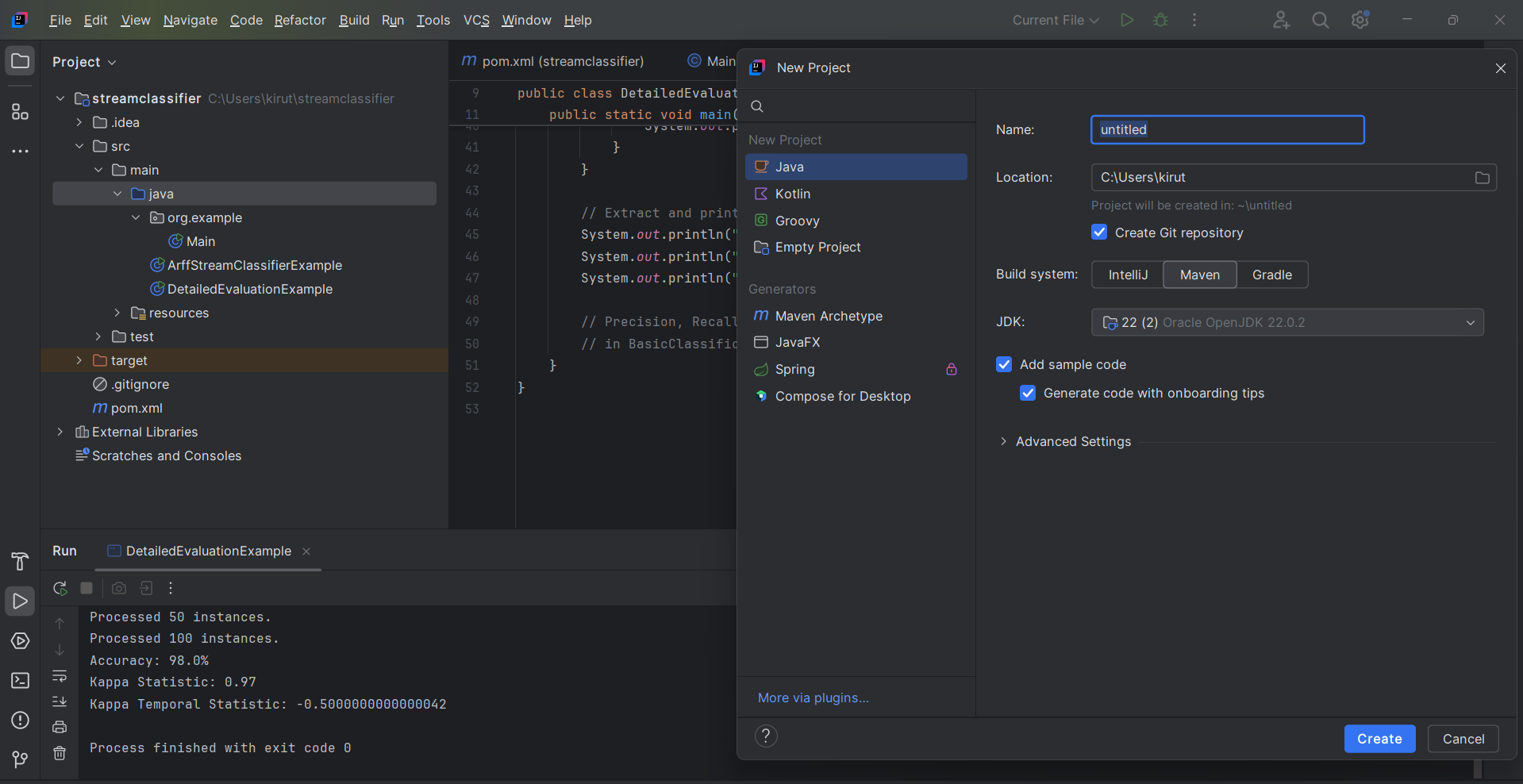
Using MOA API in java code- stream classifier.

Open intellij idea.

Create a java project .

To create the new project file\project\navigate to the location where you want to save the project.

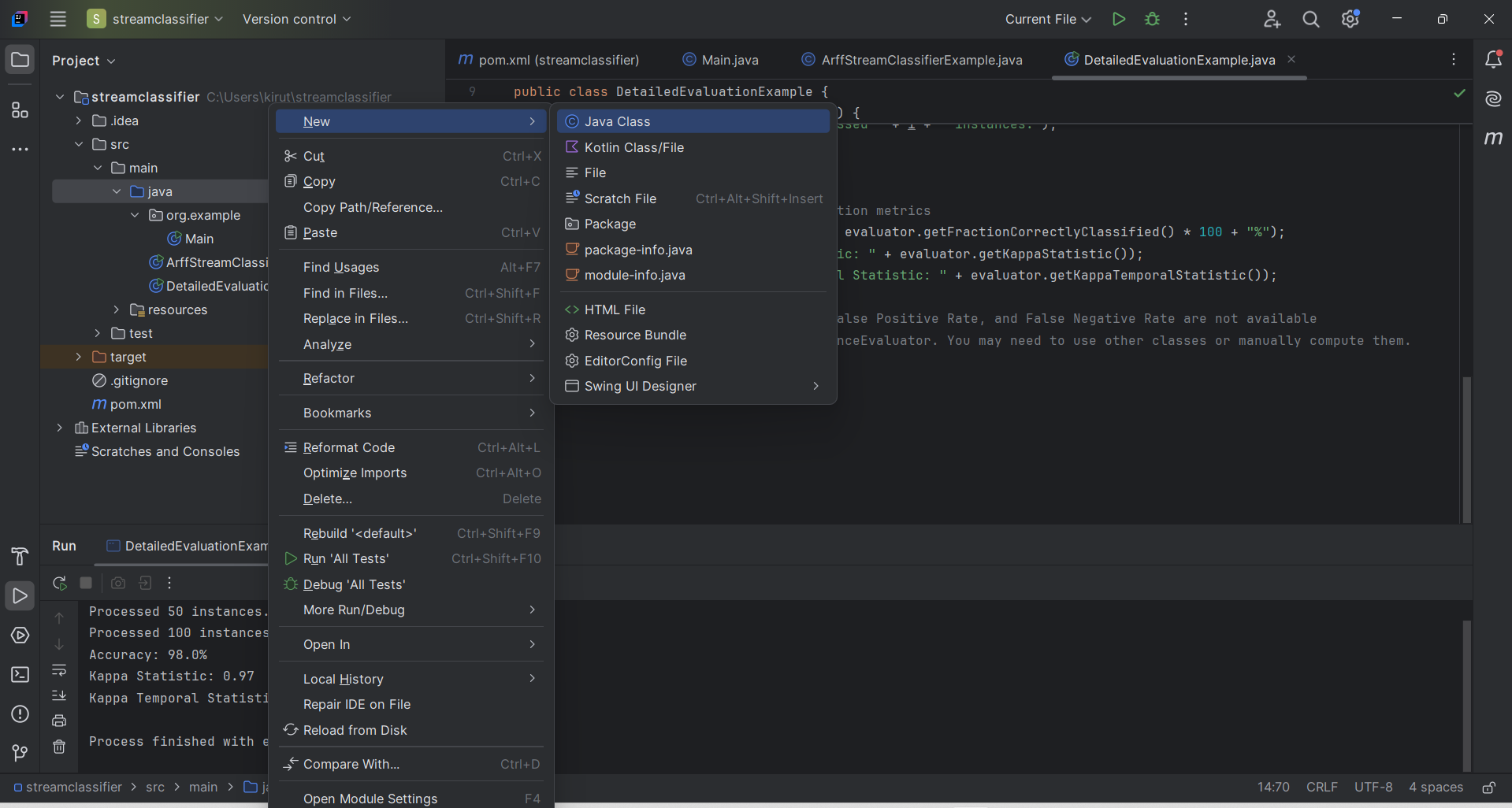


Then click create to create the project.

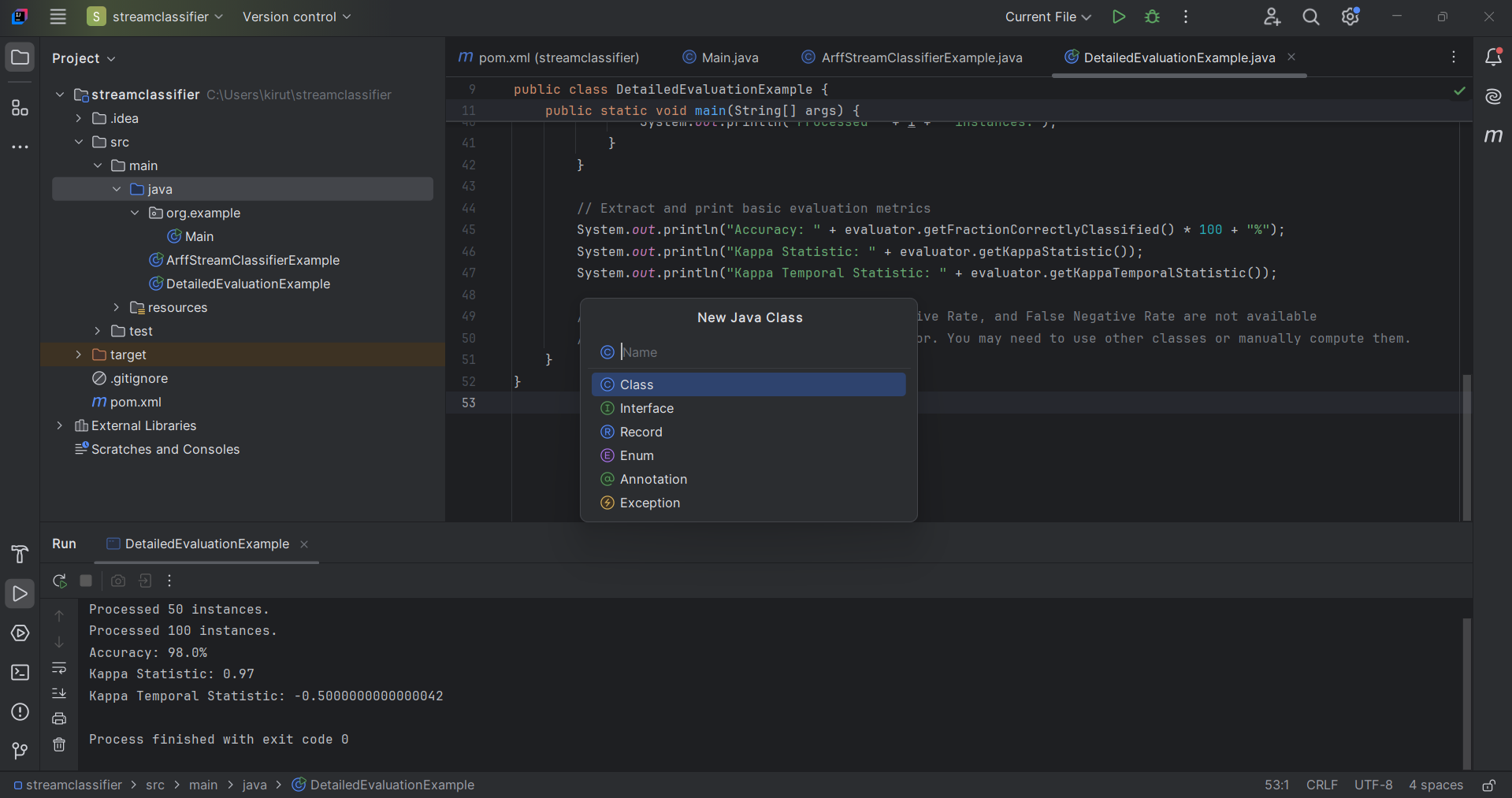
Step2:

Navigate to src\main\java\right click java to create new class\click new

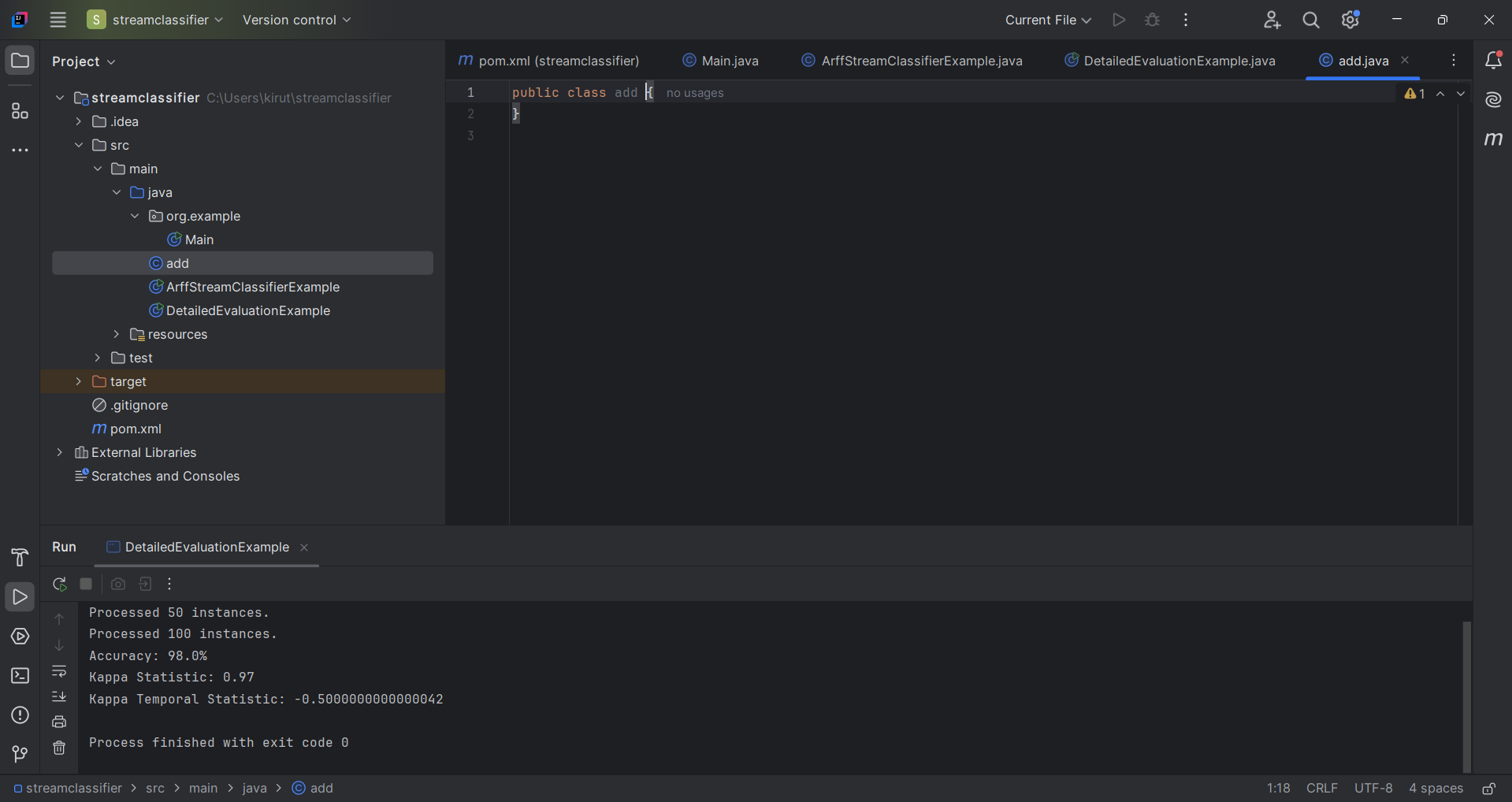
Java class.



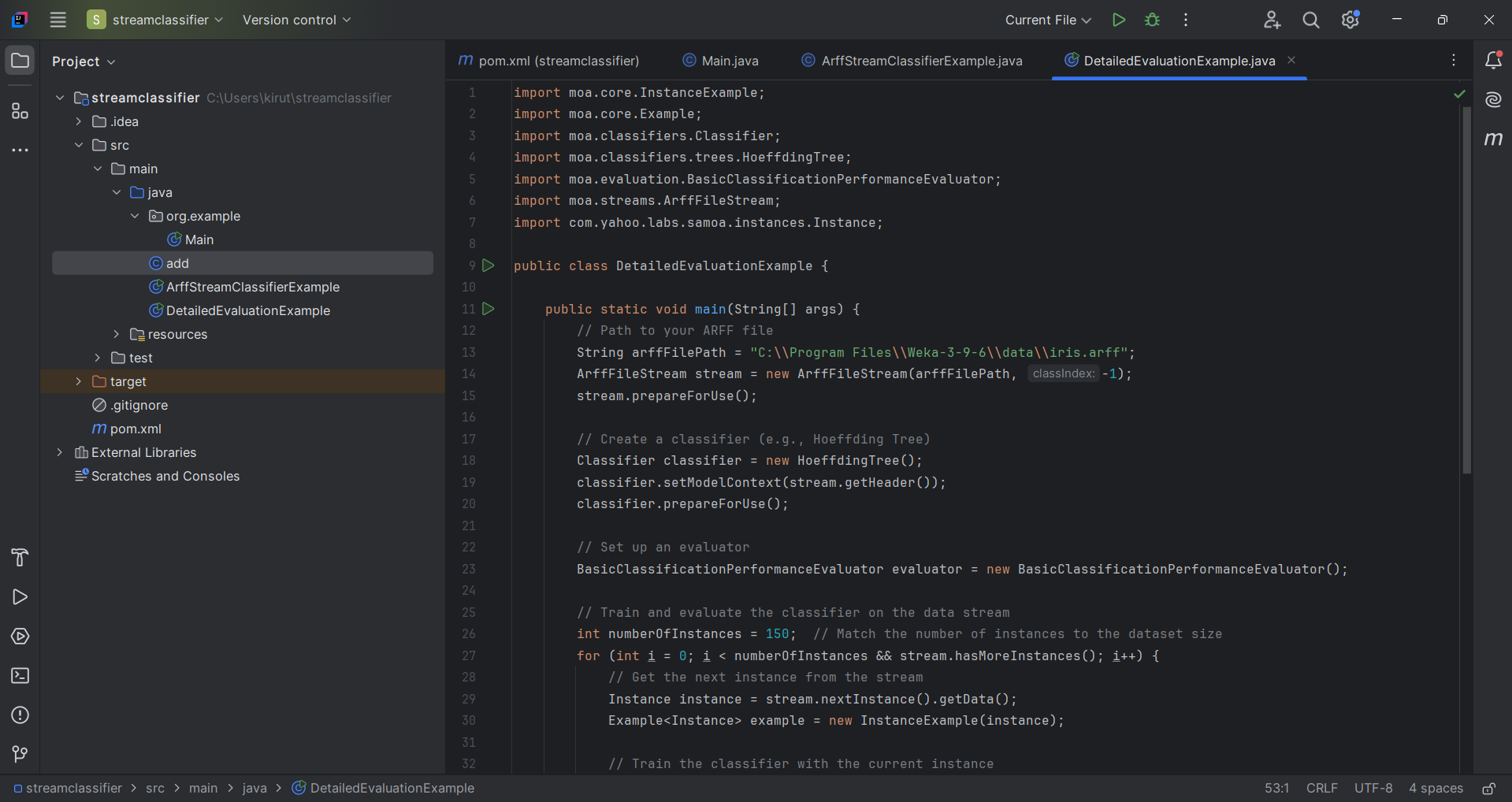
Click java class



Enter your class name then new file is created



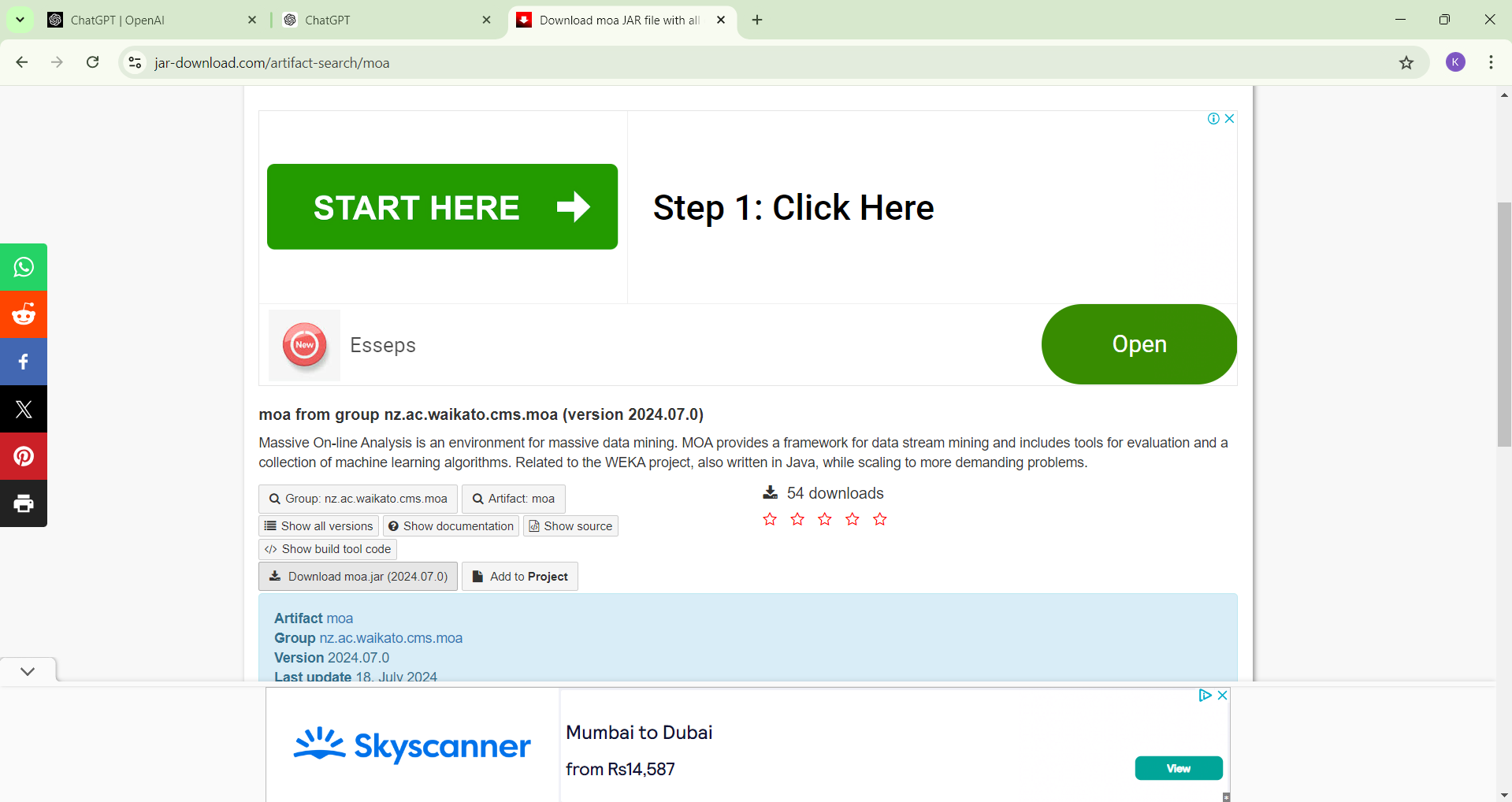
Write the java program here



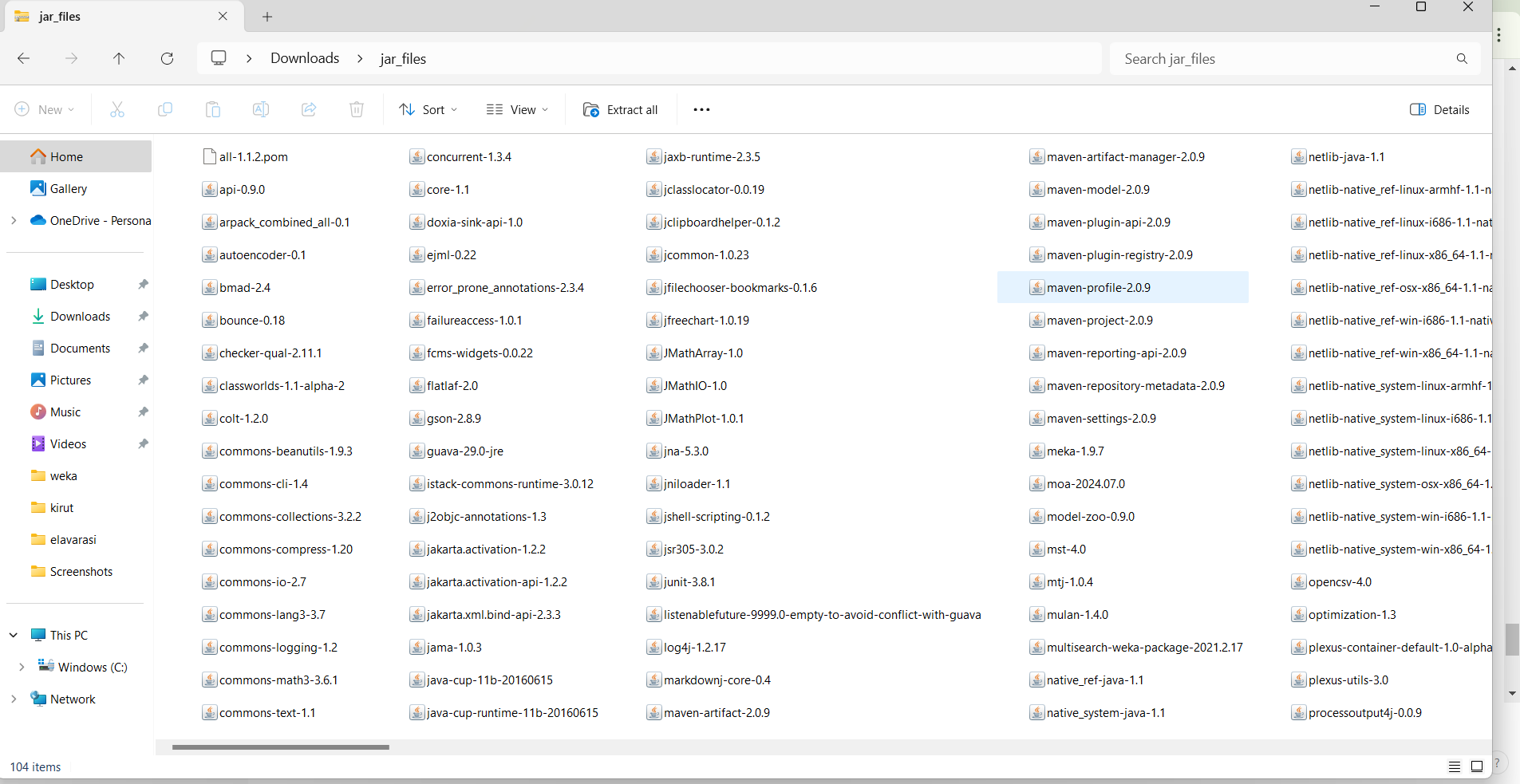
Add your arff file to the program.

Step 3:

Download moa jar file from the moa official websites.



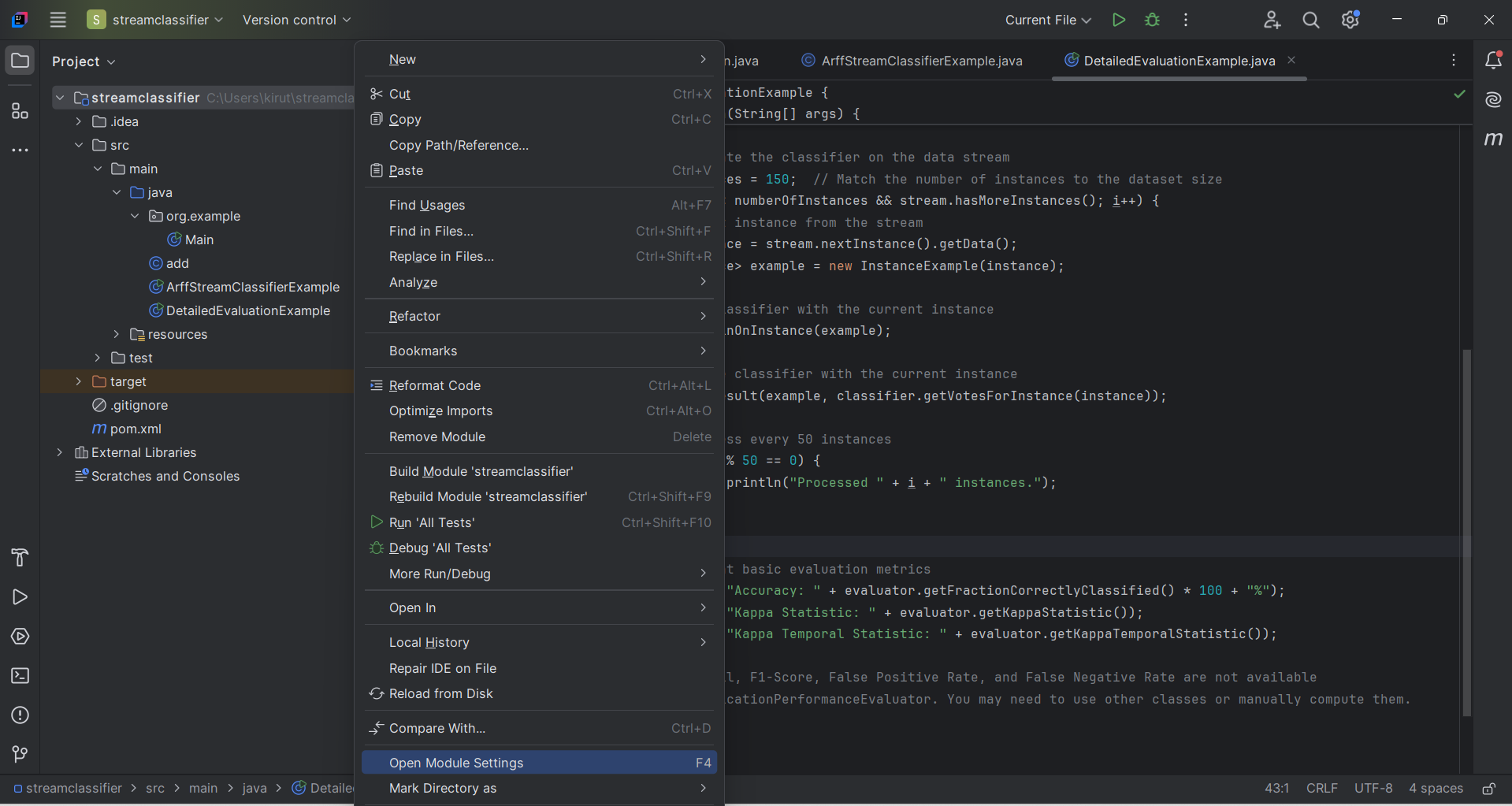
Extract all the jar file.



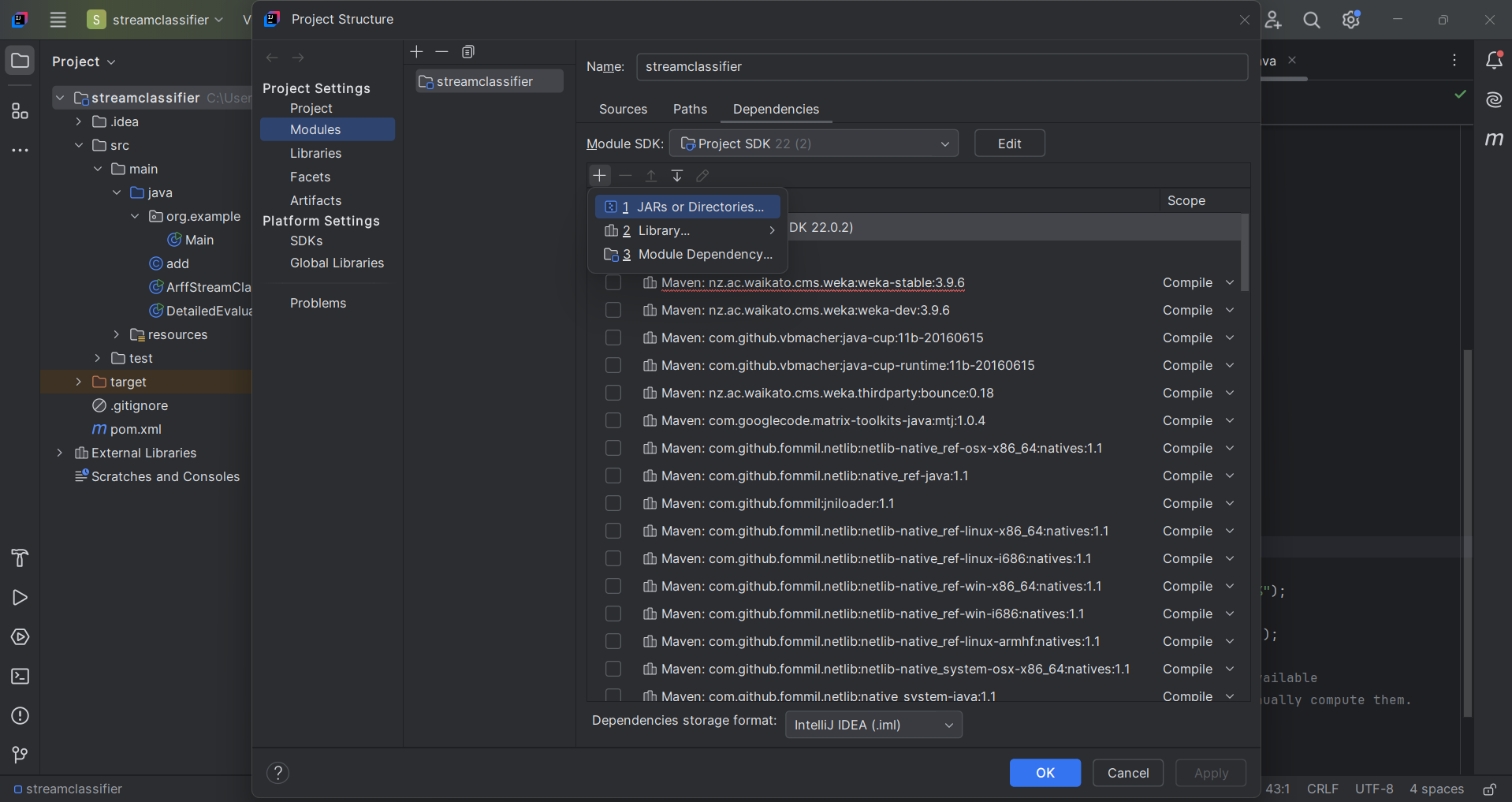
Step 4:

Add the jar file to project.

Right click project\click open module settings.

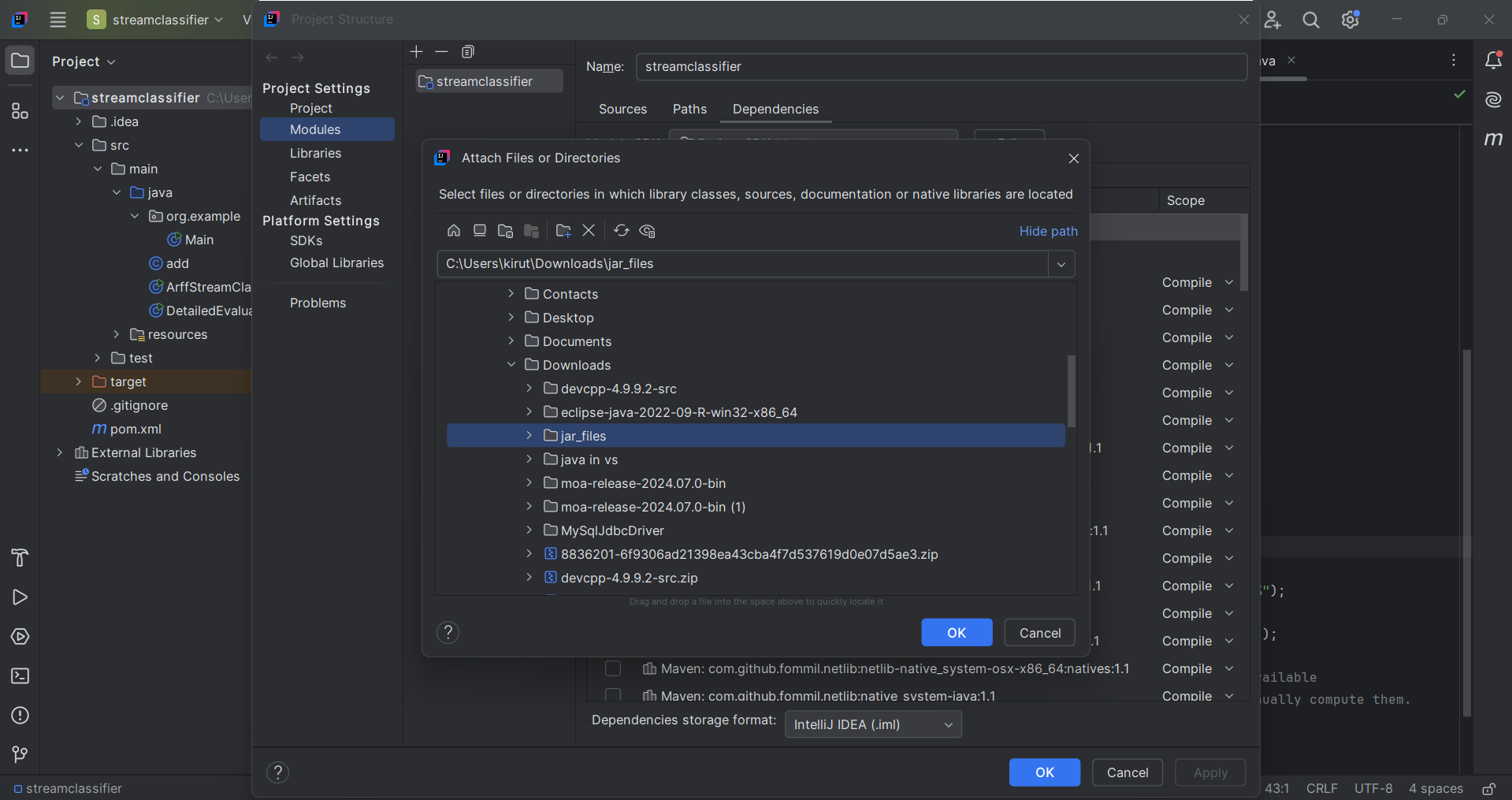


Click on “+” to add jar

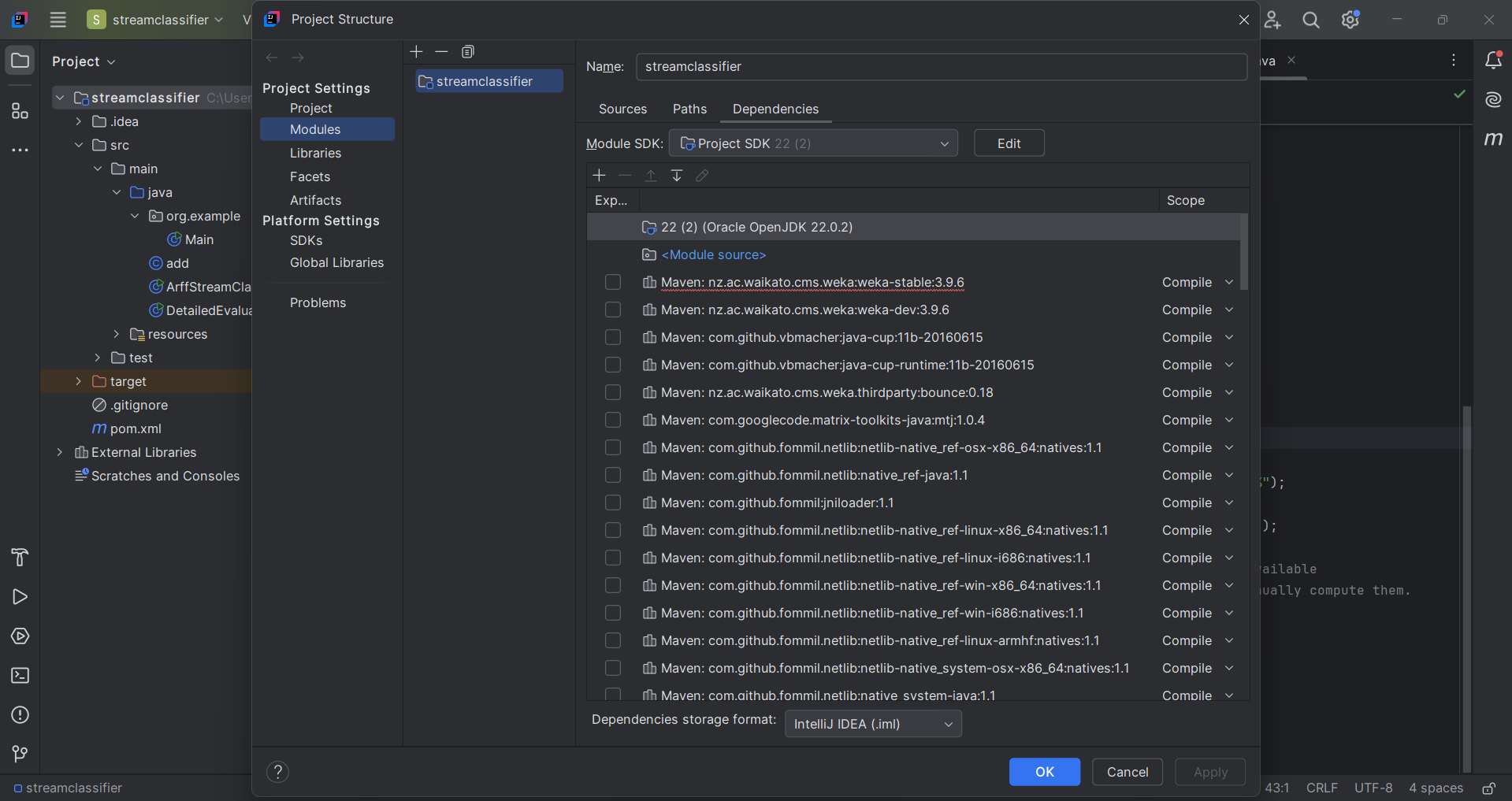


Click jars or directories and navigate to the location where the jar file is stored.

Click ok.



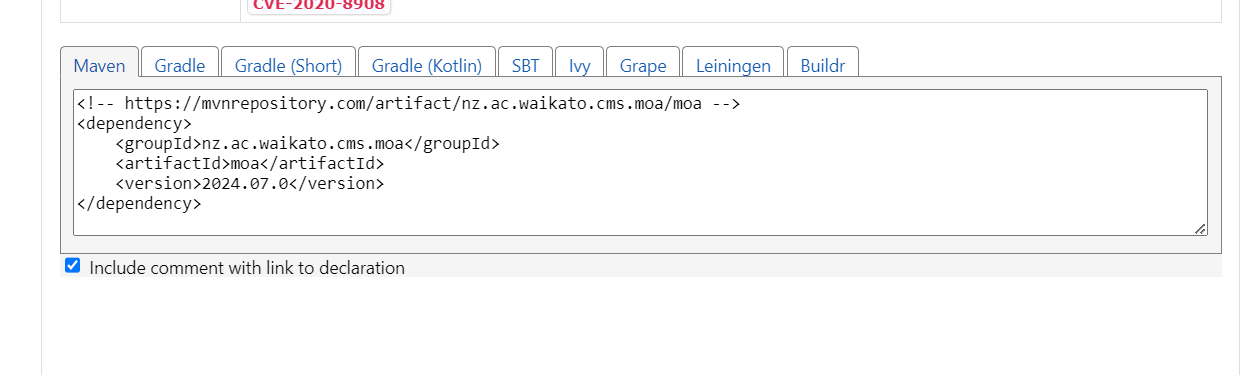
Click apply and ok. The moa.jar file is added to project.



Step 4:

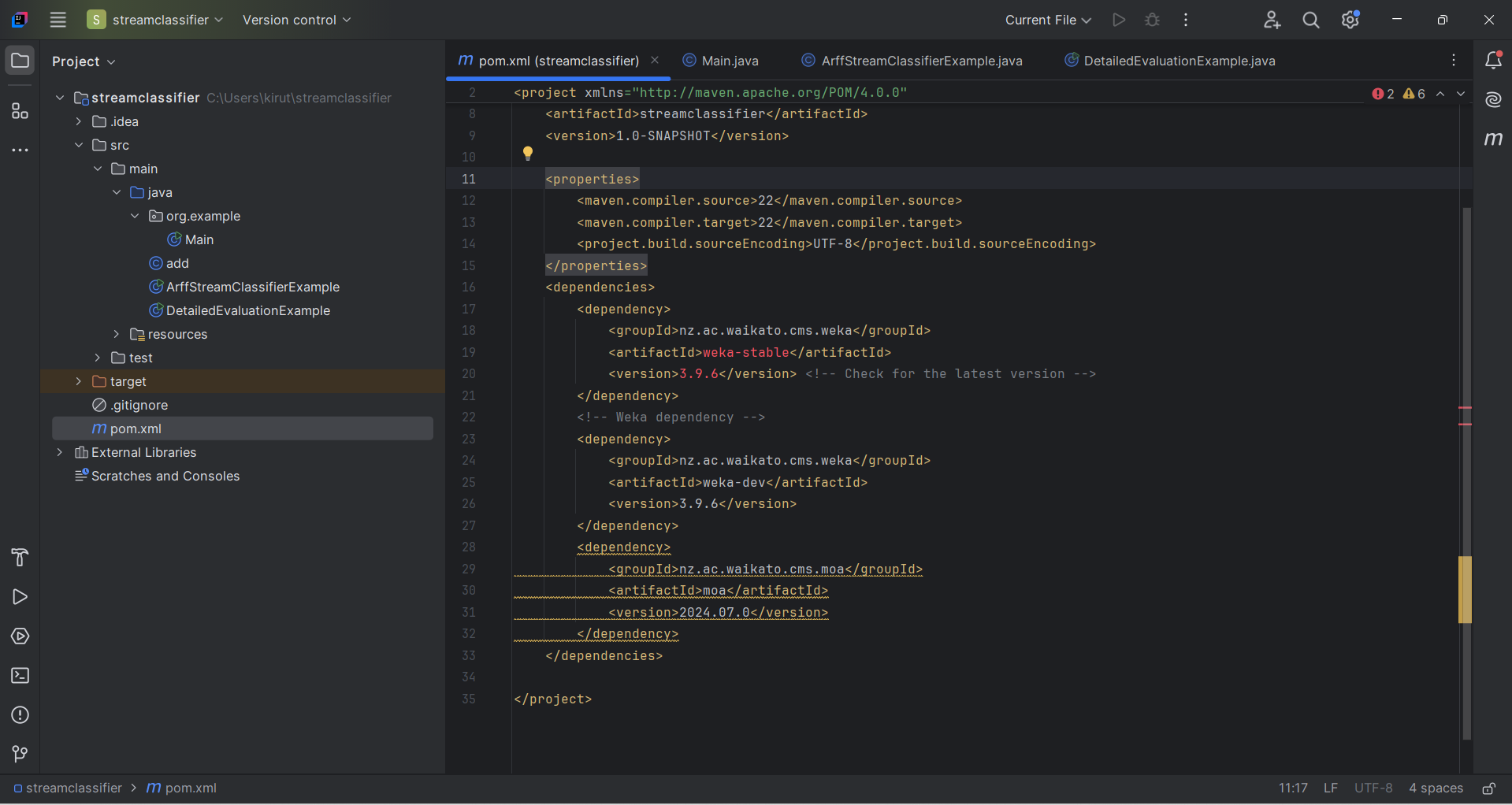
Add dependencies to the project.

Get the dependencies from the moa official website for moa version you have downloaded.



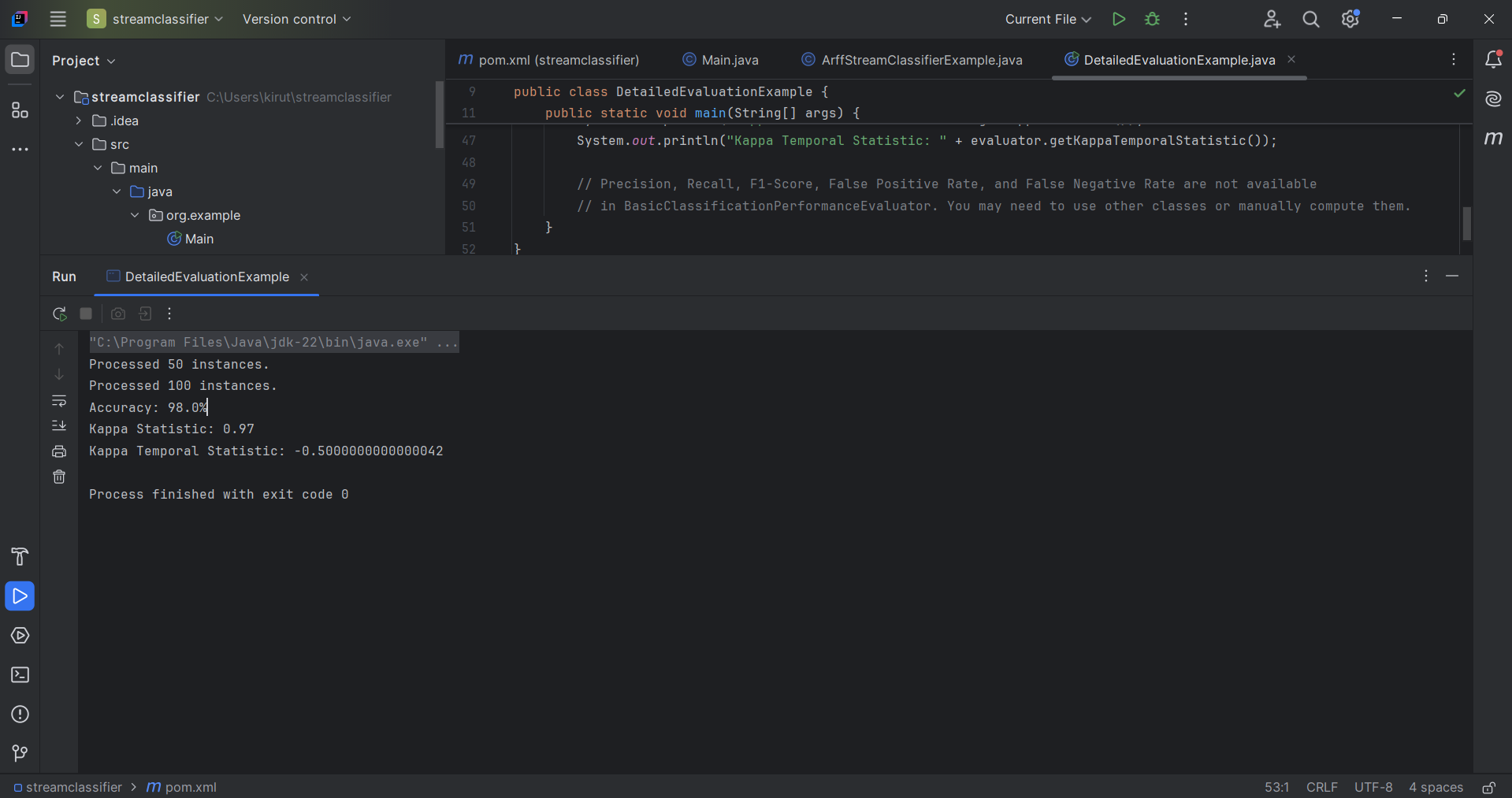
To add dependencies.

Click pom.xml\add dependencies to the project.



Now the run the program.

Output:



Program:

import moa.core.InstanceExample;  
import moa.core.Example;  
import moa.classifiers.Classifier;  
import moa.classifiers.trees.HoeffdingTree;  
import moa.evaluation.BasicClassificationPerformanceEvaluator;  
import moa.streams.ArffFileStream;  
import com.yahoo.labs.samoa.instances.Instance;  
  
public class DetailedEvaluationExample {  
  
 public static void main(String[] args) {  
 // Path to your ARFF file  
 String arffFilePath = "C:\\Program Files\\Weka-3-9-6\\data\\iris.arff";  
 ArffFileStream stream = new ArffFileStream(arffFilePath, -1);  
 stream.prepareForUse();  
  
 // Create a classifier (e.g., Hoeffding Tree)  
 Classifier classifier = new HoeffdingTree();  
 classifier.setModelContext(stream.getHeader());  
 classifier.prepareForUse();  
  
 // Set up an evaluator  
 BasicClassificationPerformanceEvaluator evaluator = new BasicClassificationPerformanceEvaluator();  
  
 // Train and evaluate the classifier on the data stream  
 int numberOfInstances = 150; // Match the number of instances to the dataset size  
 for (int i = 0; i < numberOfInstances && stream.hasMoreInstances(); i++) {  
 // Get the next instance from the stream  
 Instance instance = stream.nextInstance().getData();  
 Example<Instance> example = new InstanceExample(instance);  
  
 // Train the classifier with the current instance  
 classifier.trainOnInstance(example);  
  
 // Evaluate the classifier with the current instance  
 evaluator.addResult(example, classifier.getVotesForInstance(instance));  
  
 // Print progress every 50 instances  
 if (i > 0 && i % 50 == 0) {  
 System.*out*.println("Processed " + i + " instances.");  
 }  
 }  
  
 // Extract and print basic evaluation metrics  
 System.*out*.println("Accuracy: " + evaluator.getFractionCorrectlyClassified() \* 100 + "%");  
 System.*out*.println("Kappa Statistic: " + evaluator.getKappaStatistic());  
 System.*out*.println("Kappa Temporal Statistic: " + evaluator.getKappaTemporalStatistic());  
  
 // Precision, Recall, F1-Score, False Positive Rate, and False Negative Rate are not available  
 // in BasicClassificationPerformanceEvaluator. You may need to use other classes or manually compute them.  
 }  
}