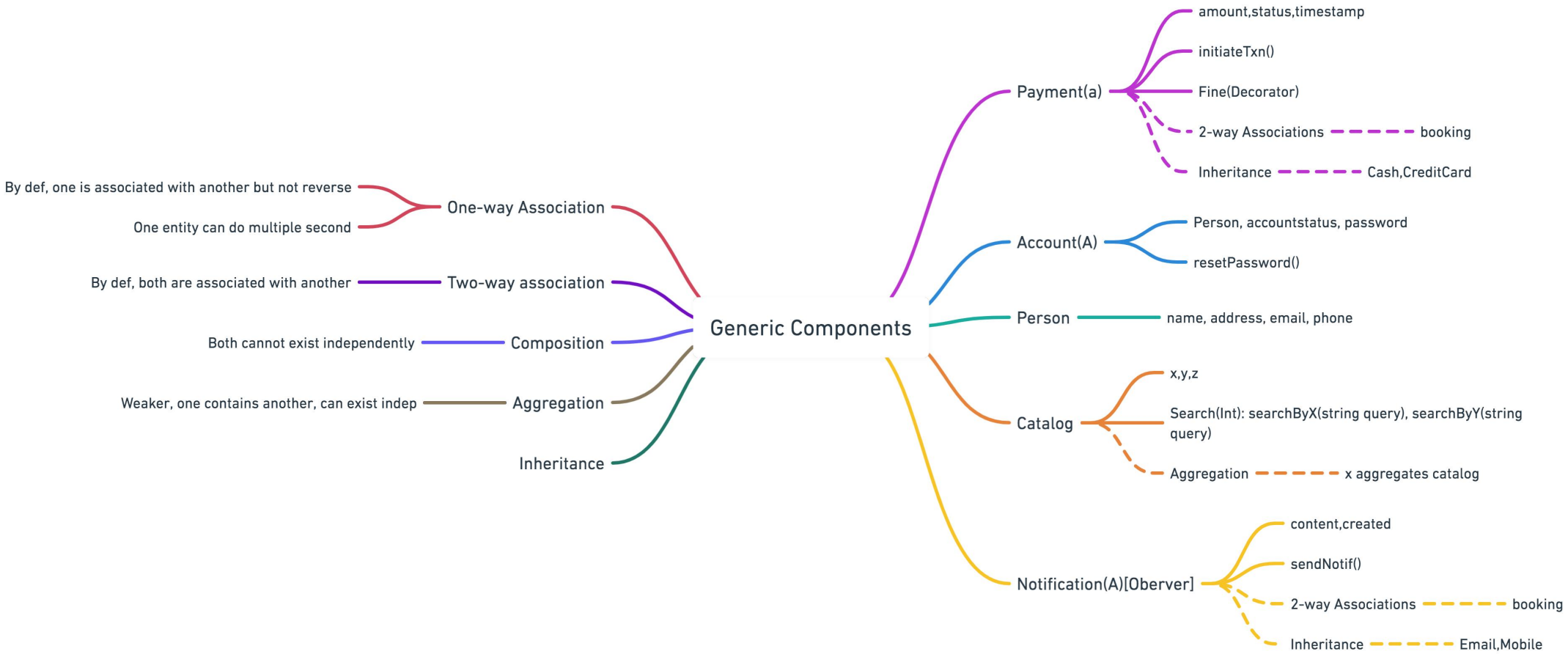
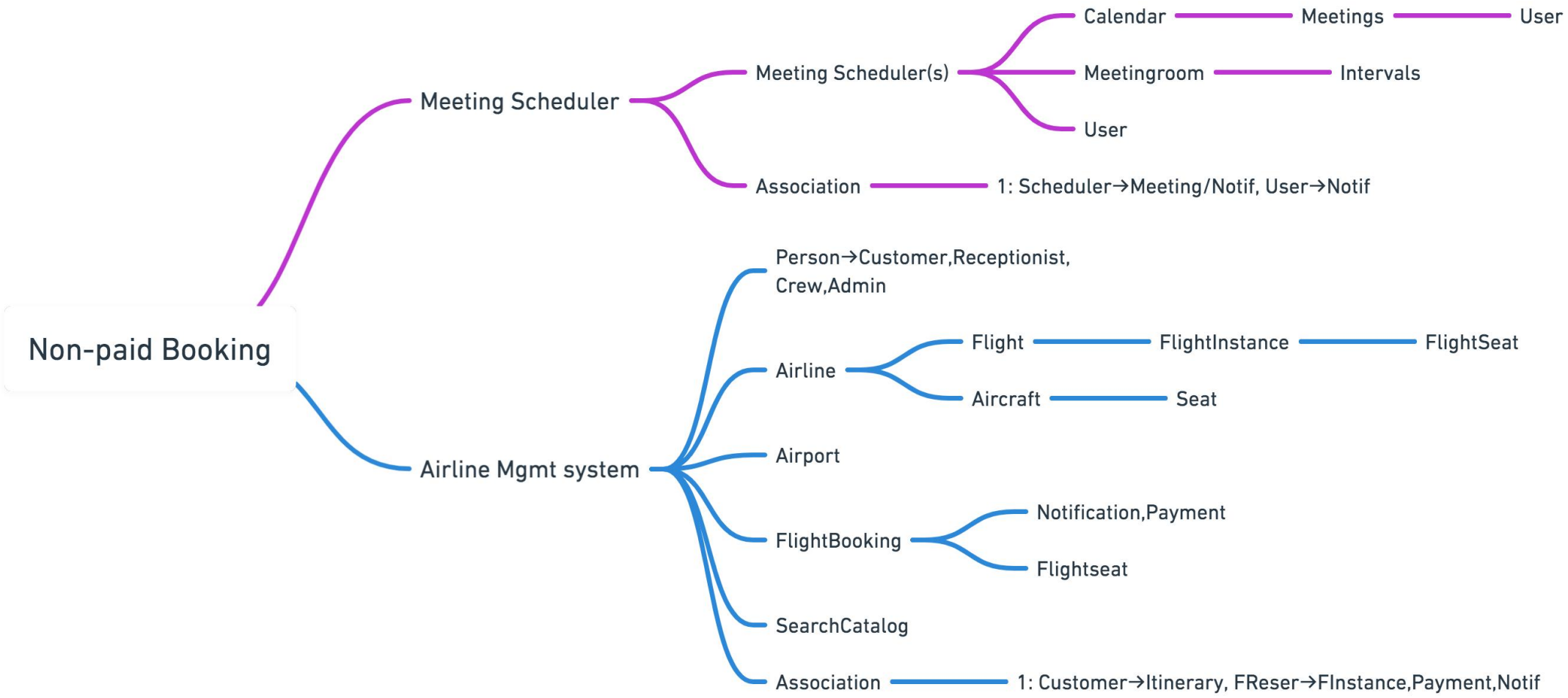
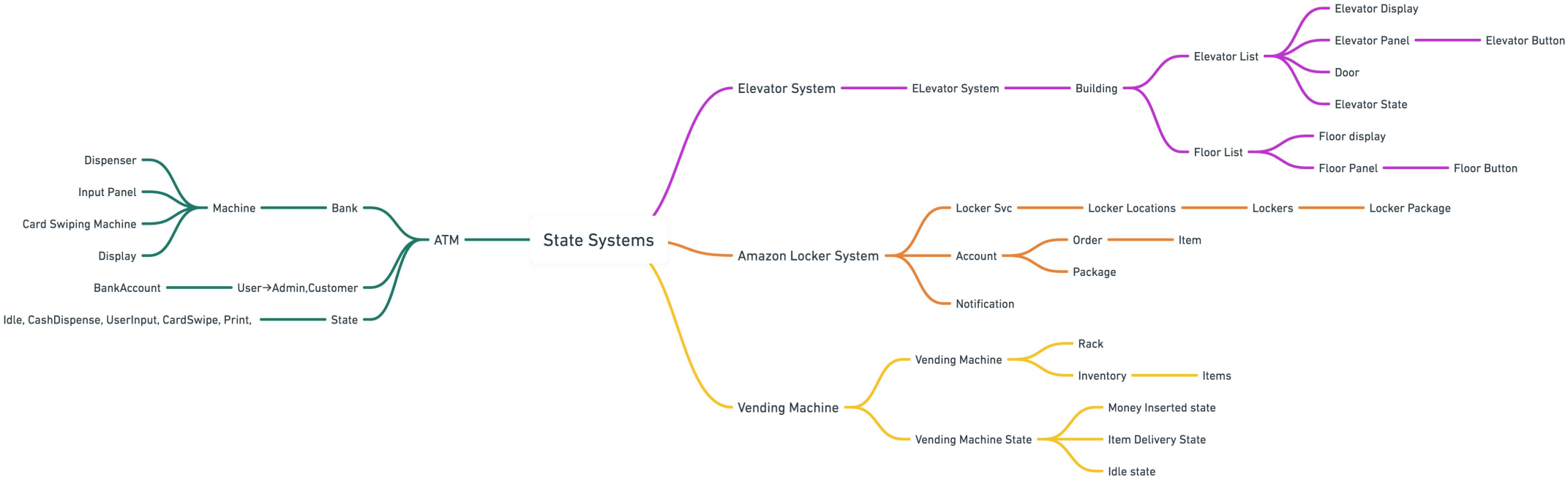
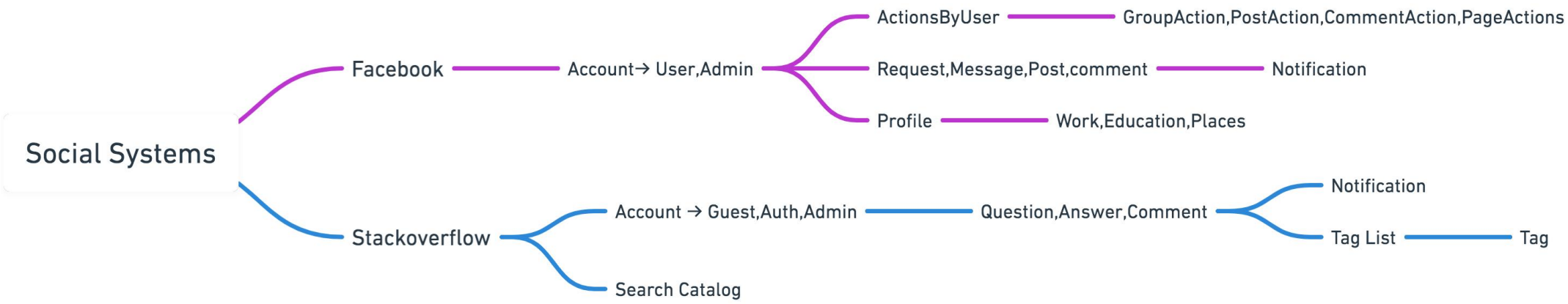
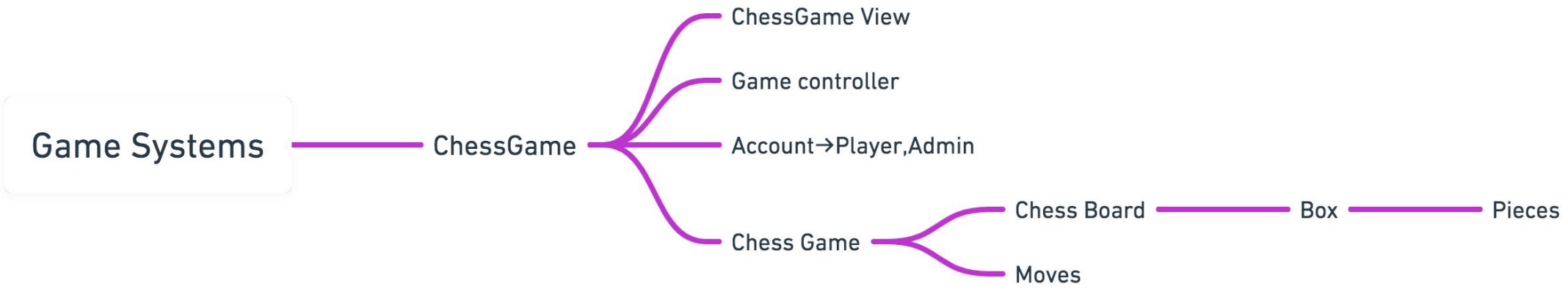

Generic Components
One-way Association
By def, one is associated with another but not reverse
One entity can do multiple second
Two-way association
By def, both are associated with another
Composition
Both cannot exist independently
Aggregation
Weaker, one contains another, can exist indep
Inheritance
Payment(a)
amount,status,timestamp
initiateTxn()
Fine(Decorator)
2-way Associations
booking
Inheritance
Cash,CreditCard
Account(A)
Person, accountstatus, password
resetPassword()
Person
name, address, email, phone
Catalog
x,y,z
Search(Int): searchByX(string query), searchByY(string query)
Aggregation
x aggregates catalog
Notification(A)[Oberver]
content,created
sendNotif()
2-way Associations
booking
Inheritance
Email,Mobile

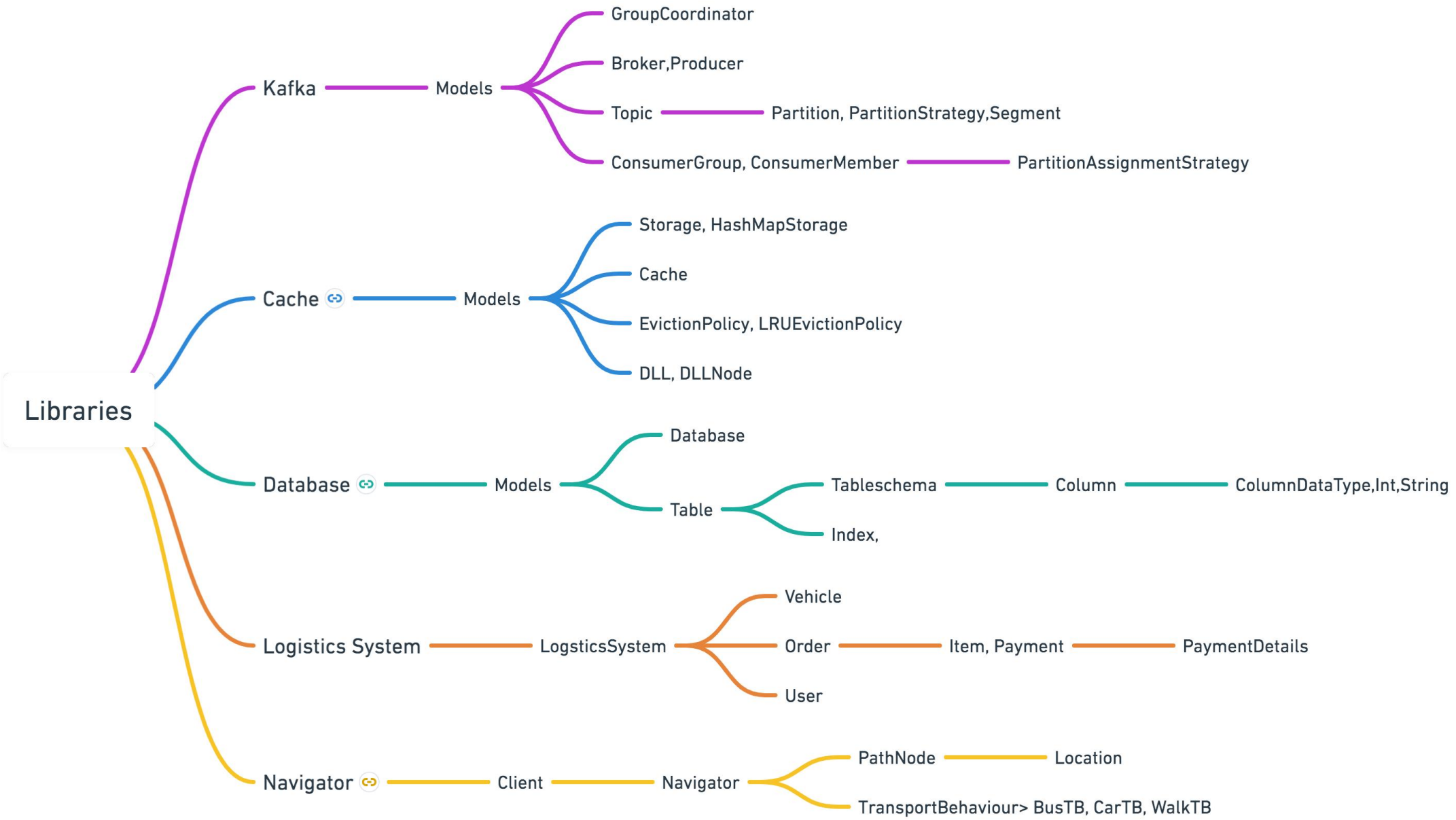

BookingSystems
Car Rental System
ParkingSlot
Branch
Car Company
VehicleLog
Vehicles
Vehicle Catalog
Person → User,Receptionist, Admin
Fine
Payment,Notification
CarReservation
Services,Equipments
1: Account/Reserv.→Vehicle
Associations
Hotel Mgmt system
Rooms(factory)
Hotel(S)
RoomCatalog
Services
Reservation
Account → User,Admin, Receptionist
Notification,Payment
1: Receptionist/Guest→RoomBooking.
RoomBooking→Room
Associations
Strategy Design for Rate Calc
Amazon
Account→Guest,Auth,Admin
Inventory
CartItem
Cart
Notification,Payment
Order
Order Log
Reviews, Ratings
Product
Product Category
1: Product→Category/Review
Associations
2: Product-Account,Product-CartItem
Restaurant
Seat
Table
Menu Item
Menu Section
Menu
Restaurant
Notification
Payments
TableReservation
Account→User,Employee,Admin
MealItem
Meal
Order
1: Employee→Branch, Meal→TableSeat,
MealItem→MenuItem, Waiter→Order, Order→Table,
Chef→Order/Kithcen, Reception→Reser, Reser→Table
Associations
2: Branch-Menu,Customer-Reser,
Parking Lot
ParkingManager(Singleton)
Parking Lots(A)
ParkingTickets
Vehicle(A)
Display Board
ParkingRate
Entrance, Exit
Account,Payment
Associations
1: ParkingSpot→Vehicle, Vehicle→ParkingTicket
2: Payment-ParkingTicket
Library MS
Book
Book Item
Library
Rack
bookLending
BookReservation
Account→Librarian,Member
Library Card
Person→Author
Payment. Search.Notification
Associations
1: BookL/R→BookItem, User→BookItem.
2: Payment-BookR/L,Notif-BookR/L,User-BookL,
Rack/Librarian→BookItem
BookMyshow
Location
Theatres
Halls
Movie Seat
Movie Show
Movie
Booking
Movie Tickets
Payment,Notification
User→Customer,Receptionist,Admin
Catalog
Associations
1: Booking→Show/Ticket/Seat, Showtime→Seat/Ticket,
Movie→Show,Ticket
2: Booking

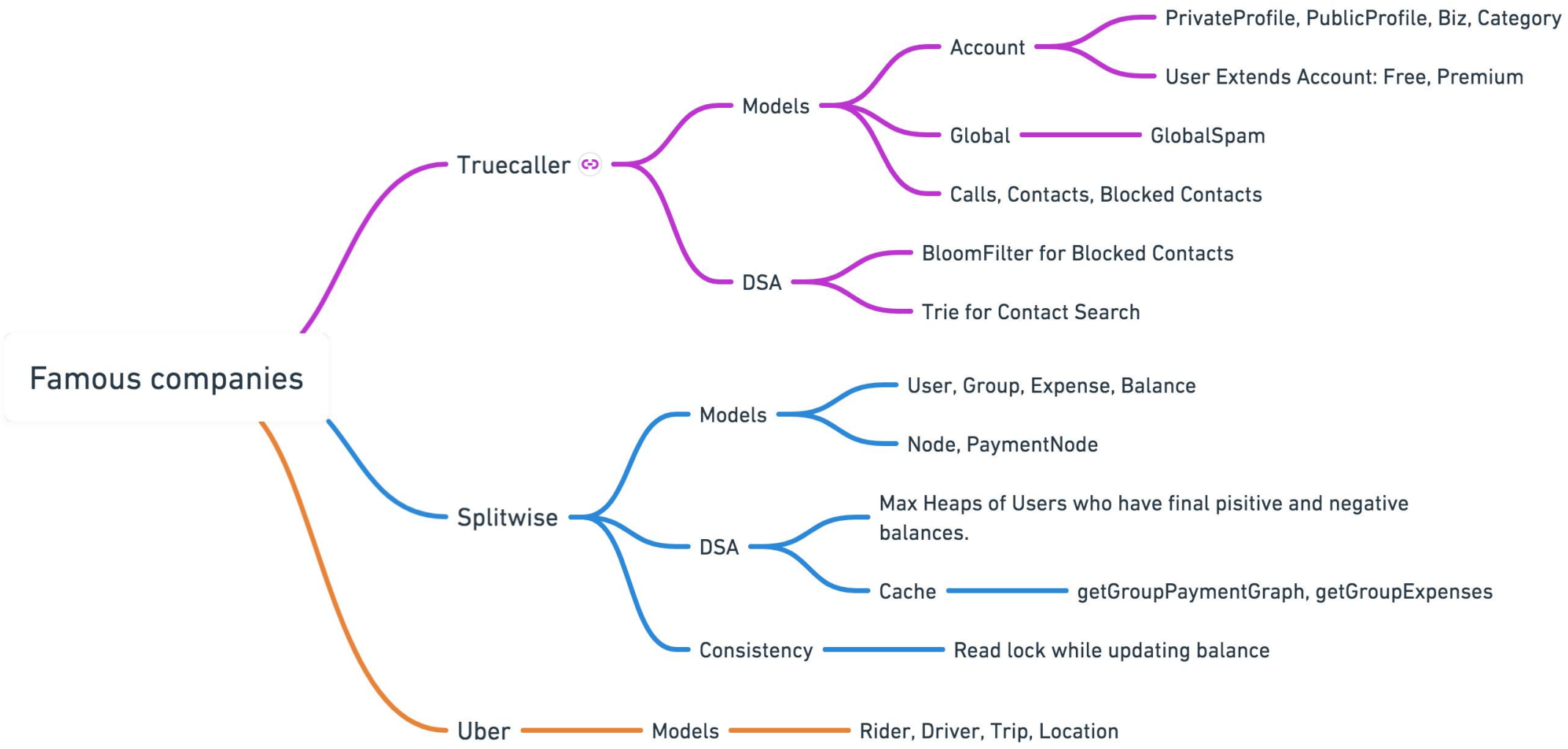

Non-paid Booking
Meeting Scheduler
Meeting Scheduler(s)
Calendar
Meetings
User
Meetingroom
Intervals
User
Association
1: Scheduler→Meeting/Notif, User→Notif
Airline Mgmt system
Person→Customer,Receptionist,
Crew,Admin
Airline
Flight
FlightInstance
FlightSeat
Aircraft
Seat
Airport
FlightBooking
Notification,Payment
Flightseat
SearchCatalog
Association
1: Customer→Itinerary, FReser→FInstance,Payment,Notif


State Systems
ATM
Machine
Dispenser
Input Panel
Card Swiping Machine
Display
Bank
BankAccount
User→Admin,Customer
State
Idle, CashDispense, UserInput, CardSwipe, Print,
Elevator System
ELevator System
Building
Elevator List
Elevator Display
Elevator Panel
Elevator Button
Door
Elevator State
Floor List
Floor display
Floor Panel
Floor Button
Amazon Locker System
Locker Svc
Locker Locations
Lockers
Locker Package
Account
Order
Item
Package
Notification
Vending Machine
Vending Machine
Rack
Inventory
Items
Vending Machine State
Money Inserted state
Item Delivery State
Idle state


Social Systems
Facebook
Account→ User,Admin
ActionsByUser
GroupAction,PostAction,CommentAction,PageActions
Request,Message,Post,comment
Notification
Profile
Work,Education,Places
Stackoverflow
Account → Guest,Auth,Admin
Question,Answer,Comment
Notification
Tag List
Tag
Search Catalog


Game Systems
ChessGame
ChessGame View
Game controller
Account→Player,Admin
Chess Game
Chess Board
Box
Pieces
Moves

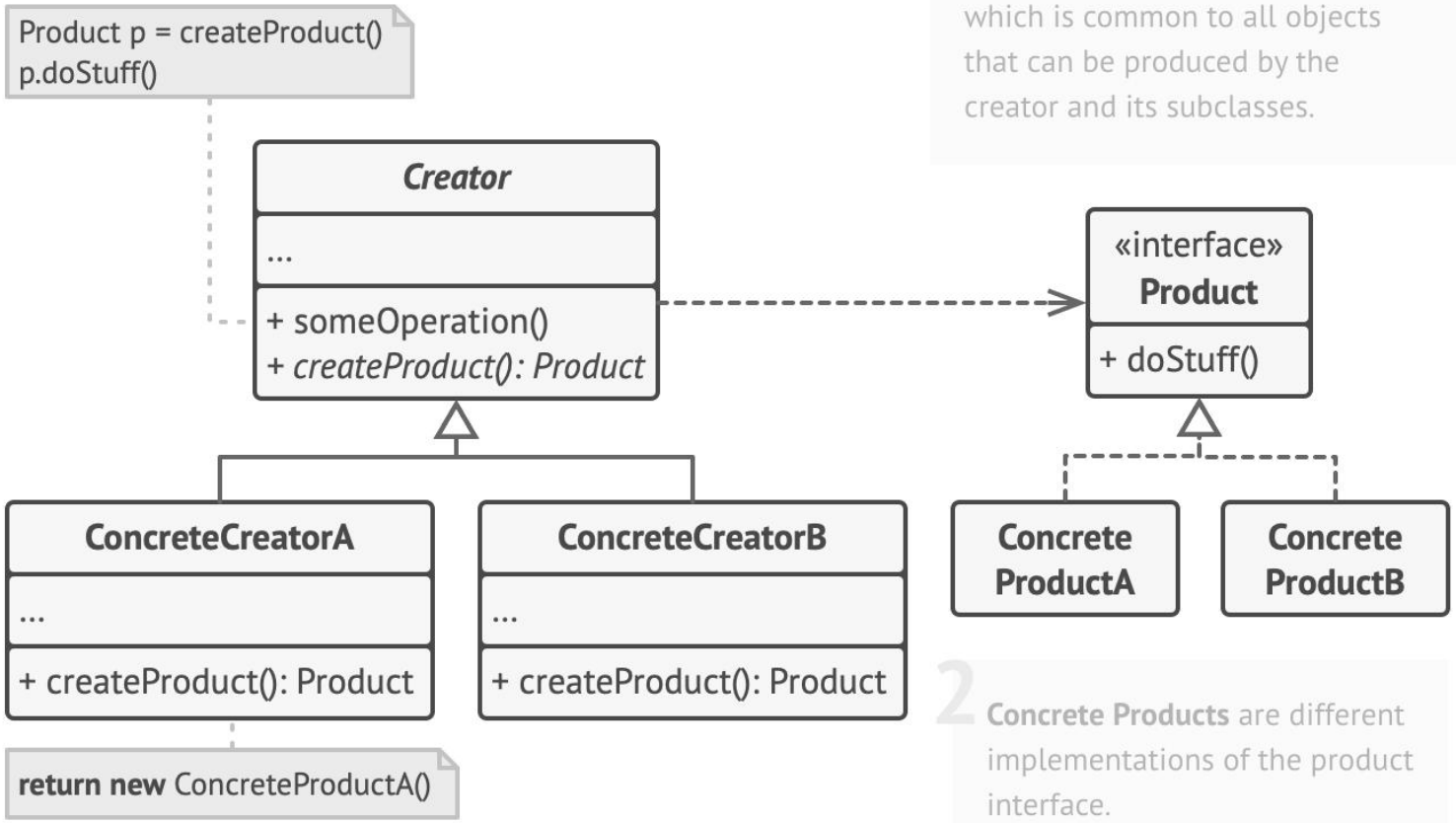

Libraries
Kafka
Models
GroupCoordinator
Broker,Producer
Topic
Partition, PartitionStrategy,Segment
ConsumerGroup, ConsumerMember
PartitionAssignmentStrategy
Cache
Models
Storage, HashMapStorage
Cache
EvictionPolicy, LRUEvictionPolicy
DLL, DLLNode
Database
Models
Database
Table
Tableschema
Column
ColumnDataType,Int,String
Index,
Logistics System
LogsticsSystem
Vehicle
Order
Item, Payment
PaymentDetails
User
Navigator
Client
Navigator
PathNode
Location
TransportBehaviour> BusTB, CarTB, WalkTB


Famous companies
Truecaller
Models
Account
PrivateProfile, PublicProfile, Biz, Category
User Extends Account: Free, Premium
Global
GlobalSpam
Calls, Contacts, Blocked Contacts
DSA
BloomFilter for Blocked Contacts
Trie for Contact Search
Splitwise
Models
User, Group, Expense, Balance
Node, PaymentNode
DSA
Max Heaps of Users who have final pisitive and negative balances.
Cache
getGroupPaymentGraph, getGroupExpenses
Consistency
Read lock while updating balance
Uber
Models
Rider, Driver, Trip, Location

3 The **Creator** class declares the factory method that returns new product objects. It's important that the return type of this method matches the product interface.

You can declare the factory method as `abstract` to force all subclasses to implement their own versions of the method. As an alternative, the base factory method can return some default product type.

Note, despite its name, product creation is **not** the primary responsibility of the creator. Usually, the creator class already has some core business logic related to products. The factory method helps to decouple this logic from the concrete product classes. Here is an analogy: a large software development company can have a training department for programmers. However, the primary function of the company as a whole is still writing code, not producing programmers.

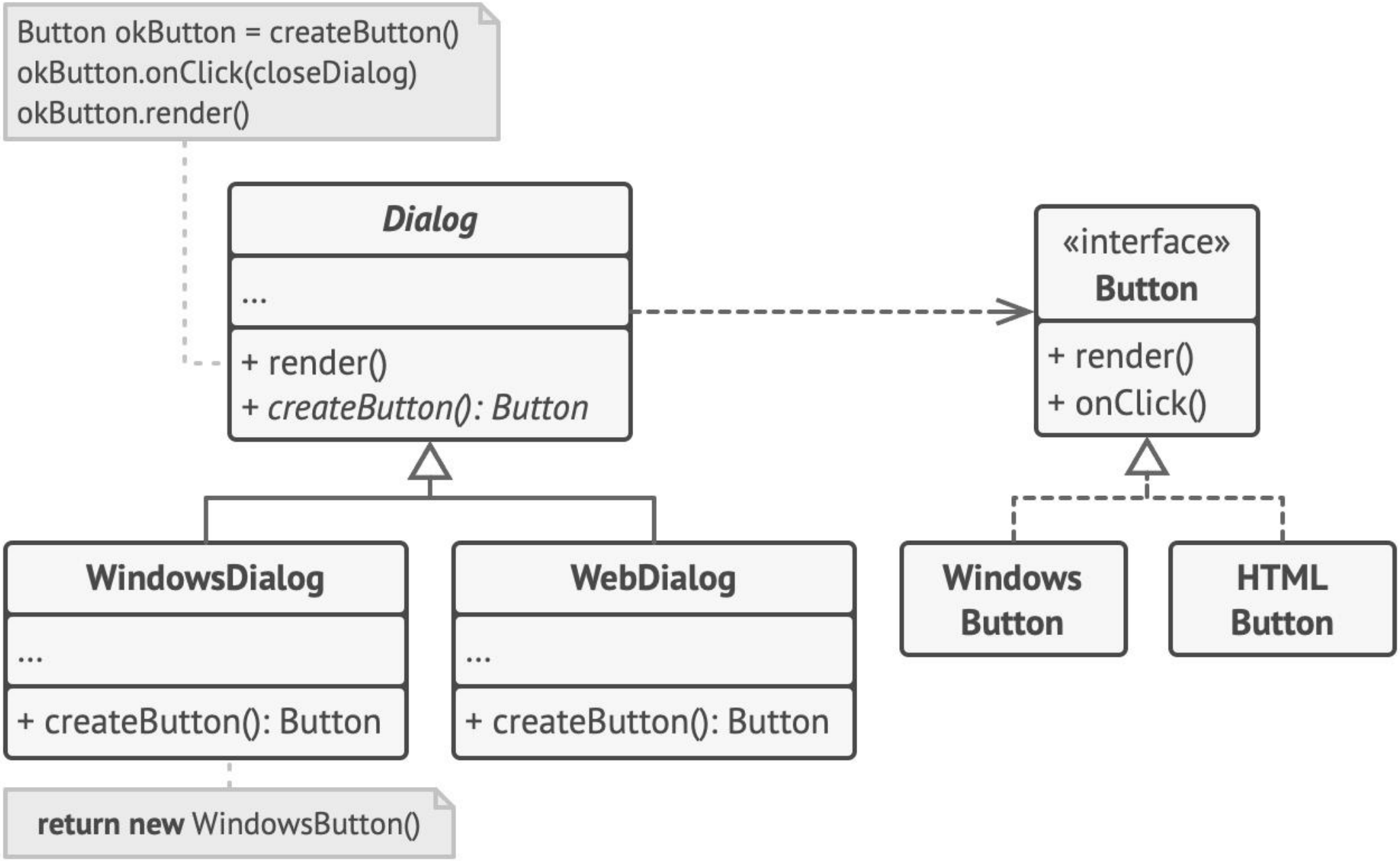


1 The **Product** declares the interface, which is common to all objects that can be produced by the creator and its subclasses.

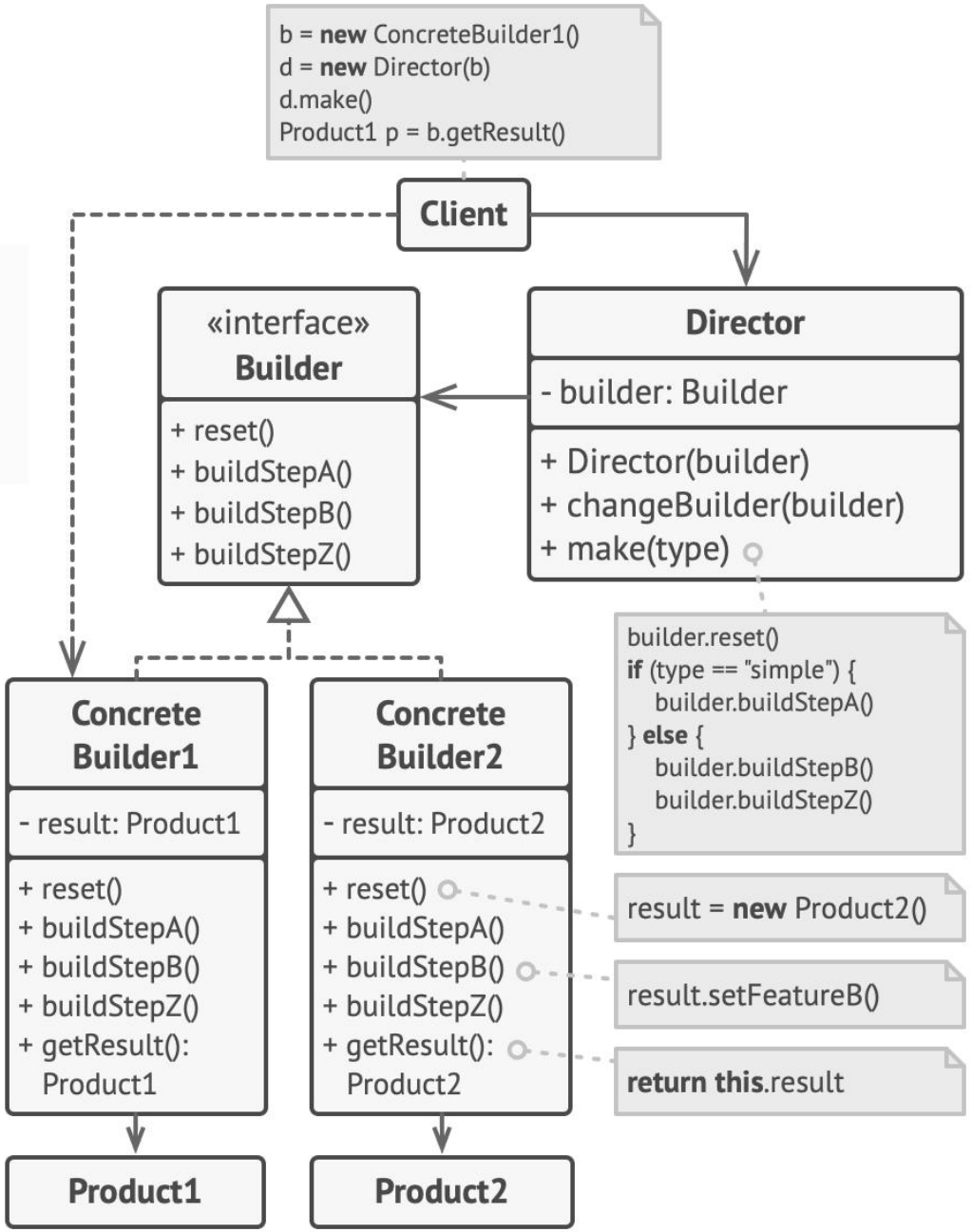
2 **Concrete Products** are different implementations of the product interface.

4 **Concrete Creators** override the base factory method so it returns a different type of product.

Note that the factory method doesn't have to **create** new instances all the time. It can also return existing objects from a cache, an object pool, or another source.



The cross-platform dialog example



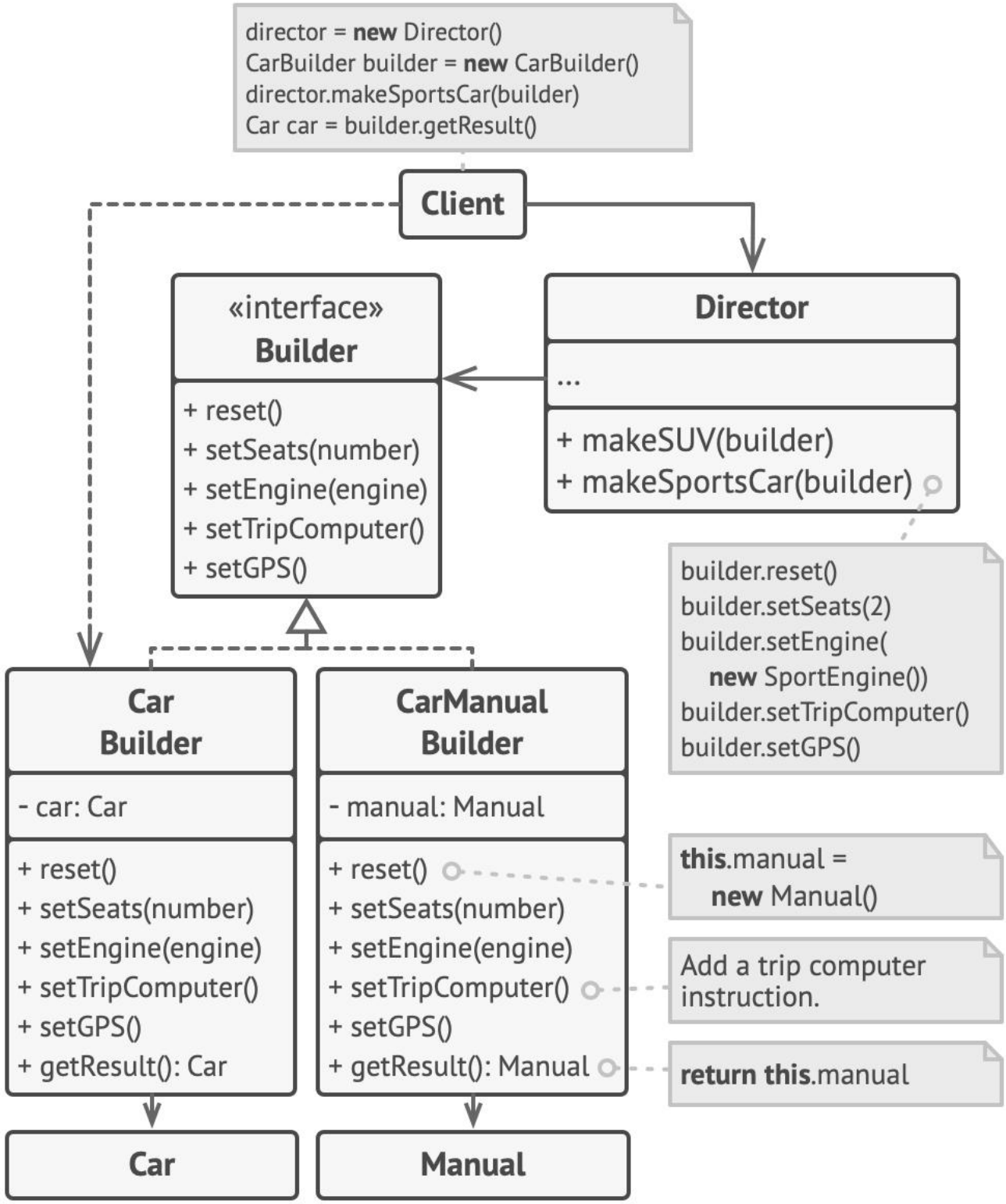
1 The **Builder** interface declares product construction steps that are common to all types of builders.

2 **Concrete Builders** provide different implementations of the construction steps. Concrete builders may produce products that don't follow the common interface.

3 **Products** are resulting objects. Products constructed by different builders don't have to belong to the same class hierarchy or interface.

4 The **Director** class defines the order in which to call construction steps, so you can create and reuse specific configurations of products.

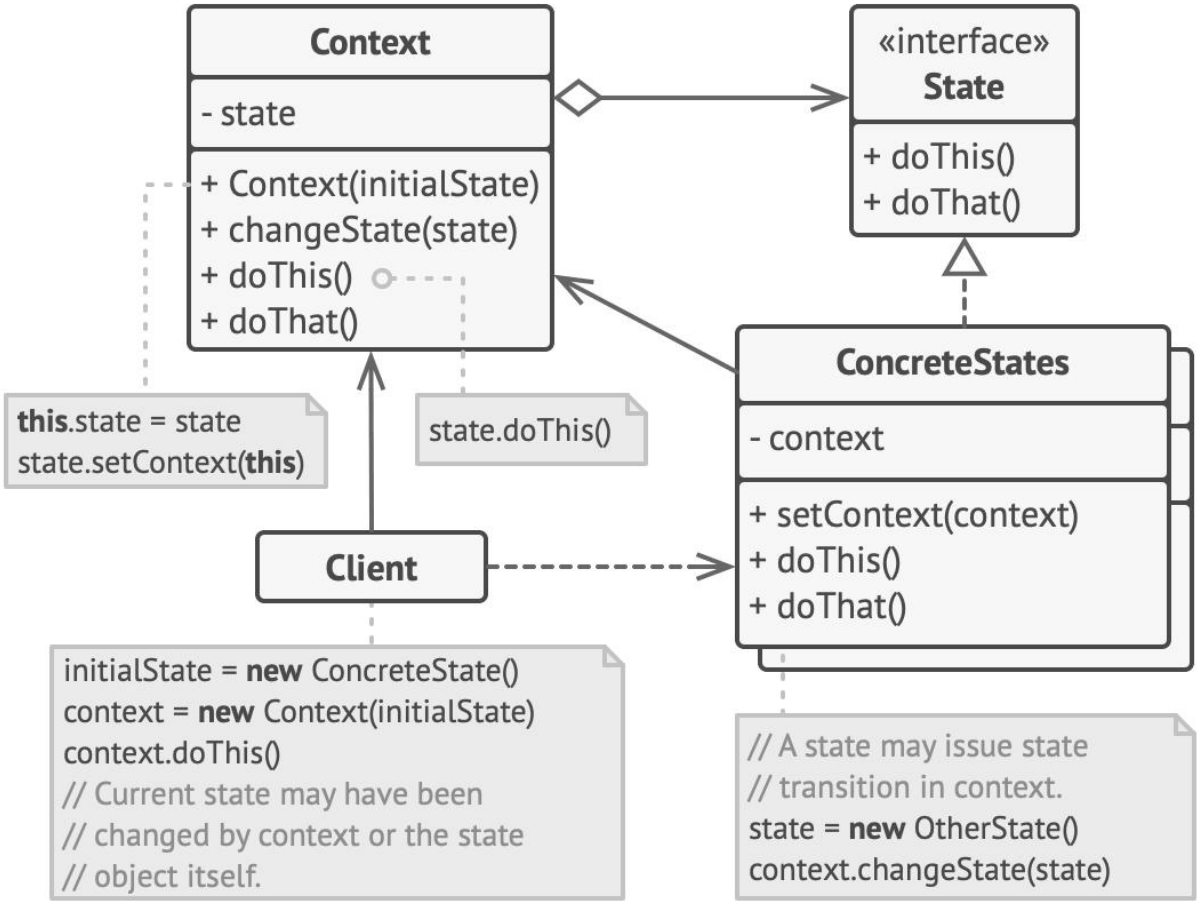
5 The **Client** must associate one of the builder objects with the director. Usually, it's done just once, via parameters of the director's constructor. Then the director uses that builder object for all further construction. However, there's an alternative approach for when the client passes the builder object to the production method of the director. In this case, you can use a different builder each time you produce something with the director.



The example of step-by-step construction of cars and the user guides that fit those car model.

1 **Context** stores a reference to one of the concrete state objects and delegates to it all state-specific work. The context communicates with the state object via the state interface. The context exposes a setter for passing it a new state object.

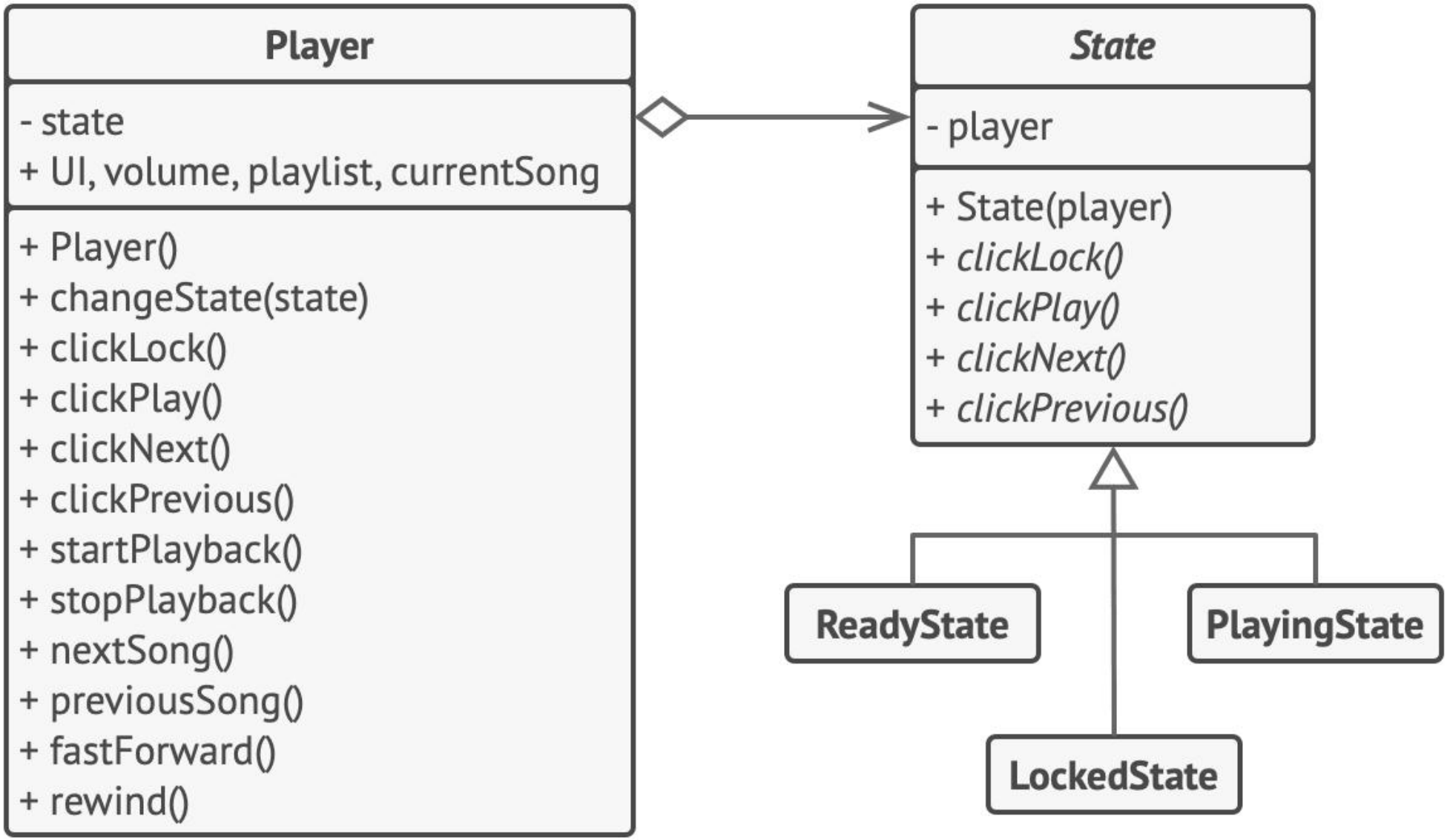
2 The **State** interface declares the state-specific methods. These methods should make sense for all concrete states because you don't want some of your states to have useless methods that will never be called.



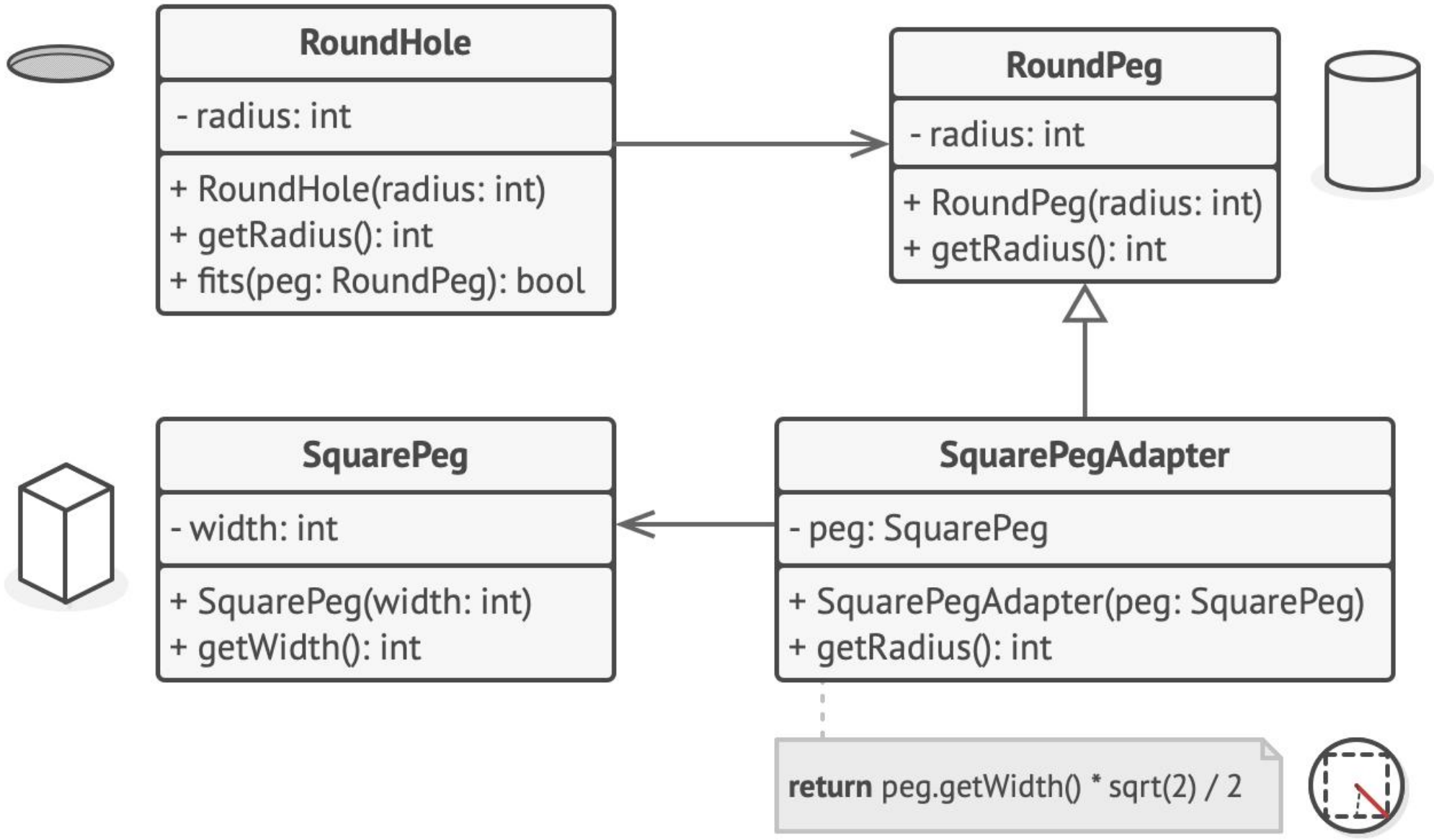
3 **Concrete States** provide their own implementations for the state-specific methods. To avoid duplication of similar code across multiple states, you may provide intermediate abstract classes that encapsulate some common behavior.

State objects may store a backreference to the context object. Through this reference, the state can fetch any required info from the context object, as well as initiate state transitions.

4 Both context and concrete states can set the next state of the context and perform the actual state transition by replacing the state object linked to the context.



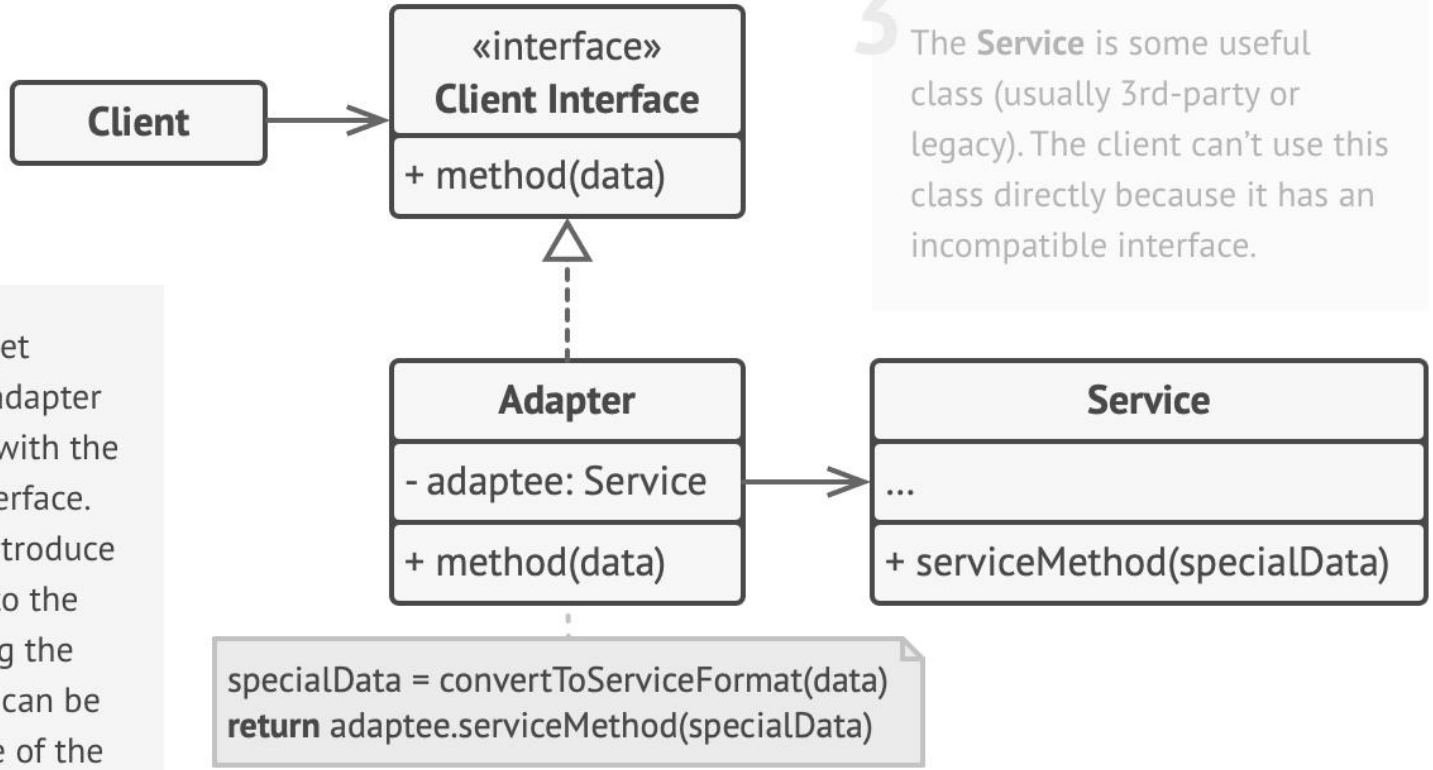
Example of changing object behavior with state objects.



Adapting square pegs to round holes.

2 The **Client Interface** describes a protocol that other classes must follow to be able to collaborate with the client code.

1 The **Client** is a class that contains the existing business logic of the program.

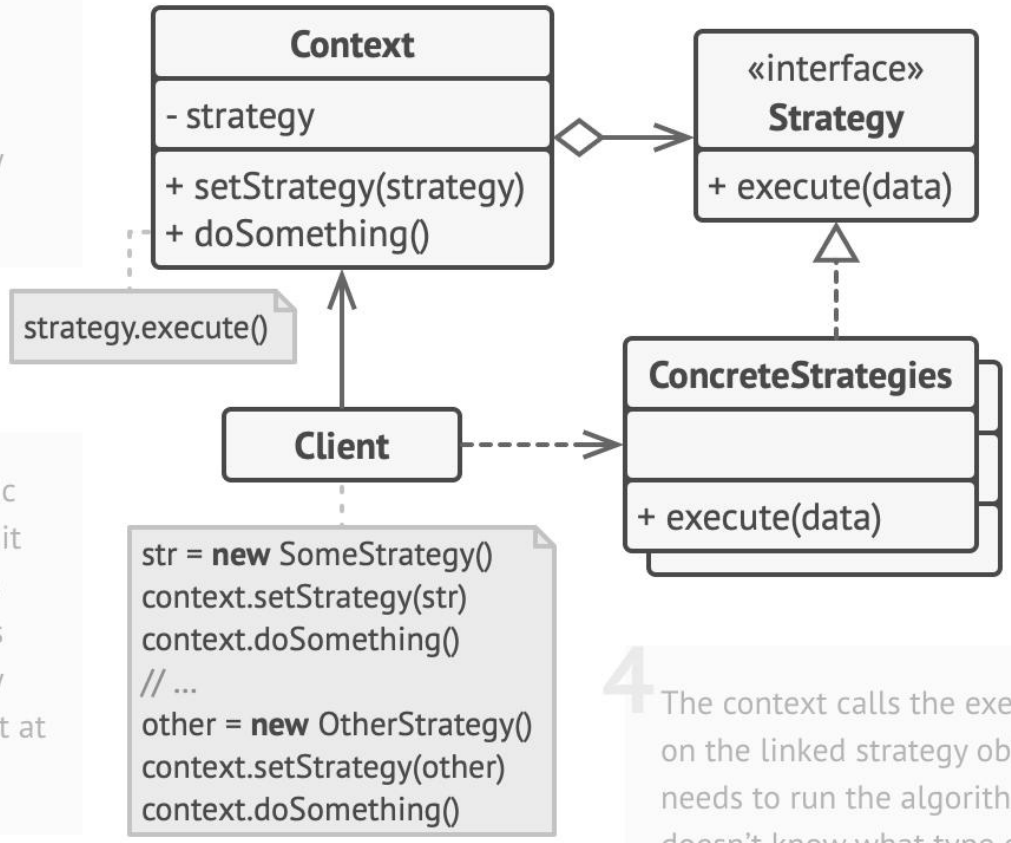


3 The **Service** is some useful class (usually 3rd-party or legacy). The client can't use this class directly because it has an incompatible interface.

5 The client code doesn't get coupled to the concrete adapter class as long as it works with the adapter via the client interface. Thanks to this, you can introduce new types of adapters into the program without breaking the existing client code. This can be useful when the interface of the service class gets changed or replaced: you can just create a new adapter class without changing the client code.

4 The **Adapter** is a class that's able to work with both the client and the service: it implements the client interface, while wrapping the service object. The adapter receives calls from the client via the client interface and translates them into calls to the wrapped service object in a format it can understand.

1 The **Context** maintains a reference to one of the concrete strategies and communicates with this object only via the strategy interface.

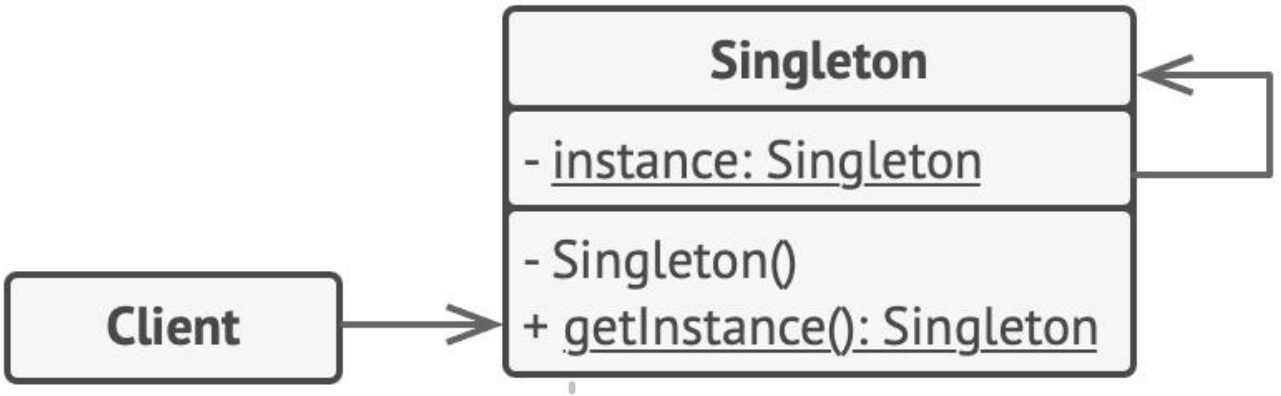


2 The **Strategy** interface is common to all concrete strategies. It declares a method the context uses to execute a strategy.

3 **Concrete Strategies** implement different variations of an algorithm the context uses.

4 The context calls the execution method on the linked strategy object each time it needs to run the algorithm. The context doesn't know what type of strategy it works with or how the algorithm is executed.

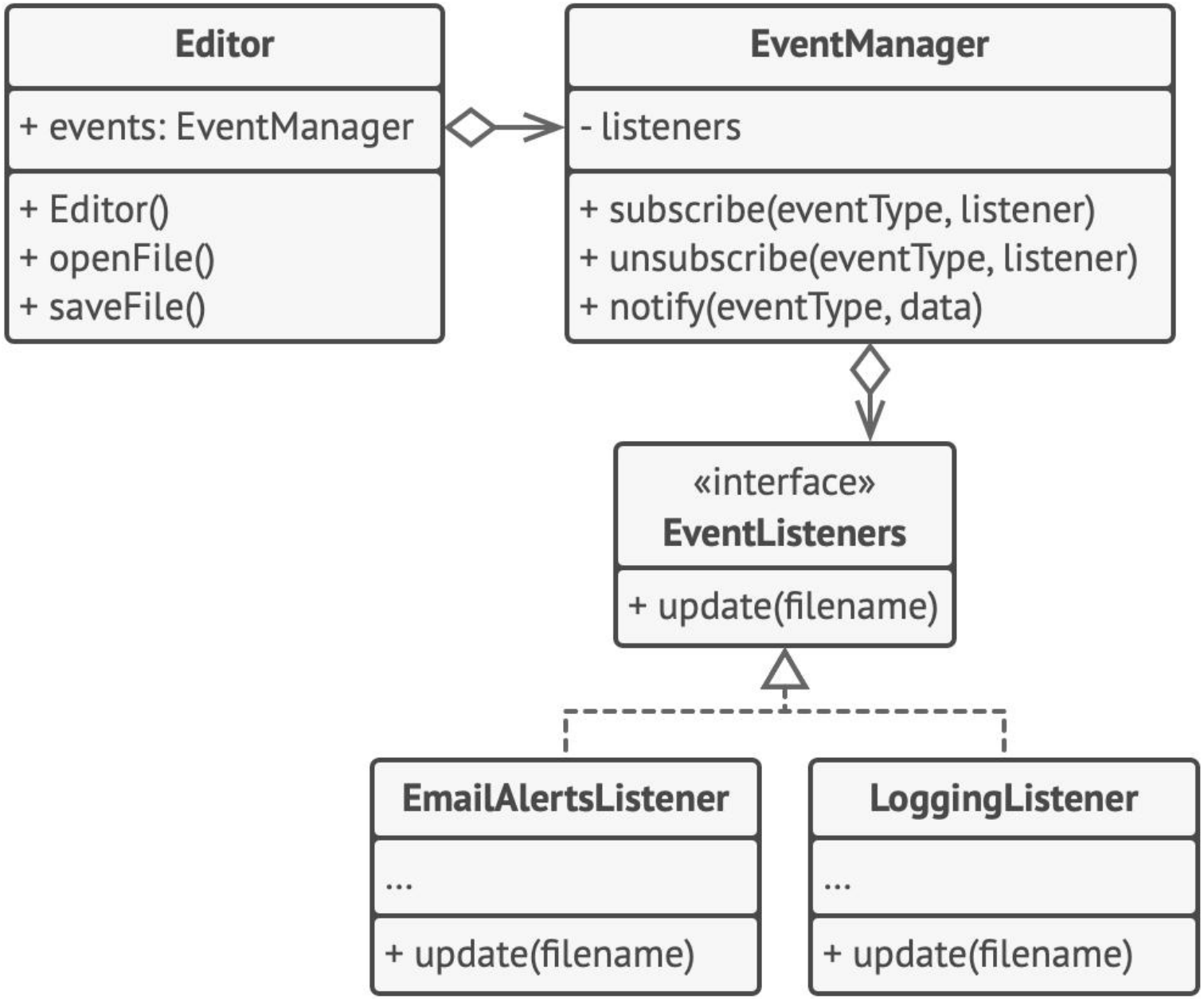
5 The **Client** creates a specific strategy object and passes it to the context. The context exposes a setter which lets clients replace the strategy associated with the context at runtime.



1 The **Singleton** class declares the static method `getInstance` that returns the same instance of its own class.

The Singleton's constructor should be hidden from the client code. Calling the `getInstance` method should be the only way of getting the Singleton object.

```
if (instance == null) {
    // Note: if you're creating an app with
    // multithreading support, you should
    // place a thread lock here.
    instance = new Singleton()
}
return instance
```



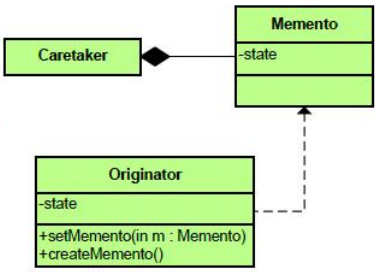
Notifying objects about events that happen to other objects.

C Abstract Factory
S Adapter
S Bridge
C Builder
B Chain of Responsibility
B Command
S Composite
S Decorator
S Facade
C Factory Method
S Flyweight
B Interpreter
B Iterator
B Mediator
B Memento
C Prototype
S Proxy
B Observer
C Singleton
B State
B Strategy
B Template Method
B Visitor

Memento

Type: Behavioral

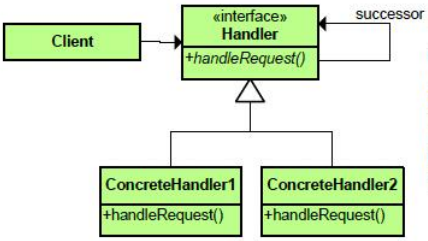
What it is:
Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.



Chain of Responsibility

Type: Behavioral

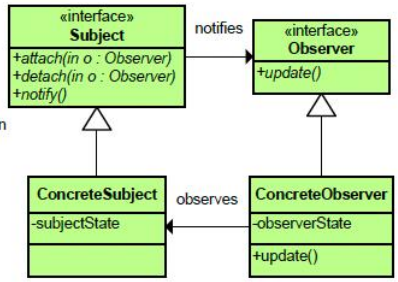
What it is:
Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.



Observer

Type: Behavioral

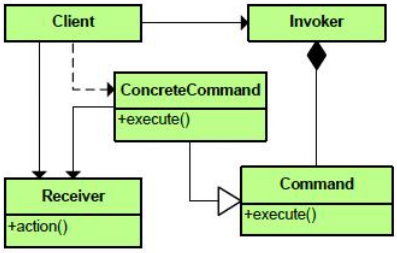
What it is:
Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.



Command

Type: Behavioral

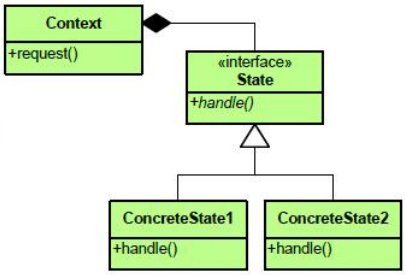
What it is:
Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.



State

Type: Behavioral

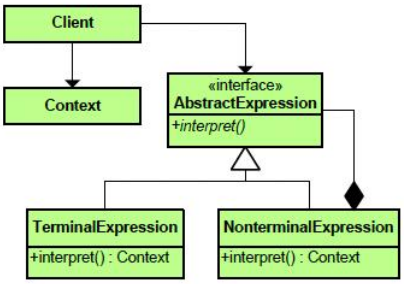
What it is:
Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.



Interpreter

Type: Behavioral

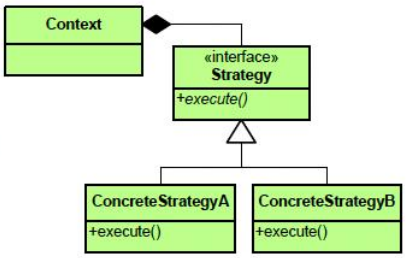
What it is:
Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.



Strategy

Type: Behavioral

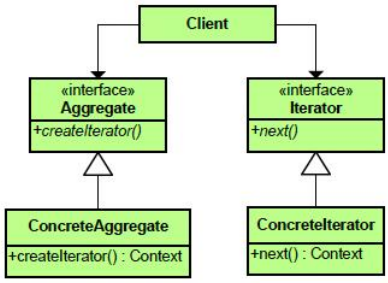
What it is:
Define a family of algorithms, encapsulate each one, and make them interchangeable. Lets the algorithm vary independently from clients that use it.



Iterator

Type: Behavioral

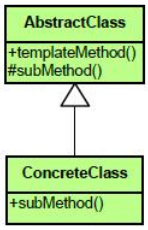
What it is:
Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.



Template Method

Type: Behavioral

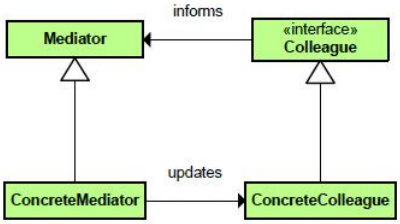
What it is:
Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.



Mediator

Type: Behavioral

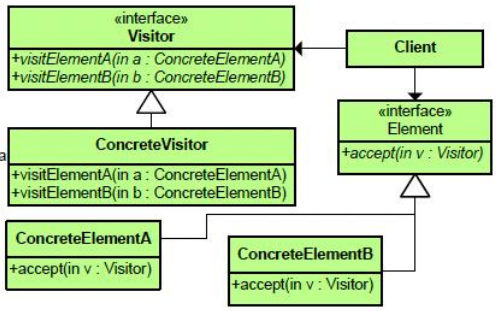
What it is:
Define an object that encapsulates how a set of objects interact. Promotes loose coupling by keeping objects from referring to each other explicitly and it lets you vary their interactions independently.



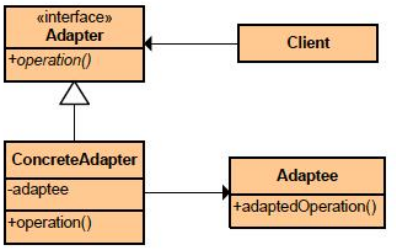
Visitor

Type: Behavioral

What it is:
Represent an operation to be performed on the elements of an object structure. Lets you define a new operation without changing the classes of the elements on which it operates.



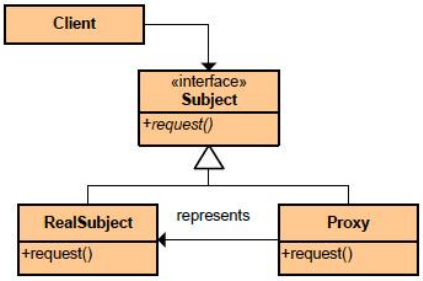
Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Massachusetts: Addison Wesley Longman, Inc..



Adapter

Type: Structural

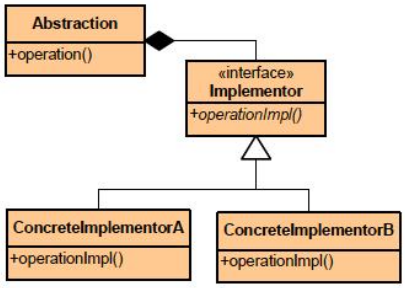
What it is:
Convert the interface of a class into another interface clients expect. Lets classes work together that couldn't otherwise because of incompatible interfaces.



Proxy

Type: Structural

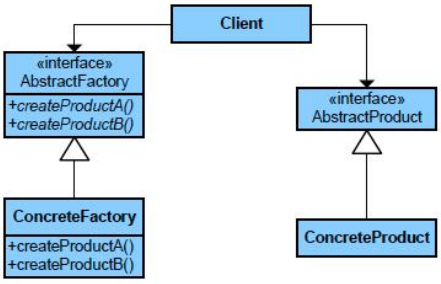
What it is:
Provide a surrogate or placeholder for another object to control access to it.



Bridge

Type: Structural

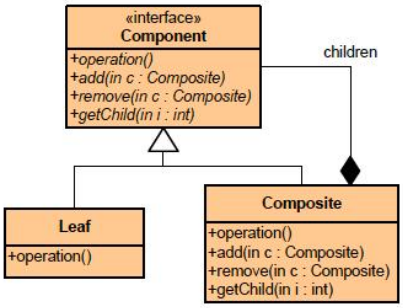
What it is:
Decouple an abstraction from its implementation so that the two can vary independently.



Abstract Factory

Type: Creational

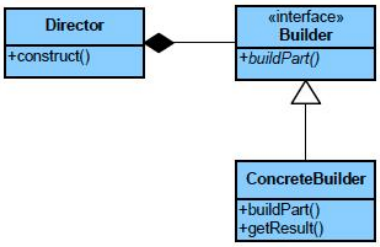
What it is:
Provides an interface for creating families of related or dependent objects without specifying their concrete class.



Composite

Type: Structural

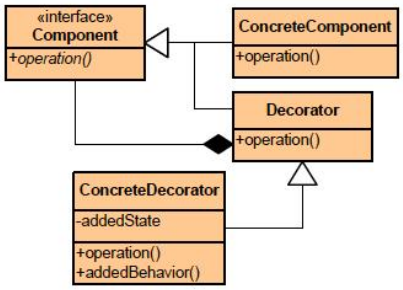
What it is:
Compose objects into tree structures to represent part-whole hierarchies. Lets clients treat individual objects and compositions of objects uniformly.



Builder

Type: Creational

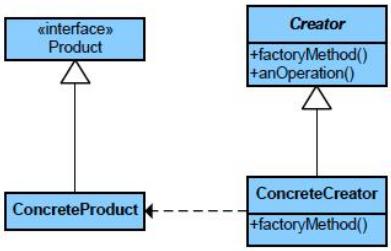
What it is:
Separate the construction of a complex object from its representing so that the same construction process can create different representations.



Decorator

Type: Structural

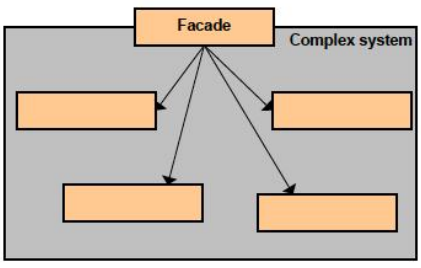
What it is:
Attach additional responsibilities to an object dynamically. Provide a flexible alternative to sub-classing for extending functionality.



Factory Method

Type: Creational

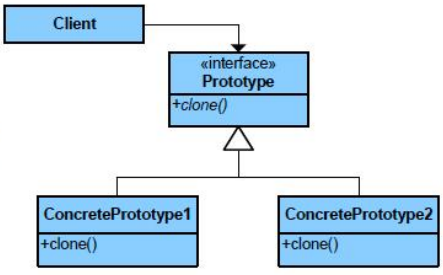
What it is:
Define an interface for creating an object, but let subclasses decide which class to instantiate. Lets a class defer instantiation to subclasses.



Facade

Type: Structural

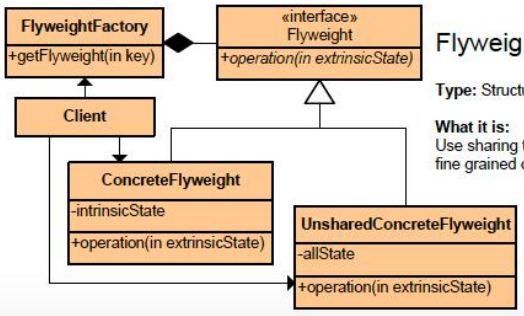
What it is:
Provide a unified interface to a set of interfaces in a subsystem. Defines a high-level interface that makes the subsystem easier to use.



Prototype

Type: Creational

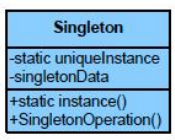
What it is:
Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.



Flyweight

Type: Structural

What it is:
Use sharing to support large numbers of fine grained objects efficiently.

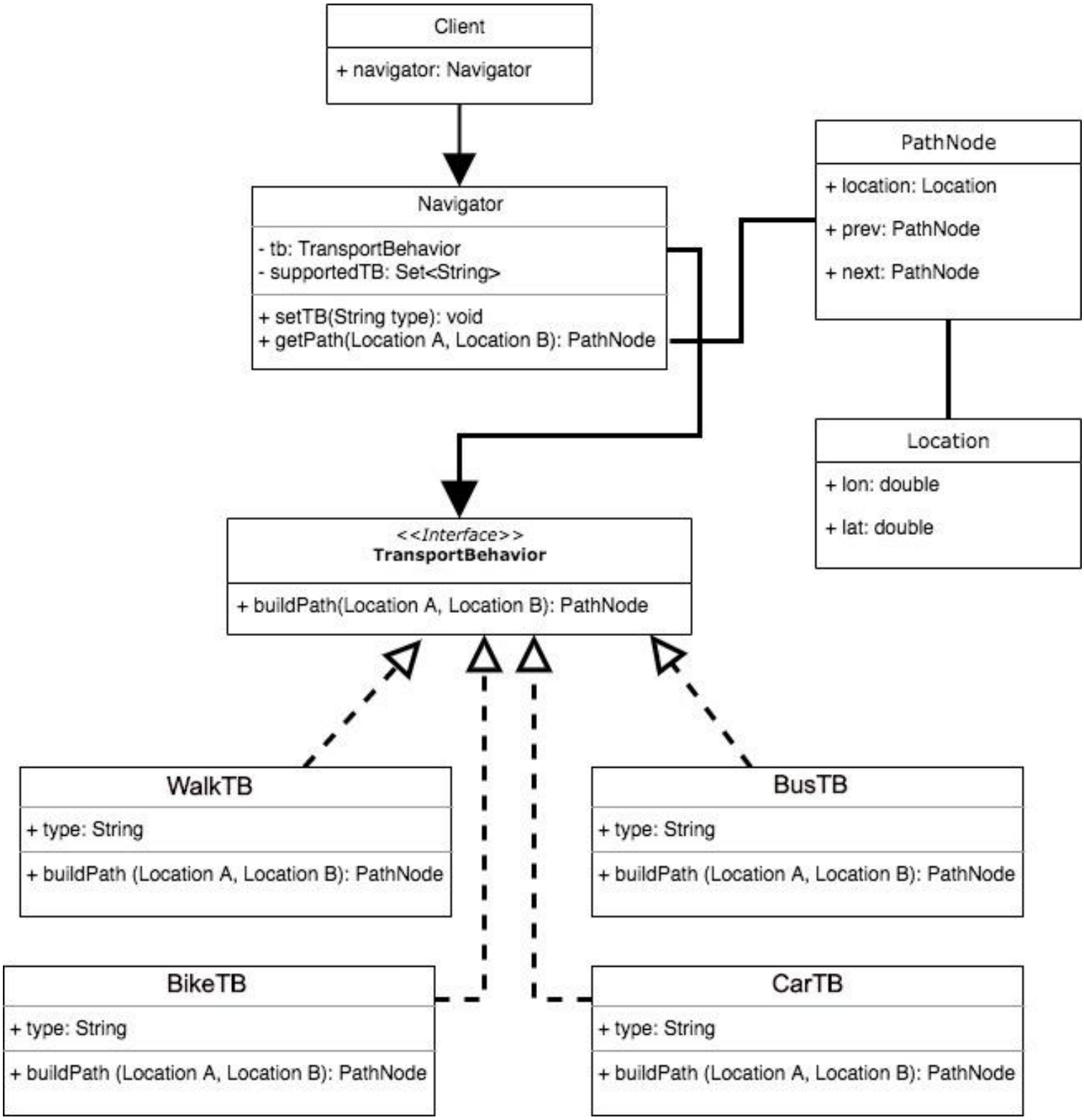


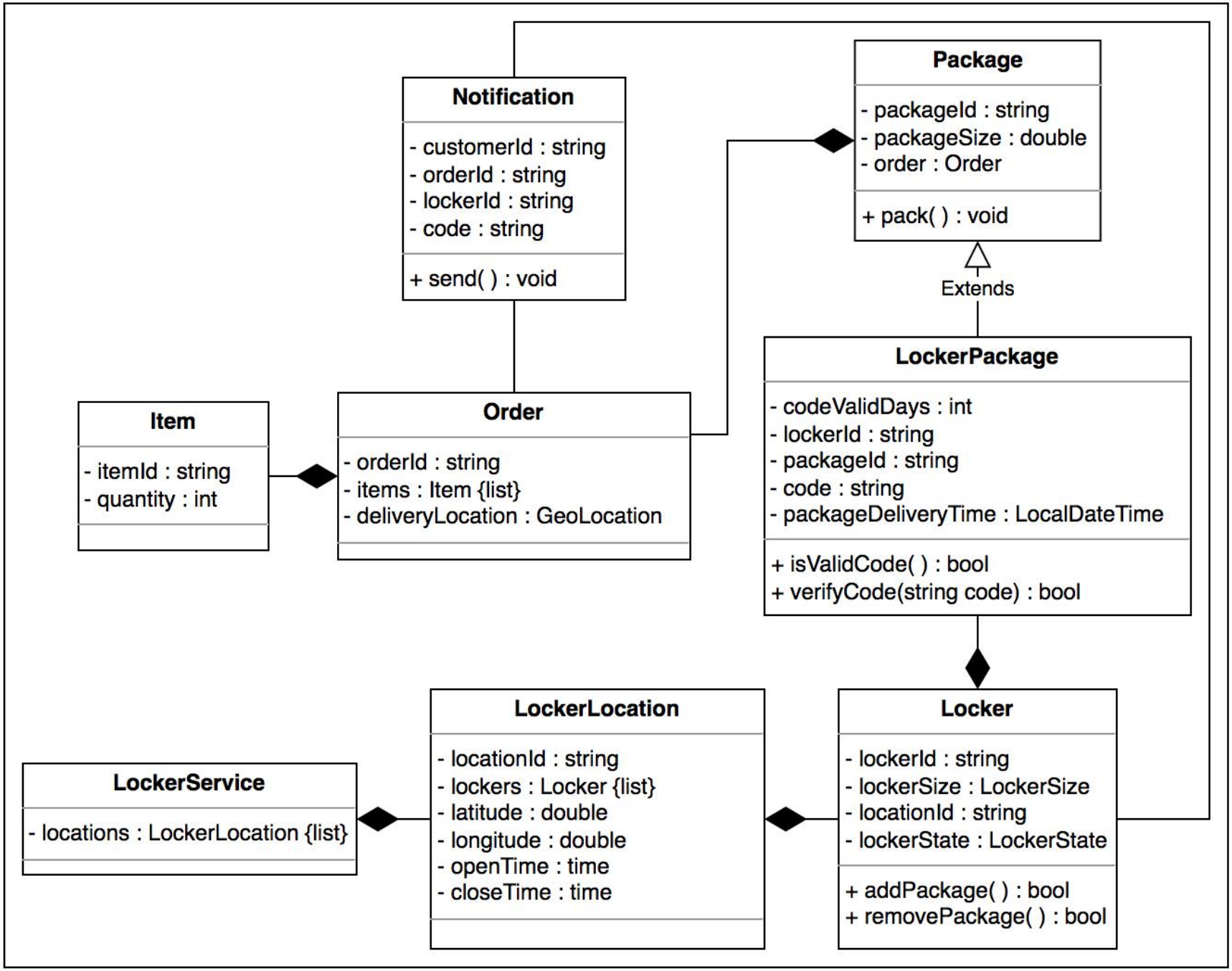
Singleton

Type: Creational

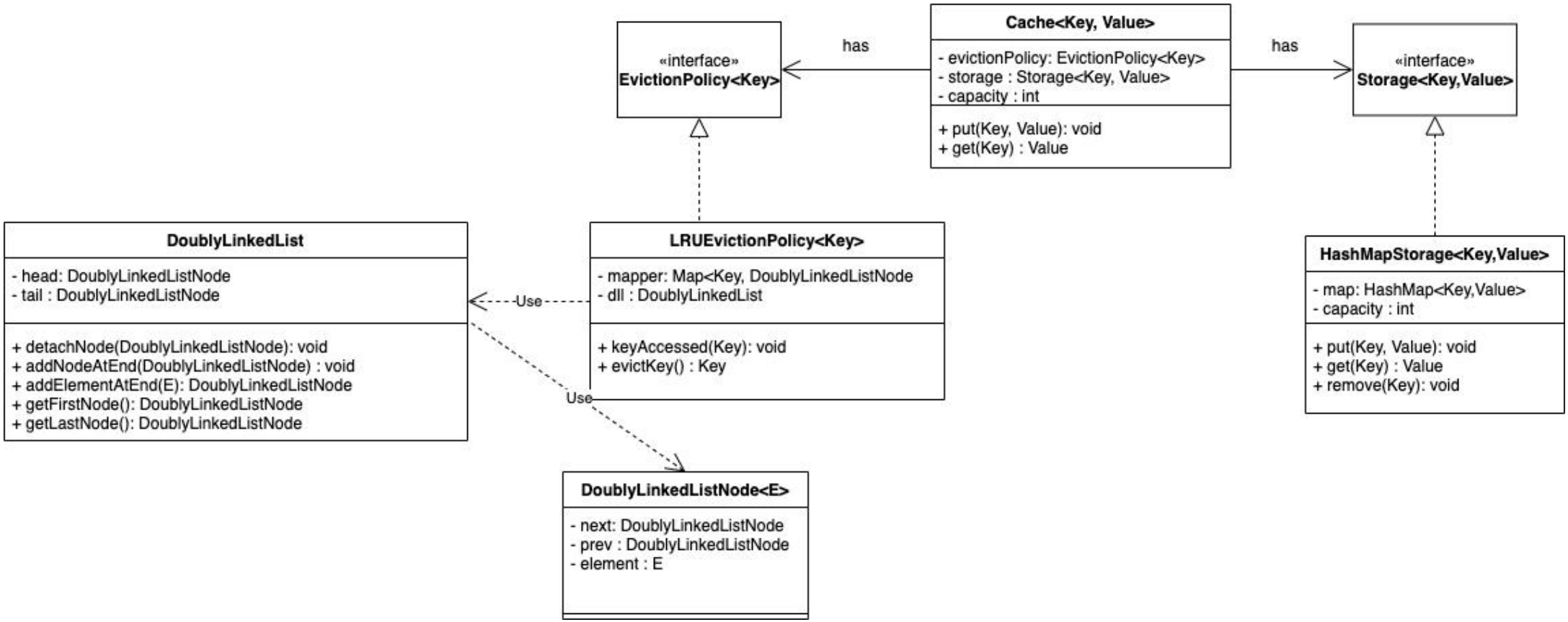
What it is:
Ensure a class only has one instance and provide a global point of access to it.

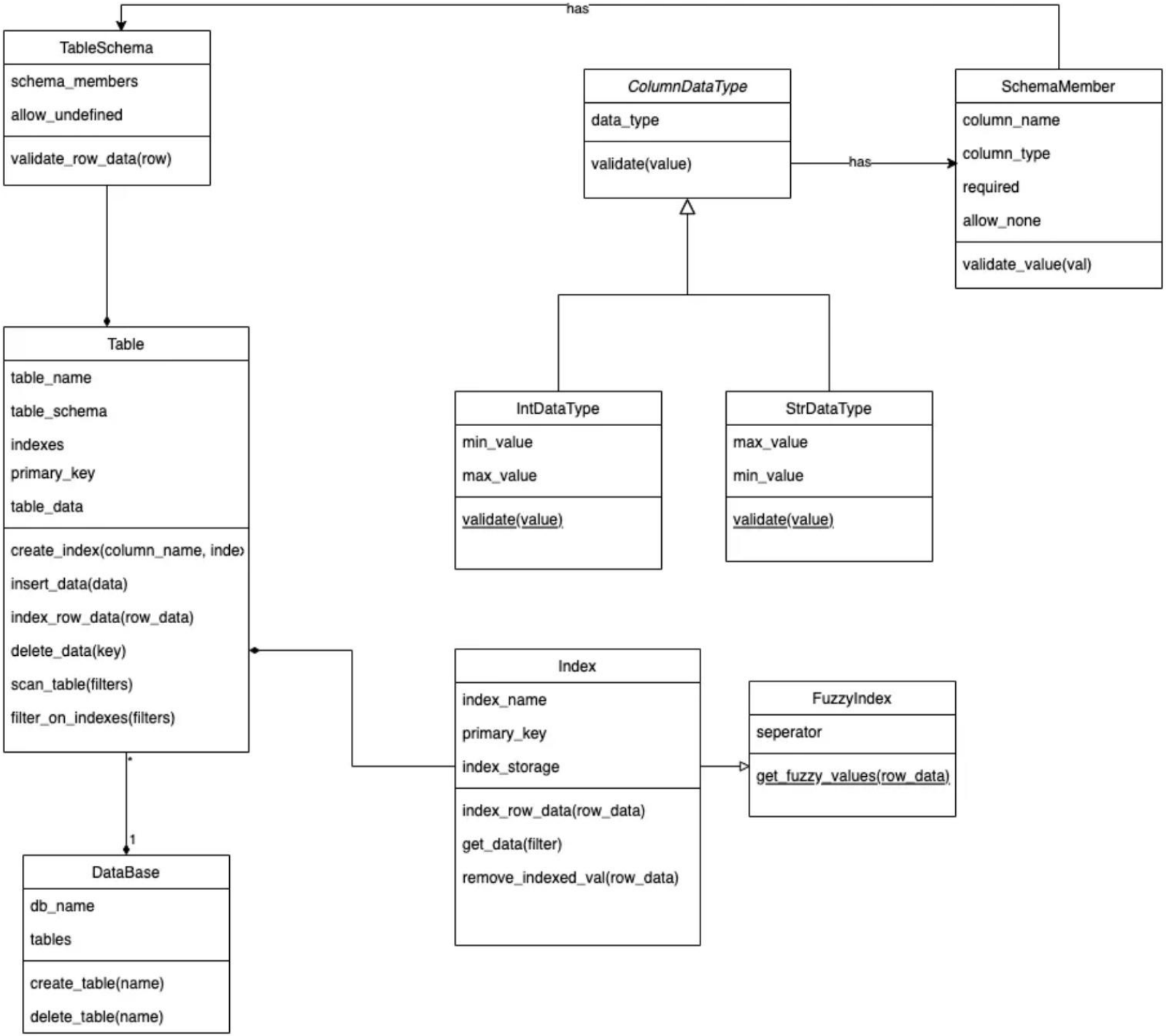
Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Massachusetts: Addison Wesley Longman, Inc..

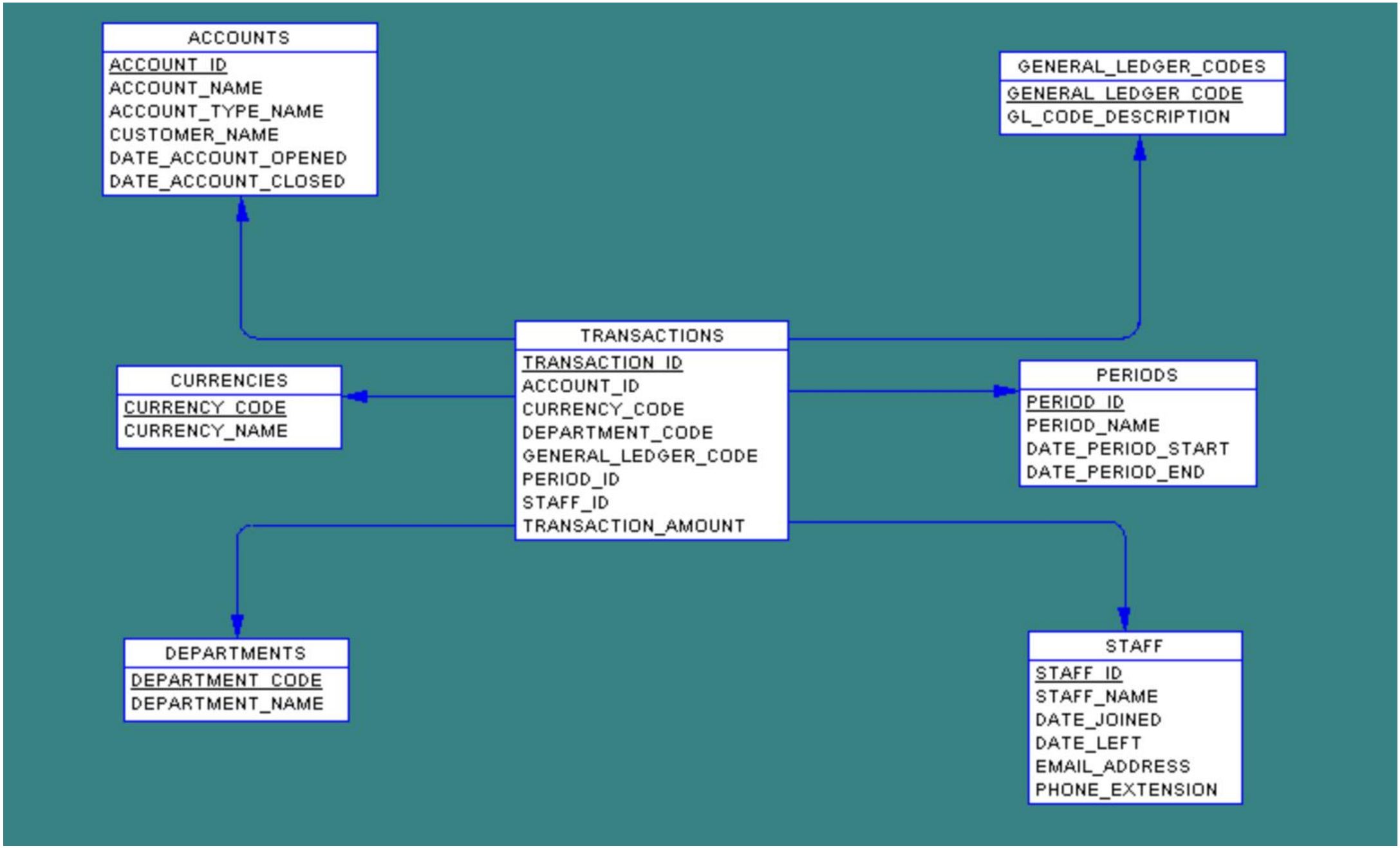

Client
+ navigator: Navigator
Navigator
- tb: TransportBehavior
- supportedTB: Set<String>
+ setTB(String type): void
+ getPath(Location A, Location B): PathNode
PathNode
+ location: Location
+ prev: PathNode
+ next: PathNode
Location
+ lon: double
+ lat: double
<<Interface>>
TransportBehavior
+ buildPath(Location A, Location B): PathNode
WalkTB
+ type: String
+ buildPath (Location A, Location B): PathNode
BusTB
+ type: String
+ buildPath (Location A, Location B): PathNode
BikeTB
+ type: String
+ buildPath (Location A, Location B): PathNode
CarTB
+ type: String
+ buildPath (Location A, Location B): PathNode

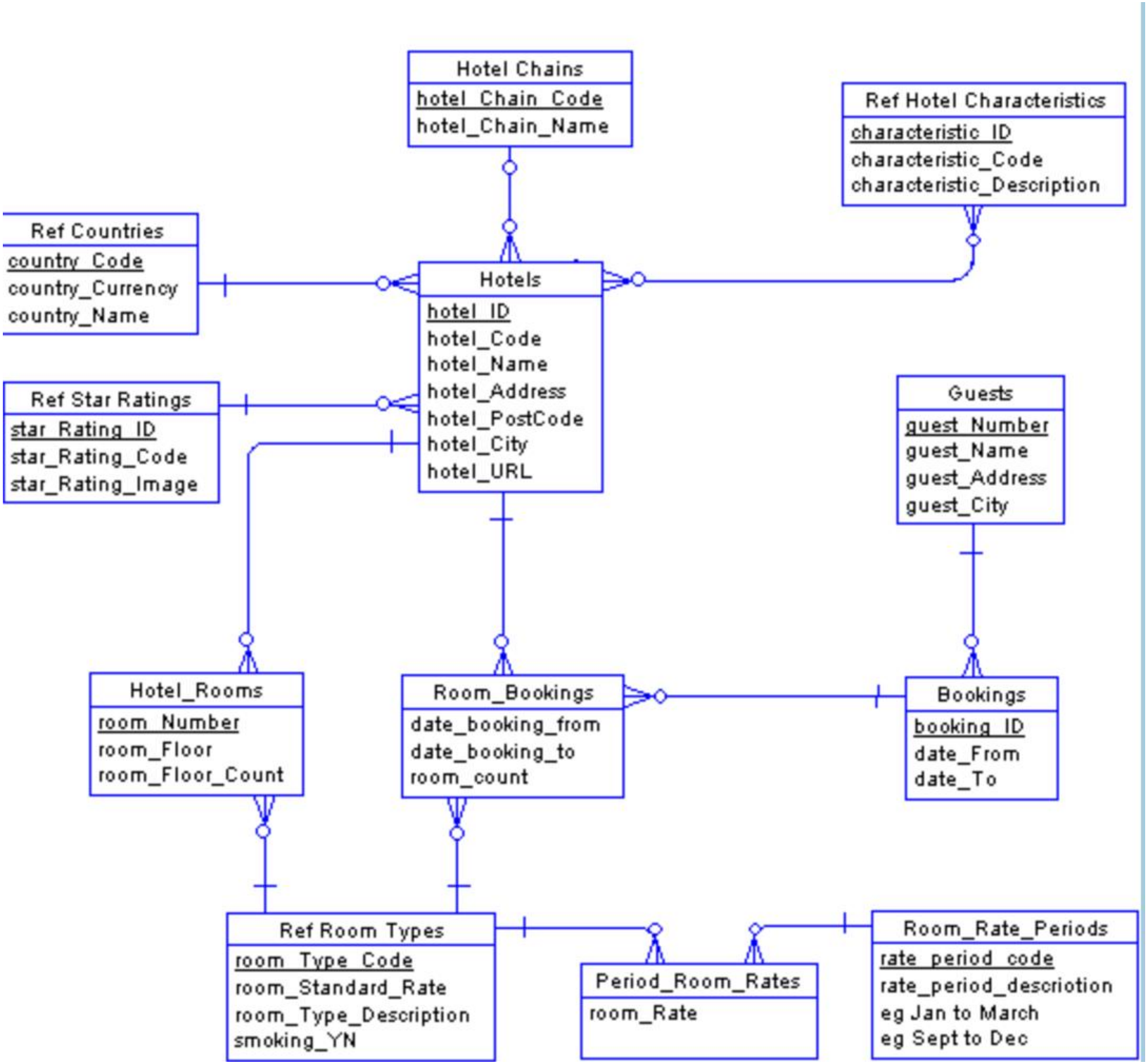


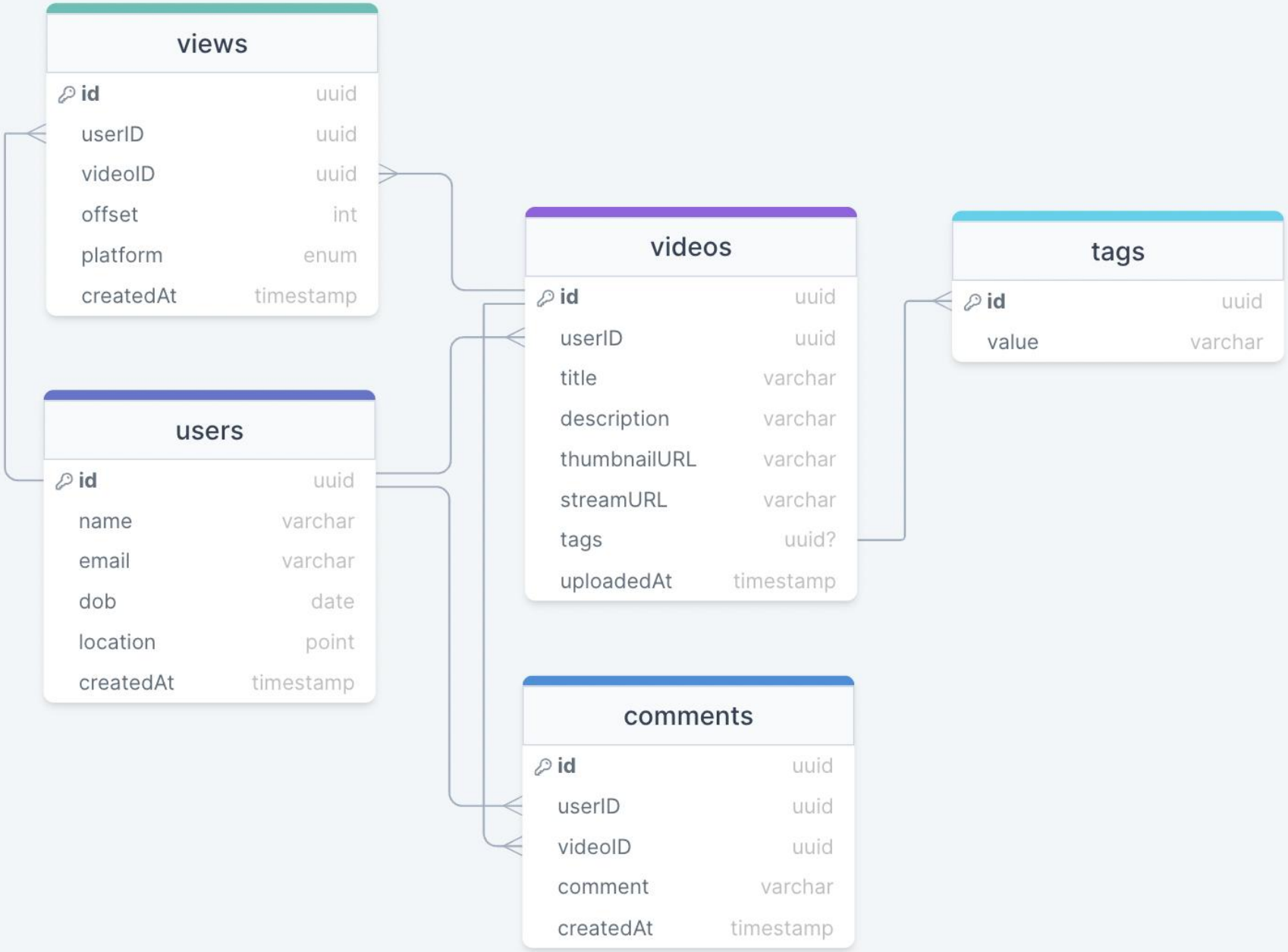
The class diagram of the Amazon Locker service

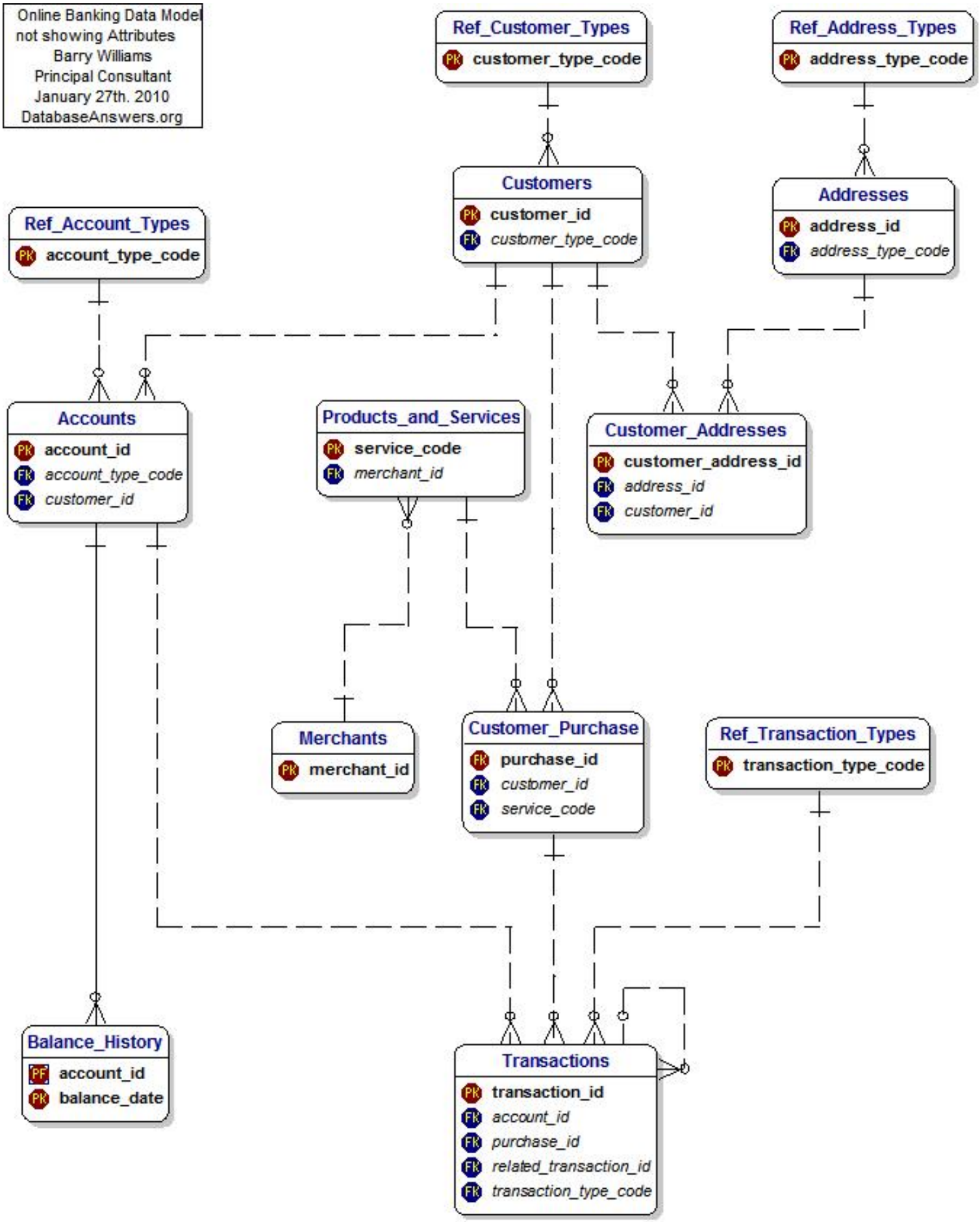

«interface»
EvictionPolicy<Key>
has
Cache<Key, Value>
- evictionPolicy: EvictionPolicy<Key>
- storage : Storage<Key, Value>
- capacity : int
+ put(Key, Value): void
+ get(Key) : Value
has
«interface»
Storage<Key,Value>
DoublyLinkedList
- head: DoublyLinkedListNode
- tail : DoublyLinkedListNode
+ detachNode(DoublyLinkedListNode): void
+ addNodeAtEnd(DoublyLinkedListNode) : void
+ addElementAtEnd(E): DoublyLinkedListNode
+ getFirstNode(): DoublyLinkedListNode
+ getLastNode(): DoublyLinkedListNode
Use
LRUEvictionPolicy<Key>
- mapper: Map<Key, DoublyLinkedListNode
- dll : DoublyLinkedList
+ keyAccessed(Key): void
+ evictKey() : Key
Use
HashMapStorage<Key,Value>
- map: HashMap<Key,Value>
- capacity : int
+ put(Key, Value): void
+ get(Key) : Value
+ remove(Key): void
DoublyLinkedListNode<E>
- next: DoublyLinkedListNode
- prev : DoublyLinkedListNode
- element : E

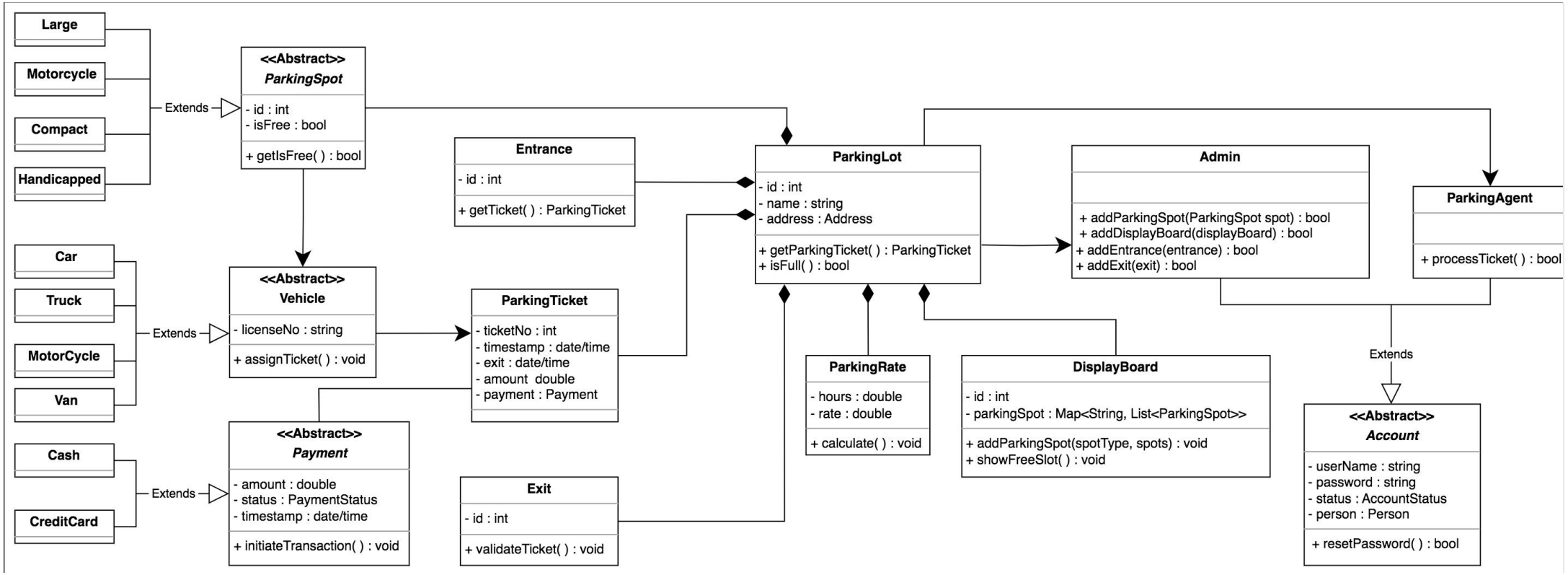

has
TableSchema
schema_members
allow_undefined
validate_row_data(row)
ColumnDataType
data_type
validate(value)
has
SchemaMember
column_name
column_type
required
allow_none
validate_value(val)
Table
table_name
table_schema
indexes
primary_key
table_data
create_index(column_name, inde
insert_data(data)
index_row_data(row_data)
delete_data(key)
scan_table(filters)
filter_on_indexes(filters)
IntDataType
min_value
max_value
validate(value)
StrDataType
max_value
min_value
validate(value)
Index
index_name
primary_key
index_storage
index_row_data(row_data)
get_data(filter)
remove_indexed_val(row_data)
FuzzyIndex
seperator
get_fuzzy_values(row_data)
*
1
DataBase
db_name
tables
create_table(name)
delete_table(name)

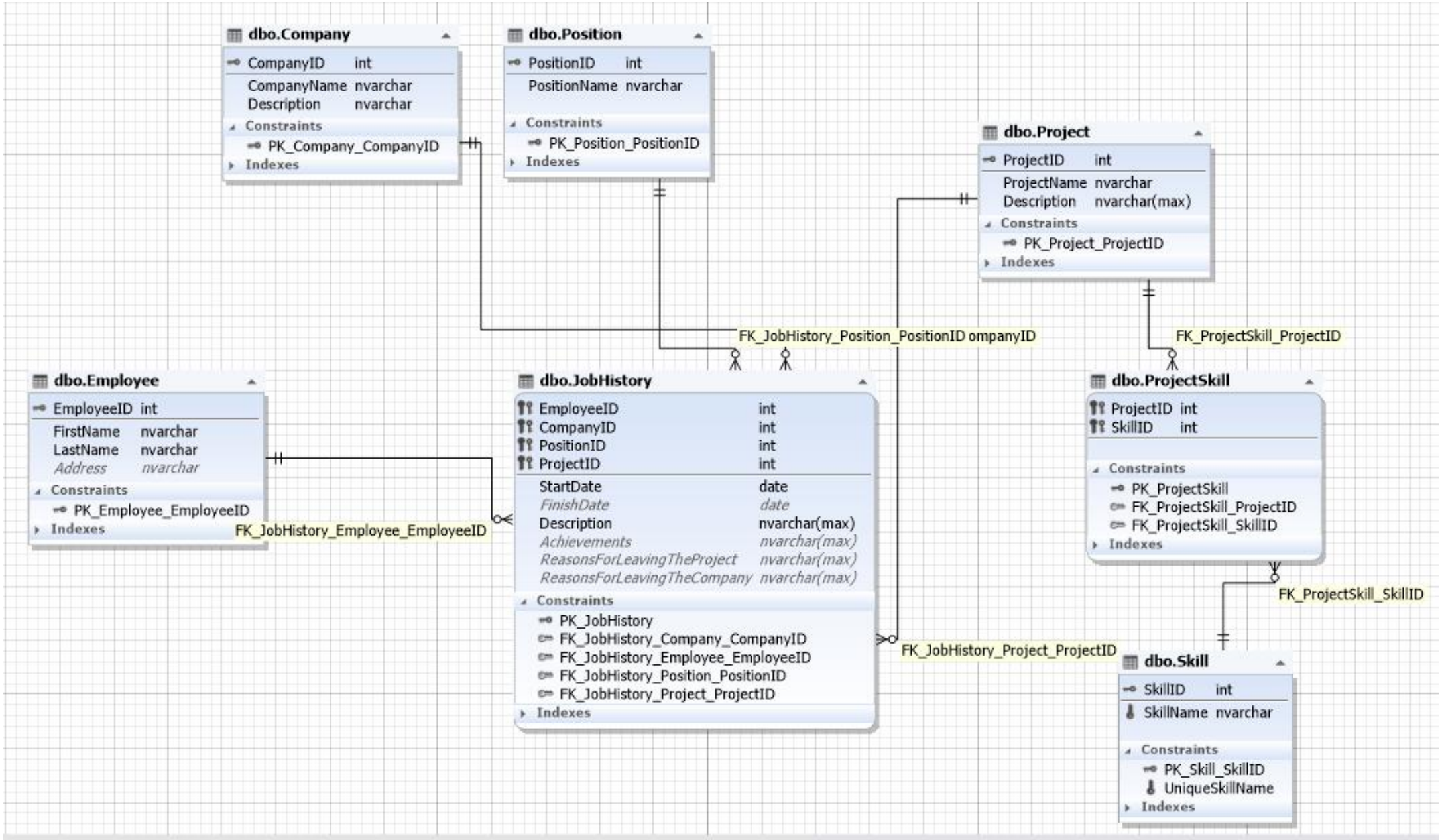

ACCOUNTS
ACCOUNT_ID
ACCOUNT_NAME
ACCOUNT_TYPE_NAME
CUSTOMER_NAME
DATE_ACCOUNT_OPENED
DATE_ACCOUNT_CLOSED
GENERAL_LEDGER_CODES
GENERAL_LEDGER_CODE
GL_CODE_DESCRIPTION
TRANSACTIONS
TRANSACTION_ID
ACCOUNT_ID
CURRENCY_CODE
DEPARTMENT_CODE
GENERAL_LEDGER_CODE
PERIOD_ID
STAFF_ID
TRANSACTION_AMOUNT
CURRENCIES
CURRENCY_CODE
CURRENCY_NAME
PERIODS
PERIOD_ID
PERIOD_NAME
DATE_PERIOD_START
DATE_PERIOD_END
DEPARTMENTS
DEPARTMENT_CODE
DEPARTMENT_NAME
STAFF
STAFF_ID
STAFF_NAME
DATE_JOINED
DATE_LEFT
EMAIL_ADDRESS
PHONE_EXTENSION

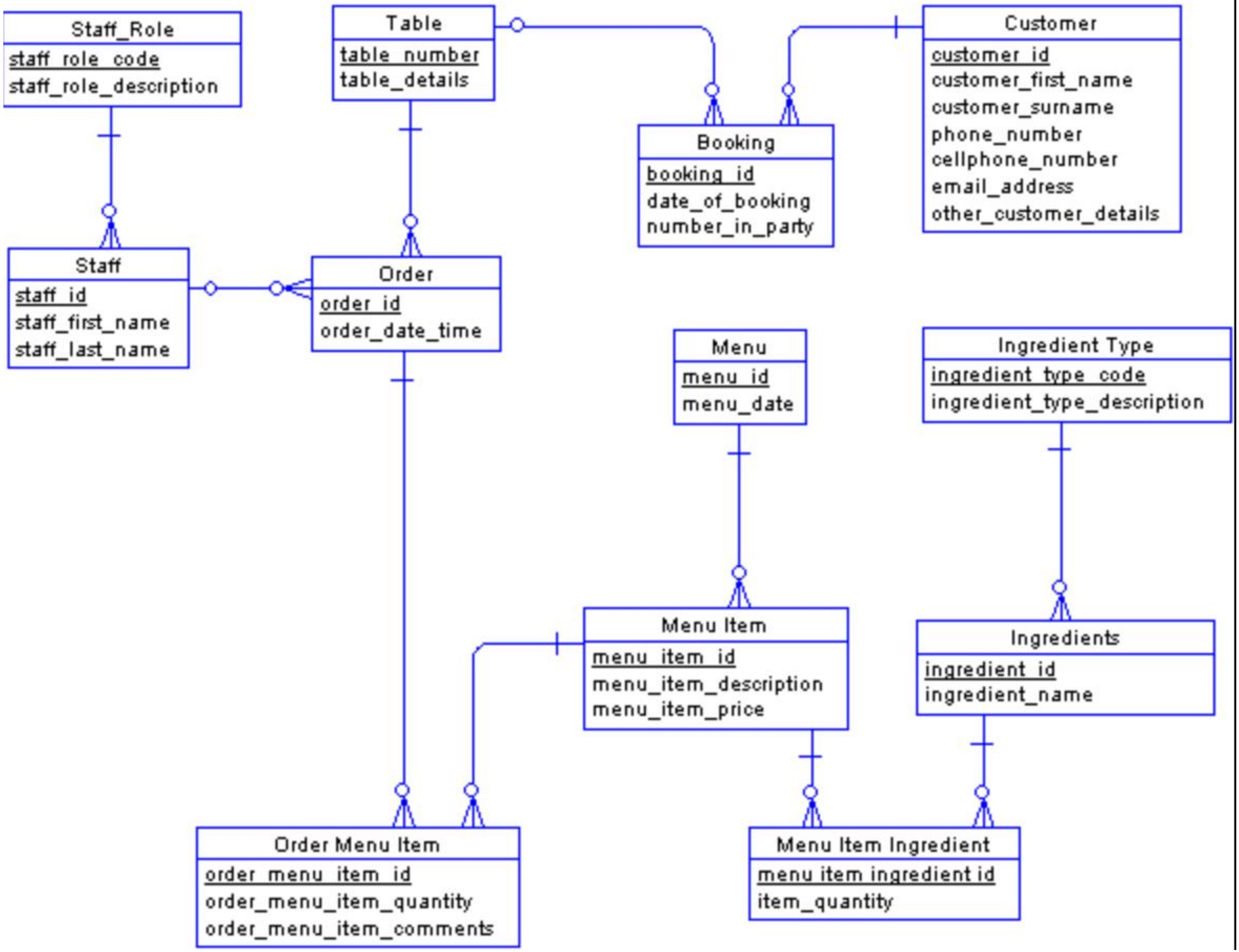

Hotel Chains
hotel_Chain_Code
hotel_Chain_Name
Ref Hotel Characteristics
characteristic_ID
characteristic_Code
characteristic_Description
Ref Countries
country_Code
country_Currency
country_Name
Hotels
hotel_ID
hotel_Code
hotel_Name
hotel_Address
hotel_PostCode
hotel_City
hotel_URL
Ref Star Ratings
star_Rating_ID
star_Rating_Code
star_Rating_Image
Guests
guest_Number
guest_Name
guest_Address
guest_City
Hotel_Rooms
room_Number
room_Floor
room_Floor_Count
Room_Bookings
date_booking_from
date_booking_to
room_count
Bookings
booking_ID
date_From
date_To
Ref Room Types
room_Type_Code
room_Standard_Rate
room_Type_Description
smoking_YN
Period_Room_Rates
room_Rate
Room_Rate_Periods
rate_period_code
rate_period_descriotion
eg Jan to March
eg Sept to Dec

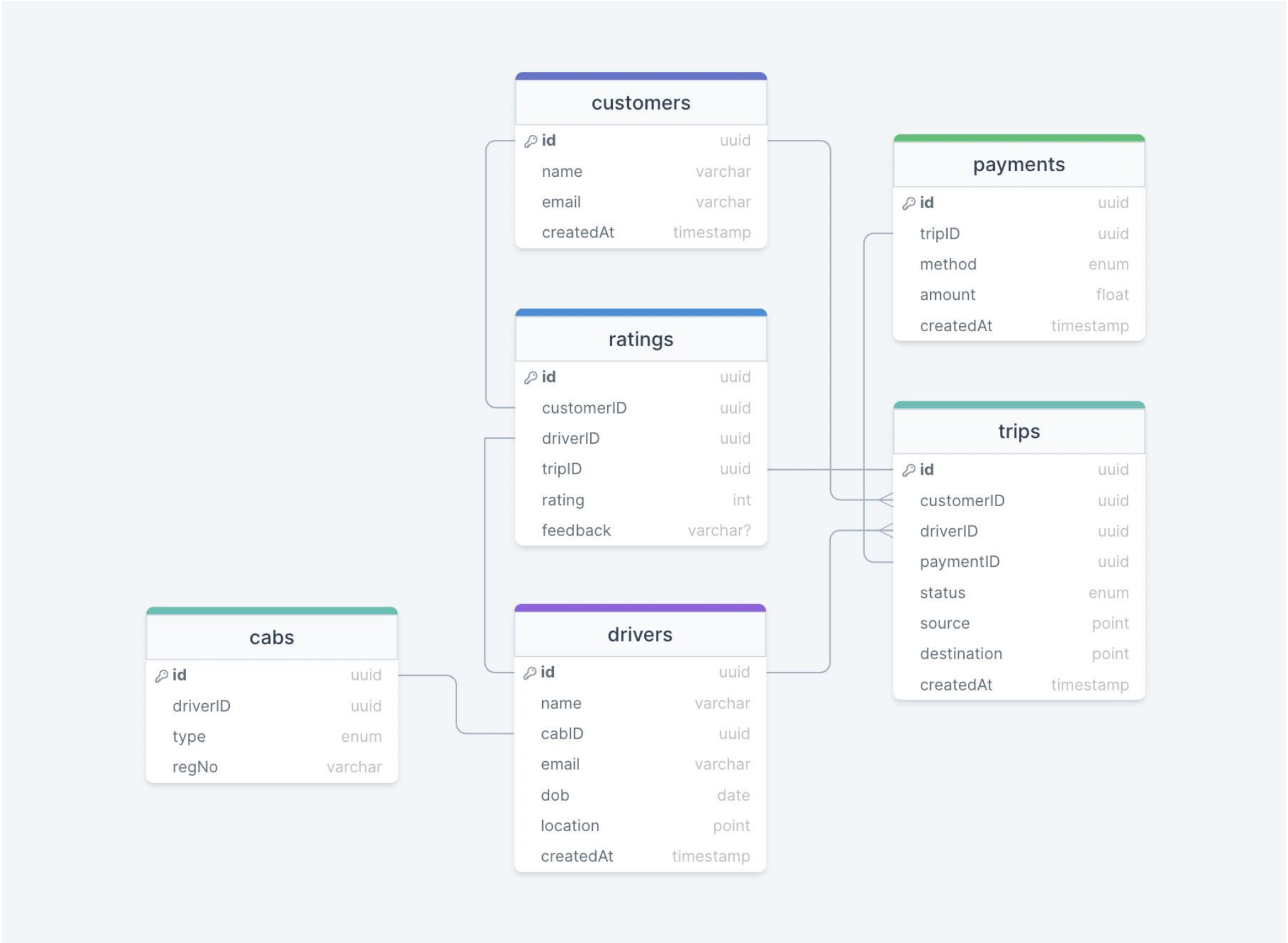

views
id uuid
userID uuid
videoID uuid
offset int
platform enum
createdAt timestamp
users
id uuid
name varchar
email varchar
dob date
location point
createdAt timestamp
videos
id uuid
userID uuid
title varchar
description varchar
thumbnailURL varchar
streamURL varchar
tags uuid?
uploadedAt timestamp
tags
id uuid
value varchar
comments
id uuid
userID uuid
videoID uuid
comment varchar
createdAt timestamp

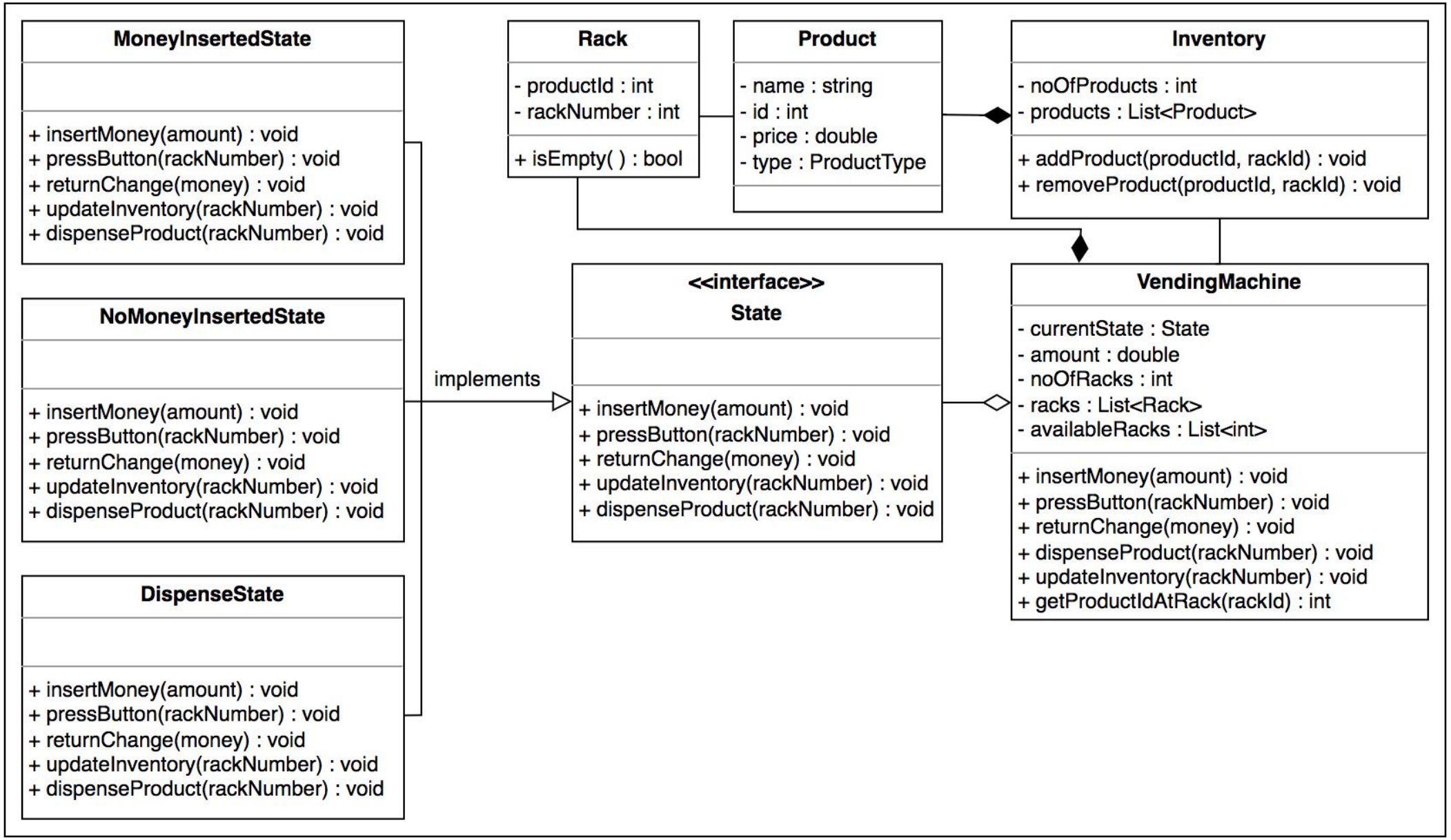

Online Banking Data Model
not showing Attributes
Barry Williams
Principal Consultant
January 27th. 2010
DatabaseAnswers.org
Ref_Customer_Types
customer_type_code
Ref_Address_Types
address_type_code
Customers
customer_id
customer_type_code
Addresses
address_id
address_type_code
Ref_Account_Types
account_type_code
Accounts
account_id
account_type_code
customer_id
Products_and_Services
service_code
merchant_id
Customer_Addresses
customer_address_id
address_id
customer_id
Merchants
merchant_id
Customer_Purchase
purchase_id
customer_id
service_code
Ref_Transaction_Types
transaction_type_code
Balance_History
account_id
balance_date
Transactions
transaction_id
account_id
purchase_id
related_transaction_id
transaction_type_code


Large
Motorcycle
Compact
Handicapped
Extends
<<Abstract>>
ParkingSpot
- id : int
- isFree : bool
+ getIsFree( ) : bool
Entrance
- id : int
+ getTicket( ) : ParkingTicket
ParkingLot
- id : int
- name : string
- address : Address
+ getParkingTicket( ) : ParkingTicket
+ isFull( ) : bool
Admin
+ addParkingSpot(ParkingSpot spot) : bool
+ addDisplayBoard(displayBoard) : bool
+ addEntrance(entrance) : bool
+ addExit(exit) : bool
ParkingAgent
+ processTicket( ) : bool
Car
Truck
MotorCycle
Van
Extends
<<Abstract>>
Vehicle
- licenseNo : string
+ assignTicket( ) : void
ParkingTicket
- ticketNo : int
- timestamp : date/time
- exit : date/time
- amount double
- payment : Payment
ParkingRate
- hours : double
- rate : double
+ calculate( ) : void
DisplayBoard
- id : int
- parkingSpot : Map<String, List<ParkingSpot>>
+ addParkingSpot(spotType, spots) : void
+ showFreeSlot( ) : void
Extends
Cash
CreditCard
Extends
<<Abstract>>
Payment
- amount : double
- status : PaymentStatus
- timestamp : date/time
+ initiateTransaction( ) : void
Exit
- id : int
+ validateTicket( ) : void
<<Abstract>>
Account
- userName : string
- password : string
- status : AccountStatus
- person : Person
+ resetPassword( ) : bool

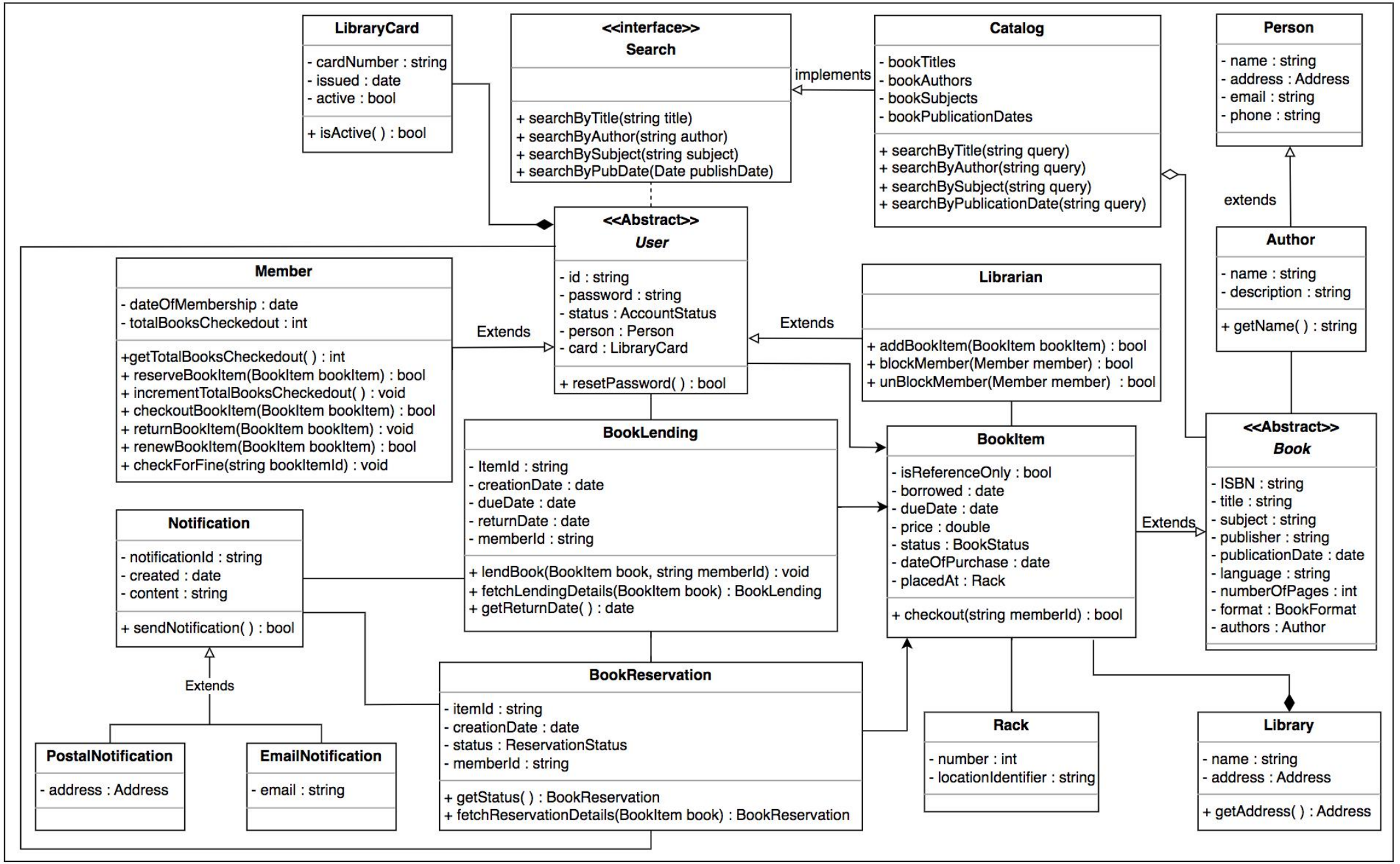

dbo.Company
CompanyID int
CompanyName nvarchar
Description nvarchar
Constraints
PK_Company_CompanyID
Indexes
dbo.Position
PositionID int
PositionName nvarchar
Constraints
PK_Position_PositionID
Indexes
dbo.Project
ProjectID int
ProjectName nvarchar
Description nvarchar(max)
Constraints
PK_Project_ProjectID
Indexes
FK_JobHistory_Position_PositionID ompanyID
FK_ProjectSkill_ProjectID
dbo.Employee
EmployeeID int
FirstName nvarchar
LastName nvarchar
Address nvarchar
Constraints
PK_Employee_EmployeeID
Indexes
FK_JobHistory_Employee_EmployeeID
dbo.JobHistory
EmployeeID int
CompanyID int
PositionID int
ProjectID int
StartDate date
FinishDate date
Description nvarchar(max)
Achievements nvarchar(max)
ReasonsForLeavingTheProject nvarchar(max)
ReasonsForLeavingTheCompany nvarchar(max)
Constraints
PK_JobHistory
FK_JobHistory_Company_CompanyID
FK_JobHistory_Employee_EmployeeID
FK_JobHistory_Position_PositionID
FK_JobHistory_Project_ProjectID
Indexes
dbo.ProjectSkill
ProjectID int
SkillID int
Constraints
PK_ProjectSkill
FK_ProjectSkill_ProjectID
FK_ProjectSkill_SkillID
Indexes
FK_ProjectSkill_SkillID
FK_JobHistory_Project_ProjectID
dbo.Skill
SkillID int
SkillName nvarchar
Constraints
PK_Skill_SkillID
UniqueSkillName
Indexes


Staff_Role
staff_role_code
staff_role_description
Table
table_number
table_details
Booking
booking_id
date_of_booking
number_in_party
Customer
customer_id
customer_first_name
customer_surname
phone_number
cellphone_number
email_address
other_customer_details
Staff
staff_id
staff_first_name
staff_last_name
Order
order_id
order_date_time
Menu
menu_id
menu_date
Ingredient Type
ingredient_type_code
ingredient_type_description
Menu Item
menu_item_id
menu_item_description
menu_item_price
Ingredients
ingredient_id
ingredient_name
Order Menu Item
order_menu_item_id
order_menu_item_quantity
order_menu_item_comments
Menu Item Ingredient
menu item ingredient id
item_quantity

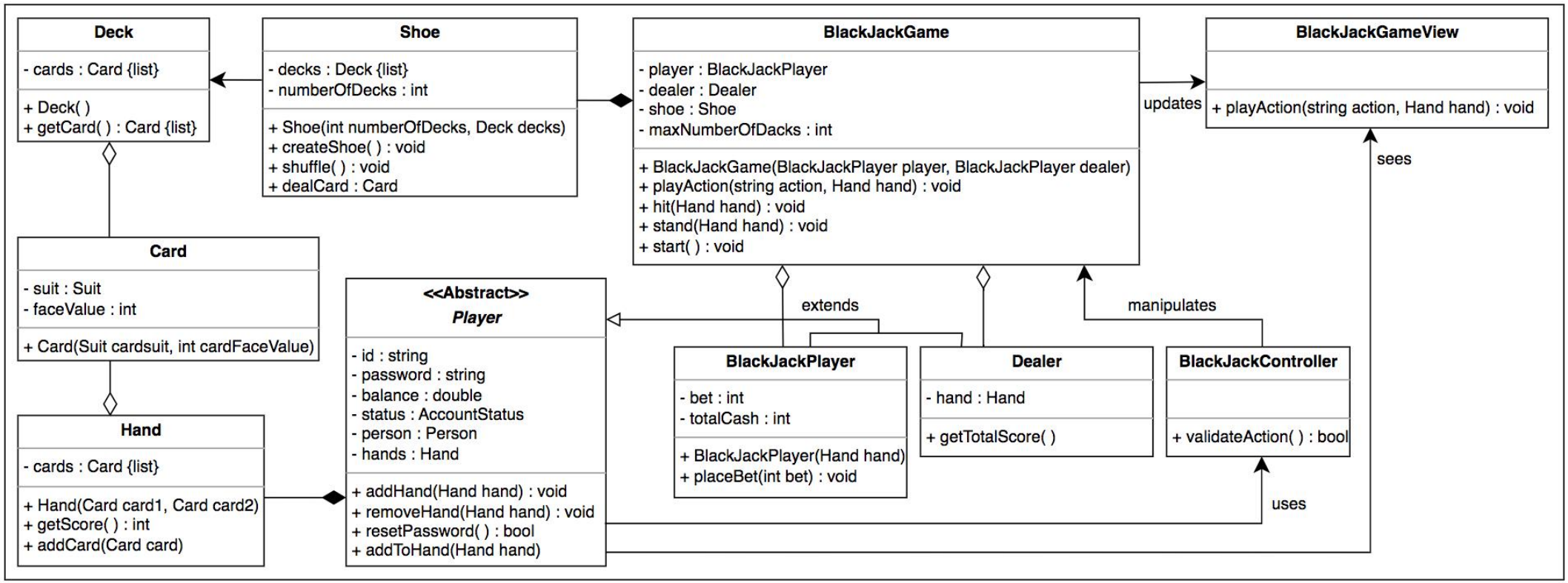

customers
id uuid
name varchar
email varchar
createdAt timestamp
payments
id uuid
tripID uuid
method enum
amount float
createdAt timestamp
ratings
id uuid
customerID uuid
driverID uuid
tripID uuid
rating int
feedback varchar?
trips
id uuid
customerID uuid
driverID uuid
paymentID uuid
status enum
source point
destination point
createdAt timestamp
cabs
id uuid
driverID uuid
type enum
regNo varchar
drivers
id uuid
name varchar
cabID uuid
email varchar
dob date
location point
createdAt timestamp



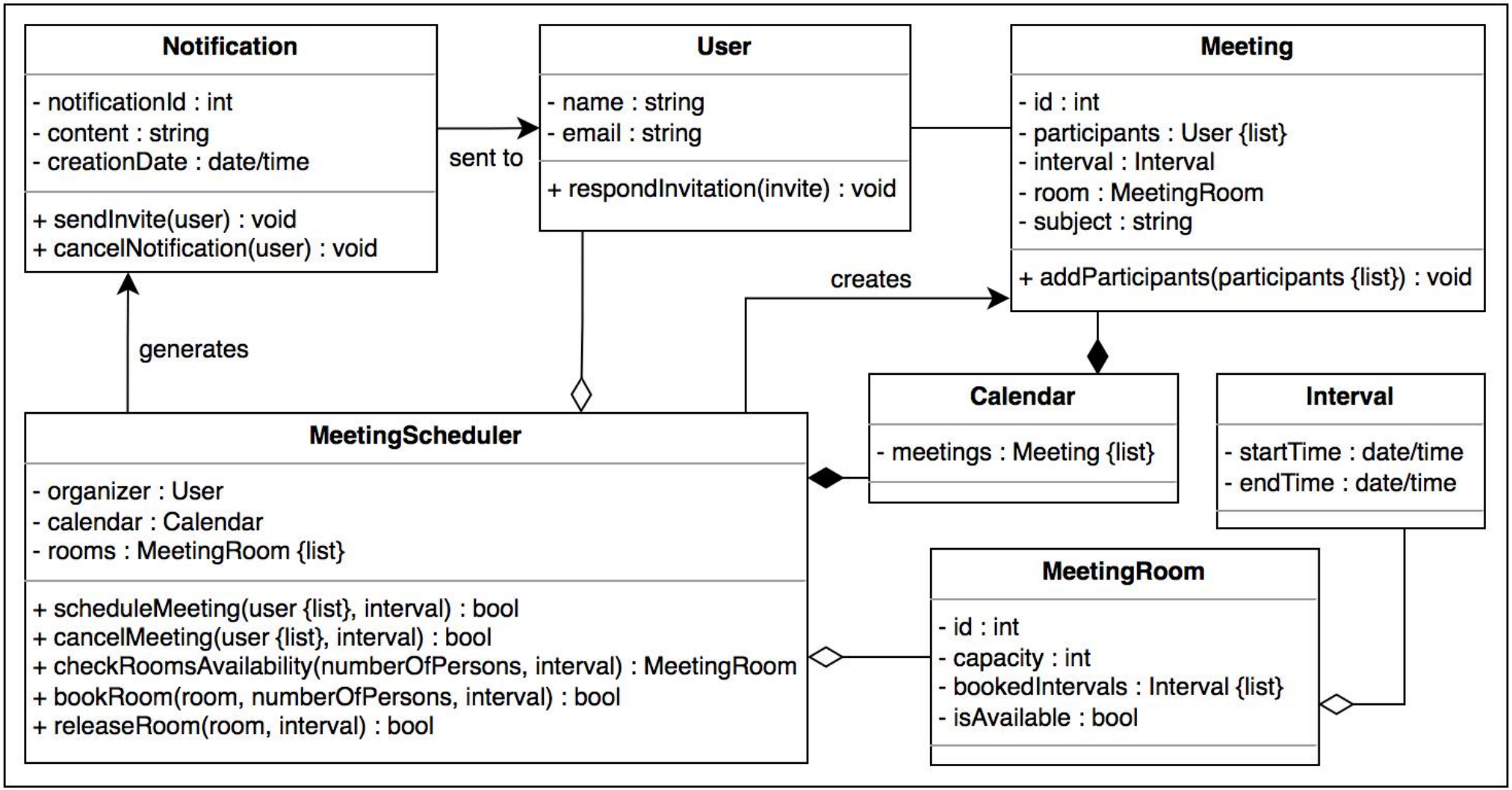
The class diagram of the vending machine



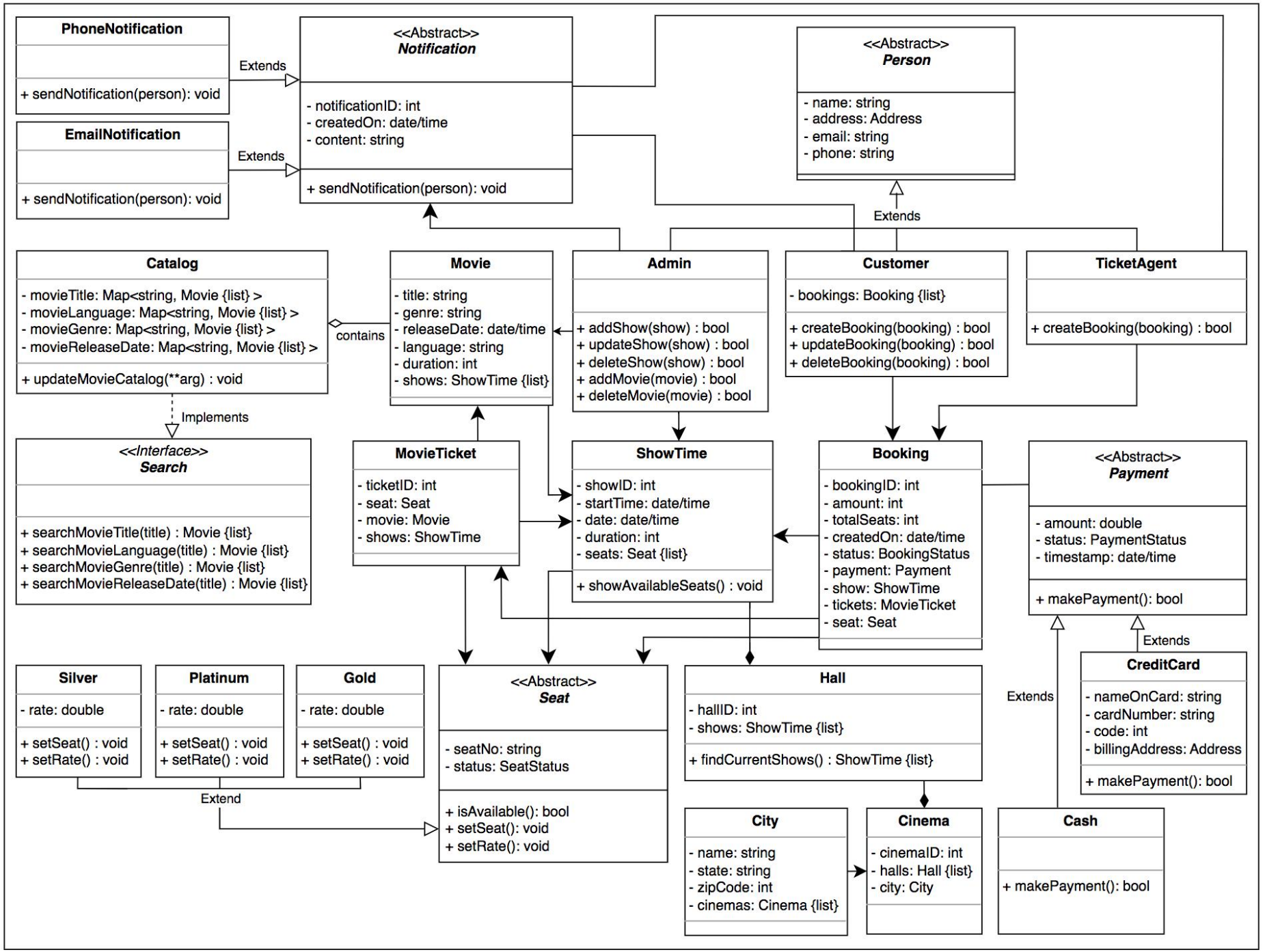
The class diagram of the library management system



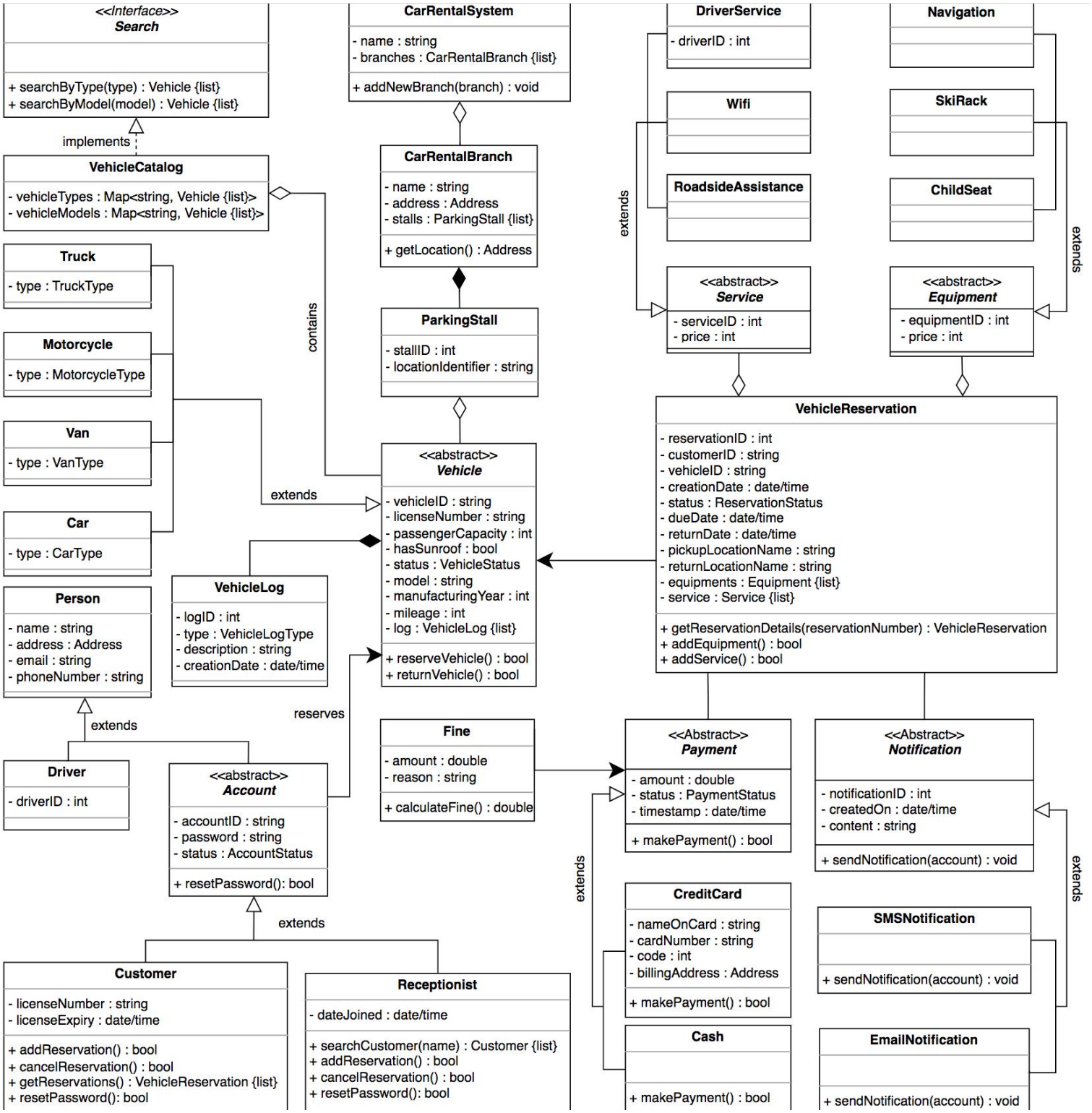
The class diagram of the Blackjack game

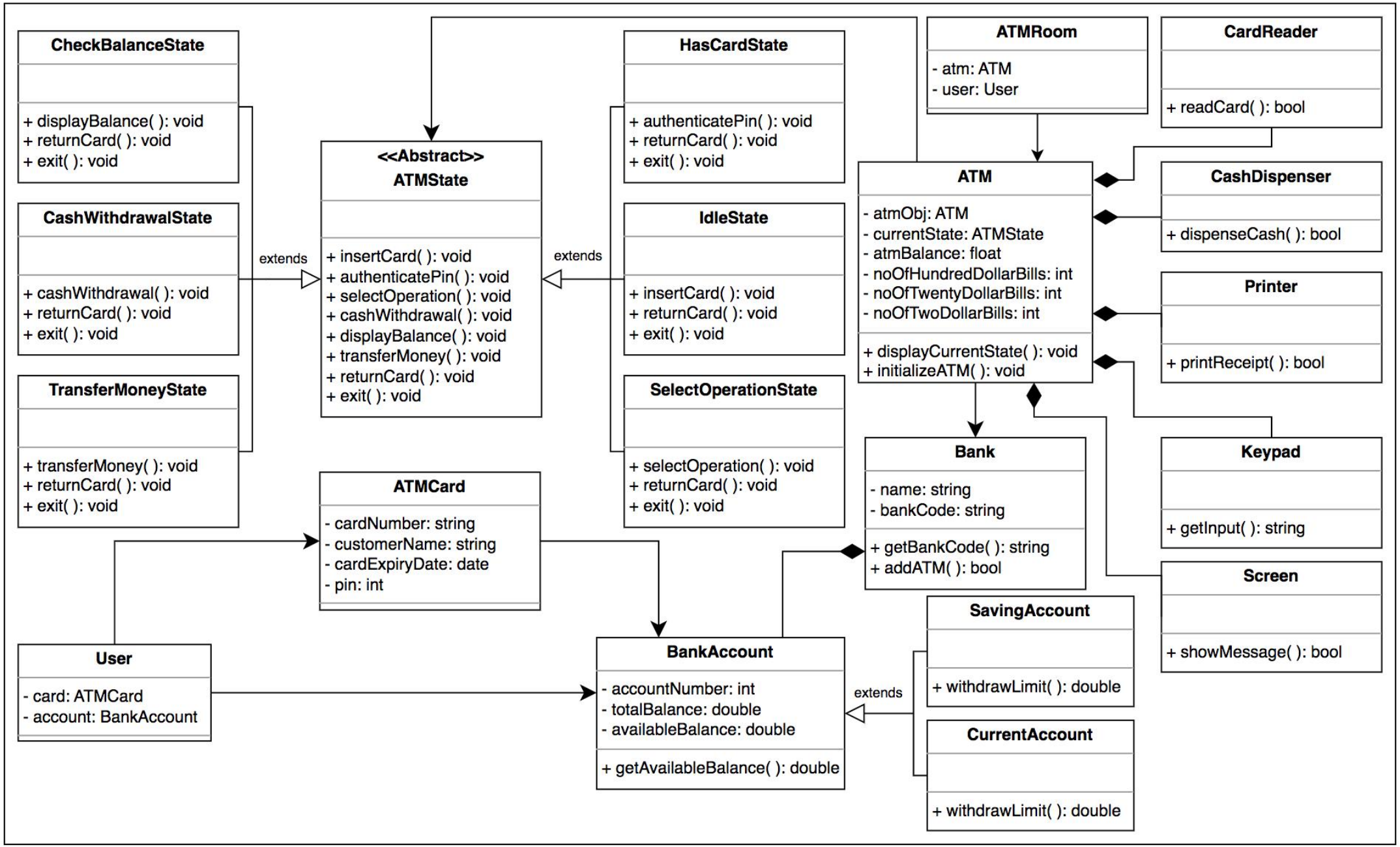


The class diagram of the meeting scheduler

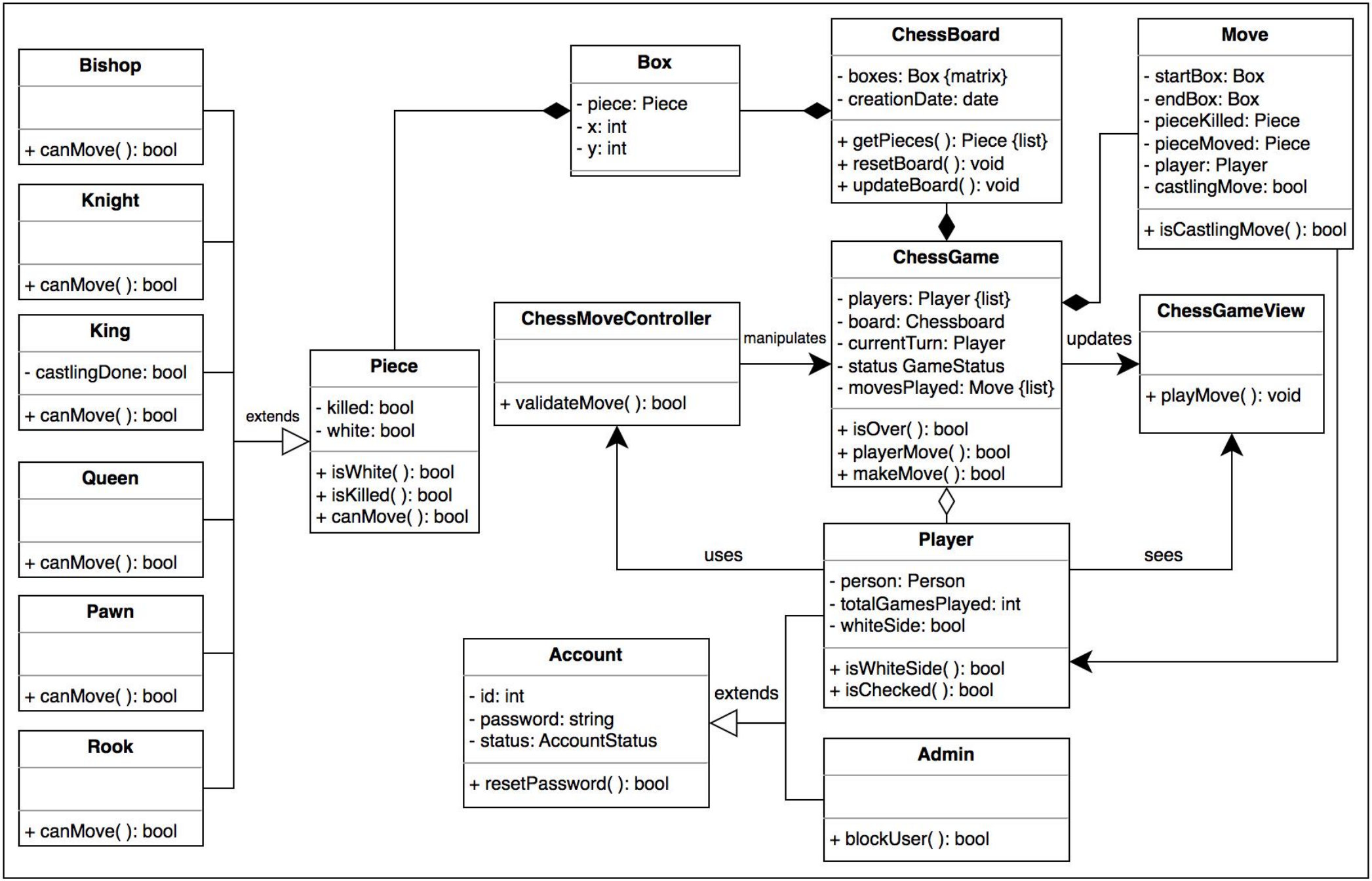


The class diagram of the movie ticket booking system

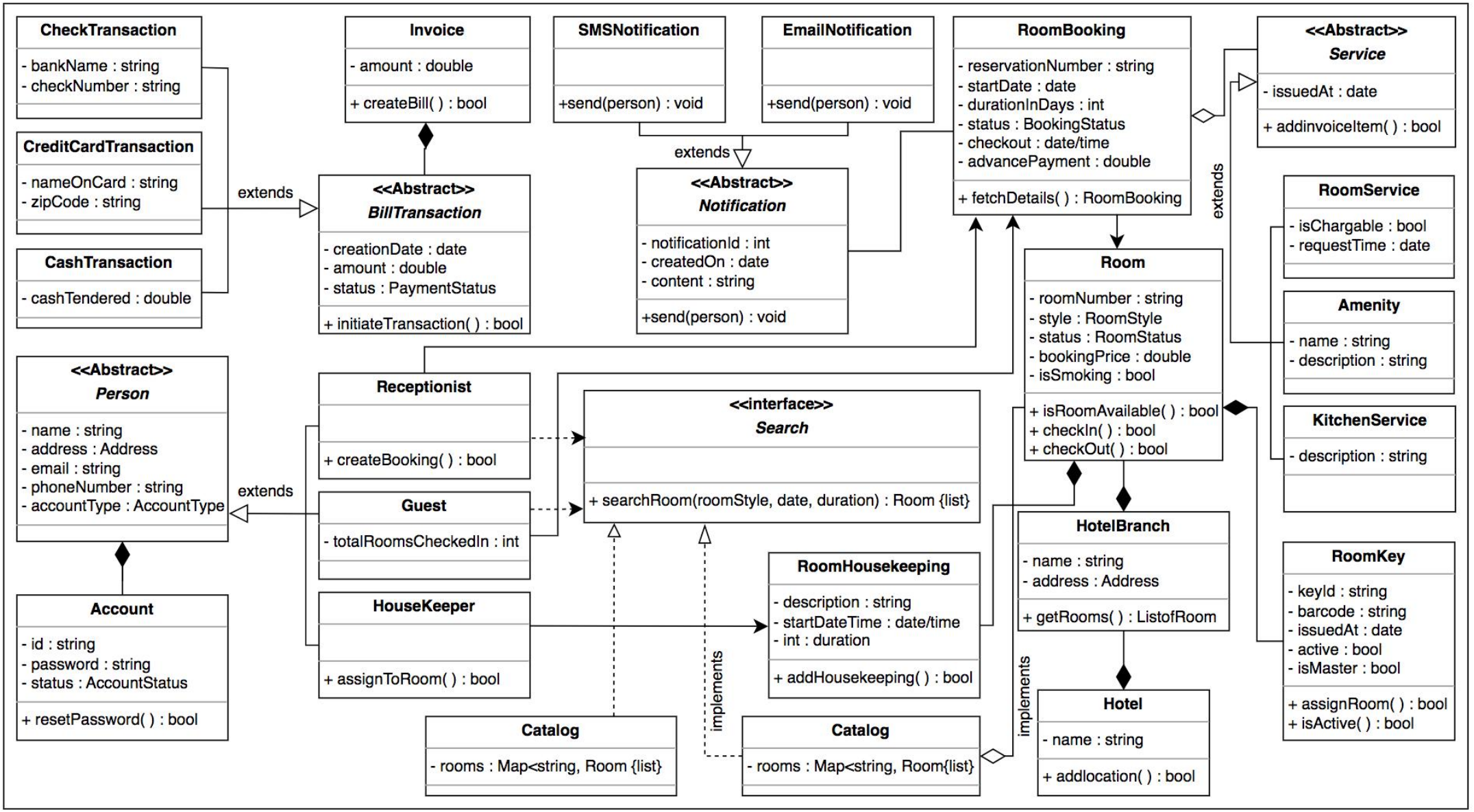

<<Interface>>
Search
+ searchByType(type) : Vehicle {list}
+ searchByModel(model) : Vehicle {list}
implements
VehicleCatalog
- vehicleTypes : Map<string, Vehicle {list}>
- vehicleModels : Map<string, Vehicle {list}>
CarRentalSystem
- name : string
- branches : CarRentalBranch {list}
+ addNewBranch(branch) : void
CarRentalBranch
- name : string
- address : Address
- stalls : ParkingStall {list}
+ getLocation() : Address
ParkingStall
- stallID : int
- locationIdentifier : string
contains
Truck
- type : TruckType
Motorcycle
- type : MotorcycleType
Van
- type : VanType
Car
- type : CarType
extends
<<abstract>>
Vehicle
- vehicleID : string
- licenseNumber : string
- passengerCapacity : int
- hasSunroof : bool
- status : VehicleStatus
- model : string
- manufacturingYear : int
- mileage : int
- log : VehicleLog {list}
+ reserveVehicle() : bool
+ returnVehicle() : bool
VehicleLog
- logID : int
- type : VehicleLogType
- description : string
- creationDate : date/time
Person
- name : string
- address : Address
- email : string
- phoneNumber : string
extends
Driver
- driverID : int
reserves
<<abstract>>
Account
- accountID : string
- password : string
- status : AccountStatus
+ resetPassword(): bool
extends
Customer
- licenseNumber : string
- licenseExpiry : date/time
+ addReservation() : bool
+ cancelReservation() : bool
+ getReservations() : VehicleReservation {list}
+ resetPassword(): bool
Receptionist
- dateJoined : date/time
+ searchCustomer(name) : Customer {list}
+ addReservation() : bool
+ cancelReservation() : bool
+ resetPassword(): bool
Fine
- amount : double
- reason : string
+ calculateFine() : double
DriverService
- driverID : int
Wifi
RoadsideAssistance
extends
<<abstract>>
Service
- serviceID : int
- price : int
Navigation
SkiRack
ChildSeat
extends
<<abstract>>
Equipment
- equipmentID : int
- price : int
VehicleReservation
- reservationID : int
- customerID : string
- vehicleID : string
- creationDate : date/time
- status : ReservationStatus
- dueDate : date/time
- returnDate : date/time
- pickupLocationName : string
- returnLocationName : string
- equipments : Equipment {list}
- service : Service {list}
+ getReservationDetails(reservationNumber) : VehicleReservation
+ addEquipment() : bool
+ addService() : bool
<<Abstract>>
Payment
- amount : double
- status : PaymentStatus
- timestamp : date/time
+ makePayment() : bool
extends
CreditCard
- nameOnCard : string
- cardNumber : string
- code : int
- billingAddress : Address
+ makePayment() : bool
Cash
+ makePayment() : bool
<<Abstract>>
Notification
- notificationID : int
- createdOn : date/time
- content : string
+ sendNotification(account) : void
extends
SMSNotification
+ sendNotification(account) : void
EmailNotification
+ sendNotification(account) : void



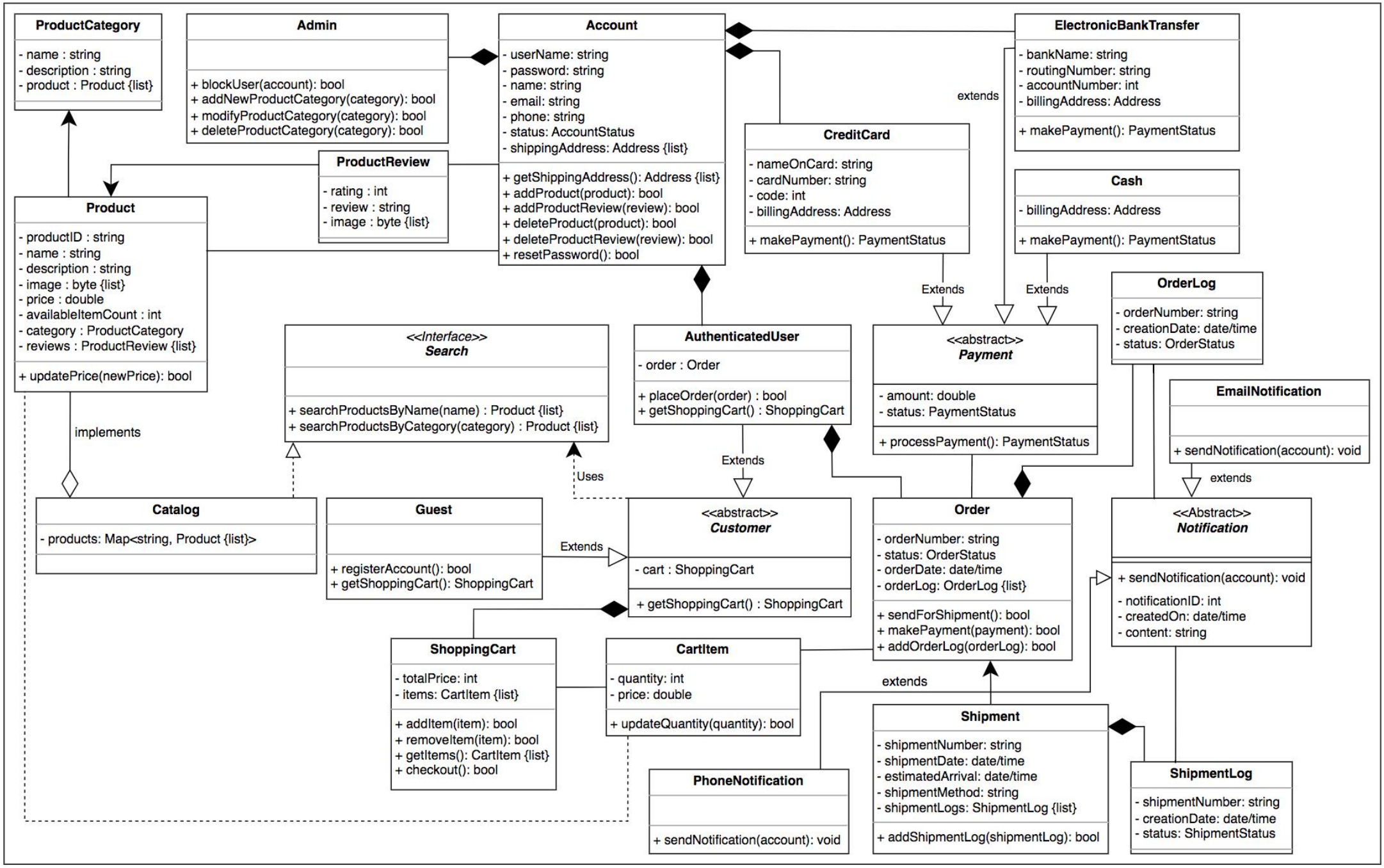
The class diagram of the ATM design problem



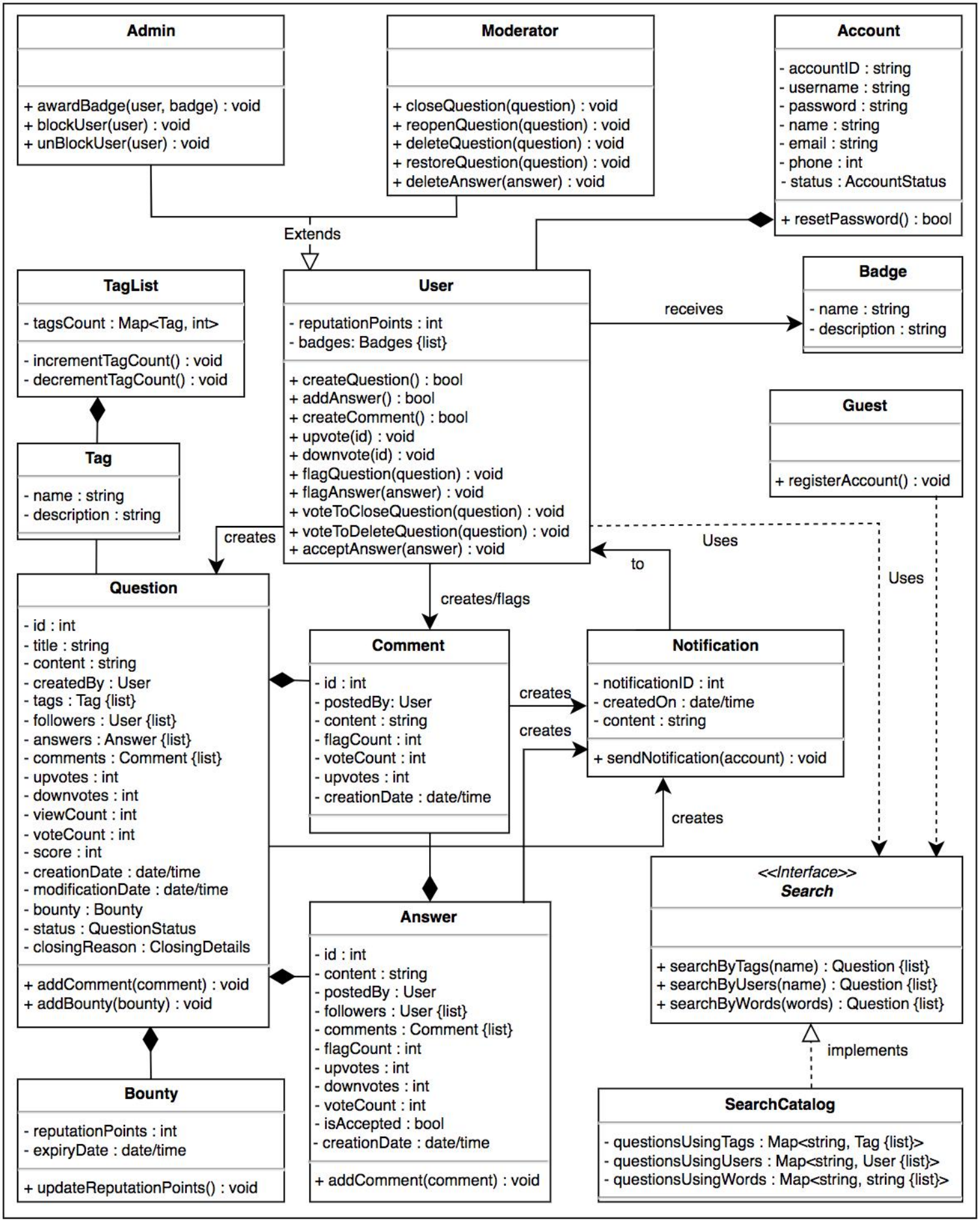
The class diagram of the chess game

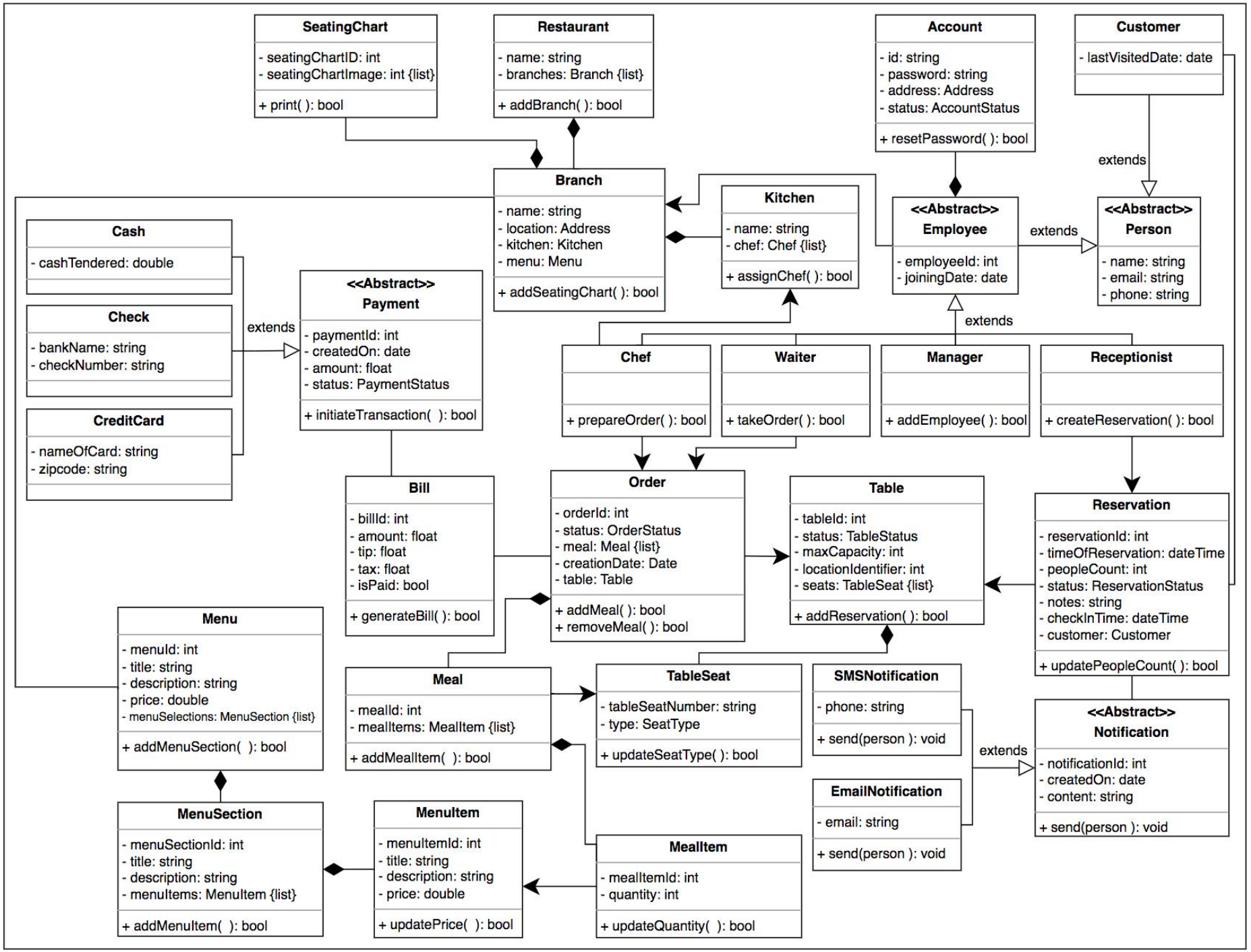


The class diagram of hotel management system

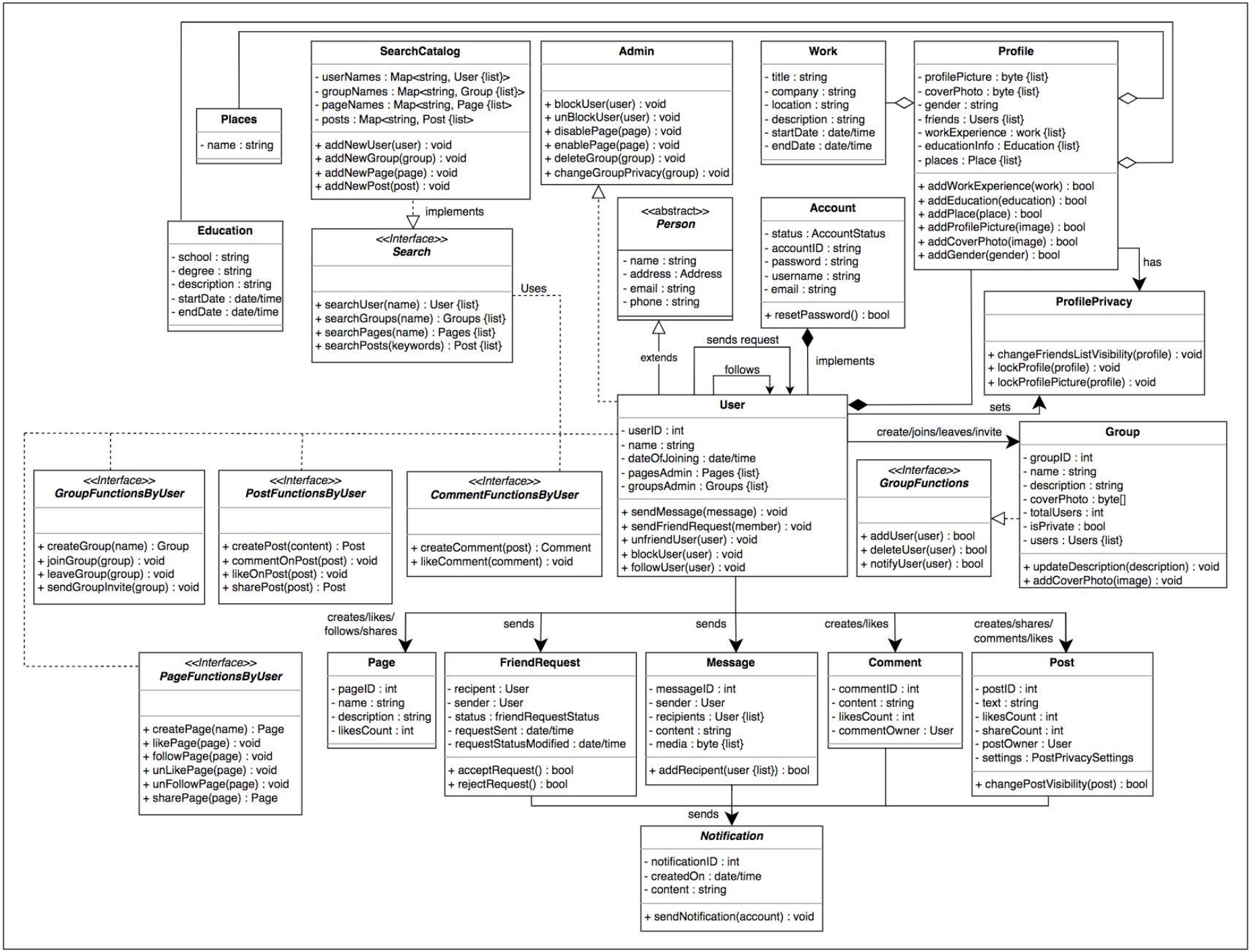


The class diagram of Amazon

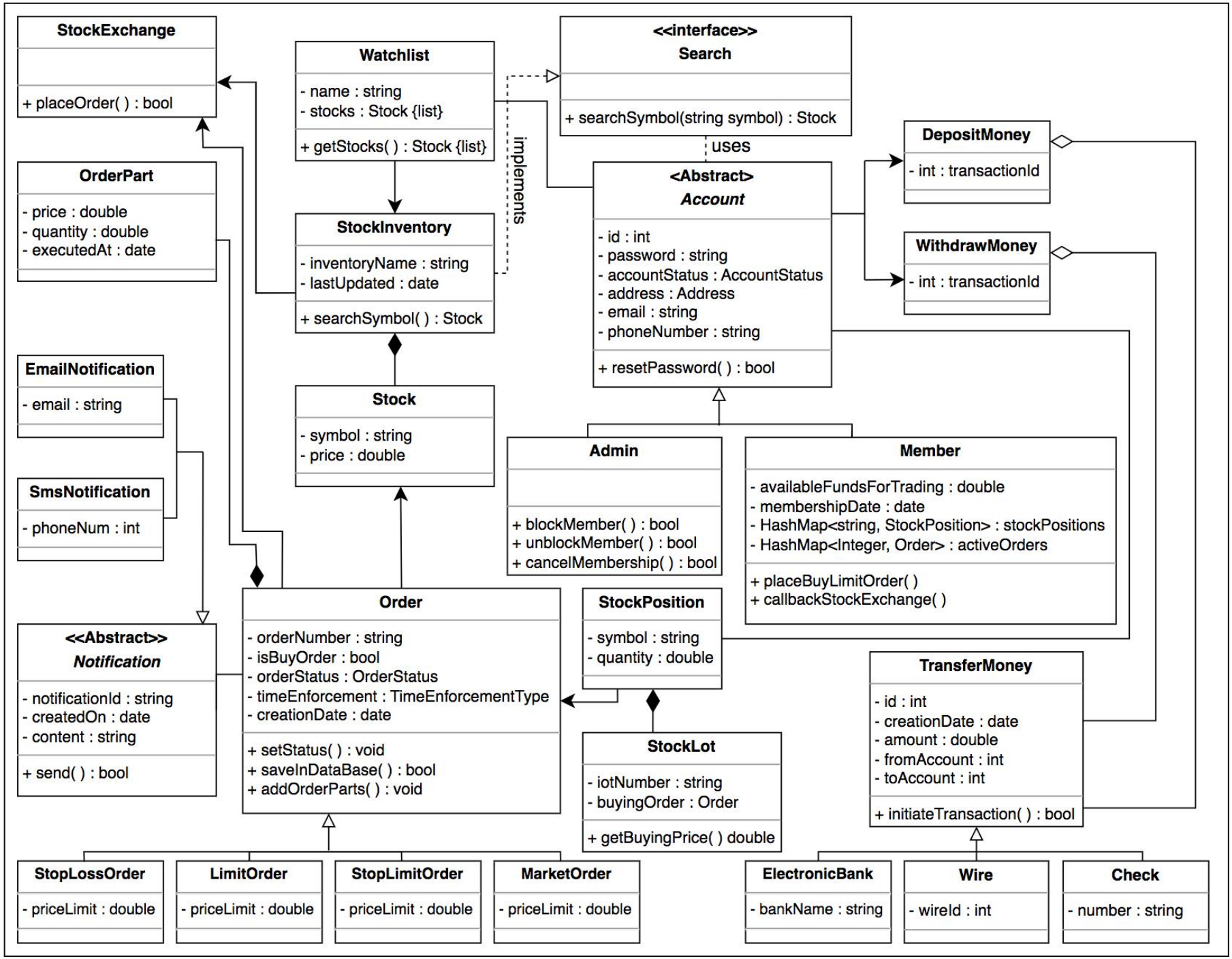

Admin
+ awardBadge(user, badge) : void
+ blockUser(user) : void
+ unBlockUser(user) : void
Moderator
+ closeQuestion(question) : void
+ reopenQuestion(question) : void
+ deleteQuestion(question) : void
+ restoreQuestion(question) : void
+ deleteAnswer(answer) : void
Account
- accountID : string
- username : string
- password : string
- name : string
- email : string
- phone : int
- status : AccountStatus
+ resetPassword() : bool
Extends
TagList
- tagsCount : Map<Tag, int>
- incrementTagCount() : void
- decrementTagCount() : void
User
- reputationPoints : int
- badges: Badges {list}
+ createQuestion() : bool
+ addAnswer() : bool
+ createComment() : bool
+ upvote(id) : void
+ downvote(id) : void
+ flagQuestion(question) : void
+ flagAnswer(answer) : void
+ voteToCloseQuestion(question) : void
+ voteToDeleteQuestion(question) : void
+ acceptAnswer(answer) : void
receives
Badge
- name : string
- description : string
Tag
- name : string
- description : string
Guest
+ registerAccount() : void
creates
Uses
to
Uses
Question
- id : int
- title : string
- content : string
- createdBy : User
- tags : Tag {list}
- followers : User {list}
- answers : Answer {list}
- comments : Comment {list}
- upvotes : int
- downvotes : int
- viewCount : int
- voteCount : int
- score : int
- creationDate : date/time
- modificationDate : date/time
- bounty : Bounty
- status : QuestionStatus
- closingReason : ClosingDetails
+ addComment(comment) : void
+ addBounty(bounty) : void
creates/flags
Comment
- id : int
- postedBy: User
- content : string
- flagCount : int
- voteCount : int
- upvotes : int
- creationDate : date/time
creates
creates
Notification
- notificationID : int
- createdOn : date/time
- content : string
+ sendNotification(account) : void
creates
<<Interface>>
Search
+ searchByTags(name) : Question {list}
+ searchByUsers(name) : Question {list}
+ searchByWords(words) : Question {list}
Answer
- id : int
- content : string
- postedBy : User
- followers : User {list}
- comments : Comment {list}
- flagCount : int
- upvotes : int
- downvotes : int
- voteCount : int
- isAccepted : bool
- creationDate : date/time
+ addComment(comment) : void
implements
Bounty
- reputationPoints : int
- expiryDate : date/time
+ updateReputationPoints() : void
SearchCatalog
- questionsUsingTags : Map<string, Tag {list}>
- questionsUsingUsers : Map<string, User {list}>
- questionsUsingWords : Map<string, string {list}>



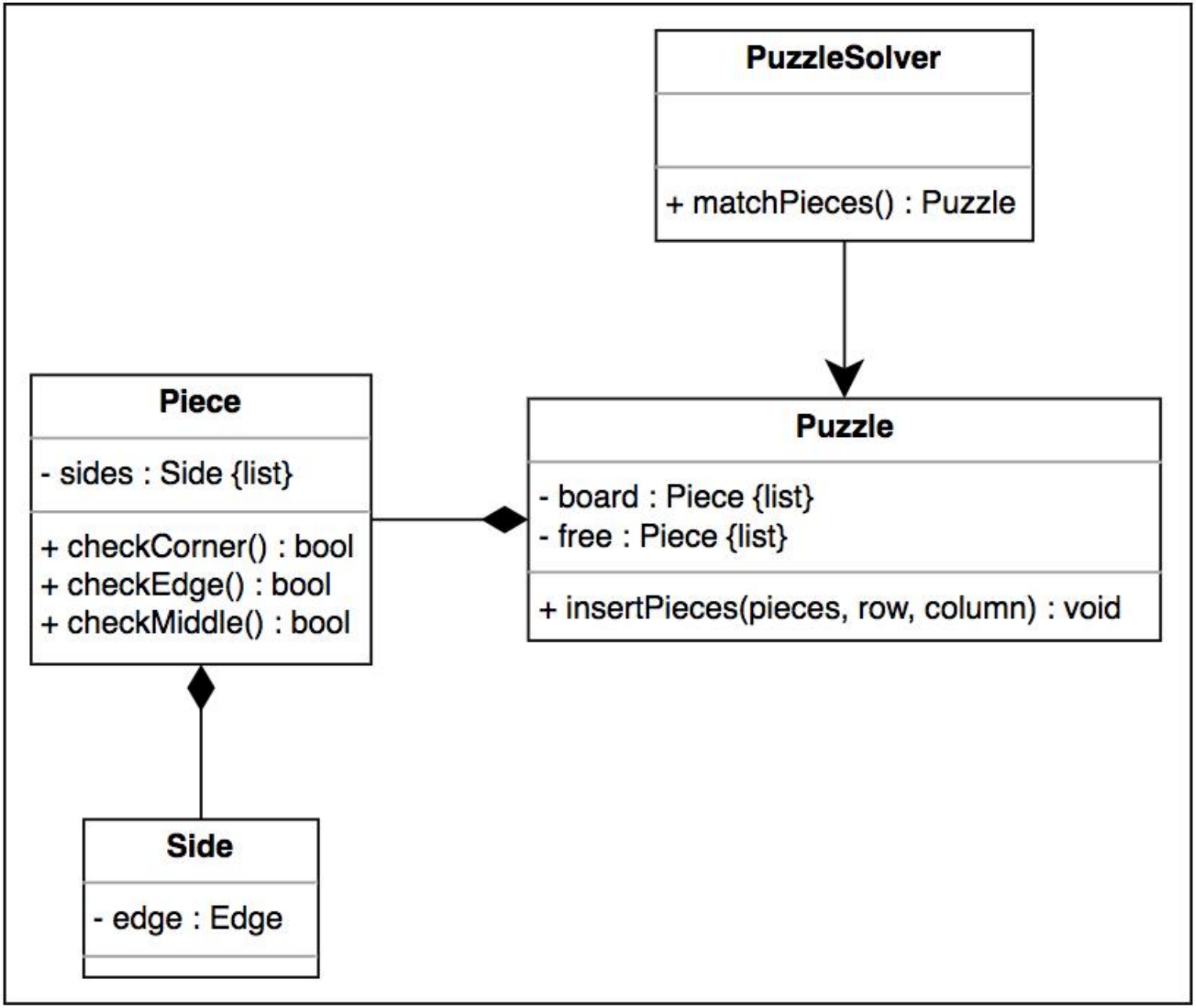
The class diagram of restaurant management system



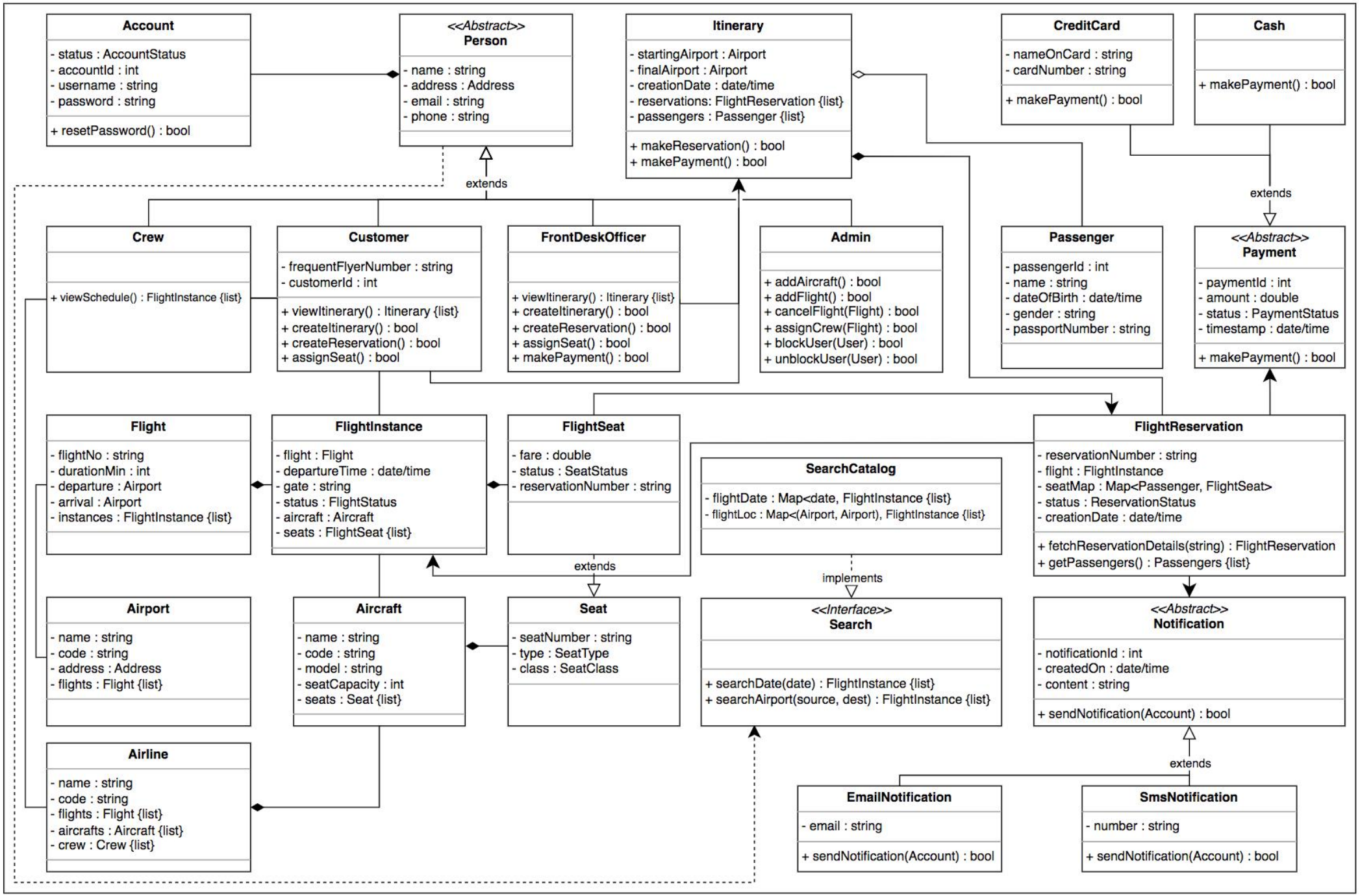
The class diagram of Facebook



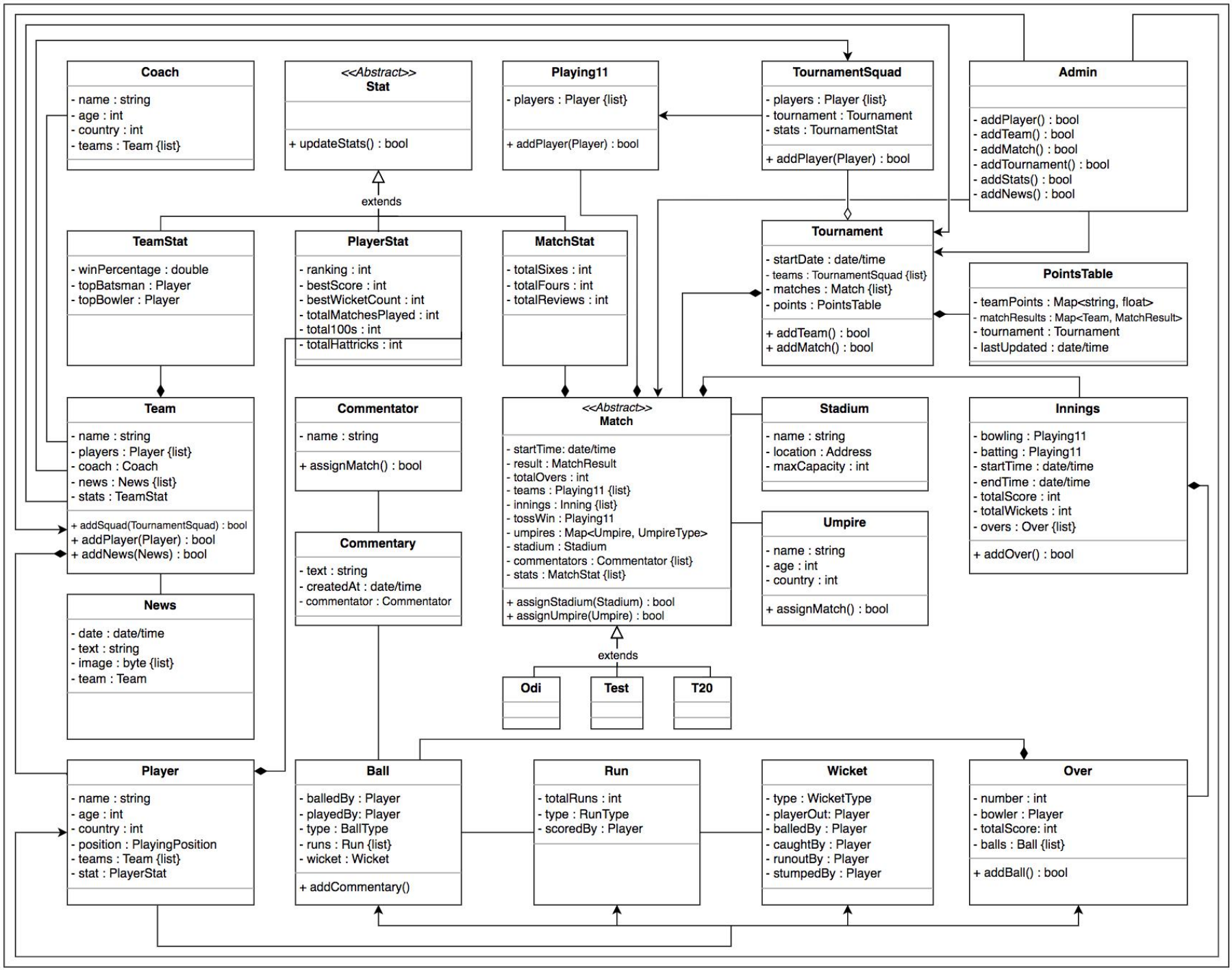
The class diagram of the online stock brokerage system



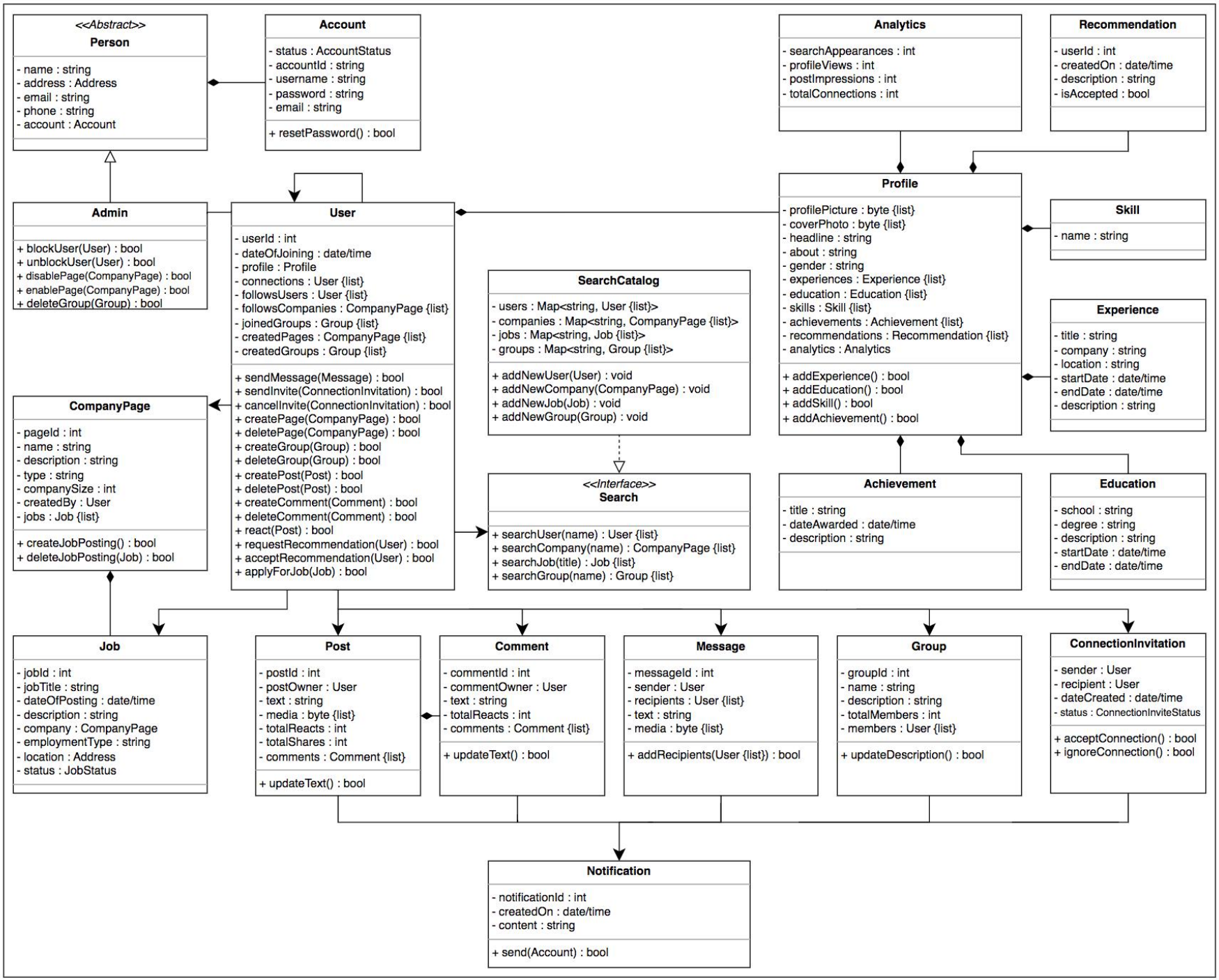
The class diagram of the jigsaw puzzle



The class diagram of the airline management system



The class diagram of Cricinfo



The class diagram of LinkedIn

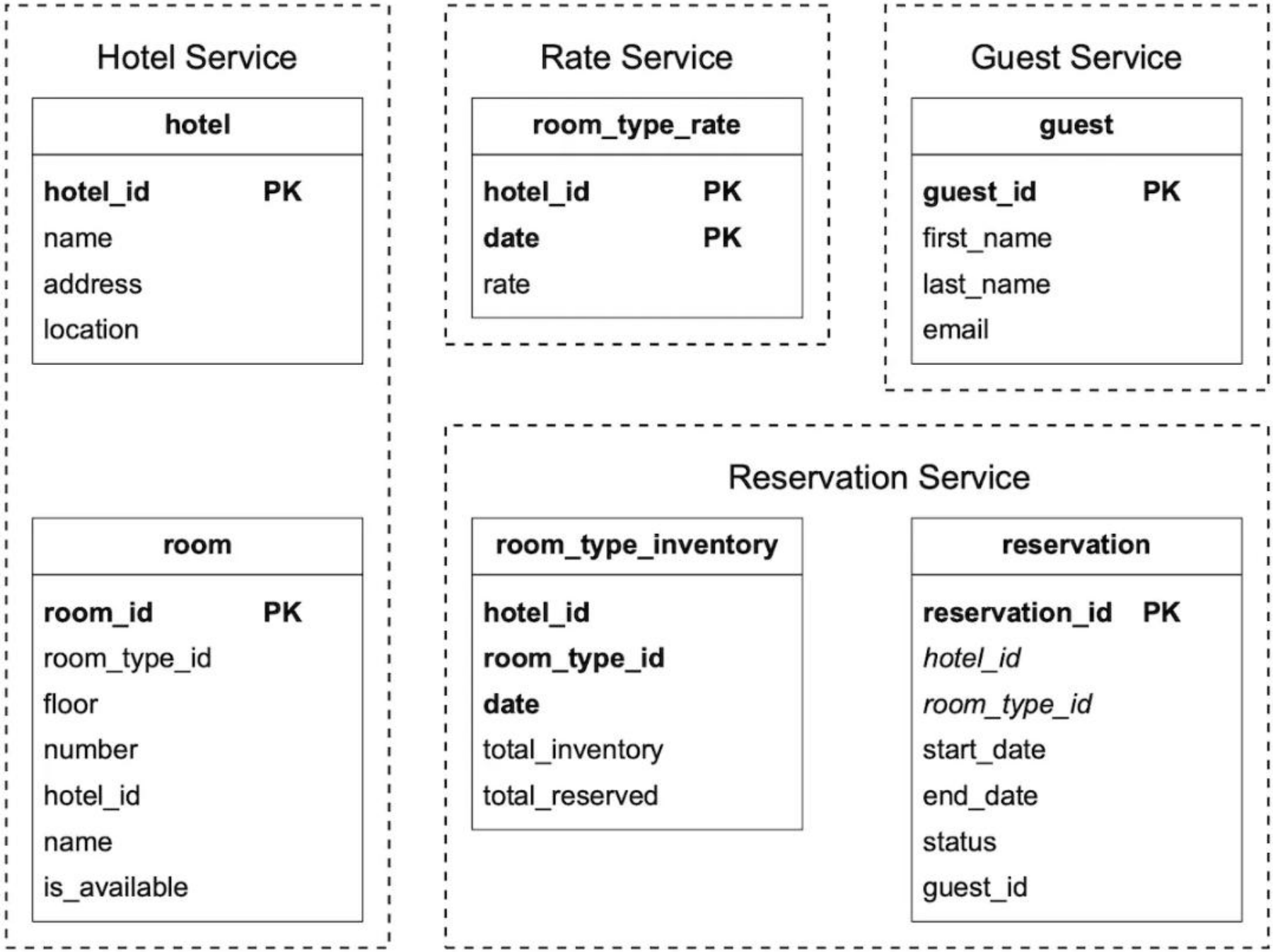


Figure 6 Updated schema

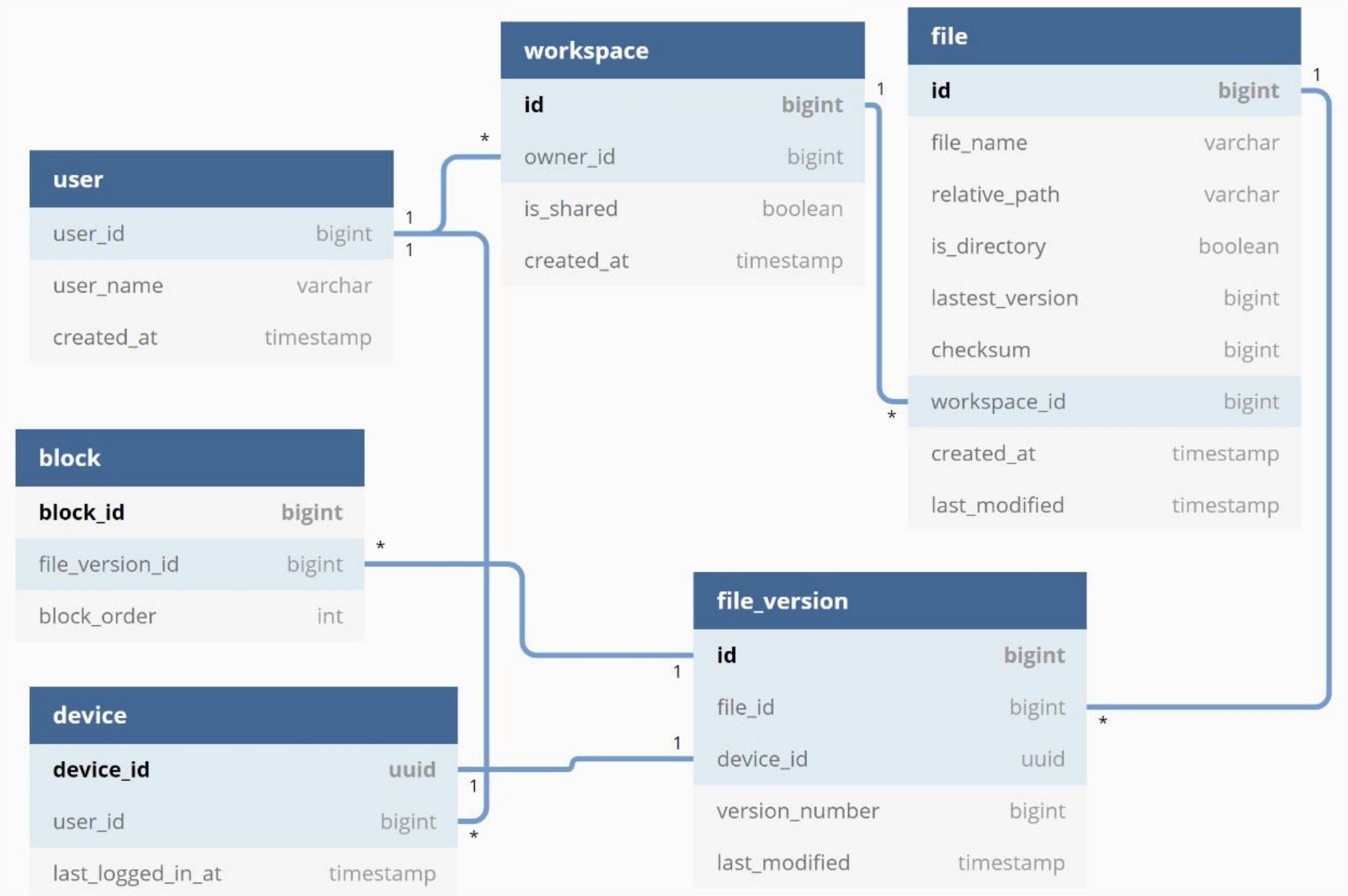
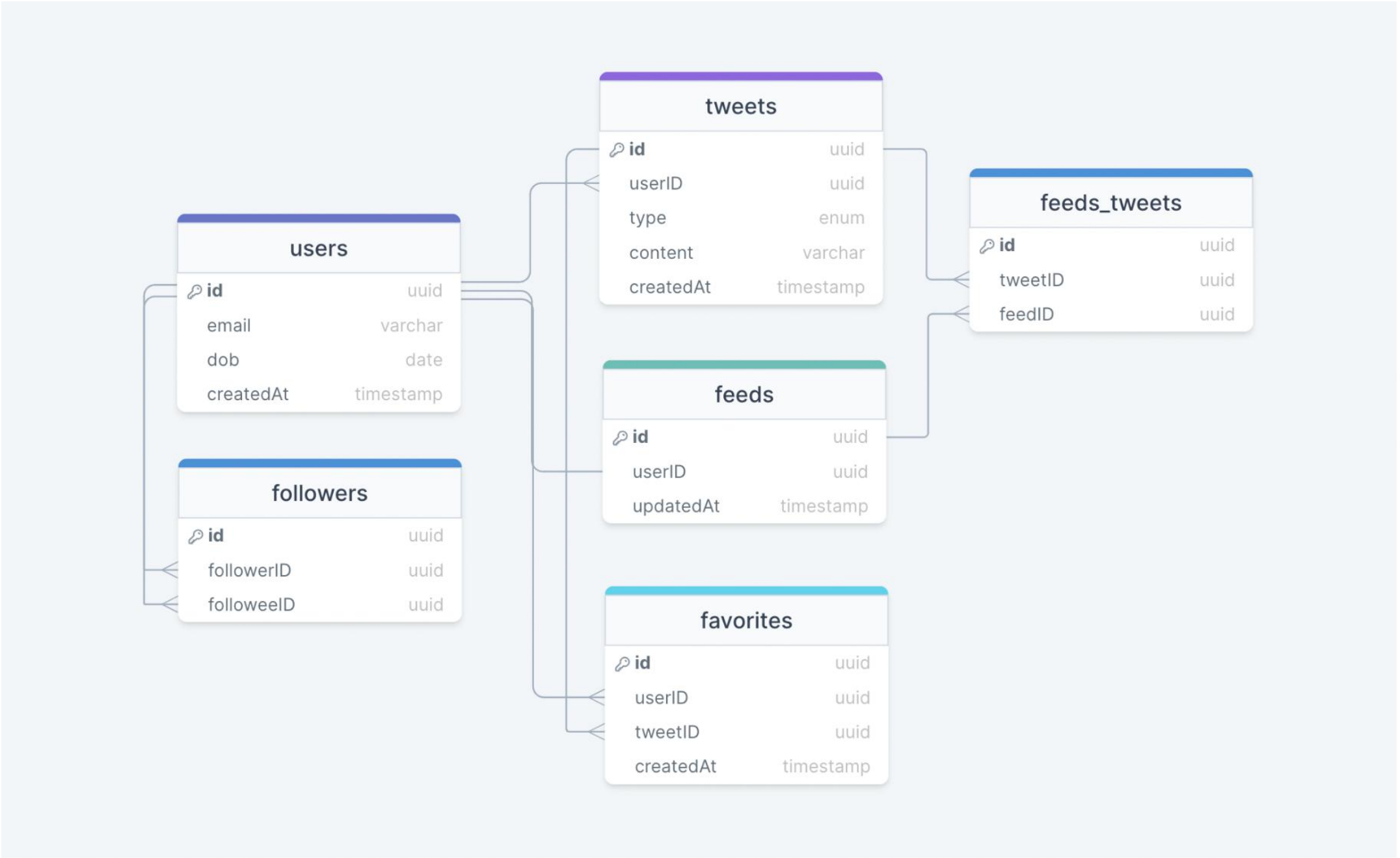

workspace
id bigint
owner_id bigint
is_shared boolean
created_at timestamp
file
id bigint
file_name varchar
relative_path varchar
is_directory boolean
lastest_version bigint
checksum bigint
workspace_id bigint
created_at timestamp
last_modified timestamp
user
user_id bigint
user_name varchar
created_at timestamp
block
block_id bigint
file_version_id bigint
block_order int
file_version
id bigint
file_id bigint
device_id uuid
version_number bigint
last_modified timestamp
device
device_id uuid
user_id bigint
last_logged_in_at timestamp
1
*

Figure 13

Global Secondary Index		Attributes		
Partition key (PK)	**Sort key (score)**	**user_id**	**email**	**profile_pic**
chess#2020-02#p0	309	lovelove	love@test.com	https://cdn.example/3.png
chess#2020-02#p1	209	i_love_tofu	test@test.com	https://cdn.example/p.png
chess#2020-03#p2	103	golden_gate	gold@test.com	https://cdn.example/2.png
chess#2020-02#p1	203	pizza_or_bread	piz@test.com	https://cdn.example/31.png
chess#2020-02#p2	10	ocean	oce@test.com	https://cdn.example/32.png
…	…	…	…	…

Figure 24 Updated partition key


users
id uuid
email varchar
dob date
createdAt timestamp
followers
id uuid
followerID uuid
followeeID uuid
tweets
id uuid
userID uuid
type enum
content varchar
createdAt timestamp
feeds
id uuid
userID uuid
updatedAt timestamp
favorites
id uuid
userID uuid
tweetID uuid
createdAt timestamp
feeds_tweets
id uuid
tweetID uuid
feedID uuid