**Midterm Code**

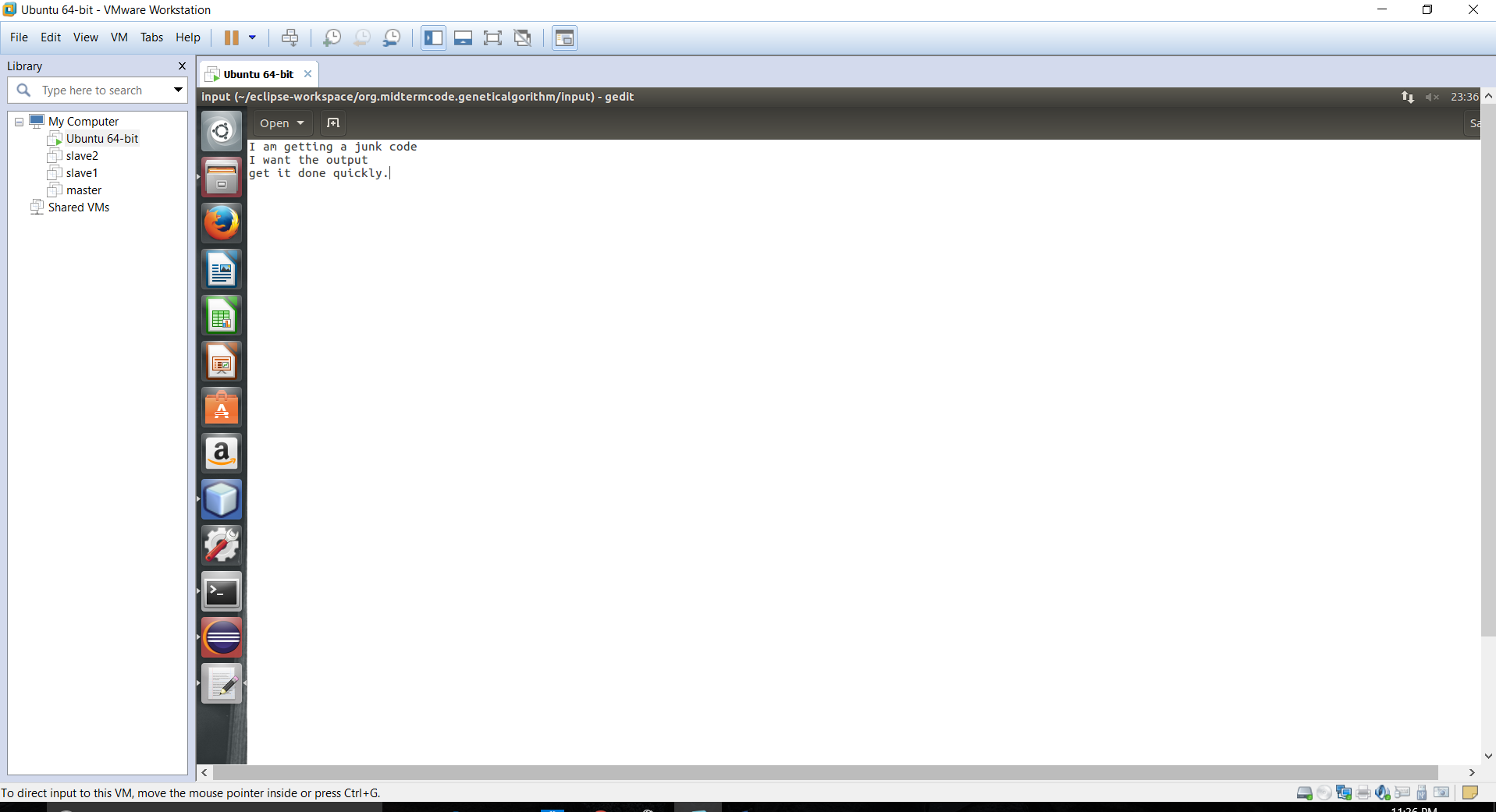
Mapreduce using genetic Algorithm:

**Mapper phase:**

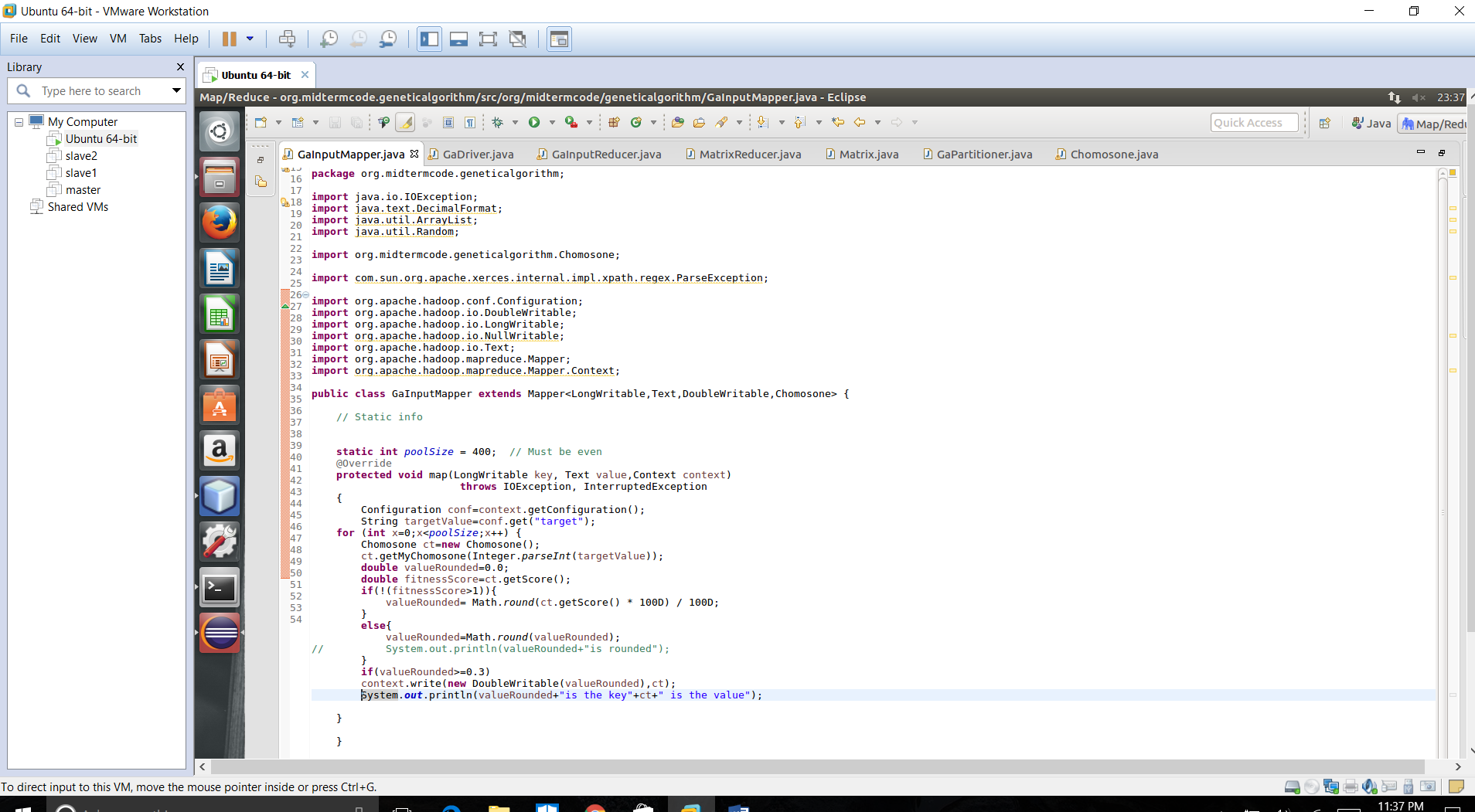
I am generating chromosomes in my mapper, calculating the fitness score compared using the target value.

**Map Input** : A document with few records.( Not relevant to the program, but just to trigger the map function).

**Input file Screenshot:**



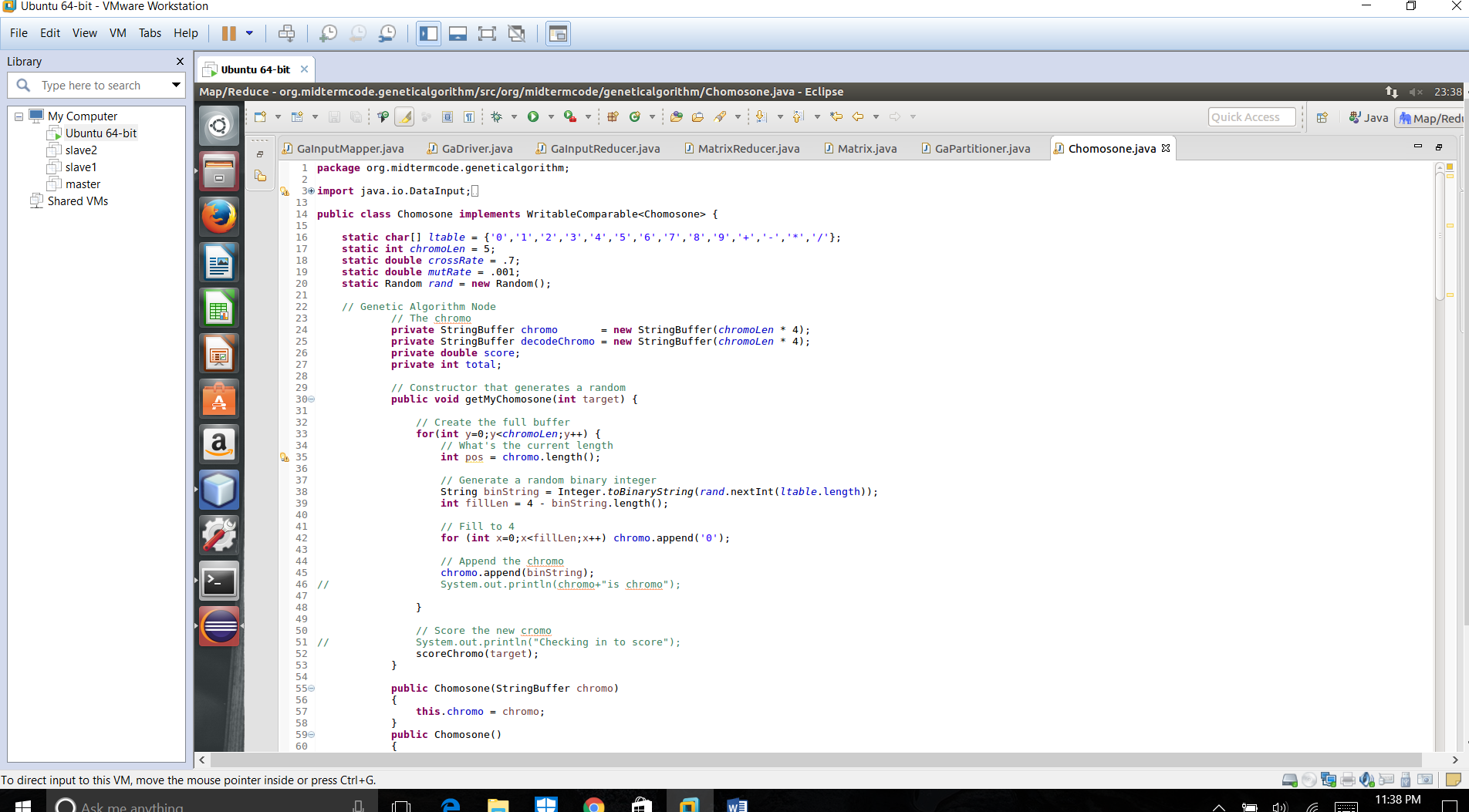
**Mapper code:**

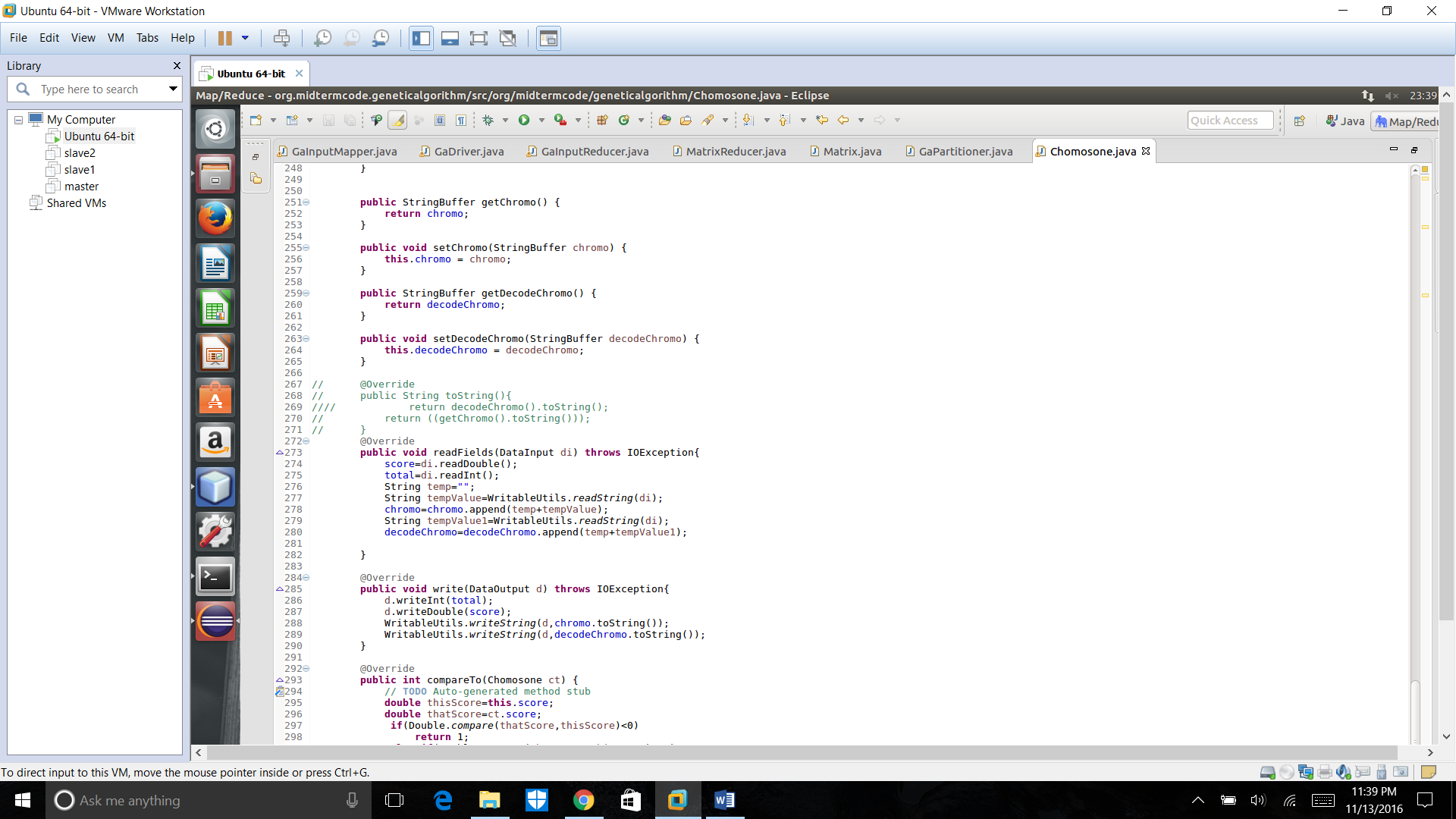


**Map Outputs: Fitness value of a chromosome as Key, Chromosome object as Value.**

The Chromosome class is made to implement a Writable interface to emit as a value. (Initially, I was trying to emit chromosome as a key and so implemented a WritableComparable interface).

**Chromosome code**:





Any value emitted from mapper should be implementing Writable interface so that it could be converted to object of a type(IntWritable,LongWritable or customWritable) in Hadoop. As Hadoop deserializes all the values of type and serializes it for efficient operations when working in a cluster.( And moreover, deserialization and serialization consumes more time and very slow in java.)

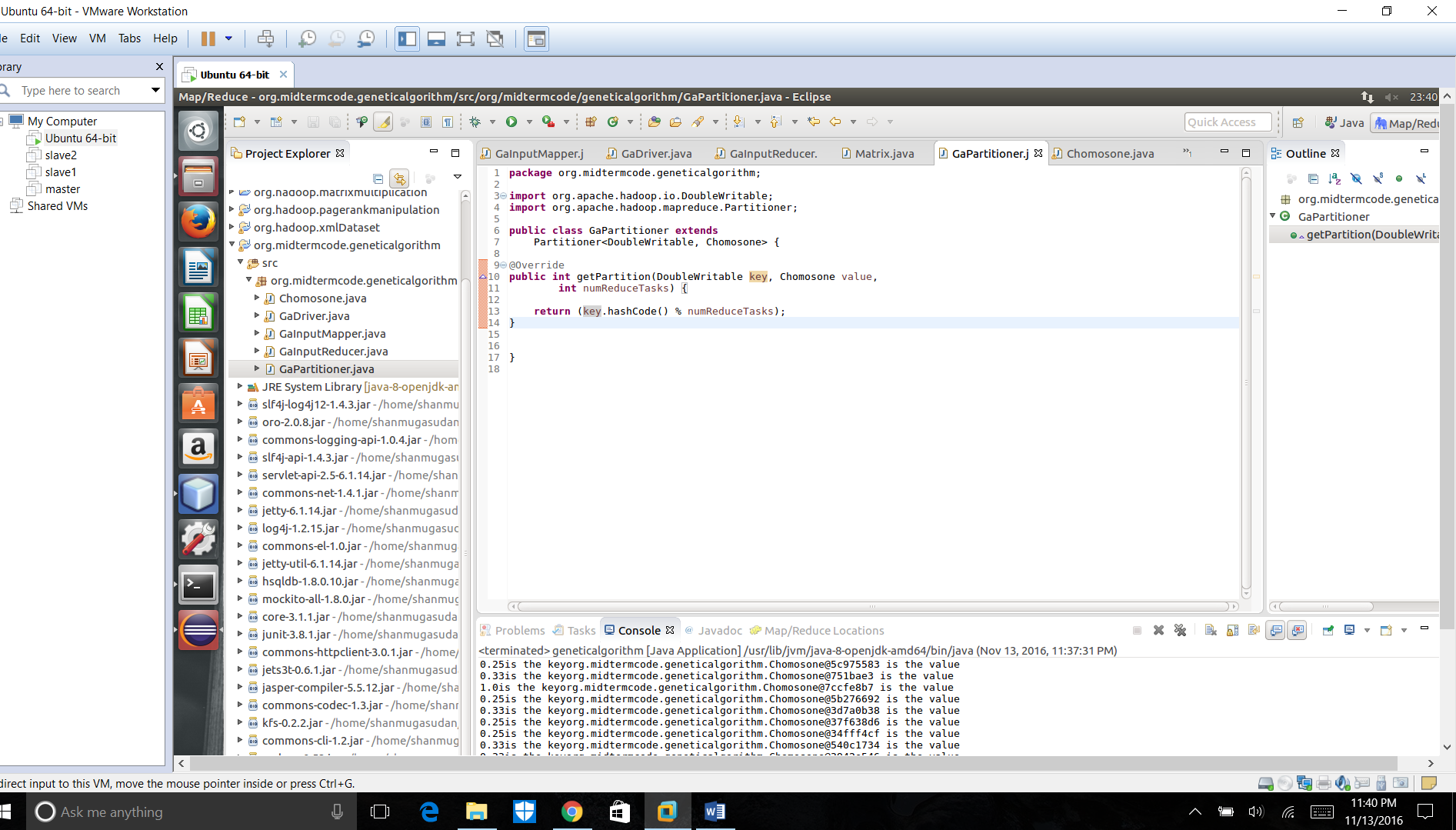
By emitting the chromosome tuple, I could access all the attributes and methods safely and efficiently.

I **filter out** the values whose **fitness score is less than 0.3**, so that I consider only the better fit chromosomes for my crossover and mutation.

**Partitioner**:

I have a Partitioner to make sure I send the values with **same key are sent to a particular reducer**. This way, the most fit chromosomes are grouped to a particular reducer. This could help achieve the fitness factor (convergence factor) to reach 1 quickly, since we breed against a good grade of chromosomes.

**Partitioner code:**



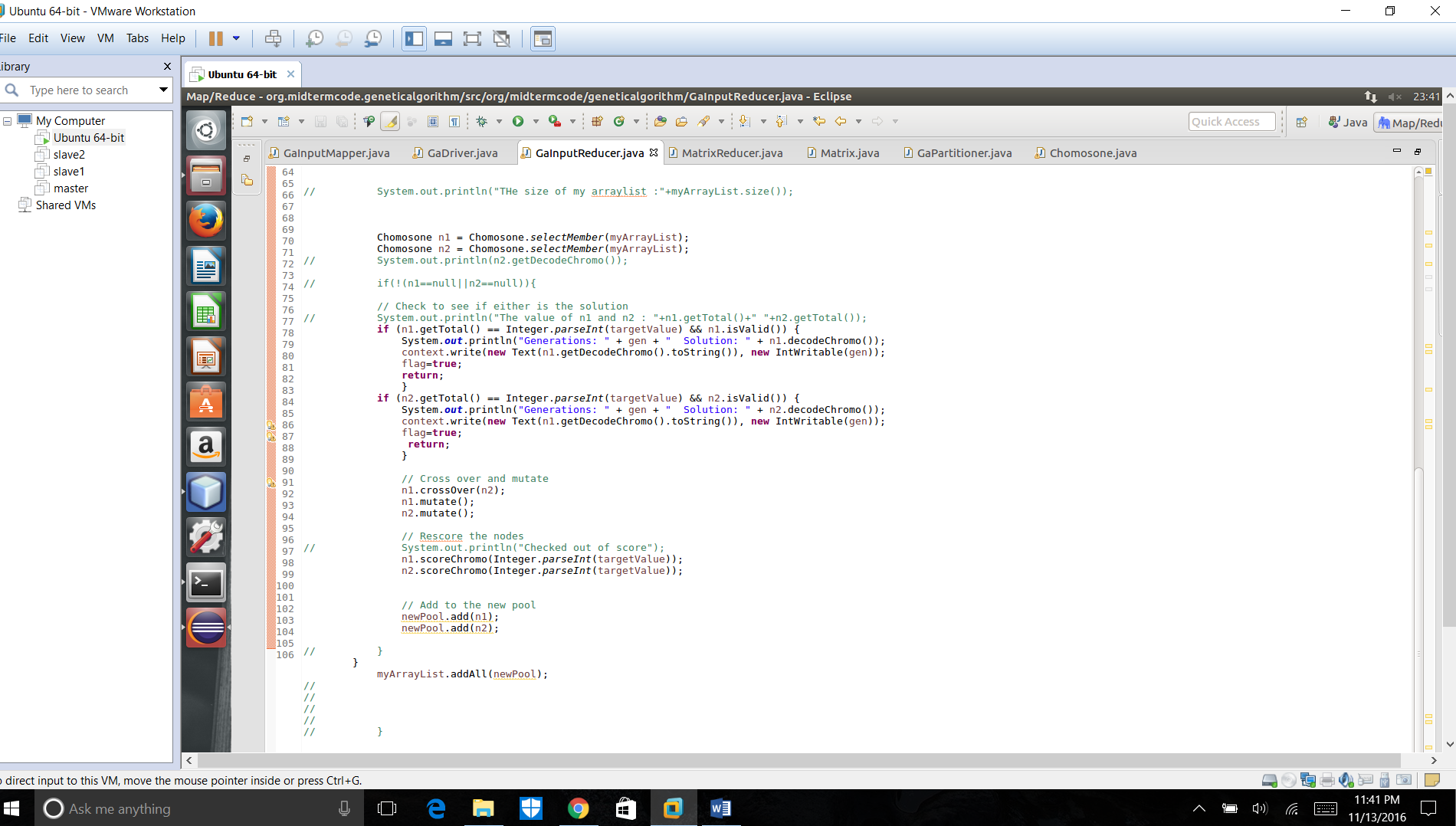
**Reducer**:

Now, I create an arraylist to collect all the chromosomes and figure out the size of the pool.

Later I perform the crossover and mutation over the chromosomes until a chromosome of the required targetvalue is generated. Finally,

I emit the **decoded chromosome** as **key** and the **number of generations** as **value.**

Screenshot of reducer:



**Final Output:**

I am not able to generate a final output from the reducer however. I wanted to try this approach to achieve convergence, however I could not. I have made changes to the code provided for this exam. The changes are made in methods like decodeChromo String(), addUp(), selectChromo(), a parameterized constructor of chromosome() (this constructer has been converted to a function and a default constructer is replaced in its place.)

**Artificial Neural Networks**:

**Input Layer:** We use the **mapper phase** to emit the key and values from an input file. The input layer of ANN, will contain the generated chromosomes and their corresponding fitness value.

**Hidden Layers**:

We filter the chromosomes to having a good fitness from those with poor values.

The ones with poor fitness are **crossover and mutated** to look for an increase in fitness.

We set the **bias value to 0.1 (10%)** and if the fitness value has increased by 10% we consider the chromosomes in good list. This could be performed in a combiner phase of mapreduce.

We could **check for the fitness and update the scores** of every poorly performing chromosomes.

**Output Layer:**

The output layer **emits the decoded string and number of generations** to achieve the result. This is equivalent to reducer phase of a mapreduce. We perform the crossover and mutation in reducer as well so that the results and convergence is quick.