EKONOMICKÁ UNIVERZITA

Fakulta hospodárskej informatiky

Katedra aplikovanej informatiky

Map Reduce príklady

**Predmet:** Big Data

**Meno študentov:** Martin Jankech, Patrik Hajdučík

**Stupeň/ročník:** druhý/2.ročník

**Školský rok:** 2022/2023

**Meno prednášajúceho:** Ing. Jaroslav Kultán PhD.

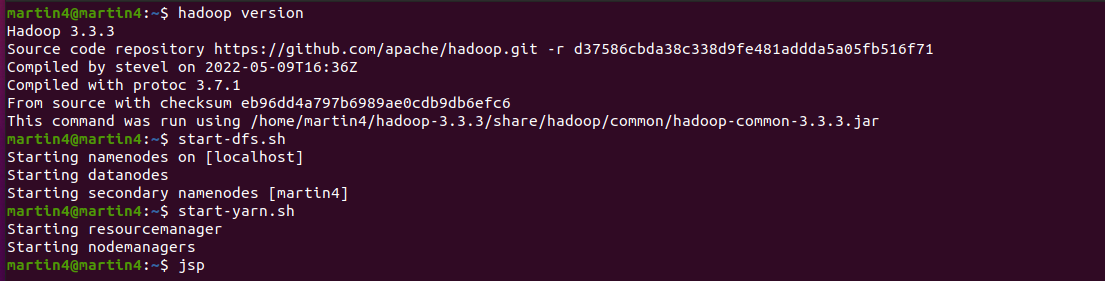
**Meno cvičiaceho:** SCHMIDT, Peter, Ing. Mgr., PhD.

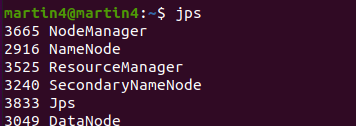
# Úvod

V tomto zadaní si predstavíme 2 map reduce príklady vytvorené v programovacom jazyku Java. Ako vývojové prostredie sme využili InteliJ a ako operačný systém sme použili Linux Distribúciu Ubuntu 20.04

# WordCount-cez príkazový riadok

Tento program implementuje Hadoop MapReduce aplikáciu, ktorá vracia početnosť jednotlivých slov. Ako vstup programu bol použitý textový súbor s dielom Romeo a Julia. Program tokenizuje tento text, čím rozdelí slová a vypočíta výskyt každého slova. Výsledkom programu sú dvojice kľúč-hodnota, kde kľúčom je slovo a hodnotou počet výskytov tohto slova. Výstup sa ukladá do súboru. Program používa triedy z Hadoop MapReduce knižnice a využíva konfiguráciu systému Hadoop pre určenie vstupného a výstupného adresára.

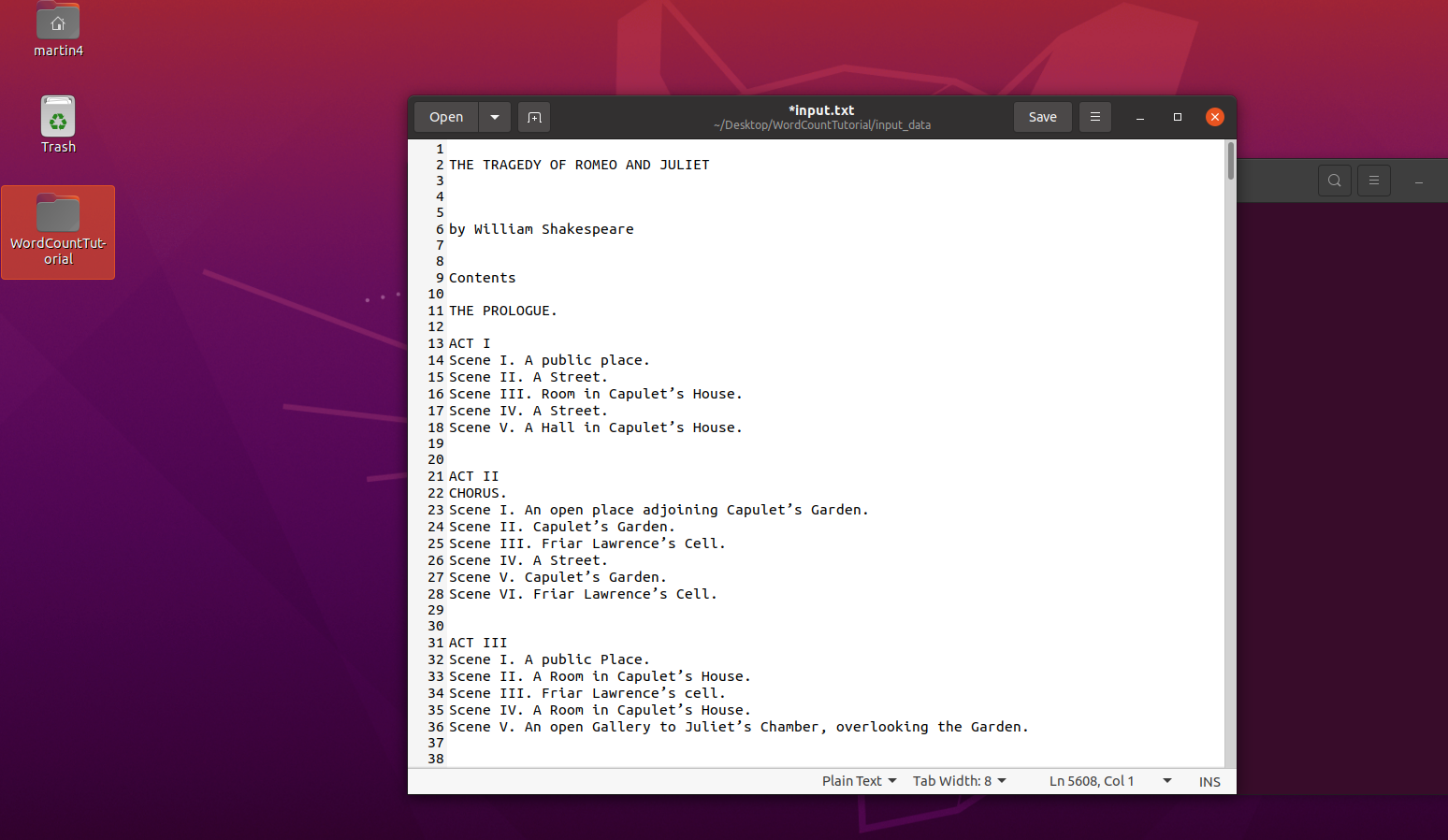
1. Najprv sa uistíme že máme nainštalovaný a spustený Hadoop



1. Uistíme sa že nám funguje java



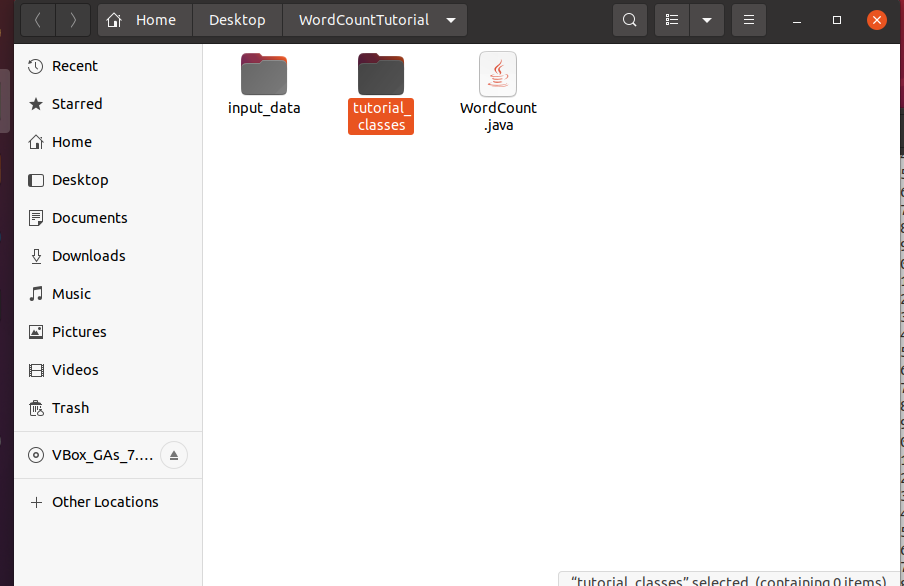
1. Vytvoríme si na ploche alebo ľubovoľnom mieste priečinok WordCountTutorial
2. V tomto priečinku vytvoríme ďalší prečinoch s názvom input-data
3. V priečinku input-data vytvoríme textový dokument s názvom input.txt do ktorého vložíme text na ktorom budeme robiť map reduce. My sme si zvoli textové dielo Romeo a Julia.



1. Pridáme Java súbor s map reduce kódom, ktorý bol získaný z Apache dokumentácie

import java.io.IOException;  
import java.util.StringTokenizer;  
  
import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Job;  
import org.apache.hadoop.mapreduce.Mapper;  
import org.apache.hadoop.mapreduce.Reducer;  
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
  
public class WordCount {  
  
 public static class TokenizerMapper  
 extends Mapper<Object, Text, Text, IntWritable>{  
  
 private final static IntWritable one = new IntWritable(1);  
 private Text word = new Text();  
  
 public void map(Object key, Text value, Context context  
 ) throws IOException, InterruptedException {  
 StringTokenizer itr = new StringTokenizer(value.toString());  
 while (itr.hasMoreTokens()) {  
 word.set(itr.nextToken());  
 context.write(word, one);  
 }  
 }  
 }  
  
 public static class IntSumReducer  
 extends Reducer<Text,IntWritable,Text,IntWritable> {  
 private IntWritable result = new IntWritable();  
  
 public void reduce(Text key, Iterable<IntWritable> values,  
 Context context  
 ) throws IOException, InterruptedException {  
 int sum = 0;  
 for (IntWritable val : values) {  
 sum += val.get();  
 }  
 result.set(sum);  
 context.write(key, result);  
 }  
 }  
  
 public static void main(String[] args) throws Exception {  
 Configuration conf = new Configuration();  
 Job job = Job.getInstance(conf, "word count");  
 job.setJarByClass(WordCount.class);  
 job.setMapperClass(TokenizerMapper.class);  
 job.setCombinerClass(IntSumReducer.class);  
 job.setReducerClass(IntSumReducer.class);  
 job.setOutputKeyClass(Text.class);  
 job.setOutputValueClass(IntWritable.class);  
 FileInputFormat.addInputPath(job, new Path(args[0]));  
 FileOutputFormat.setOutputPath(job, new Path(args[1]));  
 System.exit(job.waitForCompletion(true) ? 0 : 1);  
 }  
}

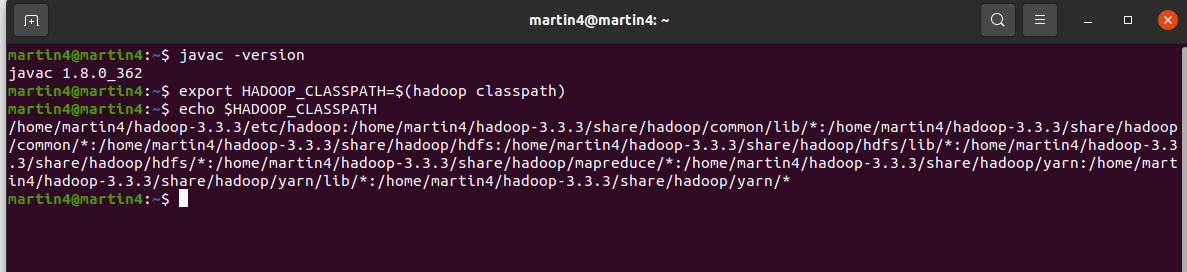
1. V priečinku WordCountTutorial vytvoríme ďalší priečinok tutorial\_classes



1. V ďalšom kroku nastavíme HADOOP\_CLASSPATH premenné prostredia

export HADOOP\_CLASSPATH=$(hadoop classpath)

echo $HADOOP\_CLASSPATH

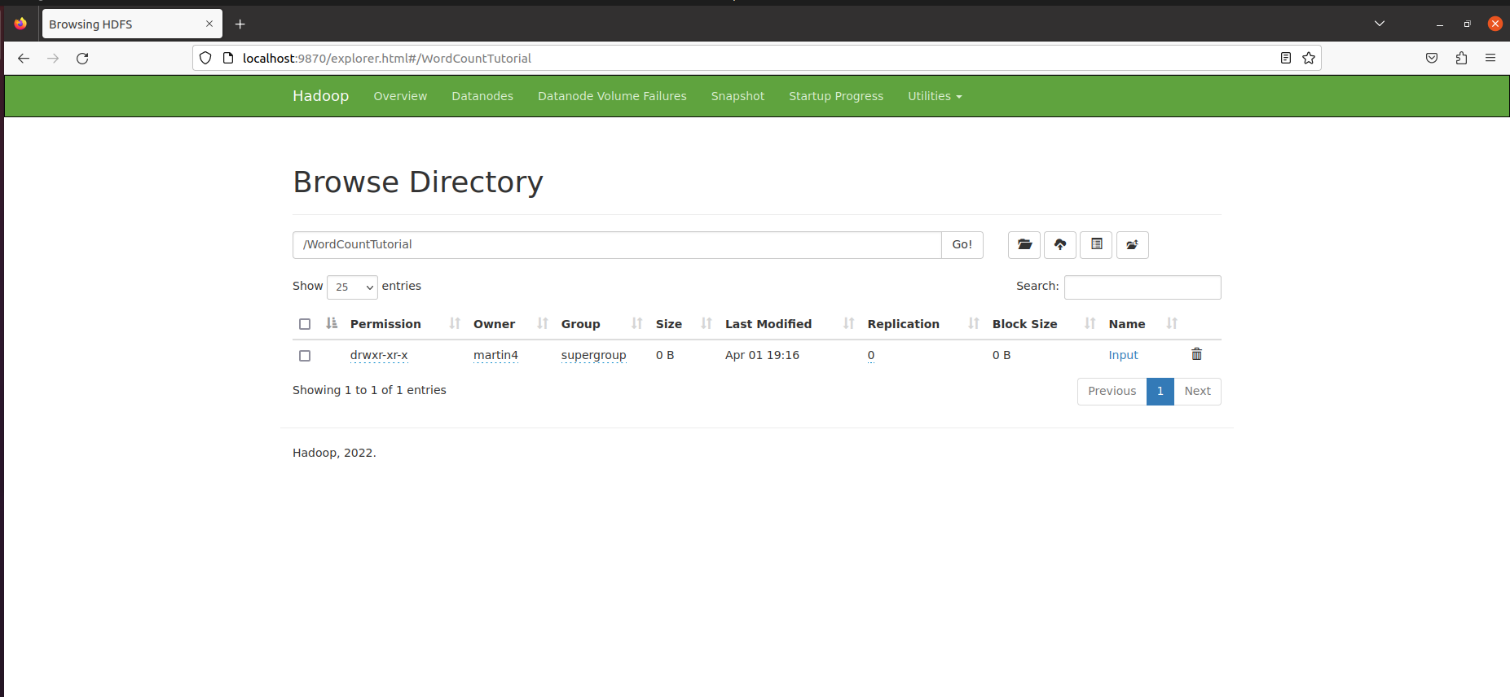


1. Taktiež vytvoríme priečinok v HDFS

hadoop fs -mkdir /WordCountTutorial

1. A taktiež vytvoríme priečinok pre Input

hadoop fs -mkdir /WordCountTutorial/Input

1. Otvoríme localhost:9870 a skontrolujeme či sa nám vytvorili priečinky
2. ďalej pridáme do priečinka HDFS input náš textový dokument input.txt

hadoop fs -put

'/home/martin4/Desktop/WordCountTutorial/input\_data/input.txt' /WordCountTutorial/Input

1. Zmeníme cestu pomocou príkazu cd

cd /home/martin4/Desktop/WordCountTutorial

1. Skompilujeme Java kód

javac -classpath ${HADOOP\_CLASSPATH} -d

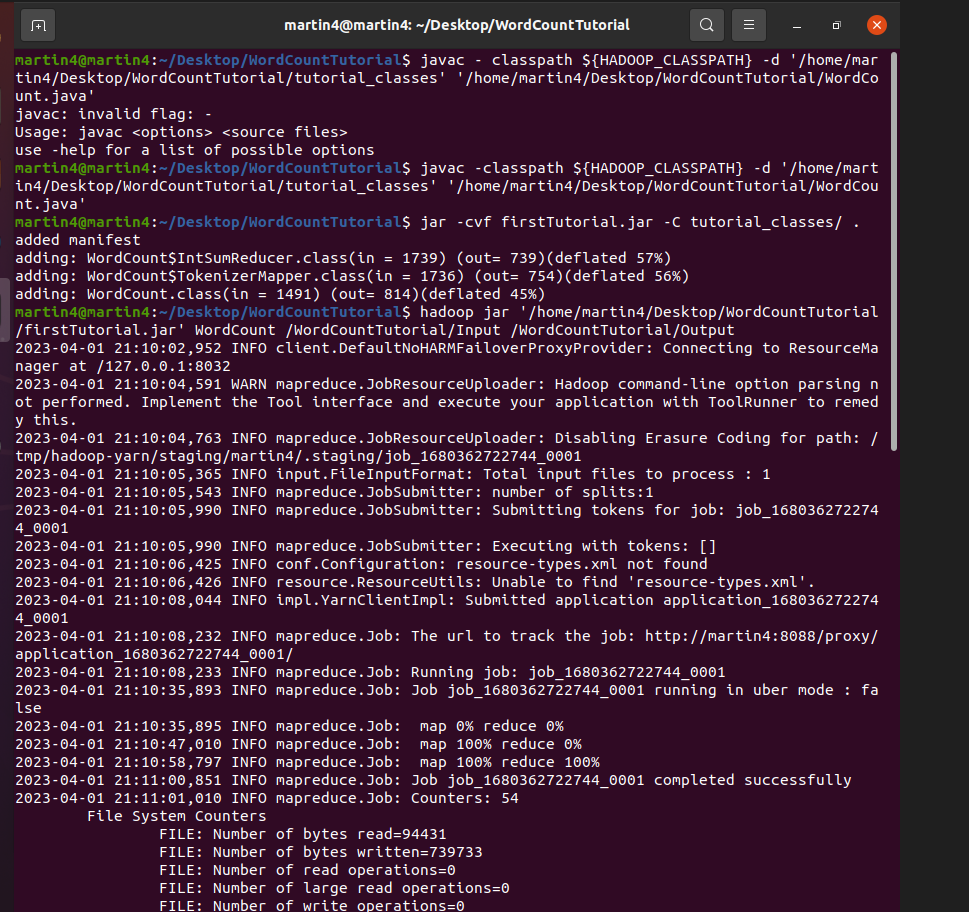
'/home/martin4/Desktop/WordCountTutorial/tutorial\_classes' '/home/martin4/Desktop/WordCountTutorial/WordCount.java'

1. Výstupne súbory vložíme do jedného súboru jar

jar -cvf firstTutorial.jar -C tutorial\_classes/ .

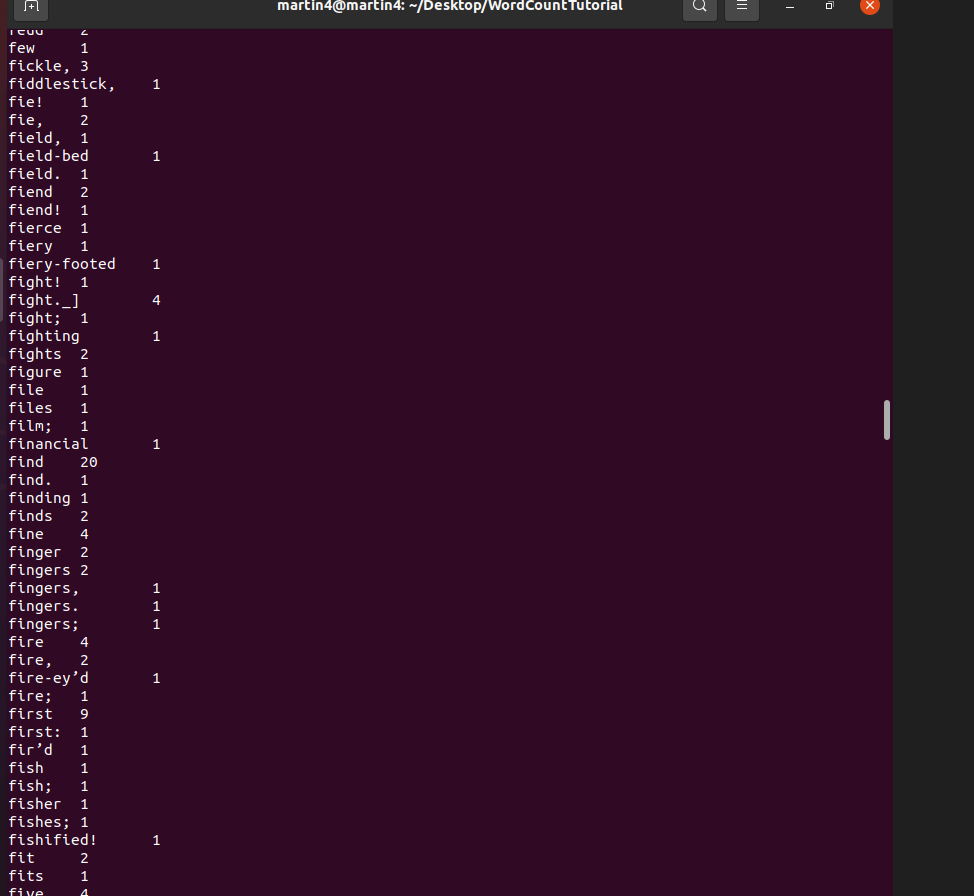
1. Teraz spustime jar na hadoope

hadoop jar '/home/martin4/Desktop/WordCountTutorial/firstTutorial.jar' WordCount /WordCountTutorial/Input /WordCountTutorial/Output



1. Pre zobrazenie výstupu zadáme nasledujúci príkaz

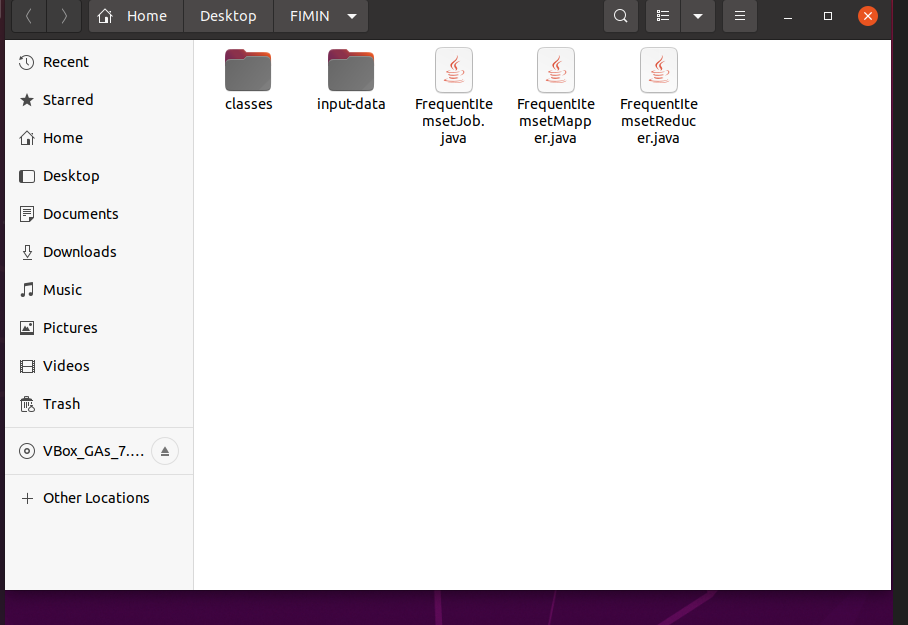
hadoop dfs -cat /WordCountTutorial/Output/\*



Frequent Itemset Mining: cez príkazový riadok

Tento program v jazyku Java s využitím MapReduce implementuje algoritmus na ťaženie frekventovane vyskytujúcich sa položiek. Na základe sady transakcií, ktoré obsahujú zoznam položiek program identifikuje dvojice položiek, ktoré sa často vyskytujú spoločne. Konkrétne program vytvorí všetky dvojice položiek v každej transakcii, ktoré sa spoločne vyskytujú, a potom agreguje počty každej dvojice, aby určil, ktoré dvojice sa často vyskytujú. Program prijíma tri argumenty z príkazového riadku: cestu k vstupnému súboru, cestu k výstupnému súboru a prah podpory. Prah podpory je celé číslo, ktoré určuje minimálny počet transakcií, v ktorých sa musí dvojica vyskytovať, aby bola považovaná za frekventovanú. Výstupom programu je zoznam frekventovaných dvojíc položiek a ich počty.

1. podobne ako v prvom príklade si vytvoríme priečinok so zdrojovými údajmi. My sme ho nazvali FIMIN



1. do priečinka input-data vložíme náš textový dokument s vstupnými údajmi

apple orange grapes

apple mango

orange apple chickoo

apple grapes orange

milk butter

butter cheese

bread milk butter

cheese bread butter milk

cheese apple orange

Taktiež tam vložíme tri Java súbory so zdrojovým kódom

**FrequentItemsetJob.java**

import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Job;  
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
  
public class FrequentItemsetJob {  
  
 public static void main(String[] args) throws Exception {  
 Configuration conf = new Configuration();  
 *//conf.setInt("support", Integer.parseInt(args[2]));* Job job = Job.*getInstance*(conf, "Frequent Itemset Mining");  
 job.setJarByClass(FrequentItemsetJob.class);  
 job.setMapperClass(FrequentItemsetMapper.class);  
 job.setReducerClass(FrequentItemsetReducer.class);  
 job.setOutputKeyClass(Text.class);  
 job.setOutputValueClass(IntWritable.class);  
 FileInputFormat.*addInputPath*(job, new Path(args[0]));  
 FileOutputFormat.*setOutputPath*(job, new Path(args[1]));  
 System.*exit*(job.waitForCompletion(true) ? 0 : 1);  
 }  
}

**FrequentItemsetMapper.java**

import java.io.IOException;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Mapper;  
  
  
public class FrequentItemsetMapper extends Mapper<Object, Text, Text, IntWritable> {  
  
 private static final IntWritable *ONE* = new IntWritable(1);  
  
 @Override  
 public void map(Object key, Text value, Context context) throws IOException, InterruptedException {  
 String[] items = value.toString().split(" ");  
 for (int i = 0; i < items.length; i++) {  
 for (int j = i + 1; j < items.length; j++) {  
 String itemset = items[i] + "," + items[j];  
 context.write(new Text(itemset), *ONE*);  
 }  
 }  
 }  
}

import java.io.IOException;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Reducer;  
public class FrequentItemsetReducer extends Reducer<Text, IntWritable, Text, IntWritable> {  
  
 private int supportThreshold;  
  
 @Override  
 public void setup(Context context) {  
 supportThreshold = context.getConfiguration().getInt("support", 2);  
 }  
  
 @Override  
 public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {  
 int count = 0;  
 for (IntWritable value : values) {  
 count += value.get();  
 }  
 if (count >= supportThreshold) {  
 context.write(key, new IntWritable(count));  
 }  
 }  
}

1. V ďalšom kroku nastavíme HADOOP\_CLASSPATH premenné prostredia

export HADOOP\_CLASSPATH=$(hadoop classpath)

echo $HADOOP\_CLASSPATH

1. Taktiež vytvoríme priečinok v HDFS

hadoop fs -mkdir /FIMIN

1. A taktiež vytvoríme priečinok pre Input

hadoop fs -mkdir /FIMIN/Input

1. ďalej pridáme do priečinka HDFS input náš textový dokument input.txt

hadoop fs -put '/home/martin4/Desktop/FIMIN/input-data/input.txt' /FIMIN/Input

1. Zmeníme cestu pomocou príkazu cd

cd /home/martin4/Desktop/FIMIN

1. Skompilujeme Java kód

javac -classpath ${HADOOP\_CLASSPATH} -d

'/home/martin4/Desktop/FIMIN/classes' '/home/martin4/Desktop/FIMIN/FrequentItemsetJob.java' '/home/martin4/Desktop/FIMIN/FrequentItemsetMapper.java' '/home/martin4/Desktop/FIMIN/FrequentItemsetReducer.java'

1. Výstupné súbory vložíme do jedného súboru jar

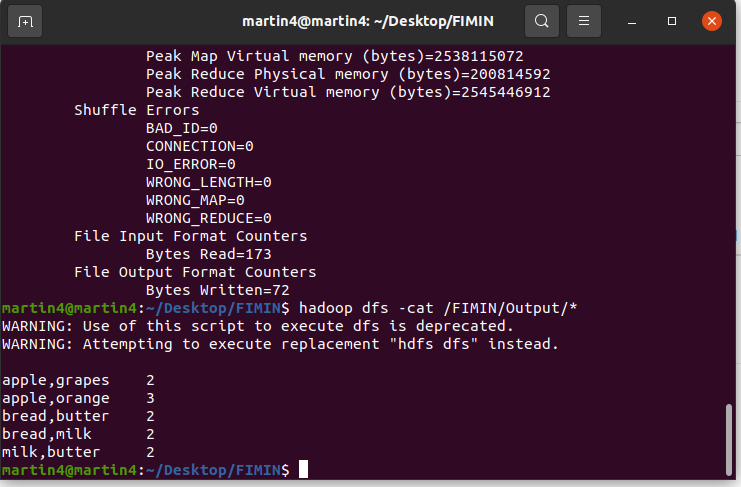
jar -cvf firstFIM.jar -C tutorial\_classes/ .

1. Teraz spustime jar na hadoope

hadoop jar '/home/martin4/Desktop/FIMIN/firstFIM.jar' FrequentItemsetJob /FIMIN/Input /FIMIN/Output

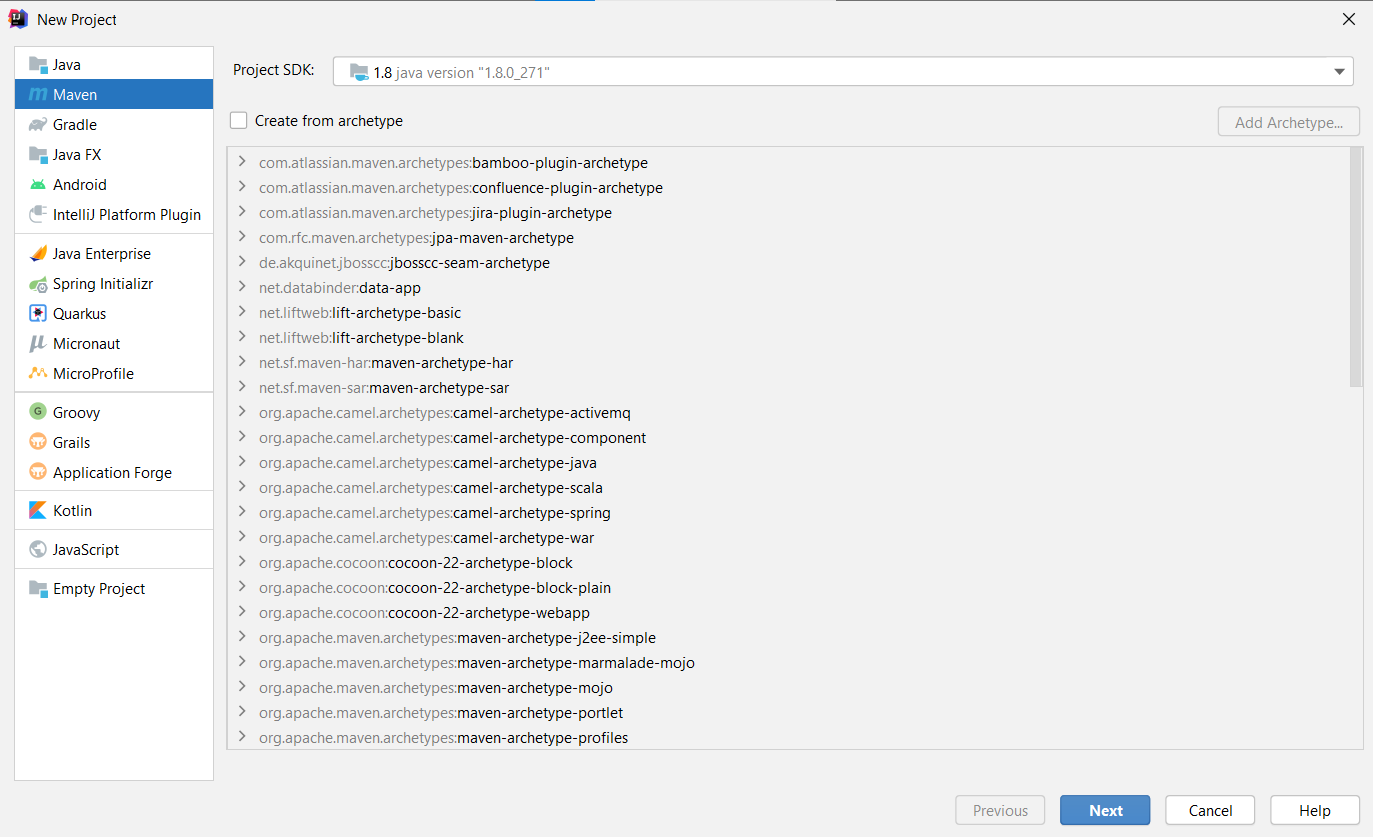
1. Pre zobrazenie výstupu zadáme nasledujúci príkaz

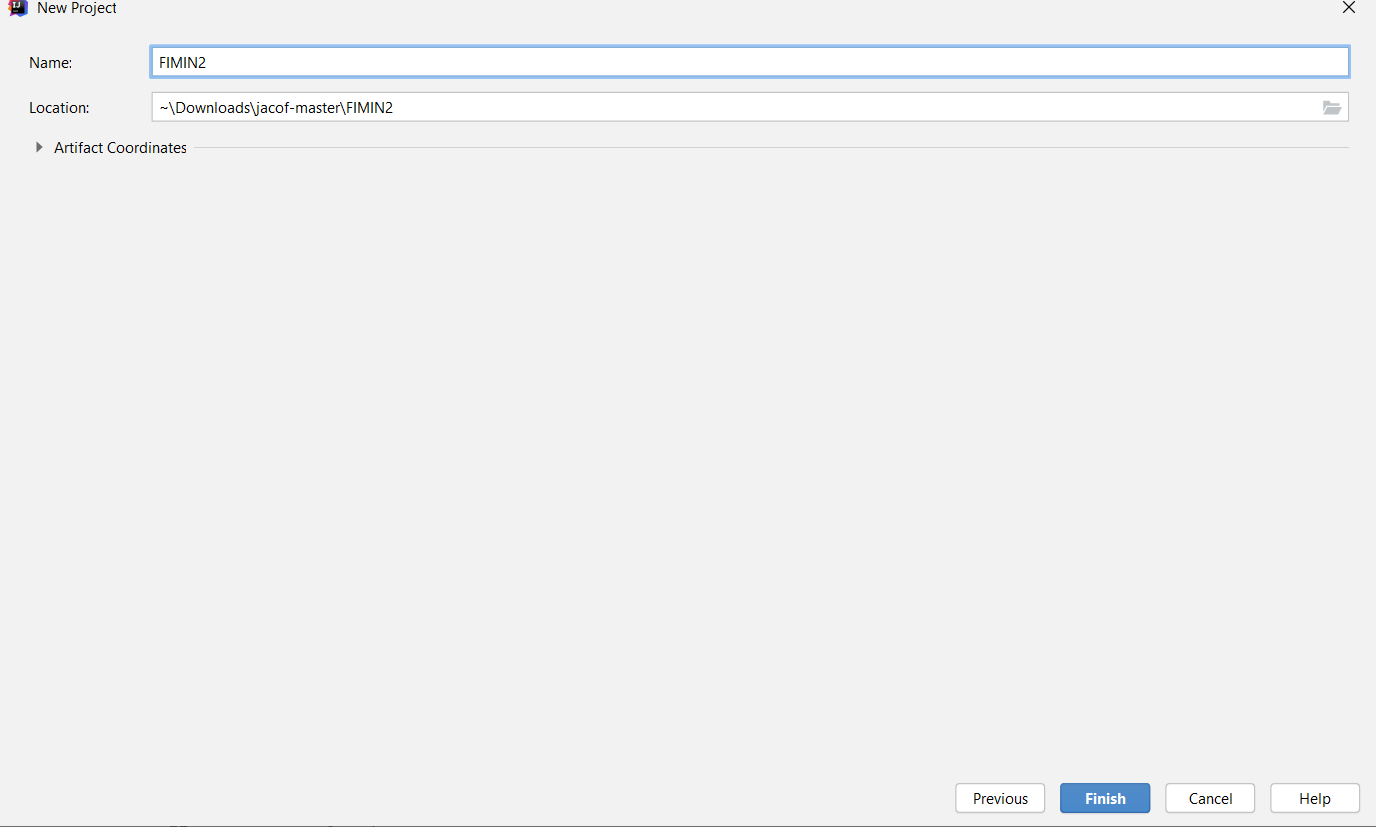
hadoop dfs -cat /FIMIN/Output/\*



Frequent Itemset Mining: v InteliJ

Pre testovacie účely vieme rovnaký program spustiť aj v cez IDE inteliJ

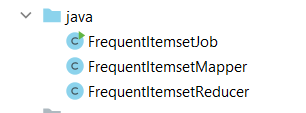
1. Vytvoríme nový maven projekt
2. Dáme mu ľubovoľný názov



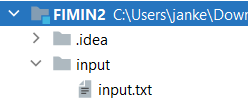
1. Do súboru pom.xml pridáme všetky potrebné depencencie. Po pridaní treba reštartovať cache inteliJ

*<?*xml version="1.0" encoding="UTF-8"*?>*<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>org.example</groupId>  
 <artifactId>FIMIN2</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <repositories>  
 <repository>  
 <id>apache</id>  
 <url>http://maven.apache.org</url>  
 </repository>  
 </repositories>  
 <dependencies>  
 <dependency>  
 <groupId>org.apache.hadoop</groupId>  
 <artifactId>hadoop-client</artifactId>  
 <version>3.2.2</version>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.hadoop</groupId>  
 <artifactId>hadoop-common</artifactId>  
 <version>3.2.1</version>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.hadoop</groupId>  
 <artifactId>hadoop-core</artifactId>  
 <version>1.2.1</version>  
 <scope>provided</scope>  
 </dependency>  
 </dependencies>  
  
  
</project>

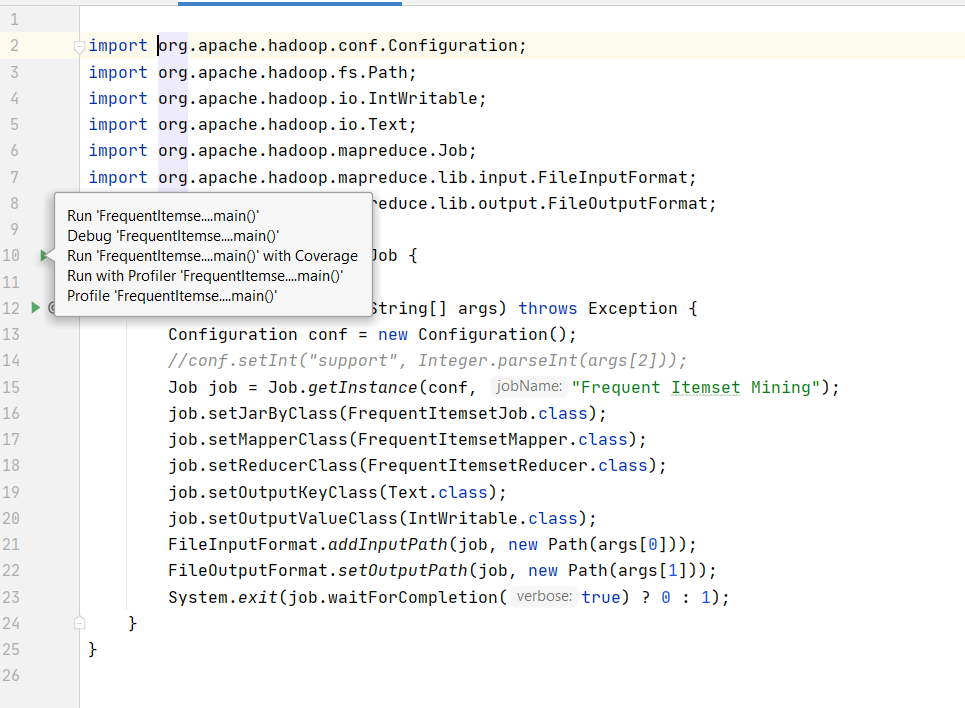
1. Pridáme Java súbory so zdrojovým kódom(kód rovnaký ako cez príkazový riadok)



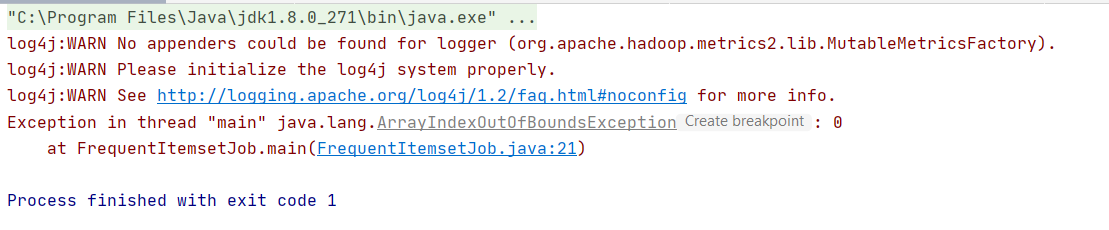
1. Pridáme vstupný súbor (dáta rovnaké ako cez príkazový riadok)



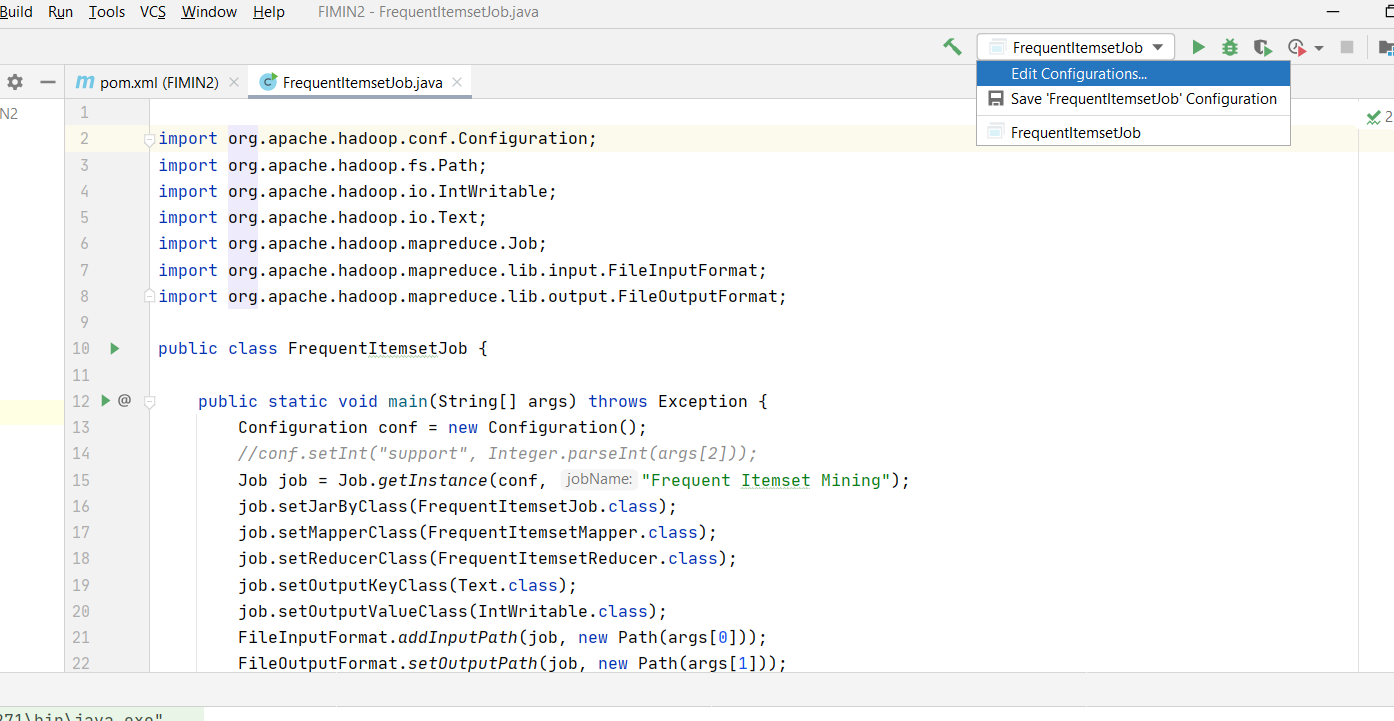
1. Spustíme program

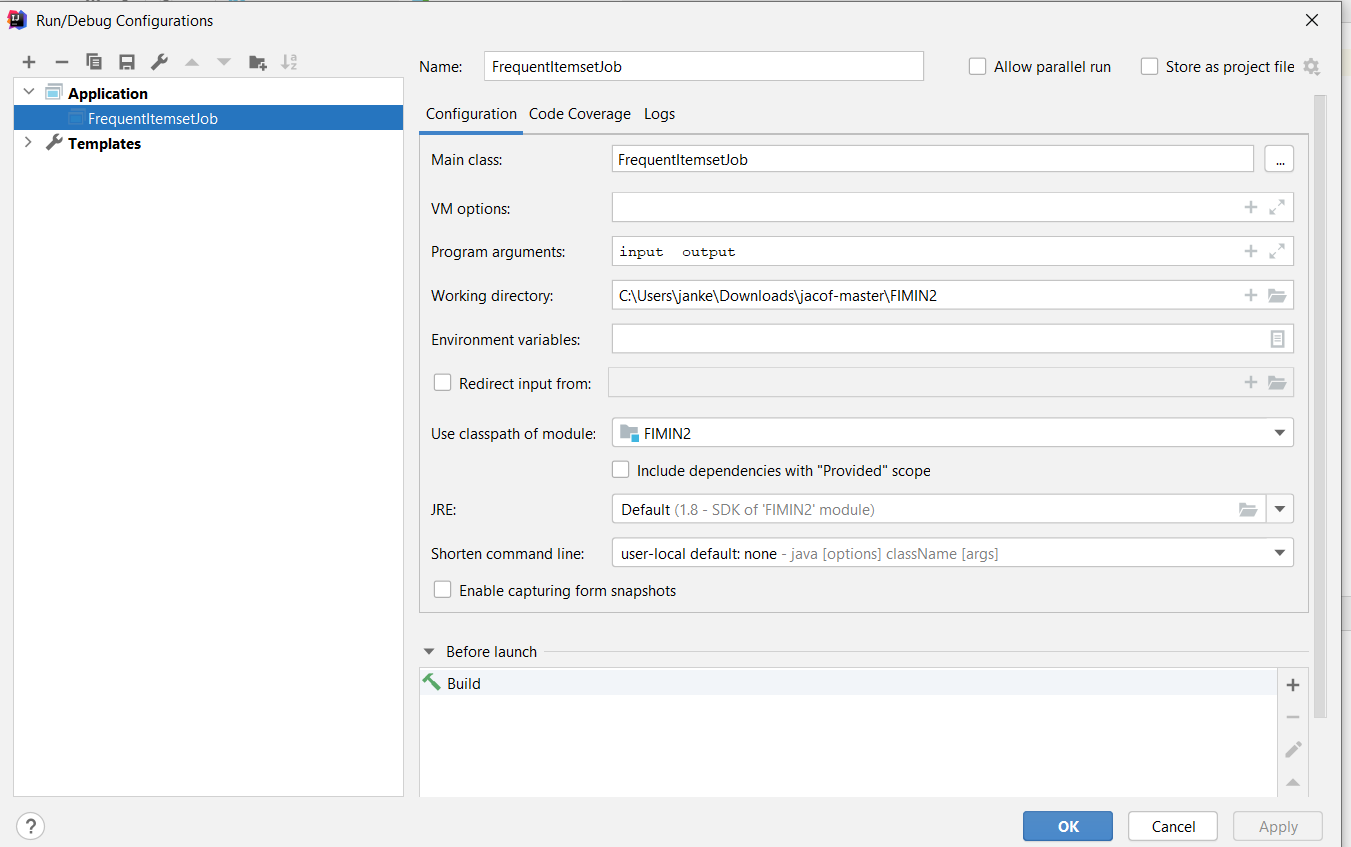


Dostaneme nasledovnú chybu

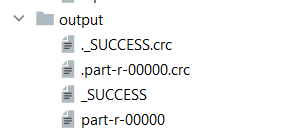


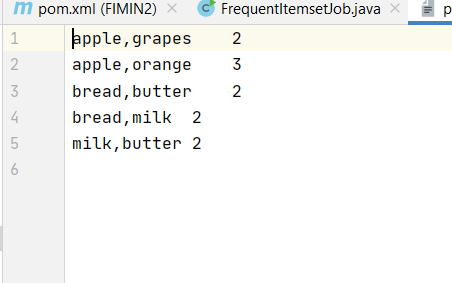
1. Je potreba ešte nastaviť programové premenne pre input a output





1. Program spustíme znova a mali by sme už dostať výstup





Podobný postup sa dá aplikovať aj na WordCount.

# Záver

V tomto zadaní sme si ukázali 2 map reduce programy v Jave. Ukázali sme že existujú viacero spôsobov ako tieto programy spojazdniť.