***UML DİAGRAMS***

**Class Diagram:** A class diagram represents the static structure of the system, showcasing the classes, their attributes, methods, and relationships. Below is the class diagram for the Inventory Management System.

metin, ekran görüntüsü, diyagram, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**Sequence Diagram:** The sequence diagram illustrates the interaction between objects in the system for key processes such as user authentication.

metin, ekran görüntüsü, diyagram, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**Use Case Diagram:** The use case diagram presents the various functionalities available to the user within the system.

metin, çizgi, diyagram, taslak içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**Class Diagrams**

**User Class**

* **Attributes**: username, password
* **Methods**: authenticate(), getUsername(), getPassword()
* **Encapsulation**: username and password are private attributes, ensuring data security.

**InventoryItem Class**

* **Attributes**: name, quantity, cost
* **Methods**: getQuantity(), setQuantity(), getCost(), setCost()
* **Inheritance**: Extends the Record class.
* **Polymorphism**: Overrides displayInfo() to provide specific inventory details.

**Project Class**

* **Attributes**: materials
* **Methods**: addMaterial(), listMaterials(), displayInfo()
* **Implements Interface**: Manageable

**Expense Class**

* **Attributes**: amount, description
* **Methods**: getAmount(), getDescription(), displayInfo()

**Sale Class**

* **Attributes**: item, quantity, price
* **Methods**: getQuantity(), getPrice(), displayInfo()

**Sequence Diagrams**

**Login Process**

1. The user inputs their username and password.
2. Inventory.login() is called.
3. The system fetches user credentials from the database.
4. If authentication is successful, mainMenu() is executed.
5. Otherwise, an error message is displayed.

**Design Justification**

**Encapsulation**

* Sensitive user data (username, password) is kept private.
* Getter and setter methods ensure controlled access to attributes.

**Inheritance**

* The Record class serves as a base for InventoryItem, Project, Expense, and Sale, ensuring code reusability.
* The Person class is extended by User to maintain a structured user hierarchy.

**Polymorphism**

* The displayInfo() method is overridden in multiple classes to present context-specific information dynamically.

**Abstraction**

* Record is defined as an abstract class, enforcing the implementation of displayInfo() in all derived classes.

**Class Diagram Creation Code**

@startuml

package com.beyza.gokce.inventory {

abstract class Record {

- name: String

+ getName(): String

+ displayInfo(): void

}

class Person {

- username: String

- password: String

+ getUsername(): String

+ getPassword(): String

+ authenticate(password: String): boolean

}

class User {

+ User(username: String, password: String)

}

class InventoryItem {

- quantity: int

- cost: double

+ getQuantity(): int

+ getCost(): double

+ setQuantity(quantity: int): void

+ setCost(cost: double): void

+ displayInfo(): void

}

interface Manageable {

+ addMaterial(item: InventoryItem): void

+ listMaterials(): void

}

class Project {

- materials: List<InventoryItem>

+ Project(name: String)

+ addMaterial(item: InventoryItem): void

+ listMaterials(): void

+ displayInfo(): void

}

class Expense {

- amount: double

- description: String

+ getAmount(): double

+ getDescription(): String

+ displayInfo(): void

}

class Sale {

- quantity: int

- price: double

- item: String

+ getQuantity(): int

+ getPrice(): double

+ displayInfo(): void

}

class Inventory {

+ createTables(): void

+ addUserToDatabase(user: User): void

+ loadUsersFromDatabase(): void

+ addMaterialToDatabase(item: InventoryItem): void

+ loadInventoryFromDatabase(): void

+ addProjectToDatabase(project: Project): void

+ addExpenseToDatabase(expense: Expense): void

+ addSaleToDatabase(sale: Sale): void

}

class InventoryApp {

+ main(args: String[]): void

}

Record <|-- InventoryItem

Record <|-- Project

Record <|-- Expense

Record <|-- Sale

Person <|-- User

Project ..|> Manageable

Inventory ..> User

Inventory ..> InventoryItem

Inventory ..> Project

Inventory ..> Expense

Inventory ..> Sale

}

@enduml

**Sequence Diagram Creation Code**

@startuml

actor User

participant "InventoryApp" as App

participant "Inventory" as Inv

participant "Database" as DB

User -> App: Login()

App -> Inv: authenticate(username, password)

Inv -> DB: SELECT username, password FROM users

DB --> Inv: Return user data

Inv --> App: Authentication result

App --> User: Display login success/fail

@enduml

**Use Case Diagram Creation Code**

@startuml

actor User

rectangle "Inventory Management System" {

usecase "Login" as UC1

usecase "Register" as UC2

usecase "Add Material" as UC3

usecase "View Inventory" as UC4

usecase "Track Projects" as UC5

usecase "Log Expenses" as UC6

usecase "Manage Sales" as UC7

}

User -> UC1

User -> UC2

User -> UC3

User -> UC4

User -> UC5

User -> UC6

User -> UC7

@enduml