



MBIT School
Madrid Business Intelligence Technology

La API Java MapReduce

Una breve introducción

Diego J. Bodas Sagi



Índice

- El entorno
- Configuración del proyecto
- Librerías MapReduce
- Introducción a la Java API
- Primeros ejemplos
- Resumen





Entorno recomendado

- Eclipse es uno de los IDEs más recomendados para programar en Java. También está preparado para trabajar con Hadoop
- Recomendamos el uso de Maven. Maven es una herramienta open source para administrar proyectos de software. Permite gestionar el ciclo de vida desde la creación de un proyecto en un lenguaje dado, hasta la generación de un binario que pueda distribuirse con el proyecto





Descargar código fuente

- ❶ `$ git clone https://github.com/diegobodas/api_samples`
- ❷ `$ git clone https://github.com/diegobodas/resources`
- ❸ `$ git clone https://github.com/diegobodas/authors`





Procedimiento

 Seguiendo instrucciones de clase.....





Maven

- Maven es un complemento al compilador Java Ant (uno de los más usados en Java) que proporciona una estructura consistente de proyectos (todos los proyectos Maven tienen por defecto los mismos directorios)
- También proporciona herramientas para gestionar la complejidad de los proyectos de software: gestión avanzada de dependencias, informes sobre pruebas...





Configuración del proyecto Maven

- Archivo pom.xml: en este archivo se describen las librerías necesarias, el orden de compilación ...
- Podemos encontrar un ejemplo de este archivo de configuración en: <https://gist.github.com/jnatkins/3517129>
- La ejecución de un archivo POM siempre genera un "**artefacto**"
- Maven trabaja modularizando los proyectos. De esta forma tendremos varios módulos que conforman un sólo proyecto
- Para denotar esta relación en Maven, se crea un proyecto **padre** de tipo POM (esto es que no genera un binario en sí) y los módulos se definen como otros archivos pom que heredan del primero
- Lo anterior permite centralizar en el pom padre las variables (como el nombre del proyecto o el número de versión), las dependencias, los repositorios, etc. que son comunes a los módulos, eliminando duplicidad de código





Crear un proyecto Maven

● Maven Archetypes

- Los **arquetipos** son artefactos especiales de Maven que sirven como plantillas para crear proyectos
- Maven cuenta con algunos predefinidos y terceros han hecho los suyos para crear proyectos con tecnologías específicas, como es el caso de la plantilla anterior
- Desde el directorio en el que queramos crear el proyecto:
 - *mvn archetype:generate*
 - *Pero ¡salen más de 300 opciones! Muy incómodo*





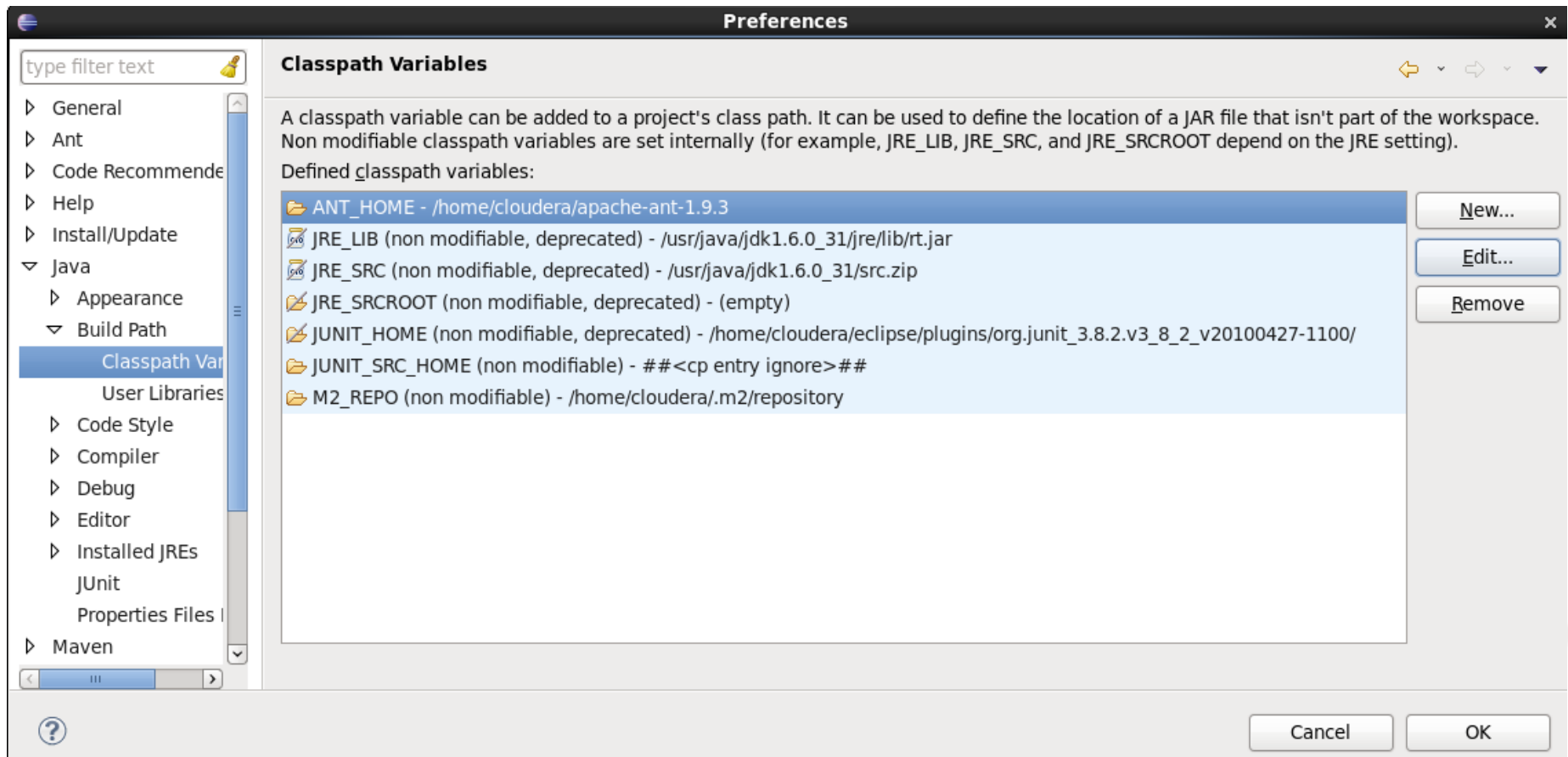
Configuración de Eclipse

- Actualizar software nada más empezar
- Añadir m2eclipse Plugin
 - <http://download.eclipse.org/technology/m2e/releases>
- Descargar ANT y configurar la variable ANT_HOME





Class Paths





Hadoop necesita tools.jar

type filter text

General

Ant

Code Recommender

Help

Install/Update

Java

Appearance

Build Path

Classpath Variables

User Libraries

Code Style

Compiler

Debug

Editor

Installed JREs

JUnit

Properties Files

Maven

Installed JREs

Add, remove or edit JRE definitions. By default, the build path of newly created Java projects uses the default JRE.

Installed JREs:

Name	Location
<input checked="" type="checkbox"/> jdk1.6.0_31	/usr/java/jdk1.6.0_31

JRE Definition

Specify attributes for a JRE

JRE home:

/usr/java/jdk1.6.0_31

Directory...

JRE name:

jdk1.6.0_31

Default VM arguments:

Variables...

JRE system libraries:

/usr/java/jdk1.6.0_31/jre/lib/resources.jar

/usr/java/jdk1.6.0_31/jre/lib/rt.jar

/usr/java/jdk1.6.0_31/jre/lib/jsse.jar

/usr/java/jdk1.6.0_31/jre/lib/jce.jar

/usr/java/jdk1.6.0_31/jre/lib/charsets.jar

/usr/java/jdk1.6.0_31/jre/lib/ext/localedata.jar

/usr/java/jdk1.6.0_31/jre/lib/ext/sunpkcs11.jar

/usr/java/jdk1.6.0_31/jre/lib/ext/sunjce_provider.jar

/usr/java/jdk1.6.0_31/jre/lib/ext/dnsns.jar

/usr/java/jdk1.6.0_31/lib/tools.jar

Add External JARs...

Javadoc Location...

Source Attachment...

Remove

Up

Down

Restore Default





Formato de código en Hadoop

- Window -> Preferences.
- Java->Code Style -> Formatter.
- Importar archivo de referencia
- Buenas prácticas
 - Window->Preferences->Java->Editor->Save Actions
 - Seleccionar “Perform the selected actions on save”, “Format source code”, “Format edited lines”.
 - NO seleccionar “Organize imports”
 - Riesgo de colisión con la API vieja, mejor a mano





Problemas con el pom.xml

- Debe examinarse cada problema de forma particular
- El manejo no es obvio
- Aunque compensa debido al potencial de reutilización
- ¡Mucho cuidado con las versiones reflejadas en el archivo pom.xml!



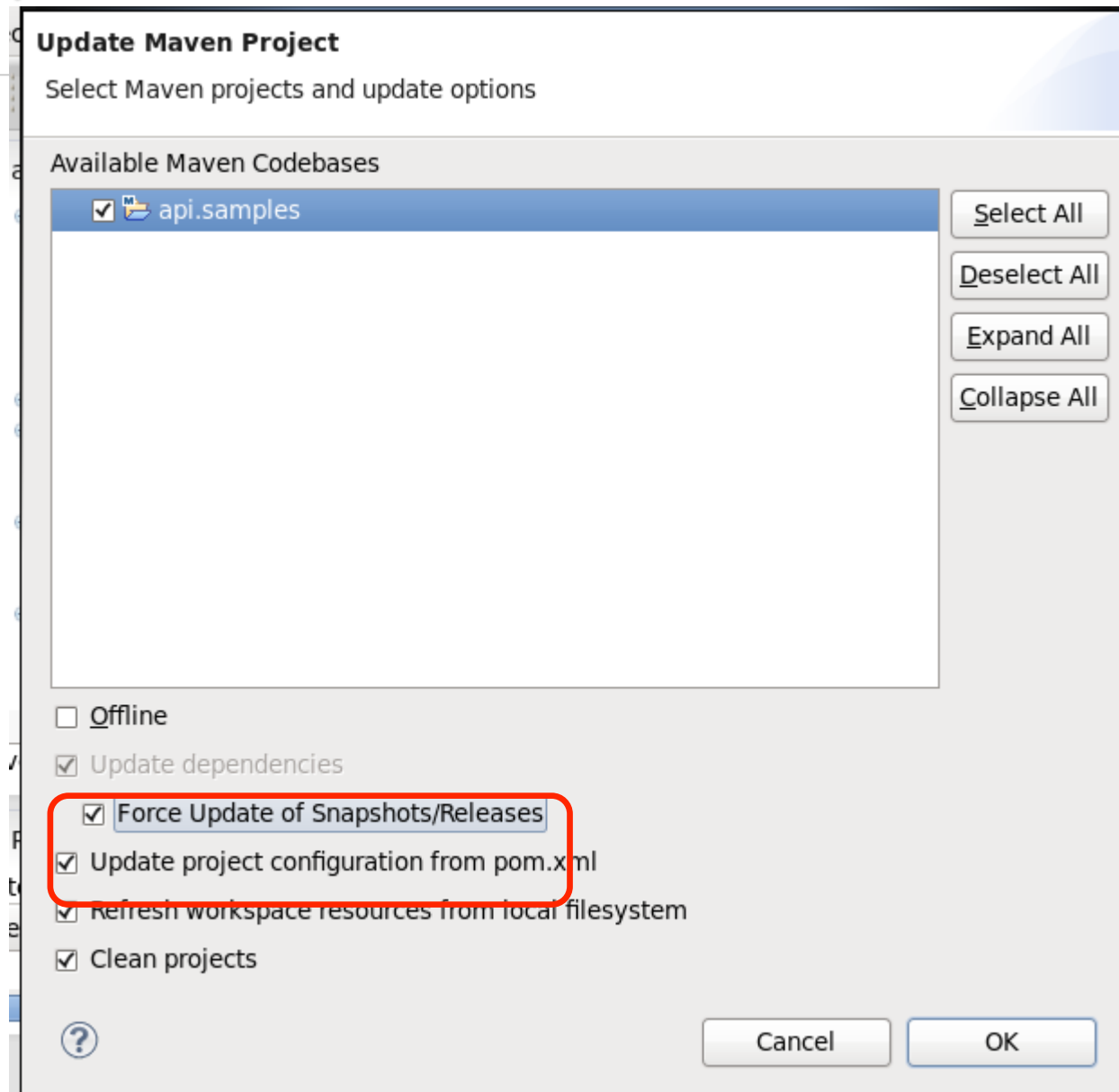


Versiones

- A veces, arreglar un error es tan sencillo como consultar (en Cloudera por ejemplo) y poner la versión adecuada
 - http://www.cloudera.com/content/cloudera-content/cloudera-docs/CDH4/latest/CDH-Version-and-Packaging-Information/cdhvd_topic_8.html
- Desde la consola
 - “\$ hadoop version”

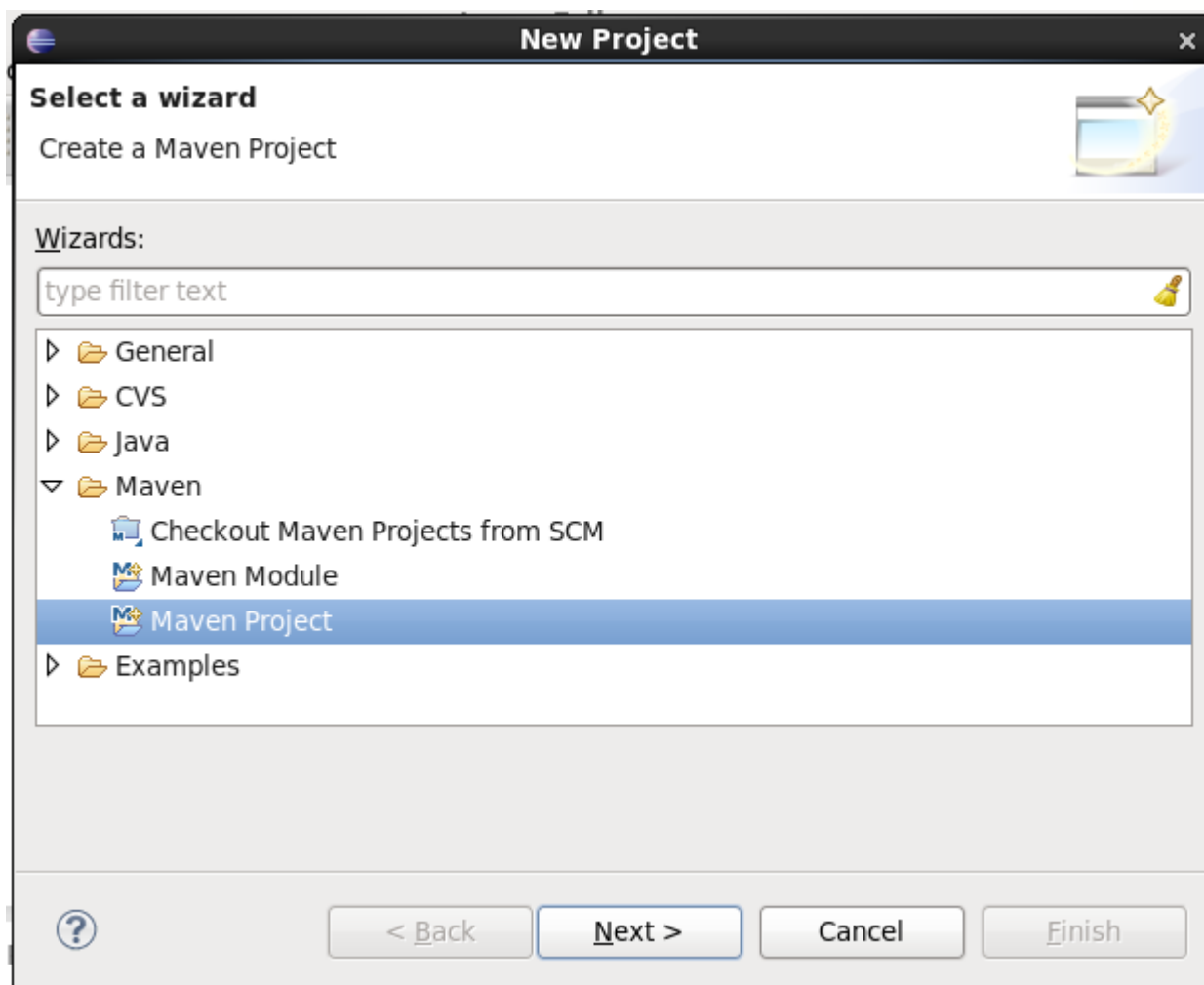


En el caso de problemas ...





Generando el proyecto Maven





Se necesita...

- Desde eclipse se debe modificar el archivo pom.xml añadiendo la información de versiones de java (1.6 por ejemplo) y repositorio
- Importante también las dependencias con hadoop core y hadoop client (MapReduce y APIs)
- Cada vez que se modifique el archivo de configuración pom.xml hay que actualizar el proyecto (opciones del menú del botón derecho del ratón)






Paso a paso

- El proyecto anterior que hemos creado lo vamos a dejar para cuestiones más complicadas
- Para realizar los primeros ejemplos con la API Hadoop para Java vamos a empezar paso a paso con proyectos muy sencillos
- Procedimiento 2: importar proyectos facilitados por el profesor



New Maven Project

Configure project



Artifact

Group Id:

Artifact Id:

Version:

Packaging:

Name:

Description:


Parent Project

Group Id:

Artifact Id:

Version:

► **Advanced**





¡Cuidado!

- Está intentando usar la versión 1.5 de Java
- Debemos completar añadiendo el código respectivo en el archivo de configuración pom.xml
- Se añade desde eclipse
 - Seleccionando la pestaña de pom.xml porque es más fácil trabajar con el fichero fuente





Añadido

- Se añade la parte de maven-compiler del pom anterior (se especifica 1.6 como la versión correcta de java)
- Se añade la información del repositorio de cloudera (donde están todas las librerías)
- Se añaden las dependencias que tenemos
 - Para este ejemplo, sólo hadoop-common y hadoop-client
- Fijar las propiedades que sean necesarias



Actualizar proyecto



Java - api.samples/pom.xml - Eclipse

File Edit Source

Package Explorer

api.samples

- src/main
- src/test
- JRE System
- Maven
- src
- target
- pom.xml
- maven-hadoop
- training

Context Menu:

- New
- Go Into
- Open in New Window
- Open Type Hierarchy F4
- Show In Shift+Alt+W
- Copy Ctrl+C
- Copy Qualified Name
- Paste Ctrl+V
- Delete Delete
- Remove from Context Shift+Ctrl+Alt+Down
- Build Path
- Source Shift+Alt+S
- Refactor Shift+Alt+T
- Import...
- Export...
- Refresh F5
- Close Project
- Assign Working Sets...
- Run As
- Debug As
- Validate
- Team
- Compare With
- Restore from Local History...
- Maven

Right-click context menu options:

- Add Dependency
- Add Plugin
- New Maven Module Project
- Download JavaDoc
- Download Sources
- Update Project... Alt+F5
- Disable Workspace Resolution

XML Content:

```
<cdh-version></version>
<group>org.apache.hadoop</group>
<artifactId>hadoop-client</artifactId>
<cdh-version></version>
<id>cloudera-releases</id>
<url>https://repository.cloudera.com/artifactory/cloudera-releases</url>
<enabled>true</enabled>
<enabled>false</enabled>
<scope>test</scope>
<scope>runtime</scope>
<scope>test</scope>
```





Listos para trabajar

- Si observamos que la librería Java que aparece es la 1.6 y
- No hay errores o avisos
- ¡Ya estamos listos para trabajar y hemos descubierto otra forma sencilla de empezar!





Librerías MapReduce

- Hadoop proporciona dos APIs MapReduce para Java.
 - La vieja (y depreciada): `org.apache.hadoop.mapred`
 - La nueva: `org.apache.hadoop.mapreduce`
- La nueva API es más sencilla y funcional
 - Algunas funcionalidades no están cubiertas en las versiones más antiguas de Hadoop
 - En cuestiones genéricas y comunes, no debe presentar mayores problemas para integrarse con MRv1



Java API

`org.apache.hadoop.mapred`

- Depreciada
- MRv1

`org.apache.hadoop.mapreduce`

- A partir de la versión 0.20.x
- Mejor organizada
- Más flexible
- Orientada a la evolución de MRv2, pero se puede emplear para MRv1



Pasos

 Comentar aquí ligeramente los pasos que seguiremos

- Mapper
- Driver
- Reducer
- ...





Un Mapper

- Pares <clave, valor>
- Necesitamos especificar los tipos tanto de las claves como de los valores
- Creamos un MapperIdentity en main/java
 - Este Mapper escribe lo mismo que recibe
- Paquete com.api.samples
- Hereda de Mapper
- Asegurarse de coger el mapper de
 - org.apache.hadoop.mapreduce





Configurando el mapper

- Configurar los tipos de las claves y valores de entrada y las claves y valores de salida
- Los tipos disponibles se llaman **writables**
 - BooleanWritable
 - ByteWritable
 - BytesWritable (sonido, imagen, pdf...)
 - DoubleWritable
 - FloatWritable
 - IntWritable
 - LongWritable
 - NullWritable (NULLs, si queremos que no sea seriarizable)
 - ShortWritable
 - Text





Configurando el mapper

- Vamos a añadir, y en este orden:
 - `<LongWritable, Text, Text, NullWritable>`
 - LongWritable: la clave, el “byte offset” del principio de la línea
 - Text: el valor, el contenido de la línea
 - La clave de la salida es el propio valor
- Nos servimos de la ayuda de eclipse para importar las correspondientes librerías
 - Importamos de `hadoop.io`
- Buscamos ahora los métodos de la clase “padre” Mapper para implementarlos (Override/Implement Methods)





Métodos de la clase Mapper

- Marcan el ciclo de vida de la clase:
 - Setup: se ejecuta cuando se está creando el objeto mapper
 - Map: mapea, se ejecuta para cada entrada (ej: línea de texto) que tenga que procesar el mapper
 - Cleanup: se ejecuta al final del proceso para liberar lo que sea necesario (por ejemplo, eliminar un objeto que hayamos creado en el setup)
 - En esta práctica sólo vamos a emplear el map
 - El último parámetro se puede dejar como
 - Context context: no hace falta el nombre tan largo que propone eclipse





Añadiendo contenido al Mapper

- El CONTEXTO es esencial y es lo que utilizaremos para “transmitir” la información que deseamos
- En este mapper tan sencillo, simplemente cogeremos el texto que recibimos de valor de entrada y lo pondremos como clave de salida
- El valor de salida será (como hemos especificado) un valor `NullWritable`
 - `NullWritable.get()`





MapperIdentity

```
package com.utad.api.samples;

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class MapperIdentity extends
    Mapper<LongWritable, Text, Text, NullWritable> {

    /*
     * (non-Javadoc)
     *
     * @see org.apache.hadoop.mapreduce.Mapper#map(java.lang.Object,
     * java.lang.Object, org.apache.hadoop.mapreduce.Mapper.Context)
     */
    @Override
    protected void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        context.write(value, NullWritable.get());
    }
}
```





Driver

- Todo proceso MapReduce en Hadoop requiere un controlador responsable de gestionar el proceso
- Es habitual llamar a dicho controlador Driver o Job (son los nombres más empleados)
- Esta clase hereda de la clase Configured
 - `org.apache.hadoop.conf`
- Implementa la interfaz Tool
 - `org.apache.hadoop.util`
- Entre sus misiones encontramos:
 - Cargar la configuración
 - Definir los ficheros de entrada y salida
 - Definir el formato de estos ficheros
 - ...





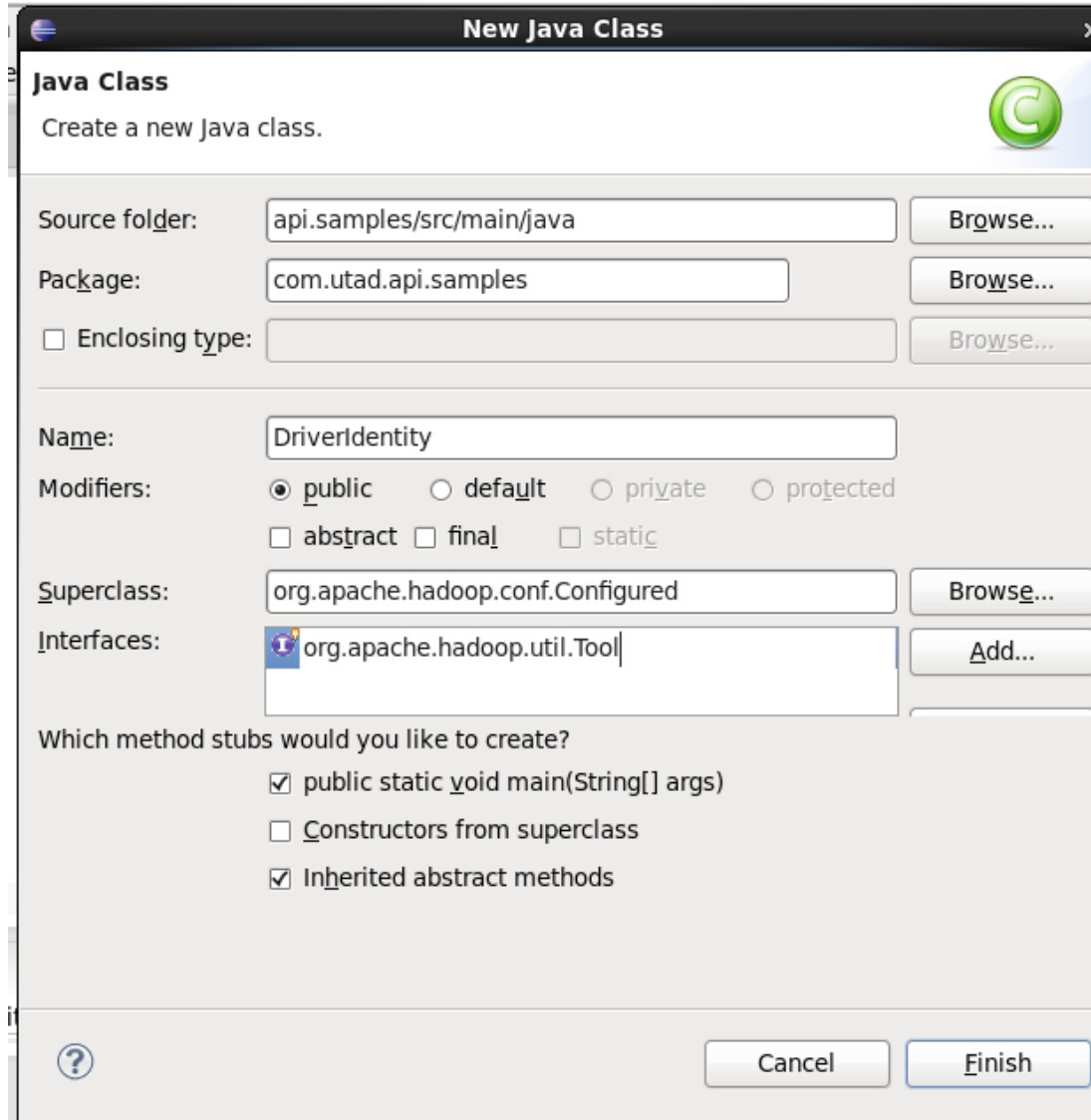
Pasos para generar el Driver

- Generamos una nueva clase en la misma ruta que la clase MapperIdentity
- Esta clase se llamará DriverIdentity
- Indicaremos que la superclase será Configured
- Además implementa la interfaz Tool
- Por último indicamos que queremos tener un método “main”





Configuración del Driver



New Java Class

Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ default ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☒ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods





Codificando el método Main

- Primero obtenemos la configuración de Hadoop a través de la clase Configuration
 - Se importa de: `org.apache.hadoop.conf`
 - `Configuration conf = new Configuration (true);`
- Ejecutaremos nuestro programa MapReduce usando la clase ToolRunner
 - `ToolRunner.run(conf, new IdentityDriver (), args);`
 - `conf`: la configuración creada anteriormente
 - `new IdentityDriver()`: una instancia de nuestro Driver
 - `args`: los mismos argumentos que leemos de la línea de comandos. Estos mismos argumentos se pasan al método `run`
 - Se completa el código siguiendo, por ejemplo, las recomendaciones de Eclipse (`throws exception`)





Main - primera parte

```
public static void main(String[] args) throws Exception {  
    Configuration conf = new Configuration(true);  
    ToolRunner.run(conf, new DriverIdentity(), args);  
}
```





Main - run

1. Obtenemos la Configuración
2. Creamos un Job indicando la configuración que queremos emplear
3. Ahora necesitamos pasar el código ejecutable de nuestro MapReduce, dos opciones:
 - a. Pasar el archivo Jar (setJar)
 - b. Os pasar la clase (setJarByClass)
4. Indicamos cuál es la clase Map que vamos a emplear (¡ojo!, ya viene un Identity Mapper en Hadoop pero queremos el nuestro)





Main – run (II)

5. Configuramos los ficheros de entrada con los que queremos trabajar (si se añade un directorio, por defecto, coge todos los ficheros del directorio)
 - a. Indicamos el formato (por ejemplo texto plano, porque en nuestro mapper hemos puesto que los valores que recibimos son textos)
 - b. Especificamos los directorios de entrada (puede haber varios, haciendo varias llamadas)
 - Empleamos `hadoop.fs`





Main – run (III)

6. ¿Cómo vamos a generar la salida del procesamiento de los ficheros de entrada?
 - a. En este caso, la clave de salida indicada en texto y el valor Null
7. ¿Dónde vamos a depositar la salida?
 - a. Especificamos un formato texto estándar (TextOutputFormat)
 - Escribe toda la salida (claves y valores como texto)
 - b. Indicamos el directorio de salida (sólo puede haber uno)
 - c. ¿Qué formato tendrá la clave y valor de la salida?
 - Volvemos a consultar el mapper y ponemos la clave de texto y el valor Null
8. Por último, la orden de ejecución
 - a. ***return job.waitForCompletion(true) ? 0 : 1***
 - b. El parámetro booleano indica si queremos esperar o no a que termine el proceso. ¡Pero el método “run” espera un entero





public int run(...)

DriverIdentity.java

```
Configuration conf = getConf();  
// Job configuration  
Job job = Job.getInstance(conf);  
// Our class  
job.setJarByClass(DriverIdentity.class);  
// The Mapper  
job.setMapperClass(MapperIdentity.class);  
  
// The input file  
job.setInputFormatClass(TextInputFormat.class);  
// We suppose that the input path is introduced in command line (first  
// place)  
FileInputFormat.addInputPath(job, new Path(args[0]));  
// The output --- Check the Mapper  
job.setMapOutputKeyClass(Text.class);  
job.setMapOutputValueClass(NullWritable.class);  
  
// Output File  
job.setOutputFormatClass(TextOutputFormat.class);  
// We suppose that the input path is introduced in command line (2nd place)  
FileOutputFormat.setOutputPath(job, new Path(args[1]));  
job.setOutputKeyClass(Text.class);  
job.setOutputValueClass(NullWritable.class);  
  
return job.waitForCompletion(true) ? 0 : 1;
```





Ejecutando nuestro 1er ejemplo

- Debemos crear una configuración de ejecución nueva (“Run Configurations”)
- Esta aplicación Hadoop es una aplicación Java
- Como clase principal, coge por defecto donde tenemos el método “main”
- En la pestaña de “argumentos” añadimos los parámetros necesarios
 - Añadimos una ruta relativa al proyecto actual “\${project_loc}”
 - En el directorio inputFiles y pasamos el archivo fileinput.txt
 - Y como fichero de salida, en el directorio outputFiles
`${project_loc}/inputFiles/fileinput.txt`
`${project_loc}/outputFiles/`





Culminando el ejemplo

- Añadimos al directorio inputFiles un fichero de entrada
- Para una mejor gestión, podemos asegurarnos de que aparece en la ruta del proyecto en eclipse (refrescar el proyecto)
- Ejecutar la configuración que hemos creado
- Observar los resultados
 - La salida siempre es una carpeta
 - Además hay información de log



Java - api.samples/outputFiles/fileoutput.txt/part-r-00000 - Eclipse

File Edit Navigate Search Project Run Window Help

Quick Access Java

Package Explorer

- api.samples
 - src/main/java
 - src/main/resources
 - src/test/java
 - src/test/resources
 - Maven Dependencies
 - JRE System Library [JavaSE-1.6]
 - inputFiles
 - outputFiles
 - fileoutput.txt
 - _SUCCESS
 - part-r-00000
 - src
 - target
 - pom.xml
 - maven-hadoopCDH-project
 - training

DriverIdentity.java part-r-00000

En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo
 Es, pues, de saber, que este sobredicho hidalgo, los ratos que estaba ocioso (que eran los más del año) se
 allí se promete; y sin duda alguna lo hiciera, y aun saliera con ello, si otros mayores y continuos pensamientos no
 aunque por conjeturas verosímiles se deja entender que se llama Quijana; pero esto importa poco a nuestro
 caballero el juicio, y desvelábase por entenderlas, y desentrañarles el sentido, que no se lo sacara, ni las
 cuento; basta que en la narración del no se salga un punto de la verdad.
 daba a leer libros de caballerías con tanta afición y gusto, que olvidó casi de todo punto el ejercicio de
 daba y recibía, porque se imaginaba que por grandes maestros que le hubiesen curado, no dejaría de tener el rostro
 de Silva: porque la claridad de su prosa, y aquellas intrincadas razones suyas, le parecían de perlas; y más cuando
 de aquella inacabable aventura, y muchas veces le vino deseo de tomar la pluma, y darle fin al pie de la letra como
 de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor. Una olla de algo más vaca
 entendiera el mismo Aristóteles, si resucitara para sólo ello. No estaba muy bien con las heridas que don Belianis
 hacen merecedora del merecimiento que merece la vuestra grandeza. Con estas y semejantes razones perdía el pobre
 la caza, y aun la administración de su hacienda; y llegó a tanto su curiosidad y desatino en esto, que vendió
 llegaba a leer aquellos requiebros y cartas de desafío, donde en muchas partes hallaba escrito: la razón de la
 muchas hanegas de tierra de sembradura, para comprar libros de caballerías en que leer; y así llevó a su casa
 palomino de añadidura los domingos, consumían las tres partes de su hacienda. El resto della concluían
 que carnero, salpicón las más noches, duelos y quebrantos los sábados, lentejas los viernes, algún
 sayo de velarte, calzas de velludo para las fiestas con sus pantuflos de lo mismo, los días de entre
 se lo estorbaran.
 seco de carnes, enjuto de rostro; gran madrugador y amigo de la caza. Quieren decir que tenía el
 semana se honraba con su vellori de lo más fino. Tenía en su casa una ama que pasaba de los cuarenta,
 sinrazón que a mi razón se hace, de tal manera mi razón enflaquece, que con razón me quejo de la vuestra fermosura,
 sobrenombre de Quijada o Quesada (que en esto hay alguna diferencia en los autores que deste caso escriben),
 todos cuantos pudo haber dellos; y de todos ningunos le parecían tan bien como los que compuso el famoso Feliciano
 tampa la podadera. Escriba la edad de nuestro hidalgo con las cincuenta años. Era de complexión recia

Problems @ Javadoc Declaration Console

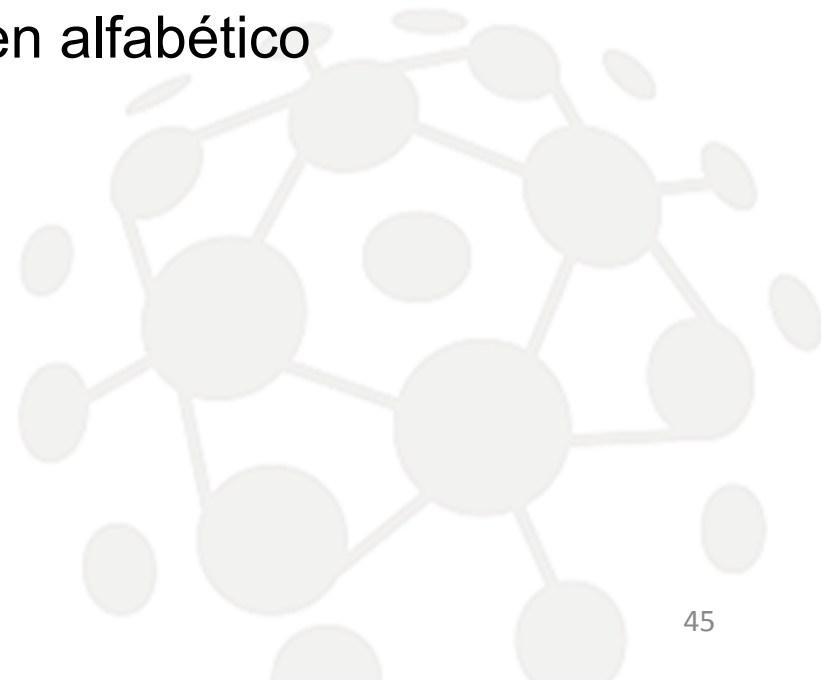
Writable Insert 1:1 Building workspace: (100%)

Java - api.samples/out...



¿Qué hemos hecho?

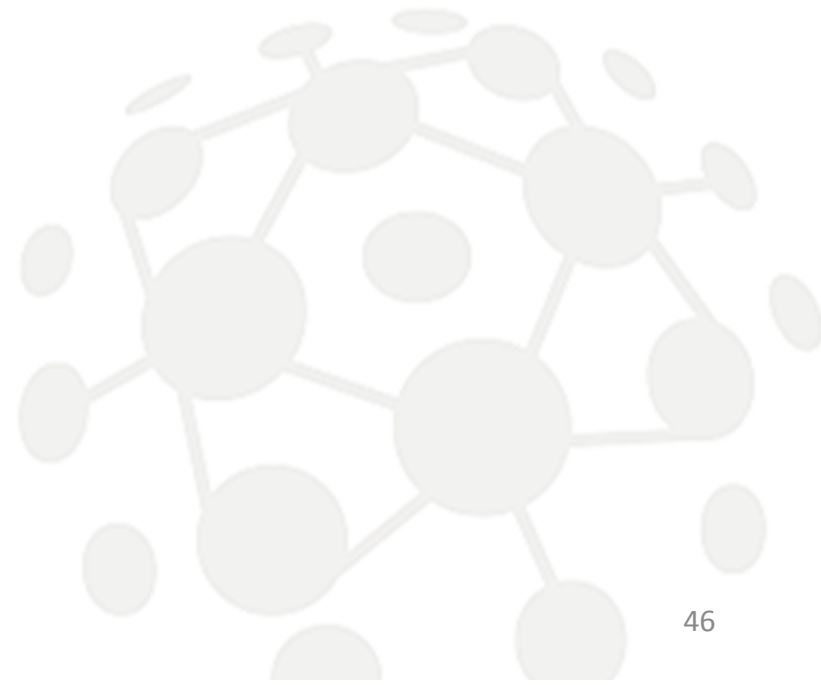
- El fichero de salida tiene las mismas filas que el de entrada
- Pero el orden NO es el mismo
- MapReduce procesa todas las filas del archivo original
- Pero debido a las mezclas internas de MapReduce **no** se conserva el orden original
- El resultado mostrado está en orden alfabético





Todavía hay cosas que corregir

- Warning: el objeto logger (encargado de configurar información del proceso no está bien configurado)
- Para ello y en main/resources debemos crear un fichero *log4j.properties* (estamos empleando la librería log4j)
 - Estos ejemplos de configuración se pueden buscar de forma sencilla por internet





log4j.properties

Root logger option

#--- qué tipo de mensajes quiero que salgan en el log y por dónde

informativos, por la salida estándar stdout

log4j.rootLogger=INFO, stdout

Direct log messages to stdout

#¿Cuál es la salida estándar?

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.Target=System.out

#Formateo de los mensajes

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss}

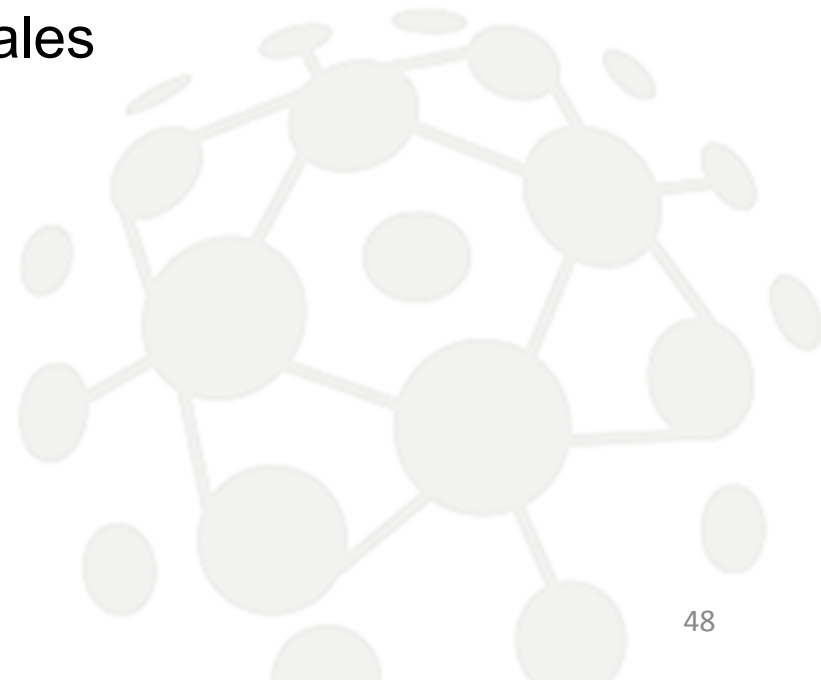
%-5p %c{1}:%L - %m%n





Nueva ejecución

- Borrار el fichero de salida
 - En caso contrario puede dar error
 - A veces se tarda mucho tiempo en ejecutar un proceso
 - Hadoop “nos protege” del posible error de sobrescribir un trabajo realizado por descuido
- Podemos ojear las estadísticas finales





Depurando programas MapReduce

Varias opciones

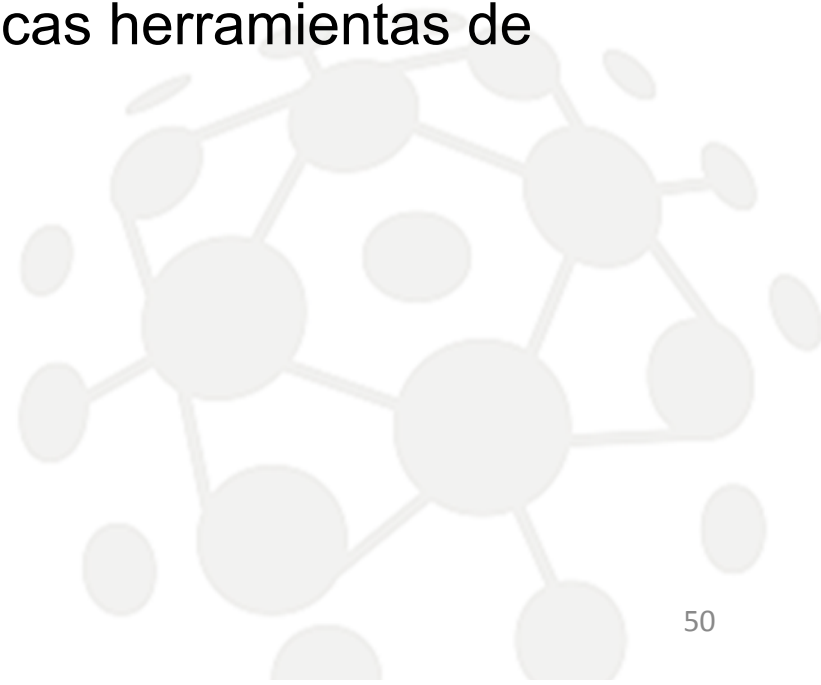
1. Acudir a los típicos mensajes que se imprimen en la consola para controlar cómo funciona el proceso
2. Añadir un punto de interrupción
 - a. Ejemplo: ponemos un punto de interrupción en la línea de código del mapper
 - b. Volvemos a borrar el fichero de salida generado
 - c. Lo lanzamos (con el Debug) pasando a un tipo de ventana Debug





Examinando variables

- En la ventana de variables podemos ir examinando los valores que van recogiendo los pares <key, value>
- Es habitual poner la vista “sólo de variables”
- Entre los valores, encontramos bytes que contienen el fichero de texto
- Tenemos a nuestro alcance las típicas herramientas de depuración de eclipse





Añadiendo un Reducer

- Reducer <Input Key, Input Value, Output Key, Output Value>
 - Output Value: realmente es una lista de valores
- Creamos la clase ReducerIdentity
 - Hereda de Reducer (org.apache.hadoop.mapreduce)
 - En este ejemplo tan sencillo, generamos la misma entrada que recibimos
 - Recibimos un par (Text, NullWritable) porque eso es lo que genera el Mapper
 - Devolvemos lo mismo





Métodos del Reducer

- Marcan el ciclo de vida de la clase:
 - Setup: se ejecuta cuando se está creando el objeto reducer
 - Reduce: encargado de recoger y procesar los resultados del mapper
 - Cleanup: se ejecuta al final del proceso para liberar lo que sea necesario (por ejemplo, eliminar un objeto que hayamos creado en el setup)
 - En esta práctica sólo vamos a emplear el método reduce
 - El último parámetro se puede dejar como
 - Context context: no hace falta el nombre tan largo que propone eclipse





Nuestro primer “reduce”

- Muy sencillo. Descartamos las claves y devolvemos todos los valores que hemos recibido

```
for (Writable null_writable: values)  
    context.write(key, null_writable);
```





Añadiendo el Reduce DriverIdentity

- Justo después de donde aparece la información del mapper añadimos la configuración del reducer
 - `job.setReducerClass (ReducerIdentity.class);`
- Borramos el fichero de salida y ejecutamos
- Al editar el fichero de salida, comprobamos que vuelven a aparecer las claves (texto) emitido por el mapper





Distribuyendo la solución

- Generando un archivo “jar”
 - Exportando el proyecto desde eclipse
 - Export Java Jar
 - Indicamos dónde lo queremos guardar
- Empleando el plugin de Maven
 - Run As → Maven Build
 - Goals → package
 - Nos genera un Jar porque así se lo hemos indicado en las opciones de empaquetado de archivo pom.xml
 - En nuestro caso, nos aparece en el mismo directorio del proyecto dentro de “Target”





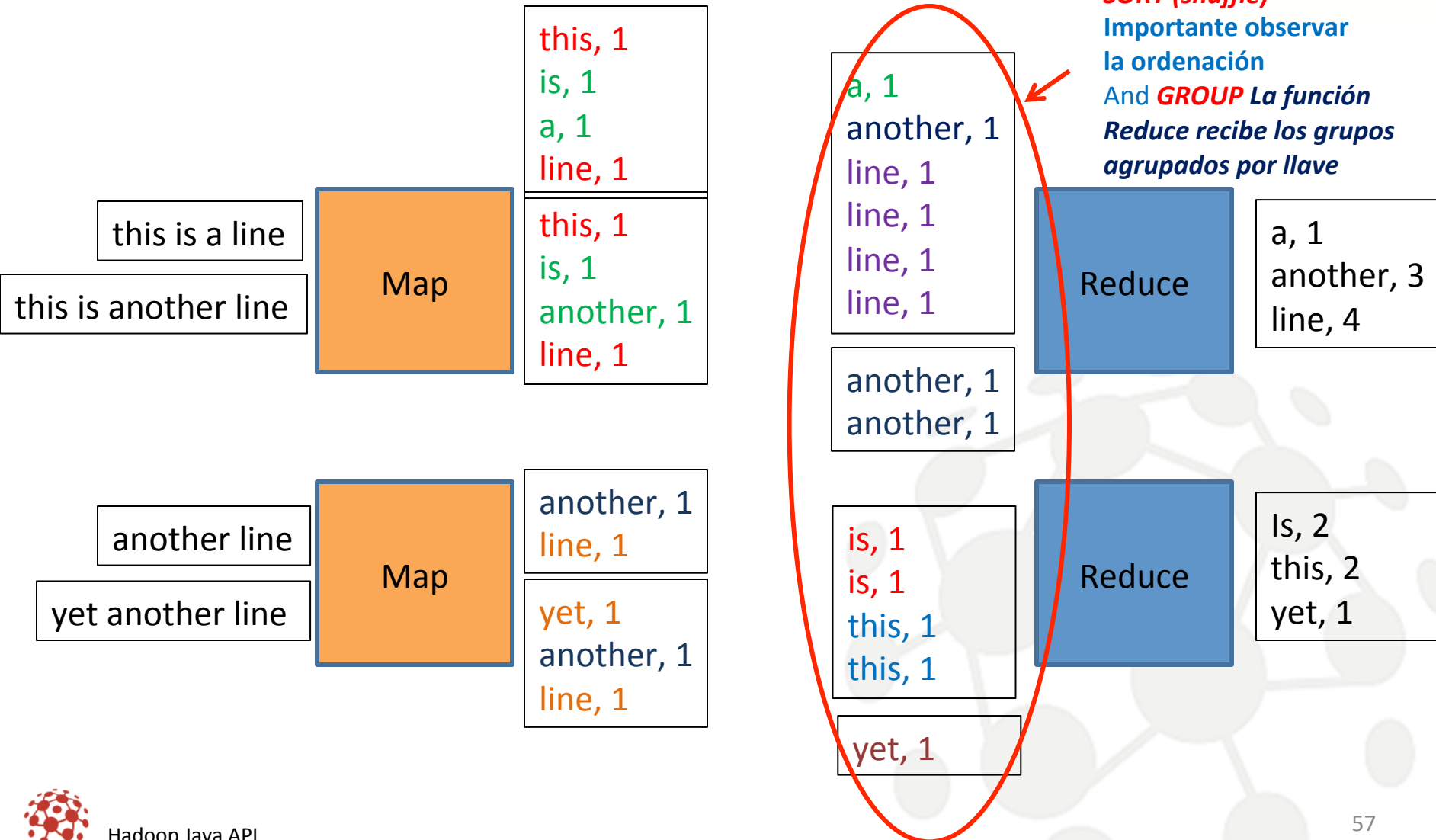
Ejemplo típico: WordCount

- Input: un fichero de texto
- Objetivo: contar el número de apariciones para cada palabra en el texto
 - Suposición: no hay palabras entre líneas
 - Procesaremos todo en minúsculas
- Arquitectura de la solución
 - Programa principal (Driver)
 - Mapper
 - Reducer





WordCount (recordatorio)





¿Tiene alguna utilidad este ejemplo?

Reglas de asociación

- Una relación de implicación $X \rightarrow Y$, donde X e Y son conjuntos de elementos
- Ejemplo:
 $\{\text{Milk, Diaper}\} \rightarrow \{\text{Beer}\}$

Ejemplos:

$\{\text{Milk, Diaper}\} \Rightarrow \text{Beer}$

$$s = \frac{\sigma(\text{Milk, Diaper, Beer})}{|T|} = \frac{2}{5} = 0.4$$

$$c = \frac{\sigma(\text{Milk, Diaper, Beer})}{\sigma(\text{Milk, Diaper})} = \frac{2}{3} = 0.67$$

<i>TID</i>	<i>Items</i>
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke





Análisis de comentarios (Twitter)

COUNT	TEXT	CLASS
50	I really like this course and am learning a lot	positive
20	I really hate this course and think it is a waste of time	negative
20	I do not like this course and It's quite bore	negative
90	I'm enjoying myself a lot and learning something too	positive
25	I did not enjoy this course enough	negative

El cálculo de probabilidades está basado en CONTAR

El estudio de las probabilidades condicionadas se puede emplear para el análisis de txto

Total comments=235; Total+=140; Total-=95

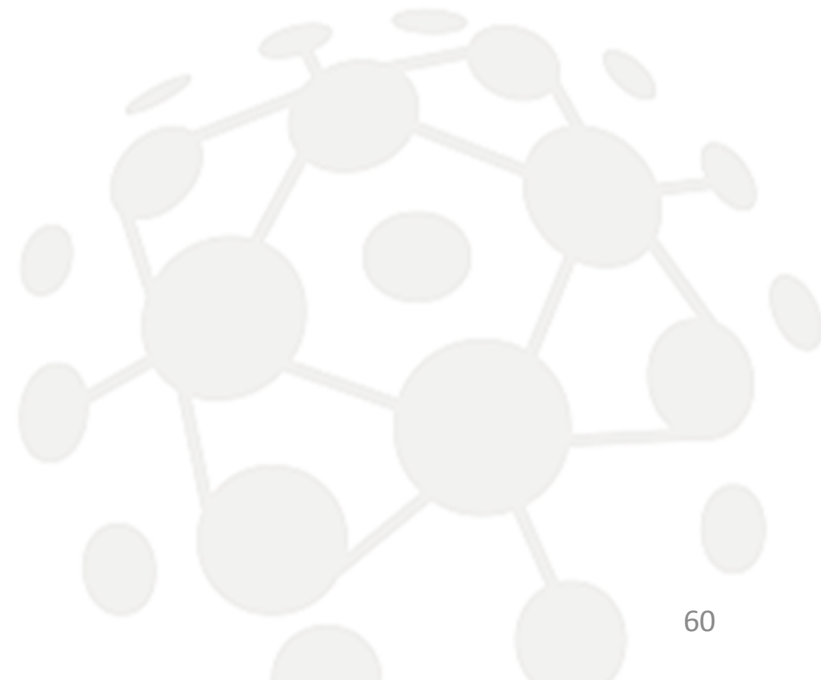
$Prob(+ | "like") = Prob(+ \cap "like") / Prob("like") = 50/70$





Programando nuestro WordCount

- Emplearemos el paquete creado para los ejemplos anteriores como plantilla, lo copiamos y pegamos en nuestro mismo proyecto
 - `com.utad.api.wordcount`
- Renombramos (refactorizar) todas las clases cambiando “Identity” por WordCount





MapperWordCount

• Recibimos

- **(clave)** Número (LongWritable): byte offset de la línea
- **(valor)** Text: una línea de texto

• Emitimos

- **(clave)** Text: una lista de palabras (texto)
- **(valor)** Número (IntWritable): contamos apariciones de palabras (**1** – no agrupa)

• Mapper<LongWritable, Text, Text, IntWritable>





Configurando el Mapper

- Arreglamos librerías
 - ¿Hace falta importar NullWritable?
 - Los parámetros del método map no sufren cambios
- ¿Qué hay que hacer?
 - Para cada palabra de cada línea del texto emitir el par <palabra, 1> (palabra en minúscula)
 - String word: `textLine.toString().split("\\b+")`
 - Empleamos los espacios en blanco como separadores (estamos empleando Expresiones Regulares)

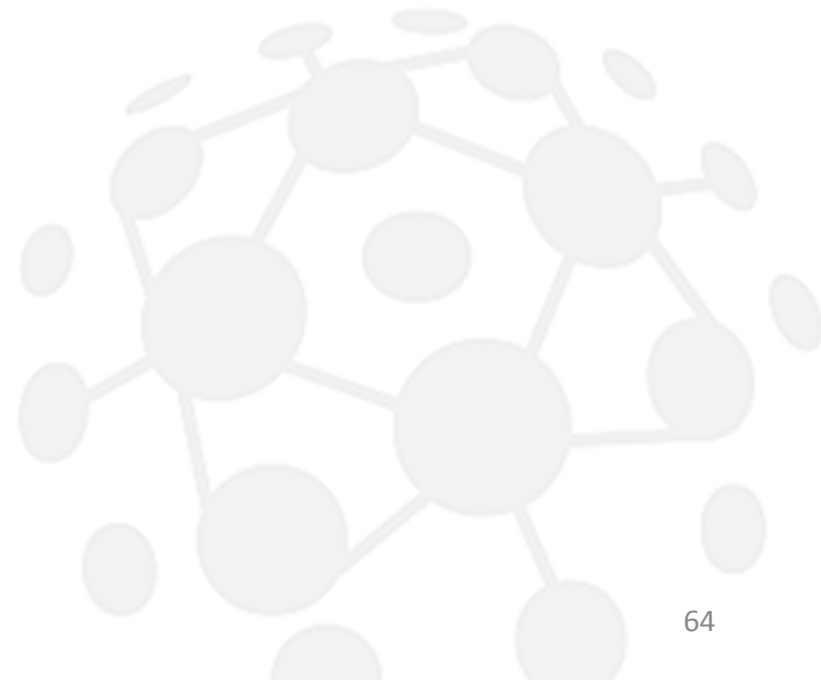


- `b` = word boundary
- Matches at the position between a word character (anything matched by `\w`) and a non-word character (anything matched by `[^\w]` or `\W`) as well as at the start and/or end of the string if the first and/or last characters in the string are word characters.



¿Qué emite el Mapper?

- Para cada palabra, se emite la palabra en minúscula y un 1
 - `context.write (word.toLowerCase(), 1)`
 - No obstante, esto no funciona. MapReduce espera una variable `Text` y una variable `IntWritable`
 - Por tanto debemos realizar las conversiones adecuadas





Creando variables MapReduce

- Como siempre emitimos 1, nos creamos una constante LongWritable asociada a este valor
 - `private static IntWritable_1 = new IntWritable(1);`
- Para el texto, nos crearemos un atributo de tipo Text Writable
 - `private Text WordAsText = new Text();`
- Empleo el método “set” para fijar el valor de WordAsText
 1. `WordAsText.set(word.toLowerCase());`
 2. `context.write(WordAsText, IntWritable_1);`
 - *Observar que si la línea 1 se introduce directamente en el primer parámetro de context.write, el compilador avisa de un error porque interpreta que recibe un “void”, no un Text Writable*





Filtrando palabras

- El procedimiento anterior, puede incluir palabras como “Var_0”, “.”, “;”, “i”, “?”...
- Para eliminar del proceso estas palabras, podemos emplear un filtro basado en la utilización de ERs, indicando los caracteres que pueden tener las palabras que nos interesan. Por ejemplo (recordemos que la palabra está en minúscula):
 - “[a-zAÉÍÓÚÑÇ]”
 - El método “matches()” de un Text Writable comprueba si el texto se ajusta a la ER que recibe de parámetro
 - Nosotros sólo pasaremos al contexto las palabras que Sí cumplan el filtro





MapperWordCount

```
for (String word : textLine.toString().split("\\b+")) {  
    word = word.toLowerCase();  
    if (word.matches("[a-záéíóúñç]+")) {  
        WordAsText.set(word);  
        context.write(WordAsText, IntWritable_1);  
    }  
}
```





ReducerWordCount

• Recibimos

- **(clave)** Text: una lista de palabras (texto)
- **(valor)** Número (IntWritable): contamos apariciones de palabras (**1** – no agrupa)

• Emitimos

- **(clave)** Text: una lista de palabras (texto)
- **(valor)** Número(LongWritable): contamos apariciones totales de palabras

• Reducer <Text, IntWritable, Text, LongWritable>





Configurando el Reducer

- Arreglamos librerías
- El método reduce quedaría:
 - El mapper emite <word, 1>
 - Antes de llegar al reducer, la información emitida por el mapper pasa por un proceso sort / shuffle
 - Por tanto, al reducer le llega para cada clave (palabra), una lista de 1's, que hay que procesar
 - `protected void reduce(Text WordKey, Iterable<IntWritable> ListOfOnes, Context context)...`





Tarea del Reducer

- El reducer lo único que tiene que hacer es recorrerse la lista de unos contando cuantos 1's hay
- Emite como clave la palabra
- Emite como valor el número de unos (que puede ser un valor long puesto que puede haber muchos)
- Utilizamos una variable Long de Java para contar y antes de emitir la salida creamos una variable LongWritable recogiendo ese valor





ReducerWordCount

```
public class ReducerWordCount extends  
    Reducer<Text, IntWritable, Text, LongWritable> {  
    private LongWritable countAllWords = new LongWritable();  
    @Override  
    protected void reduce(Text WordKey, Iterable<IntWritable> ListOfOnes,  
        Context context)  
        throws IOException, InterruptedException {  
        long count1s = 0;  
        for (@SuppressWarnings("unused")  
        IntWritable one : ListOfOnes) {  
            count1s++;  
        }  
        countAllWords.set(count1s);  
        context.write(WordKey, countAllWords);  
    }  
}
```





DriverWordCount y ejecución

La refactorización debe haber funcionado bien, sólo es necesario actualizar el tipo de datos **emitido** por el mapper y por el reducer

- `job.setMapOutputKeyClass(Text.class);`
- `job.setMapOutputValueClass(IntWritable.class);`
-
- `job.setOutputKeyClass(Text.class);`
- `job.setOutputValueClass(LongWritable.class);`





Ejecutando el ejemplo WordCount

- Debemos crear una nueva configuración para ejecutar correctamente este ejemplo y que detecte el método “main” que queremos emplear en esta ocasión
- Duplicamos la configuración (Java Application) del ejemplo anterior
 - Renombramos la configuración a DriverWordCount
 - `com.utad.api.wordcount.DriverWordCount`
- Antes de ejecutar, comprobemos que no hay archivo de salida puesto que estamos empleando los mismos nombres como ficheros de entrada y salida





Los datos

- Refrescamos el proyecto y aparecerá el archivo de salida con las palabras ordenadas alfabéticamente

a7

acabar1

acordarme1

adarga1

administración1

afición1





Configurando el número de maptask y reducetask **MBIT School** Madrid Business Intelligence Technology

- Lógicamente, en un entorno en producción, queremos trabajar con varias tareas map y varias tareas reducer
- Para especificar que queremos dos tareas reducer
 - `job.setNumReduceTasks(2);`
 - Sin efecto esta instrucción con un hilo local de ejecución
- En la nueva API no hay un método similar para las tareas de mapeo
- Hadoop genera una tarea de mapeo para cada partición del fichero de entrada (InputSplit) que ha generado el correspondiente "InputFormat" que hayamos configurado en el Job





Ejecución especulativa

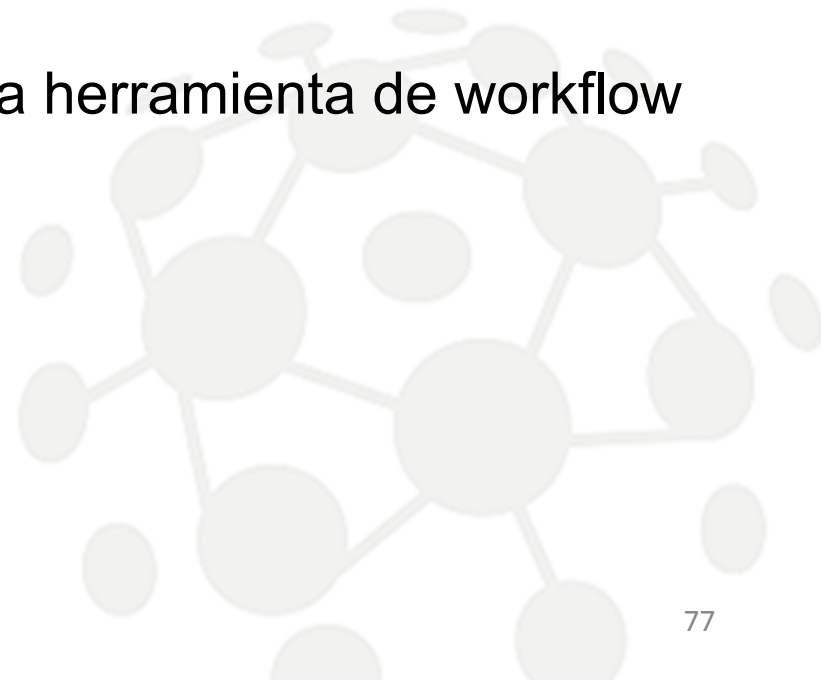
- Recordemos que la ejecución especulativa ejecuta varias veces el mismo trabajo pero en nodos distintos (de forma redundante)
- El objetivo es evitar “cuellos de botella” por la lentitud de algún nodo frente al resto
- “Gana” el primer proceso que acaba
- Por defecto está habilitado
 - `job.setMapSpeculativeExecution(false);`*
 - También hay métodos equivalentes tanto para el reducer como para el propio Job
- Se lanza siempre despues de que se hayan ordenado todas las tareas y existan algunas que ya lleven tiempo ejecutándose (al menos un minuto) y, en ese periodo, no han avanzado mucho en comparación con el resto de tareas asociadas al Job





Procesos enlazados

- Varias opciones, la más simple consiste en encadenar procesos manualmente, de forma que la salida del primero coincida con la entrada del segundo y así sucesivamente
- La segunda es emplear una instancia de la clase ***JobControl***
 - Podemos encadenar varios procesos gracias al método *addjob()*
- El tercer método es emplear alguna herramienta de workflow





Resumen

- Hemos visto que Eclipse + Maven es uno de los entornos más recomendados para trabajar con MapReduce
- Para configurar un proyecto MapReduce nos basamos en el archivo de configuración pom.xml
- Una solución básica MapReduce se compone de:
 - Driver
 - Mapper
 - Reducer
- Hemos visto dos ejemplos típicos en el aprendizaje de MapReduce
 - La identidad
 - El contador de palabras

