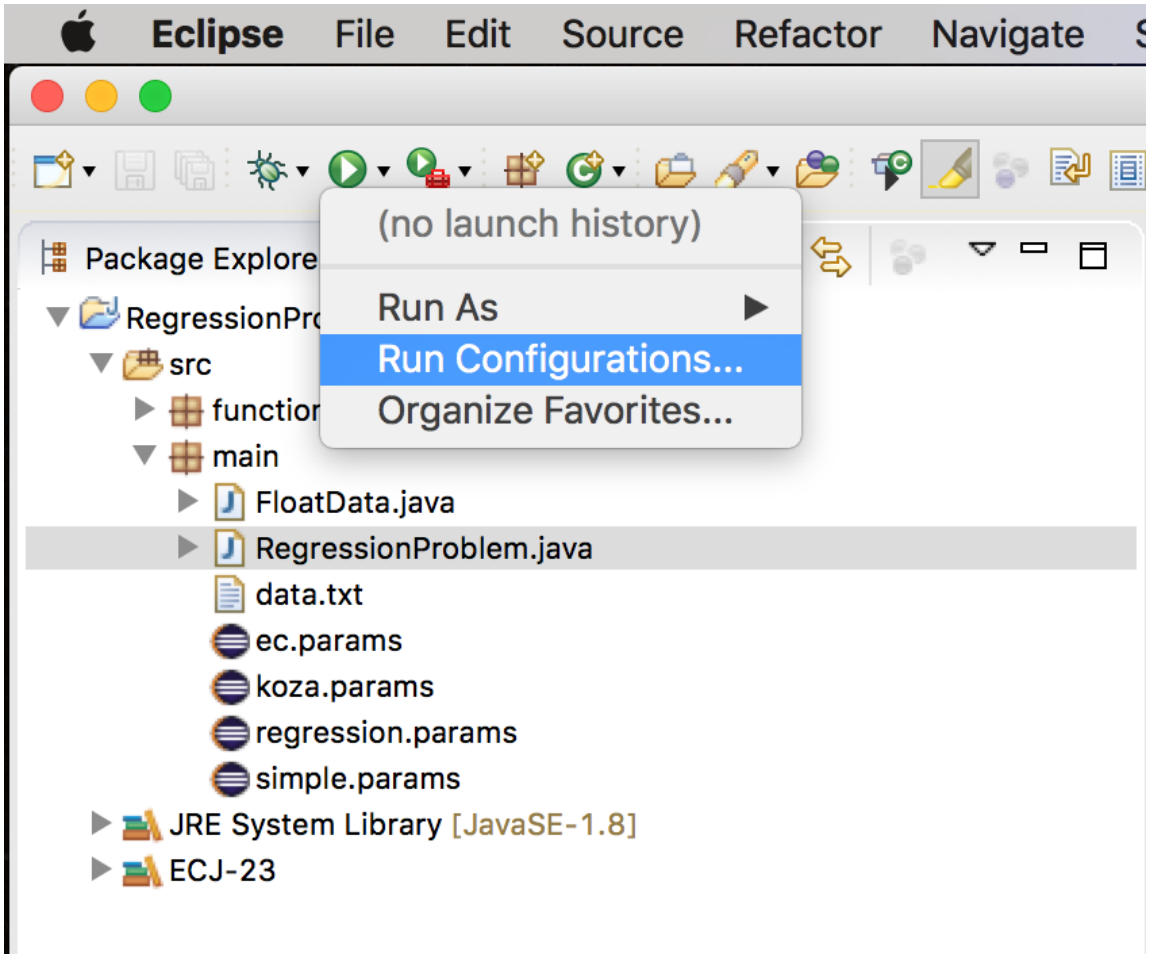
23. Finally, after we defined everything ECJ needs, we need to define GP parameters. This is done in .params files, which are technically simple text files. There is a whole bunch of

things you can define there. You are more then welcomed to check ecj's .params files to find out more. Some of the basics we will be needed for our regression problem are language, population size, number of generations, tournament size, problem, and data type, also the output file is defined there too. Here would be an example for our project:

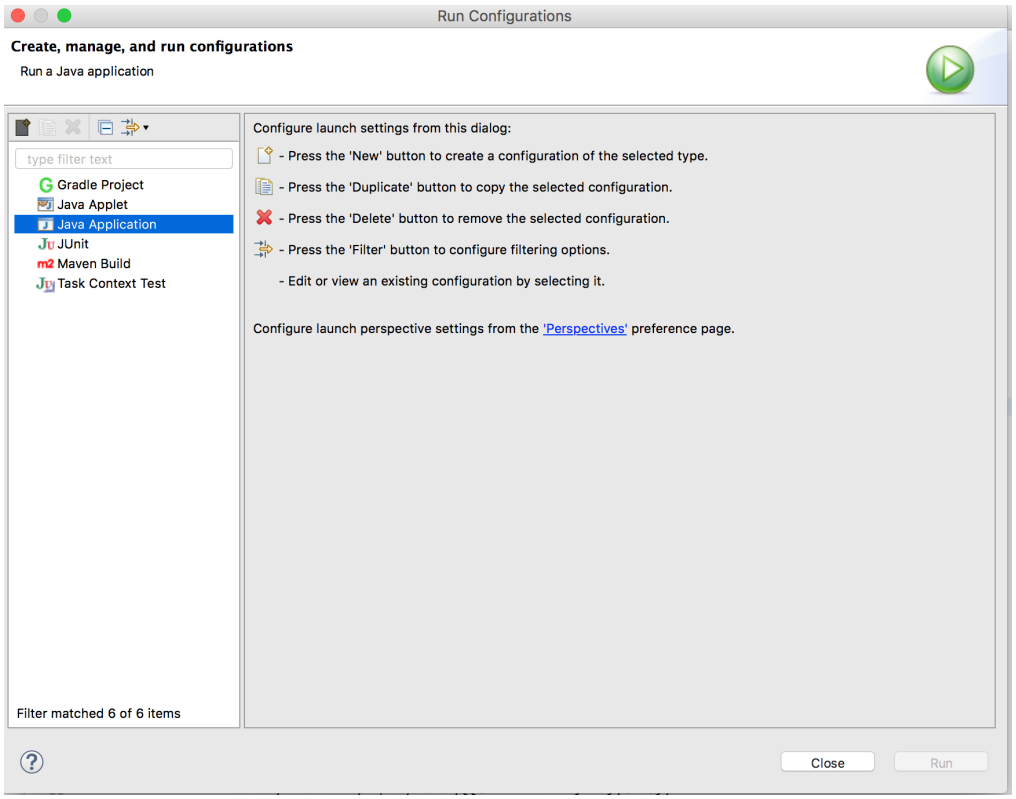
```
parent.0 = koza.params

# We have one function set, of class GPFunctionSet
gp.fs.size = 1
gp.fs.0 = ec.gp.GPFunctionSet
# We'll call the function set "f0".
gp.fs.0.name = f0

# We have five functions in the function set.  They are:
gp.fs.0.size = 8
gp.fs.0.func.0 = functions.X
gp.fs.0.func.0.nc = nc0
gp.fs.0.func.1 = functions.Mul
gp.fs.0.func.1.nc = nc2
gp.fs.0.func.2 = functions.Add
gp.fs.0.func.2.nc = nc2
gp.fs.0.func.3 = functions.Sub
gp.fs.0.func.3.nc = nc2
gp.fs.0.func.4 = functions.Div
gp.fs.0.func.4.nc = nc2
gp.fs.0.func.5 = functions.Ephemeral
gp.fs.0.func.5.nc = nc0
gp.fs.0.func.6 = functions.Sin
gp.fs.0.func.6.nc = nc1
gp.fs.0.func.7 = functions.Cos
gp.fs.0.func.7.nc = nc1

pop.subpop.0.size = 1024
generations = 51
select.tournament.size = 3

eval.problem = main.RegressionProblem
eval.problem.data = main.FloatData

# output statistics to the file "out.stat" in the directory
# the run was started in
stat.file                $out.stat
```

Notes: There is no difference in order of functions your define, just be 100% sure about ‘gp.fs. 0.size’ matching number of functions defined below. And that ‘nc’ - number of Children - matches the number you return in class definition.
Important Note: You can see this line ‘parent.0 = koza.params’ and you could be stumbled, what that kola.params means. That an ECJ definition of other params needed for the system, and you can find different param sets inside of ECJ or online. I put this file and two others, which kola.params needs to be inside of the project, you can find them under GitHub repo: https://github.com/illyaBakurov/RegressionProblemOnGPWithECJ.

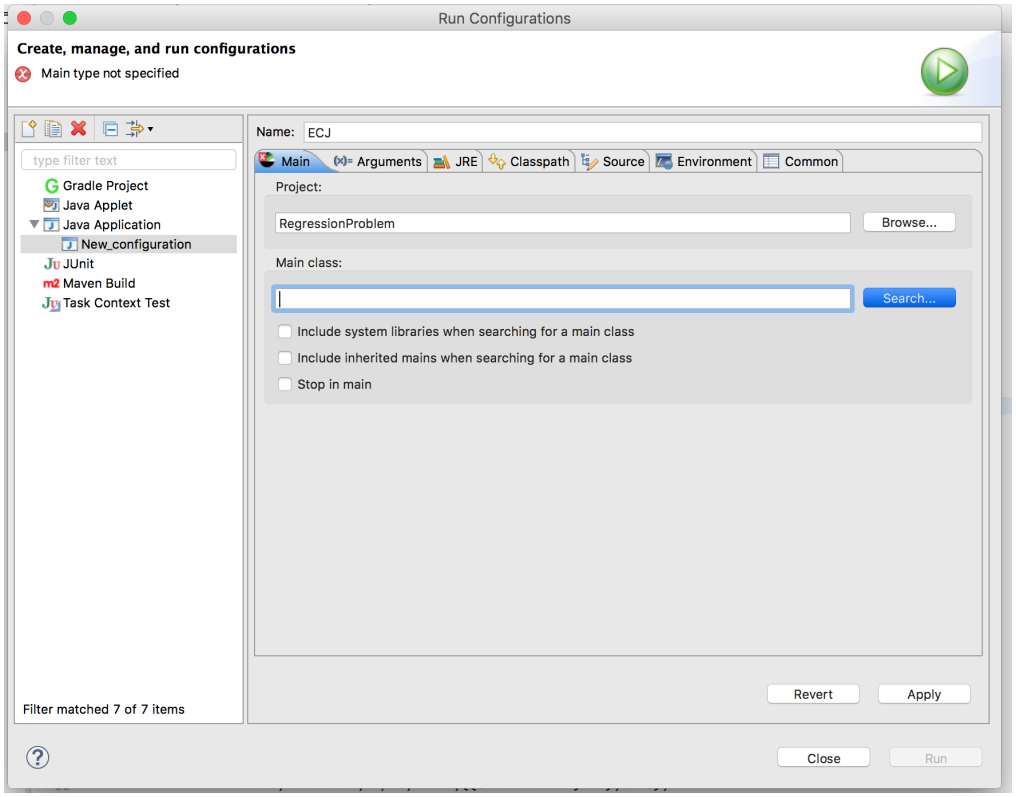
24. Don’t forget to include data.txt file or put proper file path inside of your Problem class in readFromFile function.
25. Finally, we can run the whole program. There are two ways to run ECJ app in Eclipse. With the help of configuration or using the main file. I will discuss both in that order.
26. Press on the arrow next to Play button, and then choose Run Configurations…



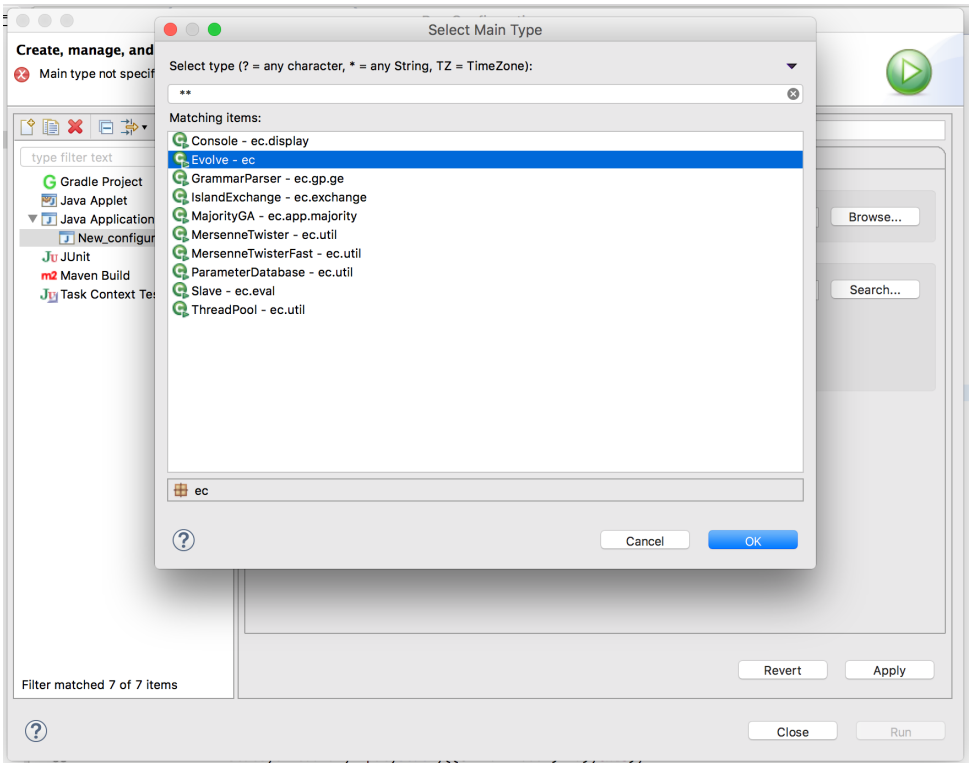
27. Choose ‘Java Application’ on the left panel, and then press this blank page with little yellow plus on top left, right above the ‘type filter text’ field. It is shaded out on the following image.



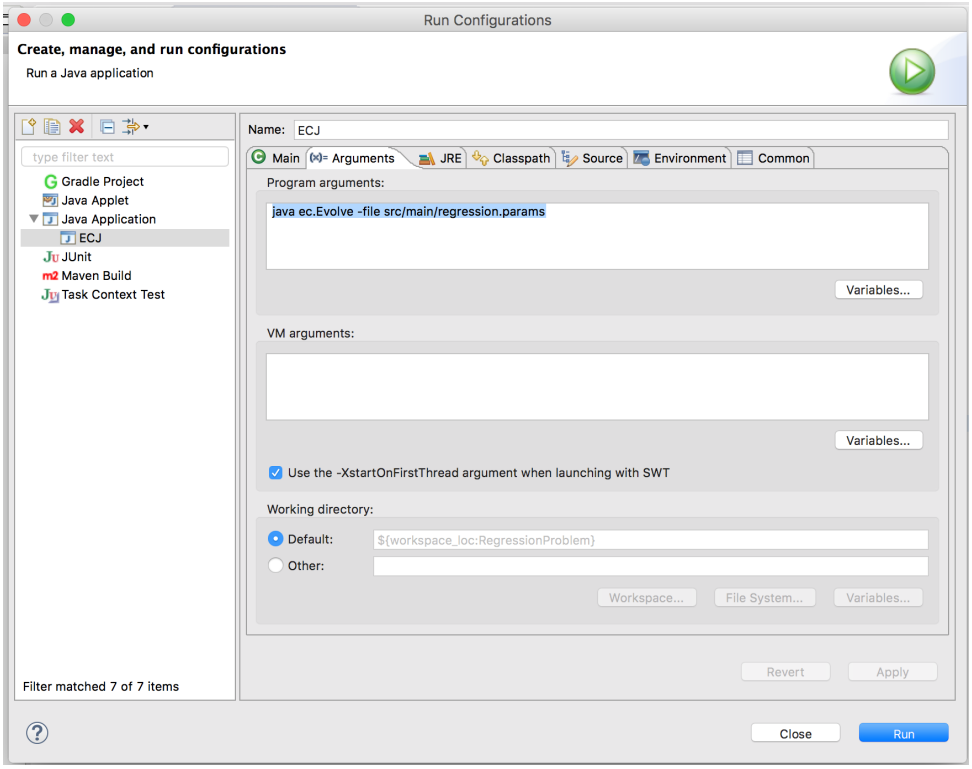
28. You can change the name on the top and then press Search… next to the Main class field



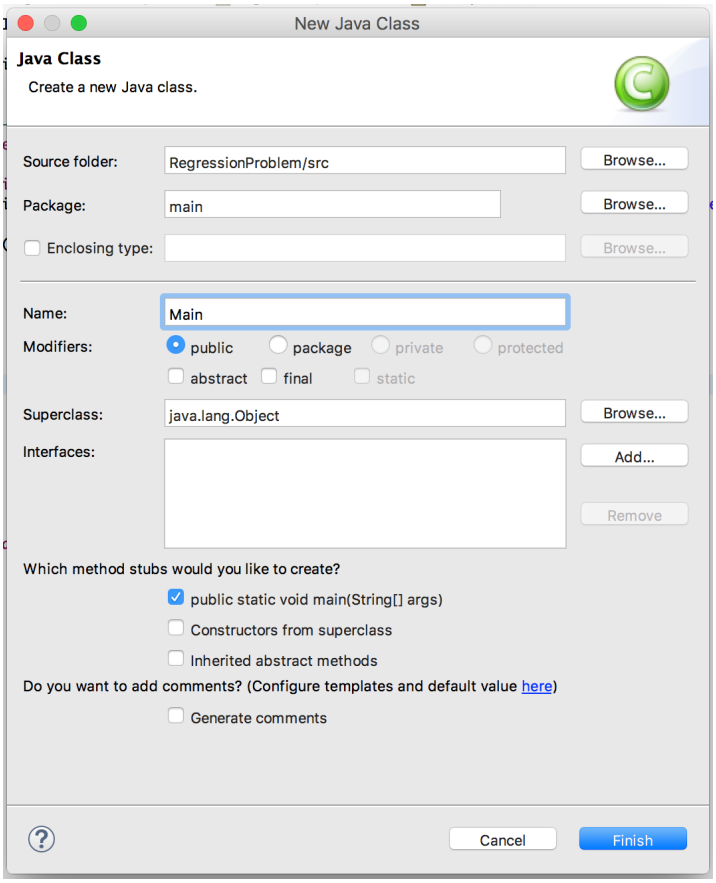
29. Choose Evolve - ec class, and press OK.



30. Switch to Arguments tab and enter this under Prgram arguments: java ec.Evolve -file src/main/regression.params. The path after -file command could change depending on where is your .params file. Also you could add extra parameters like how many jobs you want this ECJ to run or which output file to use, and so on i.e.: -p jobs=10, or -p stat.file=$out.stat, etc.



31. Finally you should be able to Run the program! That's it, you will see that your GP program started to run and some statistics are print out to the console.
32. However if you are like me and you don't like to go into Run Configurations every time you want to change something, it is time for us to create main file and run ECJ from it. Right click the main package and add Main Class, but be sure to check ***'public static void main(String[] args)'*** and uncheck ***'Inherited abstract methods'***:



33. And that's what it is going to look like:

```
package main;

import ec.Evolve;

public class Main {
	public static void main(String[] args) {
		String pathToFiles = "/Users/illyabakurov/Documents/Brock
University/GP/RegressionProblem/results/";
		int numberOfJobs = 10;
//		String statisticType = "ec.gp.koza.KozaShortStatistics";
		String[] runConfig = new String[] {
				Evolve.A_FILE, "src/main/regression.params",
```

```
//                      "-p", ("stat="+statisticType),
                        "-p", ("stat.file=$"+pathToFiles+"out.stat"),
                        "-p", ("jobs="+numberOfJobs)
                        };
            Evolve.main(runConfig);
      }
}
```

You can see that I have the path to the results folder, that's where my out.stat file would go, as a note I should say that that path should exist prior to running the program. I also has number of Jobs here, so I should not restart ECJ each time it finished running one GP program. It will also take care of out.stat files, it will create out1.stat and so on with every new job. If you do override jobs or out.stat file, be sure to comment them or delete them from .params file, because changes will take place based on .params values.
Now your press little arrow next to Play button, you should see Main there, and that's what you should run instead of ECJ, if you want the Eclipse to wind your setting from Main.java file.

If you have any questions or suggestions email me at: bakurov.illya@gmail.com