SC2002 Hospital Management System

SCS6 Group 1

Table of contents

01 Basic Functionalities

Basic Functionalities

User Functionalities
- Login
- Change Password

Patient Functionalities
- View Available Appointment Slots
- Schedule Appointment
- Cancel Appointment

Doctor Functionalities
- Set Availability for Appointments
- Accept/Decline Appointment Request
- Record Appointment Outcome

Pharmacist Functionalities
- View Appointment Outcome Record
- Update Prescription Status
- Submit Replenishment request

Administrator Functionalities
- View Appointment Details
- View and Manage Medication Inventory
- Approve Replenishment Request

02

Additional Features

Additional Features

Billing

- Developed a billing function to manage patient charges, including consultations and medications.
- Ensures accurate cost calculations and generates detailed invoices.
- Provides a payment option and integrates seamlessly with other HMS modules.

Password Encryption

- Implemented password encryption for secure storage and protection of sensitive data.
- Prevents unauthorized access to CSV files.
- Added an admin function for secure password retrieval and management.

03

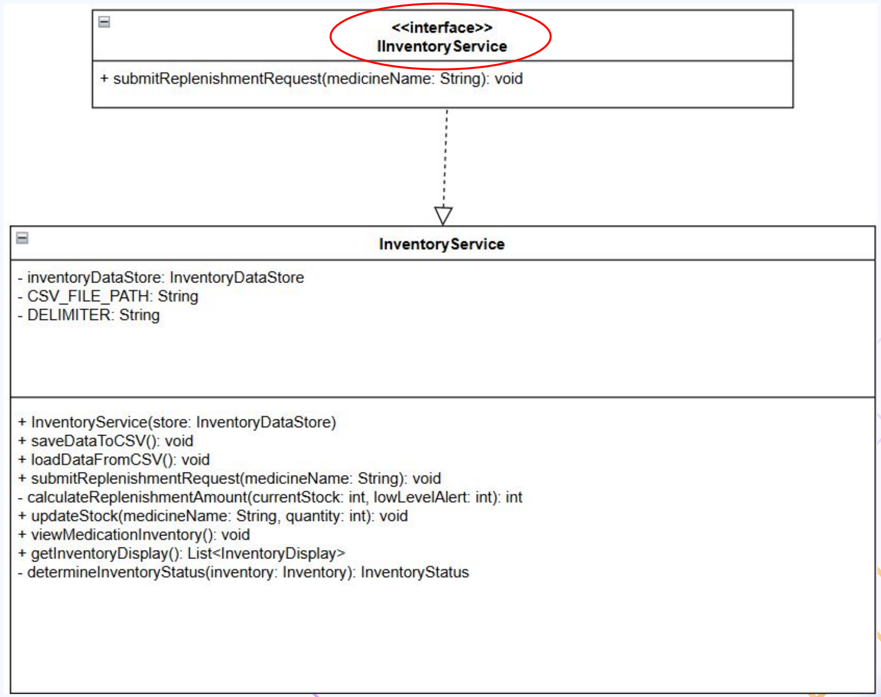
Object Oriented Concepts

Abstraction

- **Definition:** Simplifying complex systems by exposing only essential features and hiding unnecessary details.

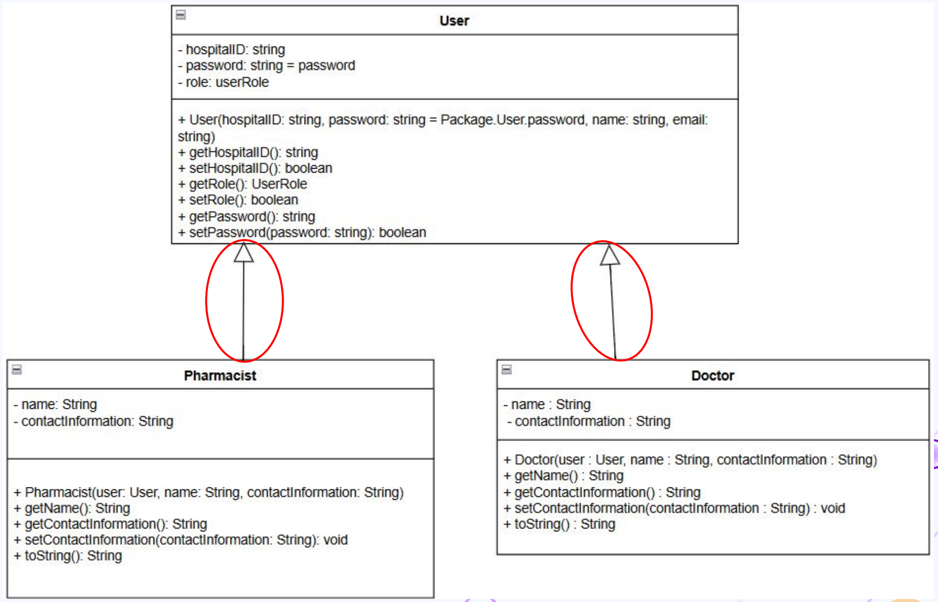
- **Application in HMS:**
  - **Interfaces:** Used to abstract away the implementation details of Services and Views.
  - **Service Example:** InventoryService uses IInventoryService to interact with various IInventoryService implementations, enhancing flexibility and modularity.

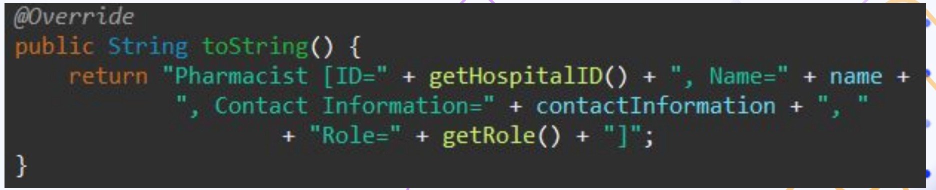
- **Benefits:**
  - Reduces system complexity.
  - Makes the system modular, enabling easier maintenance and future changes.


<<interface>>
IInventoryService
+ submitReplenishmentRequest(medicineName: String): void
InventoryService
- inventoryDataStore: InventoryDataStore
- CSV_FILE_PATH: String
- DELIMITER: String
+ InventoryService(store: InventoryDataStore)
+ saveDataToCSV(): void
+ loadDataFromCSV(): void
+ submitReplenishmentRequest(medicineName: String): void
- calculateReplenishmentAmount(currentStock: int, lowLevelAlert: int): int
+ updateStock(medicineName: String, quantity: int): void
+ viewMedicationInventory(): void
+ getInventoryDisplay(): List<InventoryDisplay>
- determineInventoryStatus(inventory: Inventory): InventoryStatus

Inheritance

- **Definition:** Creating hierarchical relationships between classes, where child classes inherit properties and behaviors of parent classes.
- **Application in HMS:**
  - **User Class Hierarchy**: `Patient`, `Doctor`, `Pharmacist` and `Administrator` inherit from the `User` class
  - **Method Overriding**: Different subclasses of `User` override methods like `toString()`
- **Benefits:**
  - Promotes code reuse, reducing redundancy.
  - Establishes clear hierarchical relationships.


User
- hospitalID: string
- password: string = password
- role: userRole
+ User(hospitalID: string, password: string = Package.User.password, name: string, email: string)
+ getHospitalID(): string
+ setHospitalID(): boolean
+ getRole(): UserRole
+ setRole(): boolean
+ getPassword(): string
+ setPassword(password: string): boolean
Pharmacist
- name: String
- contactInformation: String
+ Pharmacist(user: User, name: String, contactInformation: String)
+ getName(): String
+ getContactInformation(): String
+ setContactInformation(contactInformation: String): void
+ toString(): String
Doctor
- name : String
- contactInformation : String
+ Doctor(user : User, name : String, contactInformation : String)
+ getName() : String
+ getContactInformation() : String
+ setContactInformation(contactInformation : String) : void
+ toString() : String



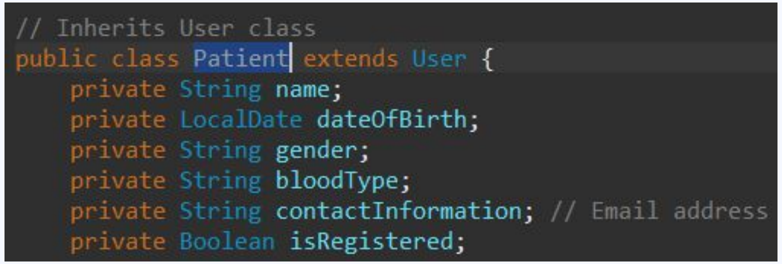
```
@Override
public String toString() {
    return "Pharmacist [ID=" + getHospitalID() + ", Name=" + name +
            ", Contact Information=" + contactInformation + ", "
                    + "Role=" + getRole() + "]";
}
```

Encapsulation

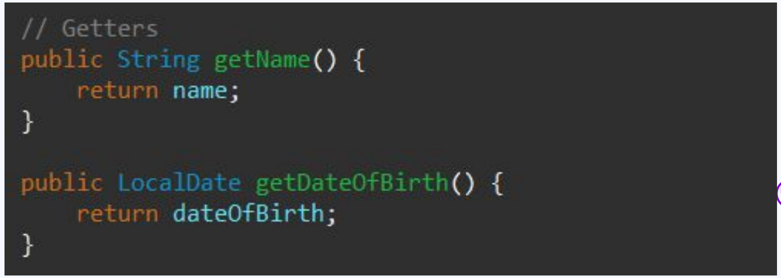
- **Definition:** Protecting the internal state of objects and controlling access through public methods.

- **Application in HMS:**
  - Attributes in classes are made **private**.
  - Access and modification of data occur via **public getters and setters**.
  - Example: Private attributes in the `Patient` class can only be accessed through the **getter methods**

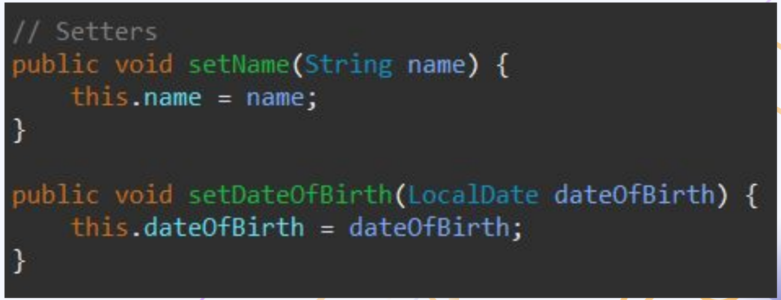
- **Benefits:**
  - Enhances security by preventing unwanted access to data.
  - Provides control over how data is accessed and modified.



```java
// Inherits User class
public class Patient extends User {
    private String name;
    private LocalDate dateOfBirth;
    private String gender;
    private String bloodType;
    private String contactInformation; // Email address
    private Boolean isRegistered;
```



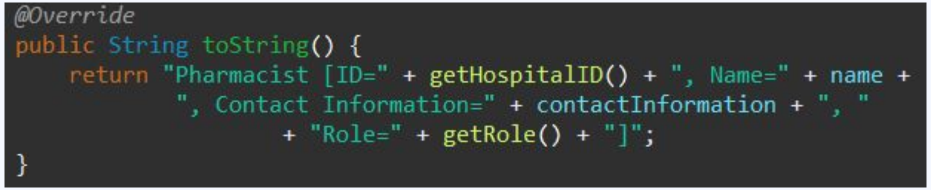
```java
// Getters
public String getName() {
    return name;
}

public LocalDate getDateOfBirth() {
    return dateOfBirth;
}
```



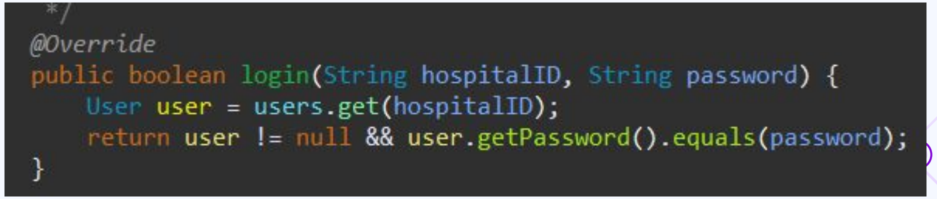
```java
// Setters
public void setName(String name) {
    this.name = name;
}

public void setDateOfBirth(LocalDate dateOfBirth) {
    this.dateOfBirth = dateOfBirth;
}
```

Polymorphism

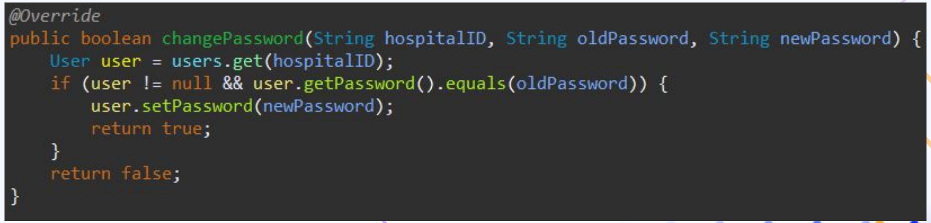
- **Definition:** Allowing objects to take on multiple forms and behaviors.
- **Application in HMS:**
  - **Method Overriding**: Different subclasses of `User` override methods like `toString()`
- **Benefits:**
  - Enables flexibility by allowing different behaviors based on context.
  - Supports extensibility by accommodating new behaviors through subclassing and interface implementations.



```
@Override
public String toString() {
    return "Pharmacist [ID=" + getHospitalID() + ", Name=" + name +
            ", Contact Information=" + contactInformation + ", "
                + "Role=" + getRole() + "]";
}
```



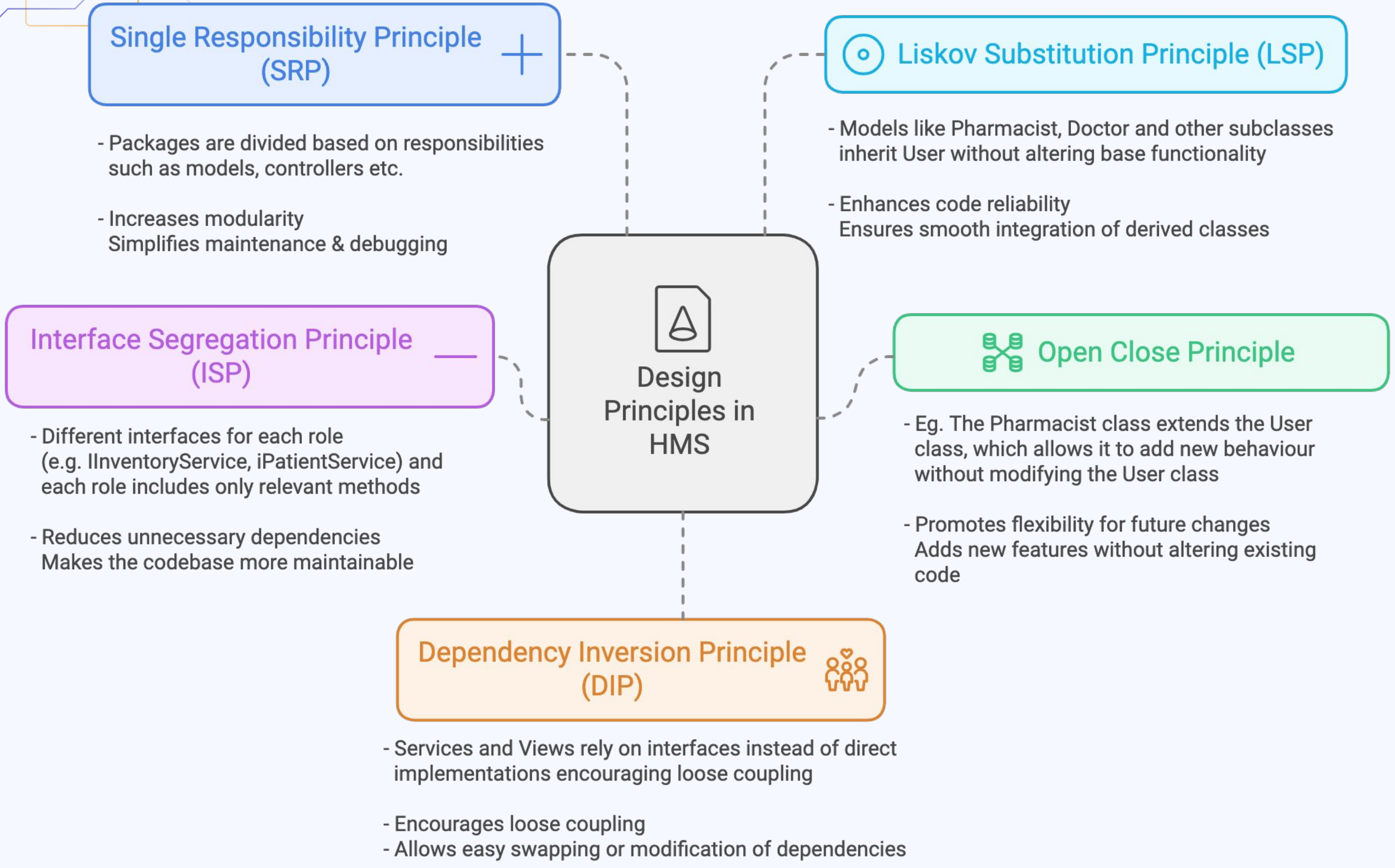
```
  */
 @Override
 public boolean login(String hospitalID, String password) {
     User user = users.get(hospitalID);
     return user != null && user.getPassword().equals(password);
 }
```



```
@Override
public boolean changePassword(String hospitalID, String oldPassword, String newPassword) {
    User user = users.get(hospitalID);
    if (user != null && user.getPassword().equals(oldPassword)) {
        user.setPassword(newPassword);
        return true;
    }
    return false;
}
```

04

Design Principles



Design Principles in HMS

Single Responsibility Principle (SRP)

- Packages are divided based on responsibilities such as models, controllers etc.

- Increases modularity
  Simplifies maintenance & debugging

Liskov Substitution Principle (LSP)

- Models like Pharmacist, Doctor and other subclasses inherit User without altering base functionality

- Enhances code reliability
  Ensures smooth integration of derived classes

Interface Segregation Principle (ISP)

- Different interfaces for each role (e.g. IInventoryService, iPatientService) and each role includes only relevant methods

- Reduces unnecessary dependencies
  Makes the codebase more maintainable

Open Close Principle

- Eg. The Pharmacist class extends the User class, which allows it to add new behaviour without modifying the User class

- Promotes flexibility for future changes
  Adds new features without altering existing code

Dependency Inversion Principle (DIP)

- Services and Views rely on interfaces instead of direct implementations encouraging loose coupling

- Encourages loose coupling
- Allows easy swapping or modification of dependencies