Test métodos de clases

A continuación mostramos la documentación de los drivers.

Producto:

Se realizan pruebas sobre la clase Producto, validando su funcionalidad principal de creación y gestión de atributos. Todas las pruebas realizadas han pasado correctamente. Se integran las clases Inventario, Relación y Producto. Estas clases ya han sido desarrolladas previamente y se utilizan para validar las funcionalidades del driver.

createProducto()

- **Objeto de la prueba:** validar la creación de productos con un nombre, tipo y atributos asociados.
- **Fichero de datos necesarios:** No es necesario ningún fichero, ya que los datos se introducen manualmente.
- **Valores estudiados:** Se utiliza la estrategia de caja gris. Se comprueba que no se pueda crear un producto con un nombre que ya exista en el inventario. Y se verifica que los atributos se almacenen correctamente.
- **Operativa:** El sistema valida que todos los datos introducidos sean correctos: nombre, tipo y atributos. Y si no existen errores, el producto se añade al inventario y se generan automáticamente relaciones con los productos existentes

updateRelaciones()

- **Objeto de la prueba:** Test de la función modifyAttributes de la clase Producto
- **Fichero de datos necesarios:** No es necesario ningún fichero, ya que los datos se introducen manualmente.
- **Valores estudiados:** Se utiliza la estrategia de caja gris. Se valida que ambos productos existen en el inventario.Se comprueba que el valor de similitud introducido esté en el rango válido [0, 1].
- **Operativa:** Se eligen dos productos del inventario.Se introduce un valor de similitud (entre 0 y 1), que se valida antes de actualizar la relación. Finalmente, se comprueba que la relación ha sido registrada correctamente.

removeProducto()

- **Objeto de la prueba:** Test de la función deleteProducto de la clase Producto
- **Fichero de datos necesarios:** No es necesario ningún fichero, ya que los datos se introducen manualmente.
- **Valores estudiados:** Se utiliza la estrategia de caja gris. Se comprueba que el producto a eliminar exista en el inventario antes de proceder.Se valida que tras la eliminación, el producto ya no está presente y sus relaciones asociadas
- **Operativa:** Se selecciona un producto existente en el inventario.Se ejecuta la función de eliminación.Posteriormente, se verifica que el producto ha sido eliminado correctamente y no queda ninguna referencia a él en las relaciones.

RestriccionDriver:

La clase RestrictionDriver permite gestionar restricciones en las distribuciones activas de productos en una estantería. Esto incluye agregar, eliminar, modificar y aplicar restricciones, vinculándolas a productos y posiciones específicas.Se integran las clases Inventario, Relación y Producto. Estas clases ya han sido desarrolladas previamente y se utilizan para validar las funcionalidades del driver.

addRestriction()

- **Objeto de la prueba:** Probar la funcionalidad de agregar una nueva restricción en una distribución activa, asociando un producto a una posición específica.
- **Fichero de datos necesarios:** No es necesario ningún fichero, ya que los datos se introducen manualmente.
- **Valores estudiados: :** Se utiliza la estrategia de caja gris. Comprobar que el producto existe en el inventario.Validar que la posición especificada es válida dentro del rango permitido por el tamaño del inventario. También se utiliza la estrategia de la caja negra.Confirmar que la restricción se añade correctamente y se asocia al producto.
- **Operativa:** Introducir el nombre del producto y la posición deseada.Validar que la restricción se registra correctamente en la distribución activa.

applyRestriction()

- **Objeto de la prueba:** Validar la aplicación de todas las restricciones configuradas en la distribución activa.
- **Fichero de datos necesarios:** No es necesario ningún fichero, ya que los datos se introducen manualmente.
- **Valores estudiados:** Se utiliza la estrategia de caja negra. Confirmar que las restricciones configuradas se aplican correctamente en la distribución activa.
- **Operativa:** Ejecutar la función de aplicación de restricciones en una distribución activa.erificar que el sistema confirma la operación y que las restricciones se han aplicado sin errores.

eliminateRestriction()

- **Objeto de la prueba:**Probar la eliminación de una restricción asociada a un producto en una distribución activa.
- **Fichero de datos necesarios:** No es necesario ningún fichero, ya que los datos se introducen manualmente.
- **Valores estudiados:** Se utiliza la estrategia de caja gris. Comprobar que el producto existe en el inventario.Validar que la restricción asociada al producto se elimina correctamente.
- **Operativa:** Introducir el nombre del producto cuya restricción se desea eliminar.Confirmar que el sistema elimina la restricción y que el producto queda sin restricciones en la distribución activa.

modifyRestriction()

- **Objeto de la prueba:** Validar la modificación de una restricción existente asociada a un producto, cambiando la posición asignada.

- **Fichero de datos necesarios:** No es necesario ningún fichero, ya que los datos se introducen manualmente.
- **Valores estudiados:** Se utiliza la estrategia de caja gris. Comprobar que el producto existe en el inventario.Verificar que la posición nueva es válida y que se actualiza correctamente en la distribución.También se utiliza la estrategia de la caja negra.Confirmar que la modificación reemplaza la restricción antigua con la nueva.
- **Operativa:** Introducir el nombre del producto y la nueva posición deseada.Confirmar que el sistema actualiza la restricción correctamente en la distribución activa.

LoadInitialDataDriver:

La clase LoadInitialDataDriver se utiliza para cargar datos iniciales en el inventario y las relaciones entre los productos. Los datos incluyen productos con atributos específicos y relaciones de similitud entre ellos. Se integran las clases Inventario, Relación y Producto. Estas clases ya han sido desarrolladas previamente y se utilizan para validar las funcionalidades del driver.

Execute()

- **Objeto de la prueba:** Verificar que la función cargue correctamente los productos iniciales y sus relaciones de similitud en el inventario y el sistema de relaciones
- **Fichero de datos necesarios:**No se necesita ningún fichero externo, ya que los datos iniciales están definidos de forma interna dentro de la función.
- **Valores estudiados:** Se utiliza la estrategia de caja gris. Verificar que cada producto definido en el código se agrega correctamente al inventario.Comprobar que las relaciones entre productos se almacenan con los valores de similitud indicados. Se utiliza la estrategia de caja negra.Confirmar que, tras la ejecución, el inventario contiene todos los productos previstos.Verificar que las relaciones establecidas coinciden con los valores definidos en el código.
- **Operativa:**
    - Se crean seis productos con atributos comunes y se agregan al inventario.
    - Se establecen relaciones de similitud entre los productos, especificando un valor entre 0 y 1 para cada par.
    - Tras la ejecución, se valida que:
        - Todos los productos están presentes en el inventario.
        - Las relaciones entre productos reflejan los valores definidos en la función.

Estanteria:

La clase EstanteriaDriver permite realizar pruebas relacionadas con la gestión de estanterías y la visualización de distribuciones activas e inactivas en un sistema. Esto incluye agregar o eliminar estantes y listar distribuciones almacenadas en la clase Estanteria.Otros elementos integrados a la prueba son la clase Estanteria y la clase Distribución. Estas clases ya han sido desarrolladas previamente y se utilizan para validar las funcionalidades del driver.

AddEstante()

- **Objeto de la prueba:** tValidar la adición de un número positivo de estantes al sistema.
- **Fichero de datos necesarios:** Datos introducidos manualmente no hace falta ningún fichero.
- **Valores estudiados:** Se utiliza la estrategia de caja gris. Comprobar que el número de estantes agregado es mayor a cero.Verificar que la cantidad total de estantes se actualiza correctamente después de la operación.
- **Operativa:** Introducir el número de estantes a añadir.Validar que la operación se realiza solo si el número es positivo.

removeEstante()

- **Objeto de la prueba:** Probar la eliminación de un número determinado de estantes del sistema.
- **Fichero de datos necesarios:** Datos introducidos manualmente no hace falta ningún fichero.
- **Valores estudiados:** Se utiliza la estrategia de caja gris. Validar que la cantidad total de estantes se actualiza correctamente tras la eliminación. Y también estrategia de caja negra. Asegurarse de que la operación no permite eliminar más estantes de los que existen.
- **Operativa:** Introducir el número de estantes a eliminar.Confirmar que el sistema actualiza correctamente el número total de estantes.

printAllDistributions()

- **Objeto de la prueba:** Validar la visualización de las distribuciones activas e inactivas en el sistema.
- **Fichero de datos necesarios:** No se necesita ningún fichero, los datos se generan dinámicamente en el sistema.
- **Valores estudiados:** Se utiliza la estrategia de caja negra. Comprobar que las distribuciones activas e inactivas se muestran correctamente con todos sus datos.Validar que, si no hay distribuciones activas, el sistema lo indica explícitamente.
- **Operativa:** Llamar a la función de impresión.Verificar que las distribuciones activas e inactivas se muestran en el formato correcto.

Distribucion:

La clase DistributionDriver sirve como controlador para realizar pruebas relacionadas con la creación, activación, eliminación, visualización y modificación de distribuciones de productos en una estantería. Estas distribuciones son calculadas mediante algoritmos seleccionables y basadas en relaciones y atributos de los productos.
Se integran las clases Inventario, Relación y Producto. Además de los dos tipos de Algoritmos (Fuerza Bruta y Aproximacion). Estas clases ya han sido desarrolladas previamente y se utilizan para validar las funcionalidades del driver.

createDistribution()

- **Objeto de la prueba:** Validar la creación de una nueva distribución en la estantería con un identificador único y un algoritmo seleccionado.
- **Fichero de datos necesarios:** No es necesario ningún fichero, ya que los datos se introducen manualmente.
- **Valores estudiados:** Se utiliza la estrategia de caja gris. Verificar que no exista una distribución con el mismo identificador (id) .Probar que la selección de algoritmos (aproximación o fuerza bruta) afecta correctamente la generación de distribuciones.
- **Operativa:** Introducir un identificador (id) único. Seleccionar el algoritmo deseado mediante un booleano (option).Validar que la distribución se calcula y se almacena correctamente en la estantería

deleteDistribution()

- **Objeto de la prueba:** Probar la eliminación de una distribución almacenada en la estantería.
- **Fichero de datos necesarios:** No es necesario ningún fichero, ya que los datos se introducen manualmente..
- **Valores estudiados:** Se utiliza la estrategia de caja gris. Validar que el identificador (id) existe antes de proceder a la eliminación.Comprobar que después de la eliminación, la distribución ya no está disponible.
- **Operativa:** Introducir un identificador (id) existente.Ejecutar la función de eliminación.Confirmar que el sistema notifica el resultado de la operación y que la distribución ya no está presente.

activateDistribution()

- **Objeto de la prueba:** Probar la activación de una distribución almacenada en la estantería.
- **Fichero de datos necesarios:** No es necesario ningún fichero, ya que los datos se introducen manualmente.
- **Valores estudiados:** Se utiliza la estrategia de caja gris. Verificar que una distribución inactiva pasa a estar activa tras la operación.
- **Operativa:** Introducir el identificador (`id`) de una distribución inactiva.Activar la distribución mediante la función activateDistribution.

displayDistribution()

- **Objeto de la prueba:** Validar la correcta visualización de los datos de una distribución existente.
- **Fichero de datos necesarios:** No es necesario ningún fichero, ya que los datos se introducen manualmente.
- **Valores estudiados:** Se utiliza la estrategia de caja gris. Verificar que los datos mostrados corresponden a la distribución seleccionada.
- **Operativa:** Introducir el identificador (id) de una distribución existente.Llamar a la función displayDistribution y validar que la información devuelta coincide con los datos almacenados.

modificarDistribucion()

- **Objeto de la prueba:** Validar la modificación de una distribución activa mediante el intercambio de dos productos.
- **Fichero de datos necesarios:** No es necesario ningún fichero, ya que los datos se introducen manualmente.
- **Valores estudiados:** Se utiliza la estrategia de caja gris. Verificar que ambos productos existen en el inventario antes de proceder.Comprobar que tras la modificación, los datos de la distribución activa reflejan el cambio
- **Operativa:** Verificar que existe una distribución activa.Seleccionar dos productos por nombre (product1Name y product2Name).Ejecutar la función de modificación y confirmar que los productos han sido intercambiados correctamente.

JUEGOS DE PRUEBA

AlgoritmoFuerzaBruta:

De la clase AlgoritmoFuerzaBruta se realizan *n* tests y todos estos han pasado correctamente.

testEjecutarAlgoritmoConMultiplesProductos()

- **Objeto de la prueba:** Test de la ejecutarAlgoritmo de la clase AlgoritmoFuerzaBruta
- **Fichero de datos necesarios:** No es necesario ningún fichero, ya que los datos se introducen manualmente.
- **Valores estudiados:** Se utiliza la estrategia de caja gris. Se crean 5 productos con sus respectivas relaciones
- **Operativa:** En este caso se realiza un juego de pruebas de 5 productos con sus relaciones y se comprueba que la solución que nos retorna sea correcta.
- Comprueba también el comportamiento de las funciones privadas auxiliares para realizar “ejecutarAlgoritmo”

testEjecutarAlgoritmoConProductosMultiples()

- **Objeto de la prueba:** Test de la ejecutarAlgoritmo de la clase AlgoritmoFuerzaBruta
- **Fichero de datos necesarios:** No es necesario ningún fichero, ya que los datos se introducen manualmente.
- **Valores estudiados:** Se utiliza la estrategia de caja gris. Se utiliza el ejemplo de TSP del datasheet.
- **Operativa:** En este caso se realiza un juego de pruebas de 15 productos, los cuales son los caminos, con sus relaciones, las cuales son sus distancias en función de relación (de 0-1) y se comprueba que la solución que nos retorna sea correcta.
- Comprueba también el comportamiento de las funciones privadas auxiliares para realizar “ejecutarAlgoritmo”

testOptimizarSolucionConVecinos()

- **Objeto de la prueba:** Test de la ejecutarAlgoritmo de la clase AlgoritmoFuerzaBruta
- **Fichero de datos necesarios:** No es necesario ningún fichero, ya que los datos se introducen manualmente.
- **Valores estudiados:** Se utiliza la estrategia de caja gris. Se crean 10 productos con sus respectivas relaciones
- **Operativa:** En este caso se realiza un juego de pruebas de 10 productos con sus relaciones y se comprueba que la solución que nos retorna sea correcta.
- Comprueba también el comportamiento de las funciones privadas auxiliares para realizar “ejecutarAlgoritmo”

AlgoritmoAproximado:

De la clase AlgoritmoAproximado se realizan *n* tests y todos estos han pasado correctamente..

testEjecutarAlgoritmoConCicloCompleto()

- **Objeto de la prueba:** Test de la ejecutarAlgoritmo de la clase AlgoritmoAproximado
- **Fichero de datos necesarios:** No es necesario ningún fichero, ya que los datos se introducen manualmente.
- **Valores estudiados:** Se utiliza la estrategia de caja gris. Se crean 5 productos con sus respectivas relaciones
- **Operativa:** En este caso se realiza un juego de pruebas de 5 productos con sus relaciones y se comprueba que la solución que nos retorna sea correcta.
- Comprueba también el comportamiento de las funciones privadas auxiliares para realizar “ejecutarAlgoritmo”

testEjecutarAlgoritmoConProductosMultiples()

- **Objeto de la prueba:** Test de la ejecutarAlgoritmo de la clase AlgoritmoAproximado
- **Fichero de datos necesarios:** No es necesario ningún fichero, ya que los datos se introducen manualmente.
- **Valores estudiados:** Se utiliza la estrategia de caja gris. Se utiliza el ejemplo de TSP del datasheet.
- **Operativa:** En este caso se realiza un juego de pruebas de 15 productos, los cuales son los caminos, con sus relaciones, las cuales son sus distancias en función de relación (de 0-1) y se comprueba que la solución que nos retorna sea correcta.
- Comprueba también el comportamiento de las funciones privadas auxiliares para realizar “ejecutarAlgoritmo”
-

testOptimizarSolucionConVecinos()

- **Objeto de la prueba:** Test de la ejecutarAlgoritmo de la clase AlgoritmoAproximado
- **Fichero de datos necesarios:** No es necesario ningún fichero, ya que los datos se introducen manualmente.
- **Valores estudiados:** Se utiliza la estrategia de caja gris. Se crean 10 productos con sus respectivas relaciones
- **Operativa:** En este caso se realiza un juego de pruebas de 10 productos con sus relaciones y se comprueba que la solución que nos retorna sea correcta.
- Comprueba también el comportamiento de las funciones privadas auxiliares para realizar “ejecutarAlgoritmo”