1. Clase Mapper

La clase Mapper procesará las líneas de entrada y emitirá pares clave-valor donde la clave será el tipo de producto y el valor será el ID de la persona.

```
package ProductView;

import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;

public class ProductViewMapper extends MapReduceBase implements
Mapper<LongWritable, Text, Text, IntWritable> {
    public void map(LongWritable key, Text value,
OutputCollector<Text, IntWritable> output, Reporter reporter) throws
IOException {
        String valueString = value.toString();
        String[] visitData = valueString.split(",");
        int personID = Integer.parseInt(visitData[0]);
        String productType = visitData[2];
        output.collect(new Text(productType), new
IntWritable(personID));
    }
}
```

2. Clase Reducer

La clase Reducer agrupará los valores por clave (tipo de producto) y emitirá una lista de IDs de personas que han observado ese tipo de producto.

```
package ProductView;

import java.io.IOException;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;

public class ProductViewReducer extends MapReduceBase implements
Reducer<Text, IntWritable, Text, Text> {
    public void reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text, Text> output, Reporter reporter) throws
IOException {
        Set<Integer> uniquePersonIDs = new HashSet<>();
        while (values.hasNext()) {
            uniquePersonIDs.add(values.next().get());
        }
        output.collect(key, new Text(uniquePersonIDs.toString()));
    }
}
```

3. Clase Driver

La clase Driver configurará y ejecutará el trabajo MapReduce.

```
package ProductView;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

public class ProductViewDriver {
    public static void main(String[] args) {
        JobClient my_client = new JobClient();
        JobConf job_conf = new JobConf(ProductViewDriver.class);

        job_conf.setJobName("ProductViewCount");
        job_conf.setOutputKeyClass(Text.class);
        job_conf.setOutputValueClass(IntWritable.class);

        job_conf.setMapperClass(ProductViewMapper.class);
        job_conf.setReducerClass(ProductViewReducer.class);

        job_conf.setInputFormat(TextInputFormat.class);
        job_conf.setOutputFormat(TextOutputFormat.class);

        FileInputFormat.setInputPaths(job_conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(job_conf, new Path(args[1]));

        my_client.setConf(job_conf);
        try {
            JobClient.runJob(job_conf);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Ejemplo de archivo CSV

Aquí tienes un ejemplo de cómo podría verse un archivo CSV con datos sobre visitas a páginas web:

```
PersonID,VisitDate,ProductType,ProductID
1,2025-01-01,Electronics,101
2,2025-01-02,Clothing,102
1,2025-01-03,Electronics,103
3,2025-01-04,Books,104
2,2025-01-05,Clothing,105
3,2025-01-06,Electronics,106
4,2025-01-07,Books,107
4,2025-01-08,Clothing,108
5,2025-01-09,Electronics,109
5,2025-01-10,Books,110
```

Guarda este contenido en un archivo llamado `web_visits.csv` y úsalo como entrada para tu trabajo de MapReduce.

Crear el archivo JAR y ejecutar el trabajo

Sigue los mismos pasos que antes para compilar las clases, crear el archivo JAR y ejecutar el trabajo MapReduce:

1. Compilar las clases Java:

```
2. javac -classpath $(hadoop classpath) -d . ProductView/*.java
```

3. Crear el archivo JAR:

```
4. jar -cvf ProductView.jar -C . ProductView
```

5. Ejecutar el trabajo MapReduce:

```
hadoop jar ProductView.jar ProductView.ProductViewDriver
/input/web_visits.csv /output
```