FitTipz Pseudocode:

Class User:

Attributes

- userId: int

- firstName: String

- lastName: String

- username: String

- email: String

- phoneNumber: String

- dateCreate: date

- clothingHistory: FitzStorage

Method:

**+ createAccount(userId: int, email: String, username: String) -> String:**

// Code to create a new account

// Set user attributes (userId, email, username)

// Set default values for other attributes (firstName, lastName, etc.)

return "Account created successfully."

Algorithm: **createAccount(userId: int, email: String, username: String)**

1. Begin

2. Validate if the userId is unique in the system:

- If userId exists, return an error message ("Account creation failed: userId already exists").

- If userId is unique, proceed.

3. Set userId to the provided value.

4. Set email to the provided email.

5. Set username to the provided username.

6. Set default values for:

- firstName = "Not Provided"

- lastName = "Not Provided"

- phoneNumber = "Not Provided"

- dateCreate = current date

- clothingHistory = empty FitzStorage object

7. Return "Account created successfully."

8. End

Method:

**+ deleteAccount() -> void:**

// Code to delete user account

// Set all attributes to default or null values

// Deactivate or delete account from the system

Algorithm: **deleteAccount()**

1. Begin

2. Clear all attributes of the user object:

- Set userId, firstName, lastName, username, email, phoneNumber to null or default values.

- Set dateCreate to null.

- Set clothingHistory to null or an empty FitzStorage object.

3. Return "Account deleted successfully."

4. End

Method:

**+ updateAccountDetails() -> void:**

// Code to update user details

// User can modify attributes like firstName, lastName, email, etc.

Algorithm: **updateAccountDetails()**

1. Begin

2. Prompt user to enter the details to update:

- First name, last name, email, phone number

3. If the user provides new details:

- Update the respective attributes with the new values.

4. Return "Account details updated successfully."

5. End

Method:

**+ displayAccountInfo() -> String:**

// Code to display user's account info

return "User information: userId, firstName, lastName, email, phoneNumber, etc."

Algorithm: **displayAccountInfo()**

1. Begin

2. Display the following account information:

- userId, firstName, lastName, username, email, phoneNumber, and dateCreate

3. End

Class FitzStorage:

Attributes

- storedClothing: Clothing

- favorite: Boolean = false

- totalAmountOfClothing: int = 0

- clothingCat: char

Method: Save a clothing item

**+ saveClothing() -> void:**

// Code to save clothing item to storedClothing

// Increase totalAmountOfClothing

// Mark clothing item as favorite if necessary

Algorithm: **saveClothing()**

1. Begin

2. Add the clothing item (stored in the "storedClothing" attribute) to the storage.

3. Increase totalAmountOfClothing by 1.

4. If the clothing item is marked as a favorite:

- Set the favorite attribute to true.

5. Return "Clothing item saved successfully."

6. End

Method:

**+ removeClothing() -> void:**

// Code to remove clothing item from storedClothing

// Decrease totalAmountOfClothing

Algorithm: **removeClothing()**

1. Begin

2. Check if there is any clothing in the storage:

- If the storage is empty, return an error message ("No clothing to remove").

3. Remove the clothing item from "storedClothing".

4. Decrease totalAmountOfClothing by 1.

5. Return "Clothing item removed successfully."

6. End

Method:

**+ enterSellerSiteRef() -> void:**

// Code to enter seller site reference

// Store URL or reference for clothing item sale

Algorithm: **enterSellerSiteRef()**

1. Begin

2. Ask user to input the seller's website URL.

3. Store the entered URL in the redirectionLink attribute.

4. Return "Seller site reference entered successfully."

5. End

Method:

**+ createRandomizedOutfit() -> String:**

// Code to randomly select a combination of stored clothing

// Return the combination of clothing as an outfit suggestion

return "Randomized Outfit: Clothing items"

Algorithm: **createRandomizedOutfit()**

1. Begin

2. If there is sufficient clothing in the storage (e.g., at least one item from each category, like upper wear, bottom wear, shoes):

- Randomly select one item from each category.

- Combine them to create a complete outfit.

3. Return the list or description of the randomized outfit.

4. If there is not enough clothing in the storage:

- Return "Insufficient clothing to create an outfit."

5. End

class Clothing:

Attributes

- clothingId: int

- upperWear: boolean = false

- bottomWear: boolean = false

- shoes: boolean = false

- redirectionLink: String

Method: Set clothing details

**+ setClothing(size: String, SMLX: char, gender: char, color: String, budget: double, brand: String) -> void:**

// Code to set the clothing details

// Set the attributes based on input parameters:

// - Set size (String)

// - Set SMLX (small, medium, large, extra-large, etc.)

// - Set gender (char) for clothing type (e.g., 'M' for male, 'F' for female)

// - Set color (String) of the clothing

// - Set budget (double) for the price range of the clothing

// - Set brand (String) of the clothing item

Algorithm: **setClothing(size: String, SMLX: char, gender: char, color: String, budget: double, brand: String)**

1. Begin

2. Validate the input parameters:

- Check if size, color, brand are valid (non-empty strings).

- Check if gender is 'M' or 'F'.

- Check if budget is a positive number.

- Check if SMLX is one of 'S', 'M', 'L', 'X'.

3. Set the following attributes based on the input values:

- Set size to the given size.

- Set SMLX to the given size value (S, M, L, X).

- Set gender to the provided gender ('M' or 'F').

- Set color to the given color.

- Set budget to the provided budget value.

- Set brand to the provided brand.

4. If all conditions are met, return "Clothing details updated successfully."

5. If any validation fails, return an error message about the invalid input.

6. End

Method:

**+ getClothing() > void:**

// Code to display or return clothing details

// Retrieve clothing details and display or return the information

// Example: return clothingId, upperWear, bottomWear, shoes, and redirectionLink

Algorithm: **getClothing()**

1. Begin

2. Retrieve and display the following attributes:

- clothingId, upperWear, bottomWear, shoes, redirectionLink

3. Return the clothing details.

4. End

Class: TipzTips

Attributes:

- recommendedItemId: int // key

- recommendedClothing: Clothing

Method:

**+ scanPreviouslySearched(): AdvancedSearch**

// Logic to return an AdvancedSearch object based on user history

Algorithm: **scanPreviouslySearched()**

1. Begin

2. Access the user's search history records.

3. Extract commonly used filter preferences such as:

- Brands

- Sizes

- Budget levels

- Colors

- Gender

4. Create a new AdvancedSearch object.

5. Populate the object using extracted filters.

6. Return the AdvancedSearch object.

7. End

Method:

**+ scanSavedClothing(): FitzStorage**

// Logic to scan user's saved clothing and return from FitzStorage

Algorithm: **scanSavedClothing()**

1. Begin

2. Access the user's saved clothing items from FitzStorage.

3. Analyze attributes like:

- Frequently saved brands

- Preferred sizes

- Clothing types and insulation preferences

4. Summarize patterns or preferences.

5. Return the relevant FitzStorage object or result.

6. End

Class: AdvancedSearch

Attributes:

- filterID: int // key

- brand: String = "" // default empty string

- gender: char

- budget: double = null // default null

- size: String = "" // default empty string

- insulation: Boolean = false //default false

- color: String = "" // default empty string

- mostRecentlyViewed: Clothing = null // default null

Methods:

**+ addFilter(option: int): int**

// Apply filter based on integer option

// Return new filter ID

Algorithm: **addFilter()**

1. Begin

2. Accept an input option representing a filter type.

3. Based on the value of option:

- If 1, prompt and assign a brand.

- If 2, prompt and assign a size.

- If 3, prompt and assign a gender.

- If 4, prompt and assign insulation (true/false).

- If 5, prompt and assign color.

4. Update filterID to reflect the new filter configuration.

5. Return the updated filterID.

6. End

Method:

**+ changeBudget(LMH: Char): double**

// Set budget range based on L (low), M (medium), or H (high)

// Return the numeric budget value set

Algorithm: **changeBudget(LMH: Char)**

1. Begin

2. Accept input LMH representing budget preference:

- 'L' for Low

- 'M' for Medium

- 'H' for High

3. Set the budget as follows:

- 'L' → budget = 50.0

- 'M' → budget = 100.0

- 'H' → budget = 200.0

4. Return the numeric budget value.

5. End

Method:

**+ removeFilter(): void**

// Clear all active filters in the search

Algorithm: **removeFilter()**

1. Begin

2. Reset all filter fields to default:

- brand = ""

- size = ""

- color = ""

- budget = null (unset or null)

- insulation = false

- mostRecentlyViewed = void

3. Update or reset filterID if needed.

4. Return nothing.

5. End

Class: VisualFitz

Attributes:

- pictureID: int {key} // Unique ID for the uploaded or captured image

- currentClothing: Clothing // Clothing object identified in the image

- validatedPicture: Boolean // True if image is validated for use

Methods:

**+ uploadImage(): void // Allows user to upload an image from their device**

Algorithm: uploadImage (in VisualFitz)

1. Begin

2. Prompt user to select an image from device storage or camera roll.

3. Store the image and assign a unique pictureID.

4. Validate image format (e.g., resolution, type).

5. Set validatedPicture to true if image passes checks; otherwise, false.

6. End

Method:

**+ pickGarments(): void // Detects and classifies garments in the uploaded image**

Algorithm: **pickGarments()**

1. Begin

2. Analyze the uploaded image to identify clothing items using image recognition.

3. Classify identified garments into categories (e.g., upperWear, bottomWear, shoes).

4. Create or update currentClothing object based on the analysis.

5. End

Method:

**+ takePicture(): void // Lets the user capture a new photo using the camera**

Algorithm: **takePicture()**

1. Begin

2. Activate device camera.

3. Allow user to capture a live photo.

4. Save the image and assign a pictureID.

5. Set validatedPicture to true after basic validation.

6. Store captured clothing as currentClothing.

7. End

Class: FitzCart

Attributes:

- cartId: int {key} // Unique ID for the shopping cart

- inCart: Clothing = void // Clothing item currently in the cart

- cost: double = 0.00 // Total cost of items in the cart

- clothingCat: char // Category of clothing ('U', 'B', 'S', etc.)

Method:

**+ addToCart(): void // Adds a clothing item to the cart**

Algorithm: **addToCart()**

1. Begin

2. Select a clothing item to add to the cart.

3. Set inCart to the selected clothing item.

4. Update cost with the price of the added clothing.

5. Determine and assign clothingCat based on the clothing type (e.g., 'U', 'B', 'S').

6. End

Method:

**+ removeFromCart(): void // Removes the item from the cart and resets values**

Algorithm: **removeFromCart()**

1. Begin

2. Check if inCart is not void (cart is not empty).

3. Set inCart to void (empty).

4. Reset cost to 0.00.

5. Clear clothingCat value.

6. End