**javac -cp ${HADOOP\_HOME}/share/hadoop/common/hadoop-common-3.2.1.jar:${HADOOP\_HOME}/share/hadoop/mapreduce/hadoop-mapreduce-client-core-3.2.1.jar -d wordcount\_classes \*.java**

This command compiles all Java files in the current directory using the `javac` compiler, with Hadoop libraries included in the classpath. The compiled bytecode is placed in the `wordcount\_classes` directory.

This command is used to compile Java source files into bytecode, which can then be executed by the Java Virtual Machine (JVM). Here's a breakdown of each part:

- `javac`: This is the Java compiler command. It's used to compile Java source files.

- `-cp`: This stands for classpath. The classpath is a parameter in the Java Virtual Machine or the Java compiler that specifies the location of user-defined classes and packages. The parameter may be set either on the command-line, or through an environment variable.

-`${HADOOP\_HOME}/share/hadoop/common/hadoop-common 3.2.1.jar:${HADOOP\_HOME}/share/hadoop/mapreduce/hadoop-mapreduce-client-core-3.2.1.jar`: These are the paths to the Hadoop JAR files that your Java program depends on. The `${HADOOP\_HOME}` is an environment variable that should be set to the location of your Hadoop installation. The `:` character is used to separate multiple paths in the classpath.

- `-d wordcount\_classes`: This tells the compiler to put the output `.class` files (the compiled bytecode) in the `wordcount\_classes` directory.

- `\*.java`: This is a wildcard that matches all `.java` files in the current directory. So this command will compile all `.java` files in the current directory.

So, in summary, this command is compiling all Java source files in the current directory, using the specified Hadoop JAR files in the classpath, and outputting the resulting class files to the `wordcount\_classes` directory.

**jar -cvf wordcount.jar -C wordcount\_classes/ .**

This command creates a JAR (Java Archive) file named `wordcount.jar` using the `jar` command. The `-cvf` options tell the command to create a new archive (`c`), output verbose info to the console (`v`), and specify the output file name (`f`). The `-C wordcount\_classes/ .` part changes to the `wordcount\_classes/` directory and includes all files in that directory (`.`) in the JAR.

**hadoop jar /home/hadoop/wordcount.jar Driver /data\_wordcount/wc.txt /output\_dir\_wc**

This command runs a Hadoop job using the `hadoop jar` command. The JAR file to be run is `/home/hadoop/wordcount.jar`, and `Driver` is the main class that the Hadoop job will start executing from. `/data\_wordcount/wc.txt` is the input directory or file for the Hadoop job, and `/output\_dir\_wc` is the directory where the job's output will be stored.

**hadoop dfs -cat /output\_dir\_wc/\***

This command executes a Hadoop job using the `hadoop jar` command. It runs the `Driver` class from the `wordcount.jar` file, using `/data\_wordcount/wc.txt` as input. The results are stored in the `/output\_dir\_wc` directory on HDFS.

Program-II

javac -cp ${HADOOP\_HOME}/share/hadoop/common/hadoop-common-3.2.1.jar:${HADOOP\_HOME}/share/hadoop/mapreduce/hadoop-mapreduce-client-core-3.2.1.jar -d evenodd\_count\_classes \*.java

jar -cvf evenoddcount.jar -C evenodd\_count\_classes/ .

hadoop jar /home/hadoop/evenoddcount.jar Driver /data\_wordcount/input.txt /output\_dir\_even\_odd\_count

hadoop dfs -cat /output\_dir\_even\_odd\_count/\*