Certainly! Below is a structured outline for a typical MapReduce job in Java, including method names and explanations of their parameters.

1. Mapper Class

Function Name: `map()`

Logical Outline:

- **Purpose:** Processes input key-value pairs and generates intermediate key-value pairs.
- **Parameters:**
  - `KEYIN key`: The input key from the input split. The type is typically `LongWritable` for line offsets or `Text` for line content.
  - `VALUEIN value`: The input value associated with the key. This is often `Text` for line content or other data types depending on the input format.
  - `Context context`: An instance of `Mapper.Context` used to write intermediate key-value pairs and report progress.

Example:

```java
public class MyMapper extends Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
    @Override
    protected void map(KEYIN key, VALUEIN value, Context context) throws IOException,
InterruptedException {
        // Implementation here
    }
}
```

2. Reducer Class

Function Name: `reduce()`

Logical Outline:

- **Purpose:** Processes intermediate key-value pairs generated by the `map()` function and produces final output key-value pairs.
- **Parameters:**
  - `KEYIN key`: The intermediate key that the reducer will process. This is typically the same type as the key output by the mapper.
  - `Iterable<VALUEIN> values`: A collection of values associated with the key. The values are grouped by key and passed as an iterable.
  - `Context context`: An instance of `Reducer.Context` used to write final output key-value pairs and report progress.

Example:

```java
public class MyReducer extends Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
    @Override
    protected void reduce(KEYIN key, Iterable<VALUEIN> values, Context context) throws
IOException, InterruptedException {
        // Implementation here
    }
}
```

3. Job Class

Function Name: `configureJob()`

Logical Outline:

- **Purpose:** Configures the MapReduce job, including setting input and output formats, mapper and reducer classes, and other job-specific parameters.
- **Parameters:**
  - `Job job`: An instance of the `Job` class that represents the MapReduce job configuration.
  - `Class<? extends Mapper<?, ?, ?, ?>> mapperClass`: The mapper class to be used for the job.
  - `Class<? extends Reducer<?, ?, ?, ?>> reducerClass`: The reducer class to be used for the job.
  - `Class<?> outputKeyClass`: The type of the output key.
  - `Class<?> outputValueClass`: The type of the output value.
  - `Path inputPath`: The path of the input data.
  - `Path outputPath`: The path where the output data will be written.

Example:

```
public class MyJobConfiguration {
    public static void configureJob(Job job) throws IOException {
        job.setJarByClass(MyJobConfiguration.class);
        job.setMapperClass(MyMapper.class);
        job.setReducerClass(MyReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path("input/path"));
        FileOutputFormat.setOutputPath(job, new Path("output/path"));
        // Additional configurations if needed
    }
}
```

4. Job Context Interface

Function Name: `getConfiguration()`

Logical Outline:

- **Purpose:** Provides access to the configuration settings of the current job.
- **Parameters:**
  - None (method is typically called without parameters).
- **Returns:**
  - `Configuration`: Returns the `Configuration` object associated with the job context. This object holds all the configuration settings for the

MapReduce job.

Example:

```
public interface JobContext {
    Configuration getConfiguration();
}
```

Note: In a real-world implementation, `JobContext` is an interface provided by Hadoop, and you often interact with its implementation rather than directly implementing it.

This outline should give you a clear understanding of the key methods and parameters involved in setting up and running a MapReduce job using the Hadoop framework.