

## PROGRAM 8A : Word Count program using MapReduce WITH hadoop

Step1: goto Search button run cmd prompt as administrator

Step2: Initiate all required component of hadoop by command  
start-all.cmd

Step3: Create a text file where we have folder running hadoop java program in  
my case I have created in Desktop

U can create anywhere as u wish

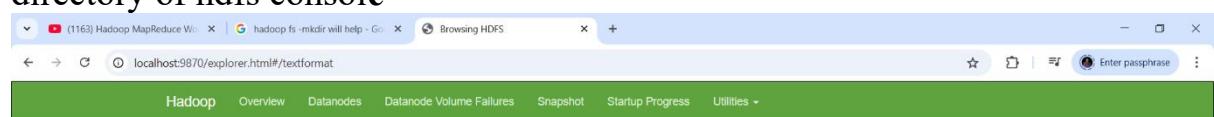
Step4: Create a Directory in HDFS console using command  
hadoop fs -mkdir /textformat

Where textform is ur input directory

Step5: We have to add text into a directory<textformat> in my case b.txt is in  
desktop folder copy the path and type command

hadoop fs -put C:\Users\User\Desktop\b.txt /textformat

After completion of above command u be able to see b.txt in /textformat  
directory of hdfs console



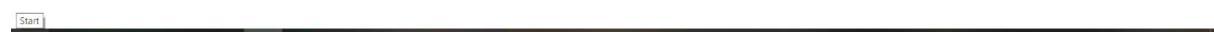
### Browse Directory

Browse Directory							
/textformat							
Show 25 entries							
□	Permission	Owner	Group	Size	Last Modified	Replication	Block Size
□	-rw-r--r--	User	supergroup	20 B	Jun 30 20:39	1	128 MB

Showing 1 to 1 of 1 entries

Previous 1 Next

Hadoop, 2024.



In cmd prmpmt if u want to check type the command as  
hadoop fs -ls /textformat

Step6: Now we need a jar file to execute mapreduce program fr wordcount  
So open Eclipse

Create a java project with name MapReduceWordCount

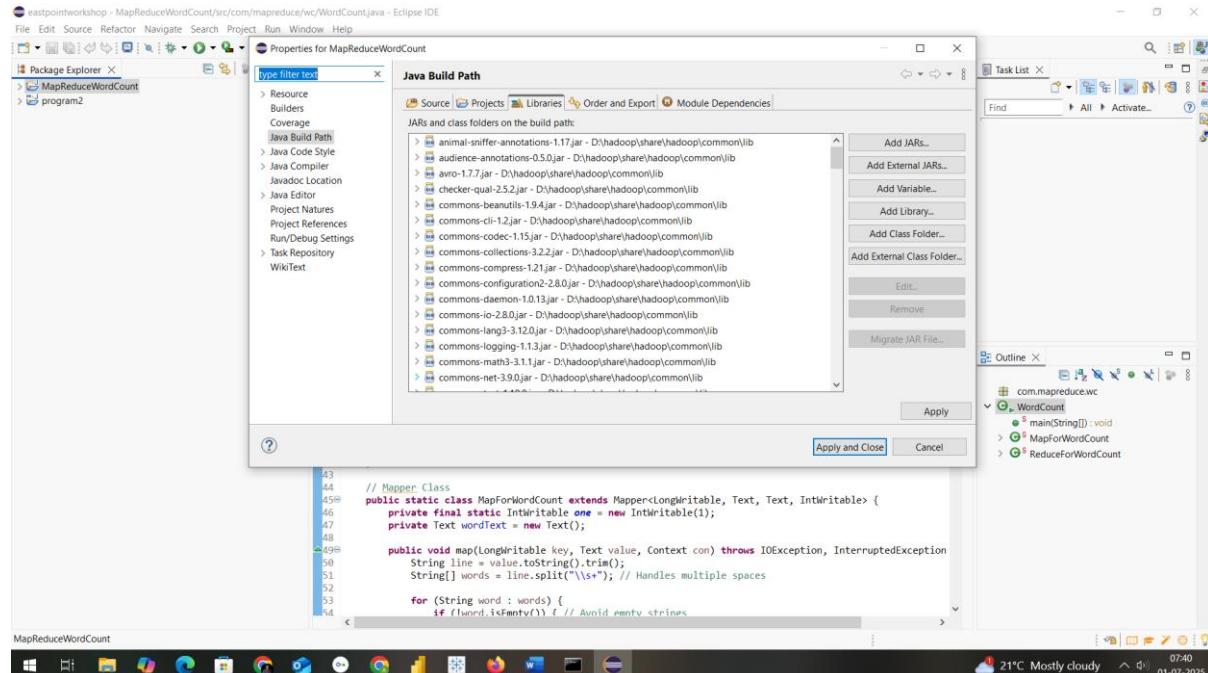
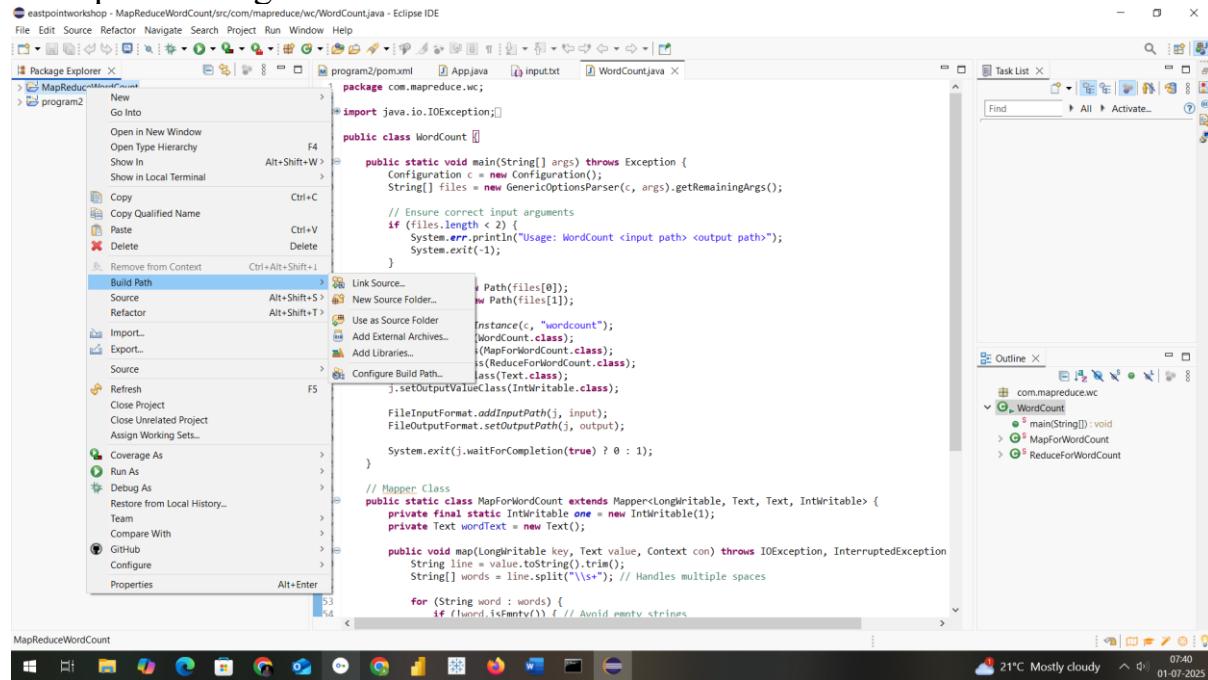
Select execution environment as JavaSE-1.8

And click on next&finish

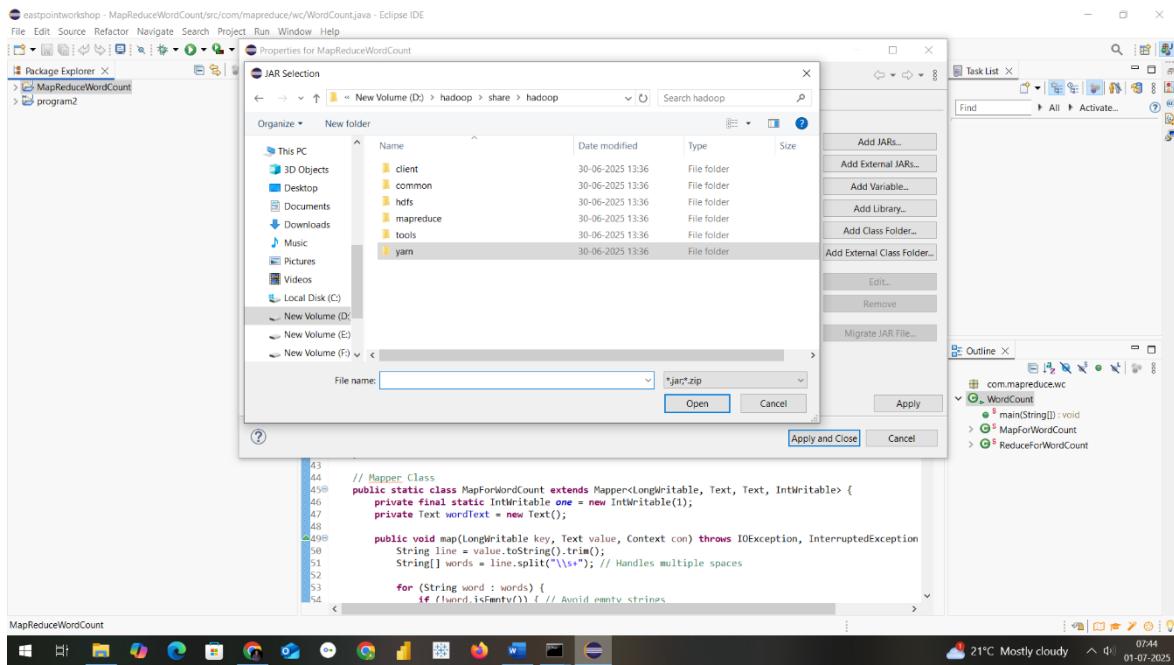
Step7: Right Click on Projectname---→ Click on New---→create a package

with name com.mapreduce.wc and click on Finish

Step8: Add required libraries to support hadoop by navigating as in below  
Right Click on Project-----→Right Click on BuildPath-----→ Configure Buildpath Then goto Libraries as in second screen below



Step9: Add necessary External jar file from D:\hadoop\share\hadoop Like clients,common,hdfs,mapreduce & yarn to support packages for hadoop



Step10: Create a class within package com.mapreduce.wc with name WordCount and paste this code  
package com.mapreduce.wc;

```

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
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;
import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {

    public static void main(String[] args) throws Exception {
        Configuration c = new Configuration();
        String[] files = new GenericOptionsParser(c, args).getRemainingArgs();

        // Ensure correct input arguments
        if (files.length < 2) {
            System.err.println("Usage: WordCount <input path> <output path>");
        }
    }
}

```

```

        System.exit(-1);
    }

    Path input = new Path(files[0]);
    Path output = new Path(files[1]);

    Job j = Job.getInstance(c, "wordcount");
    j.setJarByClass(WordCount.class);
    j.setMapperClass(MapForWordCount.class);
    j.setReducerClass(ReduceForWordCount.class);
    j.setOutputKeyClass(Text.class);
    j.setOutputValueClass(IntWritable.class);

    FileInputFormat.addInputPath(j, input);
    FileOutputFormat.setOutputPath(j, output);

    System.exit(j.waitForCompletion(true) ? 0 : 1);
}

// Mapper Class
public static class MapForWordCount extends Mapper<LongWritable, Text,
Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text wordText = new Text();

    public void map(LongWritable key, Text value, Context con) throws
IOException, InterruptedException {
        String line = value.toString().trim();
        String[] words = line.split("\\s+"); // Handles multiple spaces

        for (String word : words) {
            if (!word.isEmpty()) { // Avoid empty strings
                wordText.set(word.trim().toUpperCase());
                con.write(wordText, one);
            }
        }
    }
}

// Reducer Class
public static class ReduceForWordCount extends Reducer<Text, IntWritable,
Text, IntWritable> {
    public void reduce(Text word, Iterable<IntWritable> values, Context con)

```

```

        throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable value : values) {
                sum += value.get();
            }
            con.write(word, new IntWritable(sum));
        }
    }
}

```

## **PROGRAM 8B:Word Program Count Procedure using Apache Spark**

### **Step 1: Prerequisites**

Ensure you have the following installed:

- Java JDK (8 or 11)
  - Eclipse IDE (preferably Eclipse IDE for Java Developers)
  - Apache Maven
- 

### **Step 2: Create Maven Project in Eclipse**

1. Open Eclipse → File → New → Maven Project → Next.
  2. Select maven-archetype-quickstart, click Next.
  3. Provide the following:
    - Group Id: com.example.spark
    - Artifact Id: spark-wordcount
    - Click Finish.
- 

### **Step 3: Add Spark Dependencies in pom.xml**

Add this to your pom.xml:

```

<dependencies>
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_2.12</artifactId>
    <version>3.3.2</version>
</dependency>

<!-- Apache Spark SQL (Optional) -->
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_2.12</artifactId>

```

```

<version>3.3.2</version>
</dependency>

<!-- Logging (to avoid SLF4J warnings) -->
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-simple</artifactId>
    <version>1.7.30</version>
</dependency>
</dependencies>

<build>
    <plugins>
        <!-- Plugin to build a fat JAR -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-shade-plugin</artifactId>
            <version>3.2.4</version>
            <executions>
                <execution>
                    <phase>package</phase>
                    <goals><goal>shade</goal></goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>

```

#### Step 4: Create WordCount Program

Create a class WordCount.java under:

```

src/main/java/com/example/spark/WordCount.java
package com.example.spark.spark_wordcount;
import org.apache.spark.api.java.*;
import org.apache.spark.SparkConf;
import scala.Tuple2;
import java.util.Arrays;

```

```

public class App {
    public static void main(String[] args) {
        // Set Spark Configuration
        SparkConf conf = new
        SparkConf().setAppName("WordCount").setMaster("local[*]");

```

```

JavaSparkContext sc = new JavaSparkContext(conf);

    // Load input file
    JavaRDD<String> input = sc.textFile("D:/hadoopprograms/input.txt"); // Replace with your file path

    // FlatMap each line to words
    JavaRDD<String> words = input.flatMap(line -> Arrays.asList(line.split("")).iterator());

    // Map each word to a pair (word, 1)
    JavaPairRDD<String, Integer> wordPairs = words.mapToPair(word -> new Tuple2<>(word, 1));

    // Reduce by key (word)
    JavaPairRDD<String, Integer> wordCounts =
    wordPairs.reduceByKey(Integer::sum);

    // Print output
    wordCounts.foreach(result -> System.out.println(result._1() + ": " +
result._2()));

    sc.close();
}
}

```

### Step 5: Input File

Create a input.txt file in folder where u r saving Hadoop programs in mine its in D:\hadoopprograms

Example content:

**hello world**

hello spark

spark is fast

### Step 6: Run the Program

1. Right-click WordCount.java → Run As → Java Application.

2. Console should print:

hello: 2

world: 1

spark: 2

is: 1

fast: 1

## RESULT

```
spark-workstation@spark-vm:~/anaconda3/envs/experiments$ spark-submit --class AppMain C:\Program Files\Apache Software Foundation\Apache2.0\target\spark-0.9.0-bin-hadoop2.6.jar
```

Java File Editor Navigator Search Project Run Window Help

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer X

Problems JavaDoc Declaration Context X Eclipse IDE for Java Developers 2023-06 Release

Terminal App [Bash Application] C:\Program Files\Apache Software Foundation\Apache2.0\target\spark-0.9.0-bin-hadoop2.6\bin\spark-submit.bat [0d:1990]

25/07/07 09:14:10 INFO SparkScheduler: Created broadcast 2 from broadcast at D:\spark\src\main\scala\1513

25/07/07 09:14:10 INFO TaskScheduler: Submitted 2 missing tasks for stage 0 (TID 0) at reducedByKey at App.java:77 (FileInputFormat) for part 0

25/07/07 09:14:10 INFO TaskScheduler: Adding 2 tasks for stage 0, with 2 levels of partitioning

25/07/07 09:14:10 INFO TaskSetManager: Starting task 0.0 in stage 0 (TID 2) (DESKTOP-VLJ66G5 executor driver, partition 0, NODE\_LOCAL, 4271 bytes) tasks=2

25/07/07 09:14:10 INFO Executor: Running task 0.0 in stage 1.0 (TID 2)

25/07/07 09:14:10 INFO Executor: Running task 1.0 in stage 1.0 (TID 3)

25/07/07 09:14:10 INFO Executor: Started 2 tasks in total (2 local) 0 non-empty blocks including 2 (290.0 B) local and 0 (0.0 B) host-local and 0 (0.0 B) published to driver

25/07/07 09:14:10 INFO ShuffledBlockFetcherIterator: Iterating over 0 remote fetches in 41 ms

25/07/07 09:14:10 INFO ShuffledBlockFetcherIterator: Getting 2 (298.0 B) non-empty blocks including 2 (298.0 B) local and 0 (0.0 B) host-local and 0 (0.0 B) published to driver

25/07/07 09:14:10 INFO ShuffledBlockFetcherIterator: Started 0 remote fetches in 50 ms

spark://127.0.0.1:4040

is: 1

host: 1

hello: 2

world: 1

25/07/07 09:14:10 INFO Executor: Finished task 0.0 in stage 1.0 (TID 2). 1223 bytes result sent to driver

25/07/07 09:14:10 INFO Executor: Finished task 1.0 in stage 1.0 (TID 3). 3275 bytes result sent to driver

25/07/07 09:14:10 INFO TaskScheduler: Finished task 0.0 in stage 1.0 (TID 2) in 29 ms on DESKTOP-VLJ66G5 (execute d-driver) (1/2)

25/07/07 09:14:10 INFO TaskScheduler: Removed task 0.0 from pool, whose tasks have all completed.

25/07/07 09:14:10 INFO TaskScheduler: ResultStage 1 (reduce at App.java:80) finished in 0.30s on DESKTOP-VLJ66G5 (execute d-driver) (1/2)

25/07/07 09:14:10 INFO TaskScheduler: Removed task 1.0 from pool, whose tasks have all completed, from pool

25/07/07 09:14:10 INFO TaskScheduler: Killing all running tasks in step 1: Stage finished

25/07/07 09:14:10 INFO TaskScheduler: Job 0 finished: forward at App.java:80, took 3.09/158 s

25/07/07 09:14:10 INFO BlockManagerMaster: Registered block manager spark://127.0.0.1:4040

25/07/07 09:14:10 INFO ReplicationTrackerReactor: InputPort[spark://127.0.0.1:4040] stopped!

25/07/07 09:14:10 INFO MemorySystem: MemorySystem cleared

25/07/07 09:14:10 INFO BlockManagerMaster: BlockManagerMaster stopped

25/07/07 09:14:10 INFO OutputCommitCoordinator\$OutputCommitCoordinatorInputPort: OutputCommitCoordinator stopped!

25/07/07 09:14:10 INFO ShutdownHookManager: Shutdown hook called

25/07/07 09:14:10 INFO ShutdownHookManager: Deleting directory C:\Users\User\AppData\Local\Temp\spark-cfa6645-9945-40bd-9f5a-9f9d020751ef