UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

Departamento de Ingeniería Telemática y Electrónica

Programación Avanzada de Aplicaciones

Práctica 1

Arquitecturas en capas: capa de datos

FEBRERO DE 2022

Índice

Introducción 3

Enunciado 3

Desarrollo de la práctica 4

Requisitos previos 4

Maven 4

Estructura de directorios de un proyecto Maven 5

Dependencias de librerías 5

Ciclo de vida de Maven 6

Maven con Eclipse 6

Desarrollo de la capa de datos 6

Interface genérica DAO<T,K> 7

Clase LockerMapDAO 7

Pruebas 7

Pasos a seguir 8

Calendario de entrega 9

Normas de evaluación 9

Introducción

En esta práctica se realizará la codificación de la estructura de una capa de datos que haga uso de los conceptos vistos en teoría sobre la arquitectura en capas, separando la estructura lógica de la capa de datos del resto de la aplicación.

Enunciado

Con vistas a una integración posterior en un software de gestión de armarios de taquillas automáticas para entrega de paquetes se desea disponer de una aplicación informática que permita gestionar las distintas taquillas automáticas disponibles.

La aplicación debe almacenar la siguiente información de cada taquilla:

- Código único que identifica cada armario automático: **code**, de tipo Long
- Nombre: **name**, de tipo String
- Dirección postal donde se encuentra: **address**, de tipo String
- Cada armario tendrá un número de compartimentos para alojar paquetes grandes y para alojar paquetes pequeños: **largeCompartments** y **smallCompartments**, ambas de tipo int.
- Geolocalización del armario: **longitude** y **latitude**, ambas de tipo double.

Para desarrollar la aplicación se ha decidido utilizar una arquitectura estricta en 3 capas. En esta práctica se desarrollará una primera versión de la capa de datos, que utilizará el **patrón DAO** con la interface *DAO* genérica vista en teoría. Esta interfaz será implementada en una clase llamada **LockerMapDAO** que almacenará/recuperará la información de los armarios en una variable en memoria de clase mapa (**java.util.Map**) por simplicidad**.** Evidentemente, en esta implementación al utilizar una variable en memoria se perderá la información al salir de la aplicación, y no tiene realmente persistencia.

Para la transferencia de la información a almacenar/recuperar (DTO) se ha definido ya la clase *Locker* siguiente:

```
public class Locker {
      private Long code;
      private String name;
      private String address;
      private int largeCompartments;
      private int smallCompartments;
      private double longitude;
      private double latitude;
      // Métodos getters/setters a completar
      // Métodos equals/hashCode a completar
}
```

Para probar la capa de datos se realizará una clase de prueba básica utilizando JUnit, no se realizará ningún programa principal.

Desarrollo de la práctica

Requisitos previos

- Repasar el uso de mapas y colecciones en Java y los dos primeros puntos del tema 1.
- Instalar el entorno de desarrollo Eclipse IDE for Enterprise Java Developers.
- Para el control del ciclo de desarrollo de la práctica se utilizará Maven, que viene ya incluido con Eclipse. Puede descargar la plantilla para esta práctica desde Moodle.
- La clase **LockerMapDAO** debe pensarse antes del inicio del laboratorio, a falta de editarla, compilarla y corregir errores.

Maven

La construcción de aplicaciones Java EE es un proceso complejo en el cual deben realizarse tareas de construcción, empaquetado y despliegue en múltiples entornos. Estas aplicaciones suelen depender frecuentemente de un gran número librerías y *frameworks* de terceros, lo que implica que hay que descargar manualmente los *jar* que se necesitan en el proyecto y copiarlos manualmente en el *classpath* o configurar la herramienta de desarrollo utilizada para que lo haga.

Este proceso es muy tedioso y propenso a errores. Hay que descargar las librerías, comprobar que sean las versiones correctas, obtener las librerías de las que éstas dependen a su vez, etc. y repetir el proceso en todos los ordenadores involucrados.

Por otro lado, en los proyectos también se suelen gestionar recursos estáticos (CSS, Javascript, documentos...), ficheros de configuración, descriptores de despliegue; se realizan pruebas unitarias y de integración, y normalmente se integran en sistemas de control de versiones.

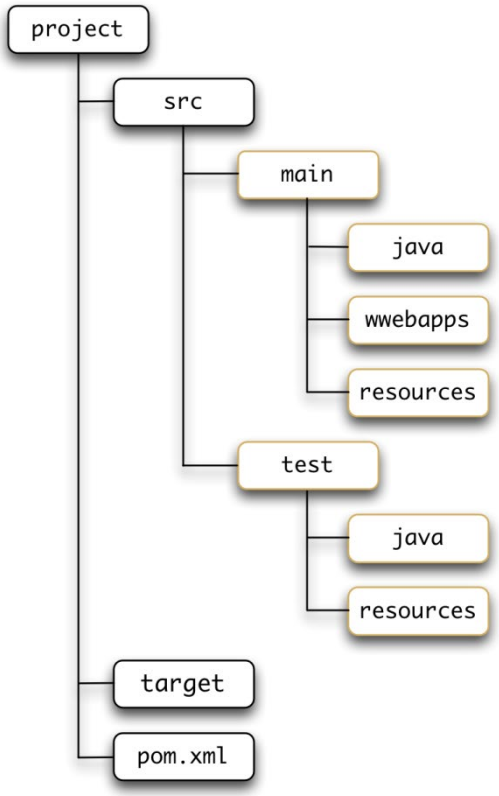
Todo esto complica el desarrollo, y es donde Apache Maven interviene para solucionar estos problemas. Maven es un "Project Management Framework", esto es, un *framework* de gestión de proyectos de software, que proporciona un modelo estándar de gestión y descripción de proyectos. Trabaja en línea de mandatos como el JDK (javac, java,etc.). Es una herramienta muy popular en proyectos open source que da soluciones a tareas que abarcan desde la compilación hasta la distribución, despliegue y documentación de los proyectos. Por ej.:

- La descarga de las librerías (ficheros JAR) externas de las que depende un proyecto es automática, al igual que las dependencias de estas librerías de forma recursiva.
- Se integra con los sistemas de control de versiones.
- Se encarga de la construcción, prueba y despliegue del proyecto desarrollado, produciendo el fichero JAR o WAR final a partir de su código fuente y del fichero de descripción del proyecto POM (*Project Object Model*), que es un fichero XML (llamado *pom.xml*) situado en el directorio raíz del proyecto).

En este fichero POM Maven obliga a definir un identificador único para cada proyecto que desarrollemos, así como a declarar sus características (URL, versión, librerías que usa, tipo y nombre del artefacto generado, etc.).

Estructura de directorios de un proyecto Maven

A continuación, se muestra la estructura de directorios de un proyecto Maven:



Dependencias de librerías

Con Maven solo se necesita definir en el POM las dependencias que se necesitan y Maven las descarga de Internet y las añade al *classpath*, incluyendo las que a su vez se necesiten y resolviendo conflictos de versiones.

Para añadir una librería a un proyecto gestionado con Maven, primero tendremos que buscarla en el repositorio de la herramienta en *Maven Repository*. Una vez localizada la librería y seleccionada su versión (tiene un buscador) hay que copiar el código XML que describe la librería como dependencia al fichero pom.xml de nuestro proyecto, como el siguiente ejemplo:

```
<dependency>
   <groupId>org.eclipse.persistence</groupId>
   <artifactId>eclipselink</artifactId>
   <version>2.7.10</version>
</dependency>
```

Maven se encarga de descargar todas las bibliotecas necesarias para un proyecto cuando ejecutamos el comando `install`. Las guarda en el denominado *repositorio local*, el directorio oculto `.m2` en el directorio raíz del usuario, y después copia referencias a ellas en el proyecto.

Ciclo de vida de Maven

Las partes del ciclo de vida principal del proyecto Maven son:

- compile: Genera los ficheros .class compilando los ficheros fuente .java
- test: Ejecuta las pruebas automáticas de JUnit existentes, abortando el proceso si alguno de ellos falla.
- package: Genera el fichero .jar con los .class compilados
- install: Copia el fichero .jar a un directorio de nuestro ordenador donde Maven deja todos los .jar. De esta forma esos .jar pueden utilizarse en otros proyectos Maven en el mismo ordenador.
- deploy: Copia el fichero .jar a un servidor remoto, poniéndolo disponible para cualquier proyecto Maven con acceso a ese servidor remoto.

Cuando se ejecuta cualquiera de los comandos Maven, por ejemplo, si ejecutamos `mvn install`, Maven irá verificando todas las fases del ciclo de vida desde la primera hasta la del comando, ejecutando solo aquellas que no se hayan ejecutado previamente.

Maven con Eclipse

Eclipse proporciona una interfaz amigable para la gestión de proyectos Maven que libera al usuario de tener que escribir a mano el fichero de configuración POM, facilita la búsqueda de bibliotecas de funciones en los repositorios y proporciona acceso a los distintos comandos de Maven sin necesidad de emplear la terminal.

Los proyectos de Java de Eclipse por defecto (menú de Eclipse *File | New | Java Project*) no emplean Maven, así que si se desea crear un proyecto gestionado por Maven hay que pedirlo explícitamente mediante la opción *File | New | Project*; en el diálogo que se abrirá, dentro de la carpeta *Maven* se encuentra la opción *Maven Project*, que deberá seleccionarse, y en el diálogo siguiente hay que marcar la opción *Create a simple project*. Lo único que queda es rellenar los campos *Group Id*, que identifica a la organización creadora del proyecto, y *Artifact Id*, que identifica el proyecto en sí.

Referencias web:

https://jarroba.com/maven-en-eclipse/

Desarrollo de la capa de datos

Para el desarrollo de la capa de datos se partirá del esqueleto del código de la clase `Locker` proporcionado, y también de la definición de la interface genérica `DAO` vista en teoría:

Interface genérica DAO<T,K>

Definición de la interfaz genérica con las operaciones comunes necesarias para gestionar mínimamente la persistencia de los datos, según el modelo visto en clase de teoría.

All Methods		
T	**find**(K id)	Devuelve el elemento con el código dado ó null si no existe.
T	**create**(**T** c)	Añade un nuevo elemento. Si el elemento ya existe lanzará la excepción DAOException. Devuelve el elemento creado.
void	**delete**(**T** c)	Borra el elemento indicado de los elementos almacenados. Si el elemento no existe lanzará la excepción DAOException.
java.util.List<**T**>	**findAll**()	Devuelve una lista con todos los elementos almacenados
T	**update**(**T** c)	Actualiza un elemento de los almacenados. Si el elemento no existe lanzará la excepción DAOException. Devuelve el elemento actualizado.

Para los errores indicados en los métodos de la interfaz, y cualquier otro posible error, se lanzará la excepción **`DAOException`**, que se deberá crear heredando de `RuntimeException`.

Clase LockerMapDAO

Esta clase implementará la interfaz anterior utilizando un mapa (`Map<Long, Locker>`) para almacenar la información de los armarios de taquillas. Al tratarse de una variable en memoria se perderá su contenido al salir de la aplicación, por lo que se trata únicamente de una prueba.

Pruebas

Para finalizar se realizarán las pruebas de la capa de datos. Este programa solo se encargará de comprobar que funcionan **todas** las operaciones anteriores de la capa de datos, tanto en el caso de que se ejecuten correctamente como incorrectamente, permitiendo validar su funcionamiento.

Para las pruebas se empleará la biblioteca *JUnit*, que permite realizar de forma automática tantas pruebas como se desee sin otro requisito que crear una o más clases cuyo nombre comience o termine en "Test" dentro del directorio *src/test/java* con métodos sin valor de retorno (void) y con la anotación *@Test* (*org.junit.Test* en realidad).

Se pueden ejecutar las pruebas de una clase "Test" individual (menú *Run As / JUnit test)* o ejecutar todas las pruebas de todas las clases "Test" del proyecto *(menú Run As / Maven test*). Al ejecutar las pruebas el entorno *JUnit* llamará a todos los métodos @Test y dará como buenos todos los que completen su ejecución sin generar una excepción, por lo que la biblioteca *JUnit* proporciona, dentro del paquete `org.junit.Assert`, métodos que permiten evaluar relaciones de igualdad, comprobaciones de valores verdaderos o falsos y generan excepciones en caso de que no se cumpla la condición especificada.

A modo de guía, se proporciona el siguiente ejemplo, que no se debe en modo alguno considerar completo, puesto que debe ser completado con pruebas exhaustivas de la funcionalidad de las clases que se pide desarrollar en la práctica:

```
package paa.locker;
import paa.locker.persistence.Locker;
import org.junit.Test;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;


public class LockerTest {
    @Test
    public void testEquals () {
        Locker locker1 = new Locker();
        Locker locker2 = new Locker();
        locker1.setCode(1L);
        locker2.setCode(1L);
        assertTrue(locker1.equals(locker2));
        locker2.setCode(2L);
        assertFalse(locker1.equals(locker2));
    }
}
```

Pasos a seguir

Los pasos que se recomienda seguir en el desarrollo de la práctica son los siguientes:

1. Importar **el proyecto plantilla Maven proporcionado** (menú *File / Open Projects from File System*).
2. Dentro del paquete `paa.locker.persistence` definir las siguientes clases:

   a. La clase `Locker`, completándola. Utilizar las opciones del menú de eclipse *Source*|*Generate* … para generar *setters*, *getters*, *equals*, etc.
   b. Crear la interface `DAO` genérica
   c. Crear la clase `LockerMapDAO` implementando la interfaz genérica anterior (indicando las clases reales de sus parámetros `T` y `K`)
   d. Crear la clase `DAOException`
3. Completar la clase `LockerTest` con el código para probar la clase `Locker` y crear otra clase `MapDAOTest` que contenga código para probar la funcionalidad de los métodos de la interface *DAO* utilizando como implementación la clase `LockerMapDAO`.

Calendario de entrega

Se entregará la siguiente documentación en formato electrónico a través de la tarea de entrega habilitada en el campus virtual en Moodle:

- Un fichero ZIP con la **estructura de ficheros completa del proyecto Eclipse**.

La duración de la práctica es de 1 semana.

Normas de evaluación

- No se admitirá la entrega de ninguna práctica fuera de los plazos de entrega.
- La práctica debe funcionar en su totalidad correctamente, por lo que se debe poner especial cuidado en las pruebas.
- La copia de las prácticas entre alumnos implicará automáticamente la aplicación de la normativa de la UPM para estas situaciones a todos los alumnos involucrados.