Universitat Politècnica de Catalunya

Facultat d'Informàtica de Barcelona

Distribució de productes a un supermercat

Grup 31.5
Èric Díez Apolo (eric.diez)
Pol Carnicer González (pol.carnicer)
Francesc Pérez Venegas (francesc.perez.venegas)
Aleix Montero Ponce (aleix.montero)

PROP - Carles Arnal Castello
16 de Desembre del 2024

Índex

1. Descripció de clases

1.1 Descripció de les classes de domini i controladors

1.1.1 Perfil

La classe perfil és la classe que conté les dades de l'usuari (usuari i contrasenya) i que emmagatzema les diferents prestatgeries creades per l'usuari. S'encarrega principalment de la gestió dels usuaris i prestatgeries d'aquests, creant ambdós i poden eliminar-los i editar-los.

Els atributs d'aquesta clase son:

- Usuari(String): Emmagatzema el nom d'usuari associat al perfil.
- Contrasenya(String): Emmagatzema la contrasenya corresponent al perfil.
- Prestatgeria(ArrayList<Prestatgeria>): Una llista d'objectes Prestatgeria que representen les prestatgeries associades al perfil.

Els mètodes son:

- añadirPrestatgeria(Prestatgeria prestatgerias): Afegeix una prestatgerua a la llista de prestatgeries del perfil.
- eliminarPrestatgeria(Prestatgeria prestatgerias): Elimina una prestatgeria de la llista de prestatgeries del perfil.

1.1.2 Prestatgeria

La classe Prestatgeria proporciona una forma de gestionar i configurar prestatgeries mitjançant atributs com el nom, l'altura i una distribució associada. Permet establir o modificar la distribució per organitzar els productes dins de la prestatgeria, així com eliminar-la si cal.

Els atributs d'aquesta clase son:

- Nom(String): Emmagatzema el nom associat a la prestatgeria
- Distribució(Distribucio): Representa a la distribució de productes associada a la prestatgeria.
- Altura(Integer): Emmagatzema l'altura de la prestatgeria / nombre de prestatges.

Els mètodes son:

- Prestatgeria(String, int, Distribucio): Inicialitza una prestatgeria amb un nom donat, una altura i la distribucio associada.
- Prestatgeria(String, int): Inicialitza una prestatgeria amb un nom donat, una altura.
- eliminaDistribució(Distribucio distribucio): Elimina la distribució de la prestatgeria

1.1.3 Distribució

La classe Distribució proporciona una forma de gestionar i organitzar productes en una estructura mitjançant una llista de productes col·locats, classe que s'explicarà més endavant. Permet afegir, eliminar, intercanviar i ordenar productes, amb flexibilitat per utilitzar diferents algoritmes de càlcul. A més, pot gestionar restriccions com la fixació de certs productes en posicions específiques.

Els atributs d'aquesta clase son:

- nombre (String): El nom de la distribució.
- numero_prestatges (int): El nombre de prestatgeries en la distribució.
- usar_cambios_fijos (boolean): Indica si s'han realitzat canvis fixes en la distribució. Si és true, els productes amb l'atribut fixe es mantindran a la seva posició. Si és false, els productes es poden ordenar sense restriccions.
- productesColocats (ArrayList<ProducteColocat>): Llista d'objectes ProducteColocat que representen els productes que han estat col·locats dins de la distribució. Els productes dins d'aquesta llista es troben ordenats segons l'estratègia seleccionada per l'usuari.
- estrategiaCalculo (EstrategiaCalculo): Una estratègia utilitzada per calcular la distribució dels productes, segons la implementació que se li assigni es farà un càlcul o un altre.
- identificador_estrategia (int): Un identificador que indica el tipus d'estratègia utilitzada, utilitzada en les funcions de canvi d'estratègia.

Els mètodes son:

- Distribucio(String nom): Inicialitza una distribució amb un nom donat, amb 0 prestatgeries.L'estratègia per defecte és BackTracking.
- Distribucio(String nom, int numero_prestatge): Inicialitza una distribució amb un nom i un número de prestatgeries especificat. L'estratègia per defecte és BackTracking.
- Distribucio(): Constructor per defecte que inicialitza una distribució amb 0 prestatgeries i utilitza BackTracking com a estratègia.

- afegeix_producte_colocat(ProducteColocat producteColocat): Afegeix un producte col·locat a la llista de productes de la distribució i incrementa el número de prestatgeries en una unitat.

- obtenirProductePos(int pos): Retorna el producte situat a la posició indicada de la llista de productes col·locats. Com l'usuari enten que la distribució va de 1 a n, es resta en 1 unitat la posició introduïda a més de fer un control de que sigui un valor vàlid.

- elimina_producte(String producte): Elimina un producte de la distribució basant-se en el nom del producte. Es guarda per defecte que la posició a borrar serà la -1, després es busca la posició en la que es troba el producte en el bucle i finalment s'elimina de la llista de productes el producte de la posició trobada, sempre que no sigui la posició -1.

- intercambiar_productes(int seleccion1, int seleccion2): Intercanvia dos productes col·locats a les posicions indicades dins de la llista de productes col·locats, comprovant de que la selecció es de productes entre el rang vàlid.

- afegeixproducte(ProducteColocat producteColocat): Afegeix un producte col·locat a al final de la llista de productes col·locats, de manera que si s'imprimeix la distribució sense donar-li a calcular prèviament s'imprimeix el producte afegit al final.

- isCambiosManualesRealizados(): Retorna true si s'han realitzat canvis manuals fixos en la distribució, false en cas contrari.

- calcula_distribucio(int altura): Crida la funció de càlcul de la interficie que calcula i ordena la distribució dels productes utilitzant l'estratègia de càlcul actual, tenint en compte l'altura proporcionada.

- canvia_estrategia_calculo(EstrategiaCalculo estrategia, int id_estrategia): Canvia l'estratègia de càlcul per una nova estratègia, juntament amb el seu identificador que prové del controlador de distribució de domini que controla la lògica del canvi.

1.1.4 Producte

La classe producte representa a cadascun dels productes d'una distribució. Aquesta classe s'encarrega principalment d'emmagatzemar les dades de cada producte així com el nom, marca, preu i quantitat.

Els atributs d'aquesta clase son:

- nom (String): El nom del producte.
- marca (String): La marca del producte.
- preu (double): El preu del producte.
- quantitat (int): La quantitat disponible del producte.
- similituds (Map<Producte, Integer>): Un mapa que emmagatzema les similituds entre aquest producte i altres productes. La clau és un altre objecte Producte i el valor és un valor enter que indica el percentatge de similitud entre els productes (en un rang de 0 a 100).

Els mètodes son:

- Producte(String nom, String marca, double preu, int quantitat): Inicialitza un producte amb el nom, marca, preu i quantitat proporcionats.
- Producte(ArrayList<String> a): Inicialitza un producte a partir d'una llista de strings, amb valors per al nom, marca, preu (convertit a double) i quantitat (convertida a int).
- Producte(String nom): Inicialitza un producte amb el nom especificat i crea un mapa de similituds buit.

1.1.5 Producte Col·locat

La classe producte col·locat representa a cadascun dels productes d'una distribució un cop ja organitzats segons un dels algorismes o manualment. Aquesta classe s'encarrega principalment d'emmagatzemar les dades de la posició i en quina altura es troba cada producte.
Els atributs d'aquesta classe son:

- Altura(Integer): Representa a quina altura de la prestatgeria (el prestatge) està situat el producte.
- Pos(Int): Representa la posició del producte dins el prestatge.
- Producte(Producte): Una instància de la classe Producte que conté tota la informació del producte associat.
- ManualmenteModificado(boolean): Indica si la col·locació del producte ha estat modificada manualment, i s'inicialitza a false.

Els mètodes son:

- ProducteColocat(int pos, int altura, Producte producte): Assigna una posició, una altura i un producte a la instància creada. Inicialitza el camp manualmenteModificado com false.
- ProducteColocat(ArrayList<String> info, Producte prod): Construeix un objecte a partir d'una llista d'informació (info) i una instància de Producte. Interpreta els valors de pos, altura i si ha estat modificat manualment.
- ProducteColocat(Producte producte): Construeix un objecte només amb la informació del producte. Els altres camps queden per inicialitzar.
- esValidaAltura(Integer): Comprova si l'altura del producte és vàlida respecte a l'altura màxima de la prestatgeria. Retorna true si és vàlida, false en cas contrari.
- isManualmenteModificado(): Obté l'estat de si la col·locació ha estat modificada manualment.
-

1.1.6 Controlador Prestatgeria

El controlador de Prestatgeria s'encarrega de la gestió dels mètodes de la classe Prestatgeria. Es comunica amb Prestatgeria per tal de realitzar totes les funcionalitats que fan referència a aquest. I fa de pont entre la classe Prestatgeria i el seu driver corresponent.

1.1.7 Controlador Producte

El controlador de Producte s'encarrega de la gestió dels mètodes de la classe Producte. Es comunica amb Producte per tal de realitzar totes les funcionalitats que fan referència a aquest.

1.1.8 Controlador Producte Col·locat

El controlador de Producte Col·locat no deixa de tenir la mateixa funcionalitat que el controlador de Producte amb Producte, però aquest ho fa amb la clase Producte Col·locat.

1.1.9 Controlador Distribució

El controlador de Distribució és qui controla o té la majoria de lògica del programa, qui fa de pont entre el driver de distribució i la classe distribució on tenim la majoria de funcionalitats importants, com la de càlcul de distribució, la de mostrar la distribució per pantalla…

1.1.10 Controlador Perfil

El controlador de Perfil s'encarrega de la gestió dels mètodes de la classe Perfil. Es comunica amb Perfil per tal de realitzar totes les funcionalitats que fan referència a aquest. I fa de pont entre la classe Perfil i el seu driver corresponent.

1.1.11 Estrategia Cálculo

La estratègia "Estrategia Cálculo" és una interfície (no té codi). Serveix de pont entre la classe distribució i les implementacions dels algorismes de càlcul.

1.1.12 Estrategia TSP

La estratègia TSP es una clase que implementa la interfície “Estrategia Calculo” d'acord amb el patró Estratègia de disseny de Software. En aquesta classe es fa el càlcul de la nostra solució al problema del TSP adaptat al nostre problema de trobar la distribució amb les màximes similituds. Aquesta classe retorna una llista de productes ordenada i els productes que la componen ja tenen la seva posició i altura dels seus atributs actualitzades.

Els mètodes d’aquesta clase son:

- List<ProducteColocat> arrangeProductsBySimililarity(List<ProducteColocat>, int): Ordena els productes per similitud utilitzant un arbre de cobertura mínima (MST) i un cicle hamiltonià.

- List<ProducteColocat> prepara_salida(List<ProducteColocat>, ArrayList<String>) :Prepara la sortida convertint el cicle hamiltonià de cadenes a text a objectes `ProducteColocat`.

- ArrayList<Edge> generateEdges(List<ProducteColocat>):Genera arestes entre els productes basant-se en la seva similitud.

- private ArrayList<Edge> prim(ArrayList<Edge> edges, List<ProducteColocat>):Implementa l'algorisme de Prim per generar un arbre de cobertura mínima (MST) a partir de les arestes.

- int calcula_similitud(ArrayList<String>, ArrayList<Edge>):Calcula la puntuació de similitud per a una solució donada.

- int get_similitud_2_productes(String s, String s1, ArrayList<Edge>):Obté la similitud entre dos productes.

- Set<String> calcula_numero_nodos(ArrayList<Edge>) :Calcula el nombre de nodes únics en el cicle eulerià.

- cicloHamiltoniano(ArrayList<Edge>,ArrayList<Edge>, ArrayList<ProducteColocat>): Genera un cicle hamiltonià a partir del cicle eulerià.

- get_arista(Set<String>, String, ArrayList<Edge>): Obté la millor aresta per continuar el cicle hamiltonià.

  -actualitza_posicions(List<ProducteColocat>, int): Actualitza les posicions dels productes basant-se en la solució i l'altura.

1.1.14 Edge

Edge és una classe auxiliar que hem creat per a poder realitzar l'algorisme que implementa la nostra solució al problema TSP que hem implementat per a ordenar la nostra distribució.
La classe Edge consta de 2 strings, un string to i un string from que representen els dos productes als extrems de l'artista i un valor "peso" que indica la similitud entre els 2 productes.

Els atributs son:

- From(String): Fa referència a un producte de la similitud.
- To(String): Fa referència a l'altre producte de la similitud.
- Peso(Integer): Integer que representa la similitud en dos productes.

Els mètodes son:

- int compareTo(Edge o):

1.1.13 Estrategia OneToOne

La estrategia One to One és una classe que implementa la interfície "Estratègia Cálculo". La utilitzem en el test de la classe Distribució, no deixa de ser un stub de la classe de càlcul que retorna la mateixa distribució que se li ha ficat.

Els mètodes son:

- List<ProducteColocat>arrangeProductsBySimilarity(List<ProducteColocat>, int): Aquest mètode l'utilitzem per a fer tests i retorna la mateixa distribució d'entrada.

1.1.15 CalculBackTracking

La classe càlcul BackTracking no deixa de ser una altra Estratègia de càlcul que implementa la interfície “Estrategia Cálculo”. Aquesta calcula amb un algorisme de força bruta la millor distribució possible, la que té les similituds màximes, a canvi de trigar exponencialment i, per tant, amb grans quantitats de productes pot trigar bastant a fer el càlcul.

Aquesta classe no te atributs específics, llavors els seus mètodes serien:

- arrangeProductsBySimilarity(List<ProducteColocat> productes, int altura):
  Aquesta funció s'encarrega de disposar els productes en funció de la seva similitud. El procés inclou calcular quantes columnes (altura) es poden fer per distribuir els productes i generar permutacions dels productes no modificats manualment.
  Després es calcula la disposició amb la millor similitud total, tenint en compte els productes adjacents i el seu percentatge de similitud.
  Finalment, es retornen els productes disposats amb les posicions i altures ajustades.

- generarPermutaciones(ProducteColocat[] productos, int index, int size, List<ProducteColocat[]> permutaciones):

  Aquesta funció genera totes les permutacions possibles dels productes no modificats manualment.
  Utilitza un algorisme recursiu per intercanviar els elements i crear les permutacions, les quals es guarden en una llista.

- calculateSimilarity(Producte p1, Producte p2):
  Calcula la similitud entre dos productes. Utilitza el mètode getSimilitud de la classe Producte per obtenir la similitud entre els productes p1 i p2.
  Si algun dels productes és nul, retorna 0 com a similitud.

1.2 Descripció de les classes de presentació

1.2.1 Presentacion_Main

La classe *Presentacion_Main.java* és la classe principal de la capa de presentació que actua com a controlador principal de l'aplicació, gestionant la navegació entre les diferents vistes i el control dels perfils, prestatgeries i distribucions del sistema.

Components

- Controladors:
    - perfilActual: Perfil seleccionat actualment.
    - prestatgeriaActual: Prestatgeria seleccionada actualment.
    - controladorDistribucio: Controlador per gestionar la distribució de la prestatgeria seleccionada.
    - ControladorPerfilAux: Gestiona la càrrega i el desament de dades.
    - ControladorPrestatgerias: Gestiona la càrrega i el desament de dades.
- Interfície gràfica:
    - La finestra principal (JFrame) conté un sistema de navegació per mostrar diferents vistes segons l'acció de l'usuari.
- Altres estructures de dades:
    - Set usuarios: Emmagatzema els strings dels perfils del sistema per a poder fer desplegables per a la selecció del perfil.
    - Set prestatgerias: Emmagatzema els strings de les prestatgeries del perfil carregat per a poder fer desplegables per a la selecció de perfils.

Funcionalitats

- **Gestió de la interfície**
    - Mostrar menús i formularis

      La classe permet canviar la vista actual en funció de l'acció seleccionada. Les funcions, com mostrarMenuPrincipal() o mostrarFormularioCrearPerfil(), carreguen els panells associats a les vistes corresponents.

    - Actualitzar la vista

      El mètode actualizarVista() s'encarrega de substituir el contingut actual del JFrame amb un nou JPanel, refrescant així la finestra de manera eficient.

- **Gestió de perfils**

  Inclou funcions per carregar, guardar, esborrar i gestionar perfils d'usuari. A més, implementa les funcionalitats definides pels casos d'ús, que han estat prèviament desenvolupades en els drivers.

- **Gestió de prestatgeries**

  Ofereix funcions similars a les de la gestió de perfils, però aplicades a les prestatgeries. Això inclou carregar, guardar, esborrar i gestionar les prestatgeries definides pels casos d'ús i implementades en els drivers.

- **Gestió de distribucions**

  La gestió de distribucions es realitza mitjançant un únic controlador que està associat a la prestatgeria carregada. També inclou vistes específiques per gestionar les característiques de la distribució seleccionada.

- **Navegació entre vistes**

  La classe coordina la visualització i navegació entre diferents vistes, com ara:
    - Menús principals (perfils, prestatgeries, distribucions).
    - Formularis per crear, editar, eliminar i consultar dades.

- **Execució de l'aplicació**

  El mètode main() inicia l'aplicació creant una instància de Presentacion_Main dins d'un fil d'execució de Swing, assegurant una execució fluida de la interfície gràfica.

1.2.2 Perfil

1.2.2.1 MenuPrincipalView

La classe *MenuPrincipalView.java* és una vista amb un JPanel que serveix com a menú principal de l'aplicació per gestionar perfils.

Components

- Botons del menú:
    - Crear Perfil: Obre el formulari per crear un perfil nou.
    - Cargar Perfil: Obre el formulari per carregar un perfil existent.
    - Borrar Perfil: Obre el formulari per eliminar un perfil.
    - Editar Perfil:
        - Si hi ha perfils disponibles, obre el formulari d'edició.
        - Si no hi ha perfils, mostra un missatge d'error.
    - Guardar Cambios: Desa les dades actuals i mostra un missatge de confirmació.
    - Salir: Desa les dades actuals i tanca l'aplicació.
- Missatges d'interacció:
    - Si no hi ha perfils disponibles per editar, es mostra un missatge informant l'usuari.
    - Quan es desa la informació, es mostra un missatge de confirmació.

Funcionalitats

- Creació de perfils:
    - Obre el formulari associat per crear un nou perfil.
- Càrrega de perfils:
    - Permet carregar un perfil existent.
- Eliminació de perfils:
    - Proporciona accés al formulari per eliminar un perfil existent.
- Edició de perfils:
    - Comprova si hi ha perfils disponibles abans de permetre l'edició.
    - Mostra un missatge d'avís si no hi ha cap perfil disponible.
- Desament de dades:
    - Desa les dades actuals al sistema.
- Sortida de l'aplicació:
    - Desa les dades i tanca el programa de manera segura.

1.2.2.2 CrearPerfilView

La classe *CrearPerfilView.java* és una vista amb un JPanel que permet a l'usuari crear un perfil al sistema.

Components

- Camps d'entrada:
    - Usuari: Un camp de text per introduir el nom d'usuari.
    - Contrasenya: Un camp de contrasenya per introduir la contrasenya del nou perfil.
- Botons:
    - Crear: Valida les dades introduïdes i crea un perfil si l'usuari no existeix.
    - Atrás: Retorna al menú principal sense crear cap perfil.
- Missatges d'interacció:
    - Si l'usuari ja existeix, es mostra un missatge d'error.
    - Si el perfil es crea correctament, es mostra un missatge de confirmació.

Funcionalitats

- Validació d'usuari existent:
    - Comprova si el nom d'usuari ja existeix en el controlador Presentacion_Main.
    - Mostra un missatge si l'usuari ja existeix i impedeix la creació del perfil duplicat.
- Creació de perfil:
    - Crea un nou perfil amb el nom d'usuari i la contrasenya introduïts.
    - Desa el perfil a través del controlador i informa l'usuari del resultat.
- Navegació:
    - Permet tornar al menú principal sense realitzar cap acció mitjançant el botó Atrás.

1.2.2.3 BorrarPerfilView

La classe *BorrarPerfilView.java* és una vista amb un JPanel que permet a l'usuari eliminar un perfil existent del sistema.

Components

- Selecció de perfil:
    - Mostra una llista desplegable amb els noms dels perfils disponibles.
- Missatges d'interacció:
    - Si no hi ha perfils, s'informa l'usuari amb un missatge.
    - Si s'elimina un perfil, es mostra un missatge de confirmació.

Funcionalitats

- Comprovació de perfils:
    - Verifica si hi ha perfils disponibles abans de mostrar l'opció d'eliminar.
- Eliminació de perfil:
    - Elimina el perfil seleccionat cridant al controlador de presentació

1.2.2.4 EditarPerfilView

La classe *EditarPerfilView.java* és una vista amb un JPanel que permet seleccionar i editar un perfil existent després de validar la seva contrasenya.

Components

- Selecció de perfil:
    - Mostra una llista amb els noms dels perfils disponibles per editar.
- Camp de contrasenya:
    - Permet introduir la contrasenya del perfil seleccionat.
- Botons:
    - Login: Valida la contrasenya i permet editar el perfil si és correcta.
    - Enrere: Torna al menú principal.

Funcionalitats

- Comprovació de perfils:
    - Verifica si existeixen perfils abans de mostrar l'opció d'edició.
- Validació de contrasenya:
    - Comprova que la contrasenya introduïda coincideixi amb la del perfil.

- Accés a l'edició:
    - Si la contrasenya és correcta, es mostra l'opció d'editar el perfil.
    - Si és incorrecta, s'informa a l'usuari amb un missatge.
- Navegació:
    - Es pot tornar al menú principal amb el botó Enrere o cancel·lant la selecció.

Aquesta vista utilitza un JOptionPane per la selecció inicial i valida l'accés de manera segura amb un JPasswordField.

1.2.2.5 CargarPerfilView

La classe *CargarPerfilView.java* és una vista amb un JPanel que permet carregar un perfil existent mitjançant un nom d'usuari i contrasenya.

Components

- Camps de text:
    - Usuari: Per introduir el nom del perfil.
    - Contrasenya: Camp segur per introduir la contrasenya.
- Botons:
    - Carregar: Valida les credencials i carrega el perfil si són correctes.
    - Enrere: Retorna al menú principal.

Funcionalitats

- Validació de credencials:
    - Comprova que el nom d'usuari existeixi i que la contrasenya introduïda coincideixi amb la guardada.
- Càrrega del perfil:
    - Si les credencials són correctes, el perfil es carrega i es mostra el menú principal de prestatgeries.
- Gestió d'errors:
    - Si les credencials són incorrectes, es mostra un missatge d'error a l'usuari.
- Navegació:
    - Permet tornar al menú principal mitjançant el botó Enrere.

1.2.2.5 MostrarEditarPerfilView

La classe *MostrarEditarPerfilView.java* és una vista amb un JPanel que permet editar les dades d'un perfil existent.

Components

- Camps de text:
    - Usuari: Mostra i permet modificar el nom d'usuari.
    - Contrasenya: Mostra i permet modificar la contrasenya.
- Botons:
    - Guardar: Guarda els canvis realitzats en el perfil.
    - Enrere: Torna al menú principal.

Funcionalitats

- Edició del perfil:
    - Permet canviar el nom d'usuari i la contrasenya del perfil seleccionat.
- Comprovació de duplicats:
    - Si el nom d'usuari ja existeix, es mostra un missatge d'error i no es guarden els canvis.
- Actualització del perfil:
    - Elimina el nom antic del perfil i l'actualitza amb les noves dades.
- Navegació:
    - Permet tornar al menú principal amb el botó Enrere.

1.2.3 Distribució

1.2.3.1 AgregarProductoView

La classe *AgregarProductoView.java* és una vista amb un JPanel que permet afegir productes al sistema de distribució.

Components:

- Camps de text: Nom, marca, preu i quantitat del producte.
- Botons:
    - Afegir: Valida les dades, comprova si el producte ja existeix i l'afegeix al sistema.
    - Enrere: Retorna al menú principal.

Funcionalitats:

- Validació:
    - Comprova que el preu i la quantitat siguin números vàlids.
- Comprovació:
    - Evita afegir productes duplicats.
- Assignació:
    - Assigna la posició següent al nou producte.
- Navegació:
    - Torna al menú principal quan es prem el botó Enrere.

1.2.3.2 CalculaDistribucionView

La classe *CalculaDistribucionView.java* és una vista amb un JPanel que permet calcular la distribució dels productes utilitzant diferents algoritmes.

Components

- Botons:
    - Backtracking: Calcula la distribució dels productes utilitzant l'algorisme de backtracking.
    - TSP (Traveling Salesman Problem): Calcula la distribució utilitzant l'algorisme de TSP.
    - Enrere: Torna al menú principal de distribució.

Funcionalitats

- Càlcul de distribució:
    - Backtracking: Crida al mètode calcula_distribucio amb el paràmetre 1 per calcular la distribució mitjançant backtracking.
    - TSP:
        - Si només hi ha un producte, utilitza backtracking.
        - Si hi ha més d'un producte, crida al mètode calcula_distribucio amb el paràmetre 2 per aplicar l'algorisme TSP.
- Visualització de resultats:
    - Mostra la matriu resultant de la distribució en una nova vista, amb un títol descriptiu segons l'algorisme utilitzat.
- Navegació:
    - Permet tornar al menú principal de distribució amb el botó Enrere.

1.2.3.3 ConsultaSimilitudesView

La classe *ConsultaSimilitudesView.java* és una vista que permet consultar les similituds d'un producte seleccionat dins del sistema.

Components

- Etiqueta i ComboBox:
    - JLabel: Mostra un text indicant l'usuari que seleccioni un producte.
    - JComboBox: Desplegable amb els productes disponibles.
- Àrea de text:
    - JTextArea dins d'un JScrollPane per mostrar els resultats de les similituds de manera ordenada.
    - L'àrea no és editable.
- Botons:
    - Consultar: Mostra les similituds associades al producte seleccionat.
    - Enrere: Torna al menú principal de distribució.

Funcionalitats

- Consulta de Similituds:
    - L'usuari selecciona un producte del JComboBox.
    - En fer clic a Consultar, es recuperen les similituds mitjançant el controlador de distribució.
    - Les similituds es mostren a l'àrea de text en format llista, amb el producte relacionat i el nivell de similitud.
    - Si no hi ha similituds, es mostra un missatge indicant-ho.
- Gestió de productes buits:
    - Si no existeixen productes al sistema, apareix un missatge d'error i la vista no es mostra.
- Navegació:
    - El botó Enrere permet tornar al menú principal de distribució.

Disseny

La vista utilitza GridBagLayout per organitzar els components de manera flexible, mantenint una estructura clara:

- Primera fila: Etiqueta i ComboBox.
- Segona fila: Àrea de text amb barra de desplaçament.

- Tercera fila: Botons Consultar i Enrere.

Aquest disseny facilita la interacció de l'usuari, mostrant de forma ordenada tant l'entrada com la sortida de dades.

1.2.3.4 ConsultarProductosView

La classe *ConsultarProductosView.java* és una vista senzilla que permet consultar la llista de productes disponibles al sistema.

Components

- JPanel: El panell principal de la vista.
- JOptionPane: Utilitzat per mostrar missatges i la llista de productes.

Funcionalitats

- Consulta de Productes:
    - Es recupera la llista de productes del controlador de distribució.
    - Si la llista està buida, es mostra un missatge d'error mitjançant un JOptionPane i no es carrega la vista.
    - Si hi ha productes, es mostra una finestra emergent amb la llista completa de productes, separats per línies.
- Gestió de casos buits:
    - Si no hi ha productes, es mostra un missatge indicant que no n'hi ha per consultar.

Disseny

La vista no conté elements visuals complexos, ja que tota la informació es mostra a través d'un JOptionPane.
El panell principal es crea, però no conté components interactius, mantenint la implementació molt senzilla i funcional.

1.2.3.5 EliminarProductoView

La classe *EliminarProductoView.java* permet eliminar un producte de la llista de productes disponibles al sistema.

Components

- JPanel: El panell principal que conté el disseny de la vista.
- JOptionPane: Utilitzat per mostrar el diàleg per seleccionar el producte a eliminar i per mostrar missatges d'informació.

Funcionalitats

- Selecció del Producte:
    - La classe recupera la llista de productes del controlador de distribució.
    - Si la llista està buida, es mostra un missatge informant que no hi ha productes per eliminar.
    - Si hi ha productes, s'obre un diàleg (JOptionPane) que permet a l'usuari seleccionar un producte per eliminar.
- Eliminació del Producte:
    - Un cop seleccionat el producte, s'eliminen les dades corresponents mitjançant el mètode eliminaProducte del controlador de distribució.
    - Un missatge informant que el producte ha estat eliminat es mostra amb un altre JOptionPane.
- Gestió de casos buits:
    - Si l'usuari no selecciona cap producte o si no hi ha productes per eliminar, es tanca el panell sense realitzar cap acció.

Disseny

La vista és senzilla, amb una interfície mínima que inclou:

- Un diàleg per a la selecció del producte a eliminar.
- Un panell de GridLayout que conté la gestió d'interacció a través del JOptionPane.

1.2.3.6 ModificarPosicionProductosView

La classe *ModificarPosicionProductosView.java* permet modificar la posició de dos productes dins de la llista de productes.

Components

- JPanel: Utilitzat per contenir els components de la interfície gràfica. S'utilitza un GridLayout per disposar els components.
- JComboBox: Utilitzat per mostrar dues llistes desplegables amb els productes disponibles per poder seleccionar els productes a intercanviar de posició.
- JOptionPane: Es fa servir per mostrar diàlegs de confirmació i missatges d'alerta.

Funcionalitats

- Selecció de Productes:
    - Es recupera la llista de productes del controlador de distribució i es mostren dues llistes desplegables (JComboBox) per seleccionar dos productes diferents que es volen intercanviar de posició.
- Intercanvi de Posició:
    - Quan l'usuari selecciona els dos productes i pressiona OK, es comprova que els productes seleccionats siguin diferents. Si són iguals, es mostra un missatge d'error.
    - Si els productes són diferents, es recuperen les posicions dels productes seleccionats i es fa l'intercanvi mitjançant el mètode intercambiar_productes del controlador de distribució.
- Gestió de casos buits:
    - Si la llista de productes està buida, es mostra un missatge informant que no hi ha productes per modificar, i es tanca la vista.

Disseny

La vista consta d'un disseny simple que conté dos JComboBox per seleccionar els productes. Els components estan disposats en una taula de GridLayout amb dues files i dues columnes. Quan l'usuari fa la selecció, un JOptionPane s'encarrega de mostrar la finestra de confirmació per efectuar l'acció.

1.2.3.7 ModificarProductoView

La classe *ModificarProductoView.java* permet modificar la informació d'un producte existent, com el preu i la quantitat.

Components

- JPanel: Conté els elements gràfics i es gestiona mitjançant un GridLayout amb cinc files i dues columnes per a organitzar els components de la vista.
- JTextField: Permet editar el preu i la quantitat dels productes.
- JOptionPane: S'utilitza per mostrar missatges de confirmació i alerta.
- JButton: Serveix per executar les accions d'guardar els canvis o tornar al menú principal.
- JLabel: Etiquetes per a la informació visual a la interfície d'usuari.

Funcionalitats

- Selecció de Producte:
    - El sistema recull la llista de productes disponibles mitjançant el controlador i presenta una finestra emergent on l'usuari pot seleccionar el producte que desitja modificar.
- Mostrar informació del Producte:
    - Un cop seleccionat el producte, el sistema mostra el preu actual i la quantitat actual, que no es poden editar. Això es fa per proporcionar informació de context a l'usuari.
- Modificació del Preu i la Quantitat:
    - L'usuari pot introduir un nou preu i una nova quantitat per al producte seleccionat.
    - Si l'usuari intenta introduir un valor no numèric, es mostra un missatge d'error per indicar que els valors no són vàlids.
- Guardar els Canvis:
    - Quan l'usuari fa clic al botó "Guardar", el sistema valida els nous valors i actualitza el preu i la quantitat del producte al controlador de distribució mitjançant els mètodes set_precio_producto i set_cantidad_producto.
- Gestió de Casos Buits:
    - Si no hi ha productes per modificar, es mostra un missatge d'error indicant que no hi ha productes disponibles.
- Botons de Navegació:
    - El botó "Guardar" actualitza el producte i mostra un missatge de confirmació.
    - El botó "Atrás" permet tornar al menú principal de distribució sense fer canvis.

Disseny

Es mostra una vista senzilla amb camps de text per al preu i la quantitat actual, així com camps de text per als nous valors que l'usuari pot editar.

1.2.3.8 ModificarSimilitudesView

La classe *ModificarSimilitudesView.java* permet a l'usuari modificar les similituds entre dos productes, establint un valor de similitud entre 0 i 100.

Components

- JPanel: Un contenidor principal per organitzar els components gràfics.

- JComboBox: Dues llistes desplegables permeten a l'usuari seleccionar els dos productes entre els quals s'establirà una similitud.
- JTextField: Permet a l'usuari introduir un valor de similitud (un número enter entre 0 i 100).
- JButton: Proporciona dos botons d'interacció:
    - Actualizar: Aplica i actualitza el valor de similitud seleccionat entre els dos productes.
    - Atrás: Torna al menú principal de distribució.
- JLabel: Etiquetes per descriure els camps i instruccions a la interfície d'usuari.

Funcionalitats

- Selecció de Productes:
    - El sistema permet a l'usuari seleccionar dos productes diferents des de dues llistes desplegables (JComboBox). Si l'usuari selecciona el mateix producte en ambdues llistes, es desactiva la possibilitat d'introduir una similitud.
- Visualització del Valor Actual de Similitud:
    - Quan l'usuari selecciona un primer i un segon producte, el sistema carrega el valor actual de similitud entre aquests productes i el mostra al camp de text JTextField.
- Actualització de la Similitud:
    - Quan l'usuari introdueix un nou valor de similitud i fa clic al botó Actualizar, el sistema valida que el valor sigui un número enter entre 0 i 100. Si el valor és vàlid, es guarda la similitud entre els dos productes seleccionats.
    - Aquest valor es guarda per ambdues direccions, és a dir, es guarda també la similitud del segon producte cap al primer.
- Validació i Errors:
    - Si l'usuari intenta establir una similitud per a dos productes iguals, es mostra un missatge d'error indicant que cal seleccionar dos productes diferents.
    - Si el valor de similitud és incorrecte (fora de l'interval de 0 a 100 o no és un nombre enter), es mostra un missatge d'error.
    - Si l'usuari introdueix un valor no numèric per a la similitud, es mostra un missatge d'error indicant que s'han d'introduir valors numèrics vàlids.
- Botons de Navegació:
    - El botó "Actualizar" actualitza el valor de la similitud entre els productes seleccionats i mostra un missatge de confirmació.
    - El botó "Atrás" permet tornar al menú principal de distribució sense realitzar canvis.

Disseny

La vista es presenta en un GridLayout amb quatre files i dues columnes, amb els components ordenats de manera coherent.

Les etiquetes i els camps de text estan alineats i fàcils de llegir per millorar l'experiència de l'usuari.

El sistema utilitza una interfície senzilla i eficaç, amb el camp de similitud formatat com un camp de text editable per a l'entrada de dades.

1.2.3.9 MostrarMatrizResultadosEditableView

La classe *MostrarMatrizResultadosEditableView.java* mostra una matriu de resultats en una taula editable dins de la interfície gràfica d'usuari (GUI).

Components

- JPanel: Un contenidor per organitzar els components de la vista.
- JTable: Taula que mostra els resultats en forma de matriu. Cada cel·la pot ser modificada en funció de la configuració.
- JButton: Dos botons d'interacció:
    - Modificar Posición: Permet modificar la posició de les dades a la matriu.
    - Atrás: Torna al menú principal de la distribució.
- JScrollPane: Un component per afegir desplaçament a la taula si és massa gran per cabre en la finestra.
- JViewport: Permet centrar la taula dins de la finestra, assegurant que la taula sempre estigui ben posicionada dins del contingut desplaçable.

Funcionalitats

- Visualització de la Matriu:
    - La matriu es mostra com una taula JTable amb les dades proporcionades a la classe. Els elements nuls o buits a la matriu es reemplaçaran per la cadena "Vacio" per evitar que es mostrin valors buits o nuls.
    - La taula es configura per ser no editable a les cel·les de la matriu (amb isCellEditable establert a false), tot i que el codi podria ser ajustat per permetre edicions directes si és necessari.
- Desplaçament:
    - Es fa servir un JScrollPane per permetre l'scroll si la taula excedeix la mida de la finestra. La taula es posiciona automàticament al centre de la finestra utilitzant un JViewport, millorant l'experiència d'usuari quan la matriu és gran.

- Interacció amb Botons:
    - Modificar Posición: Aquest botó obre una nova vista per modificar la posició de les dades a la taula, executant el mètode ModificarPosicionProductos de la classe Presentacion_Main.
    - Atrás: Aquest botó torna al menú principal de la distribució, executant el mètode mostrarMenuPrincipalDistribucio.

Disseny

La vista utilitza un GridBagLayout per col·locar els components de manera flexible, amb l'objectiu de controlar millor la posició dels components dins del contenidor.

La taula ocupa la major part de l'espai de la finestra, amb els botons col·locats a la part inferior.

Es fa servir GridBagConstraints per ajustar les propietats de la taula i els botons, com l'espai de les vores, l'ancoratge, i la distribució del pes dins del contenidor.

1.2.3.10 MostrarMatrizResultadosView

La classe *MostrarMatrizResultadosView.java* s'encarrega de mostrar una matriu de resultats en una taula no editable dins de la interfície gràfica d'usuari (GUI).

Components

- JPanel: Contenidor principal de la vista, que utilitza un GridBagLayout per organitzar els components dins de la finestra.
- JTable: Taula que mostra els resultats, utilitzant un model de taula personalitzat a través de DefaultTableModel.
- JButton: Un botó anomenat "Atrás" que permet a l'usuari tornar al menú principal de la distribució.
- JScrollPane: Un component que afegeix funcionalitat de desplaçament a la taula per a la visualització de matrius grans.
- JTableCellRenderer: S'utilitza per centrar el contingut dins de les cel·les de la taula.

Funcionalitats

- Visualització de la Matriu:
    - La matriu es mostra com una taula JTable amb les dades passades a la classe. Els valors nuls o buits es substitueixen per la paraula "Vacio".
    - La taula es crea amb un model de taula DefaultTableModel que utilitza la matriu proporcionada i genera automàticament les columnes amb el nom "Columna n", on n és el número de la columna.

- Desplaçament:
    - Es fa servir un JScrollPane per permetre l'scroll si la taula excedeix les dimensions de la finestra. Això permet navegar fàcilment per matrius grans sense que es tallin o no siguin accessibles.
- Interacció amb Botó:
    - Atrás: Aquest botó permet als usuaris tornar al menú principal de distribució mitjançant l'execució del mètode mostrarMenuPrincipalDistribucio de la classe Presentacion_Main.

Disseny

El GridBagLayout es fa servir per al disseny flexible de la finestra, controlant el pes, les mides i l'ancoratge de cada component.

La taula ocupa la major part de l'espai de la finestra, mentre que el botó es col·loca a la part inferior de la finestra, proporcionant una interfície neta i clara.

Els GridBagConstraints s'utilitzen per ajustar la disposició i la mida dels components dins del contenidor.

1.2.3.11 MenuDistribucionView

La classe *MenuDistribucionView.java* és la vista que presenta un menú d'operacions disponibles per gestionar productes i distribucions dins de l'aplicació. A través d'aquest menú, l'usuari pot afegir, eliminar o modificar productes, així com consultar similituds, calcular distribucions i realitzar altres accions relacionades amb el control de distribució.

Components

- JPanel: Conté tots els botons i organitza la disposició del menú.
- JButton: S'utilitzen per les diferents opcions del menú, com afegir, eliminar, modificar productes, calcular distribució, etc.
- GridBagLayout: Es fa servir per organitzar els botons de manera flexible dins del panell. Cada botó es col·loca en una posició diferent del panell amb l'ús de GridBagConstraints.
- ActionListeners: Cada botó té associat un ActionListener per capturar l'acció de l'usuari i executar les operacions corresponents, com obrir formularis o mostrar pantalles de resultats.

Funcionalitats

- Gestió de Productes:
    - Añadir Producto: Obre un formulari per afegir nous productes.
    - Eliminar Producto: Obre un formulari per eliminar productes existents.
    - Modificar Producto: Permet modificar les propietats d'un producte existent.
- Gestió de Similituds:
    - Modificar Similitudes: Permet modificar les similituds entre productes.
    - Consultar Similitudes: Permet veure les similituds entre productes.
- Consulta de Productes:
    - Consultar Productos: Obre un formulari per consultar la informació dels productes existents.
- Distribució:
    - Calcular Distribución: Obre un formulari per calcular la distribució dels productes.
    - Mostrar Distribución: Mostra la matriu de distribució calculada anteriorment en format editable.
- Gestió de Dades:
    - Guardar: Guarda les dades actuals de l'aplicació.
- Control de Navegació:
    - Atrás: Torna al menú anterior (menú principal de la prestatgeria).
    - Salir: Desa les dades i tanca l'aplicació.

1.2.4 Prestatgeria

1.2.4.1 CrearPrestageriaView

La classe *CrearPrestatgeriaView.java* s'encarrega de crear una nova prestatgeria amb un nom i una alçada especificada per l'usuari. Aquesta vista permet a l'usuari introduir les dades per crear una prestatgeria i, en cas de voler-ho, tornar al menú principal.

Components

- JPanel: El contenidor principal de la vista utilitza un GridLayout per organitzar els components en una matriu de 3 files i 2 columnes. Això facilita la disposició dels camps de text i els botons de manera alineada i clara.
- JTextField: Dos camps de text són utilitzats per a l'entrada de dades de l'usuari: un per al nom de la prestatgeria i un altre per a l'alçada. Ambdós camps estan alineats al centre i tenen un estil de font sans-serif per garantir llegibilitat i un aspecte modern.

- JLabel: S'utilitzen etiquetes per descriure el que l'usuari ha de posar a cada camp de text. Les etiquetes es mostren centrades per millorar la coherència visual i l'estructura de la interfície.
- JButton: Dos botons són usats:
    - Crear: Aquest botó activa la creació de la prestatgeria amb les dades introduïdes.
    - Atrás: Permet tornar al menú principal. El botó està dissenyat de manera que sigui fàcil d'identificar i d'usar, amb un estil coherent amb el botó "Crear".
- JOptionPane: S'utilitza per mostrar missatges d'error o confirmació. Per exemple, si l'usuari intenta crear una prestatgeria amb un nom existent o si no introdueix una alçada vàlida, es mostren missatges d'alerta.

Funcionalitats

- Creació de Prestatgeria: Quan l'usuari omple els camps amb el nom i l'alçada de la prestatgeria, es comprova si el nom ja existeix. Si no, es crea la prestatgeria amb les dades introduïdes i es mostra un missatge de confirmació.
- Validació de Dades: Si l'usuari introdueix un valor no numèric per a l'alçada, es mostrarà un missatge d'error que indica que cal una alçada vàlida.
- Interacció amb Botons:
    - Crear: Inicia el procés de creació de la prestatgeria i mostra un missatge de confirmació si la creació és exitosa.
    - Atrás: Permet a l'usuari tornar al menú principal sense realitzar cap canvi.

1.2.4.2 BorrarPrestageriaView

La classe *BorrarPrestatgeriaView.java* s'encarrega de permetre a l'usuari eliminar una prestatgeria existent. L'usuari pot seleccionar la prestatgeria que vol esborrar d'una llista desplegable, i, si la selecció és vàlida, la prestatgeria és eliminada del sistema.

Components

- JPanel: El contenidor principal de la vista, que utilitza un GridLayout amb una sola cel·la (1 fila x 1 columna). Aquest disseny simple ajuda a mantenir la finestra clara i enfocada en la selecció de la prestatgeria.
- JOptionPane: Un component clau per interactuar amb l'usuari. Es fa servir per mostrar un quadre de diàleg que permet a l'usuari seleccionar una prestatgeria de la llista.

    - Si no hi ha prestatgeries disponibles, es mostra un missatge d'alerta informant que no es poden eliminar prestatgeries.
    - Un JOptionPane.showInputDialog es fa servir per presentar una llista de noms de prestatgeries disponibles a l'usuari. Aquest element mostra un llistat de noms que l'usuari pot seleccionar i eliminar.
- String[]: Una matriu de cadenes que conté els noms de totes les prestatgeries disponibles per eliminar. Aquest array és generat a partir de la llista de prestatgeries de l'usuari.

Funcionalitats

- Selecció de Prestatgeria: Si l'usuari té prestatgeries disponibles, es presenta un quadre de diàleg perquè seleccionin la prestatgeria que volen eliminar. Aquesta llista és generada dinàmicament a partir de les prestatgeries existents al sistema.
- Eliminació de la Prestatgeria: Un cop seleccionada la prestatgeria, aquesta es elimina tant de la col·lecció de prestatgeries del sistema com de la llista de prestatgeries del perfil actual. Això es fa mitjançant l'ús del mètode remove per actualitzar les col·leccions.
- Retorn al Menú: Si l'usuari no selecciona cap prestatgeria o decideix cancel·lar la selecció, el mètode mostrarMenuPrincipalPrestatgeria de la classe Presentacion_Main és cridat per tornar al menú principal de prestatgeries.

1.2.4.3 CargarPrestageriaView

La classe *CargarPrestatgeriaVie.java* és responsable de permetre a l'usuari carregar una prestatgeria existent en el sistema, seleccionant una prestatgeria d'una llista per treballar-hi en el context actual.

Components

- JPanel: El contenidor principal de la vista, que utilitza un GridLayout amb una sola cel·la (1 fila x 1 columna). Aquesta disposició simple permet que tota la funcionalitat es concentri en la selecció de la prestatgeria que l'usuari vol carregar.
- JOptionPane: Es fa servir per mostrar un quadre de diàleg emergent, on es presenta una llista de prestatgeries disponibles perquè l'usuari pugui seleccionar la prestatgeria a carregar.
    - Si no hi ha prestatgeries disponibles, es mostra un missatge informant que no es poden carregar prestatgeries.
    - Si es selecciona una prestatgeria, es carrega com a prestatgeria actual.

- String[]: Una matriu de cadenes que conté els noms de les prestatgeries existents en el sistema. Aquest array es genera a partir de les prestatgeries disponibles al sistema per tal que l'usuari pugui triar-ne una.

Funcionalitats

- Selecció de Prestatgeria: Quan l'usuari accedeix a la vista, el sistema l'informa sobre la disponibilitat de prestatgeries. Si n'hi ha disponibles, es mostra un quadre de diàleg amb una llista de noms. L'usuari ha de seleccionar una prestatgeria per carregar-la.
- Carregar la Prestatgeria: Si l'usuari selecciona una prestatgeria, el sistema la carrega com a prestatgeria actual mitjançant el mètode setPrestatgeriaActual. A continuació, es crea una distribució inicial per a aquesta prestatgeria amb el mètode crearDistribucionInicial.
- Retorn a Menú: Si l'usuari no selecciona cap prestatgeria o decideix cancel·lar, el sistema torna al menú principal de prestatgeries mitjançant mostrarMenuPrincipalPrestatgeria.

1.2.4.4 EditarPrestageriaView

La classe *EditarPrestatgeriaView.java* és responsable d'editar una prestatgeria existent al sistema, permetent a l'usuari canviar el nom i l'altura d'una prestatgeria seleccionada.

Components

- JPanel: El contenidor principal de la vista, utilitza un GridLayout de 1 fila i 1 columna. Aquesta disposició simple facilita la visualització i interacció de la informació de la prestatgeria a editar.
- JOptionPane: S'utilitza per mostrar diversos quadres de diàleg:
    - Un quadre per seleccionar la prestatgeria a editar, mostrant una llista de les prestatgeries existents.
    - Un altre quadre per introduir els nous valors del nom i l'altura de la prestatgeria seleccionada.
    - Si l'usuari confirma els canvis, la prestatgeria es guarda amb els nous valors; si no, la vista es tanca sense realitzar modificacions.
- JTextField: S'utilitza per a introduir els nous valors del nom i l'altura de la prestatgeria seleccionada.

Funcionalitats

- Selecció de Prestatgeria: Quan l'usuari accedeix a la vista, se li mostra una llista de prestatgeries disponibles. Si no hi ha prestatgeries per editar, es mostra un missatge d'error.
- Edició de Prestatgeria: Un cop seleccionada la prestatgeria, es mostra un quadre de diàleg amb els camps per modificar el nom i l'altura de la prestatgeria. Si l'usuari confirma l'edició:
    - El sistema elimina la prestatgeria anterior de la llista de prestatgeries.
    - S'actualitzen el nom i l'altura amb els nous valors introduïts.
    - La prestatgeria modificada es torna a afegir a la llista de prestatgeries amb el seu nou nom.
- Confirmació i Cancel·lació: Si l'usuari confirma els canvis, se'l notifica que la prestatgeria s'ha editat amb èxit. Si cancela, es torna al menú principal de prestatgeries sense realitzar canvis.

1.2.4.5 MenuPrestageriaView

La classe *MenuPrestatgeriaView.java* gestiona la vista del menú principal per a la gestió de les prestatgeries. Des d'aquesta vista, l'usuari pot crear, carregar, editar o esborrar prestatgeries, així com guardar canvis i sortir de l'aplicació.

Components

- JPanel: El contenidor principal de la vista utilitza un GridLayout. Aquesta estructura de disseny proporciona una presentació neta i clara dels botons del menú.
- JButtons:
    - Crear Prestatgeria: Obrir el formulari per crear una nova prestatgeria.
    - Cargar Prestatgeria: Carregar una prestatgeria existent per treballar amb ella.
    - Editar Prestatgeria: Permet editar una prestatgeria existent.
    - Borrar Prestatgeria: Permet eliminar una prestatgeria.
    - Guardar Cambios: Desa els canvis realitzats en les prestatgeries o en altres dades del sistema.
    - Atrás: Torna al menú principal.
    - Salir: Tanca l'aplicació després de desar les dades.

Funcionalitats

- Gestió de Prestatgeries: Els botons de crear, carregar, editar i esborrar permeten a l'usuari interactuar amb les prestatgeries del sistema:
    - Crear: Obre una vista per a la creació d'una nova prestatgeria.
    - Cargar: Permet carregar una prestatgeria existent per gestionar-la.
    - Editar: Permet modificar una prestatgeria seleccionada.
    - Borrar: Permet eliminar una prestatgeria de la llista.
- Guardar Canvis: El botó Guardar Cambios permet desar tots els canvis realitzats dins del sistema. Un missatge emergent confirma que els canvis s'han desat correctament.
- Navegació entre Menús: Els botons Atrás i Salir permeten navegar entre diferents seccions de l'aplicació:
    - Atrás: Torna al menú principal del sistema.
    - Salir: Tanca l'aplicació després de desar les dades.

1.3 Descripció de les classes de la capa de persistencia

La capa de persistència és responsable de gestionar la interacció amb l'emmagatzematge persistent, com ara fitxers o bases de dades, assegurant que les dades de les classes de domini es puguin guardar i recuperar de forma segura i eficient. En aquest cas, la persistència s'implementa mitjançant la serialització d'objectes en fitxers al disc utilitzant classes de Java com *ObjectOutputStream* i *ObjectInputStream*.

Les classes ObjectOutputStream i ObjectInputStream són fonamentals per implementar la persistència en aquesta arquitectura:

- **ObjectOutputStream:**
  Aquesta classe s'encarrega de convertir un objecte en una seqüència de bytes a través d'un procés anomenat serialització. Aquests bytes són escrits a un fitxer, permetent que l'objecte es pugui emmagatzemar de forma persistent.

- **ObjectInputStream:**
  Aquesta classe fa l'operació inversa, coneguda com deserialització. Llegeix la seqüència de bytes d'un fitxer i la converteix en un objecte Java que es pot utilitzar al programa.

Aquestes dues classes són molt potents perquè permeten guardar i recuperar objectes complets (amb tots els seus camps i estructures internes) de forma senzilla. Això redueix la complexitat a l'hora de manipular dades a nivell de fitxers.

Per tant, aquesta capa de persistencia actua com un intermediari entre la lògica del domini i l'emmagatzematge físic de les dades.

1.3.1 Controlador Persistència Distribució

El controlador de persistencia de distribució no l'utilitzem en el nostre cas i l'hem deixat ja que els drivers l'utilitzen i no volem que apareguin més errors.

1.3.2 Controlador Persistència Perfil

El controlador de persistència de perfils és l'encarregat de gestionar els perfils en memòria. El model que hem seguit és que els perfils es guardin sense cap informació addicional més enllà del seu nom i la seva contrasenya. El controlador de persistència de perfils té diverses funcions implementades:

guardarPerfil(Perfil perfil)
Aquesta funció s'encarrega de guardar l'objecte perfil a memòria dins la carpeta PERFILES. Com s'ha mencionat anteriorment, abans de ser guardat, es crea un nou objecte buit sense cap classe adjacent i se li afegeix l'usuari i la contrasenya, de manera que només es guarda aquesta informació.

cargar1Perfil(String filePath)
Aquesta funció retorna un perfil corresponent al filePath introduït, i és utilitzada per la funció getPerfil.

cargarTodosPerfiles()
Aquesta funció retorna un Set<String> amb tots els perfils disponibles al sistema, permetent a la capa de presentació generar un desplegable.

limpiarDatos()
Aquesta funció serveix per esborrar totes les dades guardades.

getPerfil(String username)

Aquesta funció serveix per obtenir un perfil a partir d'un String, en aquest cas l'usuari introduït des de la capa de presentació.

deletePerfil(String username)

Aquesta funció s'encarrega d'eliminar un perfil a partir d'un String, en aquest cas l'usuari introduït des de la capa de presentació.

1.3.3 Controlador Persistència Prestatgeria

El controlador de persistència de prestatgeria és l'encarregat de gestionar les prestatgeries dels perfils, guardant-les parcialment en memòria amb el mateix nom del perfil que les identifica. A diferència dels perfils, aquest controlador sí que guarda la classe Prestatgeria amb tots els atributs i classes associades (en aquest cas, una distribució per prestatgeria i els productes). El controlador de persistència de prestatgeria té diverses funcions implementades:

guardarPrestatgeria(Prestatgeria prestatgeria, String nombreUsuario)

Aquesta funció s'encarrega de guardar una prestatgeria en memòria. Els paràmetres d'entrada són la prestatgeria que s'ha de guardar i el nom d'usuari del perfil que l'ha creada o guardada, per tal de saber a quin perfil pertany cada prestatgeria en disc.

cargar1Prestatgeria(String filePath)

Aquesta funció carrega una prestatgeria a partir del filePath d'entrada.

getArxiusPrestatgeria(String nomUsuari)

Aquesta funció retorna un Set<String> amb totes les prestatgeries associades a un perfil, permetent a la capa de presentació generar desplegables amb les prestatgeries disponibles.

editar_nombre_prestatgeria(String nombreAntic, String nombreNou)

Funció que modifica el nom de la prestatgeria i aplica les modificacions pertinents en memòria.

limpiarDatos()

Funció que neteja totes les dades.

buscarPrestatgeria(String nombreUsuario, String nombrePrestatgeria)

Funció que retorna una prestatgeria donat un usuari i el nom d'una prestatgeria. Aquesta funció essencialment crida a una altra funció.

getPrestatgeria(String nombreUsuario, String nombrePrestatgeria)

Funció que retorna una prestatgeria donat un usuari i el nom d'una prestatgeria, buscant a la carpeta PRESTATGERIES, que és diferent de la carpeta on s'emmagatzemen els perfils.

deletePrestatgeria(String nombreUsuario, String nombrePrestatgeria)

Funció que busca la prestatgeria identificada pel nom d'usuari i el nom de la prestatgeria, i l'esborra de memòria.

deletePrestatgerias(String nombreUsuario)

Funció que esborra totes les prestatgeries d'un usuari associades a un perfil en memòria.

2. Estructures de dades i algorismes utilitzats

2.1 Estructures de dades de tot el projecte

En aquesta secció es descriuen les estructures de dades implementades en el projecte per construir les prestatgeries i gestionar les distribucions, així com els algorismes utilitzats per optimitzar la distribució dels productes segons les seves similituds.

2.1.1 ArrayList

Un ArrayList és una estructura de dades basada en un array dinàmic que permet gestionar col·leccions d'elements de mida variable durant l'execució d'un programa. A diferència d'un array tradicional, que té una mida fixa, l'ArrayList s'expandeix automàticament quan es necessària més capacitat.

Internament, l'ArrayList utilitza un array per emmagatzemar els elements i redimensiona aquest array quan s'arriba a la capacitat màxima, augmentant la mida en una proporció constant (generalment un 50%). Això garanteix una gestió eficient de memòria i un accés ràpid per indexació, encara que les operacions d'inserció o eliminació poden tenir un cost més elevat si impliquen desplaçaments d'elements.

Hem utilitzat ArrayList en diverses parts del projecte per la seva versatilitat. Per exemple, la classe Distribució conté un ArrayList de productes col·locats, mentre que la classe Perfil gestiona un ArrayList de prestatgeries.

Per l'ús que l'hem donat a les Arraylist el cost d'insertar un objecte depèn de la seva posició, en la seva majoría el hem insertat al final de l'Arraylist amb un cost de promig de O(1) i un cost en cas pitjor de O(n). Als casos d'insertar a l'inici o en un punt intermig el cost és en tots casos O(n). I el cost d'eliminar objectes de l'Arraylist també depèn de la seva posició, en el nostre cas ho fem per valor, en aquest cas el seu cost és O(n).

2.1.2 HashMap

Un HashMap és una estructura de dades que emmagatzema parelles clau-valor, permetent un accés eficient als valors mitjançant les claus associades. Aquesta eficiència

s'aconsegueix utilitzant una funció hash que distribueix els elements en buckets (contenidors), reduint el temps de cerca, inserció i eliminació a gairebé constant, , en condicions ideals.

En el projecte, fem servir HashMap per gestionar les similituds entre productes. En lloc d'una estructura lineal que compari tots els productes entre si, cada instància de Producte conté un HashMap on les claus són altres productes i els valors representen els coeficients de similitud entre aquests. Això optimitza considerablement la gestió de relacions entre els productes.

També per guardar i comprovar perfils i prestatgeries en la capa de presentació, on es carreguen els perfils en un map i després també es carreguen les prestatgeries del perfil carregat en un altre map per a facilitar la comprovació i per poder accedir de forma còmoda a les prestatgeries.

Les funcions utilitzades amb HashMap entre les quals fem: cercar, insertar i eliminar tenen costos similars, en quant a la cerca el seu cost promig és O(1) i en cas pitjor, en utilitzar una versió de Java >= 8, el seu cost és O(log n). En el cas de la inserció, els costos son els mateixos que el de la cerca, O(1) en promig i O(log n) en cas pitjor per a la versió de Java >= 8. I finalment per a eliminar elements el seu cost també és el mateix que en els altres dos casos O(1) en promig i O(log n) en cas pitjor per a la versió de Java >= 8.

2.1.3 PriorityQueue

Una PriorityQueue és una estructura de dades basada en una cua amb prioritats, on cada element s'insereix amb una prioritat associada. A diferència d'una cua tradicional, els elements amb prioritats més altes es processen abans, independentment de l'ordre d'inserció. Internament, la implementació de la PriorityQueue es basa en un heap binari, garantint un temps de complexitat O(log n) per a operacions d'inserció i extracció.

Hem utilitzat aquesta estructura per al càlcul del Maximum Spanning Tree (MST) com a part de l'algorisme d'aproximació del problema del Traveling Salesman Problem (TSP) implementat en el projecte. Aquesta estructura és molt útil per a aquest algorisme, ja que permet tenir una estructura on els elements s'ordenen automàticament de major a menor prioritat, amb un temps d'inserció i extracció baix respecte altres estructures de dades.

2.1.4 Set

Un Set és una col·lecció que garanteix la unicitat dels seus elements, és a dir, no permet duplicats. Les operacions com add() i contains() estan optimitzades gràcies a la implementació basada en un HashSet, que utilitza una funció hash per a un accés eficient, amb una complexitat mitjana de .

En el projecte, fem servir el Set per gestionar els nodes visitats en l'algorisme d'aproximació del TSP. Això ens permet verificar fàcilment si un node ja ha estat visitat durant l'execució de l'algorisme, millorant l'eficiència global.

Per tal d'inserir, eliminar o cercar un producte, el cost de totes aquestes funcionalitats és le mateix, O(1) en cas promig i O(n) en cas pitjor.

2.1.5 Array unidimensional i bidimensional

Un array unidimensional és una estructura de dades que emmagatzema elements del mateix tipus en una seqüència lineal. Cada element s'identifica per un índex únic, permetent un accés directe i eficient. La seva mida és fixa un cop creat.

Un array bidimensional, per altra banda, és una estructura tabular que organitza elements en files i columnes, formant una matriu. Els elements s'indexen mitjançant dues dimensions: una per accedir a les files i una altra per les columnes. Aquesta estructura és ideal per representar dades tabulars de forma compacta i accedir-hi de manera directa.

En el cas específic d'arrays de cadenes (String), tant unidimensionals com bidimensionals, s'utilitzen per emmagatzemar text. Un array unidimensional de String podria representar una llista de noms, mentre que un bidimensional podria representar una taula amb múltiples columnes d'informació textual.
En el nostre projecte, hem utilitzat arrays bidimensionals principalment per transferir dades entre la capa de domini i la capa de presentació, com en el cas dels drivers. Aquesta estructura simplifica la serialització i deserialització de dades, especialment per a propòsits de depuració i visualització. Els arrays unidimensionals, per la seva banda, s'han emprat per gestionar llistes simples de dades textuals.

2.1.6 Elements de la llibreria de Swing

Swing és una biblioteca gràfica per a Java que proporciona components per crear interfícies gràfiques d'usuari (GUI) [6] . Swing ofereix un conjunt d'eines més avançat i flexible que el seu predecessor, l'Abstract Window Toolkit (AWT) i és part de les llibreries per defecte de Java.

Característiques principals de Swing

- Ofereix una àmplia gamma de widgets per a la interfície d'usuari.
- Proporciona un aspecte i aparença natius que emulen diverses plataformes.
- Permet personalitzar l'aparença de les aplicacions independentment de la plataforma subjacent.

Alguns elements importants de Swing i que hem utilitzat són:

- JButton: És un component que representa un botó clicable. Permet afegir text i/o icones, i es pot configurar per respondre a esdeveniments com clics.
- JFrame: És una classe fonamental que s'utilitza per crear finestres on s'afegeixen altres components.
- JPanel: Un contenidor que s'utilitza per agrupar i organitzar altres components.
- JLabel: S'utilitza per mostrar text o imatges no editables.
- JTextField: Permet a l'usuari introduir text en una sola línia.
- GridBagLayout: Gestor de disseny flexible de Swing que col·loca components en una quadrícula de files i columnes. Utilitza coordenades (gridx, gridy) i permet ajustar amplada (gridwidth) i alçada (gridheight). Ofereix control amb weightx, weighty i alineació amb anchor.
- JOptionPane:Classe de Swing per crear diàlegs estàndard, com missatges (showMessageDialog) o entrades (showInputDialog). També presenta opcions de confirmació (showConfirmDialog) o personalitzades (showOptionDialog). Són diàlegs modals.

Aquests components es poden personalitzar i combinar per crear interfícies d'usuari complexes i interactives. Swing utilitza un model basat en esdeveniments per gestionar les interaccions de l'usuari, permetent als desenvolupadors definir com l'aplicació ha de respondre a diferents accions.
Hem utilitzat Swing ja que ens ha semblat una llibreria molt útil i senzilla per a implementar una interfície senzilla, que es lo que ens demanen en Prop.

2.1.7 Fitxers .dat

Un fitxer amb l'extensió .dat és un tipus de fitxer que s'utilitza per emmagatzemar dades de manera genèrica. Aquestes dades poden provenir de diversos programes i poden incloure informació com text, imatges, àudio o vídeo.

Els fitxers .dat són creats normalment per aplicacions específiques i no estan dissenyats per ser oberts manualment; en general, el seu contingut es pot visualitzar amb un editor de text només si s'ha desat com a text.
Els fitxers .dat són molt versàtils i es poden utilitzar en una àmplia gamma d'aplicacions, com ara bases de dades, programes multimèdia, videojocs i sistemes de seguretat. La seva flexibilitat els permet emmagatzemar gairebé qualsevol tipus d'informació, cosa que els fa útils per a l'emmagatzematge de dades de programes.

Hem implementat la persistencia amb fitxers .dat per la facilitat que ens dona la gestió de dades.

2.2 Algorismes capa de domini

2.2.1 Algorisme de Backtracking

El backtracking és una tècnica de cerca exhaustiva que explora totes les possibles solucions d'un problema de manera sistemàtica, retrocedint quan es detecta que un camí no condueix a una solució vàlida o subòptima a una solució guardada previament calculada. Aquest enfocament és especialment adequat per a problemes d'optimització i combinatoris, encara que presenta una elevada complexitat computacional, amb un cost temporal de O(n!) [1] en el cas general.

En el context del nostre projecte, hem utilitzat aquest algorisme com a mètode de força bruta per determinar la millor distribució possible de productes en una prestatgeria. L'objectiu és assegurar que els productes estiguin ubicats al costat d'altres amb una alta similitud, maximitzant així l'eficiència de la distribució segons els criteris definits. Tot i que aquest algorisme garanteix una solució òptima, el seu cost computacional és significativament superior al de l'altre algorisme implementat, que ofereix una aproximació menys precisa i no sempre donant la solució més correcte però molt més eficient en termes de temps d'execució.

2.2.2 Algorisme Travelling Salesman Problem (TSP)

El TSP (Travelling Salesman Problem) [1] és un problema d'optimització que, originalment, consisteix a trobar el camí de cost mínim que passi per cada una de les ciutats i retorni al punt d'origen. En el nostre cas hem resolt el problema aplicant el mateix principi, però en comptes de trobar el camí de cost mínim, hem calculat el camí de cost màxim.

Com és un problema NP-complet [2], no coneixem cap manera de resoldre el problema en temps polinòmic.

Nosaltres hem implementat la nostra versió de la 2-Aproximació del TSP, obtenint un Maximum Spanning Tree (que és un Minimum Spanning Tree [3] però en comptes de tenir el menor pes possible de les arestes, té el major pes possible) amb l'algoritme de Prim [4] del nostre graf d'entrada que hem creat, ja que nosaltres estem treballant amb una Llista de Productes i no amb un graf. Amb el resultat, que és un conjunt d'arestes ponderades no dirigides, hem construït un graf dirigit, fent que cada aresta no dirigida passin a ser 2 arestes dirigides ponderades. Amb això ja hem obtingut un cicle eulerià, però ara mateix apareixen nodes repetits, així que hem calculat un cicle hamiltonià. L'algorisme del càlcul del cicle hamiltonià que hem implementat

3. Referències

[1] Explicació Notació O gran de la Wikipedia en anglès
https://en.wikipedia.org/wiki/Big_O_notation

[2] Explicació TSP de la Wikipèdia en anglès
https://en.wikipedia.org/wiki/Travelling_salesman_problem

[3] Explicació NP-Complet de la Viquipèdia
https://ca.wikipedia.org/wiki/NP-complet

[4] Explicació Minimum Spanning Tree de Wikipedia en anglès
https://en.wikipedia.org/wiki/Minimum_spanning_tree

[5] Explicació algoritme de Prim de la Viquipèdia
https://ca.wikipedia.org/wiki/Algorisme_de_Prim

[6] Explicació UI wikipedia
https://es.wikipedia.org/wiki/Interfaz_de_usuario