
# MEMORIA

Mario Blanco Domínguez
Iker Burgoa Muñoz
David García Sánchez
Javier I. Sotelino Barriga
Francisco Boccassi
Alberto Chaves López
Fernando Quijada Díaz
Yule Zhang

# ÍNDICE

# ARQUITECTURA

La aplicación está estructurada por una arquitectura multicapa, la cual posee tres capas: integración, negocio y presentación.

La capa de integración es la responsable de llamar a base de datos , es llamada por el servicio de aplicaciones de negocio tras una serie de comprobaciones de los datos y acceden al DAO, donde sus funciones ya no comprueban validaciones de los datos, directamente se encargan de extraer o guardar datos a la base de datos.

La capa de negocio, en ella se implementa el servicio de aplicaciones y los transfers del sistema. Aquí se validarán las reglas de negocio y comprobaciones de que algunos datos puedan existir y que posean el formato correcto. En el momento de estar todo validado hará llamada a la capa de integración para su posterior guardado o extracción de datos.

La capa de presentación encapsula la lógica de presentación para dar el servicio al cliente. Posee una interfaz gráfica donde el usuario podrá interactuar con ella y según sus necesidades llamará al controlador mediante los ActionListener y llama al servicio de aplicación deseado.

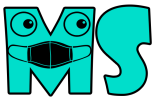
En esta capa se crean los transfer y también se hace unas series de validaciones de inserción de datos por teclado y en el momento que todo sea correcto llama a negocio.

# DIAGRAMAS

A continuación se hará un listado de los diagramas realizados con el programa IBM Rational Architect Designer del proyecto.

## DIAGRAMA DE CLASES

- **Integración**
    - Cliente::DAOCliente
    - Producto::DAOProducto
    - FactoriaDAO

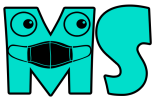


- Marca::DAOMarca
- query::Query
- Transaction::Transaction
- Venta::DAOVenta

- **Negocio**
  - Cliente::SACliente
  - FactoriaSA
  - Marca::SAMarca
  - Producto::SAProducto
  - Venta::SAVenta
  - Curso::SACurso
  - Empresa::SAEmpresa
  - EntityManagerFactory::EntityManagerFactory
  - Trabajador::SATrabajador
  - Departamento::SADepartamento
- **Presentación**
  - GuiCliente
  - GuiMain
  - GuiVenta
  - CommandCliente
  - CommandTrabajador
  - ClaseCommand
  - ClaseController
  - ClaseDispatcher
  - GUICurso
  - GUITrabajador
  - GUIDepartamento
  - GUIEmpresa

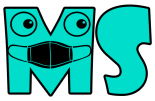
## DIAGRAMA DE SECUENCIAS

- **Integración**
  - Cliente::DAOCreateCliente
  - Cliente::DAODeleteCliente



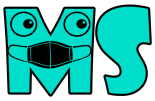
- Cliente::DAOReadAllCliente
- Cliente::DAOReadCliente
- Cliente::DAOUpdateCliente
- Query::NewQuery
- Query::ClienteXY
- Transaction::CreateTransaction
- Transaction::DeleteTransaction
- Transaction::GetTransaction
- Transaction::NewTransaction
- Transaction::Commit
- Transaction::Rollback
- Transaction::start
- Venta::DAODeleteVenta
- Venta::DAODevolucionVenta
- Venta::DAOReadAllOneClient
- Venta::DAOReadAllVenta
- Venta::DAOReadVenta
- Venta::DAORealizarVenta
- Venta::DAOUpdateVenta

- **Negocio**
  - Cliente::CreateCliente
  - Cliente::DeleteCliente
  - Cliente::ReadAllCliente
  - Cliente::ReadCliente
  - Cliente::UpdateCliente
  - Cliente::queryXY
  - Departamento::calcularNomina
  - Empresa::actualizarEmpresa
  - Empresa::altaEmpresa
  - Empresa::bajaEmpresa
  - Empresa::listarEmpresa
  - Empresa::mostrarEmpresa
  - Trabajador::altaTrabajador



- Trabajador::bajaTrabajador
- Trabajador::asignarCurso
- Trabajador::desvincularCurso
- Trabajador::mostrarCursosTrabajador
- Trabajador::mostrarTrabajador
- Trabajador::mostrarTrabajadores
- Trabajador::updateTrabajador
- Venta::AbrirVenta
- Venta::DeleteVenta
- Venta::DevolucionVenta
- Venta::ReadAllVenta
- Venta::ReadVenta
- Venta::RealizarVenta
- Venta::UpdateVenta
- Venta::ReadAllOneCliente

- **Presentación**
  - Cliente::ActionListenerAltaCliente
  - Cliente::ActionListenerGUICliente
  - Cliente::UpdateGUICliente
  - Venta::ActionListenerAñadirProducto
  - Venta::ActionListenerQuitarProducto
  - Command::Baja Cliente
  - Command::cmListarEntreXY
  - Command::cmListarTodosCliente
  - Command::cmMostrarDatosCliente
  - Command::cmActualizarDatosCliente
  - Command::cmAltaCliente
  - Command::cmAltaTrabajador
  - Command::cmBajaTrabajador
  - Command::cmAsignarCursoTrabajador
  - Command::cmDesvincularTrabajador
  - Command::cmModificarTrabajador
  - Command::cmMostrarCursosTrabajador



- Command::cmMostrarTrabajador
- Command::cmMostrarTrabajadores
- Command::getCommand
- Command:getCommand_JPA
- Controller::Controller
- Dispatcher::Dispatcher
- Curso::actionListenerAltaCurso
- Departamento::actionListenerAltaDepartamento
- Departamento::actionListenerBajaDepartamento
- Departamento::actionListenerCalcularNominaDepartamento
- Departamento::actionListenerGUIDepartamento
- Departamento::actionListenerModificarBuscarIdDepartamento
- Departamento::actionListenerMostrarDepartamento
- Departamento::actionListenerMostrarDepartamentos
- Departamento::actionListenerMostrarTrabajadoresDepartamento
- Departamento::actionListenerUpdateDepartamento
- Departamento::updateGUIDepartamento
- Empresa::actionListenerAltaEmpresa
- Trabajador::actionListenerAltaTrabajadorI

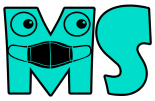
## DIAGRAMA DE DESPLIEGUE.

- Diagrama de despliegue.

# PATRONES

## PATRÓN SINGLETON

El objetivo del patrón Singleton es el hecho de poder garantizar que en una clase posea una sola instancia y sea de un punto de acceso global a ella. El uso de este patrón proporciona al proyecto una serie de beneficios como la reducción de variables globales, ya que no reservamos espacio, sólo existen instancias.



El control de acceso a la única instancia, el refinamiento de operaciones y representación,permite un número variable de instancias, es fácilmente configurable y es más flexible que las operaciones de clase.

## PATRÓN TRANSFER

El objetivo del patrón TOA es obtener un modelo que agregue objetos de transferencia de distintos componentes de negocio. Se encapsula la lógica de negocio de forma centralizada, se desea minimizar las llamadas a objetos remotos y ofrecer un modelo complejo en presentación.

El TOA agrega estos objetos de transferencia provenientes de otros componentes.

## PATRÓN DAO

El objetivo del patrón DAO es separar por completo la lógica de negocio de la lógica para poder acceder a los datos, así proporcionar esos métodos necesarios para inserciones, actualizaciones, consultas de datos y también la capa de negocio solo se preocupa por la lógica de negocio y utiliza el DAO para interactuar con la fuente de datos.

## PATRÓN SERVICIO DE APLICACIÓN

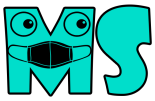
El objetivo del patrón Servicio de Aplicación es centralizar la lógica de negocio mediante distintos componentes de la capa de negocio y servicios.

También mejora la reusabilidad de la lógica de negocio y evita posibles duplicaciones de código.

## PATRÓN COMMAND

El patrón Command aporta una interfaz donde se puede crear familias de objetos relacionados, dando una serie de listas de clases de producto donde sólo se sabe sus interfaces y no sus implementaciones.

Independiza al sistema de cómo se crean y se representan sus productos.



## PATRÓN CONTEXTO

Con el uso de este patrón se desea poder evitar utilizar información del sistema específica del protocolo fuera de su contexto relevante.

Con este patrón se mejora la reusabilidad y la mantenibilidad, las pruebas se mejoran y se reducen las restricciones en la evolución de GUIs, aunque se reduce el rendimiento.

## PATRÓN CONTROLADOR

El patrón Controlador centraliza el control de la gestión de acciones y de las vistas. También mejora la reutilización de código de gestión en vistas y acciones y la extensibilidad del manejo de peticiones.

Relaciona las capas de presentación con la de negocio.

## PATRÓN FACTORÍA ABSTRACTA

El patrón Factoría Abstracta nos proporciona una interfaz para poder crear familias de objetos relacionados. También nos permite configurar familias de productos dentro del sistema software, independiza al sistema de cómo se crean y se representan los productos y aporta una lista de clases de productos, donde sólo se sabe sus interfaces y no las representaciones.

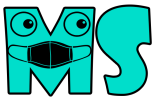
## PATRÓN QUERY

La idea de este patrón es poder desvincular de los DAOs las querys que sean más complejas.

Cuando una query, más allá de ser una simple inserción, modificación o selección de base de datos, se convierte en una query compleja, por ejemplo , donde existen relación entre varias tablas, se puede desvincular de la parte DAO e incluirlo como objeto query.

## PATRÓN ALMACÉN DEL DOMINIO

La idea del patrón es separar la persistencia del modelo de objetos. Se caracteriza ya que hace que el sistema sea persistente, gestione carga dinámica, resuelva problemas de gestión de transacciones y concurrencia, todo ello gracias al Transaction Manager.



# JPA

La segunda parte del proyecto corresponde a JPA, donde se ha realizado una gestión de concurrencia optimista, caracterizada por detectar los conflictos, en vez de evitarlos. La parte del equipo era gestionar aquellos casos donde se puedan dar bloqueos y se use el bloqueo optimista con incremento forzado.

Como entidades de nuestro proyecto tenemos: Departamento, Curso, Trabajador y Empresa. Destacar del departamento, posee una función que calcule la nómina de todos los empleados y se ha usado polimorfismo en la entidad Trabajador (Indefinido y Temporal) para que el cálculo de sus nóminas sea distinto y se especialice.

# TEST DAO

Para la realización de los test de la parte DAO, en cada clase de test se hace una limpieza de la base de datos para que los resultados que se quieren obtener sean los deseados.

Los test están definidos en un orden concreto para la reutilización de objetos en las siguientes pruebas que se realicen de su misma clase.

## PRODUCTOS

### CREAR PRODUCTO – SA

Para la creación de producto del servicio de aplicaciones, con la tabla vacía se insertan cuatro productos. El primer producto es una inserción correcta con todos los datos correctos, el segundo producto posee un identificador de marca que no existe, el tercer producto tiene una marca que ha sido dada de baja y el cuarto producto posee un nombre ya existente en base de datos.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertTrue | Se inserta un producto correctamente | El identificador debe ser mayor que cero | True |
| assertTrue | Se inserta un producto con un identificador de marca que no existe | El identificador es igual a -7 | True |
| assertTrue | Se inserta un producto con una marca que haya sido dada de | El identificador es igual a -13 | True |

| | | | |
|---|---|---|---|
| | baja anteriormente | | |
| assertEquals | Se inserta un producto con un nombre de producto que ya existe en base de datos | El identificador es igual a -39 | True |

## ELIMINAR PRODUCTO – SA

Para la eliminación de un producto del servicio de aplicaciones, comprobaremos tres productos, el primero con un producto existente en la base de datos, el segundo el mismo producto ya eliminado, da error ya que su campo activo ha sido puesto a false y otro con otro producto inexistente.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertEquals | Se elimina un producto correctamente | El identificador de entrada al llamar al delete debe ser igual al id que devuelve la función. | True |
| assertFalse | Se elimina el producto anterior | El identificador de entrada es igual al de salida de la función delete | True |
| assertTrue | Se elimina un producto no existente en la bbdd | El identificador que devuelve el delete debe ser -5 | True |

## MODIFICAR PRODUCTO – SA

Para la modificación de productos se insertan tres productos con datos correctos para su posterior testeo. El primer y el tercer producto serán de la misma marca y el segundo de una marca distinta. Al primer producto se le modificarán datos simples como el precio, su descripción.. y se hará un update en base de datos correctamente.

Al segundo producto se le modificará la marca , añadiendo una marca inexistente en la base de datos. Para el tercero, se eliminará de la base de datos el producto dos y la marca del anterior producto para ponerle ese producto, añadiendo así una marca dada de baja, no dejando su inserción en base de datos.

| Método | Prueba | Comprobación | Resultado |
|---|---|---|---|

| | | | Esperado |
|---|---|---|---|
| assertTrue | Se modifica un producto correctamente | El identificador creado en el producto debe ser igual al identificador que devuelve la función update | True |
| assertTrue | Se modifica un producto con un identificador de marca que no existe | El identificador creado en el producto es distinto al identificador que devuelve la función update | True |
| assertTrue | Se modifica un producto con un identificador de marca dado de baja | El identificador devuelto debe ser -13 | True |
| assertEquals | Se modifica un producto cambiando su nombre a un nombre ya existente en base de datos | El identificador devuelto debe ser igual a -39 | True |

### MOSTRAR UN PRODUCTO – SA

Para la lectura de un producto se harán dos pruebas con dos identificadores, uno con un id existente y otro con un identificador que no exista en la base de datos.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertTrue | Se leerá un producto correctamente | El identificador de entrada de la función read debe ser igual al identificador del producto devuelto. | True |
| assertTrue | Se leerá un producto que no existe | El producto devuelto es nulo | True |

### MOSTRAR TODOS LOS PRODUCTOS – SA

Para la lectura de todos los productos se llama a la función redlAll() y devuelve una lista de productos, en caso de estar vacía se comprueba que es cero y en el caso contrario se recorre en un bucle que cada producto no es nulo.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertNotNull | Se leerá todos los | Los productos existen en la | True |

| | | | |
|---|---|---|---|
| | productos | base de datos | |
| assertTrue | La lista está vacía | El tamaño de la lista es cero | True |

# MARCA

## CREAR MARCA- SA

Para la creación de marcas se crean cuatro marcas donde las dos primeras son inserciones correctas, la tercera marca posee un número de teléfono erróneo y la última marca da un error debido a que su nombre es el mismo al de una marca anteriormente insertada correctamente.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertTrue | Se inserta una marca con datos válidos | Devuelve un id > 0 | True |
| assertTrue | Se inserta una marca con datos válidos | Devuelve un id > 0 | True |
| asserTtrue | Se inserta una marca con un teléfono incorrecto | Devuelve un identificador con valor -22 | True |
| assertTrue | Se inserta una marca con un nombre ya existente | Devuelve un identificador con valor -37 | True |

## ELIMINAR MARCA- SA

En la eliminación de marcas, se insertan dos marcas con valores correctos y un producto creado con el identificador de la segunda marca para la comprobación de si existen marcas con productos. La primera marca anteriormente creada se elimina correctamente, a continuación se vuelve a eliminar la misma marca dando un error debido a que ya fue dada de baja. La segunda marca, al intentar eliminar produce un error debido a que tiene un producto asociado a la marca. También se comprueba hacer una eliminación de una marca no existente.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertEquals | Se elimina una marca con | El identificador de la marca | True |

| | | | |
|---|---|---|---|
| | datos válidos | es igual al valor devuelto por la función delete | |
| assertTrue | Se elimina de nuevo la anterior marca ya eliminada | El identificador vale -13 | True |
| asserTtrue | Se elimina una marca inexistente | El identificador vale -21 | True |
| assertTrue | Se elimina una marca con productos asociados a ella | El identificador vale -7 | True |

## MODIFICAR MARCA – SA

Para la modificación de marcas se crean dos marcas con datos correctos y se insertan en base de datos, a continuación a uno de ellos se le cambia el correo con un valor válido y a la otra marca se le cambia el teléfono con valores incorrectos. Luego se restaura correctamente el teléfono de la última marca y se modifica su nombre al de una marca ya existente dando también error.

| **Método** | **Prueba** | **Comprobación** | **Resultado Esperado** |
|---|---|---|---|
| assertEquals | Se modifica una marca con valores correctos | El identificador de la marca es igual al valor devuelto por la función update | True |
| assertEquals | Se modifica una marca con el teléfono mal | El identificador vale -22 | True |
| assertEquals | Se modifica una marca con un nombre ya existente en base de datos | El identificador vale -37 | True |

## LEER UNA MARCA –SA

Para la lectura de marcas, al tener insertadas en las pruebas anteriores, se recoge el identificador 1 y un identificador con valor elevado (5578) a sabiendas de que no existe.

La primera llamada devolverá la marca y la segunda llamada devolverá nulo debido a que no existe.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertTrue | Se hace una lectura del identificador 1 que existe en base de datos | Devuelve una marca y se comprueba que el id de la marca es 1 | True |
| assertEquals | Se hace una lectura del identificador 5578 que no existe en base de datos | Se comprueba que la marca devuelta vale nulo | True |

### LEER TODAS LAS MARCAS -SA

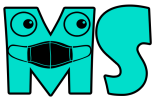
Para la comprobación de todas las marcas se recoge en una lista el total de marcas registradas en la base de datos, en caso de no haber, se comprueba que la lista es cero y en caso de que existan marcas, se recorre las marcas comprobando que no son nulas.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertTrue | La lista está vacía | El tamaño de la lista es 0 | True |
| assertNotNull | Se comprueba que cada marca existe y no es nula | La marca recorrida es diferente de nula | True |

### LEER TODOS LOS PRODUCTOS DE UNA MARCA -SA

Se crea una marca con datos correctos y se crean tres productos para asociarlos a dicha marca. En caso de que la lista de la marca esté vacía se comprueba que su tamaño es de cero y en caso de tener productos se comprueba que son distintos y que la marca del producto coincide con la marca anteriormente creada.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertEquals | La lista está vacía | El tamaño de la lista es 0 | True |
| assertNotNull | Se comprueba que cada producto existe y no es nulo | El producto recorrido es diferente de nula | True |
| assertEquals | El identificador de la marca del producto es el mismo al de la marca | Producto.getMarca == id_marca | True |



| | | | |
|---|---|---|---|
| | creada | | |

## CLIENTE

Para los test de cliente se empezará con la base de datos vacía para evitar problemas en la revisión. Se crea al principio un array con tres clientes: dos empresariales y un particular donde posteriormente se insertarán para ser usado de prueba.

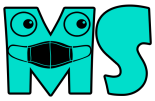
También se crearán dos arrays, en uno se guardarán los identificadores de cliente y en otro los identificadores de los productos junto a una venta inicial a valor nulo.

### CREAR CLIENTES – SA

Se crean cinco nuevos clientes: tres empresariales, los cuales uno de ellos su CIF es repetido a un CIF de los clientes empresariales del array creado, otro tiene el correo mal y otro el número de cuenta mal.

Los clientes particulares, uno de ellos tiene el DNI mal y otro posee el mismo DNI que el cliente particular del array.

| **Método** | **Prueba** | **Comprobación** | **Resultado Esperado** |
|---|---|---|---|
| assertTrue | Se inserta el cliente empresarial 1 de la lista correctamente | La función create devuelve 1 y se comprueba que ese id es 1 | True |
| assertTrue | Se inserta el cliente empresarial 2 de la lista correctamente | La función create devuelve 2 y se comprueba que ese id es 2 | True |
| assertTrue | Se inserta el cliente particular de la lista correctamente | La función create devuelve 3 y se comprueba que ese id es 3 | True |
| assertTrue | Se inserta el cliente empresarial con CIF ya creado | Se compara que el resultado devuelto es igual a -31 | True |



| | | | |
|---|---|---|---|
| assertTrue | Se inserta el cliente particular con DNI ya creado | Se compara que el resultado devuelto es igual a -32 | True |
| assertTrue | Se inserta el cliente particular con el DNI malo | Se compara que el resultado devuelto es igual a -36 | True |
| assertTrue | Se inserta el cliente empresarial con el número de cuenta mal | Se compara que el resultado devuelto es igual a -35 | True |
| assertTrue | Se inserta el cliente empresarial con el correo mal | Se compara que el resultado devuelto es igual a -34 | True |

## LEER UN CLIENTE- SA

La idea para la lectura es coger los valores del primer cliente empresarial que hay en base de datos (insertado anteriormente con identificador 1) y comparar si los valores que nos devuelve son los mismos que el cliente de base de datos.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertTrue | Se intenta leer el cliente con identificador (99999) no existente | La función devuelta se compara con null | True |
| assertTrue | Se intenta leer el cliente con identificador 1 y comprobar su activo | El activo del cliente 1 es true | True |
| assertTrue | Se intenta leer el cliente con identificador 1 y comprobar su id | El identificador del cliente 1 es 1 | True |
| assertTrue | Se intenta leer el cliente con identificador 1 y comprobar su teléfono | El teléfono del cliente 1 es 912345678 | True |
| assertEquals | Se intenta leer el cliente con identificador 1 y comprobar su correo | El correo del cliente 1 es cliente_1@gmail.com | True |
| assertEquals | Se intenta leer el cliente con identificador 1 y comprobar su dirección | La dirección del cliente 1 es calle peral | True |

| | | | |
|---|---|---|---|
| assertEquals | Se intenta leer el cliente con identificador 1 y comprobar su nombre | El nombre del cliente 1 es Mapfre | True |
| assertEquals | Se intenta leer el cliente con identificador 1 y comprobar su CIF | El CIF del cliente 1 es 11416988H | True |

## MODIFICAR CLIENTES – SA

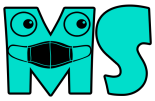
Se desea modificar tres clientes: dos empresariales, el cual uno de ellos tendrá mal su número de cuenta y otro tendrá mal su email, y un cliente particular con el dni incorrecto.

También se hará la comprobación con un update correcto.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertTrue | Se intenta modificar el cliente particular con el DNI incorrecto | El identificador devuelto es -36 | True |
| assertTrue | Se intenta modificar el cliente empresarial con el número de cuenta incorrecto | El identificador devuelto es -35 | True |
| assertEquals | Se intenta modificar el cliente empresarial con el correo incorrecto | El identificador devuelto es -34 | True |
| assertTrue | Se modifica el cliente correctamente (con identificador 1 ) | El identificador devuelto es el identificador 1 del cliente | True |
| assertEquals | Se comprueba la lectura del cliente que la dirección ha sido modificada | La dirección nueva del cliente es calle trucha | True |

## ELIMINAR CLIENTES – SA

Al eliminar clientes se comprobará la eliminación de un cliente con un identificador no existente en base de datos, una eliminación de un cliente válido y a posteriori de nuevo su eliminación siendo un error, debido a que fue eliminado anteriormente.



Se crea una marca y una serie de productos asociados a dicha marca y abrimos una venta con el cliente activo. Realizamos una compra con el cliente y se procede a su eliminación , dará error debido a que tiene una venta asociada a dicho cliente.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertTrue | Se intenta eliminar un cliente inexistente (identificador : 9999) | El identificador devuelto es -3 | True |
| assertTrue | Se elimina correctamente un cliente (Identificador: 1) | El identificador devuelto es 1, que es su identificador | True |
| assertTrue | Se elimina el cliente anterior de nuevo (Identificador: 1) | El identificador devuelto es -41 | True |
| assertEquals | Se elimina el cliente (modificado a activo) con una venta activa | El identificador devuelto es -42 | True |

## LEER TODOS LOS CLIENTES – SA

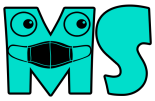
Para la lectura completa de los clientes se hace una comprobación que el número de clientes en la lista de clientes devuelto es un total de 3.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertTrue | Se comprueba el número de clientes que tiene la lista de clientes | El total de la lista es de 3 | True |

## CLIENTES CON VENTAS EN UN INTERVALO DE PRECIOS– SA

Para la lectura de los clientes que hayan hecho compras entre un rango de precios , se harán dos comprobaciones, la primera comprueba que en el intervalo [0.5 , 1] la función devuelve 1 cliente y la segunda comprueba que en el intervalo [200 , 300] la función devuelve 0 clientes.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|



| | | | |
|---|---|---|---|
| assertTrue | Se comprueba el número de clientes que sus compras están entre 0.5 y 1 | El total de la lista es de 1 | True |
| assertTrue | Se comprueba el número de clientes que sus compras están entre 200 y 300 | El total de la lista es de 0 | True |

## VENTA

Para probar la venta, se borra todo lo que existe en la base de datos, y se procede a insertar varias entidades necesarias para probar los métodos de venta.

Se inserta una marca, que servirá para poder insertar productos.

Se insertan 4 productos, siendo completamente válidos el primero y el segundo, el tercero válido pero no se podrá comprar al no tener stock, y el cuarto tampoco válido ya que no estará activo.

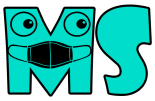
Se insertan también 3 clientes, siendo el primero (cliente empresa) y el tercero (cliente particular) válidos , mientras que el segundo no estará activo.

Añadir producto en venta y quitar producto en venta no se prueban ya que son métodos que no pasan por el SA, la venta se va guardando localmente y al realizarla se prueba todo.

### REALIZAR VENTA – SA

Se crea una venta para el primer cliente. Se añaden 10 unidades del primer producto y 5 del segundo. Tras hacer los cálculos de precios que hace la GUI, se realiza la venta.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertTrue | Se realiza la venta y se obtiene su id | Id >0 | True |
| assertEquals | Se comprueba que le ha bajado el stock al primer producto | read(producto 1) .stock == 990 | True |
| assertEquals | Se comprueba que le ha bajado el stock al segundo producto | read(producto 2) .stock == 995 | True |



Esta venta se utilizará en más tests.

A continuación se prueba a realizar otras ventas con datos incorrectos.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertEquals | Venta con id cliente 9999(no existe) | id == -3 | True |
| assertEquals | Venta con el segundo cliente (no activo) | id == -3 | True |
| assertEquals | Venta con el producto 9999 (no existe) | id==-5 | True |
| assertEquals | Venta con el cuarto producto (no activo) | id==-5 | True |
| assertEquals | Venta con el tercer producto (no tiene stock) | id==-11 | True |

## DAR DE BAJA VENTA – SA

Se prueba a eliminar la venta (que se creó en el test anterior) y más pruebas

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertEquals | Se borra la venta creada en el paso anterior | id venta == delete(idventa) | True |
| assertEquals | Se borra la venta creada en el paso anterior OTRA VEZ | delete(idventa) == -12 | True |
| assertEquals | Se borra la venta con id 9999(no existe) | delete(9999) == -1 | True |

## ACTUALIZAR UNA VENTA – SA

Se prueba a actualizar la venta (que se creó en el primer test) y más pruebas incorrectas

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|

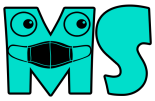
| | | | |
|---|---|---|---|
| assertEquals | Se prueba a reactivar la venta | idVenta== update(venta) | True |
| assertEquals | Se lee para ver si realmente se activó | read(idVenta).activo ==true | True |
| assertEquals | Se cambia la dirección de entrega | idVenta== update(venta) | True |
| assertTrue | Se lee para ver si realmente se cambió la dirección | read(idVenta).dirección.equals (nueva) ==true | True |
| assertEquals | Se actualiza una venta que no existe | update(new TVenta(9999,...) == -1 | True |

## DEVOLVER UN PRODUCTO DE UNA VENTA – SA

Se prueba a devolver un producto de una venta(que se creó en el primer test) y más pruebas incorrectas. Devolución devuelve un TVenta, que es la venta actualizada.

Se devuelven 5 unidades del primer producto, del cual se habían comprado 10.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertEquals | Se prueba que se ha devuelto | ventaActualizada.id== devolucion ().id | True |
| assertTrue | El precio de la venta ha bajado | ventaActualizada.precio< venta.precio | True |
| assertEquals | Las cantidades del producto están bien | cantidades iguales | True |
| assertEquals | Se lee para ver si realmente me ha devuelto el producto | read(productoid).stock. == 995 | True |
| assertTrue | Se devuelve un producto que no estaba en la venta | devolucion (producto 9999) ==null | True |
| assertTrue | Se devuelve algo de una venta que no existe | devolucion (venta 9999) ==null | True |



## LEER UNA VENTA – SA

Se prueba a leer una venta(que se creó en el primer test) y más pruebas incorrectas.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertNotNull | No es nula | | True |
| assertEquals | Son iguales direccion | ventaRead.direccion== venta.direccion | True |
| assertEquals | Son iguales estado | ventaRead.estado== venta.estado | True |
| assertEquals | Son iguales fechas | rventaRead.fechas== venta.fechas | True |
| assertTrue | Son iguales id cliente | ventaRead. id cliente== venta. id cliente | True |
| assertNotNull | Carrito no nulo | carrito != null | True |
| (En bucle) assertEquals | Cantidades iguales de producto | ventaRead.producto[i] .cantidad== venta.producto[i].cantidad | True |
| (En bucle) assertEquals | Precios iguales de producto | ventaRead.producto[i] .precio== venta.producto[i].precio | True |
| assertEquals | Se lee una venta que no existe | read(9999) ==null | True |

## LEER TODAS LAS VENTAS – SA

Se prueba a leer todas las ventas

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertNotNull | La lista de ventas devuelta por el SA no es nula | readAll not null | True |

# TEST JPA

## DEPARTAMENTO

### ALTA DE UN DEPARTAMENTO – SA

Para la inserción de departamentos se hará una inserción correcta de un departamento y se comprobará en otro los campos de correo, teléfono y que dicho departamento ya existe en base de datos.

Tras dichas comprobaciones se insertará ese departamento.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertEquals | Inserción departamento correcto | id==1 | true |
| assertEquals | Inserción con correo invalido | id==-83 | true |
| assertEquals | Inserción con telefono invalido | id==-84 | true |
| assertEquals | Inserción con nombre repetido | id==-80 | true |
| assertEquals | Inserción departamento correcto | id==2 | true |

### BAJA DE UN DEPARTAMENTO – SA

Para la baja del departamento se dará de primeras baja a un departamento ya existente en base de datos para después volver a eliminarlo, dando mensaje de error.

También se comprobará que el departamento no existe y no se puede eliminar departamentos con trabajadores.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertEquals | Borrar departamento correcto | id==2 | true |

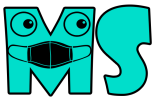
| | | | |
|---|---|---|---|
| assertEquals | Borrar departamento ya borrado | id==-82 | true |
| assertEquals | Borrar departamento que no existe | id==-81 | true |
| assertEquals | Borrar departamento con trabajadores activos | id==-86 | true |

## ACTUALIZAR UN DEPARTAMENTO – SA

Para actualizar departamentos se actualizará de primeras un departamento correctamente y posteriormente se intentará actualizar un departamento que no existe.

Se realizarán las comprobaciones fallidas para los fallos de correo, nombre de departamento, teléfono, dar de baja a un departamento con trabajadores activos y posteriormente activarlo.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertEquals | Actualizar departamento (correo,nombre,direccion y telefono) | id==1 | true |
| assertEquals | Actualizar departamento que no existe | id==-81 | true |
| assertEquals | Actualizar departamento con el nombre repetido | id==-80 | true |
| assertEquals | Actualizar con correo invalido | id==-83 | true |
| assertEquals | Actualizar con telefono invalido | id==-84 | true |
| assertEquals | Actualizar departamento con trabajadores, dando de baja | id==-86 | true |
| assertEquals | Reactivar departamento borrado | id==2 | true |



## MOSTRAR UN DEPARTAMENTO – SA

Para mostrar un departamento se comprobará con el método get cada campo del departamento buscado y se insertará un departamento inexistente para que devuelva un valor nulo.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertEquals (1 x cada atributo) | Leer un departamento que existe(id 1, insertado antes) | t.get... = t1.get... | true |
| assertNull | Leer un departamento que no existe | tRead ==null | true |

## MOSTRAR TODOS LOS DEPARTAMENTOS – SA

Para mostrar todos los departamentos se recogerá la información de todos los departamentos que haya en la base de datos y se hará la comprobación de que hay un total de 2 departamentos.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertEquals | Leer todos los departamentos | lista.size()==2 | True |

## MOSTRAR TODOS LOS TRABAJADORES DE UN DEPARTAMENTO – SA

Para mostrar los trabajadores que posee un departamento se comprobará primero un departamento válido que contenga un número exacto de empleados y la misma comprobación de un departamento sin empleados.

Para finalizar se buscará un departamento que no existe.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assert Equals | El dep 1 tendrá 4 trabajadores, se comprueba eso | lista.size()=4 | True |

| | | | |
|---|---|---|---|
| assertEquals | El dep 2 tendra 0 trabajadores | lista.size()=0 | True |
| assertNull | El dep 999 no existe | lista = null | True |

### CALCULAR NOMINA DE UN DEPARTAMENTO – SA

Para el cálculo de nómina se recogerá el sueldo de cada empleado (dependiendo de su tipo de contrato se recogerá una cosa u otra) y se hará la suma de los totales. Se comprobará el sueldo de los dos departamentos existentes y un tercero que no exista.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertEquals | Nómina del departamento 1 | nomina=5513.75 | True |
| assertEquals | Nómina del departamento 2 | nomina=0 | True |
| assertEquals | Nomina del departamento 999 inexistente | nomina = -81 | True |

## TRABAJADORES

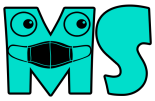
Para trabajadores se trabajará con dos trabajadores, uno temporal y uno indefinido y en la clase se guardaran sus identificadores para su posterior uso en diferentes test.

### ALTA DE UN TRABAJADOR– SA

Antes de crear los trabajadores se debe crear un departamento para dichos trabajadores. Una vez el departamento creado y su identificador guardado en una variable.

Las pruebas que se realizará serán sobre el trabajador temporal donde se insertará mal su correo, el departamento, y el DNI. Cuando se hayan realizado dichas pruebas, se insertará el trabajador correctamente y haremos la inserción del trabajador indefinido con el DNI del anterior trabajador, dando un error. Se acabará insertando para las siguientes pruebas.

| Método | Prueba | Comprobación | Resultado |
|---|---|---|---|



| | | | Esperado |
|---|---|---|---|
| assertEquals | El correo está mal | El identificador devuelto es -98 | True |
| assertEquals | El departamento mal | El identificador devuelto es -81 | True |
| assertEquals | El DNI es incorrecto | El identificador devuelto es -97 | True |
| assertTrue | Inserción correcta | El identificador devuelto es mayor que 0 | True |
| assertEquals | El DNI ya existe | El identificador devuelto es -90 | True |
| assertEquals | Inserción correcta | El identificador devuelto es mayor que 0 | True |

## BAJA DE UN TRABAJADOR – SA

Para las bajas se comprueba que el identificador insertado no existe en base de datos. A continuación se dará de baja a un trabajador y después se dará de baja al mismo trabajador dando error.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertEquals | El identificador no existe | El identificador devuelto es -91 | True |
| assertEquals | Se ha dado la baja correctamente | El identificador devuelto es igual al id del trabajador indefinido | True |
| assertEquals | Se da de baja al trabajador indefinido ya dado de baja | El identificador devuelto es -92 | True |

## ACTUALIZAR UN TRABAJADOR – SA

Para actualizar los trabajadores, recuperaremos al trabajador indefinido y se harán las comprobaciones del correo , departamento y dni inadecuados. Posteriormente se le modificará el campo activo a *false* y la modificación se producirá correctamente.

Al trabajador indefinido también se le modificará su DNI con uno ya existente.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|

| | | | |
|---|---|---|---|
| assertEquals | El correo está mal | El identificador devuelto es -98 | True |
| assertEquals | El departamento mal | El identificador devuelto es -81 | True |
| assertEquals | El DNI es incorrecto | El identificador devuelto es -97 | True |
| assertTrue | Inserción correcta | El identificador devuelto es mayor que 0 | True |
| assertEquals | El DNI ya existe | El identificador devuelto es -90 | True |

### MOSTRAR UN TRABAJADOR – SA

Para mostrar un trabajador se realizará el ejemplo de un trabajador existente, del trabajador anteriormente dado de baja, comprobando que su activo es *false* y otro mostrar en donde el trabajador no exista en nuestra base de datos.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertEquals | Se muestra un usuario | El activo es false | True |
| assertTrue | Se muestra un usuario | El identificador mayor que 0 | True |
| assertTrue | El usuario no existe | El usuario devuelto es nulo | True |

### MOSTRAR TODOS LOS TRABAJADOR – SA

Se recogerá el listado de todos los trabajadores y se comprobará que el identificador es mayor que cero.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertTrue | Se muestra un usuario | El identificador mayor que 0 | True |

### MOSTRAR TODOS LOS CURSOS DE UN TRABAJADOR – SA

Se recogerá el listado de todos los cursos de un trabajador y se comprobará que el listado no es nulo.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertNotNull | Se muestra el listado | El listado no es null | True |
| assertTrue | Se muestra un curso | El identificador mayor que 0 | True |

### ASIGNAR TRABAJADOR Y DESVINCULAR TRABAJADOR DE UN CURSO – SA

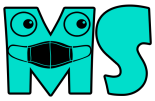
Para asignar o desvincular un trabajador de un curso se comprobará primero que ambos existen en la BBDD y ambos están activos, luego para asignar se comprobará que el trabajador y el curso no están asignados actualmente, para desvincular se comprobará que el trabajador y el curso no están desvinculados actualmente.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertEquals | el curso es null o el trabajador es null | El identificador devuelto es -95 | True |
| assertEquals | el curso no está activo o el trabajador no está activo | El identificador devuelto es -94 | True |
| assertEquals | el curso y el trabajador ya están asignados | El identificador devuelto es -93 | True |
| assertEquals | el curso y el trabajador ya están desvinculados | El identificador devuelto es -96 | True |
| assertEquals | el curso se ha asignado o desvinculado correctamente | El identificador devuelto es mayor que 0 | True |

## CURSOS

Para los cursos se trabajará con un solo curso y en la clase se guardaran su identificador para su posterior uso en diferentes test.

Además se creará un idempresa para trabajar con una empresa en diferentes test.



## ALTA DE UN CURSO – SA

Primero se creará una empresa que irá asociada al curso. Una vez la empresa creada. su identificador se guardará en una variable. Se crearán dos cursos, uno que se cree bien y el otro que dará fallo, ya que repetiremos el nombre del curso anterior.

Las pruebas que se realizarán serán sobre el primer curso donde se insertará mal las horas, el nivel, y el id de la empresa. El segundo curso creado se le asignará el nombre del primero, dando así un error. Cuando se hayan realizado dichas pruebas, se insertará el curso correctamente Se acabará insertando para las siguientes pruebas.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertEquals | El nombre ya existe | El identificador devuelto es -101 | True |
| assertEquals | La hora está mal | El identificador devuelto es -104 | True |
| assertEquals | El nivel está mal | El identificador devuelto es -105 | True |
| assertEquals | La empresa asociada no existe | El identificador devuelto es -74 | True |
| assertTrue | Se ha dado de alta el curso | El identificador devuelto es mayor que 0 | True |

## BAJA DE UN CURSO – SA

Para las bajas se comprueba que el identificador insertado no existe en base de datos. A continuación se dará de baja a un curso y después se dará de baja al mismo curso dando error.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertEquals | No existe el curso | El identificador devuelto es -100 | True |
| assertEquals | Se da de baja el curso | El identificador devuelto es igual al id del curso | True |
| assertEquals | Curso ya dado de baja | El identificador devuelto es -102 | True |

## ACTUALIZAR UN CURSO – SA

Para actualizar un curso se utiliza el curso dado de alta previamente, al cual le iremos insertando erróneamente la empresa(por una que no existe), la hora y el nivel, después crearemos otro curso, y al curso principal le cambiamos el nombre al mismo que tiene el curso nuevo, por último se actualiza el curso de forma correcta.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertEquals | No existe la empresa asociada al curso | El identificador devuelto es -74 | True |
| assertEquals | La hora está mal | El identificador devuelto es -104 | True |
| assertEquals | El nivel está mal | El identificador devuelto es -105 | True |
| assertEquals | El nombre ya existe | El identificador devuelto es -101 | True |
| assertTrue | Se ha actualizado correctamente el curso | El identificador devuelto es mayor que 0 | True |

## MOSTRAR UN CURSO – SA

Primero se muestra un curso ya creado y dado de baja, después se comprobará que está dado de baja viendo si está activo y por último si el id del curso existe.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertEquals | Comprueba si el curso no está activo | El identificador devuelto es false | True |
| assertTrue | Se ha mostrado un curso con un Id existente | El identificador devuelto es mayor que 0 | True |

## MOSTRAR TODOS LOS CURSOS– SA

Se listan todos los cursos existentes y para cada uno se comprueba si su Id existe.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|

| | | | |
|---|---|---|---|
| assertTrue | Comprueba para cada curso si el Id existe | El identificador devuelto es mayor que 0 | True |

### MOSTRAR TODOS LOS TRABAJADORES DE UN CURSO – SA

Se listan todos los trabajadores de un curso existente y para cada uno se comprueba si su Id existe.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertTrue | Comprueba para cada trabajador si el Id existe | El identificador devuelto es mayor que 0 | True |

## EMPRESAS

### ALTA DE UN EMPRESA – SA

Partiendo de una base de datos vacía creo diferentes empresas distintas, cada uno con un error distinto (email, cif o telf incorrecto), una empresa creada correctamente y otra repetida. A continuación, se ejecutan también dar de alta una empresa desactivada y una con cif repetido, probando así la reactivación y el error de dar de alta una activa. Por último se introducen dos empresas más para probar altas correctas.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertTrue | Correo incorrecto | El id devuelto es -71 | true |
| assertTrue | CIF Incorrecto | El id devuelto es -72 | true |
| assertTrue | Tel incorrecto | El id devuelto es -73 | true |
| assertTrue | ID correcta | La id es 1 porque no había otras. | true |
| assertTrue | CIF REPETIDO | La id devuelta es -75 | true |
| assertTrue | Id correcta | Antes he dado de baja una empresa, la 1, ahora, la vuelto a dar de alta con todo igual excepto el campo activo, ahora a true. | true |

| | | | |
|---|---|---|---|
| assertTrue | ID 2, ID 3 al introducir dos nuevas empresas | Doy de alta empresas sin problemas | true |

## BAJA DE UN EMPRESA – SA

Se darán de baja todos los posibles casos de una empresa, entre ellos se encuentran empresas con estado activo a true, empresas con estado activo a false, empresas con estado activo a true y pendientes de curso y por último empresas que no existen y dadas ya de baja. Para ello dejo dado de alta un curso, suponiendo que funciona, y una empresa.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertTrue | Empresa que no existe | Doy de baja la empresa 999, espero -74 | true |
| assertTrue | Empresa activa y existe | Doy de alta la empresa 1 y la doy de baja, espero la id de la empresa 1. | true |
| assertTrue | Doy de baja una empresa ya desactivada | Con la empresa 1, que ya está dada de baja, la intento volver a dar de baja, espero el id -76 | true |
| assertTrue | Doy de baja una empresa activa y con un curso pendiente. | Doy de baja la empresa dos, que está dada de alta ya activa y tiene un curso pendiente, el id curso id 1, espero id -77 | true |

## ACTUALIZAR UNA EMPRESA – SA

Para actualizar una empresa, primero se hará una actualización correcta de la empresa con id 1. Después se intentará actualizar una empresa con valores no válidos de correo, cif y teléfono. Se seguirá con un intento de actualización de empresa que no existe y una empresa con cif repetido. Por último se intentará hacer una actualización de una empresa que tiene un curso asociado, poniendo el campo activo a false.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertTrue | Actualizar la empresa 1 | El identificador devuelto es 1 | True |

| | | | |
|---|---|---|---|
| | con valores válidos | | |
| assertTrue | Correo no válido | El identificador devuelto es -71 | True |
| assertTrue | Cif no válido | El identificador devuelto es -72 | True |
| assertTrue | Teléfono no válido | El identificador devuelto es -73 | True |
| assertTrue | Id no existe | El identificador devuelto es -74 | True |
| assertTrue | Cif repetido | El identificador devuelto es -75 | True |
| assertTrue | “Baja” curso asociado | El identificador devuelto es -77 | True |

### MOSTRAR UNA EMPRESA – SA

Se mostrará una empresa que no exista y los campos de una empresa, en concreto, la 1.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertTrue | Busco la empresa 9999 | La empresa devuelta es nula | true |
| assertTrue | Busco la empresa 1, y la compruebo con esa misma empresa en local, campo a campo | Los campos de la empresa traída son los campos de la empresa en local | true |

### MOSTRAR TODAS LAS EMPRESAS – SA

Para mostrar todos las empresas se traen todas las empresas de base de datos y se comprueba que son 3.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertEquals | Se muestran todas las empresas | El tamaño de la lista de empresas es 3 | True |

### MOSTRAR TODOS LOS CURSOS DE UNA EMPRESA – SA

Se cargarán unos cursos a una empresa 1 y se comprobará que el size sea igual, además también se comprueba que una lista es nula si la empresa no tiene ningún curso.

| Método | Prueba | Comprobación | Resultado Esperado |
|---|---|---|---|
| assertTrue | una empresa que no tiene cursos | La empresa 3 no tiene cursos, espero una lista .isEmpty() = true | True |
| assertTrue | Se añaden dos cursos a la empresa dos, y se comprueba si el tamaño de la lista de vuelta es dos. | Cursos2 su tamaño es 2. | True |

# REPOSITORIOS

| REPOSITORIO | URL |
|---|---|
| **Documentación** | https://versiones.fdi.ucm.es/svn/MS/2021E/mscarillas/doc |
| **Modelo** | https://versiones.fdi.ucm.es/svn/MS/2021E/mscarillas/mod |
| **Código** | https://versiones.fdi.ucm.es/svn/MS/2021E/mscarillas/cod |