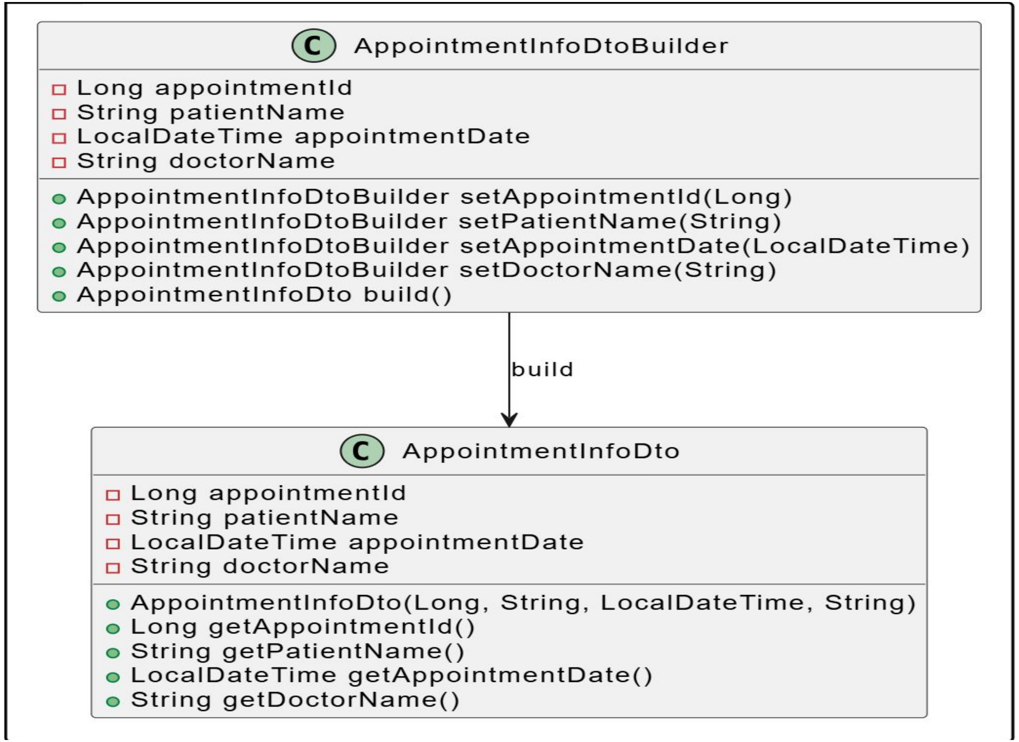
Software Design Pattern

Easy Care

Video link:

https://youtu.be/mAN4JOzny9c?si=0iX3lEWiPUvs9QgT

Builder Design Pattern



C AppointmentInfoDtoBuilder

□ Long appointmentId
□ String patientName
□ LocalDateTime appointmentDate
□ String doctorName

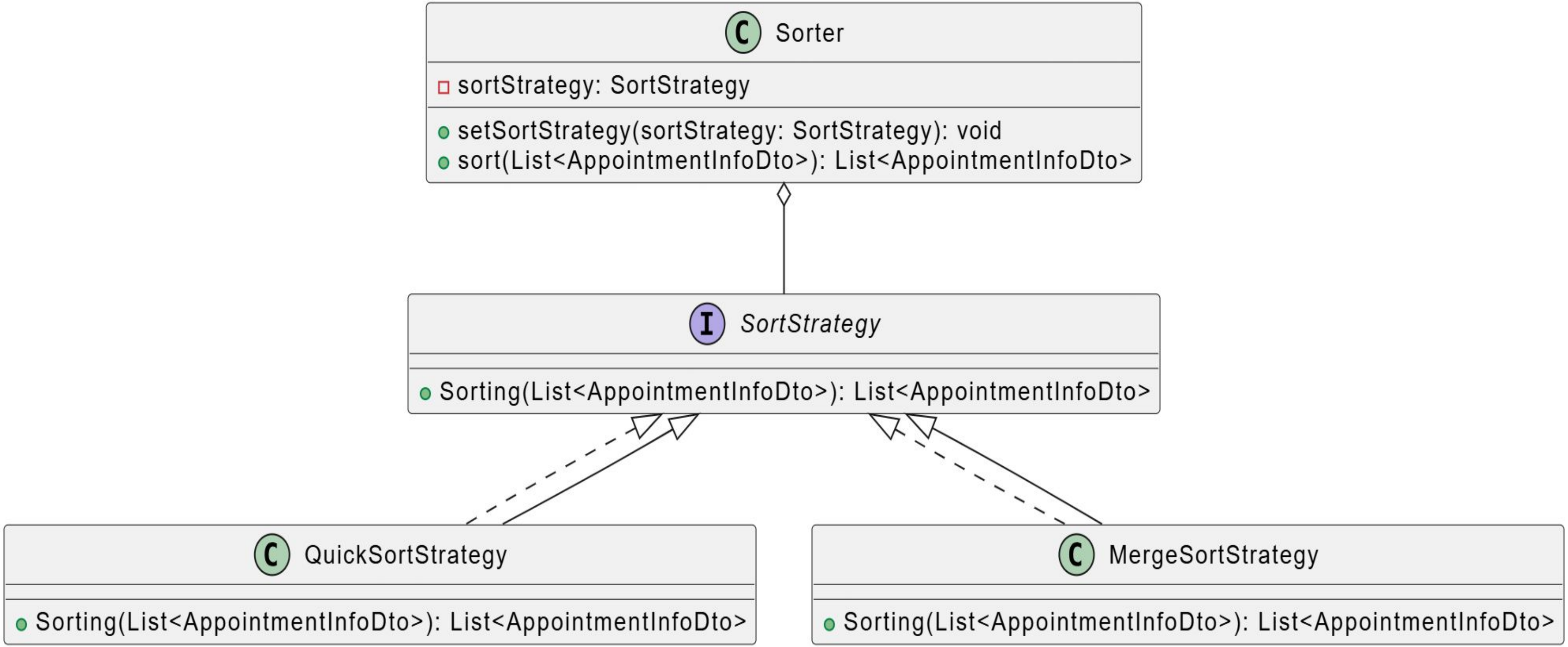
● AppointmentInfoDtoBuilder setAppointmentId(Long)
● AppointmentInfoDtoBuilder setPatientName(String)
● AppointmentInfoDtoBuilder setAppointmentDate(LocalDateTime)
● AppointmentInfoDtoBuilder setDoctorName(String)
● AppointmentInfoDto build()

C AppointmentInfoDto

□ Long appointmentId
□ String patientName
□ LocalDateTime appointmentDate
□ String doctorName

● AppointmentInfoDto(Long, String, LocalDateTime, String)
● Long getAppointmentId()
● String getPatientName()
● LocalDateTime getAppointmentDate()
● String getDoctorName()

Strategy Design Pattern



C Sorter

□ sortStrategy: SortStrategy

● setSortStrategy(sortStrategy: SortStrategy): void
● sort(List<AppointmentInfoDto>): List<AppointmentInfoDto>

I *SortStrategy*

● Sorting(List<AppointmentInfoDto>): List<AppointmentInfoDto>

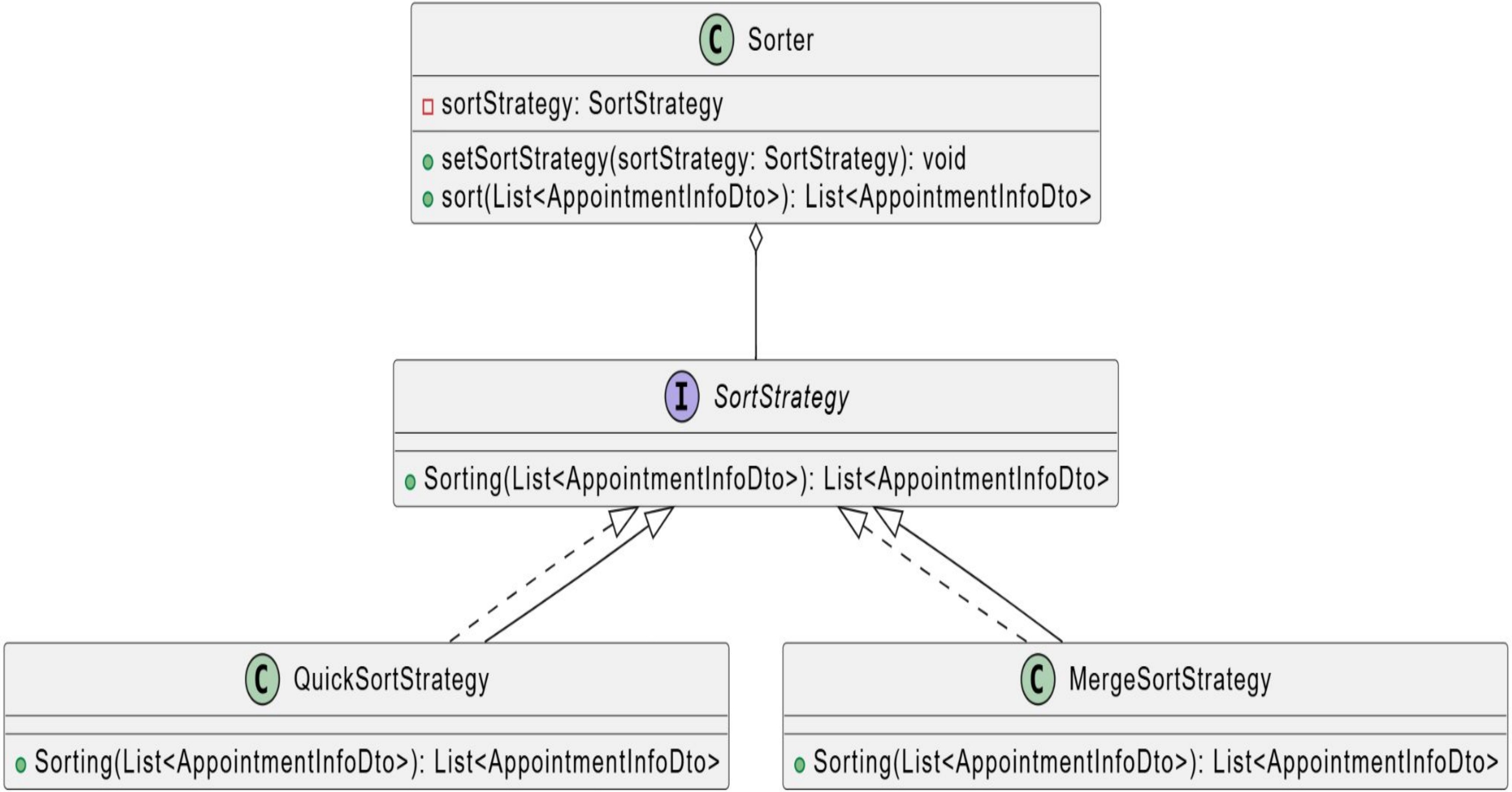
C QuickSortStrategy

● Sorting(List<AppointmentInfoDto>): List<AppointmentInfoDto>

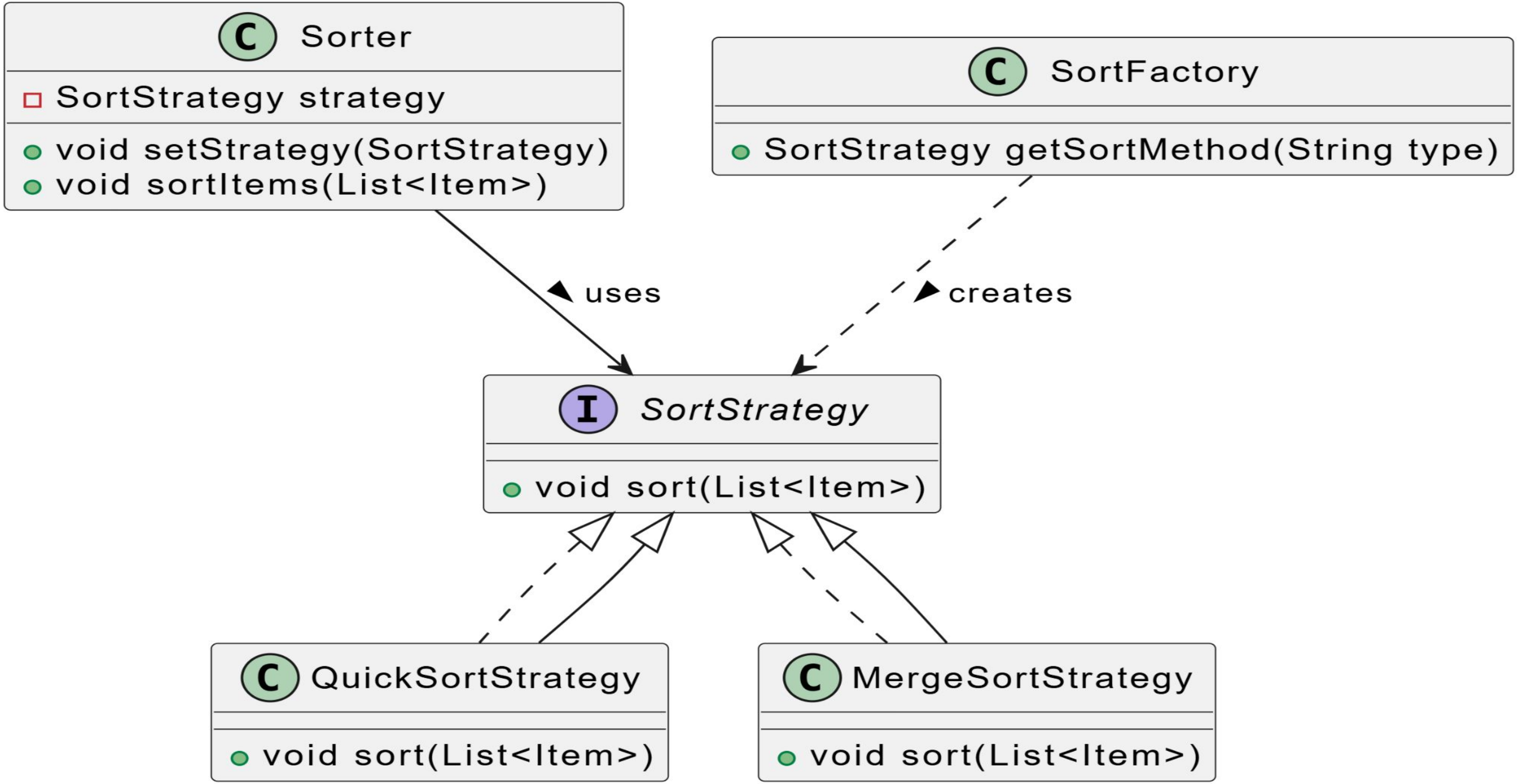
C MergeSortStrategy

● Sorting(List<AppointmentInfoDto>): List<AppointmentInfoDto>

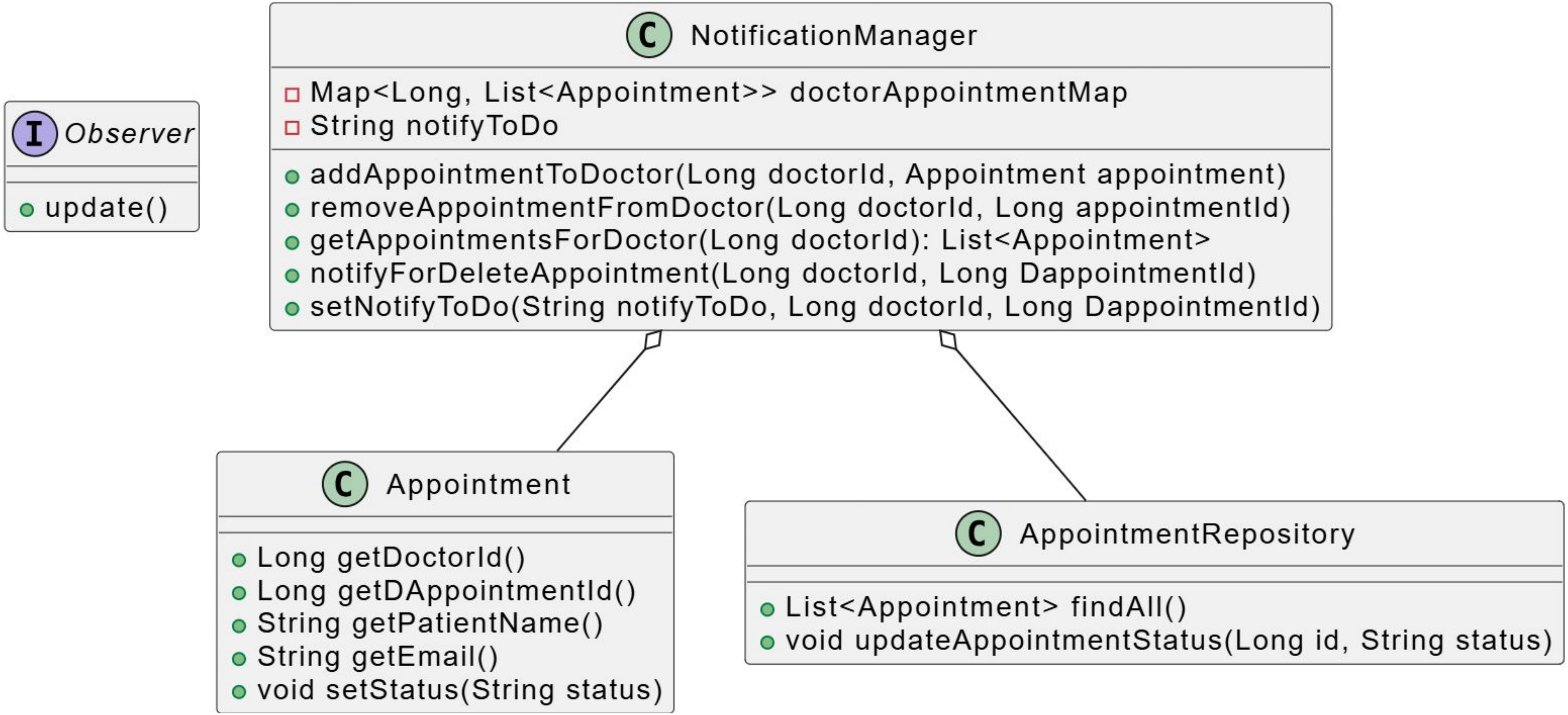
Strategy Design Pattern


C Sorter
sortStrategy: SortStrategy
setSortStrategy(sortStrategy: SortStrategy): void
sort(List<AppointmentInfoDto>): List<AppointmentInfoDto>
I SortStrategy
Sorting(List<AppointmentInfoDto>): List<AppointmentInfoDto>
C QuickSortStrategy
Sorting(List<AppointmentInfoDto>): List<AppointmentInfoDto>
C MergeSortStrategy
Sorting(List<AppointmentInfoDto>): List<AppointmentInfoDto>

Singleton , Factory Design Pattern



Observer Design Pattern(delete appointment)



Observer

● update()

NotificationManager

□ Map<Long, List<Appointment>> doctorAppointmentMap
□ String notifyToDo

● addAppointmentToDoctor(Long doctorId, Appointment appointment)
● removeAppointmentFromDoctor(Long doctorId, Long appointmentId)
● getAppointmentsForDoctor(Long doctorId): List<Appointment>
● notifyForDeleteAppointment(Long doctorId, Long DappointmentId)
● setNotifyToDo(String notifyToDo, Long doctorId, Long DappointmentId)

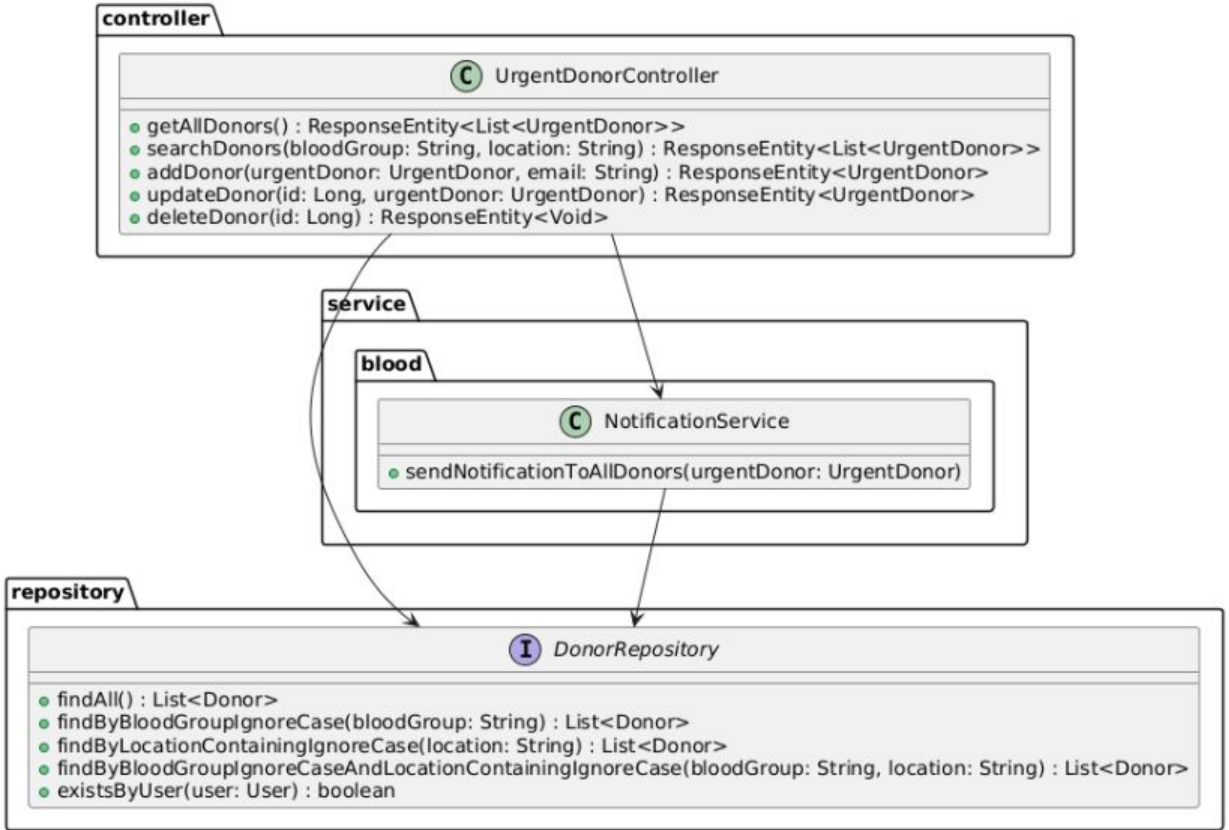
Appointment

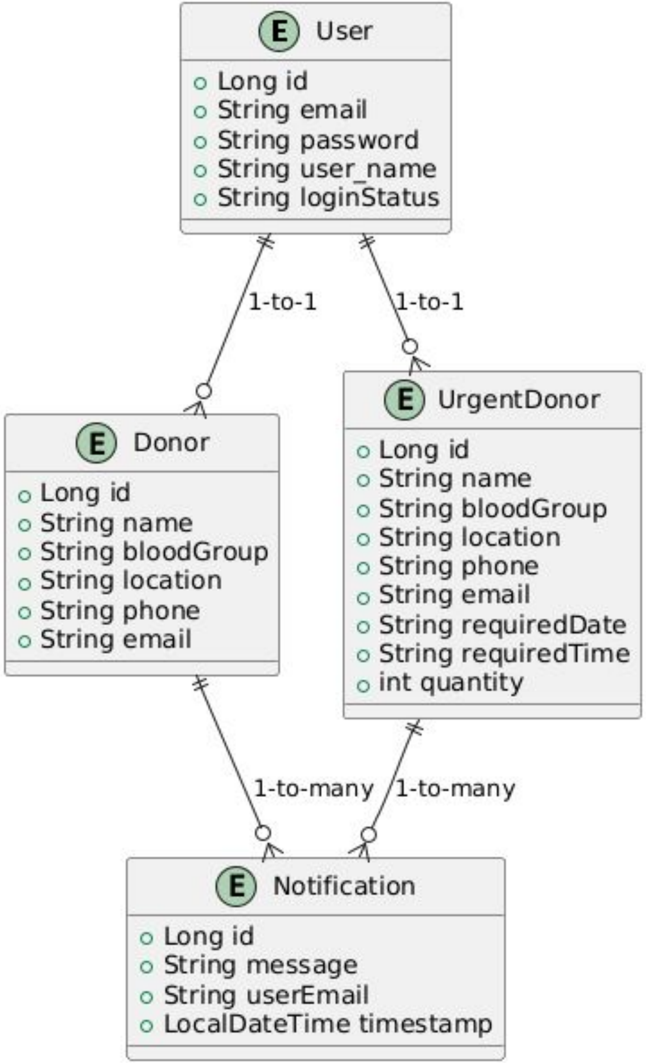
● Long getDoctorId()
● Long getDAppointmentId()
● String getPatientName()
● String getEmail()
● void setStatus(String status)

AppointmentRepository

● List<Appointment> findAll()
● void updateAppointmentStatus(Long id, String status)

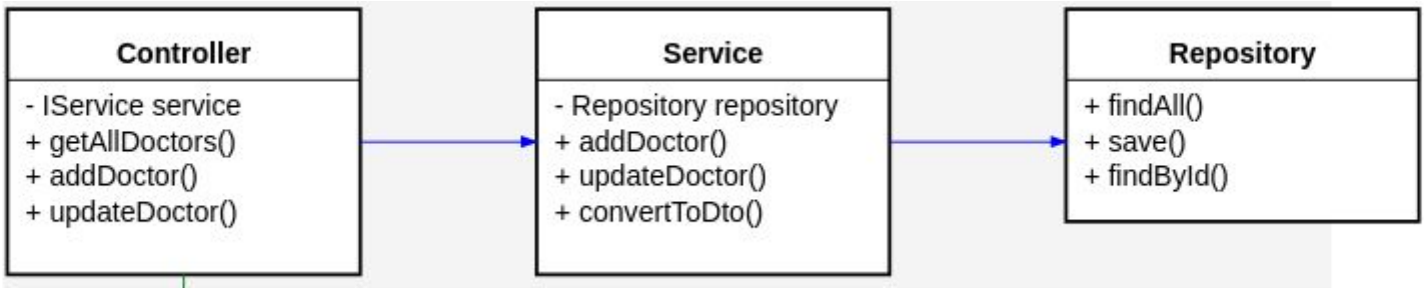
Observer Design Pattern


controller
C UrgentDonorController
getAllDonors() : ResponseEntity<List<UrgentDonor>>
searchDonors(bloodGroup: String, location: String) : ResponseEntity<List<UrgentDonor>>
addDonor(urgentDonor: UrgentDonor, email: String) : ResponseEntity<UrgentDonor>
updateDonor(id: Long, urgentDonor: UrgentDonor) : ResponseEntity<UrgentDonor>
deleteDonor(id: Long) : ResponseEntity<Void>
service
blood
C NotificationService
sendNotificationToAllDonors(urgentDonor: UrgentDonor)
repository
I DonorRepository
findAll() : List<Donor>
findByBloodGroupIgnoreCase(bloodGroup: String) : List<Donor>
findByLocationContainingIgnoreCase(location: String) : List<Donor>
findByBloodGroupIgnoreCaseAndLocationContainingIgnoreCase(bloodGroup: String, location: String) : List<Donor>
existsByUser(user: User) : boolean


E User
Long id
String email
String password
String user_name
String loginStatus
1-to-1
1-to-1
E Donor
Long id
String name
String bloodGroup
String location
String phone
String email
E UrgentDonor
Long id
String name
String bloodGroup
String location
String phone
String email
String requiredDate
String requiredTime
int quantity
1-to-many
1-to-many
E Notification
Long id
String message
String userEmail
LocalDateTime timestamp

Service Layer Pattern(for doctor)

- Encapsulation of Business Logic
- By isolating the business logic into distinct service classes, it allows for easier reusability and maintainability.



Controller

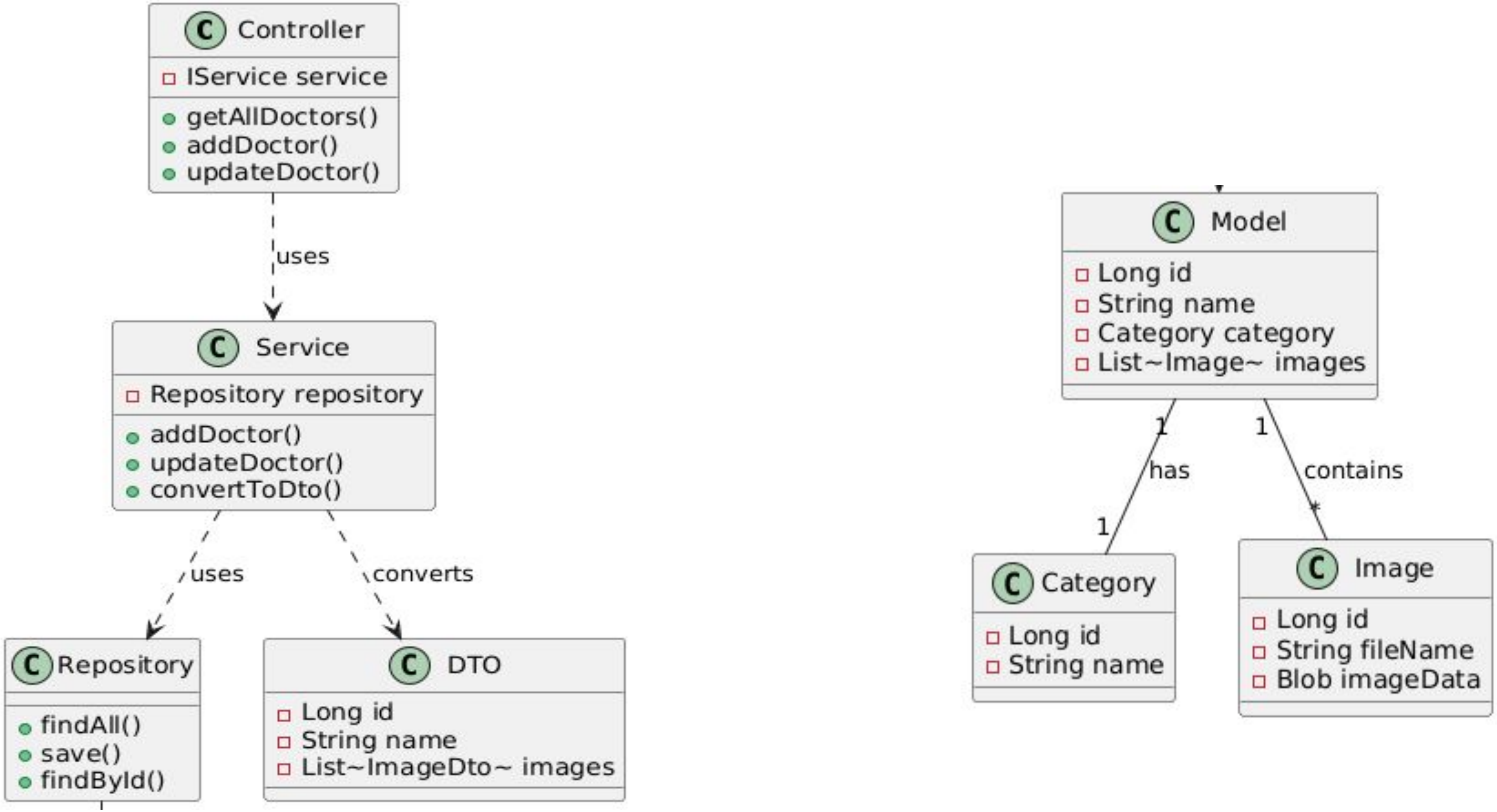
- IService service
+ getAllDoctors()
+ addDoctor()
+ updateDoctor()

Service

- Repository repository
+ addDoctor()
+ updateDoctor()
+ convertToDto()

Repository

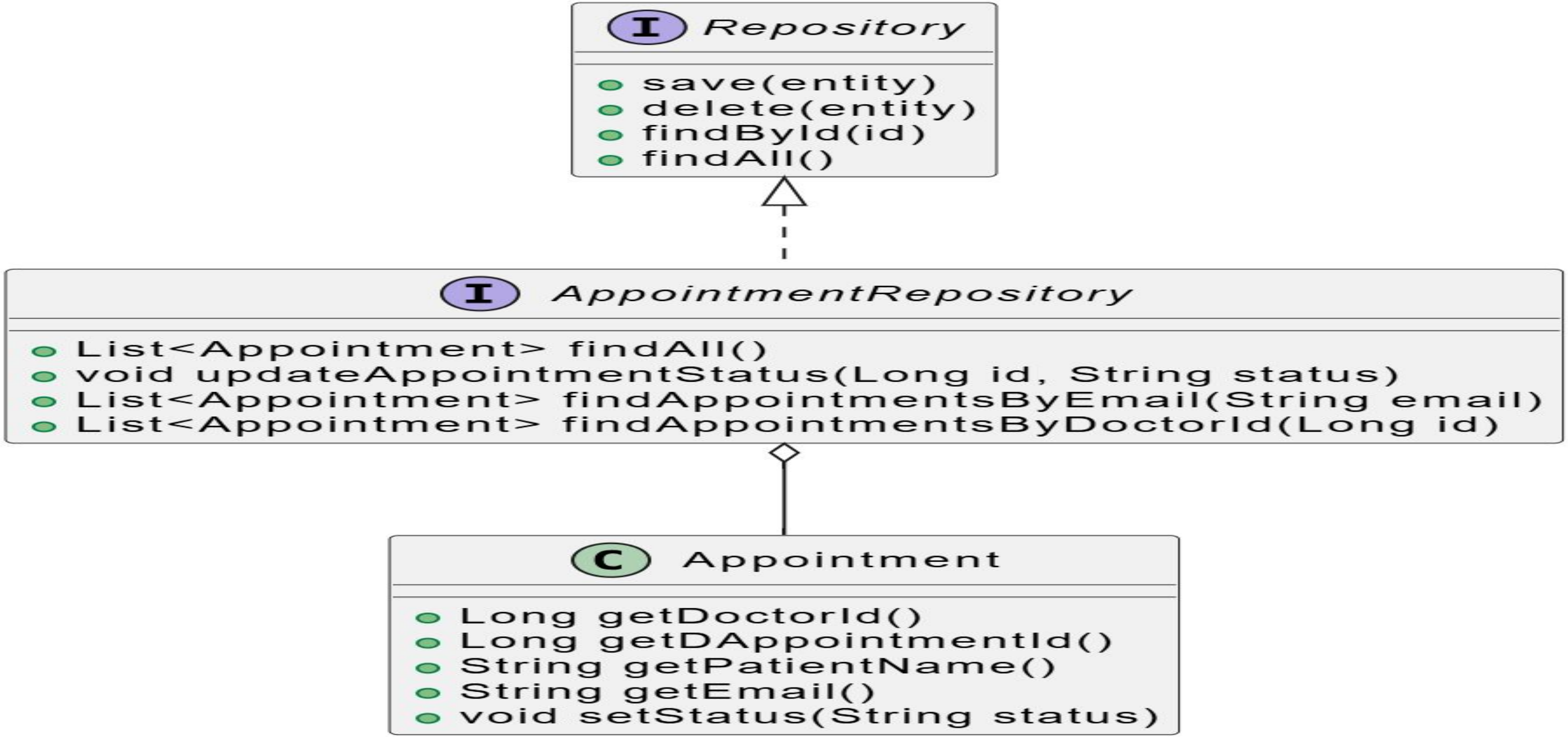
+ findAll()
+ save()
+ findById()


Controller
IService service
getAllDoctors()
addDoctor()
updateDoctor()
uses
Service
Repository repository
addDoctor()
updateDoctor()
convertToDto()
uses
converts
Repository
findAll()
save()
findById()
DTO
Long id
String name
List~ImageDto~ images
Model
Long id
String name
Category category
List~Image~ images
1
has
1
1
contains
*
Category
Long id
String name
Image
Long id
String fileName
Blob imageData

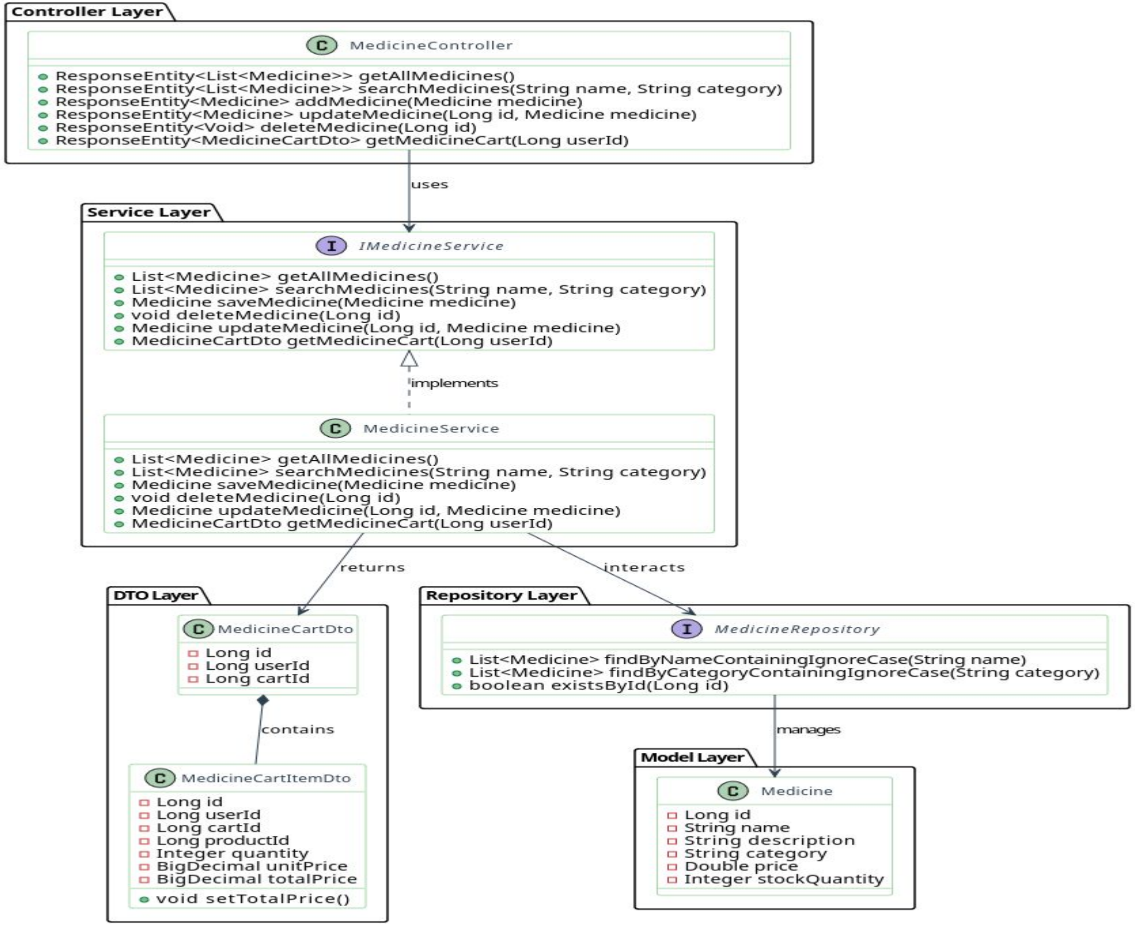
Repository Design Pattern

- The Repository Design Pattern provides an abstraction over the data access layer.
- It separates business logic from database logic.
- Offers a clean API for querying and modifying data.

UML diagram for Repository Design Pattern


I Repository
save(entity)
delete(entity)
findById(id)
findAll()
I AppointmentRepository
List<Appointment> findAll()
void updateAppointmentStatus(Long id, String status)
List<Appointment> findAppointmentsByEmail(String email)
List<Appointment> findAppointmentsByDoctorId(Long id)
C Appointment
Long getDoctorId()
Long getDAppointmentId()
String getPatientName()
String getEmail()
void setStatus(String status)

Strategy Design Pattern



Controller Layer

C MedicineController

- ResponseEntity<List<Medicine>> getAllMedicines()
- ResponseEntity<List<Medicine>> searchMedicines(String name, String category)
- ResponseEntity<Medicine> addMedicine(Medicine medicine)
- ResponseEntity<Medicine> updateMedicine(Long id, Medicine medicine)
- ResponseEntity<Void> deleteMedicine(Long id)
- ResponseEntity<MedicineCartDto> getMedicineCart(Long userId)

Service Layer

I *IMedicineService*

- List<Medicine> getAllMedicines()
- List<Medicine> searchMedicines(String name, String category)
- Medicine saveMedicine(Medicine medicine)
- void deleteMedicine(Long id)
- Medicine updateMedicine(Long id, Medicine medicine)
- MedicineCartDto getMedicineCart(Long userId)

C MedicineService

- List<Medicine> getAllMedicines()
- List<Medicine> searchMedicines(String name, String category)
- Medicine saveMedicine(Medicine medicine)
- void deleteMedicine(Long id)
- Medicine updateMedicine(Long id, Medicine medicine)
- MedicineCartDto getMedicineCart(Long userId)

DTO Layer

C MedicineCartDto

- Long id
- Long userId
- Long cartId

C MedicineCartItemDto

- Long id
- Long userId
- Long cartId
- Long productId
- Integer quantity
- BigDecimal unitPrice
- BigDecimal totalPrice
- void setTotalPrice()

Repository Layer

I *MedicineRepository*

- List<Medicine> findByNameContainingIgnoreCase(String name)
- List<Medicine> findByCategoryContainingIgnoreCase(String category)
- boolean existsById(Long id)

Model Layer

C Medicine

- Long id
- String name
- String description
- String category
- Double price
- Integer stockQuantity

Thank You