SuperMarket Product Distribution

Max Estrade* Victor Diez† Victor Llorens‡ Sergio Shmyhelskyy Yaskevych§

Versió d'Entrega : v2

Identificador d'Equip: 23.2

*max.estrade@estudiantat.upc.edu
†victor.diez@estudiantat.upc.edu
‡victor.llorens@estudiantat.upc.edu
§sergio.shmyhelskyy@estudiantat.upc.edu

Contents

1 Introducció

Aquest document proporciona una descripció detallada de les classes implementades en el projecte "SuperMarket Product Distribution". L'objectiu principal és oferir una visió clara de les funcionalitats i característiques de cada classe, així com de les estructures de dades utilitzades. A més, es descriuen els mètodes i operacions disponibles per a la gestió de productes, usuaris i distribucions dins del sistema.

El document està estructurat de la següent manera:

- Descripció de les Classes Implementades: Una descripció detallada de cada classe, incloent-hi els seus mètodes i funcionalitats.
- Persistència de Dades: Informació sobre com es gestionen i emmagatzemen les dades de persistència del sistema.
- Estructura de Dades per Classes: Descripció de les estructures de dades utilitzades per a la gestió de productes, llistes de productes i distribucions.

2 Descripció de les Classes Implementades

A continuació es descriuen les classes implementades per cada membre de l'equip, així com les seves característiques i funcionalitats.

2.1 Capa de Domini

2.1.1 Classe UserManager

La classe `UserManager` és un singleton que s'encarrega de gestionar les operacions dels usuaris, com ara el registre, l'inici de sessió i la gestió de dades. Proporciona mètodes per afegir, eliminar i consultar usuaris, així com per gestionar les llistes de productes i distribucions dels usuaris actius.

- **registerUser**
  - **Descripció:** Registra un nou usuari al sistema.
  - **Paràmetres:**
    - `username` - el nom d'usuari
    - `email` - el correu electrònic
    - `password` - la contrasenya
  - **Llança:** `UserException` si el nom d'usuari o el correu electrònic ja estan en ús
- **loginUser**
  - **Descripció:** Inicia sessió amb un usuari existent.
  - **Paràmetres:**
    - `userInput` - el nom d'usuari o el correu electrònic
    - `password` - la contrasenya
  - **Llança:** `UnauthorizedAccessException` si el nom d'usuari/correu electrònic o la contrasenya són incorrectes
- **logoutUser**
  - **Descripció:** Tanca la sessió de l'usuari actiu.
  - **Retorna:** cert si la sessió s'ha tancat correctament, fals en cas contrari
- **deleteUser**
  - **Descripció:** Elimina un usuari pel seu nom d'usuari.
  - **Paràmetres:**
    - `username` - el nom d'usuari a eliminar
  - **Retorna:** cert si l'usuari s'ha eliminat correctament, fals en cas contrari

2.1.2 Classe UserSet

La classe `UserSet` representa un conjunt d'usuaris i proporciona mètodes per gestionar-los. Permet afegir, eliminar i consultar usuaris, així com comprovar l'existència d'usuaris pel seu ID, nom d'usuari o correu electrònic.

- **addUser**
  - **Descripció:** Afegeix un usuari al conjunt.
  - **Paràmetres:**
    - `user` - l'usuari a afegir
- **removeUser**
  - **Descripció:** Elimina un usuari del conjunt pel seu ID.
  - **Paràmetres:**
    - `userID` - l'ID de l'usuari a eliminar
- **getUser**
  - **Descripció:** Obté un usuari pel seu ID.
  - **Paràmetres:**
    - `userID` - l'ID de l'usuari a obtenir
  - **Retorna:** l'usuari amb l'ID especificat, o null si no es troba

2.1.3 Classe User

La classe `User` representa un usuari del sistema. Gestiona els atributs de l'usuari, com ara el seu ID, nom d'usuari, contrasenya i correu electrònic.

- **getUserID**
  - **Descripció:** Obté l'ID de l'usuari.
  - **Retorna:** l'ID de l'usuari
- **getUsername**
  - **Descripció:** Obté el nom d'usuari.
  - **Retorna:** el nom d'usuari
- **setUsername**
  - **Descripció:** Estableix el nom d'usuari.
  - **Paràmetres:**
    - `username` - el nou nom d'usuari

2.1.4 Classe CurrentUser

La classe `CurrentUser` representa l'usuari actualment actiu en el sistema. Gestiona els atributs de l'usuari, així com les seves llistes de productes i distribucions.

- **addProductList**
  - **Descripció:** Afegeix una llista de productes a l'usuari.
  - **Paràmetres:**
    - `productListName` - el nom de la llista de productes a afegir
- **removeProductList**
  - **Descripció:** Elimina una llista de productes de l'usuari pel seu nom.
  - **Paràmetres:**
    - `productListName` - el nom de la llista de productes a eliminar
- **getUserProductLists**
  - **Descripció:** Obté totes les llistes de productes de l'usuari.
  - **Retorna:** un conjunt de totes les llistes de productes

2.1.5 Classe Shelf

La classe `Shelf` és una classe que representa una prestatgeria del sistema. Conté una matriu de productes que representa la distribució de productes en la prestatgeria.

Té com atributs id (que és la seva clau primaria), xsize(representa el tamany horizontal de la prestatgeria), ysize (vertical), distributionHistory(un historial de les ultime distribusions que s'han fet) i té una relació amb les classes DIstribution. Quan es crea una instància de `Shelf`, aquesta es crea buida i es pot anar omplint amb distribucions de productes (que es poden afegir, eliminar, consultar, etc.), que inicialment serà generada per un algorisme d'ordenació de productes (`BruteForceAlgorithm` o `HillClimbingAlgorithm`). És a dir, té una llista de productes associada i una llista de distribucions, on cada distribució és una matriu de productes. La distribució actual de la prestatgeria (l'última de la llista de distribucions) es pot consultar en qualsevol moment, i es pot canviar per una altra distribució.

2.1.6 Classe Similarity

La classe `Similarity` és una classe associativa que representa el grau de relació i similitud entre dos productes del sistema, de forma que un producte que tingui major similitud amb un altre producte es pugui després tenir en compte per distribuir-los en una prestatgeria. És la classe més granular del sistema, ja que representa la relació entre dos productes.

2.1.7 Classe Product

La classe `Product` representa un producte del sistema. Cada producte té un nom, una categoria, un preu (es guarda en tot cas el seu preu original) i una quantitat. Inclou característiques addicionals com la validació, el seguiment de descomptes i els mètodes d'utilitat.

2.1.8 Classe ProductList

La classe `ProductList` representa una llista de productes del sistema. Cada llista de productes té un nom i una llista de productes associats (usant un diccionari `Set<>`). Aquesta classe permet afegir productes a la llista, eliminar-los, consultar-los, etc. Per dur a terme aquestes operacions, depèn de la classe `Product`. Cada vegada que es modifica la llista de productes, es guarda un registre de la modificació amb la data i hora en què s'ha fet en el ControllerDomain.

2.1.9 Classe ProductManager

La classe `ProductManager` és un singleton que s'encarrega de gestionar les instàncies de `Product` del sistema i les seves relacions amb les llistes de productes `ProductList` . S'aplica aquest patró de disseny, de forma que els controladors del domini puguin gestionar correctament les instàncies en el sistema. Gestiona tota mena d'operacions relacionades amb afegir productes, eliminar-los o consultar-los, juntament amb les múltiples llistes de productes, proporcionant mètodes per afegir, eliminar i editar productes. S'encarrega de mantenir un mapa de similituts* entre tots els productes del catàleg i les llistes de productes. Per gestionar els productes, fa servir diccionaris per guardar els productes i les llistes amb els seu nom com a clau per poder buscar-los ràpidament de manera eficient.

- **addProductToCatalog**
  - **Descripció:** Afegeix un producte únic al catàleg de productes del supermercat.
  - **Paràmetres:**
    - `productName` - el nom del producte
    - `category` - la categoria del producte
    - `price` - el preu del producte
    - `amount` - la quantitat del producte
    - `similarities` - una llista de noms de productes i valors de similitud
  - **Llança:** `ProductException` si el producte ja existeix al catàleg
- **removeProductFromCatalog**
  - **Descripció:** Elimina un producte del catàleg.
  - **Paràmetres:**
    - `productName` - el nom del producte a eliminar
  - **Llança:** `ProductException` si el producte no existeix al catàleg
- **increaseProductQuantity**
  - **Descripció:** Augmenta la quantitat d'un producte al catàleg.
  - **Paràmetres:**
    - `productName` - el nom del producte
    - `amount` - la quantitat a augmentar
  - **Retorna:** la quantitat restant del producte
  - **Llança:** `ProductException` si el producte no existeix o la quantitat no és positiva
- **decreaseProductQuantity**
  - **Descripció:** Disminueix la quantitat d'un producte al catàleg.
  - **Paràmetres:**
    - `productName` - el nom del producte
    - `amount` - la quantitat a disminuir
  - **Retorna:** la quantitat restant del producte

  - **Llança:** `ProductException` si el producte no existeix, la quantitat no és vàlida o la quantitat resultant és negativa

- **applyDiscountToProduct**
  - **Descripció:** Aplica un descompte a un producte i registra la modificació.
  - **Paràmetres:**
    * `productName` - el nom del producte
    * `percentage` - el percentatge de descompte a aplicar
  - **Llança:** `ProductException` si el producte no existeix al catàleg

- **createProductList**
  - **Descripció:** Crea una nova llista de productes al supermercat.
  - **Paràmetres:**
    * `listName` - el nom de la llista de productes
    * `category` - la categoria de la llista de productes
  - **Retorna:** cert si la llista s'ha creat, fals si ja existeix
  - **Llança:** `ProductListException` si la llista de productes ja existeix

- **removeProductList**
  - **Descripció:** Elimina una llista de productes del supermercat.
  - **Paràmetres:**
    * `listName` - el nom de la llista de productes a eliminar
  - **Llança:** `ProductListException` si la llista de productes no existeix

- **addProductToList**
  - **Descripció:** Afegeix un producte existent del catàleg a una llista de productes específica.
  - **Paràmetres:**
    * `listName` - el nom de la llista de productes a la qual s'afegirà el producte
    * `productName` - el nom del producte a afegir
  - **Llança:** `ProductListException` si la llista de productes no existeix
  - **Llança:** `ProductException` si el producte no existeix

- **removeProductFromList**
  - **Descripció:** Elimina un producte d'una llista de productes especificada.
  - **Paràmetres:**
    * `listName` - el nom de la llista de productes de la qual s'eliminarà el producte
    * `productName` - el nom del producte a eliminar de la llista
  - **Llança:** `ProductListException` si la llista de productes no existeix
  - **Llança:** `ProductException` si el producte no existeix a la llista

- **setSimilarity**
  - **Descripció:** Afegeix una similitud entre dos productes.
  - **Paràmetres:**
    * `product1` - el nom del primer producte
    * `product2` - el nom del segon producte
    * `similarity` - el valor de similitud
  - **Llança:** `ProductException` si algun dels productes no existeix

*Nota: Respecte a la primera entrega: S'ha deixat d'utilitzar Similarity Manager.

2.1.10 Classe Distribution

La classe `Distribution` representa una distribució de productes en una prestatgeria del sistema. Guarda la distribució de productes com una matriu que representa el prestatge de la mateixa manera que la imatge. Aquesta classe permet afegir productes a la distribució, eliminar-los, consultar-los, etc. Conté la data de creació i modificació de la distribució, així com la llista de productes associats a la distribució.

2.1.11 Classe DistributionManager

La classe `DistributionManager` és un singleton que s'encarrega de gestionar les instàncies de `Distribution` del sistema. Permet gestionar més fàcilment les distribucions de productes en les prestatgeries del sistema, podent afegir distribucions, eliminar-les, modificar-les, consultar-les, etc. Cada vegada que es modifica una distribució, es guarda un registre de la modificació amb la data i hora en què s'ha fet.

2.1.12 Classe AbstractAlgorithm

La classe `AbstractAlgorithm` és una classe abstracta que conté la funció polimòrfica `orderProductList`. Aquesta funció permet ordenar una llista de productes en una prestatgeria, i retorna la distribució de productes ordenada. Aquesta classe conté també dues funcions comunes a les classes que heretin d'aquesta.

2.1.13 Classe BruteForceAlgorithm

La classe `BruteForceAlgorithm` és una classe que hereta de `AbstractAlgorithm` i implementa la funció `orderProductList`. Implementa un algorisme de força bruta *backtracking* per trobar una distribució de productes ordenada en una prestatgeria.

2.1.14 Classe HillClimbingAlgorithm

La classe `HillClimbingAlgorithm` és una classe que hereta de `AbstractAlgorithm` i implementa la funció `orderProductList`. Implementa un algorisme d'aproximació *Hill Climbing* per trobar una distribució de productes ordenada en una prestatgeria.

2.1.15 Classe InputManager

La classe `InputManager` és un singleton que s'encarrega de gestionar les entrades del sistema. S'aplica aquest patró de disseny, de forma que els controladors del domini puguin gestionar correctament les entrades en el sistema. Gestiona tota mena d'operacions relacionades amb les entrades del sistema, com ara llegir fitxers, escriure fitxers, etc. És usada principalment per la classe `ControllerDomain/DriverControllerDomain` per llegir les dades i comandes d'entrada del sistema.

Permeten l'execució a través d'una sèrie de comandes que es poden llegir des d'un fitxer o des de la consola:

- `CREATE_LIST listName and listCategory`
- `DELETE_LIST listId`
- `ADD_PRODUCT nameProduct and nameList`
- `REMOVE_PRODUCT nameProduct and nameList`
- `SHOW_SIMILARITIES nomProduct1, nomProduct2, and newSimilarity`
- `CHANGE_SIMILARITIES nomProduct1, nomProduct2, and newSimilarity`
- `DECREASE_PRODUCT_QUANTITY nameProduct and quantity`
- `INCREASE_PRODUCT_QUANTITY nameProduct and quantity`
- `ADD_PRODUCT_TO_CATALOG productName, category, price, and amount`
- `REMOVE_PRODUCT_CATALOG productName`
- `SHOW_MODIFICATION_LOG`
- `CREATE_SHELF id, maxCapacity, listName, algorithm`
- `CHANGE_SHELF_ALGORITHM shelfId and algorithm`
- `CHANGE_PRODUCT_LIST_AT_SHELF shelfId and listName`
- `CREATE_NEW_DISTRIBUTION_FILE newName, path, and shelfId`
- `SHOW_PRODUCTS`
- `SHOW_ALL_LISTS`
- `SHOW_DISTRIBUTION idShelf`
- `CREATE_DISTRIBUTION idShelf, nomDistribution and (optional) Depth`
- `MODIFY_DISTRIBUTION nameDistribution, nameProduct1, nameProduct2`
- `SHOW_SHELVES`

2.1.16 Classe ControllerDomain

La classe `ControllerDomain` és un singleton que s'encarrega de gestionar les operacions del sistema. Aquesta és la classe principal que gestiona la capa de domini del sistema, i s'encarrega de gestionar les operacions del sistema cap a les classes del domini i retornar les sortides del sistema cap a la capa de presentació (`Controlador de Presentació`). S'aplica aquest patró de disseny, de forma que els controladors del domini puguin gestionar correctament les operacions en el sistema. S'encarrega d'instanciar la resta de controladors, i proporciona una interfície més pròxima a l'usuari dels mètodes de la resta de controladors.

També es comunica amb la classe `InputManager` per llegir les dades i comandes d'entrada del sistema. Gestiona tota mena d'operacions relacionades amb les operacions del sistema, com ara afegir productes, eliminar-los, consultar-los, etc. És usada principalment per la classe `DriverControllerDomain` per gestionar les operacions del sistema.

A més, el `ControllerDomain` també s'encarrega de la gestió dels usuaris i de l'usuari actual del sistema.

- **registerUser**
  - **Descripció:** Registra un nou usuari al sistema.
  - **Paràmetres:**
    - `username` - el nom d'usuari
    - `email` - el correu electrònic
    - `password` - la contrasenya
  - **Llança:** `UserException` si el nom d'usuari o el correu electrònic ja estan en ús
- **loginUser**
  - **Descripció:** Inicia sessió amb un usuari existent.
  - **Paràmetres:**
    - `userInput` - el nom d'usuari o el correu electrònic
    - `password` - la contrasenya
  - **Llança:** `UnauthorizedAccessException` si el nom d'usuari/correu electrònic o la contrasenya són incorrectes
- **logoutUser**
  - **Descripció:** Tanca la sessió de l'usuari actiu.
  - **Retorna:** cert si la sessió s'ha tancat correctament, fals en cas contrari
- **deleteUser**
  - **Descripció:** Elimina un usuari pel seu nom d'usuari.
  - **Paràmetres:**
    - `username` - el nom d'usuari a eliminar
  - **Retorna:** cert si l'usuari s'ha eliminat correctament, fals en cas contrari
- **addProductToCatalog**
  - **Descripció:** Afegeix un producte únic al catàleg de productes del supermercat.
  - **Paràmetres:**
    - `productName` - el nom del producte
    - `category` - la categoria del producte
    - `price` - el preu del producte
    - `amount` - la quantitat del producte
    - `similarities` - una llista de noms de productes i valors de similitud
  - **Llança:** `ProductException` si el producte ja existeix al catàleg
- **removeProductFromCatalog**
  - **Descripció:** Elimina un producte del catàleg.

  - **Paràmetres:**
    - `productName` - el nom del producte a eliminar
  - **Llança:** `ProductException` si hi ha un error en eliminar el producte del catàleg

- **updateProduct**
  - **Descripció:** Actualitza un producte en el catàleg.
  - **Paràmetres:**
    - `name` - el nom del producte
    - `category` - la categoria del producte
    - `price` - el preu del producte
    - `amount` - la quantitat del producte
    - `similarities` - una llista de parells amb el nom del producte i el valor de similitud
  - **Llança:** `ProductException` si el nom del producte és invàlid

- **createProductList**
  - **Descripció:** Crea una nova llista de productes amb una categoria especificada.
  - **Paràmetres:**
    - `listName` - el nom de la llista de productes
    - `category` - la categoria dels productes en la llista
  - **Llança:** `ProductListException` si el nom de la llista és invàlid

- **addProductToList**
  - **Descripció:** Afegeix un producte a una llista especificada.
  - **Paràmetres:**
    - `productName` - el nom del producte a afegir a la llista
    - `listName` - el nom de la llista a la qual s'afegirà el producte
  - **Llança:** `ProductException` si hi ha un error en afegir el producte a la llista
  - **Llança:** `ProductListException` si hi ha un problema amb la llista de productes especificada

- **removeProductFromList**
  - **Descripció:** Elimina un producte d'una llista específica.
  - **Paràmetres:**
    - `productName` - el nom del producte
    - `listName` - el nom de la llista de productes
  - **Llança:** `ProductException` si el nom del producte és invàlid
  - **Llança:** `ProductListException` si el nom de la llista és invàlid

- **decreaseProductQuantity**
  - **Descripció:** Disminueix la quantitat d'un producte especificat en el catàleg.
  - **Paràmetres:**
    - `productName` - el nom del producte la quantitat del qual es disminuirà
    - `quantity` - la quantitat en què es reduirà la quantitat del producte
  - **Llança:** `ProductException` si hi ha un error en disminuir la quantitat del producte

- **increaseProductQuantity**
  - **Descripció:** Augmenta la quantitat d'un producte especificat en el catàleg.
  - **Paràmetres:**
    - `productName` - el nom del producte la quantitat del qual s'augmentarà
    - `quantity` - la quantitat en què s'augmentarà la quantitat del producte
  - **Llança:** `ProductException` si hi ha un error en augmentar la quantitat del producte

- **applyDiscountToList**
  - **Descripció:** Aplica un descompte a tots els productes d'una llista de productes especificada.
  - **Paràmetres:**
    - `listName` - el nom de la llista de productes a la qual s'aplicarà el descompte
    - `percentage` - el percentatge de descompte a aplicar als productes de la llista
  - **Llança:** `ProductListException` si hi ha un error en aplicar el descompte a la llista de productes
- **createShelf**
  - **Descripció:** Crea una nova prestatgeria amb la capacitat, llista de productes i algorisme especificats.
  - **Paràmetres:**
    - `id` - l'identificador únic de la prestatgeria
    - `xcapacity` - la capacitat en l'eix x (amplada) de la prestatgeria
    - `ycapacity` - la capacitat en l'eix y (alçada) de la prestatgeria
    - `listName` - el nom de la llista de productes a associar amb la prestatgeria
    - `algorithm` - l'algorisme a utilitzar per a l'ordenació de productes (1 per Força Bruta, 2 per Hill Climbing)
  - **Llança:** `ShelfException` si l'identificador de la prestatgeria ja existeix, o si hi ha un algorisme invàlid, o una capacitat de prestatgeria invàlida
  - **Llança:** `ProductListException` si la llista de productes està buida, o si la llista de productes no cap a la prestatgeria a causa de la seva mida
- **changeProductListAtShelf**
  - **Descripció:** Canvia la llista de productes associada a una prestatgeria especificada.
  - **Paràmetres:**
    - `idShelf` - l'identificador de la prestatgeria la llista de productes de la qual es canviarà
    - `listName` - el nom de la nova llista de productes a assignar a la prestatgeria
  - **Llança:** `ShelfException` si la prestatgeria amb l'identificador especificat no existeix
  - **Llança:** `ProductListException` si la nova llista de productes excedeix la capacitat de la prestatgeria
- **changeShelfAlgorithm**
  - **Descripció:** Canvia l'algorisme utilitzat per a l'ordenació de productes en una prestatgeria especificada.
  - **Paràmetres:**
    - `id` - l'identificador de la prestatgeria l'algorisme de la qual es canviarà
    - `algorithm` - l'identificador de l'algorisme a aplicar (1 per Força Bruta, 2 per Hill Climbing)
  - **Llança:** `ShelfException` si la prestatgeria amb l'identificador especificat no existeix o si l'identificador de l'algorisme és invàlid
- **createNewDistribution**
  - **Descripció:** Crea una nova distribució per a una prestatgeria especificada.
  - **Paràmetres:**
    - `idShelf` - l'identificador de la prestatgeria per a la qual es crearà la distribució
    - `nameDistribution` - el nom de la nova distribució
    - `limit` - el límit de profunditat per a la distribució (ha de ser superior a 0)
  - **Llança:** `ShelfException` si la prestatgeria especificada no existeix
  - **Llança:** `DistributionException` si el límit de profunditat és invàlid (és a dir, zero o negatiu)
- **deleteShelf**
  - **Descripció:** Elimina una prestatgeria del mapa de prestatgeries pel seu identificador.
  - **Paràmetres:**
    - `id` - l'identificador únic de la prestatgeria a eliminar
  - **Llança:** `ShelfException` si la prestatgeria amb l'identificador especificat no existeix

2.2 Capa de Persistència

2.2.1 Classe ControllerPersistence

La classe `ControllerPersistence` és un singleton que s'encarrega de gestionar les operacions de persistència del sistema. Proporciona mètodes per carregar i guardar dades de productes, usuaris i distribucions.

- **loadProductCatalog**
  - **Descripció:** Carrega el catàleg de productes des del fitxer de configuració.
  - **Retorna:** el catàleg de productes carregat
- **saveProductCatalog**
  - **Descripció:** Desa el catàleg de productes al fitxer de configuració.
- **loadUserCredentials**
  - **Descripció:** Carrega les credencials dels usuaris des del fitxer de configuració.
  - **Retorna:** un conjunt d'usuaris carregat
- **saveUserCredentials**
  - **Descripció:** Desa les credencials dels usuaris al fitxer de configuració.
- **loadUserDistributions**
  - **Descripció:** Carrega les distribucions d'un usuari des del directori de l'usuari.
  - **Paràmetres:**
    - `username` - el nom d'usuari
  - **Retorna:** un mapa de distribucions carregat
- **saveUserDistributions**
  - **Descripció:** Desa les distribucions d'un usuari al directori de l'usuari.
  - **Paràmetres:**
    - `username` - el nom d'usuari
    - `distributions` - la llista de distribucions a desar

2.2.2 Classe ProductManagerData

La classe `ProductManagerData` és responsable de gestionar les dades del catàleg de productes. Proporciona mètodes per carregar, desar i inicialitzar el catàleg de productes.

- **initConfigFile**
  - **Descripció:** Inicialitza el fitxer de configuració. Crea el fitxer de configuració si no existeix.
- **loadProductCatalog**
  - **Descripció:** Carrega el catàleg de productes des del fitxer.
  - **Retorna:** el catàleg de productes carregat
- **saveProductCatalog**
  - **Descripció:** Desa el catàleg de productes i les similituds al fitxer.
- **getProductCatalog**
  - **Descripció:** Obté el catàleg de productes.
  - **Retorna:** el catàleg de productes
- **getProductSimilarities**
  - **Descripció:** Obté les similituds dels productes.
  - **Retorna:** el mapa de similituds dels productes
- **getProduct**
  - **Descripció:** Recupera un producte del catàleg pel seu nom.
  - **Paràmetres:**
    - `productName` - el nom del producte a recuperar
  - **Retorna:** el producte amb el nom especificat, o null si no es troba

2.2.3 Classe UserManagerData

La classe `UserManagerData` és responsable de gestionar les dades de les credencials dels usuaris. Proporciona mètodes per carregar, desar i verificar les credencials dels usuaris, així com per gestionar les llistes de productes i distribucions específiques dels usuaris.

- **initConfigFile**
  - **Descripció:** Inicialitza el fitxer de configuració. Crea el fitxer de configuració si no existeix.
- **saveUserCredentials**
  - **Descripció:** Desa les credencials dels usuaris al fitxer de configuració.
- **saveUserDistributions**
  - **Descripció:** Desa les distribucions d'un usuari al directori de l'usuari.
  - **Paràmetres:**
    - `username` - el nom d'usuari
    - `distributions` - la llista de distribucions a desar
- **saveUserProductLists**
  - **Descripció:** Desa les llistes de productes d'un usuari al directori de l'usuari.
  - **Paràmetres:**
    - `username` - el nom d'usuari
    - `productLists` - el mapa de llistes de productes a desar
- **loadUserCredentials**
  - **Descripció:** Carrega les credencials dels usuaris des del fitxer de configuració.
  - **Retorna:** un conjunt d'usuaris carregat
- **loadUserProductLists**
  - **Descripció:** Carrega les llistes de productes d'un usuari des del directori especificat.
  - **Paràmetres:**
    - `username` - el nom d'usuari
  - **Retorna:** un mapa de llistes de productes carregat
- **loadUserDistributions**
  - **Descripció:** Carrega les distribucions d'un usuari des del directori especificat.
  - **Paràmetres:**
    - `username` - el nom d'usuari
  - **Retorna:** un mapa de distribucions carregat
- **registerUser**
  - **Descripció:** Registra un nou usuari al sistema.
  - **Paràmetres:**
    - `username` - el nom d'usuari
    - `email` - el correu electrònic
    - `hashedPassword` - la contrasenya encriptada

2.3 Capa de Presentació

La capa de presentació està formada per vistes, gestionades per un únic controlador de presentació que s'encarrega de gestionar les vistes i fer crides al controlador de domini per tenir-ho tot a punt.

2.3.1 Classe PresentationController

La classe `PresentationController` és un singleton que s'encarrega de gestionar la capa de presentació de l'aplicació. Proporciona mètodes per interactuar amb les vistes i delega les operacions al controlador de domini.

Les operacions més destacables són, la de canvi de vistes, les d'omplir les llistes de les vistes, i operacions per operacions del controlador de domini (entre elles, getters i setters) per tal de reduir l'acoblament entre vistes i el controlador de domini (capa de domini) i les operacions que creen una vista auxiliar per mostrar a l'usuari informació addicional o errors.

Les vistes consisteixen en una sèrie de finestres que mostren la informació de l'aplicació a l'usuari, on principalment es mostren llistes de productes, llistes de distribucions, i llistes de prestatgeries. Es criden prement botons, permetent navegar per l'aplicació. A més, permeten a l'usuari interactuar amb l'aplicació mitjançant la introducció de dades i la realització d'operacions.

3 Persistència de Dades

Les dades de persistència es guarden en els següents fitxers i directoris:

- **Catàleg de Productes:**
  - Fitxer: `src/main/resources/catalog/products_catalog.txt`
  - Descripció: Aquest fitxer conté el catàleg de productes i les seves similituds.
  - Format:
    - Cada línia representa un producte o una similitud entre productes.
    - Les línies que representen productes tenen el format: `ProductName,Category,Price,Amount`
    - Les línies que representen similituds tenen el format: `Product1,Product2,SimilarityScore`
- **Credencials d'Usuaris:**
  - Fitxer: `src/main/resources/config/user_credentials.txt`
  - Descripció: Aquest fitxer conté les credencials dels usuaris registrats.
  - Format:
    - Cada línia representa un usuari.
    - El format de cada línia és: `username:email:hashedPassword`
- **Dades d'Usuaris:**
  - Directori: `src/main/resources/users/`
  - Descripció: Aquest directori conté subdirectoris per a cada usuari, on es guarden les seves llistes de productes i distribucions.
    - `productLists/`: Conté les llistes de productes de l'usuari.
      - Format:
      - La primera línia conté el nom de la llista.
      - La segona línia conté la categoria de la llista.
      - La tercera línia conté la data de l'última modificació.
      - Les línies següents contenen els noms dels productes.
    - `distributions/`: Conté les distribucions de productes de l'usuari.

4 Estructura de Dades per Classes

4.1 UserManager

- **UserSet userSet**:
  - Representa el conjunt d'usuaris del sistema.
  - Guardem els usuaris en un conjunt per poder gestionar-los fàcilment.
  - Cost d'accedir a un usuari concret: O(1).
  - Cost d'afegir un usuari: O(1).
  - Cost d'eliminar un usuari: O(1).
  - Cost d'accedir a tots els usuaris: O(n).
- **CurrentUser activeUser**:
  - Representa l'usuari actualment actiu en el sistema.
  - Guardem l'usuari actiu per poder gestionar les seves operacions.
  - Cost d'accedir a l'usuari actiu: O(1).
  - Cost d'actualitzar l'usuari actiu: O(1).

4.2 UserSet

- **Map<Integer, User>**:
  - Representa el conjunt d'usuaris amb el seu ID com a clau (resultant de la funció de hash del username i correu electrònic).
  - Guardem els usuaris amb el seu ID com a clau per poder buscar-los ràpidament.
  - Cost d'accedir a un usuari concret: O(1).
- **Map<String, User>**:
  - Representa el conjunt d'usuaris amb el seu nom d'usuari com a clau.
  - Guardem els usuaris amb el seu nom d'usuari com a clau per poder buscar-los ràpidament.
  - Cost d'accedir a un usuari concret: O(1).
- **Map<String, User>**:
  - Representa el conjunt d'usuaris amb la combinació del seu nom d'usuari i correu electrònic com a clau.
  - Guardem els usuaris amb la combinació del seu nom d'usuari i correu electrònic com a clau per poder buscar-los ràpidament.
  - Cost d'accedir a un usuari concret: O(1).

4.3 ProductManager

- **Map<String, Product>**:
  - Representa el catàleg de productes que tenim a la botiga (registre de productes en sistema).
  - Guardem els productes amb el seu nom com a clau per poder buscar-los ràpidament.
  - Cost d'accedir a un producte concret: O(1).
- **Map<String, ProductList>**:
  - Representa les seccions del catàleg com a llistes de productes que tenim a la botiga.
  - Poden ser llistes buides o amb productes. Només es poden afegir productes que ja existeixen al catàleg.
  - Guardem les llistes amb el seu nom com a clau per poder buscar-les ràpidament.
  - Cost d'accedir a una llista concreta: O(1).
- **Map<String, Map<String, Double>>**:
  - Representa les similituds entre productes del catàleg: per cada producte guardem les similituds amb la resta de productes.
  - Guardem les similituds amb el nom dels dos productes com a clau per poder buscar-les ràpidament.
  - Cost d'accedir a accedir a un producte concret: O(1).
  - Cost d'accedir a una similitud concreta: O(1).

4.4 ProductList

- **Set<Product>**:
  - Representa la llista de productes que tenim a la botiga.
  - No podem tenir productes repetits a la mateixa llista, per això hem escollit un Set.
  - Guardem els productes amb el seu nom com a clau per poder buscar-los ràpidament.
  - Cost d'accedir a un producte concret: O(1).
  - Cost d'afegir un producte: O(1).
  - Cost d'eliminar un producte: O(1).
  - Cost d'actualitzar un producte: O(1).
  - Cost d'accedir a tots els productes: O(n).
  - Cost d'accedir a tots els productes d'una categoria: O(n).

4.5 Distribution

- **ArrayList<ArrayList<Product>>**:
  - Guardem la distribució de productes com una matriu que representa el prestatge.
  - La mida de la matriu és variable, depenent de la mida de la prestatgeria (xsize, ysize).

4.6 ControllerDomain

- **HashMap<int, Shelf>**:
  - HashMap de Prestatgeries on la clau és la id d'una prestatgeria i el contingut és la prestatgeria.
  - Guardem les prestatgeries amb la seva id com a clau per poder buscar-les ràpidament.
  - Cost d'accedir a una prestatgeria concreta: O(1).
  - Cost d'afegir una prestatgeria: O(1).
  - Cost d'eliminar una prestatgeria: O(1).
  - Cost d'actualitzar una prestatgeria: O(1).
  - Cost d'accedir a totes les prestatgeries: O(n).
  - Cost d'accedir a totes les prestatgeries amb un algorisme concret: O(n).
  - Cost d'accedir a totes les prestatgeries amb una llista de productes concreta: O(n).
- **TreeMap<LocalDateTime, String >**:
  - S'ha escollit un TreeMap perquè ens permet tenir les modificacions ordenades per data i hora, segons l'ordre natural de les dates.
  - Guardem un registre de les modificacions que s'han fet al sistema amb la data i hora en què s'han fet.
  - Guardem el nom de la modificació com a clau i la data i hora com a valor.
  - Ens permet tenir un registre de les modificacions que s'han fet al sistema.
  - Cost d'accedir a una modificació concreta: O(log n).
  - Cost d'afegir una modificació: O(log n).
  - Cost d'eliminar una modificació: O(log n).
  - Cost d'actualitzar una modificació: O(log n).
  - Cost d'accedir a totes les modificacions: O(n).

4.7 Algorismes

- **ArrayList<ArrayList<Product>>:**
  - És com s'espera que es retorni la distribució ordenada. És una matriu de productes de mida variable.
  - Encara que fem els algorismes amb altres estructures, vam decidir retornar-ho amb aquesta per poder representar-la millor per pantalla ja que aquesta estructura de dades es similar a la forma que tindria una prestatgeria a la vida real.
  - La distribució esta ordenada com a la imatge.

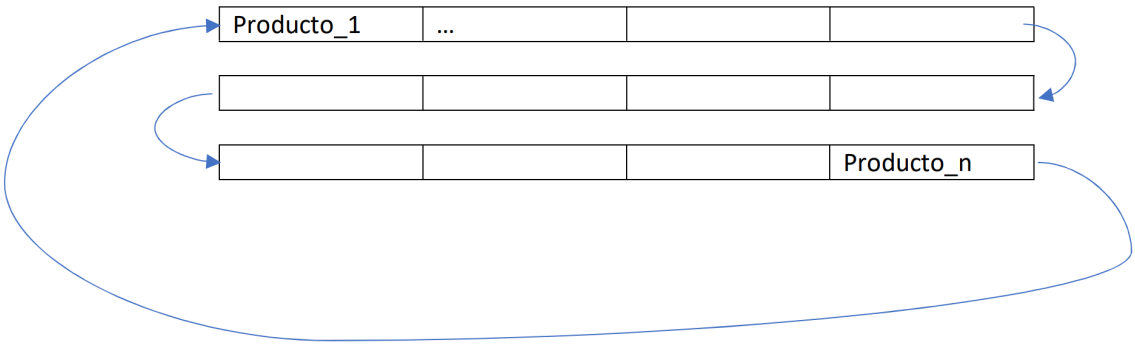

Figure 1: Distribució de Productes en una Prestatgeria

- **Hashmap<String, Pair<Integer, Integer>>:**
  - A aquest tipus de estructura de dades guardem les coordenades dels productes de la distribució. La clau del mapa es el nom del producte i el seu valor es una parella d'enters que representa les coordenades (x,y) del producte a la matriu.
  - Vem decidir afegir aquesta estructura de dades perquè voliem una manera ràpida de trobar els productes a la matriu i un mapa ens permet buscar productes per el seu nom sense tenir que recorrer tota la matriu.
  - Amb aquesta estructura de dades, si volem obtenir la similaritat de 2 productes només hem de fer dos gets, que tenen una complexitat cadascun de O(1). Abans d'aquesta implementació, guardavem la similaritat dels productes com a una llista de similaritats a cada producte amb la relació amb la resta de productes, amb aquesta implementació trobar la similaritat de dos productes tenia una complexitat de O(n).

- **List<Product>:**
  - Fem servir aquest tipus de dades per crear les distribucions ordenades dins de la classe. Per poder executar els algorismes necesitabem una estructura de dades que ens permeteix canviar l'ordenació dels productes, degut a aquest factor vem descartar estructures de dades com sets o maps. La decisió final va ser entre un vector i una lista, ens vam decantar per la lista.
  - Com que no és el mateix tipus que el que s'espera retornar tenim una funció privada a la classe AbstractAlgorithm que permet canviar d'aquesta estructura de dades a un ArrayList¡ArrayList¡Product¿¿.

- **HashSet<Product>:**
  - El fem servir per representar els productes que ja han sortit a la distribució i/o no han de sortir. Hem escollit un HashSet perquè així fem que es guardi només una instància de cada producte i també podem buscar si un producte és al set de manera ràpida
  - Amb el HashSet el temps per trobar si un producte ha sigut usat amb la operació contains() del HashSet es de O(1), mentres que si fessim servir una estructura de dades no ordenada hauriem de recórrer tots els productes usats, això dóna una complexitat de O(n).

- **List<List<Product>>**:
  - El fem servir al algorisme de força bruta per emmagatzemar totes les solucions trobades. Cada List¡Product¿ és una solució i vam decidir guardar-les a una llista pero podriem haver fet servir altres estructures de dades, ja que només necessitabem una estructura que pogués guardas les llistes de productes.

5 Algorismes

Volem tenir dues formes (algorismes) diferents d'ordenar les prestatgeries (shelf). Per implementar-ho hem creat una classe, abstracta AbstractAlgorithm, i una subclase d'aquesta per cada algorisme a implementar, BruteForceAlgorithm i HillClimbingAlgorithm.

La classe AbstractAlgorithm conté una funció orderProductList(...), que serà la funció que farà l'ordenació i és sobreescrita en les subclasses de AbstractAlgorithm, i un parell de funcions comunes auxiliars. A la funció orderProductList(...) li passem com a paràmetres, la llista de productes a ordenar, les mides de la prestatgeria, un límit i un mapa de les coordenades dels productes a la prestatgeria. L'atribut de límit serveix per limitar la cerca de la solució, però per calcular els costos dels algorismes no el farem servir, ja que buscarem el cost en el cas pitjor.

El cas pitjor serà quan la mida de la llista de productes sigui igual o menor al nombre de productes que poden haver-hi en aquesta prestatgeria (mida x * mida y).

5.1 BruteForceAlgorithm

Aquesta primera implementació es basa en un algorisme de força bruta (cerca exhaustiva), és a dir, l'algorisme calcula totes les possibles ordenacions i un cop les té totes, escull la millor.

La funció orderProductList crida a la funció generateCombinations. Aquesta guarda totes les combinacions i crida a la funció combineHelper que s'encarrega de la part recursiva del backtracking.
La funció combineHelper rep com atributs: la llista de productes, la combinació actual, el conjunt que conté totes les combinacions, el límit i la mida de la prestatgeria.
La funció té 2 casos base:

- El primer mira si la combinació actual té la mida de la prestatgeria, si la té insereix la combinació a la llista de totes les combinacions, si no continua amb el codi.
- L'altre cas base mira si el nombre de combinacions creades és igual al límit (si el límit és menor que 0 vol dir que s'executa ignorant els límits), si és igual acaba el backtracking, si no segueix.

La resta del codi és un backtracking normal, on es miren totes les combinacions de productes on no hi ha cap repetit.

5.1.1 Cost de BruteForceAlgorithm

L'algorisme primer fa operacions d'inicialització, totes de cost O(1) i seguidament calcula totes les possibles combinacions de productes.

El cost de generar totes les combinacions vàlides és de O(n!).
Els casos base tenen un cost de O(1) per tant, els ignorem. En el cas recursiu, el primer cop que s'accedeix el bucle farà N iteracions, la segona farà n -1 ja que no pot accedir a l'element que ja hem inserit a la combinació, la següent farà n - 2 iteracions i així fins a omplir la combinació. És a dir el cost és n * (n-1) * (n-2) * .... * 1 = n! .

Un cop tenim totes les combinacions hem de trobar la millor combinació, per això la suma de similituds de totes les combinacions.

La funció encarregada de calcular la suma de les similituds fa el següent: per cada element de la combinació, obté aquest producte i el següent de la combinació, obté la seva similitud i se la suma al total.
El cost d'obtenir cada producte és O(1), ja que accedeix directament a la memòria. El cost d'obtenir la similitud entre dos productes és O(1), pel fet que fem servir el mapa de similituds del producte. Per tant, el cost de calcular la suma de similituds és O(n*O(1)) = O(n).

En conseqüència, el cost de trobar la millor combinació del conjunt de mida n! és O(n * n!). Finalment, es crida la funció adaptToShelf que té de cost O(n). Amb tot això tenim el cost total de la funció orderProductList amb l'algorisme de força bruta, que és O(1+n!+n * n! + n) = O(n * n!)

5.1.2 Pseudocodi de BruteForceAlgorithm

```
funció ordenarLlistaProductes(llista, xsize, ysize, limit, coordenades):
    productes ← convertir a llista els productes de llista
    si productes està buida:
        llençar excepció "Llista buida"

    maxMida ← xsize * ysize
    combinacions ← generarCombinacions(productes, maxMida, limit)
    millor ← llista buida
    maxSimilitud ← 0

    per cada combinació en combinacions:
        suma ← calcularSuma(combinació)
        si suma > maxSimilitud:
            maxSimilitud ← suma
            millor ← combinació

    retornar adaptarADistribució(millor, xsize, ysize, coordenades)

funció generarCombinacions(productes, mida, limit):
    usats ← conjunt buit
    combinacions ← llista buida
    helperCombinacions(productes, mida, 0, llista buida, combinacions, usats, limit)
    retornar combinacions

funció helperCombinacions(productes, mida, inici, combinacióActual, combinacions, usats, limit):
    si mida(combinacióActual) = mínim(mida, mida(productes)):
        afegir còpia de combinacióActual a combinacions
        retornar

    si limit > 0 i mida(combinacions)  limit:
        retornar

    per cada producte a productes:
        si producte no és a usats:
            afegir producte a combinacióActual
            afegir producte a usats
            helperCombinacions(productes, mida, inici + 1,
                               combinacióActual, combinacions, usats, limit)
            eliminar producte de usats
            treure últim element de combinacióActual
```

5.2 HillClimbingAlgorithm

La segona implementació busca trobar una solució de manera més ràpida encara que aquesta no sigui perfecta. Per aconseguir-ho hem escollit fer servir un algorisme d'aproximació Hill Climbing. L'algorisme hill climbing parteix d'una solució inicial i després va mirant les solucions properes per trobar la millor de les veïnes.
Per fer la solució inicial s'escull un producte a l'atzar de la llista de productes i crea una distribució inicial respecte d'aquest producte. Com millor sigui la nostra solució inicial millor serà la nostra solució final, per això hem fet una solució tal que si representem la distribució amb un vector l'element [i] serà l'element disponible amb millor compatibilitat amb l'element [i-1].

Un cop tenim la solució inicial, es comença l'algorisme Hill Climbing. La funció hillClimbing té un bucle en el qual es busquen els veïns de la solució actual que acaba quan no es troba un veí millor o quan s'han fet tantes iteracions del bucle com l'atribut límit (si el límit és menor que 0 vol dir que s'executa ignorant els límits).

Per buscar els veïns pròxims tenim l'operació bestNeighbour on trobem els veïns de la solució actual i canvia la solució actual a la millor trobada. Els veïns s'aconsegueixen fent una sèrie de swaps entre els productes de la solució actual, aconseguint una solució pròxima a l'original. Mentre es generen els veïns guardem el millor d'aquests.

5.2.1 Cost de HillClimbingAlgorithm

El primer que fem és obtenir un producte aleatori, això ho fem amb la classe Random de java.util, agafem un integer aleatori i obtenim el producte a la posició d'aquest integer de la llista, fer això té cost O(1).

Després generem la solució inicial, que consisteix d'un bucle que es repetirà n cops, que executa la funció mostSimilarP. La funció mostSimilarP recorre tots els elements i mira la seva similitud, això té un cost de O(n) (hem explicat a l'algorisme BruteForce el cost d'aconseguir una similitud). Això ens dona com a resultat que el cost de la funció per obtenir la solució inicial és O(n^2).

Després per fer el hillClimbing es fa un bucle que en pitjor cas direm que fa k iteracions. Dins d'aquest bucle es crida a la funció bestNeighbor, que es pot explicar com un bucle dins d'un altre on a cada iteració es copia la llista (cost O(n)), es fa un swap de dos productes (cost O(1)) i es mira si és la millor solució (cost O(1)). Com a resultat tenim que la funció hillClimbing té un cost de $O(k * O(n * n * (n + 1))) = O(k * n^3)$.

Finalment, es crida la funció adaptToShelf que té de cost O(n). Amb tot això tenim el cost total de la funció orderProductList amb l'algorisme de Hill Climbing, que és $O(1 + n^2 + k * n^3) = O(k * n^3)$.

5.2.2 Pseudocodi de HillClimbingAlgorithm

```
function orderProductList(llista, xsize, ysize, limit, coordinates):
    if llista és buida:
        llança excepció "Llista buida"

    maxsize ← xsize * ysize
    randP ← selecciona un producte aleatori de la llista
    initialSolution ← iniSolution(maxsize, randP, llista)
    optimizedSolution ← hillClimbing(initialSolution, limit)

    return adaptToShelf(optimizedSolution, xsize, ysize, coordinates)


function iniSolution(size, startProduct, allProducts):
    solution ← [startProduct]
    used ← {startProduct}

    mentre la solution no estigui completa:
        nextProduct ← mostSimilarP(últimProducte a solution, allProducts, used)
        afegeix nextProduct a solution
        marca nextProduct com a utilitzat
    return solution

function hillClimbing(current, limit):
    repetir fins que no hi hagi millores o es compleixi el límit:
        neighbor ← best_neighbor(current)
        if calculaSum(neighbor) > calculaSum(current):
            current ← neighbor
        else:
            break
    return current


function best_neighbor(llista):
    bestSolution ← llista
    for cada parella de productes a llista:
        intercanvia la parella per crear un nou veí
        if calculaSum(neighbor) > calculaSum(bestSolution):
            bestSolution ← neighbor
    return bestSolution
```