Cwk 3: Furniture Management System (FMS)

This assignment contributes 25% of the total module marks

Explanation:

This project aims to design and implement a comprehensive furniture management system that caters to different categories of furniture. The software will keep track of the inventory, details of each furniture piece, purchasing information, and customers' details.

- The **FurnitureBase Class** provides a generic template for different furniture items.
- The **Purchaser Class** focuses on the details of customers, capturing essential information and their purchase history.
- The **FurnitureCategory Class** is concerned with classifying furniture into distinct categories based on specific criteria like their materials and suitability for different environments.
- **OfficeFurniture** and **KidsSet** classes cater to specialized furniture categories, each having its unique attributes and features, thereby enhancing the diversity of the furniture items the system can manage.

System Requirements

A. The following basic **functional** requirements have been established for the proposed system:

- **Inventory Management**:
    a. Maintain a catalog of available furniture items.
    b. Associate specific buyers with furniture items.
    c. Track the number of units of each furniture category.
- **Purchasing Information**:
    a. Keep a history of items purchased by customers.
    b. Determine discounts and offer gifts based on purchase history.
- **Customer Management**:
    a. Determine eligibility of purchasers based on their purchase history.
- **Furniture Classification**:
    a. Classify furniture based on categories.
    b. Track specific details of each category, like weight limits or suitability for outdoor use.
- **Specialized Furniture Management**:
    a. Maintain unique features for office furniture, like ergonomic attributes and material-based price adjustments.
    b. Manage kids' furniture sets with special offers and themed sets.

B. The following **data** requirements have been established:

The backbone of the furniture management system revolves around the robust and dynamic data sets that keep track of varying entities.

- **Furniture Data**: Central to the system is the furniture data, which encompasses several key attributes. At its core, every furniture piece is identified by its category, be it a sofa, table, or chair, captured under the 'furnitureCategory'. The 'itemCount' gives insight into how many units of a particular furniture category are available or have been sold. Additionally, each furniture piece's acquisition date is recorded under 'purchaseDate', allowing the system to track the age and potential depreciation of each item. To streamline management and provide a comprehensive view of the available items, the 'furnitureInventory' is designed as a catalog holding details of distinct furniture items.

- **Purchaser Data**: Equally crucial to the system's functionality is the data about the purchasers. Every individual or entity making a purchase is uniquely identified by an 'id'. Their personal or organizational name is stored under 'name', and the type of furniture they purchase is denoted as 'furnitureType'. To facilitate seamless communication and potential after-sales service, the system captures the contact details, such as email or phone number, under 'contactDetails'. Additionally, every transaction is timestamped, with the 'purchaseDate' reflecting the date of acquisition. For businesses looking to analyze consumer buying patterns or launch loyalty programs, the 'purchaseHistory' becomes invaluable. This attribute acts as a repository, logging every furniture item bought by a particular purchaser over time.
- **Furniture Category Data**: Diving deeper into the classifications, the system employs the 'Furniture Category Data' set. Each type or category of furniture, such as armchairs or coffee tables, is denoted by 'typeName'. Understanding the capabilities and limitations of each furniture piece is crucial, and thus, 'maximumLoad' provides the weight-bearing capacity of each item. Furthermore, the system is intuitive enough to differentiate between furniture designed for indoor or outdoor settings, indicated by the 'isOutdoor' attribute. A unique feature that sets the system apart is its ability to track the latest purchaser of a specific furniture category, stored under 'recentPurchaser'. For those keen on understanding the make and design, the 'materials' list delineates the various components, be it wood, leather, or metal, that go into crafting each piece.
- **Specialized Furniture Data**: Beyond the general categories, the system is adept at handling specialized furniture data, particularly catering to office settings and kids. Every office furniture item, from desks to cabinets, is categorized under 'furnitureCategory'. Modern emphasis on ergonomic design is reflected under the 'isErgonomic' attribute, while the core material of construction is captured in 'material'. To ensure transparency and facilitate budgeting, each item's cost is noted under 'price'. In the realm of kids' furniture, each set or collection is assigned a unique identifier termed 'kidsId'. This allows for quick identification and cataloging. The specific name of the furniture, which could range from thematic names to generic ones, is stored under 'furName'.

Paired Programming - this assignment may be done Individually or optionally in Pairs

- You may do this assignment either individually, or in a pair. You may select your own partner.
- Information about your pair should be entered into the `Teamwork` class. You must complete information required in the `Teamwork` class, even if you work on your own
- For paired programming to be successful, members must be able to discuss and work together, for suitably long periods of time. You must, therefore, identify time slots in which you can both work together. During your session together, you should vary who actually types in the code and who is observing/advising to ensure an even distribution of work. Screen sharing is a useful tool.
- Paired programming works best if the skill levels of the pair are roughly equivalent. If there is a wide disparity between skill levels, there is a high probability that the weaker of the pair will not learn anything, while the stronger will do all of the work. This will not prepare the weaker partner for future assessments.
- Partners do not have to attend the same practicals, as practical sessions should NOT be used to complete this assignment (but you may use them to get to get help).
- Partners must EACH submit the same project file to Studynet by the deadline

The Assignment

A BlueJ project `cwk3-students` is provided. Amend this BlueJ project so that it implement a version of the FIRE system by completing tasks described below. .

- Your FIRE project should display the qualities associated with good program design:
  - Your system should minimise code duplication and be modularised so that components have low coupling and high cohesion.

- You should aim to make your code reusable and easy to maintain.
- Program code should be well documented, displaying agreed standards of internal documentation.
- Lower level classes should not contain input/output statements

- Marks shown for each task include marks for both functionality and design.

- **Your project must compile**. Code which does not compile may be included as comments
- You must attend the **online timed demo session**. **If you do not attend, you will get ZERO marks.**

Assignment Tasks

Marks awarded for tasks below total 120. Your mark will be then converted to a %

Task 1 - Implement and test the FurnitureBase Class 20 marks

1. FurnitureBase Class Description: FurnitureBase is an abstract superclass that offers a generic template for various furniture categories like SofaItem and BedroomCollection. It encompasses attributes and methods that are common to all furniture kinds.

Attributes:

1. `id` (String): A unique identifier for each furniture.
2. `furnitureCategory`: Specifies the category of the furniture, like "sofa", "bed", etc.
3. `itemCount` : Notes the number of units of a particular furniture category.
4. `price`: Captures the price of the furniture item.
5. `furnitureInventory`: Holds a catalog of distinct furniture items.

Methods:

1. `getID()`: Returns the ID of the furniture.
2. `calculateDiscount()`: An abstract function designated to work out and present the discount for a given furniture unit. The precise way it works is fleshed out in the derived classes.
3. `getFurnitureCategory()`: read the category of furniture.
4. `setFurnitureCategory(String category)`: Determines the furniture category.
5. `displayDetails()`: Showcases a text representation of the object, focusing primarily on the procurement date.
6. `associatePurchaser(Purchaser buyer)`: Binds a buyer to a specific furniture unit.
7. `showAllFurnitureDetails()`: Presents a list of all furniture kinds along with their comprehensive details.
8. a method `toString()` which returns a `String` representation of an object of that class including , furniture details

This is a basic list of methods and you may need to add some more if required

Constructors:

1. A default constructor.
2. A constructor with parameters to set the `furnitureCategory` and `itemCount` (and other related attributes if required)

2. FurnitureBaseTester Class

Description:
The `FurnitureBaseTester` class is designed to validate and demonstrate the functionality of the `FurnitureBase` class. By:

- creating different `FurnitureBase` objects
- call their methods to ensures that the core furniture functionality is intact and behaves as anticipated.
- show that actual results are as expected

Note : you may add other methods if you consider it necessary.

Task 2 - Implementing other classes 16 marks

Implement the following two classes

There is NO need to write tester classes for these classes, but do try Bench Testing to check they work.

1- Purchaser Class

Description:
The `Purchaser` class encapsulates details about individuals or entities that purchase furniture.

Attributes:

- `id` (String): A unique identifier for each purchaser.
- `name` (String): Name of the purchaser.
- `furnitureType` (String): Type of furniture the purchaser bought.
- `contactDetails` (String): Contact information (could be email, phone number, or address).
- `purchaseDate` (Date): The date on which the furniture was purchased.
- `purchaseHistory` (List-Array): A list containing the history of all furniture items bought by the purchaser.

Methods:

- `getName()`: Returns the name of the purchaser.
- `getID()`: Returns the ID of the purchaser.
- `getContactDetails()`: Returns the contact information of the purchaser.
- `setContactDetails(String contact)`: Updates the contact information of the purchaser.
- `addToPurchaseHistory(String item)`: Adds a furniture item to the purchaser's purchase history.
- `getPurchaseHistory()`: Provides a list of all furniture items previously bought by the purchaser.
- `isPurchaseValid(int THRESHOLD_VALUE)`: This method takes an integer as a parameter and returns a Boolean indicating if the purchaser is eligible to buy that particular item. The decision might be based on the quantity of previous purchases. For instance, if a purchaser has already bought a certain number of items (THRESHOLD_VALUE), they might be restricted from making more purchases or, conversely, they might get exclusive access to buy certain premium items.

Constructor:

- A constructor with parameters to initialize `id`, `name`, and `furnitureType`.
- An overloaded constructor that initializes all attributes including `contactDetails` and `purchaseDate`.

2. FurnitureCategory Class

Description: This class characterizes distinct furniture categories and holds specific details about the suitability and constraints of each type.

Attributes:

- **id (String):** A unique identifier for each Category.
- **typeName (String)**: The name of the furniture type, e.g., "Sofa", "Table", "Chair".

- **maximumLoad (Double)**: Represents the maximum weight (in kilograms) the furniture type can support (. *e.g., 150.5 for Armchair, 50.0 for Coffee Table)*
- **isOutdoor (Boolean)**: Specifies if the furniture type is suitable for outdoor use.
- **recentPurchaser (Purchaser)**: The latest purchaser of this furniture type. It's an instance of the `Purchaser` class.
- **materials** (List/Array): A list of materials the furniture is made of. *e.g., ["Wood", "Leather"] for Armchair, ["Glass", "Metal"] for Coffee Table*

Methods:

- **getID():** Returns the ID of the Category.
- **getTypeName()**: Retrieves the category of the furniture.
- **getMaxLoad()**: Returns the maximum load the furniture can support.
- **getRoomRecommendation()**: Indicates the recommended room for this furniture.
- **addMaterial(String material)**: Adds a new material to the materials list.
- **isSuitableForOutdoor()**: Returns a Boolean indicating if the furniture type is suitable for outdoor placement based on its properties. The condition is based on two factors: whether the furniture is explicitly labeled as suitable for outdoor use (`isOutdoor` attribute) and if it can support more than 50 kilograms. This might be a way to ensure that lightweight furniture doesn't get easily damaged or carried away by strong winds, for example.

Constructor:

- A constructor that takes arguments to set the initial values for the attributes.

Without these classes you will find it difficult to develop the remaining classes. You may add further fields/methods to these classes if you require them during further development

---

Task 3 - Interface 20 marks

1- Interface: FURNITUREInterface: The FURNITURE interface lays out essential methods vital for the full functionality of the furniture management system. A detailed documentation of this interface is available within the "Cwk3-students" repository. Please note that you should refrain from making modifications to this interface.

A few key points to note about this interface:

- It's crucial not to modify the method signatures within the FURNITURE interface. However, exceptions might be made during hands-on demonstrations.

2- class `FurnitureStore` - a skeleton of this class is provided in the folder "`Cwk3-students`".

- It is set up to implement the interface FURNITUREInterface. Please, do not alter its header.
- We have initiated the foundation of this class by copying the methods indicated in the FURNITURE interface, presenting code "stubs" for you to expand upon.
- Declare and initiate three collections to manage references to: all furniture pieces, all suppliers & all customer orders.

- The collection of all furniture pieces should be an ArrayList, with positions of each piece in the ArrayList corresponding to its ID number, i.e., Furniture piece 0 (BASE) should be at location 0.
- The collections of customer orders (of `Purchaser` objects) & suppliers (of `FurnitureCategory`

- objects) should also be ArrayLists, but there is no predetermined order of the objects.
- The class should feature a constructor that:
    - Accepts the name of the store as a parameter.
    - Sets the store's location from the provided parameter values.
- Following this, the constructor should invoke two private methods, detailed at the end of the class:
    - loadFurnitureAndSuppliers - This method is responsible for creating all the furniture pieces detailed in Appendix A and subsequently adding them to the furniture collection. These pieces should be added based on their ID numbers. It will then generate all suppliers, linking furniture objects to their respective suppliers and appending them to the supplier collection.
    - loadOrders - This function should generate all customer orders and incorporate them into the order collection.
- Concluding the constructor's operations, it should link specific customer orders to their corresponding furniture pieces.
- The FurnitureStore class should then provide implementations for every method outlined in the FURNITURE interface.
- If there are any supplementary methods you deem essential to enhance the system's design (and are not already specified in the interface), they must be declared as private to ensure proper encapsulation.
- [Incorporating 3 private methods to retrieve each object type (furniture piece, order, supplier) might be beneficial.]

Task 4 - Implementing an application 5 marks

- **Class FurnitureStoreUI**

  Provides a command line user interface to facilitate the operations related to furniture management, purchases, and categorization.

  The application will compile (but may not function completely) even if the full implementation for `FurnitureStore` hasn't been written yet.

  • Some basic code has been provided, but you need to extend it to cater to other menu options.

  • Use the `Scanner` class to capture input from the user.

  • This UI class should strictly call only methods specified in `FURNITUREInterface`.

  Only this class and the `Tester` in Task 6 should use `System.out.println`

Task 5 - System Testing 15 marks

we have provided a skeleton `Tester` class which:

- declares a variable of `FurnitureStore` class.
- already has a main() to make it runnable (similar to `FurnitureUI` main()) -no need to do anything here
- has a `doTest()` method in which to **write code to call methods** on the `FMS` variable in a way which tests your system and demonstrates it works according to specification. You should include appropriate output to the terminal window. You must also use comments to explain what is being tested.

Marks for this task will be awarded for:

- the appropriate choice of data,
- the sequencing of method calls
- appropriate outputs
- explanations of tests.

We are looking for evidence of a systematic approach to testing and will expect you to show that you have identified and tested for the main events likely to occur when the system is running.

Task 6 - System Documentation 4 marks

- a visually neat and readable UML-style class diagram of your system within the BlueJ project
- program code should be well documented, displaying both agreed standards of internal documentation and good use of the facilities available in `Javadoc`. (`FURNITUREInterface` are already documented)
- Tester class should be well documented (see above)

Task 7 – Inheritance (Challenge Task) 15 marks

1- OfficeFurniture Class

Description:
A subclass of `FurnitureBase`, tailored for office-related furniture like desks, chairs, and filing cabinets.

Attributes:

- `furnitureCategory (String)`: Specifies the category (e.g., Desk, Chair, Cabinet).
- `isErgonomic (Boolean)`: Indicates if the furniture is ergonomic.
- `material (String)`: Specifies the material used (e.g., Wood, Metal, Plastic).
- `price (Double)`: Price of the office furniture.

Methods:

- `getCategory()`: Returns the furniture category.
- `isErgonomicFurniture()`: Returns true if the furniture is ergonomic.
- `getMaterial()`: Returns the material of the furniture.
- `getPrice()`: Returns the price of the furniture.
- `adjustPrice()`: Adjusts the price based on the furniture properties.

Description:

The `adjustPrice` method dynamically recalculates the price of an office furniture item based on its specific attributes and characteristics. It is designed to apply premium adjustments for certain desirable features to better align with market expectations and perceived value.

Logic:

1. **Ergonomic Design Adjustment:** Ergonomic furniture, which is specifically designed for comfort and to reduce user fatigue, often comes at a premium price in the market. Recognizing this, if the furniture item is marked as ergonomic (`isErgonomic` attribute is true), a 10% increase is applied to the item's price.

2. **Material-Based Adjustment:** The material used in the furniture also significantly influences its price, mainly due to factors like durability, aesthetic appeal, and cost of raw materials. Therefore:
   - For furniture made of "Wood": Wood, being a natural and often aesthetically pleasing material, tends to be priced higher. As such, a 15% price increase is applied to wooden furniture.
   - For furniture made of "Metal": Metal, while durable, might not have the same premium feel as wood. Thus, a more modest 5% price increase is applied to metal furniture.

Constructors

- Default constructor.
- Overloaded constructors for office furniture initialization.

2. KidsSet Class

Description: A subclass of `FurnitureBase` that contains details about various kids' furniture sets. For instance, if the store sells a variety of kids' furniture sets like cribs, chairs, tables, or themed sets like "Dinosaur Adventure" or "Fairy Tale Castle", each of these could be an instance of the `MyKidsSet` class.

Attributes:

- `kidsId (String)`: Identifier for each kids' set.
  - Example: "SET001" for the "Dinosaur Adventure" set.
- `furName (String)`: Name of the furniture within the set.
  - Example: "Dino Desk" as part of the "Dinosaur Adventure" set.
- `price (Double)`: Price of the kids' set.
  - Example: 299.99 for the entire "Dinosaur Adventure" set.

Methods:

- `getPrice()`: Returns the price of the set.
  - Example: If called on a "Dinosaur Adventure" set object, it returns 299.99.
- `showPrice()`: Displays the price based on the set ID.
  - Example: For ID "SET001", it would display "Price for Dinosaur Adventure set: $299.99".
- `mysteryGift()`: Provides surprise gifts based on price. For example, a price over $200 might come with a free lamp.

  Description:

  The `mysteryGift()` method is designed to incentivize purchases by providing surprise gifts based on the total value of the purchase. It checks the purchase price of a furniture item and based on predefined price thresholds, determines the appropriate gift to offer to the purchaser.

  Logic:

  1. **Price Thresholds and Gifts:**
     - Purchases **above $500**: Receive a "Luxury Office Chair".
     - Purchases **between $300 and $500**: Get a "Desk Organizer Set".
     - Purchases **between $200 and $300**: Come with a "Free Lamp".
     - Purchases **below $200**: Do not receive a mystery gift.
  2. **Special Furniture Set Gifts:** For specific furniture sets, there might be themed gifts:
     - "Dinosaur Adventure" set: "Free Dino Lamp".
     - "Space Explorer" set: "Astronaut Pen Holder".

Outcome:

The method returns a string indicating the mystery gift the purchaser is entitled to based on their purchase. If the purchase doesn't meet the minimum threshold for a gift, a message indicating "No mystery gift for this purchase" is returned.

Usage Scenario:

- If the `mysteryGift()` method is called on a furniture item priced at $250, it would return "Free Lamp".

Task 8 - Demonstration 25 marks

On the day after the assignment hand-in, you will be asked to demonstrate that you have a good working knowledge of the code that you submitted.

You will be asked to
- download to your computer, your original project and the written demo specification posted to Studynet.
- In a timed session of 100 minutes, rename the project (as instructed in the Demo specification)
- make the specified changes to the project on your own computer
- then upload their zipped re-named amended project to the `Cwk3 - Demo` assignment slot.

Note: you do not need to have internet access for the whole session. It will be enough if you download at the beginning of the session, work on your computer during the session and upload the result at the end of the session**. However, you should be aware that uploading at the end may take time so allow at least 5 mins before the end time for uploading**.

The main purpose of the demonstration is to authenticate your code by showing that you know it well enough to use it and make these changes**. If you do not undertake this demonstration, your assignment will get ZERO marks.**

You may be asked to:
- add a specified tester class
- write a demo class to test the functionality of your system
- add a specified subclass
- add a new class, or methods to an existing class
- amend `FurnitureStore` to create a store with small changes that may evolve the store layout or offerings based on a new business model.

Marking & Feedback

Marks awarded on the feedback sheet total 120 and will then be converted to a %.

The submitted assignment will be marked using a combination of automated testing or visual inspection of code or both which will form the basis for individual marks. So do NOT to change method signatures in `FurnitureInterface` or `FurnitureStore` classes

Marks for the demonstration will be for code which performs the specified tasks, not for producing correct output. To see the type of tasks which could be asked, see the mock demo posted for the original CREAM assignment

Your performance in the demonstration will put a limit on the marks you get for the coursework. In addition, a serious mismatch between your submitted work and your performance in the demonstration may result in proceedings for plagiarism/collusion. The limits on coursework marks are as follows:

Demonstration Mark	Maximum % Coursework Mark
mark < 4	15
4 <= mark <= 8	50
8 < mark <= 13	65
mark >13	100

Generic Grading Criteria	no merit	clear fail	marginal fail	satisfactory	good	very good	excellent
%Marks	0 - 19	20 - 29	30 - 39	40 - 59	60 - 69	70 - 79	80 - 100

The %mark will be 22% of the marks for the module

Feedback will be provided on a personalised feedback form posted to Studynet.

// See below for Appendix A – (sample data if needed or you can use your own)

Here's some sample data presented in a table format for different classes and lists :

FurnitureBase

ID	Name	Description	Price
1	Chair	Wooden Chair	$100
2	Table	Dining Table	$300
3	Sofa	Leather Sofa	$800
4	Bed	King Size Bed	$1200
5	Wardrobe	3-door Wooden Wardrobe	$900

Purchaser

ID	Name	Email
1	IKEA	ikea@example.com
2	Walmart	walmart@example.com
3	West Elm	westelm@example.com
4	Target	target@example.com
5	HomeGoods	homegoods@example.com

Orders

Order ID	Furniture Item (by ID)	Purchaser (by ID)	Quantity
101	1 (Chair)	1 (IKEA)	5
102	2 (Table)	2 (Walmart)	3
103	3 (Sofa)	3 (West Elm)	2
104	4 (Bed)	4 (Target)	1
105	5 (Wardrobe)	5 (HomeGoods)	4

.