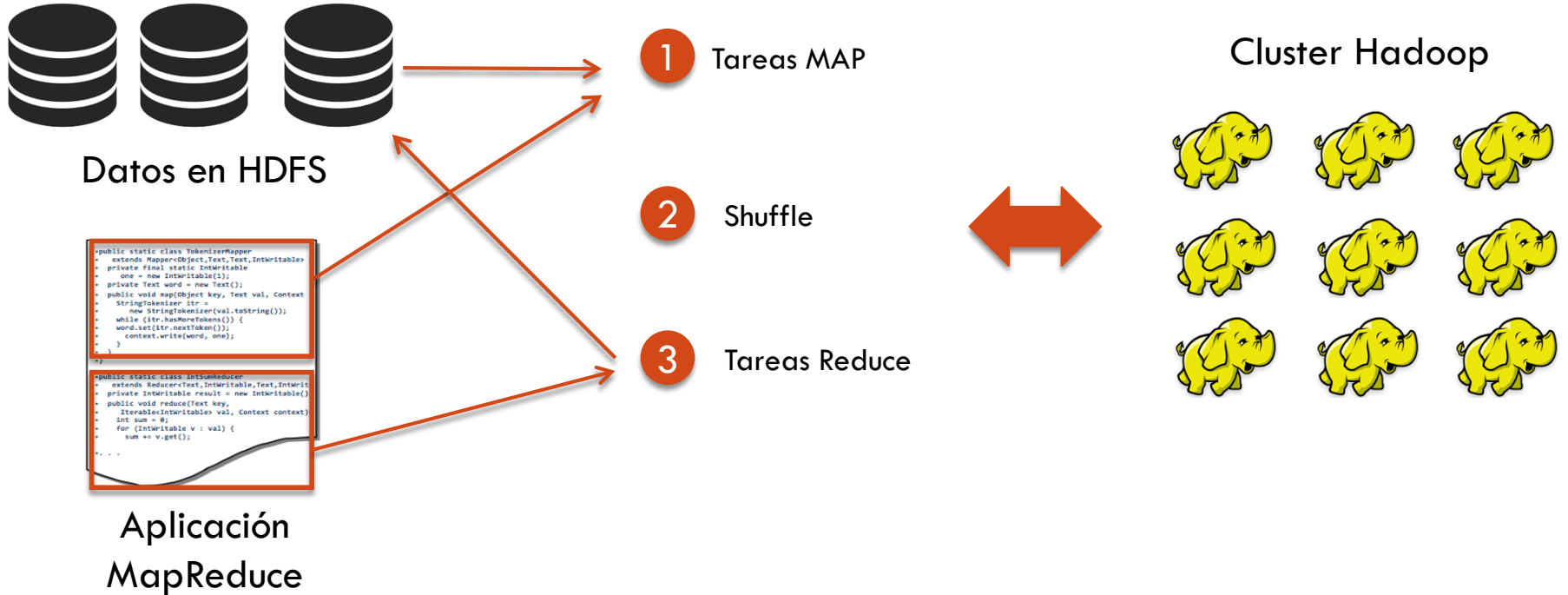


# Programación básica en Java con Hadoop

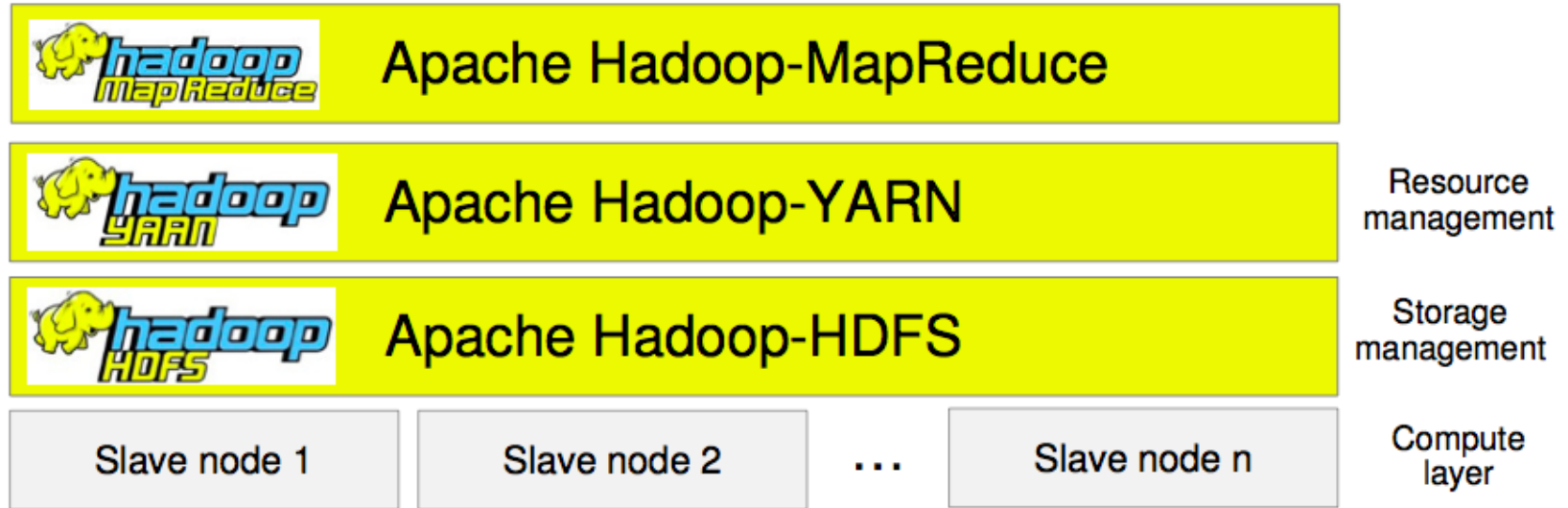
# Tabla de contenidos

- Repaso: ejecución en Hadoop (MapReduce)
- El lenguaje de programación Java
- Ejemplo Java en Hadoop: WordCount
- Crear una aplicación para Hadoop usando Java

# Cómo se ejecuta una aplicación Hadoop



# Arquitectura Hadoop



# Hadoop es...

- Framework de desarrollo para aplicaciones Big Data
  - Paradigma MapReduce
    - Google™
- Entorno de ejecución
  - Aplicaciones tipo batch
  - Lectura intensiva
- Se encarga de almacenar los datos generados
- Escalable

# ¿Cómo desarrollar aplicaciones Hadoop?

- **Desarrollos Map/reduce en JAVA**

- Muy Complejo

- **PIG**

- Lenguaje Open/Source de más alto nivel
- Estándar



- **HIVE**

- Lenguaje Open/Source
- Similar al SQL



- **JAQL**

- ✓ Lenguaje similar a PIG, mayor funcionalidad

- **Herramientas tipo BigSheets**

- ✓ Navegador/Hoja de Cálculo
- ✓ No requiere desarrollo



Difícil



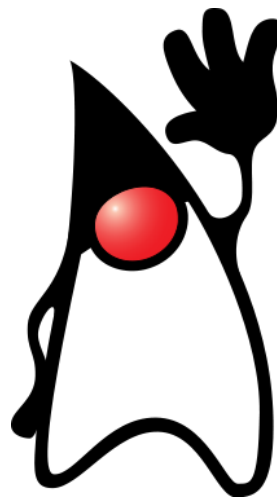
Fácil

# Hands on

- **Guión de prácticas**
  - Conexión y uso de un sistema Hadoop
  - Ejecución de una aplicación de ejemplo: WordCount

# El lenguaje de programación Java

- Creado en el año 1991 (oak) por *Sun Microsystems*
- Objetivos de diseño:
  - Lenguaje orientado objetos
  - Máquina virtual de Java
  - Fácil de utilizar
  - Ejecución de código remoto y soporte de red





# EL lenguaje de programación Java

## ➤ Lenguaje orientado a objetos

### ➤ Clases

#### ➤ Herencia

#### ➤ Interfaces

### ➤ Objetos

### ➤ Métodos

### ➤ Parámetros por referencia

### ➤ Sintaxis

SampleClass
sample
~fieldPackage:String -fieldPrivate:String #fieldProtected:String +fieldPublic:String +nestedSampleClass:NestedSampleClass
+SampleClass():void ~methodPackage():void -methodPrivate():void #methodProtected():void +methodPublic():void

Definición del “tipo de variable” más completa que en otros lenguajes: campos y métodos (y accesibilidad de los mismos).

- ¿Qué operaciones puedes hacerse sobre un objeto?
- ¿Qué campos/atributos definen un objeto?
- ¿Qué métodos y atributos son accesibles y a quién?

static: campos comunes a todos los objetos de una misma clase

# EL lenguaje de programación Java

## ➤ Lenguaje orientado a objetos

### ➤ Clases

#### ➤ Herencia

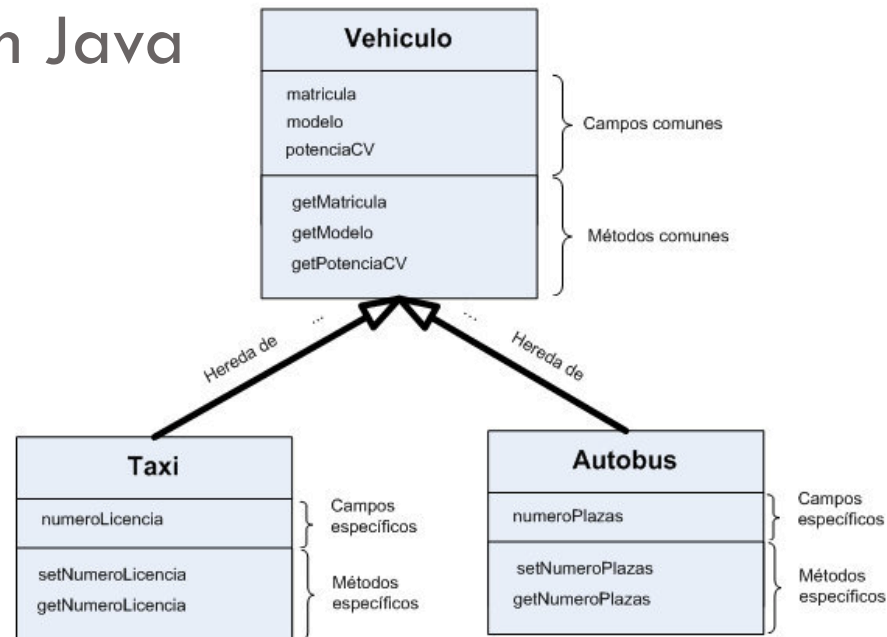
#### ➤ Interfaces

### ➤ Objetos

### ➤ Métodos

### ➤ Parámetros por referencia

### ➤ Sintaxis



Objetivo: reutilización de código y trabajo.

- Crear nuevas clases con características comunes a clases existentes
  - Los métodos heredados pueden sobrescribirse
- Ahorro de código y trabajo
- Beneficio del testeo y rendimiento de código muy utilizado

Un paso más: clases abstractas

# EL lenguaje de programación Java

## ➤ Lenguaje orientado a objetos

### ➤ Clases

#### ➤ Herencia

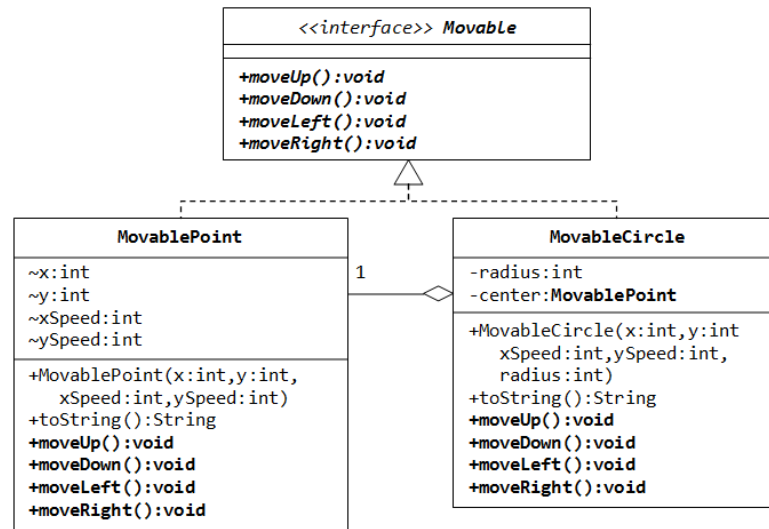
#### ➤ Interfaces

### ➤ Objetos

### ➤ Métodos

### ➤ Parámetros por referencia

### ➤ Sintaxis



Aproximación de Java para resolver el problema de la **herencia múltiple** (contemplada en otros lenguajes OO)

- Equivalente a una herencia de una clase abstracta
  - Se fuerza a que las clases que implementan la interfaz implementen los métodos
  - Cada clase lo puede implementar como desee

Ej: interfaz **Serializable**

# EL lenguaje de programación Java

## ➤ Lenguaje orientado a objetos

### ➤ Clases

#### ➤ Herencia

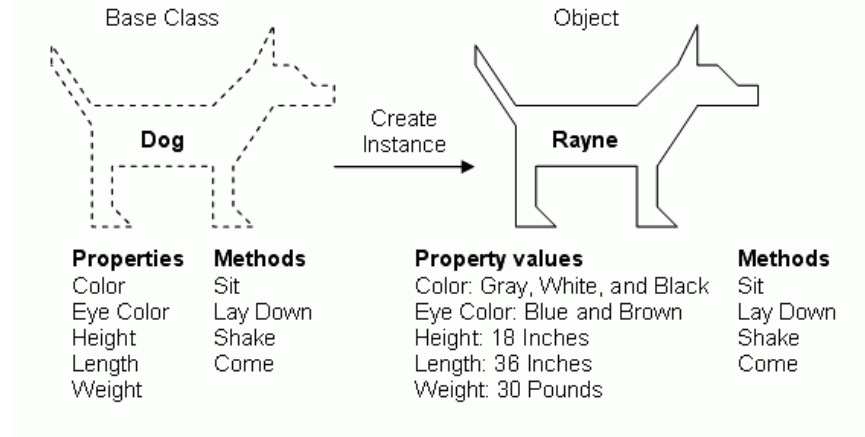
#### ➤ Interfaces

### ➤ Objetos

### ➤ Métodos

### ➤ Parámetros por referencia

### ➤ Sintaxis



Un objeto es una instancia de una clase. Ocupa tanto como el espacio requerido para almacenar los atributos (asociados a la clase) que identifican al objeto.

# EL lenguaje de programación Java

## ➤ Lenguaje orientado a objetos

### ➤ Clases

#### ➤ Herencia

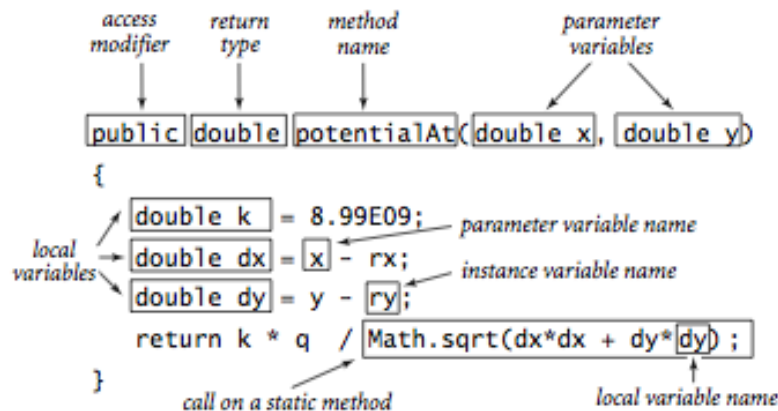
#### ➤ Interfaces

### ➤ Objetos

### ➤ Métodos

### ➤ Parámetros por referencia

### ➤ Sintaxis



Anatomy of a data-type method

Un método de una clase concreta se caracteriza por su accesibilidad, su nombre, su tipo de retorno y sus parámetros. En una clase pueden existir métodos diferentes que tengan el mismo nombre, pero que reciban parámetros de diferentes clases.

Accesibilidad: private, protected, public, default

Otros: static (solo acceso a métodos y campos static)

# EL lenguaje de programación Java

## ➤ Lenguaje orientado a objetos

### ➤ Clases

#### ➤ Herencia

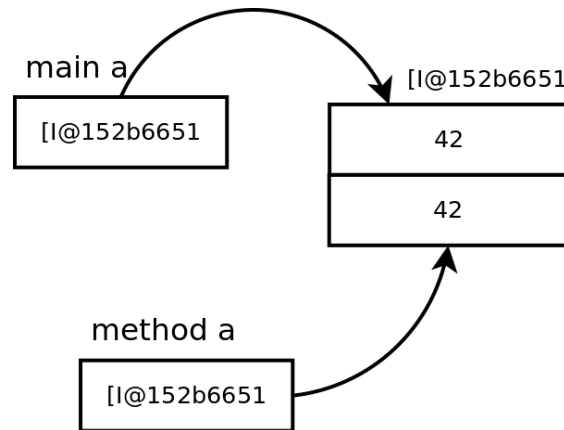
#### ➤ Interfaces

### ➤ Objetos

### ➤ Métodos

### ➤ Parámetros por referencia

### ➤ Sintaxis



Cada vez que se escribe el nombre un objeto en un programa Java, se utiliza una referencia a la zona de memoria donde está alojado el objeto, y no una copia del mismo.

# EL lenguaje de programación Java

## ➤ Lenguaje orientado a objetos

### ➤ Clases

#### ➤ Herencia

#### ➤ Interfaces

### ➤ Objetos

### ➤ Métodos

### ➤ Parámetros por referencia

### ➤ Sintaxis

```
public class ClassName extends SuperClass {  
    private static final int CONSTANT= 0;  
    /* This comment may span multiple lines. */  
    private static int staticField= 0;  
    // This comment may span only this line  
    private String field= "zero";  
    // TASK: refactor  
    public int foo(int parameter) {  
        abstractMethod();  
        int local= 42*hashCode();  
        staticMethod();  
        return bar(local) + 24;  
    }  
}
```

Clases y métodos.

Dentro, sintaxis similar a C.

# EL lenguaje de programación Java

## ➤ Máquina Virtual de Java (JVM)

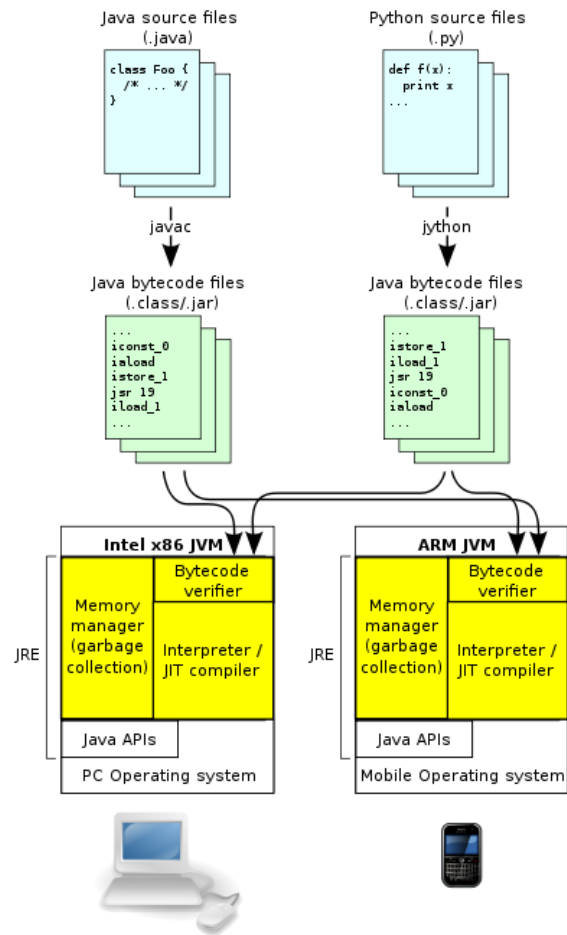
### ➤ Portabilidad

- Cualquier dispositivo
- Cualquier SO
- Java > ByteCode > Código ejecutable

### ➤ Menor rendimiento

- Etapas intermedias de ejecución
- Dificultad de programación MP/MC

### ➤ Aislamiento

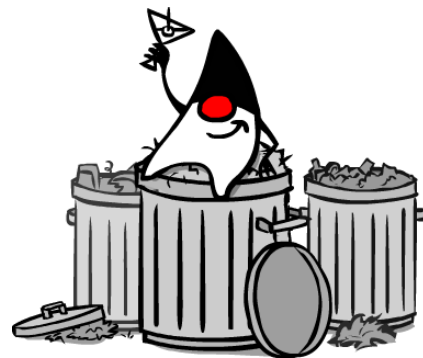




# EL lenguaje de programación Java

## ➤ Fácil de utilizar

- La comunidad Java dispone de innumerables paquetes con código listo para su reutilización
  - Java Beans
- Excepciones
- Sintaxis similar a C
- Entornos de programación IDE: NetBeans, Eclipse



## ➤ Gestión de memoria

- Garbage collector
  - El programador no se preocupa de liberar recursos una vez se ha terminado con ellos
  - Se almacena un número de referencias por objeto. Cuando llega a 0, el GC lo liberará la próxima vez que se ejecute

# EL lenguaje de programación Java

## ➤ Código existente y documentación

- Mucho código ya implementado
- Vale la pena buscar si lo que buscas ya existe
- Documentación: JavaDoc

➤ <https://docs.oracle.com/javase/7/docs/api/>



## ➤ También para Hadoop

➤ <https://hadoop.apache.org/docs/r2.6.1/api/org/apache/hadoop/mapred/>

# Caso de ejemplo: WordCount

Descargar ejemplo WordCount de moodle

```
package org.apache.hadoop.examples;
```

```
import java.io.IOException;
import java.util.StringTokenizer;

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.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 class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable> { ... }
```

```
    public static class IntSumReducer
        extends Reducer<Text, IntWritable, Text, IntWritable> { ... }
```

```
    public static void main(String[] args) throws Exception { ... }
```

```
}
```

Inclusión de paquetes necesarios

Definición de clase Mapper  
(herencia de clase)

Definición de clase Reducer  
(herencia de clase)

Configuración del entorno  
MapReduce

# Caso de ejemplo: WordCount

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length != 2) {
        System.err.println("Usage: wordcount <in> <out>");
        System.exit(2);
    }
    Job job = new Job(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokensMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

## Configuración del trabajo MR

- Configuración de parseo de los argumentos de entrada

- Clases Mapper, Combiner y Reducer

- Clases de las claves y valores de salida

- Configuración de los parámetros de entrada y salida

# Caso de ejemplo: WordCount

```
public static class TokenizerMapper
    extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
        ) throws IOException, InterruptedException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, one);
        }
    }
}
```

Separamos palabras de una misma línea, y las procesamos por separado

## Clase Mapper

La clase abstracta Mapper de la que hereda, fuerza tipo de argumentos de entrada, y la implementación del **método map**.

## Método map

### Entrada:

- Clave (en este caso no utilizada, pero contemplada para MR multi-etapa)
- Valor (cada línea de texto)

### Salida:

- Clave (palabra individual)
- Valor (en nuestro caso, 1)

# Caso de ejemplo: WordCount

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

    public void reduce(Text key, Iterable<IntWritable> values,
        Context context
        ) throws IOException, InterruptedException {

        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}
```

## Clase Reducer

La clase abstracta Reducer de la que hereda, fuerza tipo de argumentos de entrada, y la implementación del **método reduce**.

## Método reduce

**Entrada:** par clave, valor generado en la etapa map

**Salida:** nuevos pares clave, valor

- clave: la misma que antes
- valor: suma de todos los *valor* asociados a una misma clave.

# Caso de ejemplo: WordCount

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

    public void reduce(Text key, Iterable<IntWritable> values,
        Context context
        ) throws IOException, InterruptedException {

        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}
```

## Clase Combiner

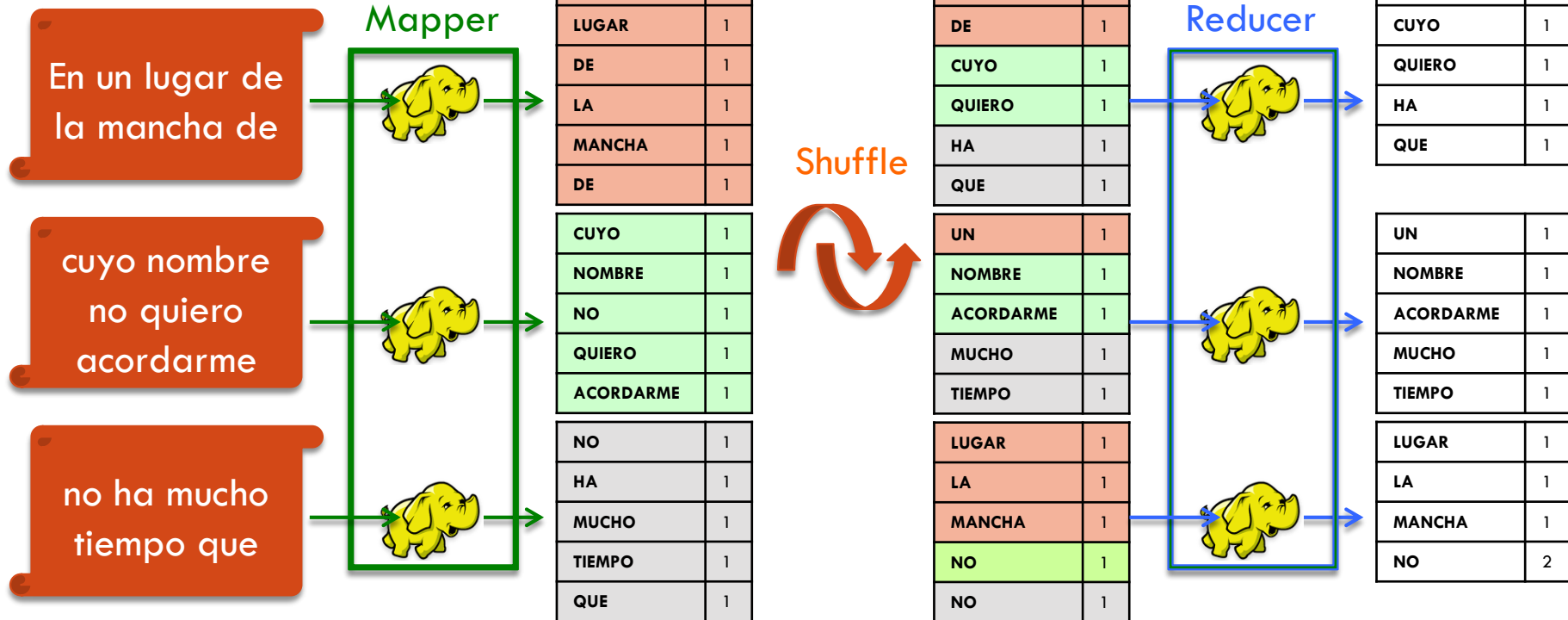
También implementa la interfaz Reducer. Reduce la cantidad de datos que se barajan para reducir movimiento de datos y “facilitar” las cosas al reducer. En este ejemplo las clases coinciden.

## Combiner vs Reducer

- Los formatos tanto de entrada como de salida deben coincidir con los de salida del Mapper
- Un Combiner toma sus datos de entrada de un solo Mapper
- Sólo pueden usarse con funciones conmutativas y asociativas (o alterarían el resultado)

# Caso de ejemplo: WordCount

## ➤ Mapper + Reducer





# Caso de ejemplo: WordCount

## ➤ Con Combiner

En un lugar de  
la mancha de

Mapper

EN	1
UN	1
LUGAR	1
DE	1
LA	1
MANCHA	1
DE	1

Comb.

EN	1
UN	1
LUGAR	1
DE	2
LA	1
MANCHA	1

Shuffle

EN	1
DE	2
CUYO	1
QUIERO	1
HA	1
QUE	1

Red.

EN	1
DE	2
CUYO	1
QUIERO	1
HA	1
QUE	1

cuyo nombre  
no quiero  
acordarme



CUYO	1
NOMBRE	1
NO	1
QUIERO	1
ACORDARM E	1
NO	1
HA	1
MUCHO	1
TIEMPO	1
QUE	1



CUYO	1
NOMBRE	1
NO	1
QUIERO	1
ACORDARM E	1
NO	1
HA	1
MUCHO	1
TIEMPO	1
QUE	1



UN	1
NOMBRE	1
ACORDARM E	1
MUCHO	1
TIEMPO	1
LUGAR	1
LA	1
MANCHA	1
NO	1
NO	1



UN	1
NOMBRE	1
ACORDARM E	1
MUCHO	1
TIEMPO	1
LUGAR	1
LA	1
MANCHA	1
NO	2

no ha mucho  
tiempo que



CUYO	1
NOMBRE	1
NO	1
QUIERO	1
ACORDARM E	1
NO	1
HA	1
MUCHO	1
TIEMPO	1
QUE	1



CUYO	1
NOMBRE	1
NO	1
QUIERO	1
ACORDARM E	1
NO	1
HA	1
MUCHO	1
TIEMPO	1
QUE	1



UN	1
NOMBRE	1
ACORDARM E	1
MUCHO	1
TIEMPO	1
LUGAR	1
LA	1
MANCHA	1
NO	1
NO	1



UN	1
NOMBRE	1
ACORDARM E	1
MUCHO	1
TIEMPO	1
LUGAR	1
LA	1
MANCHA	1
NO	2

# Caso de ejemplo: WordCount

## ➤ Ejemplo de salida

(Y	1
(a	1
(al	1
(como	1
(creyendo	1
(de	2

...

Anoche	1
Anteo,	1
Antequera	1
Antequera.	1
Antequera;	1
Antes	1

...

Has	2
Haz	1
He	1
Hechas,	2
Hecho	5
Hechos	1
Henares	1
Henares,	1
Hermandad	2
Hermandad,	1

...

has	21
has,	1
hasme	1
hasta	53
hato	1
hato.	1
hay	52
hay,	2

...

# Ejercicio

- Vamos a modificar el ejemplo de WordCount que hemos tomado como partida, para que no tenga en cuenta signos de puntuación, ni las mayúsculas/minúsculas
- Guión de prácticas
  - Crear una aplicación Hadoop en Java

# Instalar entorno JDK de java

## ➤ Instalar el entorno de desarrollo java

➤ `> yum install java-1.7.0-openjdk-devel.x86_64`

## ➤ Comprobar que funciona

➤ `> javac -version`

## ➤ Compilar ejemplo WordCount en java

➤ Descargue de moodle el fichero fuente WordCount.java y coloquelo en el directorio /opt/work

➤ `Cd /opt/work`

➤ `> javac -classpath  
/opt/hadoop/share/hadoop/common/*:/opt/hadoop/share/hadoop/mapreduce/  
* -d WordCount WordCount.java`

## ➤ Crear Jar

➤ `jar -cvf WordCount.jar -C WordCount/ .`

# Instalar entorno JDK de java

- Descarga el fichero quijote.txt de moodle y colóquelo en un directorio de trabajo y vea su contenido
  - `Cat /opt/work/quijote.txt`
- Ejecución en Hadoop de WordCount
  - `> bin/hadoop jar /opt/work/WordCount.jar uam.WordCount /opt/work/quijote.txt /opt/work/salida1`
- Revise la salida.

# Información adicional

- Parámetros de configuración de una tarea hadoop
  - Al lanzar la tarea:
    - `hadoop jar /.../hadoop-examples.jar terasort -Dmapred.map.tasks=XX -Dmapred.reduce.tasks=XX terasort-input/ terasort-output/`
    - `... -Dmapreduce.map.memory.mb=4096 ...`
  - Desde el código:
    - `Job.setNumReduceTasks(int)`
  - Configuración por defecto de los servicios hadoop
    - `cat /usr/iop/current/hadoop-yarn-client/etc/hadoop/mapred-site.xml`
  - <https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html>

# Información adicional

## ➤ Encadenar trabajos

### ➤ Dependencias entre tareas:

➤ `x.addDependingJob(y)`

➤ <https://hadoop.apache.org/docs/r2.6.1/api/org/apache/hadoop/mapred/jobcontrol/Job.html>

➤ <https://developer.yahoo.com/hadoop/tutorial/module4.html#chaining>