INFORMATION SYSTEM ANALYSIS AND DESIGN – 2024

PROJECT

BUILD PART OF SPECIFICATION, DESIGN DOCUMENT AND PROGRAMMING

Each student must take part in all activities of development process

- Requirement determination and analysis
- Design overall and detail
- Programming

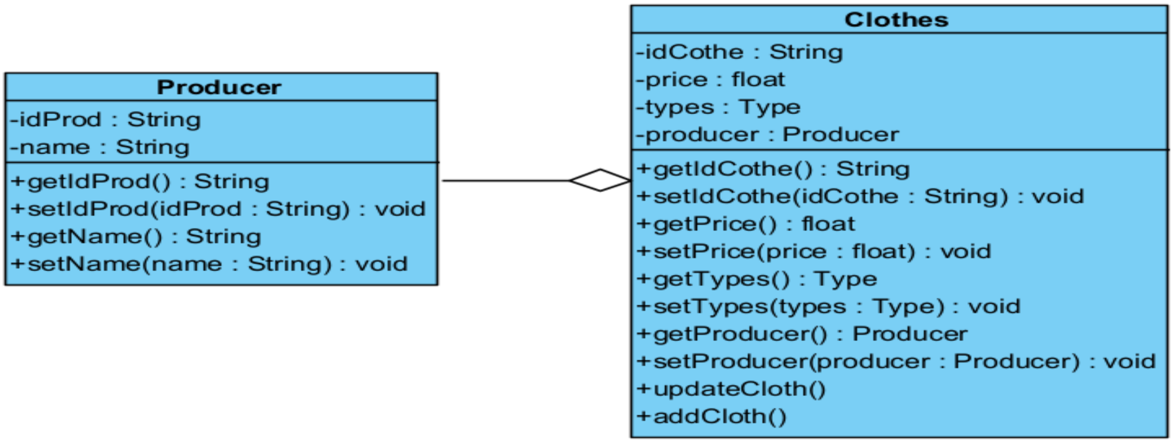
Tasks	Detail tasks	Students	Due date
Req. Deter. & Analysis			**21/10—27/10**
Design			**04/11—10/11**
Programming			**18/11—25/11**

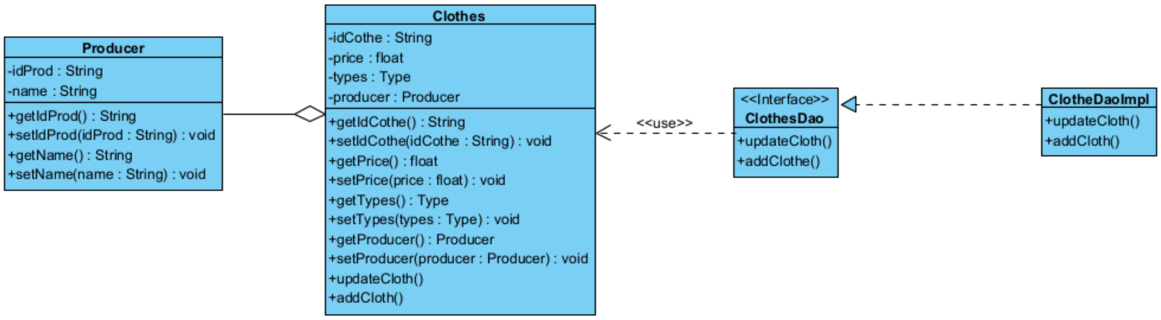
Develop an e-commerce site, which may be users by types of customers, types of staff, managers. This site provides services of purchasing *books, clothes, mobile phone, laptop, shoes, electronic devices*. Your tasks:

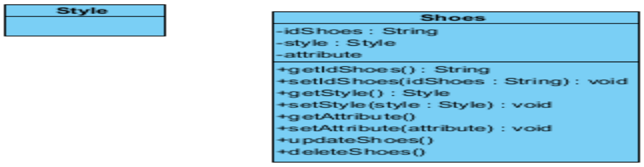
1. Determine requirements
    - Define use cases and draw use case diagram in two levels: General level (>=20 use cases) and Level 1 (detail)
    - Write 30 scenarios and 30 user stories + acceptance criteria
    - Draw 5 activity/swimlane diagrams + 5 sequence diagrams for the processes executed by customer and staff:
        - Create cart, checkout, select ship (drone), select pay and order, give comments & feed back of items
        - Process order: alert order (automatically), check order, send invoice/feedback to customer, send requirements to ship department and repository
        - Process payment
        - Process shipment (using drone)
2. Requirement Analysis
    - Write 50 CRC Cards
    - Draw a class diagram (analysis) with 50 classes
    - Construct data dictionary
    - Create data model
3. Design
    - Construct a class diagram (design)
        - Validate the class diagram in analysis (Name, attributes, methods, relationship...)

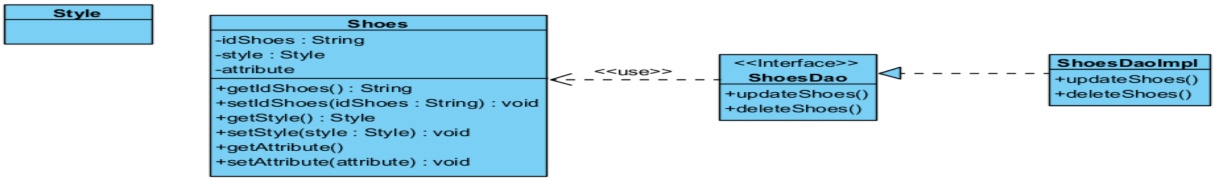
    - Add attributes, add methods, change multiplicity, change relationship
    - Change association classes into suitable relationships (aggregation, composition)
  - Using MVC model to design the e-commerce system
    - Using DAO pattern to classes
    - Using package notion in UML to group classes into packages and layers with the model MVC: Model layer, Controller model, View model
  - Construct database
    - Validate data model: check fields, key, connection multilicity...
    - Select a model for inheritance
    - Generate database with
  - Design interface (user interface and system interface)
  - Design forms such as bill, report....
4. Programming and Integration
  - Using Java Spring (Boot), JSP, CSS, NoteJS....
  - Developing/utilizing techniques in deep learning to build modules search, analyze opinions/comments, recommender items, generate plots of analyzing customer's tendency….

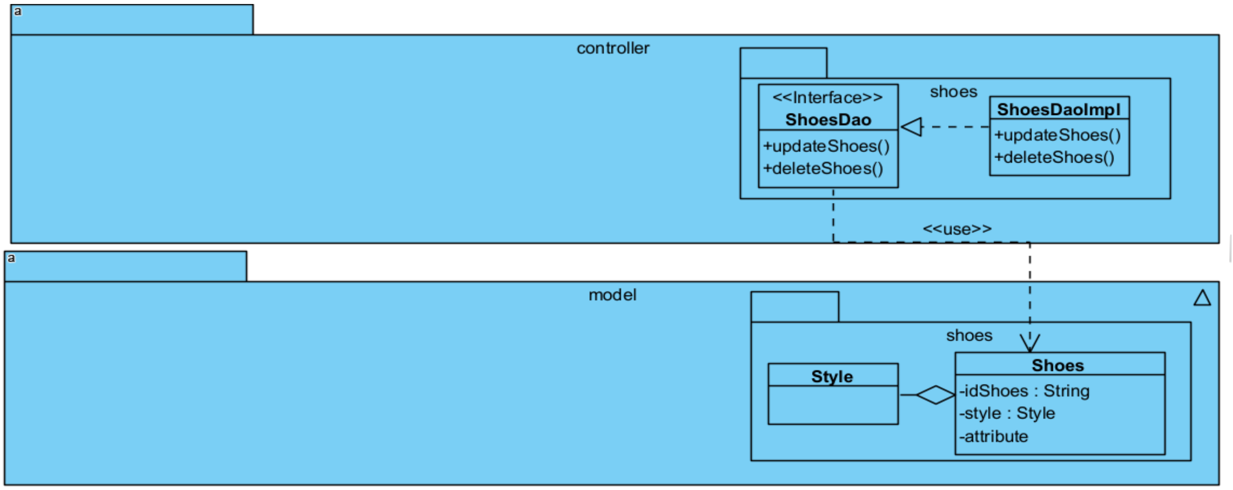
REFERENCES


Producer
-idProd : String
-name : String
+getIdProd() : String
+setIdProd(idProd : String) : void
+getName() : String
+setName(name : String) : void
Clothes
-idCothe : String
-price : float
-types : Type
-producer : Producer
+getIdCothe() : String
+setIdCothe(idCothe : String) : void
+getPrice() : float
+setPrice(price : float) : void
+getTypes() : Type
+setTypes(types : Type) : void
+getProducer() : Producer
+setProducer(producer : Producer) : void
+updateCloth()
+addCloth()


Producer
-idProd : String
-name : String
+getIdProd() : String
+setIdProd(idProd : String) : void
+getName() : String
+setName(name : String) : void
Clothes
-idCothe : String
-price : float
-types : Type
-producer : Producer
+getIdCothe() : String
+setIdCothe(idCothe : String) : void
+getPrice() : float
+setPrice(price : float) : void
+getTypes() : Type
+setTypes(types : Type) : void
+getProducer() : Producer
+setProducer(producer : Producer) : void
+updateCloth()
+addCloth()
<<use>>
<<Interface>>
ClothesDao
+updateCloth()
+addClothe()
ClotheDaoImpl
+updateCloth()
+addCloth()

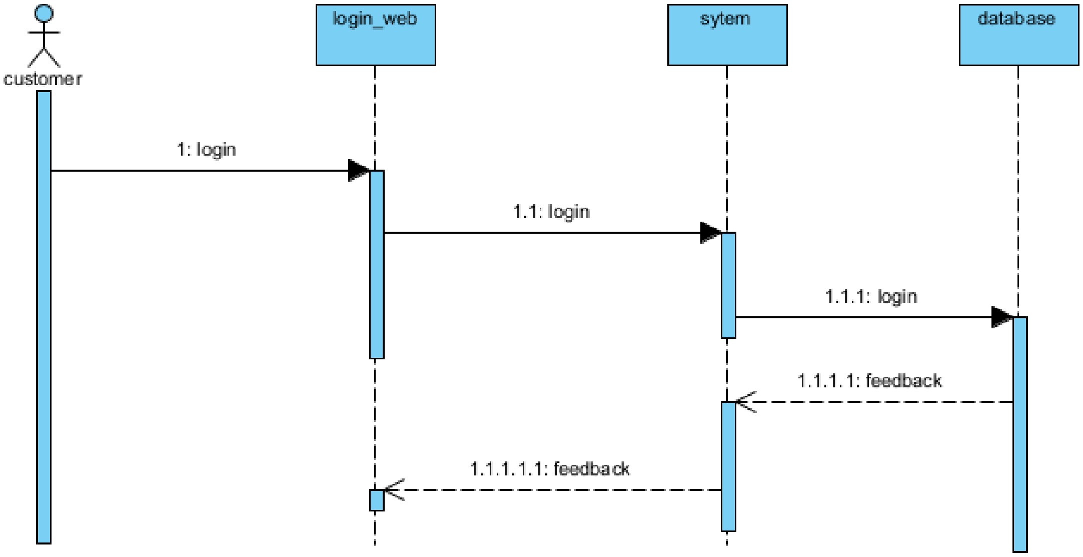

Style
Shoes
-idShoes : String
-style : Style
-attribute
+getIdShoes() : String
+setIdShoes(idShoes : String) : void
+getStyle() : Style
+setStyle(style : Style) : void
+getAttribute()
+setAttribute(attribute) : void
+updateShoes()
+deleteShoes()


Style
Shoes
-idShoes : String
-style : Style
-attribute
+getIdShoes() : String
+setIdShoes(idShoes : String) : void
+getStyle() : Style
+setStyle(style : Style) : void
+getAttribute()
+setAttribute(attribute) : void
<<use>>
<<Interface>>
ShoesDao
+updateShoes()
+deleteShoes()
ShoesDaoImpl
+updateShoes()
+deleteShoes()

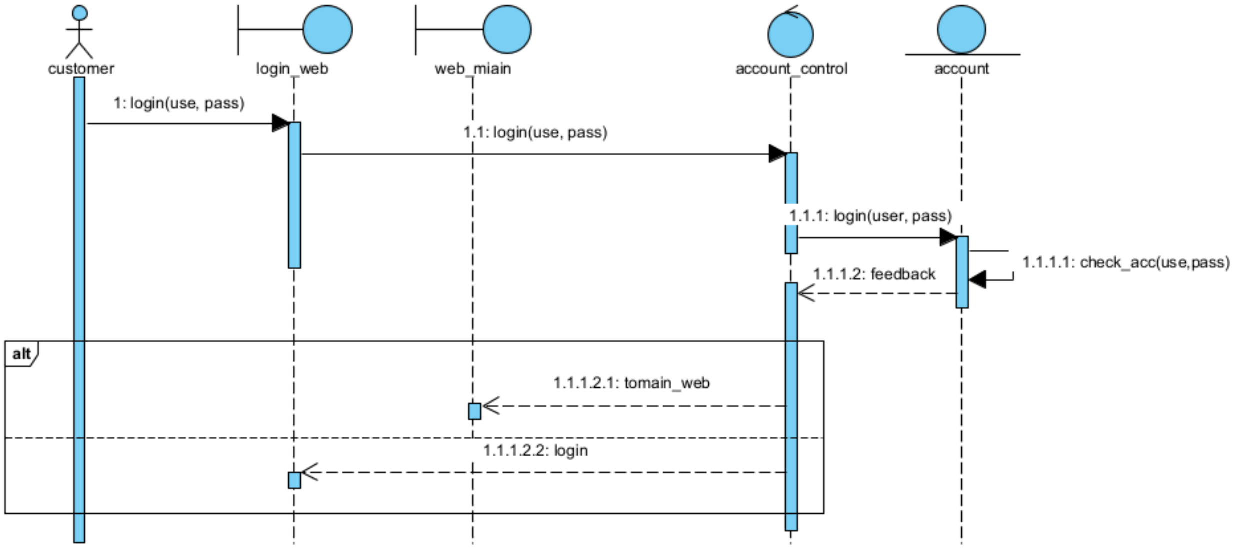

a
controller
shoes
<<Interface>>
ShoesDao
+updateShoes()
+deleteShoes()
ShoesDaoImpl
+updateShoes()
+deleteShoes()
<<use>>
a
model
shoes
Style
Shoes
-idShoes : String
-style : Style
-attribute

SEQUENCE DIAGRAM

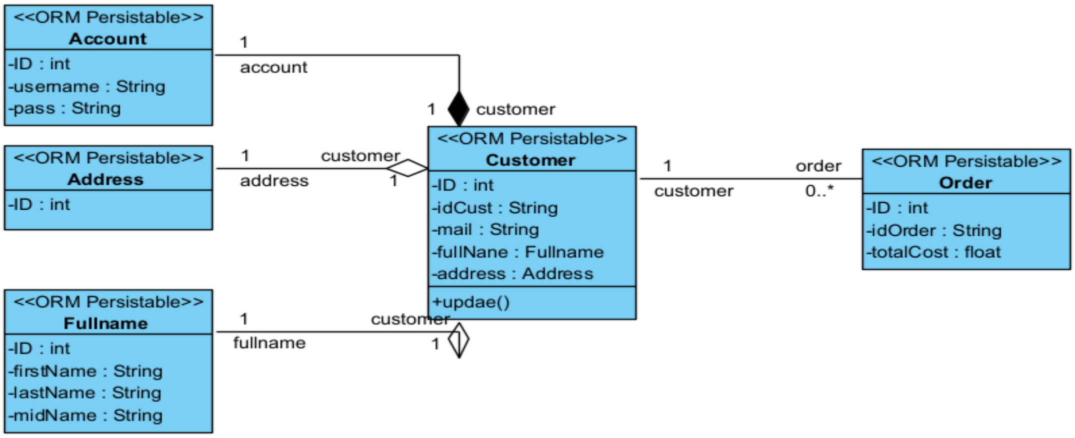
Initial diagram Sequence in the analysis phase


customer
login_web
sytem
database
1: login
1.1: login
1.1.1: login
1.1.1.1: feedback
1.1.1.1.1: feedback

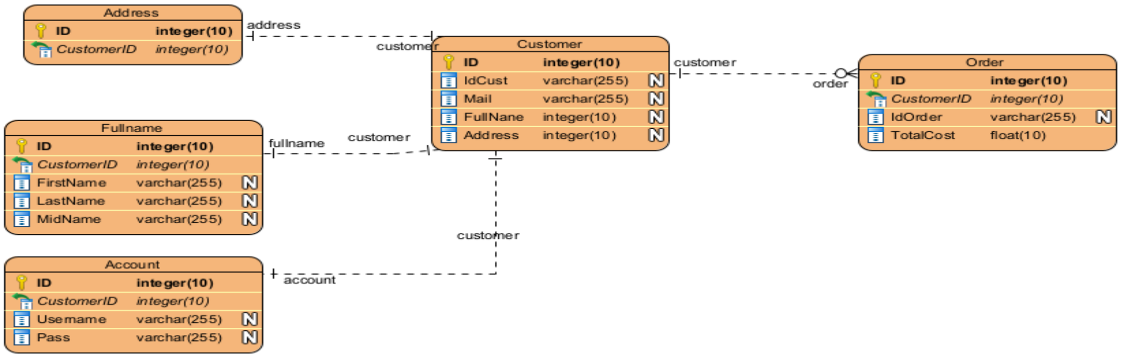
Sequence Diagram in design phase


customer
login_web
web_miain
account_control
account
1: login(use, pass)
1.1: login(use, pass)
1.1.1: login(user, pass)
1.1.1.1: check_acc(use,pass)
1.1.1.2: feedback
alt
1.1.1.2.1: tomain_web
1.1.1.2.2: login

BUILDING DATA MODEL

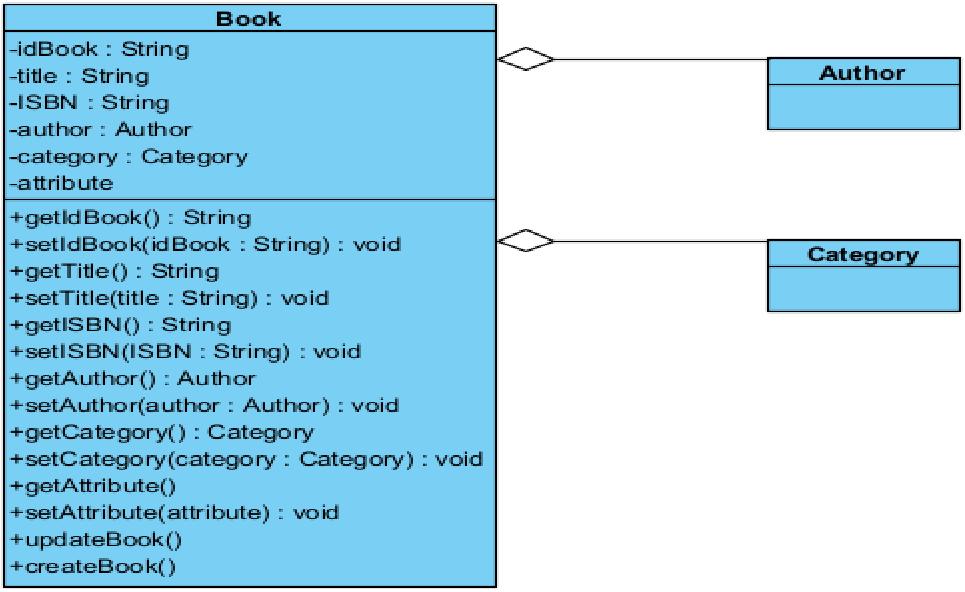

<<ORM Persistable>>
Account
-ID : int
-usemame : String
-pass : String
<<ORM Persistable>>
Address
-ID : int
<<ORM Persistable>>
Fullname
-ID : int
-firstName : String
-lastName : String
-midName : String
<<ORM Persistable>>
Customer
-ID : int
-idCust : String
-mail : String
-fullNane : Fullname
-address : Address
+updae()
<<ORM Persistable>>
Order
-ID : int
-idOrder : String
-totalCost : float
1
account
1
customer
1
customer
address
1
1
customer
fullname
1
1
customer
order
0..*
customer

DATA MODEL

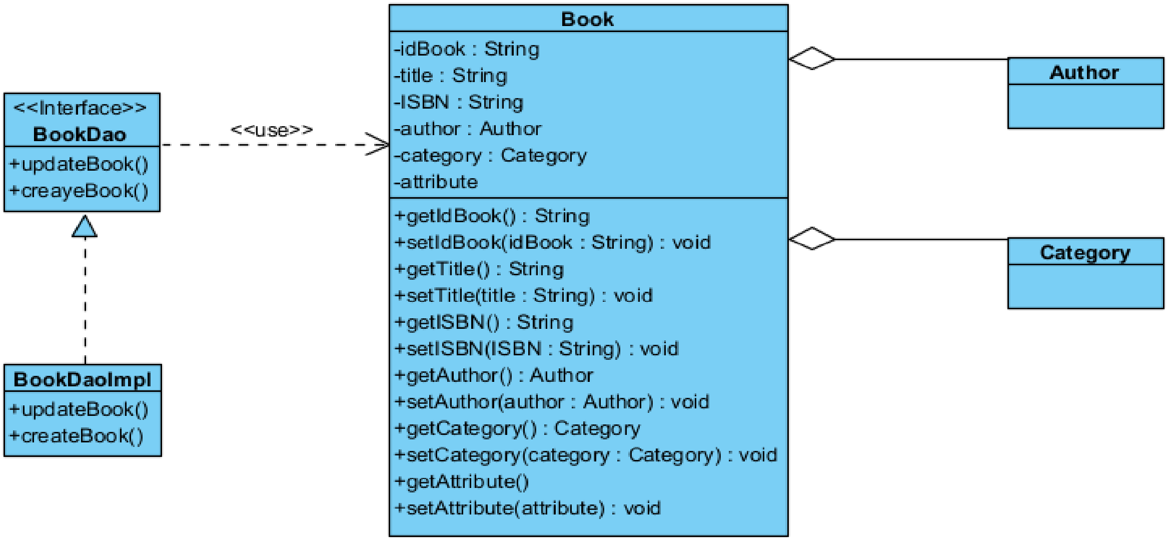

Address
ID integer(10)
CustomerID integer(10)
address
customer
Customer
ID integer(10)
IdCust varchar(255) N
Mail varchar(255) N
FullNane integer(10) N
Address integer(10) N
customer
order
Order
ID integer(10)
CustomerID integer(10)
IdOrder varchar(255) N
TotalCost float(10)
Fullname
ID integer(10)
CustomerID integer(10)
FirstName varchar(255) N
LastName varchar(255) N
MidName varchar(255) N
fullname
customer
customer
Account
ID integer(10)
CustomerID integer(10)
Username varchar(255) N
Pass varchar(255) N
account

DESIGN STEPS

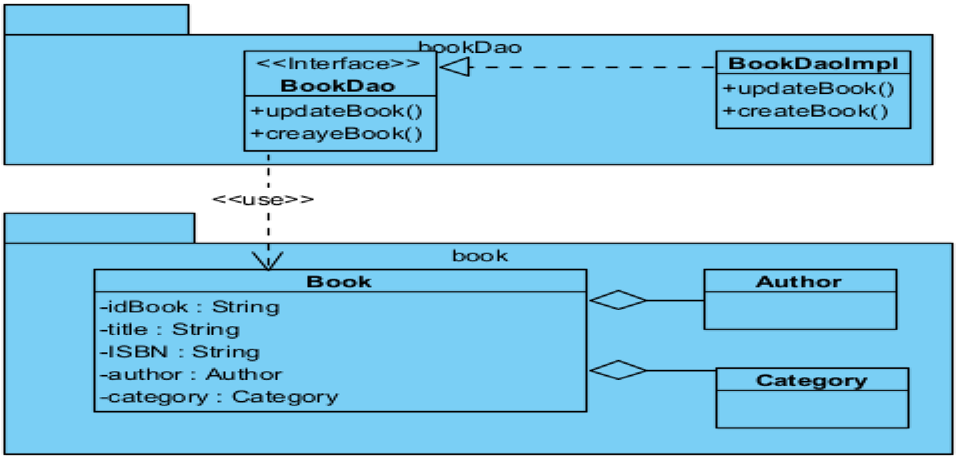
STEP 1: VALIDATE CLASS


Book
-idBook : String
-title : String
-ISBN : String
-author : Author
-category : Category
-attribute
+getIdBook() : String
+setIdBook(idBook : String) : void
+getTitle() : String
+setTitle(title : String) : void
+getISBN() : String
+setISBN(ISBN : String) : void
+getAuthor() : Author
+setAuthor(author : Author) : void
+getCategory() : Category
+setCategory(category : Category) : void
+getAttribute()
+setAttribute(attribute) : void
+updateBook()
+createBook()
Author
Category

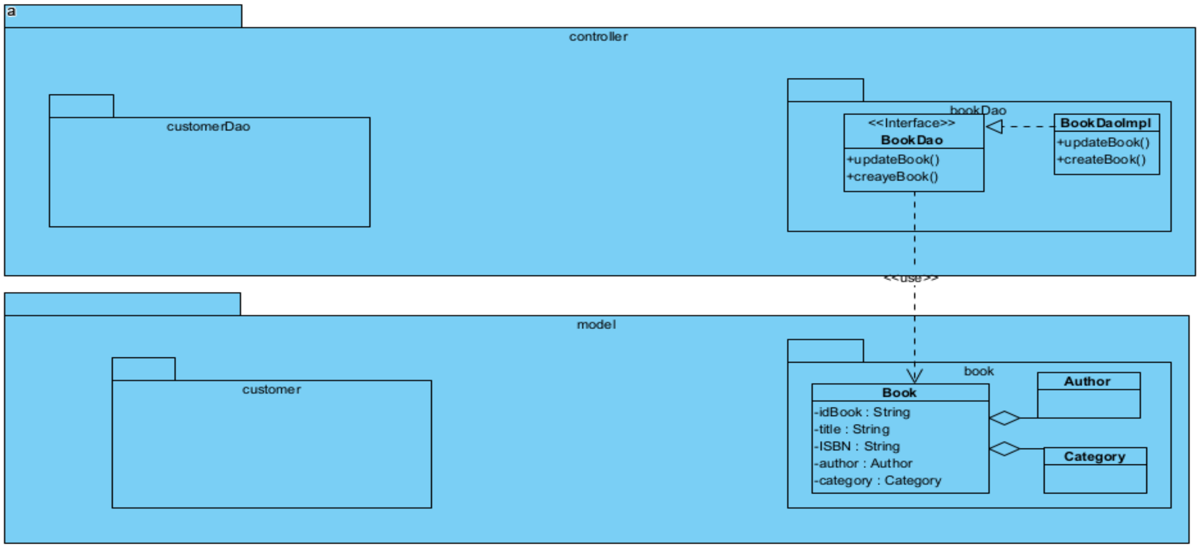
STEP 2 CREATE DAO


Book
-idBook : String
-title : String
-ISBN : String
-author : Author
-category : Category
-attribute
+getIdBook() : String
+setIdBook(idBook : String) : void
+getTitle() : String
+setTitle(title : String) : void
+getISBN() : String
+setISBN(ISBN : String) : void
+getAuthor() : Author
+setAuthor(author : Author) : void
+getCategory() : Category
+setCategory(category : Category) : void
+getAttribute()
+setAttribute(attribute) : void
Author
Category
<<Interface>>
BookDao
+updateBook()
+creayeBook()
<<use>>
BookDaoImpl
+updateBook()
+createBook()

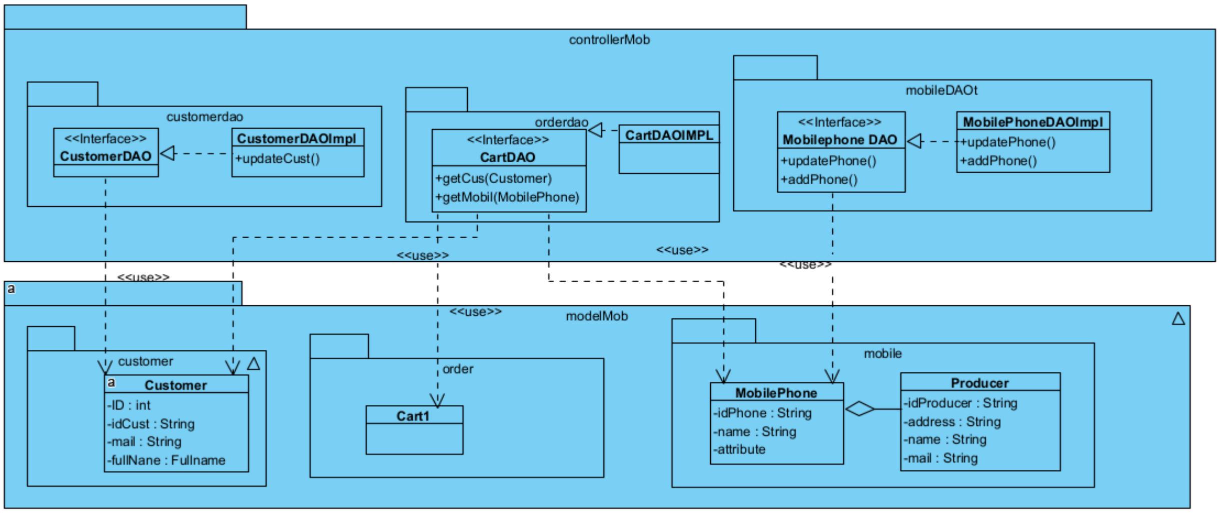
STEP 3: PACKAGE

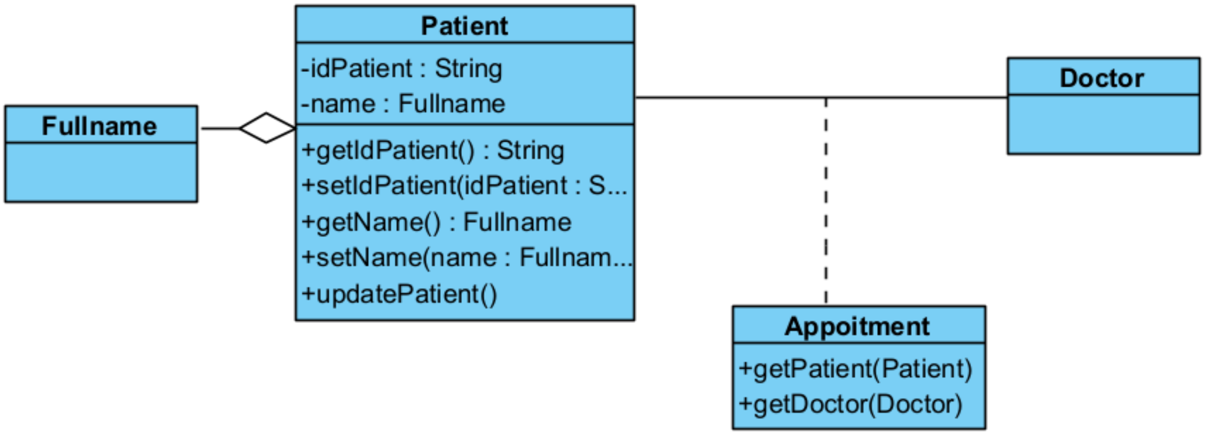

bookDao
<<Interface>>
BookDao
+updateBook()
+creayeBook()
BookDaoImpl
+updateBook()
+createBook()
<<use>>
book
Book
-idBook : String
-title : String
-ISBN : String
-author : Author
-category : Category
Author
Category

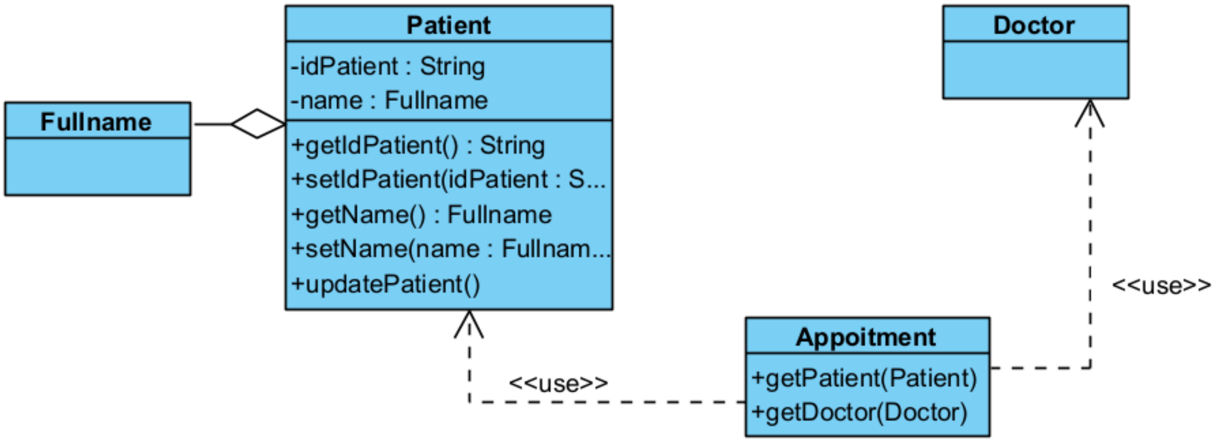
STEP 4: Layer


a
controller
customerDao
bookDao
<<Interface>>
BookDao
+updateBook()
+creayeBook()
BookDaoImpl
+updateBook()
+createBook()
<<use>>
model
customer
book
Book
-idBook : String
-title : String
-ISBN : String
-author : Author
-category : Category
Author
Category

EXAMPLE OF THE LAST DESIGN


controllerMob
customerdao
<<Interface>>
CustomerDAO
CustomerDAOImpl
+updateCust()
orderdao
<<Interface>>
CartDAO
+getCus(Customer)
+getMobil(MobilePhone)
CartDAOIMPL
mobileDAOt
<<Interface>>
Mobilephone DAO
+updatePhone()
+addPhone()
MobilePhoneDAOImpl
+updatePhone()
+addPhone()
<<use>>
<<use>>
<<use>>
<<use>>
<<use>>
a
modelMob
customer
a
Customer
-ID : int
-idCust : String
-mail : String
-fullNane : Fullname
order
Cart1
mobile
MobilePhone
-idPhone : String
-name : String
-attribute
Producer
-idProducer : String
-address : String
-name : String
-mail : String


Patient
-idPatient : String
-name : Fullname
+getIdPatient() : String
+setIdPatient(idPatient : S...
+getName() : Fullname
+setName(name : Fullnam...
+updatePatient()
Fullname
Doctor
Appoitment
+getPatient(Patient)
+getDoctor(Doctor)


Patient
-idPatient : String
-name : Fullname
+getIdPatient() : String
+setIdPatient(idPatient : S...
+getName() : Fullname
+setName(name : Fullnam...
+updatePatient()
Fullname
Doctor
<<use>>
<<use>>
Appoitment
+getPatient(Patient)
+getDoctor(Doctor)