
MEMORIA

Autores: Francisco de Borja García Lamas

Andrea María García Rodríguez

Ignacio Moll Amorós

Carlos Tendero Martínez-Algora

índice

Contenido

Introducción.....	2
Metodología aplicada	4
Validación del DTD	5
Gestión de conflictos	6
Ramas utilizadas.....	6
Visualización de la estructura empleada.	7
Estructuración de Tareas	8
Estructura del código implementado:	9
Conclusión.....	12
Bibliografía	13

Introducción

Esta primera entrega de la asignatura de Desarrollo e Integración del Software consiste en realizar un programa que nos permita hacer un seguimiento y una correcta gestión de pedidos que se puedan generar. Aparte, el otro objetivo de la práctica es mostrar el uso adecuado de la herramienta de gestión de proyectos GIT y la resolución de posibles conflictos.

El sistema a crear tiene que poder generar un documento con formato XML y su correspondiente DTD que muestre los datos de la gestión del almacén, almacenando los siguientes datos:

Producto

Código

Nombre

Descripción

Stock

Localización

o Pasillo o

Estantería o

Estante

Pendientes (de entrada en almacén)

Clientes

Nombre

Apellidos

Email

Teléfono de contacto

Dirección

o Calle

o Número o

Código postal



o Población

o País

Pedidos

Productos

Cantidad

Dirección de entrega

o Calle

o Número o

Código postal

o Población

o País

Destinatario

Fecha de entrega estimada

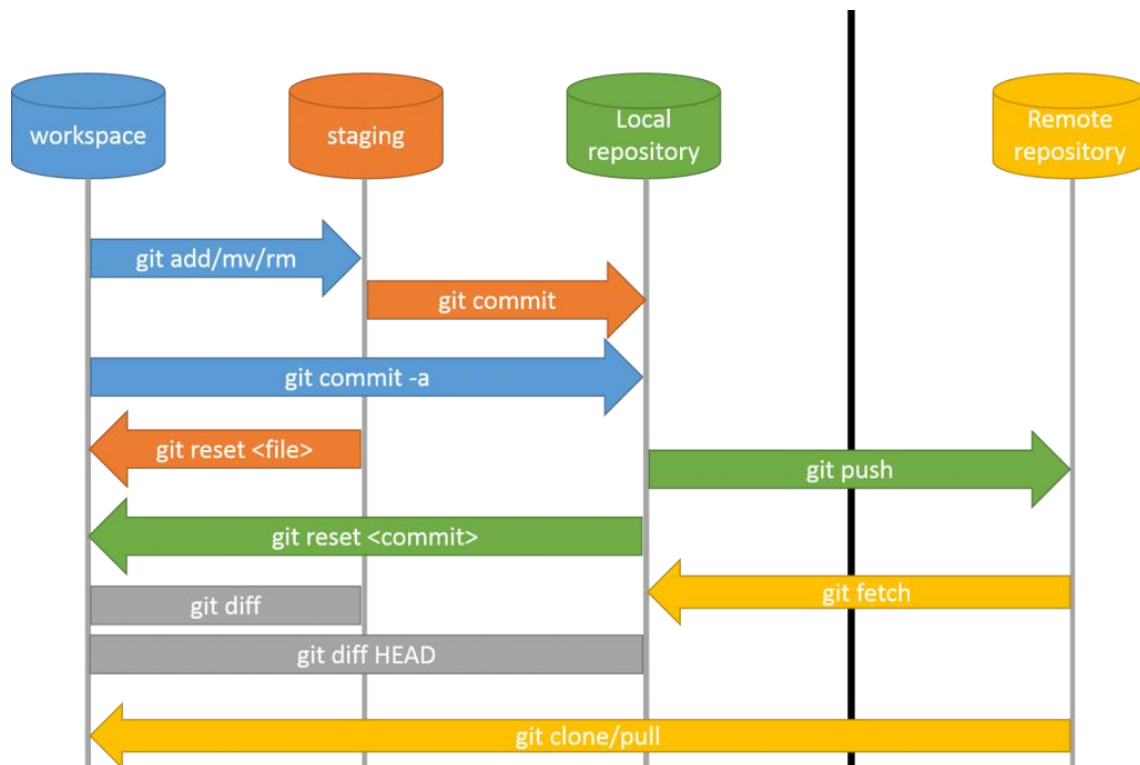
Desarrollo

En esta sección se explica la forma de trabajo empleada para realzar la práctica en equipo.

Metodología aplicada

Para trabajar usamos la metodología de trabajo de división de tareas entre los miembros y subiendo nuestras respectivas versiones del trabajo progresivamente en github. La distribución de las tareas se especifica más adelante. Nos basamos en el sistema git de repositorio local y online, teniendo cada miembro del equipo su propia rama que cuelga de la rama development. Para la comunicación usamos el entorno discord, de manera que podemos hablar entre nosotros cuando uno ha subido una nueva versión de su código evitando subidas simultáneas o el uso de versiones antiguas. Para la programación del código y del XML hemos optado por el entorno eclipse, y como lenguaje de programación usamos java.

Esta es una imagen que muestra el esquema de almacenamiento de git, el cual usamos:



Cada miembro va subiendo su trabajo propio en el staging, y de ahí al local repository cuando se ha cumplido el objetivo de la tarea. Cuando está acabada, lo comunica al resto de miembros (generalmente vía discord) y lo sube al repositorio remoto con git push.

Validación del DTD

Para realizar el DTD tenemos que ver todos los elementos que se pueden insertar en nuestro XML, y a partir de ahí controlar las restricciones que queramos, en nuestro caso que se puedan tener varios clientes pedidos y productos, con lo cual se nos queda un DTD bastante largo pero simple, ya que solo hay que seguir los elementos del xml e ir poniendo asteriscos en cada elemento que pueda tener varios elementos, el resultado nos lo ha dado como un xml bien formado y con el DTD validado. Hemos utilizado un validador online.

XML

```
<localizacion>
  <hall>0</hall>
  <shelf>0</shelf>
  <shelving>0</shelving>
</localizacion>
<name>Cafe</name>
<pending>0</pending>
<stock>0</stock>
</products>
</almacen>
```

Validate

Validation Result

Valid

Gestión de conflictos

La gestión de conflictos es algo común y muy a tener en cuenta cuando se trabaja con herramientas como git. Pueden darse situaciones en las que los códigos de distintos miembros del equipo se pisen, estén desactualizados o sean no válidos. Para resolver esto, hemos optado por hacer una división de tareas que sean en la medida de lo posible independientes para luego poder compilarlas todas juntas. Una vez realizadas dichas tareas, nos reunimos vía discord para crear el código que une a todas las clases. Al estar reunidos, no hay posibilidad de código desfasado. Las tareas que se son independientes entre ellas son, lógicamente, la implementación de las distintas clases como pedido, producto, localización, que como mucho dependen unas de otras en sólo una línea de código que se puede escribir en la fase de unión del mismo.

Ramas utilizadas

Para esta práctica, hemos dividido el entorno de trabajo de git en ramas personalizadas para un mejor funcionamiento de este. Las ramas que hemos creado y su funcionamiento se explican a continuación:

- Master

Rama de entrega. Se importa el código a esta rama una vez sea funcional. Aplicando distintas versiones posteriores si procede.

- Development

Rama principal. En esta rama se junta todo el código funcional y completado. Se adjuntan las nuevas funcionalidades una vez completada la tarea asignada para cada miembro.

- IgnacioMoll / AndreaMGR / Borja / CarlosTendero

Ramas personales. Estas ramas tienen la función de gestionar el trabajo de cada miembro individualmente, no tiene ningún requisito de formato. Pretende guardar los cambios de trabajo de cada miembro independientemente en el desarrollo de sus tareas.

Visualización de la estructura empleada.

Una de las múltiples formas para ver el registro de trabajo es con el comando log, existen múltiples aplicaciones para visualizar el registro de trabajo, pero para nuestro uso, nos hemos manejado con el que trae git. Podemos visualizar las ramas que hemos creado y los commits realizados por cada uno con la siguiente instrucción:

Pudiendo ver todo el despliegue (historial) de versiones y trabajo del producto.


```
$ git log --all --graph --decorate --oneline
* bb449bd (HEAD -> development, origin/development, origin/HEAD, origin/CarlosTendero, IgnacioMoll) Solucionados op 4 y op 5
* 26048ee Mainop1 y op3 hace lo que tiene que hacer
* 20781fb actualizandomain, ahora hace lo que tiene que hacer
* 102aa01 Dejo el coche y sigo
* 7e75007 (origin/ignaciomoll, origin/Borja, Borja) Función de crear producto hecha.
* b336fe7 Crearcliente y control de variables completado.
* 0c00c4a Añadircliente
* 8b4e06c (origin/AndreaMGR) actualizar mi rama Merge branch 'development' into AndreaMGR
* 089e37b resolviendo conflictos
* b96ba1d División del código en funciones (Legibilidad)
* 1c4359d fufusp xp
* d8e2cc4 El DTO que valida el XML generado por el programa
* a9b62b0 Actualización del main
* ec2c5d6 Actualización de XML en la clase Producto
* f7b9b77 Actualización de XML en la clase Pedidos
* ec38b27 Actualización de XML en la clase Localización
* 8322bf7 Actualización de XML en la clase Dirección
* 799f207 Actualización de XML en la clase Clientes
* fde2e59 Actualización de XML en la clase Almacen
* b595d40 Actualizando otra vez
* 57a0deb MainModificado,solo opcion1
* 014c069 MainHechoopcion1
* ad2d14e Arreglando los merge
* d41efbe clase almacen que sera el elemento raiz del XML, no necesita getters y setters, solo constructores
* 89eda71 ya esyan todas las clases con getters y setters constructores y demas, falta fijarse en las librerias para el XML, ahora a centrarse en el main
* d91a93b Clases Producto y localizacion, ahora pedidos no peta, faltan getters y setters
* dc5458e revisar funcionamiento del pedido con clase cliente Merge branch 'development' into AndreaMGR
* 73dd4a5 Merge branch 'development' of https://github.com/IMoll/Practicas_DIS into development
* 72b8c23 clientes Direccion Setters y Getters
* 4965d0a Direccion clase vacia
* 5795bb9 clase vacia cliente
* 49aadf2 Abrir el proyecto por favor
* 710aa63 No servian
* 2fe8527 Abrir por fin el proyecto intento n10
* e4341fb Clases_vacias
* 1d014d1 Toma
* 0735971 getters y setters
* caa5d1f clase con atributos y constructores
* 5e6856d clase vacia pedidos
* 65e4dad Merge branch 'development' into AndreaMGR
* a741921 Merge branch 'development' into AndreaMGR
* 38aca65 borrar todo para despues pullear el proyecto bueno
* 35224a3 He creado las clases vacias Direccion y Localización
* 9b5dcf1 creación de producto vacío definitivo.
* 57f3601 resolviendo conflicto Merge branch 'AndreaMGR' into development
* 502ebf6 proyecto vacío
* ec279d9 Update README.md
* 523555a Merge branch 'development' of https://github.com/IMoll/Practicas_DIS into development Yo solo quiero hacer un push
* dd49676 Update README.md
* 14010be Merge branch 'ignaciomoll' into development
* 87fd98a He añadido la memoria básica
* fda4d87 Update README.md
* abb2a3c (origin/master) update README.md
```

El uso que le estamos dando a git es grande y la adaptación que estamos teniendo es progresiva y muy positiva.

Estructuración de Tareas

Hemos dividido el trabajo a realizar en tareas para un mejor funcionamiento del equipo. Dichas tareas se han gestionado en el propio github, para ello hemos creado un nuevo proyecto con tasks asociadas, donde cada uno ha ido realizando sus tareas y modificándolas.

Las tareas establecidas se muestran a continuación:

Practical1  Updated 22 minutes ago

To do	In progress	Done
<input type="text" value="Enter a note"/> <input type="button" value="Add"/> <input type="button" value="Cancel"/>	<ul style="list-style-type: none"> Responsible de comunicación. (Andrea) <input type="checkbox"/> Added by IMoll Responsible de funcionamiento de Git. (Nacho Moll) <input type="checkbox"/> Added by IMoll Responsible de realización de la memoria. (Borja) <input type="checkbox"/> Added by IMoll Responsible de código. (Carlos Tendero) <input type="checkbox"/> Added by IMoll 	<ul style="list-style-type: none"> Crear clase Clientes y Dirección. (Carlos Tendero) <input type="checkbox"/> Added by IMoll Crear clase Almacen, Localización, Producto, Pedidos. (Andrea) <input type="checkbox"/> Added by IMoll Crear Main, inicializar variables y menú. (Carlos Tendero) <input type="checkbox"/> Added by IMoll Adaptar todas las clases para ser escritas como XML. (Nacho Moll) <input type="checkbox"/> Added by IMoll Crear función de leer pedidos en el main. (Carlos Tendero) <input type="checkbox"/> Added by IMoll Crear DTD y comprobar su funcionamiento. (Andrea) <input type="checkbox"/> Added by IMoll Dividir en Funciones el código. Tarea de Limpieza. (Nacho Moll) <input type="checkbox"/> Added by IMoll
<ul style="list-style-type: none"> Testeo de funcionalidades. (Borja) <input type="checkbox"/> Added by IMoll Crear función de cargar datos del XML. (Carlos) <input type="checkbox"/> Added by IMoll Crear función de guardar en XML en el main. (Andrea) <input type="checkbox"/> Added by IMoll 		

Para ver el progreso real de las tareas se recomienda visualizarlo en github

Estructura del código implementado:

A continuación se muestra la estructura del proyecto .java creado, con todas las clases y métodos especificados.

Hemos creado 6 clases de objetos y una clase main. Dichas clases objeto se llaman:

1. Almacén
2. Clientes
3. Dirección
4. Localización
5. Pedidos
6. Producto

A continuación se explican más detalladamente cada clase con sus atributos correspondientes:

- **Clase producto:** la clase producto es un objeto que representa a todos los productos que puede haber en el almacén. Los datos que se guardan sobre estos son:
 1. code (tipo int)
 2. name (tipo string)
 3. description (tipo string)
 4. stock (tipo int)
 5. localizacion (tipo localizacion)
 6. pending (tipo int)

- **Clase clientes:** la clase clientes es un objeto que representa a todos los clientes del almacén. Los datos guardados son:
 1. name (tipo string)
 2. surname (tipo string)
 3. email (tipo string)
 4. phone_number (tipo direccion)

- **Clase direccion:** la clase direccion es un objeto que contiene los datos de las direcciones de los clientes a los que se les entrega el producto en forma de envío. Los datos almacenados son:
 1. street (tipo string)
 2. number (tipo int)
 3. postal_code (tipo int)
 4. population (tipo string)
 5. country (tipo string)

- **Clase localizacion:** la clase localizacion representa algunas especificaciones a la hora de guardar información sobre la situación física del cliente. Los datos almacenados son:
 1. hall (tipo int)
 2. shelving (tipo int)
 3. shelf (tipo int)

- **Clase pedidos :** la clase pedidos contiene información de todos los pedidos que se realizan al almacén. Los datos almacenados son:
 1. products (arraylist tipo productos)
 2. quantity (arraylist tipo integer)
 3. delivery_adress (tipo direccion)
 4. client (tipo clientes)
 5. destinatario (tipo string)
 6. estimated_date (tipo date)

- **Clase almacén:** esta clase representa los almacenes de la compañía existentes. Los datos guardados son:
 1. clientes (arraylist tipo clientes)
 2. products (arraylist tipo producto)
 3. pedidos (arraylist tipo pedidos)

- **Función RealizarNuevoPedido(in, num_productos, fecha_estimada, almacen):**

Esta función sirve para, como su nombre indica, realizar nuevos pedidos por parte de los clientes al almacén.

- **Función RecuperarPedido():**

Esta función sirve para poder recuperar un pedido en caso de que sea eliminado accidentalmente o se quieran reusar los datos del mismo.

- **Función GuardarPedido(almacen):**

Esta función sirve para poder guardar los distintos pedidos en el almacén correspondiente.

- **Función CrearProducto(in, almacen.getProducts()):**

Esta función sirve para poder crear nuevos productos si es necesario y poder añadirlos al catálogo de la empresa.



- **Función CrearCliente(in, almacen.getClientes()):**

Esta función sirve para poder generar nuevos clientes y guardar sus datos en el listado de clientes de la empresa.



Conclusión

Esta práctica nos ha servido para ver de primera mano, entender y profundizar el funcionamiento de git para la gestión de proyectos, así como para practicar con el lenguaje java y la utilidad del XML. Hemos podido trabajar frente a frente con los comandos, repositorios y demás funcionalidades de git y hemos, de la misma manera, podido entender y reflexionar sobre cómo poder resolver los conflictos tanto técnicos como sociales y de equipo que se pueden generar al usar este tipo de herramientas.



Bibliografía

- GitHub (29 de octubre de 2008) Recuperado de https://github.com/IMoll/Practicas_DIS.git
- Pidal, A; Serrano, I, Tema 1 *tema1.version control.git* (Apuntes) Universidad Francisco de Vitoria
- StarckOverflow (15 de septiembre de 2002) Recuperado de <https://es.stackoverflow.com/>
- <https://howtodoinjava.com/jaxb/write-object-to-xml/>
- <https://www.javatpoint.com/jaxb-unmarshalling-example>

