## Write the Java MapReduce Program for Inverted Index

Now, let's write the complete Java code for the MapReduce job.

### Inverted Index MapReduce Job

```
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.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;
import java.util.HashSet;
import java.util.StringTokenizer;
import java.util.Set;

public class InvertedIndex {

    // Mapper class
    public static class InvertedIndexMapper extends Mapper<Object, Text,
Text, Text> {
        private Text word = new Text();
        private Text docID = new Text();

        public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
            // Get the document ID from the input line (you can use the key
itself as a document ID)
            String document = value.toString();
            String[] parts = document.split("\t", 2);
            String documentID = parts[0];
            String content = parts[1];

            // Tokenize the document content
            StringTokenizer tokenizer = new StringTokenizer(content);

            // Emit each word with the document ID
            while (tokenizer.hasMoreTokens()) {
                String token = tokenizer.nextToken().toLowerCase();
                word.set(token);
                docID.set(documentID);
                context.write(word, docID);
            }
        }
    }
```

```
    // Reducer class
    public static class InvertedIndexReducer extends Reducer<Text, Text,
Text, Text> {
        private Text result = new Text();

        public void reduce(Text key, Iterable<Text> values, Context
context) throws IOException, InterruptedException {
            Set<String> documentIDs = new HashSet<>();
            for (Text val : values) {
                documentIDs.add(val.toString());
            }
            result.set(String.join(",", documentIDs));
            context.write(key, result);
        }
    }

    // Main driver method
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Inverted Index");

        job.setJarByClass(InvertedIndex.class);
        job.setMapperClass(InvertedIndexMapper.class);
        job.setReducerClass(InvertedIndexReducer.class);

        // Set the output key and value types for the job
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        // Set the input and output paths (taken from arguments)
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        // Wait for the job to complete
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

## Step 3: Explanation of the Code

1. **Mapper Class** (`InvertedIndexMapper`):
   - The `map()` method is where the document is read, and words are emitted as keys with the document ID as the value.
   - The document is split into words using a `StringTokenizer`, and each word is converted to lowercase.
   - We use `context.write(word, documentID)` to emit the word as the key and the document ID as the value.
2. **Reducer Class** (`InvertedIndexReducer`):
   - The `reduce()` method aggregates the document IDs for each word.
   - A `HashSet` is used to store document IDs to avoid duplicates.
   - Finally, the document IDs are joined with commas and written out as the result.
3. **Main Method**:
   - Configures the job, sets the mapper and reducer classes, and specifies the input and output paths.
   - It expects two arguments: the input directory and the output directory.

- After setting up the job, it runs the job and waits for completion.

## Step 4: Running the Program

To run the program on Hadoop, follow these steps:

1. **Compile the Java Code**:
   - First, compile the Java code using `javac` and create a JAR file.
   - Ensure that you include the Hadoop libraries in your classpath.

```
javac -classpath $(hadoop classpath) -d /path/to/output InvertedIndex.java
jar -cvf invertedindex.jar -C /path/to/output .
```

2. **Run the Program on Hadoop**:
   - Once the JAR file is created, you can run the program using Hadoop's `hadoop hadoop jar invertedindex.jar InvertedIndex input_directory output_directory`

Where:

- `input_directory` is the directory containing the input text files (where each line is a document, and the format is `docID<tab>content`).
- `output_directory` is where the output (inverted index) will be stored.

## Step 5: Example Input and Output

**Input (in `input.txt`):**

```
swift
Copy
1    The quick brown fox
2    The lazy dog
3    The quick dog
```

**Output (in `output` directory, `part-r-00000`):**

```
swift
Copy
brown   1
dog     2,3
fox     1
lazy    2
quick   1,3
the     1,2,3
```

This output shows each word from the documents along with the list of document IDs where the word appears.