**Partitioner**

* Objective - use custom Partitioner

1. copy as PartitionerExample.java

* Save Java MapReduce code in a file named PartitionerExample.java

1. hdfs dfs -put /scripts/volcheck /volcheck

* hdfs dfs -put /scripts/volcheck /volcheck uploads the local file /scripts/volcheck to the HDFS directory /volcheck

1. mkdir -m 755 partitionerexample\_classes

* mkdir -m 755 partitionerexample\_classes creates a directory named partitionerexample\_classes with read, write, and execute permissions for the owner and read and execute permissions for others

1. javac -classpath $(hadoop classpath) -d partitionerexample\_classes PartitionerExample.java

* javac -classpath $(hadoop classpath) -d partitionerexample\_classes PartitionerExample.java compiles PartitionerExample.java with Hadoop libraries and outputs the class files to the partitionerexample\_classes directory.

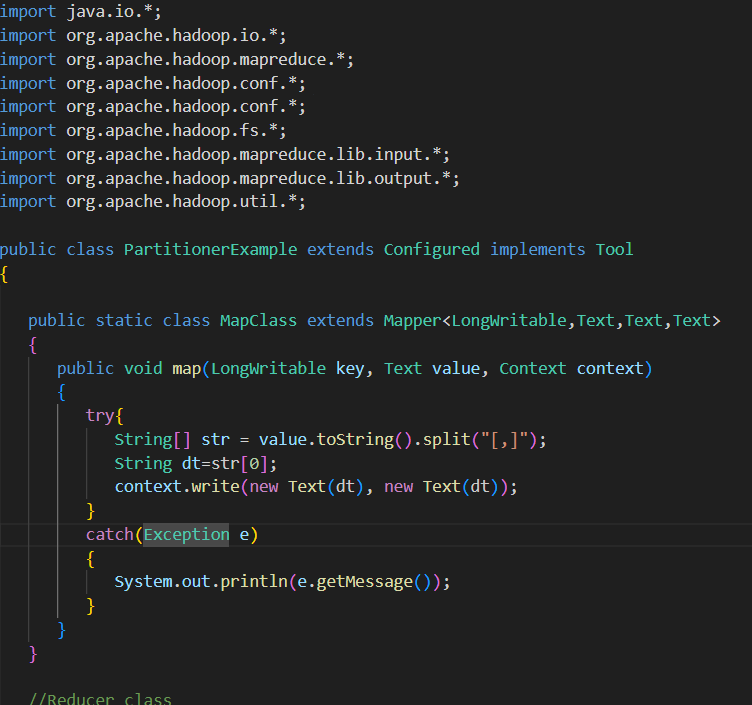
1. jar -cvf ${HOME}/scripts/partitionerexample.jar -C partitionerexample\_classes/ .

* jar -cvf ${HOME}/scripts/partitionerexample.jar -C partitionerexample\_classes/ . packages the compiled classes from partitionerexample\_classes into a JAR file named partitionerexample.jar in the specified path

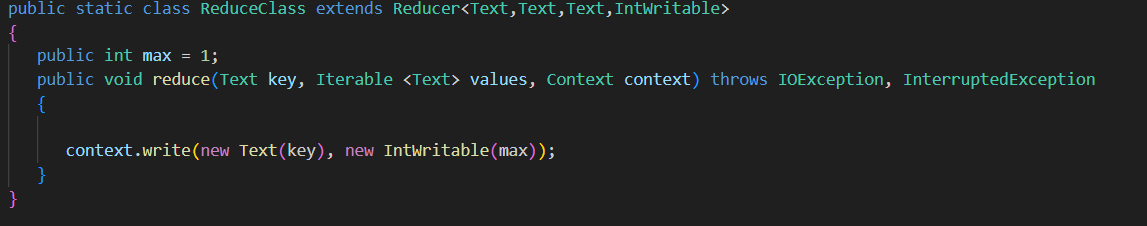
1. hadoop jar ${HOME}/scripts/partitionerexample.jar PartitionerExample /volcheck /v1

* hadoop jar ${HOME}/scripts/partitionerexample.jar PartitionerExample /volcheck /v1 executes the MapReduce job using the JAR file, with PartitionerExample as the main class, processing input from /volcheck and writing output to /v1 in HDFS

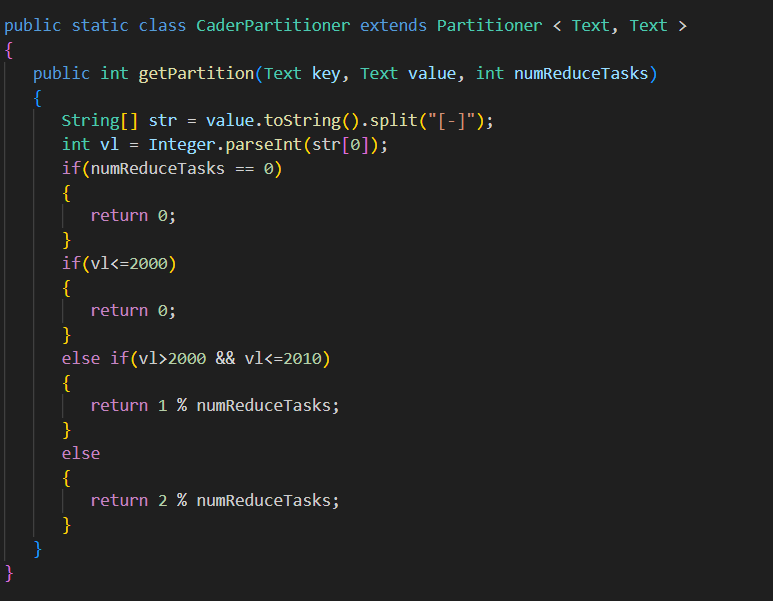
1. copy as PartitionerExample.java



* CaderPartitioner splits the value by [-], but in the MapClass, you're writing the key as dt (date) and value as dt. This implies that the partitioner should be using key rather than value. Ensure the partitioning logic aligns with the data format.
* The calculation of partition number is using modulo with numReduceTasks, which is correct. Ensure that numReduceTasks is greater than 0 to avoid division by zero errors.



* Current reducer implementation doesn't seem to be using the values from the reducer. The commented-out code seems to be intended for computing a maximum value, but it's not operational. If you want to find the maximum, you'll need to process the values properly.



max = -1;

for (Text val : values)

{

String [] str = val.toString().split("\t");

if(Integer.parseInt(str[4])>max)

max=Integer.parseInt(str[4]);

}



* The ToolRunner.run method should be used to pass command-line arguments to the run method. Your main method is missing proper argument handling and should use ToolRunner.run correctly.
* Be sure to handle exceptions and edge cases, particularly in parsing and conversion operations.
* Partitioner Class: Updated to use key for partitioning and added error handling.
* Main Method: Fixed argument handling and used ToolRunner.run correctl
* Reducer Class: Fixed the handling of values to find the maximum value.