POKHARA ENGINEERING COLLEGE

ASSIGNMENT

NAME/SUBMITTED BY : Britant Shrestha

SEMESTER : V

SUBJECT : Software Engineering

ROLL NO : 06

SUBMITTED TO : ER. Shiva Ram Dam

USE CASE DIAGRAM

1. Draw a detailed use-case diagram for the following case study:
[2018 spring]

Case study:

A customer visits online shopping portal. A customer may buy item or just visit the page and logout. The customer can select a segment, then a category and brand to get different products in the desired band. The customer cam select product for purchasing. The process can be repeated for more items. Once the customer finishes selecting the product/s, the cart cane be viewed. If the customer wants to edit the final cart it can be done here. For final payment, the customer has to login to portal. If the customer is visiting for the first time, he must register with the site, else the customer use login page to proceed. Final cart is submitted for payment and card details and address details are to be confirmed with customer. Customer is confirmed with the shipment id and delivery if goods within 15 days.

Solution:**Title: Use-Case Diagram for Online Shopping Portal**

Introduction

This document presents a detailed use-case diagram for an online shopping portal based on the provided case study. The diagram illustrates the interactions between a customer and the system, covering various actions from visiting the portal to receiving the shipment ID after order confirmation.

Use case Diagram

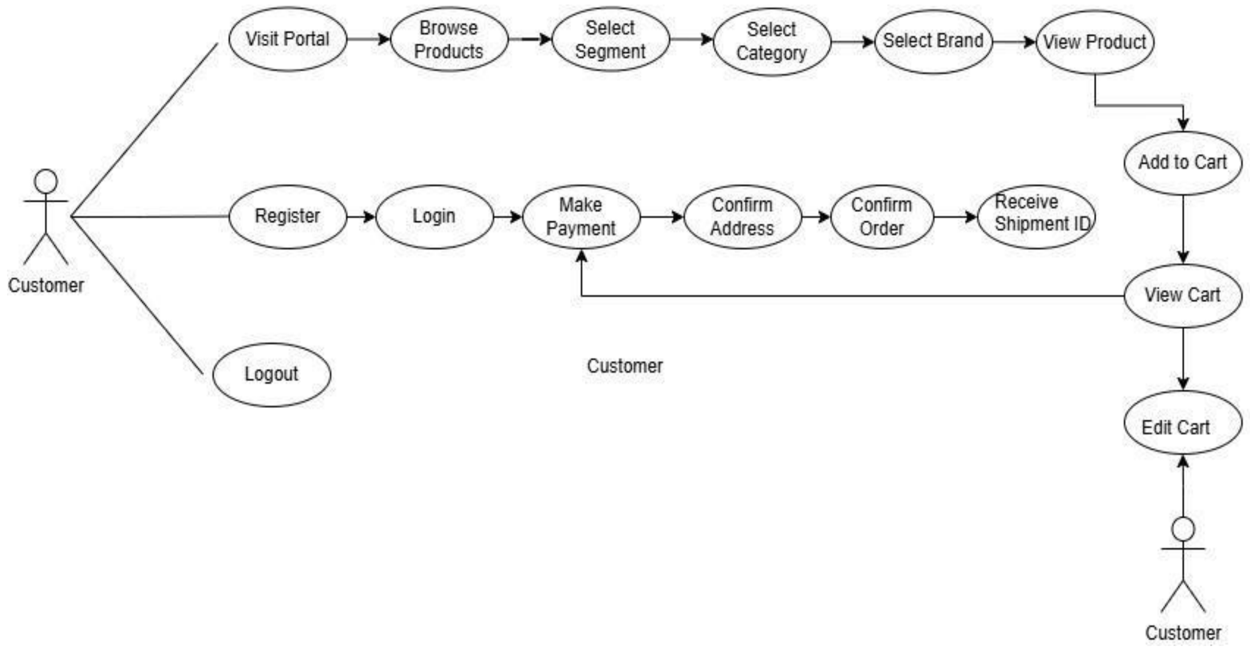


Diagram 1: Online Shopping Portal (System)

Description of Use Cases

1. Visit Portal: The customer visits the online shopping portal to explore or purchase items.

2. Browse Products: Customers can browse various products by selecting different segments, categories, and brands.

3. Select Segment, Category, and Brand: Customers can filter products by selecting a segment, category, and brand to find desired items.

4. View Product: The customer views detailed information about a specific product.
5. Add to Cart: The customer adds selected products to their shopping cart.

6. View Cart: The customer views the items in their cart to review their selections.

7. Edit Cart: The customer can edit the items in the cart before proceeding to checkout.

8. Register: New customers must register on the portal before making a purchase.

9. Login: Returning customers log in to access their accounts and proceed with purchases.

10. Make Payment: The customer proceeds to payment, confirming their card and address details.

11. Confirm Address and Order: The customer confirms their shipping address and finalizes the order.

12. Receive Shipment ID: The system provides a shipment ID, and the customer is informed about the delivery within 15 days.

13. Logout: The customer logs out after completing their session.

Conclusion

This use-case diagram provides a comprehensive view of the interactions between a customer and the online shopping portal, covering all essential functionalities from browsing products to confirming an order and receiving shipment details.

2. A customer presents a cheque to a clerk. The clerk checks the ledger containing all account numbers and make sure whether the account number in the cheque is valid, whether adequate balance is there in the account to pay the cheque, and whether the signature is authentic. Having done these, the clerk gives the customer a token. The clerk also debits customer’s account by the amount specified on the cheque. If cash cannot be paid due to an error in the cheque, the cheque is returned. The token number is written on the top of the cheque and it is passed on to the cashier. The cashier calls out the token number, takes the customer’s signature, pays cash, enters cash paid in ledger called day book, and file the cheque. Derive use-cases from the above scenario and model them into a Use-case diagram. [2016 fall]

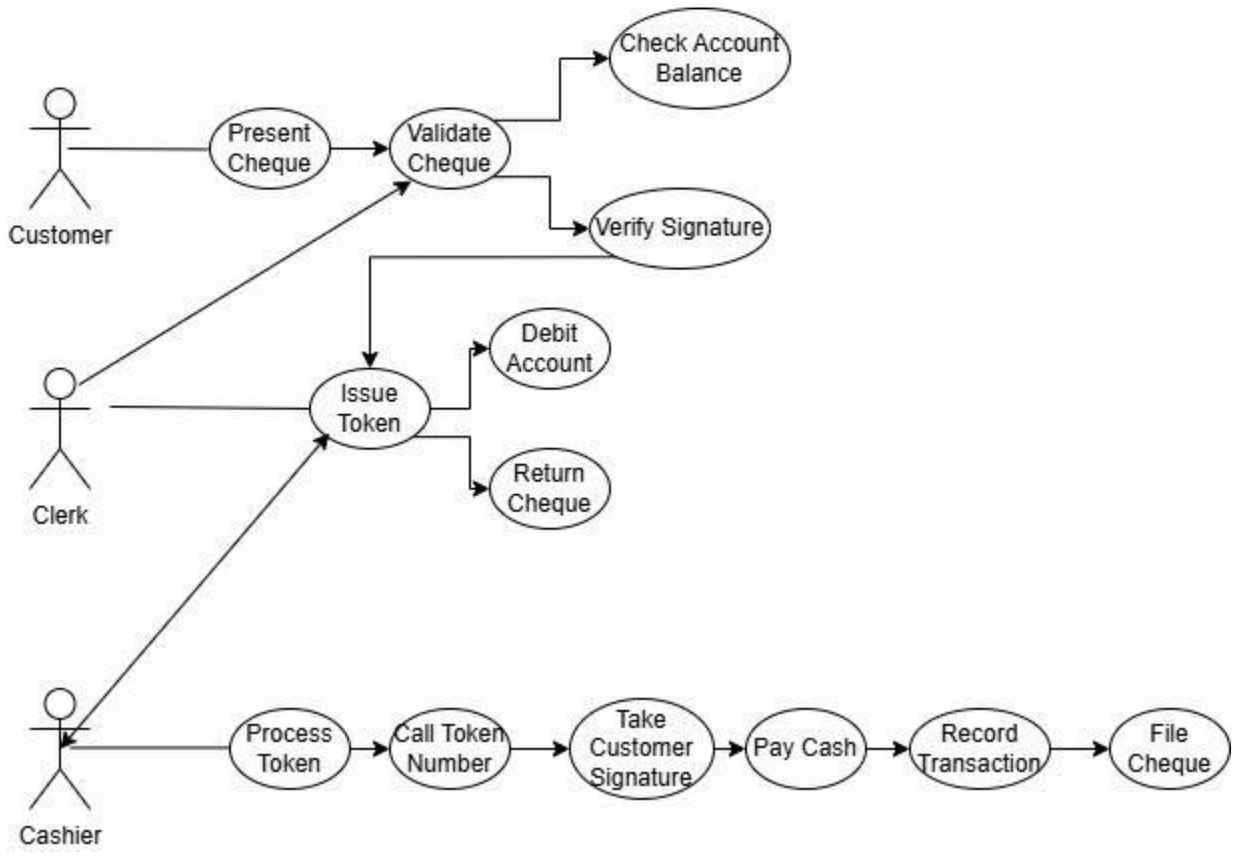
Solution:

Title: Use-Case Diagram for Bank Cheque Processing System

Introduction

This document presents a detailed use-case diagram for a bank cheque processing system based on the provided case study. The diagram illustrates the interactions between a customer, a clerk, and a cashier during the process of cheque validation and cash withdrawal.

Use case Diagram



Digram 2: Bank Cheque Processing System

Description of Use Cases

1. Present Cheque: The customer presents a cheque to the clerk for processing.
2. Validate Cheque: The clerk validates the cheque by checking the account number, balance, and signature.
3. Check Account Balance: The clerk checks if there is adequate balance in the customer's account to cover the cheque amount.
4. Verify Signature: The clerk verifies if the signature on the cheque matches the one on file.
5. Issue Token: If the cheque is valid, the clerk issues a token to the customer and writes the token number on the cheque.
6. Debit Account: The clerk debits the customer's account by the amount specified on the cheque.
7. Return Cheque: If there is an error in the cheque, the clerk returns it to the customer.
8. Process Token: The cashier processes the token received from the clerk.
9. Call Token Number: The cashier calls out the token number to notify the customer.

10. Take Customer Signature: The cashier takes the customer's signature as a confirmation before paying cash.

11. Pay Cash: The cashier pays the specified amount in cash to the customer.

12. Record Transaction: The cashier records the transaction in the day book.

13. File Cheque: The cashier files the cheque after the transaction is completed.

Conclusion

This use-case diagram provides a comprehensive view of the interactions between the customer, clerk, and cashier during the cheque processing in a bank, ensuring a smooth transaction flow from cheque presentation to cash payment.

3. Suppose you want to develop software for an alarm clock. The clock shows the time of day.

Using buttons, the user can set the hours and minutes fields individually, and choose between 12 and 24-hour display. It is possible to set one or two alarms. When an alarm fires, it will sound some noise. The user can turn it off, or choose to snooze. It the user does not respond at all, the alarm turns off itself after 2 minutes. Snoozing means to turn off the sound, but the alarm will fire again after some minutes of delay. This snoozing time is pre adjusted. Draw use case for this system. [2016 fall]

Solution:

Title: Use-Case Diagram for Alarm Clock System

Introduction

This document presents a detailed use-case diagram for an alarm clock system. The diagram illustrates the interactions between the user and the system, covering functionalities such as setting the time, setting alarms, snoozing, and turning off alarms.

▯ Use case Diagram

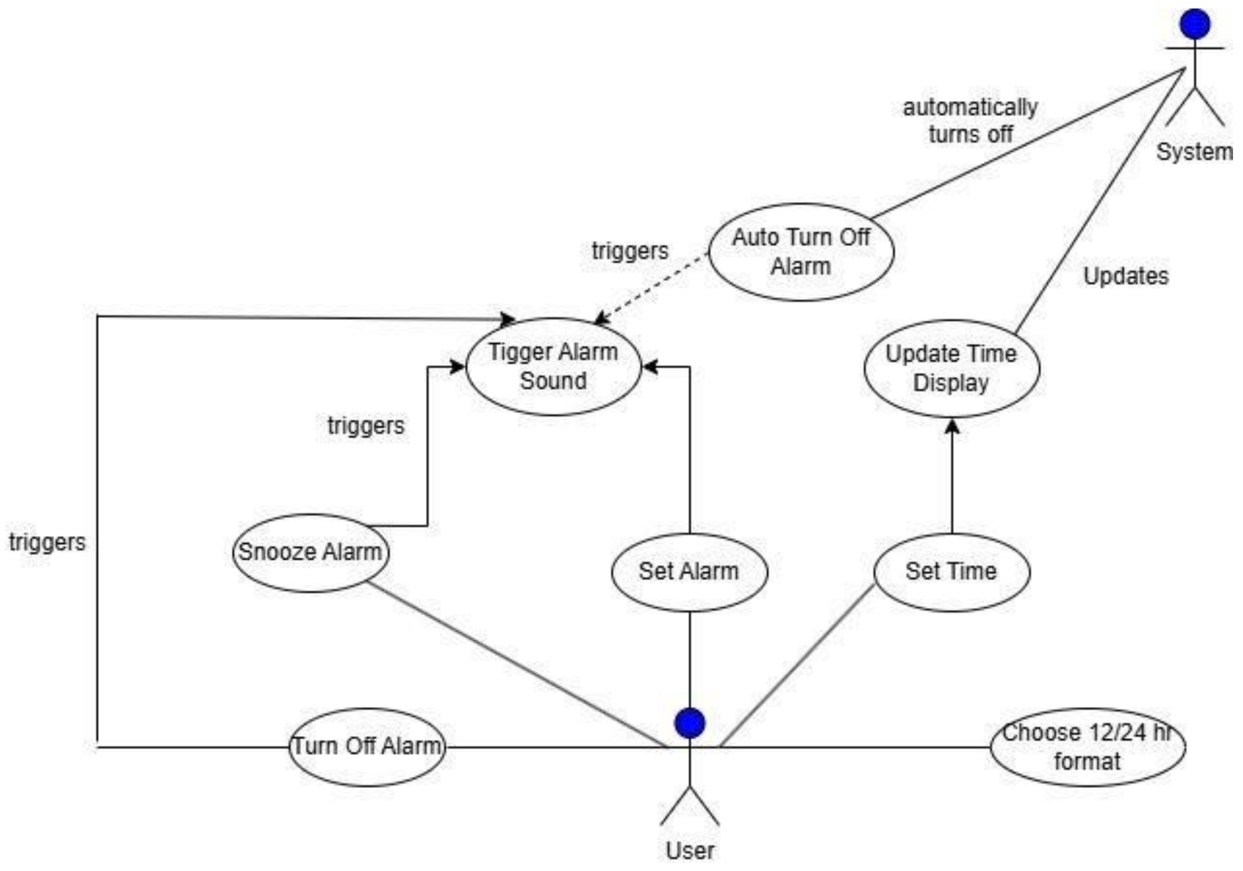


Diagram: Alarm Clock System

▯ Description of Use Cases

1. Set Time: The user sets the current time on the alarm clock.

2. Set Alarm: The user sets one or two alarms on the clock.

3. Choose 12/24 Hour Format: The user selects between a 12-hour or 24-hour time format.
4. Snooze Alarm: The user snoozes the alarm, which will stop the sound and trigger again after a pre-set delay.

5. Turn Off Alarm: The user turns off the alarm when it sounds.

6. Auto Turn Off Alarm: The alarm automatically turns off if the user does not respond within two minutes.

7. Update Time Display: The system updates the display to show the current time.

8. Trigger Alarm Sound: The system triggers the alarm sound when an alarm goes off.

Conclusion

This use-case diagram provides a comprehensive view of the interactions between the user and the alarm clock system, ensuring a smooth operation from setting the time to handling alarms effectively.

CLASS DIAGRAM

4. Design a class diagram for class Books which has following attributes and operations:

Attributes: name, Author Name, ISSN Number, Price
Operations: to take value for the above data, to search a book by its ISSN and to display the details of a book. [2016 spring]

Solution:

Title: Class Diagram for Books Class

Introduction

This document provides an advanced class diagram for the Books class, detailing its attributes, operations, and relationships. The class encapsulates key functionalities for handling book information, including managing attributes such as name, author, ISSN number, and price. It also defines operations for setting values, searching for books by ISSN, and displaying book details. Additionally, the relationships between the Books class and the Author and Publisher classes are described to provide a comprehensive view of the system.

Class Diagram

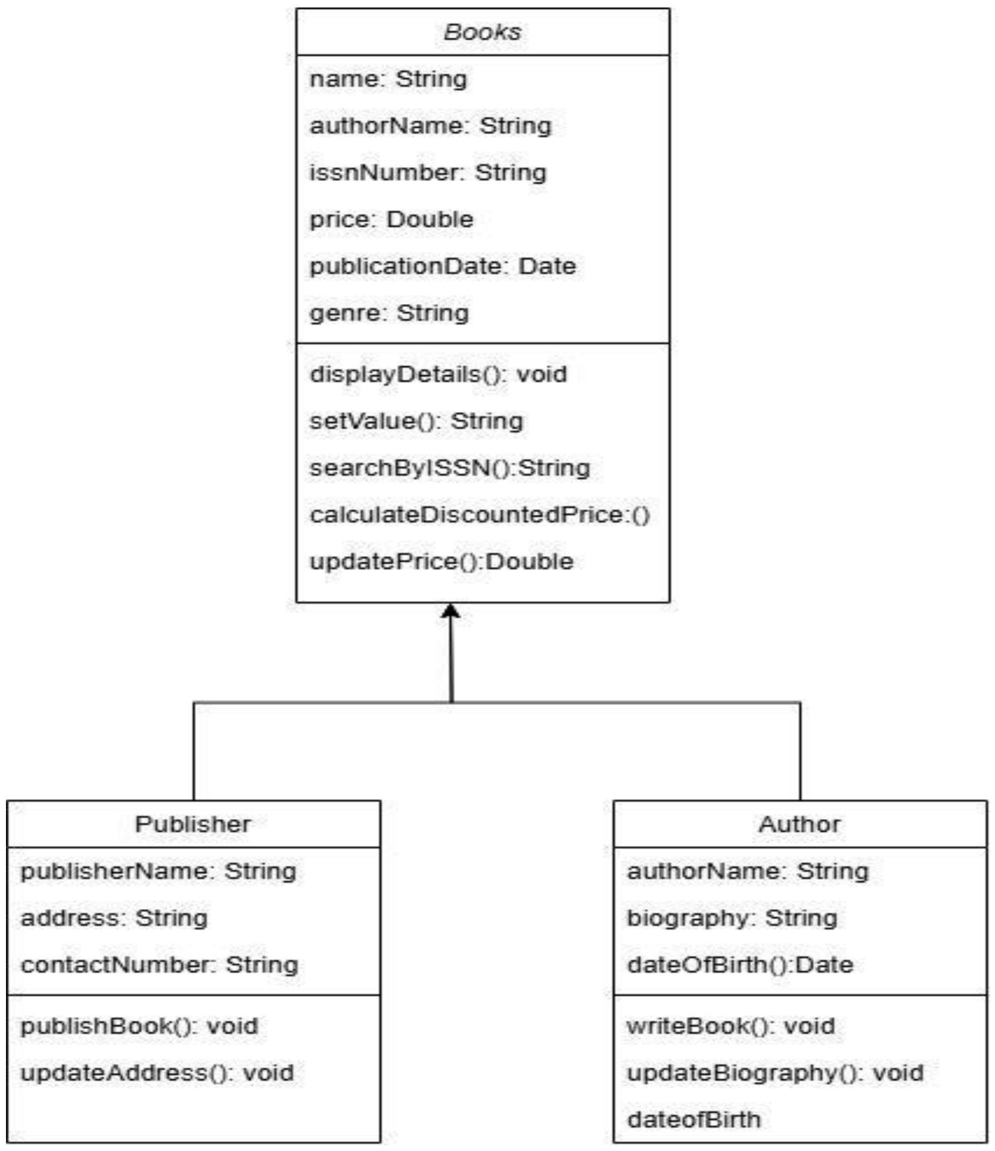


Diagram: Books Class

Description of Class Components

Attributes

name: Represents the name of the book.

authorName: Stores the name of the book's author.

issnNumber: Holds the ISSN number of the book for

identification. **price**: Represents the price of the book.

publicationDate: Indicates the date the book was

published. **genre**: Specifies the genre of the book.

Operations

setValue(): Allows setting values for the book's attributes, including name, author name, ISSN number, price, publication date, and genre.

searchByISSN(): Searches for a book using its ISSN number and returns the corresponding book object if found.

displayDetails(): Displays the details of the book, including all attributes such as name, author name, ISSN number, price, publication date, and genre.

calculateDiscountedPrice(): Calculates and returns the discounted price of

the book based on a given discount rate. **updatePrice()**: Updates the price of

the book to a new specified value.

Author Class **authorName**: Stores

the name of the author.

biography: Contains a short biography of the

author. **dateOfBirth**: Stores the author's date of

birth. Operations **writeBook()**: Represents the

action of writing a new book.

updateBiography(): Updates the author's

biography. Publisher Class **publisherName**:

Stores the name of the publisher. **address**:

Contains the address of the publisher.

contactNumber: Stores the contact number of

the publisher. Operations **publishBook()**:

Represents the action of publishing a new

book. **updateAddress()**: Updates the

publisher's address.

Relationship between Books and Author

The **Books** class has a many-to-one relationship with the **Author** class, indicating that multiple books can be written by a single author. This relationship allows the system to associate each book with its respective author, enabling functionalities like listing all books by a particular author or accessing the author's details from a book instance.

Relationship between Books and Publisher

The **Books** class also has a many-to-one relationship with the **Publisher** class, meaning that multiple books can be published by a single publisher. This relationship helps in managing publisher-specific information and operations, such as retrieving all books published by a specific publisher or updating publisher details associated with a book.

Conclusion

The advanced class diagram for the Books class, along with its associations with the Author and Publisher classes, offers a comprehensive view of its attributes, operations, and relationships. This structured approach facilitates the handling of complex book data and provides enhanced functionality for book management systems.

5. A simple system is to be developed to support the management of exercises completed by students taking a course. Students first meet with course tutor to register for a course, and then during the course they submit a number of exercises. Every course has a certain deadline assigned by the course tutor. Tutors can allow an exercise to be submitted late. At any point, a student can find out from the system the marks they have received for any exercises already completed. A student shall also be able to view any comments made by the tutor on certain exercise. The course tutor can also

enter a mark for an exercise, and print out a summary of the marks gained by all students on course.

Identify classes and draw a class diagram to model an efficient solution for the problem.

Solution:

Title: Class Diagram for Student Exercise Management System

Introduction

This document presents a detailed class diagram for a student exercise management system based on the provided case study. The diagram models the relationships between students, courses, exercises, and tutors, supporting the management of exercises submitted by students.

Class Diagram

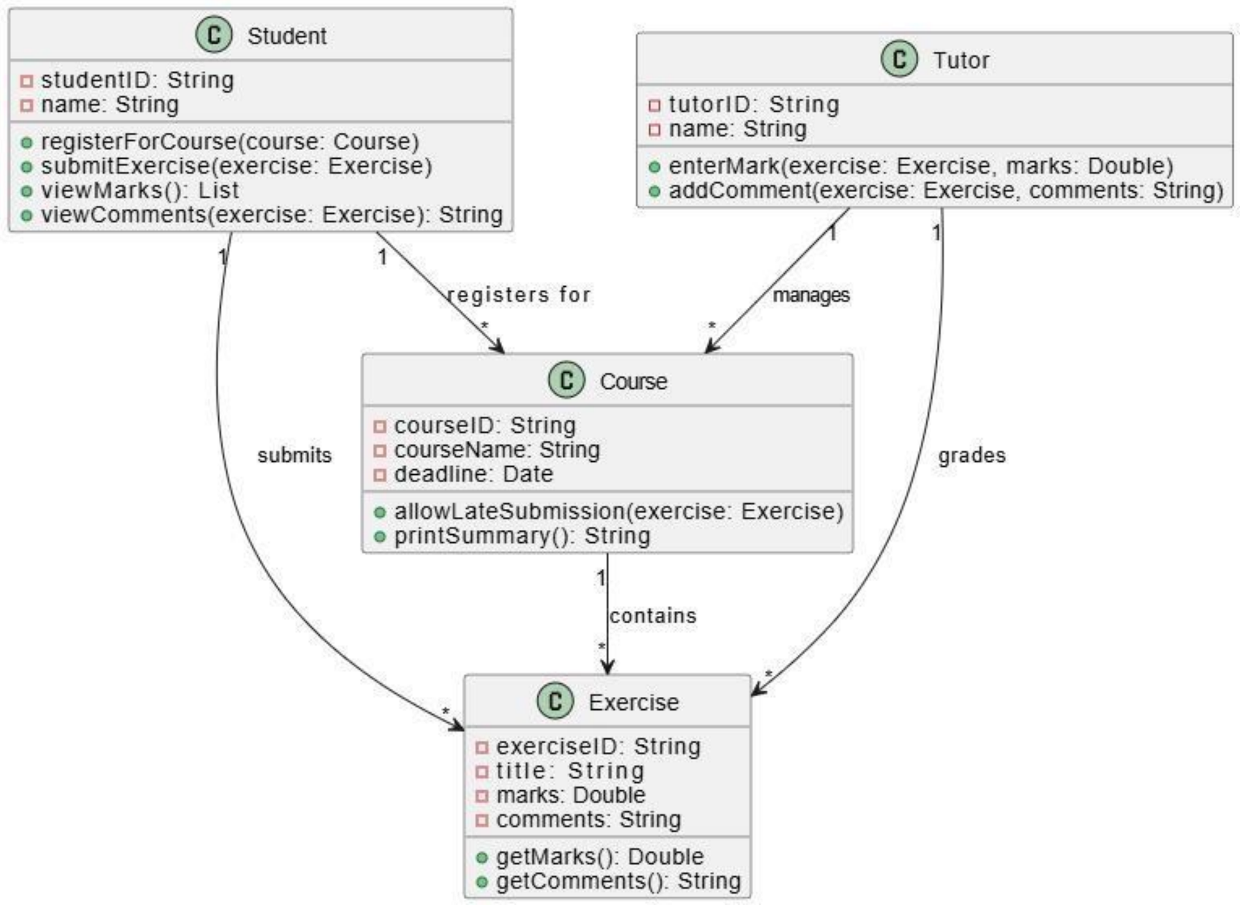


Diagram: Student Exercise Management System

Description

1. **Student Class**: Has attributes studentID and name. It includes methods for registering for a course, submitting an exercise, viewing marks, and viewing comments.
2. **Course Class**: Contains attributes courseID, courseName, and deadline. It has methods for allowing late submission and printing a summary of marks.

3. **Exercise Class**: Holds attributes exerciseID, title, marks, and comments. It provides methods for retrieving marks and comments.
4. **Tutor Class**: Contains attributes tutorID and name, with methods for entering marks and adding comments to exercises.

Relationships:

1. A Student registers for one or more Courses and submits exercises.
2. A Course contains multiple Exercises.
3. A Tutor manages and grades exercises in the Course.

Conclusion

This class diagram provides a structured model for managing student exercises within a course, allowing tutors to manage deadlines, marks, and comments, while enabling students to register for courses, submit exercises, and view their performance.

6. Draw a UML class diagram representing the following elements from the problem domain for a hockey league. A hockey league is made up of at least four hockey teams. Each hockey team is composed of 6 to 12 players, and one player captains the team. A team has a name and a record. Players have a number and a position. Hockey team play games against each other. Each game has a score and a location. Teams are sometimes led by a coach. A coach has a level of accreditation and a number of

years of experience, and can coach multiple teams. Coaches and players are people, and people have names and addresses. Draw a class diagram for this information, and be sure to label all associations with appropriate multiplicities. [2015 fall]

Solution:

Title: Hockey League UML Class Digram

Introduction

The league consists of teams, players, games, coaches, and individuals. The diagram will represent the structure of these entities, their attributes, and the relationships between them.
Each team has a name and a record, while players have a number, position, and captain. Coaches can coach multiple teams and have accreditation levels and years of experience. The relationships between teams, players, coaches, and games will also be represented, including their multiplicities.

Class Diagram

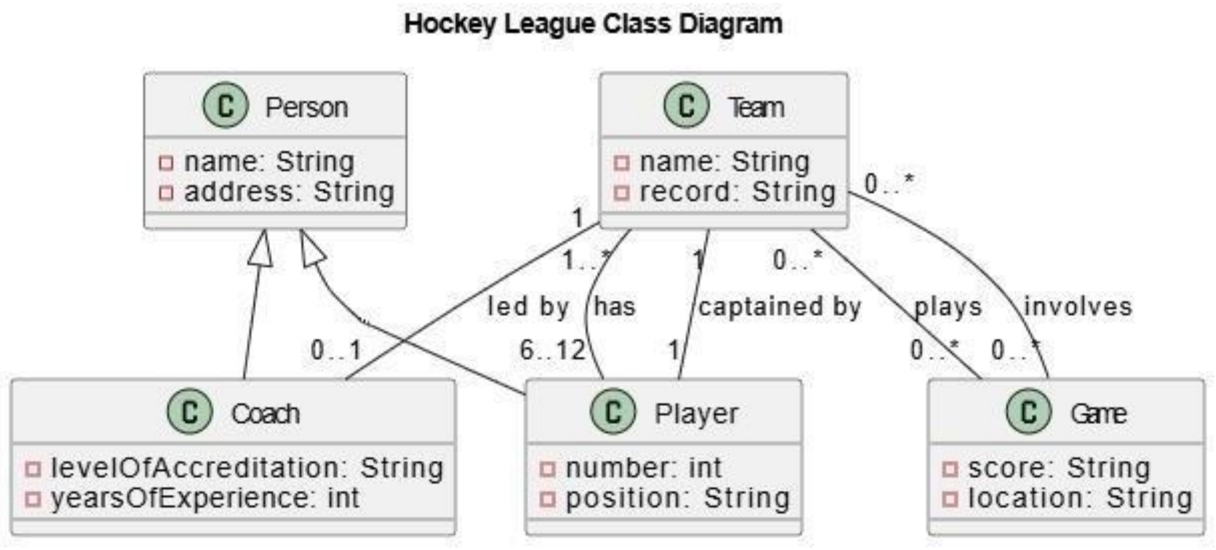


Diagram:Hockey League Diagram

Description of Relationships:

1. **Person and Player/Coach**: A Player and a Coach both inherit from the Person class, meaning they share common attributes like name and address.
2. **Team and Player**: A team has between 6 to 12 players, which forms a one-to-many relationship. Each team can have multiple players, but each player belongs to exactly one team.
3. **Team and Coach**: A team can have zero or one coach, meaning that a coach can optionally lead a team. A coach can lead multiple teams, forming a one-to-many relationship.
4. **Team and Player (Captain)**: Each team has one captain who is a player. This is a one-toone relationship, where one player captains one team.

5. **Team and Game**: Teams play multiple games against other teams. This forms a many-to many relationship, where each game involves two teams, and a team can play many games.
6. **Game and Team**: Each game involves two teams, forming a many-to-many relationship.

Conclusion:

This UML class diagram for a hockey league captures the essential elements of the league's structure, including teams, players, coaches, and games. The relationships between these classes are well defined, such as a coach managing multiple teams, players being part of teams, and games involving teams. The multiplicities are accurately represented to reflect real-world constraints (e.g., a team has between 6 to 12 players, a coach can lead multiple teams).

7. A health clinic provides medical services to patients in a small town. Five doctors and three nurses work at the clinic. They consult with patients, prescribe medicines and carry out minor medical treatments. Patients with more serious conditions are referred to specialists at the local hospital. A medical information system is being designed for the use in the clinic. The system will manage information about employees (doctors, nurses and administrator), patients and their contact details, appointments and consultations, medicines and prescriptions, treatments given, and referrals. Produce a UML class diagram for use in constructing the system using an object-oriented programming language. Your diagram must include all applicable classes and relationships.

There is no need to show the attributes and operations for each class.

[2014 Fall]

Solution:

Title: Health Clinic Class Diagram

Introduction

The clinic employs doctors, nurses, and an administrator, and it provides medical services to patients. The system will manage various aspects of the clinic's operations, including employee details, patient records, appointments, consultations, prescriptions, treatments, and referrals. The diagram will represent the relevant classes and relationships between them, focusing on entities like employees (doctors, nurses, and administrator), patients, medicines, treatments, and referrals.

Class Diagram

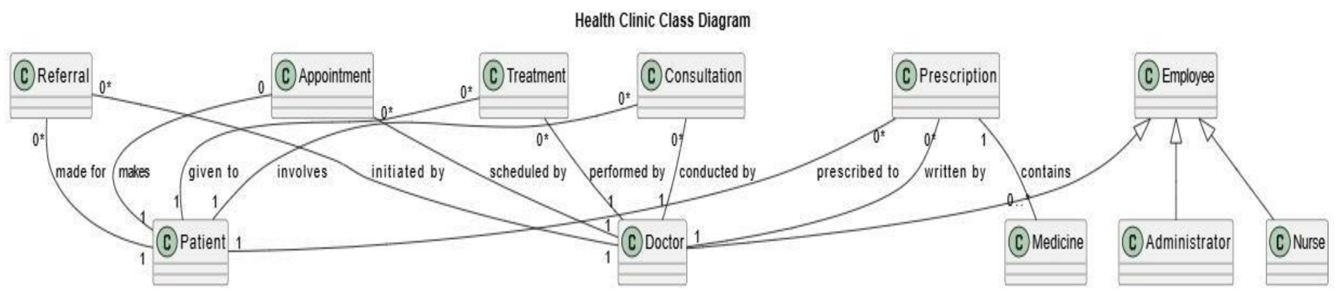


Diagram: Heath Clinic Class

Diagram ## Description of Relationships:

1. **Employee and Doctor/Nurse/Administrator**: The Employee class is the parent class, and it is inherited by Doctor, Nurse, and Administrator. These specific roles represent different types of employees in the clinic.
2. **Patient and Appointment**: A Patient can have multiple Appointments (Many-to-One relationship). Each appointment is scheduled with a Doctor.
3. **Consultation and Doctor/Patient**: A Consultation represents a medical consultation between a Doctor and a Patient. Each consultation is associated with one doctor and one patient (Many-to-One relationships).

4. **Prescription and Medicine**: A Prescription contains multiple Medicines. A medicine can be part of several prescriptions. This forms a One-to-Many relationship between Prescription and Medicine.
5. **Prescription and Doctor/Patient**: A prescription is written by a Doctor and is prescribed to a Patient. This creates a Many-to-One relationship between Prescription and Doctor/Patient.
6. **Treatment and Patient/Doctor**: A Treatment is given to a Patient by a Doctor. Each treatment is performed by one doctor and given to one patient, forming a Many-to-One relationship.
7. **Referral and Doctor/Patient**: A Referral is made by a Doctor for a Patient to see a specialist. This represents a Many-to-One relationship from Referral to Doctor and Patient.

Conclusion

This UML class diagram provides a structure for managing the health clinic's operations, including employee details, patient records, consultations, prescriptions, treatments, and referrals. The relationships between classes accurately reflect the real-world interactions, such as doctors consulting with patients, writing prescriptions, and referring patients to specialists. The diagram provides a clear framework for implementing the clinic's medical information system in an object-oriented programming language.

8.A simple system is to be developed to support the management of exercises completed by
students taking a course. Students first meet with course tutor to register for a course, and
then during the course they submit a number of exercises. Every course has a certain deadline
assigned by the course tutor. Tutors can allow an exercise to be submitted late. At any point,

a student can find out from the system the marks they have received for any exercises already
completed. A student shall also be able to view any comments made by the tutor on certain
exercise. The course tutor can also enter a mark for an exercise, and print out a summary of the marks gained by all students on course.
Identify classes and draw a class diagram to model an efficient solution for the problem.

Solution:

Title: Class Diagram for Exercise Management System

Introduction:

The exercise management system is designed to support the management and tracking of exercises completed by students during a course. This system allows students to register for courses, submit exercises, view their marks, and read tutor comments. Tutors can enter marks, approve late submissions, and manage student progress. The class diagram models the key components of the system and their interactions, ensuring an efficient and organized solution.

Class Diagram

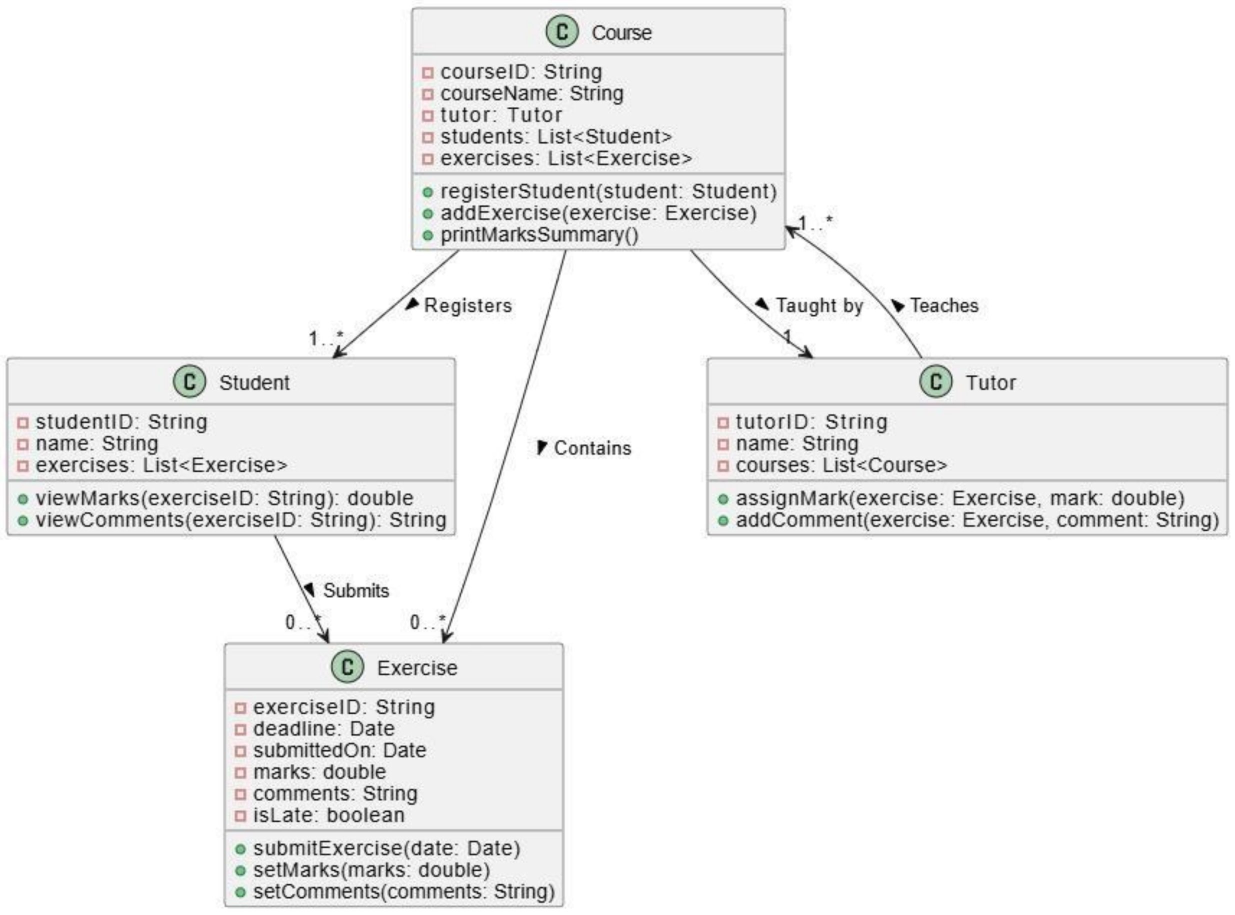

Course
- courseID: String
- courseName: String
- tutor: Tutor
- students: List<Student>
- exercises: List<Exercise>
+ registerStudent(student: Student)
+ addExercise(exercise: Exercise)
+ printMarksSummary()
Registers
1..*
Contains
Taught by
1
Teaches
1..*
Student
- studentID: String
- name: String
- exercises: List<Exercise>
+ viewMarks(exerciseID: String): double
+ viewComments(exerciseID: String): String
Tutor
- tutorID: String
- name: String
- courses: List<Course>
+ assignMark(exercise: Exercise, mark: double)
+ addComment(exercise: Exercise, comment: String)
Submits
0..*
0..*
Exercise
- exerciseID: String
- deadline: Date
- submittedOn: Date
- marks: double
- comments: String
- isLate: boolean
+ submitExercise(date: Date)
+ setMarks(marks: double)
+ setComments(comments: String)

Diagram: Exercise Management System

- Description:

The system involves four primary classes: **Student**, **Exercise**, **Course**, and **Tutor**. Each class serves a specific role in the system:

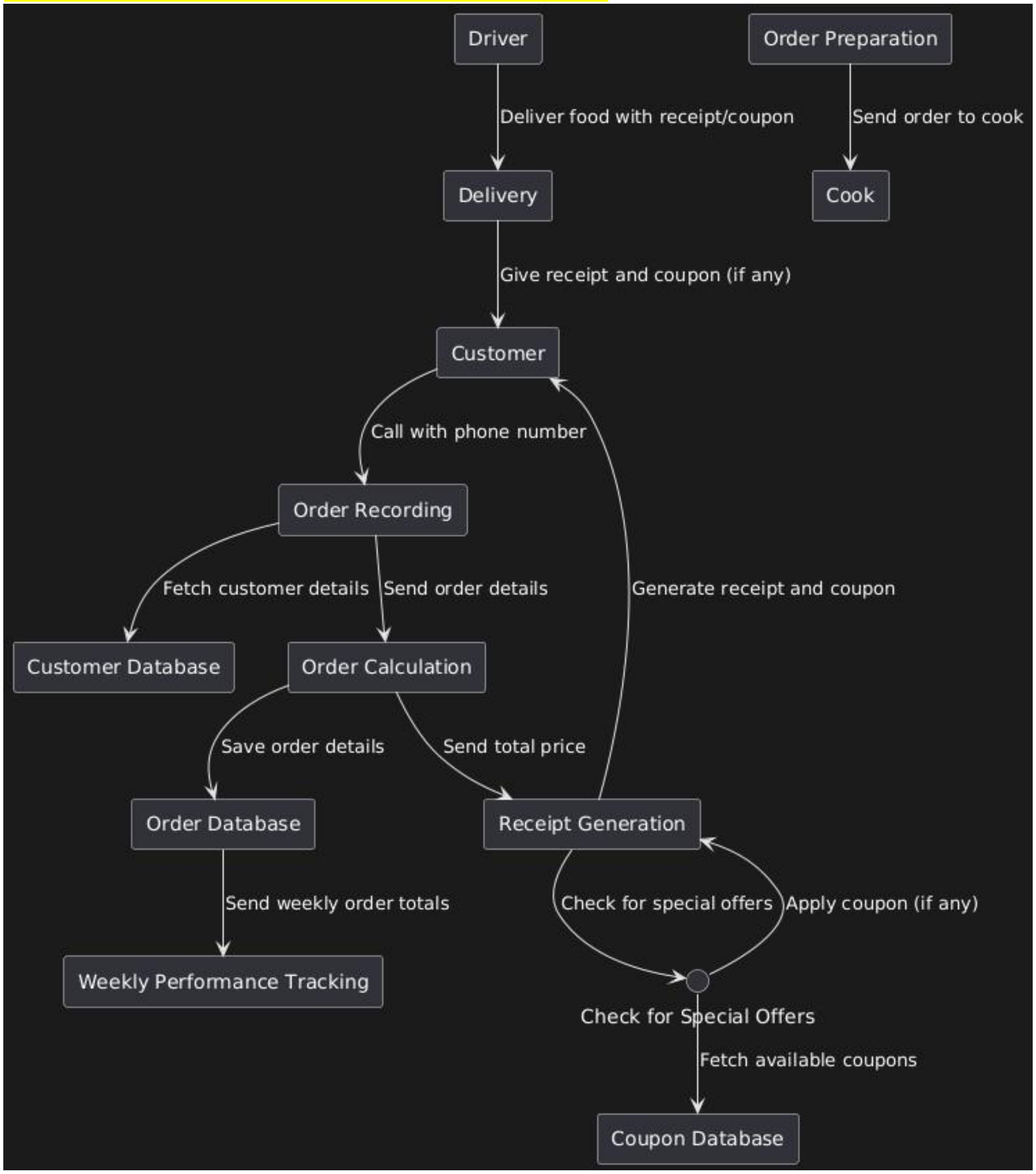
1. **Student**: Represents the students who register for courses and complete exercises. They can view their marks and comments for each exercise submitted.
2. **Exercise**: Represents an individual exercise assigned in a course. It includes attributes like the exercise ID, deadline, marks awarded, and comments from the tutor. It also tracks whether the exercise was submitted late.
3. **Course**: Represents the course offered, which contains a list of students and exercises. It also links to a tutor who teaches the course.
4. **Tutor**: Represents the course tutor who can assign marks to exercises, provide comments, and teach courses. Tutors are responsible for managing the academic progress of students by assessing their work.

- Conclusion:

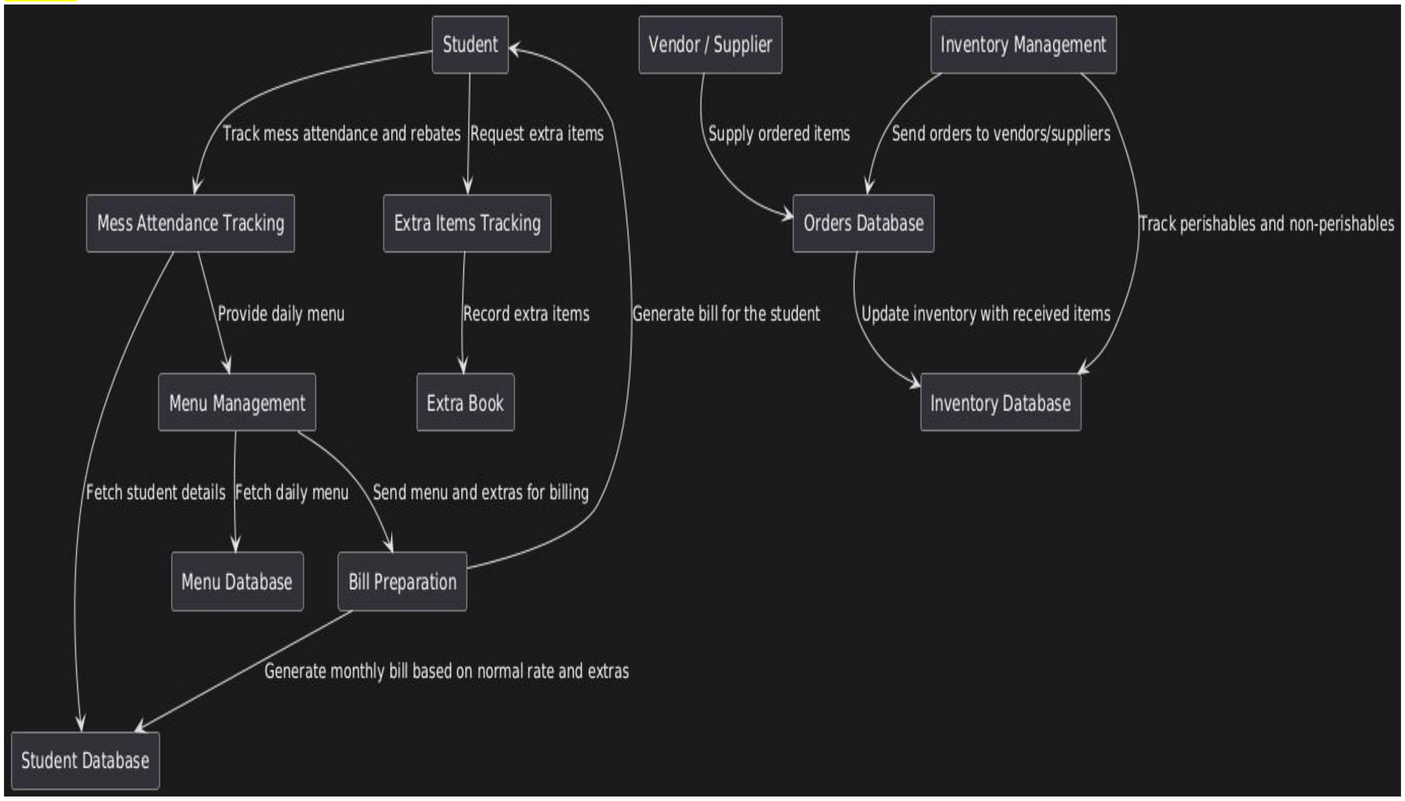
This class diagram represents a straightforward approach to managing courses, exercises, and students. The relationships between the different classes ensure that both students and tutors can interact efficiently within the system. By using this design, the system will be able to manage exercise submissions, mark assignments, and provide timely feedback to students. This model ensures flexibility, scalability, and ease of maintenance as it supports different scenarios like late submissions and course summaries.

9. Prepare level 1 DFD for the following Food Ordering System. [2020 fall]
KFC pizza wants to install a system to record orders for pizza and burger. When regular customer call KFC pizza on the phone, their phone number goes automatically into pizza system. The phone number invokes the name, address and last order date automatically on the screen. Once the order is taken, the total including tax and delivery is calculated. The order is given to the cook. A receipt is printed. Occasionally, special offers (coupons) is printed

so the customer can get discount. Drivers who make deliveries gives customers a copy of the receipt and coupon (if any). Weekly totals are kept for comparison with last year's performance.

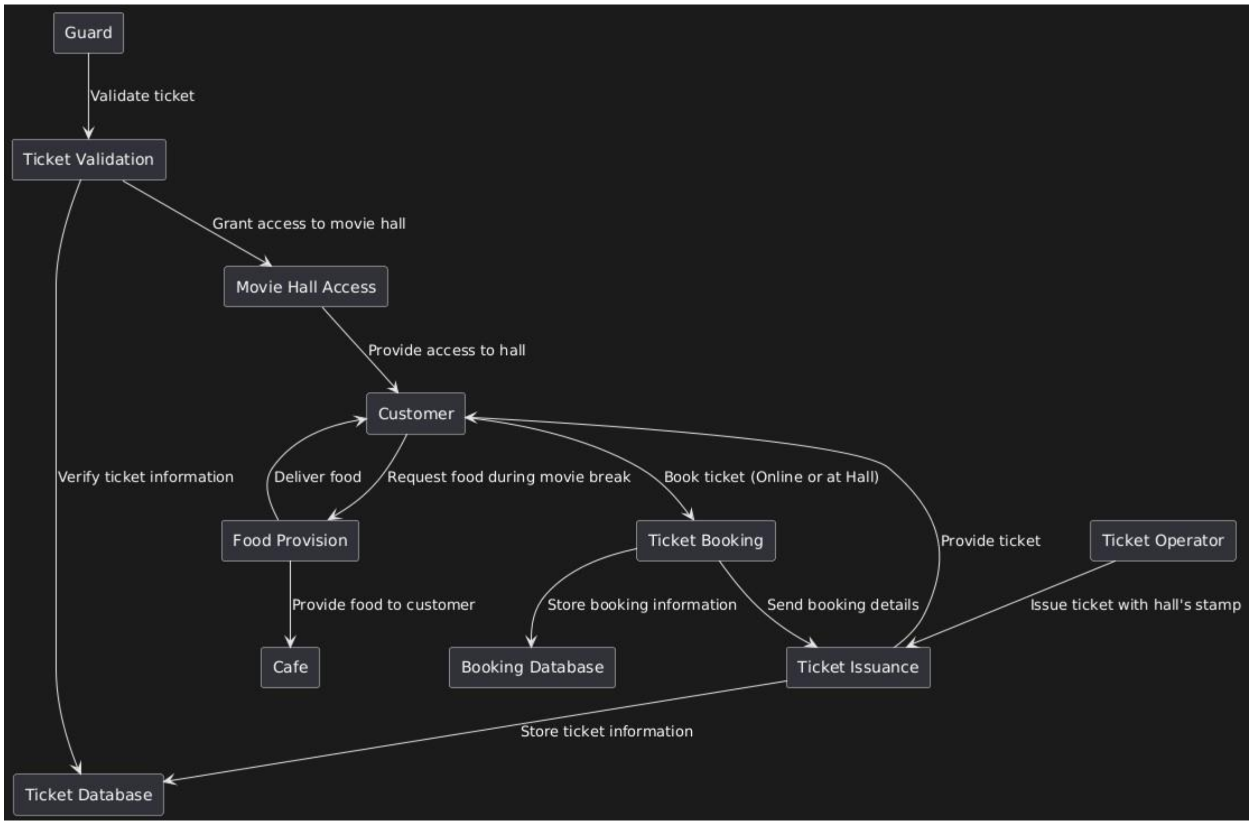


10. Obtain DFD for the following Mess management system: [2014 Spring] A hostel has 500 rooms and 4 messes. Currently, there are 1000 students in all in 2 seated rooms. They eat in any of the messes but can get rebate if they inform and do not eat for at least 4 consecutive days. Besides normal menu, extra items are also given to students when they ask for it. Such extras are entered in an extra book. At the end of the month, a bill is prepared based on the normal daily rate and extras and given to each student. System for stores issue and control is maintained for daily use of perishables and non-perishables items and order to vendor and suppliers are also maintained as well.



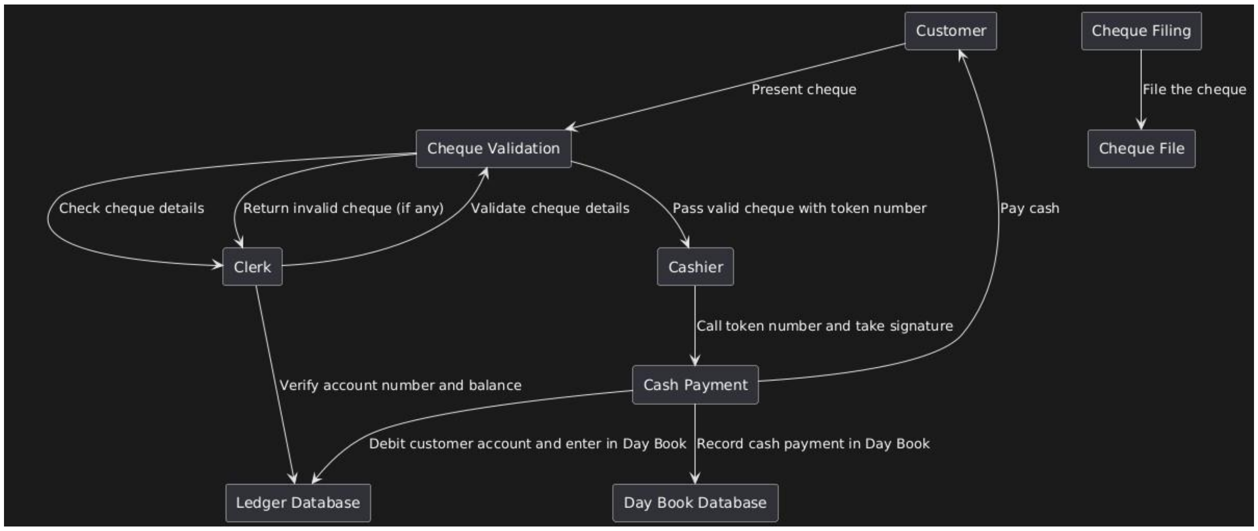
11.Obtain 1-level DFD for Movie Management System: [2015 spring]
A customer can book a ticket from the Internet or can directly buy the ticket in the Movie hall itself. There can be multiple halls within one movie theatre. The ticket operator provides a ticket with hall's stamp after checking the booking information to the customer. The guard in each hall validates the ticket and

provides access to the customer inside the hall. There is also provision of complementary food item which the café will provide in the break time of the movie.



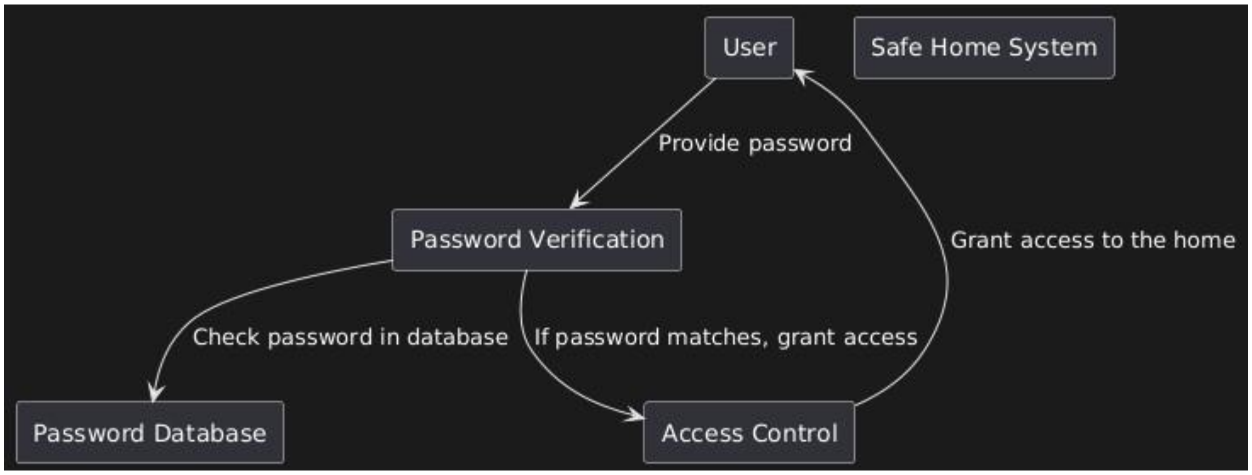
12. Obtain 1-level DFD for the following system of encashing cheque in a bank. A customer presents a cheque to a clerk. The clerk checks the ledger containing all account numbers and make sure whether the account number in the cheque is valid, whether adequate balance is there in the account to pay the cheque, and whether the clerk also debits customer's account by the amount specified on the cheque. If cash cannot be paid due to an error in the cheque, the cheque is returned. The token number is written on the top og the cheque and it is passed on to the cashier. The cashier calls

out the token number, takes the customer's signature, pays cash, enters cash paid in ledger called day book, and file the cheque. [2016 spring, 2014 fall]

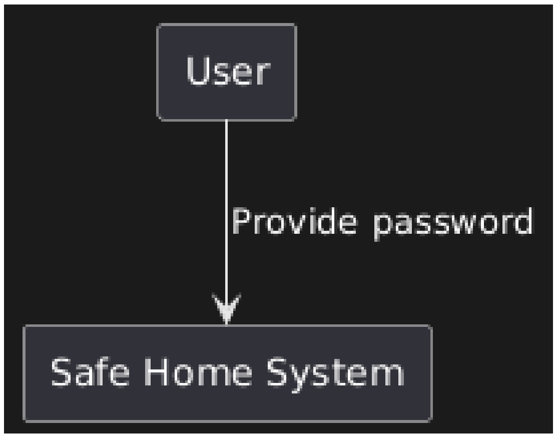

Customer
Cheque Filing
Present cheque
File the cheque
Cheque Validation
Cheque File
Check cheque details
Return invalid cheque (if any)
Validate cheque details
Pass valid cheque with token number
Pay cash
Clerk
Cashier
Call token number and take signature
Verify account number and balance
Cash Payment
Debit customer account and enter in Day Book
Record cash payment in Day Book
Ledger Database
Day Book Database

14. Draw the different levels of DFD for Safe Home System where any person can enter to the home on matching his/her password at the entrance door. [2018 fall]

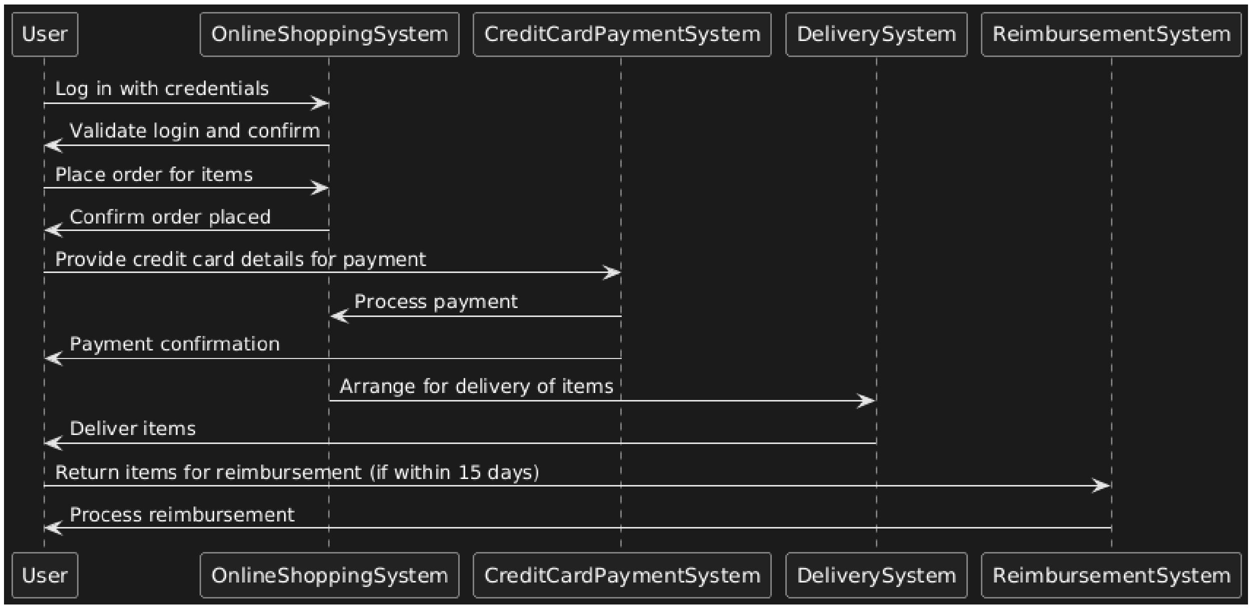
Level 1



Level 0

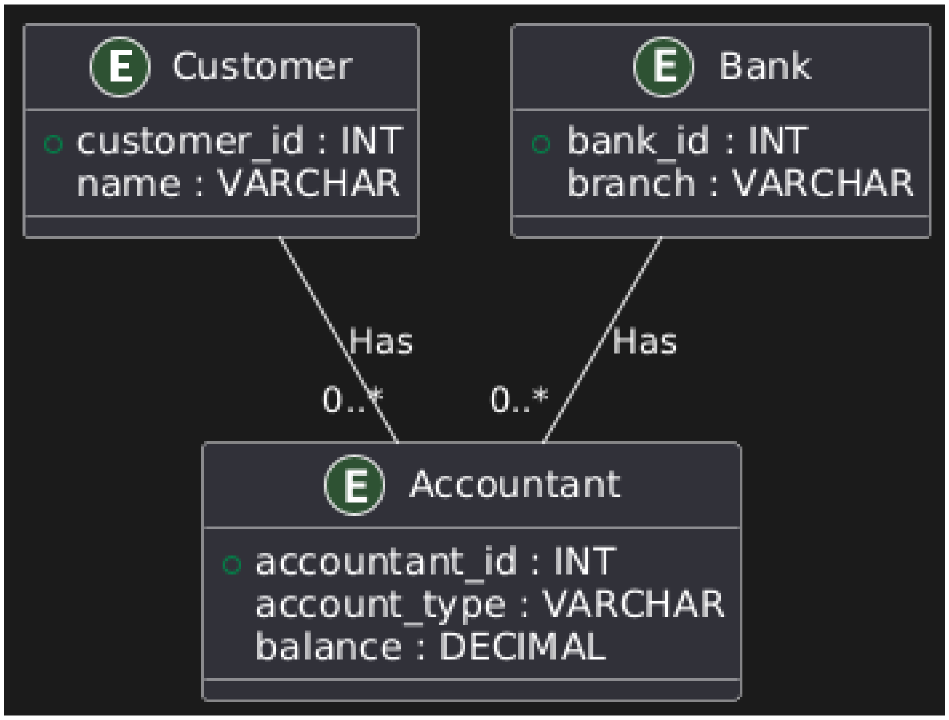


15. Draw a sequence diagram of the following: [2013 spring]
A valid user can login to the online shopping system. The user can order the items and as well as pay the Credit card. The items will be delivered later. The items within 15 days will be reimbursed.



16. Draw ERD for the following situation: [2014 Spring]

An accountant is a relationship between customer and bank. A customer has name. a bank has a branch. A customer may have several accountants of different types and balances.



17. A restaurant uses an Information system that takes customer orders, send the order to the kitchen monitors: the goods sold and inventory and generates reports for management. List functional and Non-functional requirements for this Restaurant Information System. [2018 fall]

Functional Requirements:

1. **Customer Order Management**: Place and modify customer orders.
2. **Order Processing**: Send orders to the kitchen and track status.
3. **Inventory Management**: Track and update inventory based on orders.
4. **Sales Monitoring**: Monitor items sold and generate sales reports.
5. **Billing & Payment**: Generate bills, accept various payment methods, and track payments.
6. **Management Reports**: Generate sales, inventory, and performance reports.
7. **Staff Management**: Track employee roles, schedules, and performance.
8. **Menu Management**: Update and manage menu items.

9. **Order Status Updates**: Provide real-time updates on order progress.

Non-Functional Requirements:

1. **Performance**: Fast response time (less than 2 seconds), handle multiple simultaneous orders.
2. **Scalability**: Support multiple branches and increasing users.
3. **Availability**: 24/7 availability with fault tolerance.
4. **Usability**: User-friendly interface, multi-device support, and multi-language support.
5. **Security**: Protect customer data with encryption, secure user roles.
6. **Reliability**: System stability, no downtime during peak hours.
7. **Data Integrity**: Ensure consistent and accurate data, no loss.
8. **Compatibility**: Integrate with existing hardware and third-party systems.
9. **Maintainability**: Easy updates, debugging, and maintenance.

18. Why User interface design is important in software development? Referencing a mobile application for smart agriculture, describe user interface design issues.

Importance of UI Design in Software Development

- **Usability**: Ensures the system is easy to use and navigate.
- **User Experience (UX)**: Creates a positive experience for users, increasing satisfaction.
- **Efficiency**: Helps users complete tasks quickly with minimal errors.
- **Consistency**: Provides a predictable interface for easier interaction.
- **Branding**: Reflects the app's identity and builds trust.

UI Design Issues for a Mobile Smart Agriculture App

1. **Simplicity**: Keep the interface simple and intuitive for users with varying tech skills.
2. **Data Visualization**: Use graphs and charts for easy interpretation of complex agricultural data.
3. **Offline Functionality**: Ensure the app works without internet connectivity in rural areas.
4. **Localization**: Support multiple languages and regional settings for diverse users.
5. **Accessibility**: Include features for users with visual or physical impairments (e.g., large text, voice commands).
6. **Help & Tutorials**: Offer easy access to help, FAQs, and onboarding tutorials.
7. **Task Flow Optimization**: Minimize steps to complete actions, focusing on essential features.
8. **Real-Time Notifications**: Provide customizable alerts (e.g., weather updates, irrigation status).

9. **Battery Efficiency**: Design the app to be energy-efficient for prolonged use outdoors.
10. **User Feedback**: Offer clear feedback (e.g., confirmations, visual indicators) after each action.

19. **What is Class Responsibility Collaborator model? How do you develop CRC model? Explain with example. [2013 spring]**

Class Responsibility Collaborator (CRC) Model:

The **CRC Model** is a tool used in object-oriented design to define classes, their **responsibilities**, and their **collaborators** (other classes they interact with). It helps organize and clarify how different objects in the system will interact and what they will do.

How to Develop a CRC Model:

1. **Identify Classes**: Define the key classes in the system (e.g., `Customer`, `Order`).
2. **Define Responsibilities**: List each class's responsibilities (e.g., `Customer` places orders).
3. **Identify Collaborators**: Determine which classes each class interacts with (e.g., `Order` collaborates with `Payment`).

Example: Online Shopping System

1. Customer Class

- **Responsibilities**: Register, place orders, manage account.
- **Collaborators**: `Order`, `Payment`.

2. Order Class

- **Responsibilities**: Add products, calculate total, track status.
- **Collaborators**: `Customer`, `Product`.

3. Product Class

- **Responsibilities**: Store details, update availability.
- **Collaborators**: `Order`.

4. Payment Class

- **Responsibilities**: Process payment, validate payment details.
- **Collaborators**: `Customer`, `Order`.