

# Implementación Café

Integración de la Información y Aplicaciones



Juan Hernández Robles  
Adrián Moreno Monterde  
Juan Diego Borrero Domínguez

## Índice

<b>1.- Presentación del problema.....</b>	<b>2 -</b>
Estructura de una comanda .....	3 -
Ejemplo de comanda.....	3 -
<b>2.- Diagrama DSL.....</b>	<b>4 -</b>
<b>3.- Diagrama de clases .....</b>	<b>5 -</b>
<b>4.- Realización del proyecto.....</b>	<b>6 -</b>
<b>5.- Script de creación de la base de datos.....</b>	<b>7 -</b>
<b>6.- Funcionamiento del programa .....</b>	<b>8 -</b>
<b>7.- Estimación del tiempo .....</b>	<b>11 -</b>

## Índice de ilustraciones

Ilustración 1.- Flujo de comandas y bebidas de una cafetería .....	2 -
Ilustración 2.- Estructura de una comanda .....	3 -
Ilustración 3.- Ejemplo de comanda (comanda.xml) .....	3 -
Ilustración 4.- Diagrama DSL de la implementación de Café .....	4 -
Ilustración 5.- Diagrama de clases de nuestra aplicación .....	5 -
Ilustración 6.- Script MySQL de la base de datos .....	7 -
Ilustración 7.- Fichero de entrada del programa (comanda.xml) .....	8 -
Ilustración 8.- Fases de la ejecución del programa .....	9 -
Ilustración 9.- Fichero de salida del programa (finalcomanda.xml) .....	10 -

## 1.- Presentación del problema

El proyecto consiste en la realización de una integración de varias aplicaciones en una sola para procesar las comandas y bebidas en una cafetería.

Un cliente realiza un pedido, entonces se registra la comanda en el sistema, se parsea a tipo *Document* y se añade al *Slot* del puerto de entrada. Estos mensajes se dividen con un *Splitter* formando una lista de elementos.

Las comandas incluyen pedidos de bebidas frías y/o calientes, las cuales son preparadas por diferentes bármanes, comprobando su stock en una base de datos. Las bebidas se distribuyen según el tipo con *Distributor*.

Tanto con las bebidas frías como con las bebidas calientes, replicamos el mensaje en dos para por un lado consultar el stock en la base de datos y luego unir el mensaje con su stock con un *Correlator*. Enriquecemos el mensaje con *Context Enricher* añadiendo al cuerpo del mensaje información de su stock.

Los mensajes de las bebidas frías y bebidas calientes se enrutan con un *Merger*. Finalmente se reconstruye el mensaje dividido previamente por la tarea *Splitter* con un *Aggregator*. Esta comanda final estará lista para ser entregada.

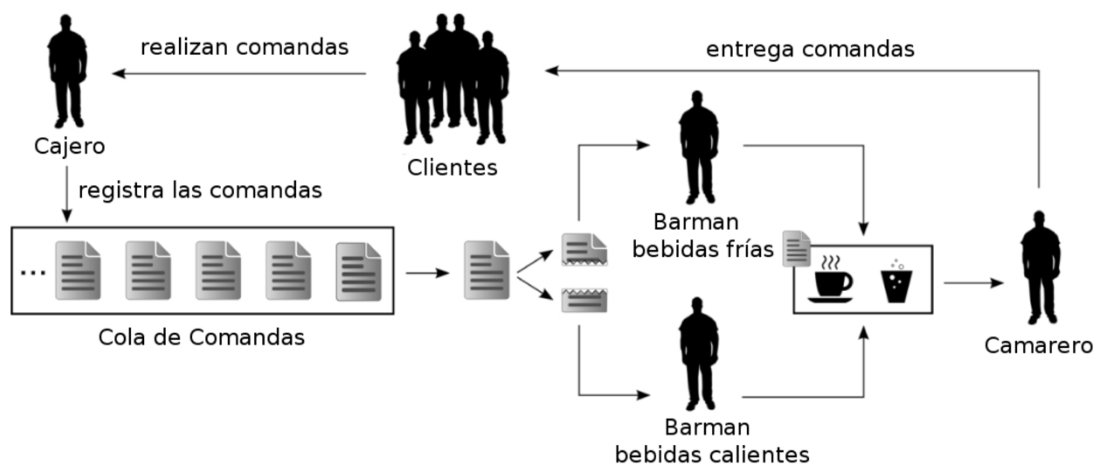


Ilustración 1.- Flujo de comandas y bebidas de una cafetería

## Estructura de una comanda

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <cafe_orden>
3      <orden_id>1</orden_id>
4      <bebidas>
5          <bebida>
6              <nombre>...</nombre>
7              <tipo>...</tipo>
8          </bebida>
9          ...
10     </bebidas>
11 </cafe_orden>
```

Ilustración 2.- Estructura de una comanda

## Ejemplo de comanda

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <cafe_orden>
3      <orden_id>1</orden_id>
4      <bebidas>
5          <bebida>
6              <nombre>cola</nombre>
7              <tipo>fria</tipo>
8          </bebida>
9          <bebida>
10             <nombre>cerveza</nombre>
11             <tipo>fria</tipo>
12         </bebida>
13         <bebida>
14             <nombre>cafe</nombre>
15             <tipo>caliente</tipo>
16         </bebida>
17         <bebida>
18             <nombre>té</nombre>
19             <tipo>caliente</tipo>
20         </bebida>
21     </bebidas>
22 </cafe_orden>
```

Ilustración 3.- Ejemplo de comanda (comanda.xml)

## 2.- Diagrama DSL

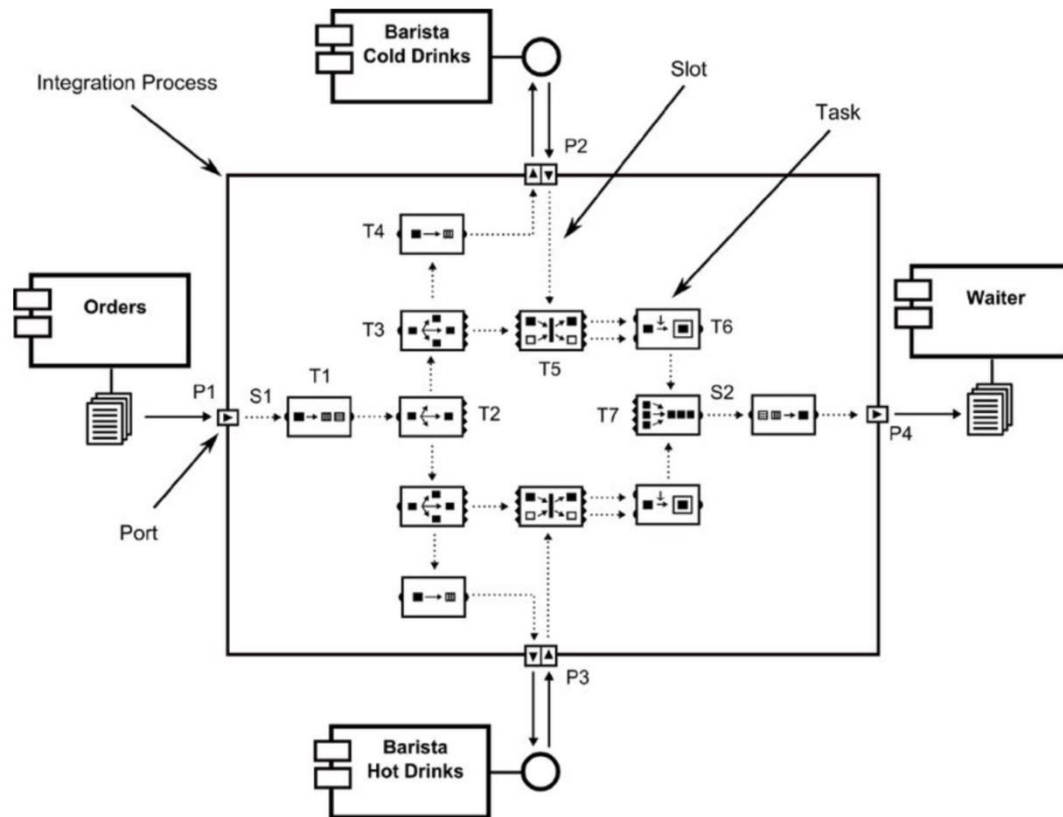


Ilustración 4.- Diagrama DSL de la implementación de Café

### 3.- Diagrama de clases

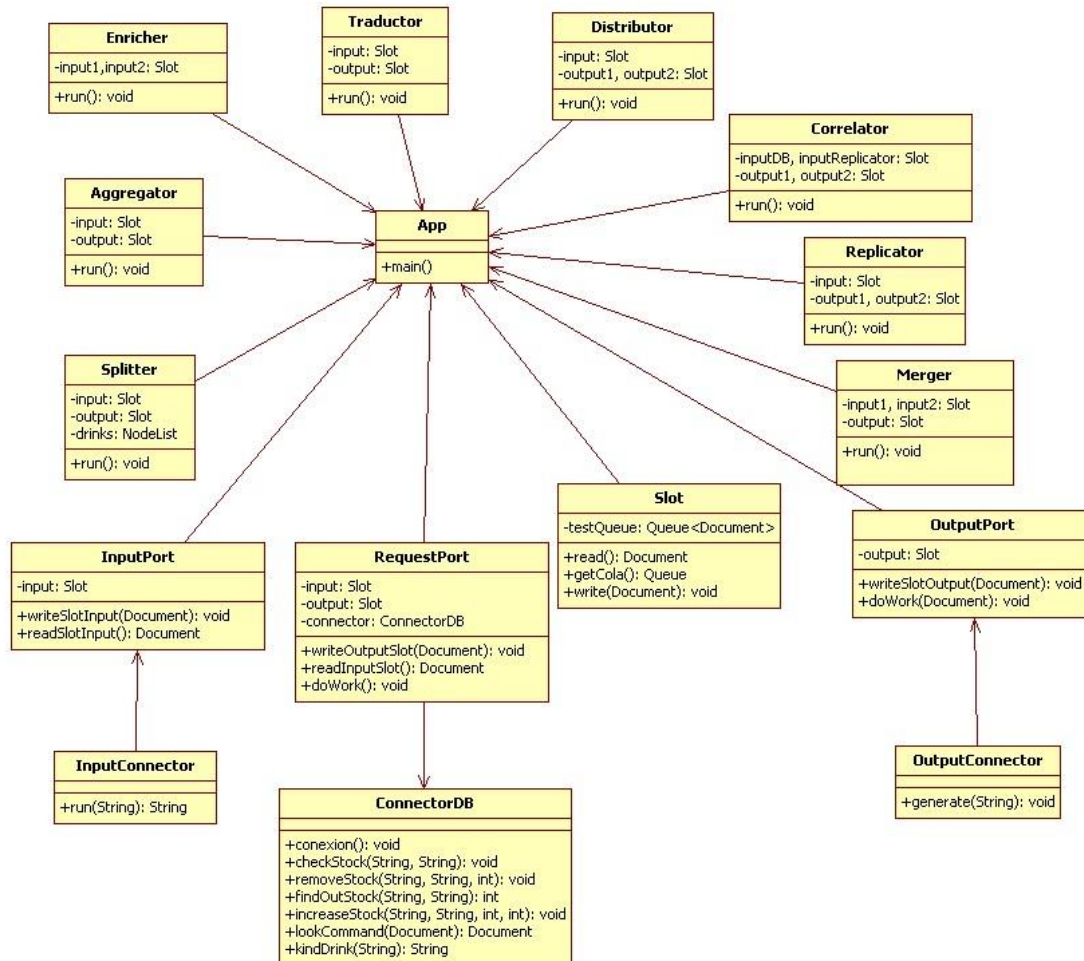


Ilustración 5.- Diagrama de clases de nuestra aplicación

## 4.- Realización del proyecto

El proyecto ha sido realizado en el lenguaje Java en el IDE de NetBeans. Para la sincronización a la hora de trabajar y la administración del proyecto hemos estado trabajando con [Github](#). Al mismo tiempo, la organización de las tareas y su desarrollo lo hemos plasmado en [Trello](#), ya que usa la metodología ágil Kanban y nos permite gestionar el proyecto de forma sencilla y muy visual.

En cuanto a la base de datos hemos usado [phpMyAdmin](#) que nos ofrece [db4free.net](#), ya que nos ofrece un servicio de prueba (servidor MySQL) donde gratuitamente hemos podido crear una cuenta y probar la aplicación.

Hemos dividido el proyecto en varios paquetes (packages) que detallaremos a continuación:

### Connectors:

En este paquete tenemos tres clases, las cuales nos dan acceso a los distintos conectores que tenemos en nuestra práctica. El conector de entrada (*InputConnector*) transforma la información de una comanda en XML a tipo *Document*. El conector de salida (*OutputConnector*) transforma la información de una comanda de tipo *Document* en un archivo XML. En el conector con la base de datos (*ConnectorDB*) tenemos métodos para comprobar la conexión con la base de datos además de otros como: *checkStock*, *removeStock*, *findOutStock*, *increaseStock*, *lookCommand* and *kindDrink*.

### Tasks:

En este paquete se encuentran las clases para ejecutar cada tarea a través del método *run()*. Hemos implementado las siguientes tareas: *Aggregator*, *ContextEnricher*, *Correlator*, *Distributor*, *Merger*, *Replicator*, *Splitter* y un *Translator*. Ésta última ha sido implementada con XSLT, pero el resto han sido implementadas en XPATH.

### Ports:

En este paquete hemos definido cuatro clases:

- Slot: implementado como una cola en la que se lee y escriben *Documents*.
- InputPort: la cual obtiene, escribe, lee e introduce un slot de entrada.
- OutputPort: escribe y hace el trabajo de un slot de salida.
- RequestPort: lee, escribe y hace el trabajo de un slot de entrada hacia un slot de salida.

### Application:

En este paquete se encuentra el main de ejecución del programa en el que se llaman a las distintas tareas, de forma secuencial, siguiendo el diagrama DSL definido anteriormente.

## 5.- Script de creación de la base de datos

```
1  SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
2  SET AUTOCOMMIT = 0;
3  START TRANSACTION;
4  SET time_zone = "+00:00";
5
6  CREATE DATABASE IF NOT EXISTS `iia2021` DEFAULT CHARACTER SET utf8mb4 COLLATE
7  utf8mb4_0900_ai_ci;
8  USE `iia2021`;
9
10 DROP TABLE IF EXISTS `BebidasCalientes`;
11 CREATE TABLE IF NOT EXISTS `BebidasCalientes` (
12   `nombre` varchar(30) NOT NULL,
13   `stock` int(11) DEFAULT NULL,
14   PRIMARY KEY (`nombre`)
15 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
16
17 INSERT INTO `BebidasCalientes` (`nombre`, `stock`) VALUES
18 ('cafe', 6),
19 ('chocolate', 6),
20 ('te', 4);
21
22 DROP TABLE IF EXISTS `BebidasFrias`;
23 CREATE TABLE IF NOT EXISTS `BebidasFrias` (
24   `nombre` varchar(30) NOT NULL,
25   `stock` int(11) DEFAULT NULL
26 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
27
28 INSERT INTO `bebidas_frias` (`nombre`, `stock`) VALUES
29 ('cocacola', 6),
30 ('pepsi', 6),
31 ('nestea', 4);
32
33 COMMIT;
```

Ilustración 6.- Script MySQL de la base de datos



## 6.- Funcionamiento del programa

Para probar el funcionamiento del programa basta con ejecutarla, nos pedirá el nombre del fichero XML donde se encuentra la comanda, hemos incluido dos ficheros para probar llamados comanda y comanda2. Se mostrará el funcionamiento de la aplicación pasando por las distintas tareas. Finalmente, pedirá un nombre para la creación del fichero de salida de la comanda final.

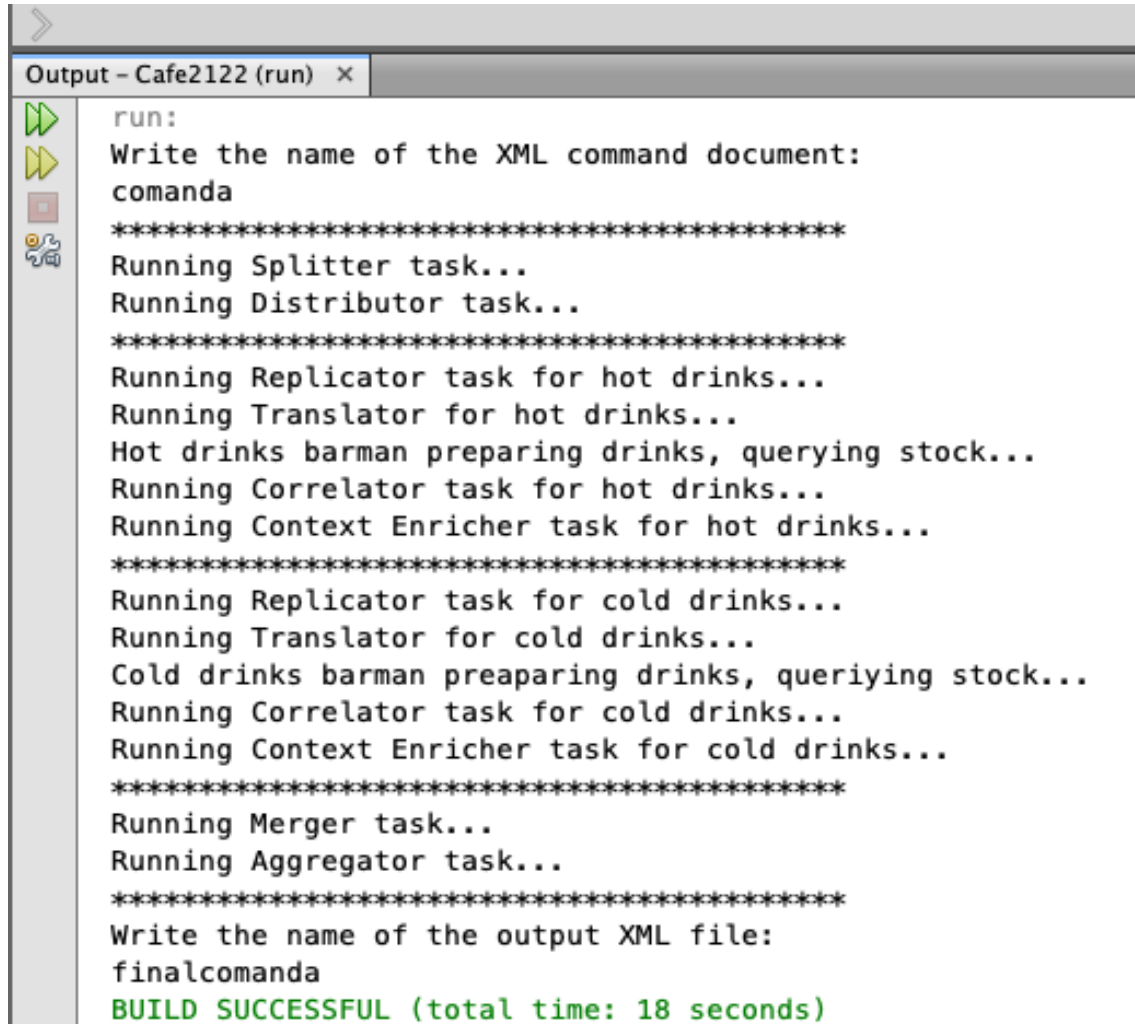
Los ficheros de prueba se encuentran dentro de la carpeta del proyecto.

Fichero de entrada del programa

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <cafe_orden>
3      <orden_id>1</orden_id>
4      <bebidas>
5          <bebida>
6              <nombre>cola</nombre>
7              <tipo>fria</tipo>
8          </bebida>
9          <bebida>
10             <nombre>cerveza</nombre>
11             <tipo>fria</tipo>
12          </bebida>
13          <bebida>
14             <nombre>cafe</nombre>
15             <tipo>caliente</tipo>
16          </bebida>
17          <bebida>
18             <nombre>té</nombre>
19             <tipo>caliente</tipo>
20          </bebida>
21      </bebidas>
22  </cafe_orden>
```

Ilustración 7.- Fichero de entrada del programa (comanda.xml)

Proceso de ejecución:



```
run:
Write the name of the XML command document:
comanda
*****
Running Splitter task...
Running Distributor task...
*****
Running Replicator task for hot drinks...
Running Translator for hot drinks...
Hot drinks barman preparing drinks, querying stock...
Running Correlator task for hot drinks...
Running Context Enricher task for hot drinks...
*****
Running Replicator task for cold drinks...
Running Translator for cold drinks...
Cold drinks barman preaparing drinks, queriying stock...
Running Correlator task for cold drinks...
Running Context Enricher task for cold drinks...
*****
Running Merger task...
Running Aggregator task...
*****
Write the name of the output XML file:
finalcomanda
BUILD SUCCESSFUL (total time: 18 seconds)
```

*Ilustración 8.- Fases de la ejecución del programa*

Fichero de salida del programa:

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <cafe_orden>
3      <orden_id>1</orden_id>
4      <bebidas>
5          <bebida>
6              <nombre>cafe</nombre>
7              <stock>true</stock>
8          </bebida>
9          <bebida>
10             <nombre>té</nombre>
11             <stock>>false</stock>
12         </bebida>
13         <bebida>
14             <nombre>cola</nombre>
15             <stock>>false</stock>
16         </bebida>
17         <bebida>
18             <nombre>cerveza</nombre>
19             <stock>>false</stock>
20         </bebida>
21     </bebidas>
22 </cafe_orden>
```

Ilustración 9.- Fichero de salida del programa (finalcomanda.xml)

## 7.- Estimación del tiempo

El tiempo de desarrollo de cada método los tenemos registrado en cada carta del board de proyecto en [Trello](#), no obstante, ese tiempo solamente es programando, por lo que habría que sumarle el tiempo de aprendizaje de los conceptos necesarios para la implementación de la aplicación e investigación.

En total hemos calculado aproximadamente 25-30 horas de trabajo por cada miembro del grupo.