Adam Corbin COP 5330 - 002

2.1

a. Use cases

Clear messages from someone

1. System owner will enter in mailbox number into voice mail system
2. Voice mail will request if owner wants to delete messages from a specific number(along with other options)
3. System owner will acknowledge to delete messages from a specific number
4. Voice mail will request for user to enter in phone number to delete messages
5. System owner enters in phone number
6. Voice mail will delete messages where phone number matches
7. Voice mail speaks "The messages have been successfully deleted".
8. The voice mail will close

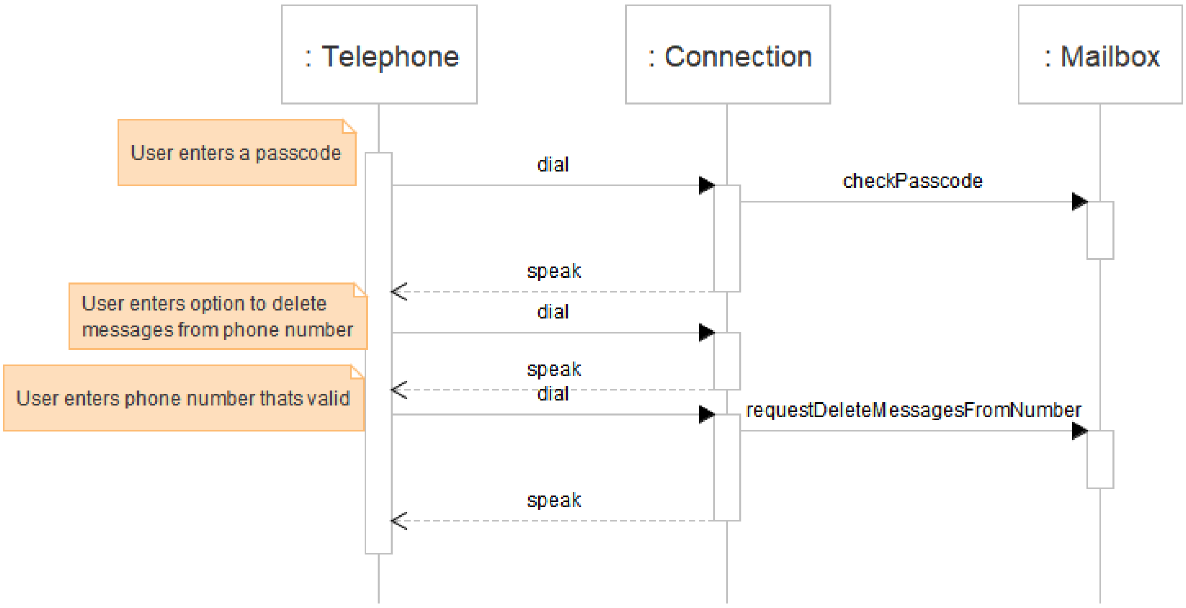
Variation 1

1.1. In step 6, Voice mail doesnt find number and speaks "Number not found"

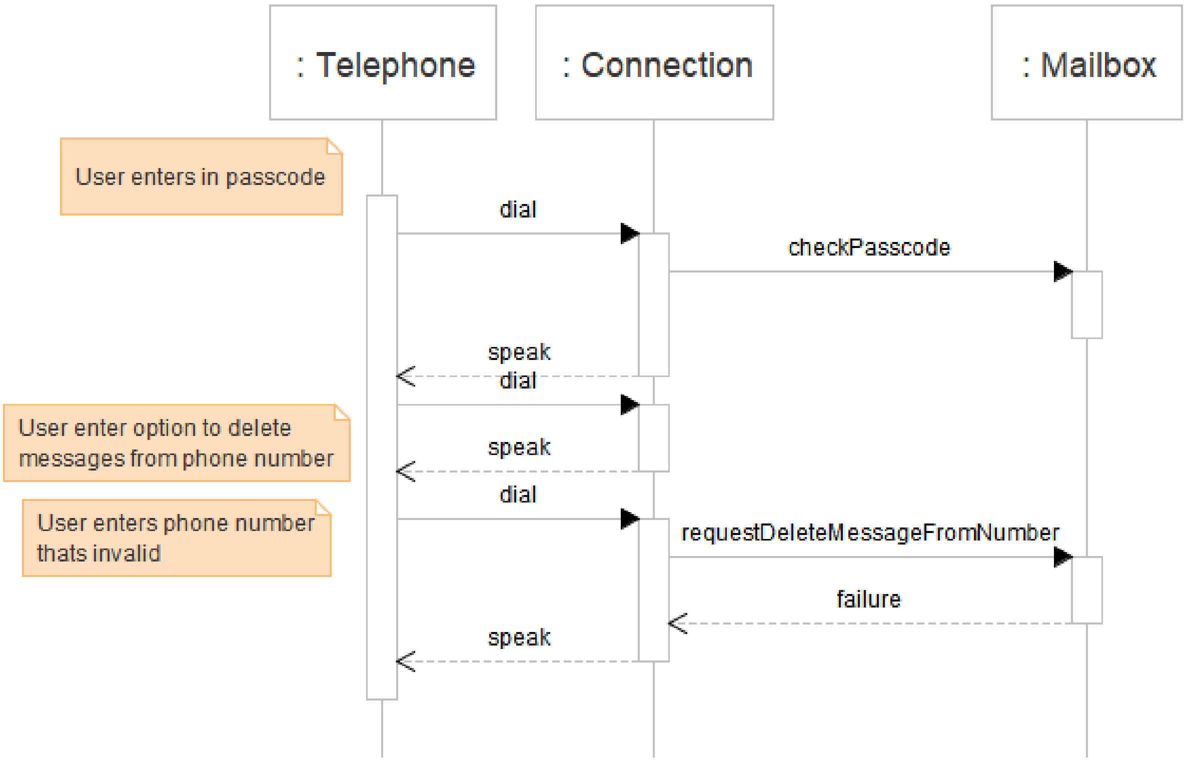
1.2. The voice mail will close

b. Sequence diagram

Success on deleting messages

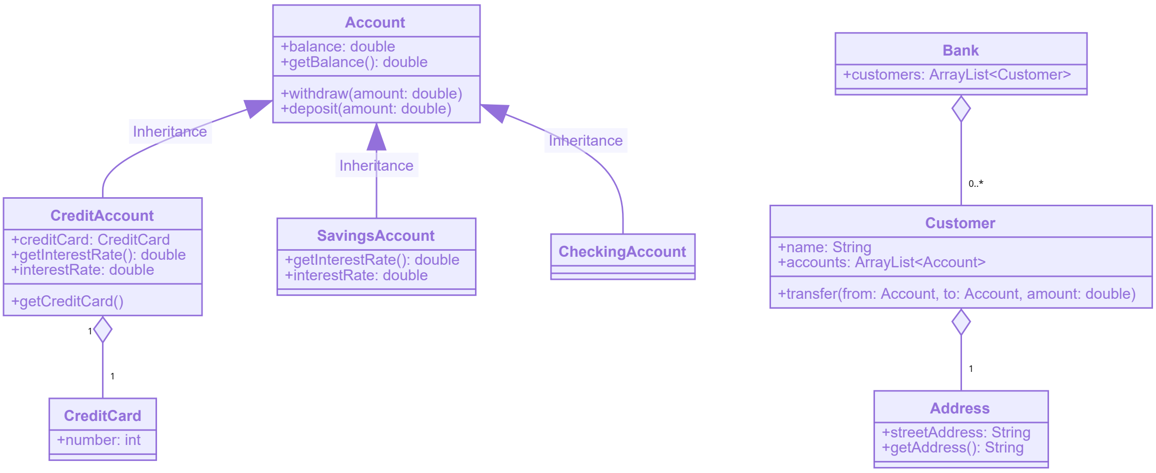

: Telephone
: Connection
: Mailbox
User enters a passcode
dial
checkPasscode
speak
User enters option to delete messages from phone number
dial
User enters phone number thats valid
speak
dial
requestDeleteMessagesFromNumber
speak

Failure on deleting messages


: Telephone
: Connection
: Mailbox
User enters in passcode
dial
checkPasscode
speak
dial
User enter option to delete messages from phone number
speak
dial
User enters phone number thats invalid
requestDeleteMessageFromNumber
failure
speak

2.2

Class diagrams



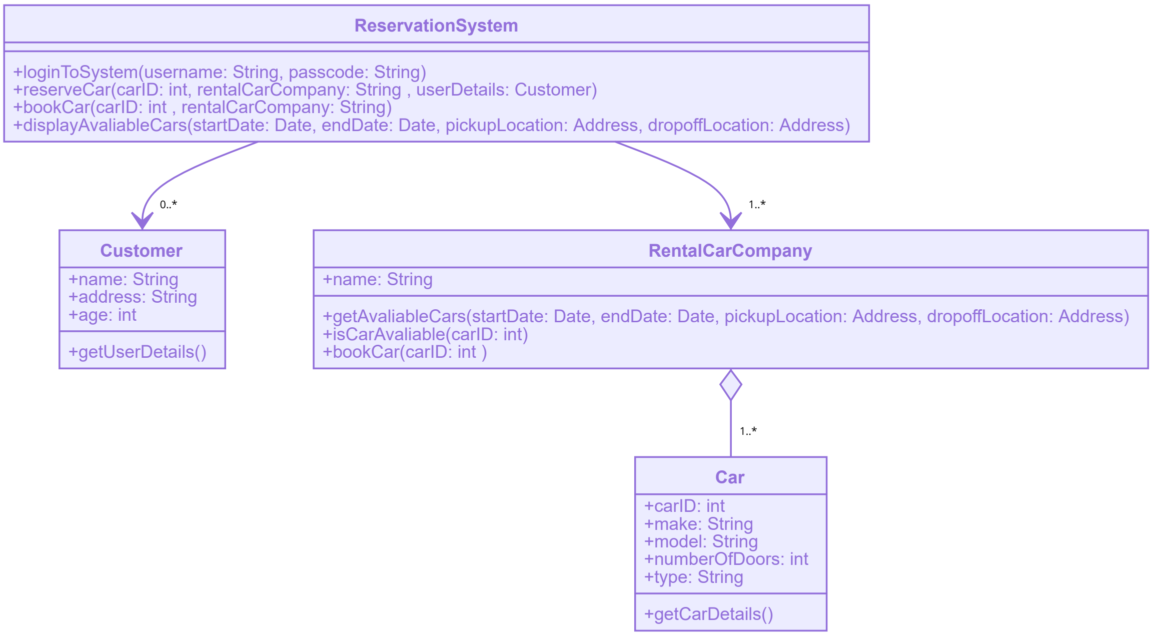
2.3

a - CRC cards

1. Car
   - Responsibilities
     - The item that will be reserved in the system
   - Collaborators
     - Rental car company owns the cars
     - Reservation system keeps track of the cars
     - Customer will use this asset

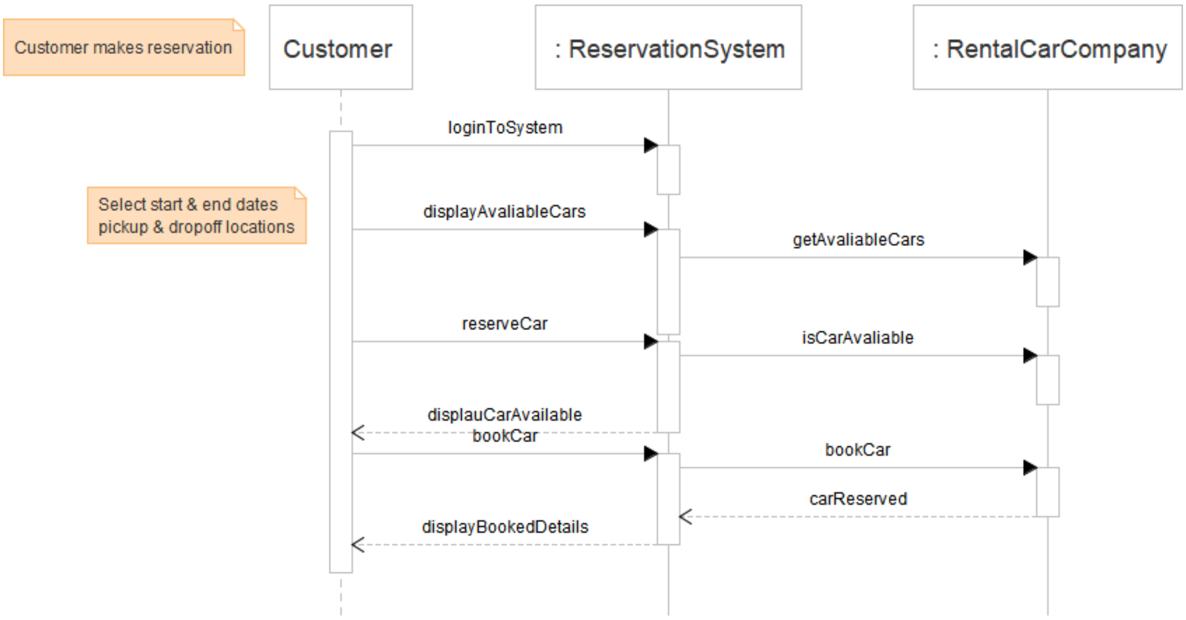
2. Reservation System
    - Responsibilities
        - Used to find available cars based on the Customers needs
        - Will coordinate between the Rental Car Company and the Customer on a booking
        - User accounts
    - Collaborators
        - Customer
        - Rental Car Company
3. Customer
    - Responsibilities
        - requesting car
    - Collaborators
        - Reservation System
4. Rental Car Company
    - Responsibilities
        - Providing available cars
    - Collaborators
        - Reservation System
        - Car

b - UML Class Diagram

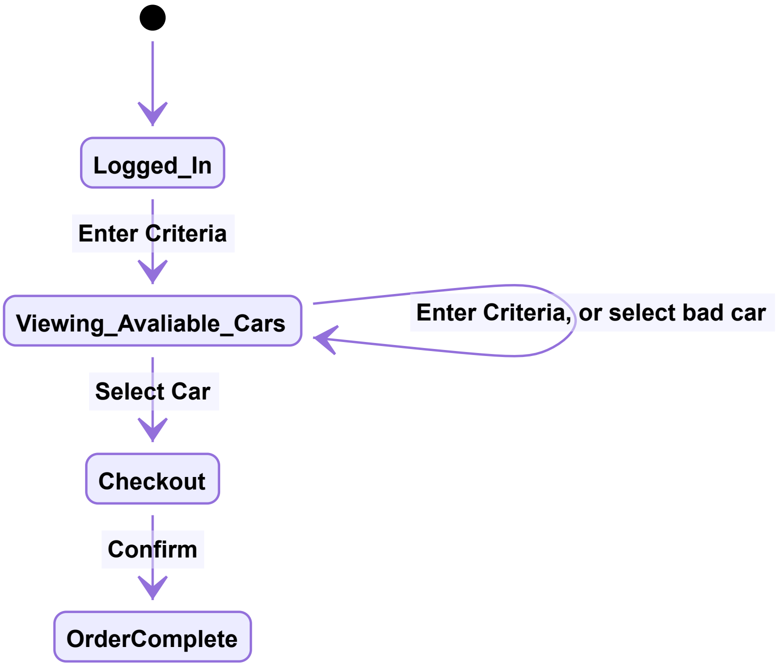

ReservationSystem
+loginToSystem(username: String, passcode: String)
+reserveCar(carID: int, rentalCarCompany: String , userDetails: Customer)
+bookCar(carID: int , rentalCarCompany: String)
+displayAvaliableCars(startDate: Date, endDate: Date, pickupLocation: Address, dropoffLocation: Address)
0..*
1..*
Customer
+name: String
+address: String
+age: int
+getUserDetails()
RentalCarCompany
+name: String
+getAvaliableCars(startDate: Date, endDate: Date, pickupLocation: Address, dropoffLocation: Address)
+isCarAvaliable(carID: int)
+bookCar(carID: int)
1..*
Car
+carID: int
+make: String
+model: String
+numberOfDoors: int
+type: String
+getCarDetails()

c - Sequence Diagram

Customer makes a reservation



d - State Diagram



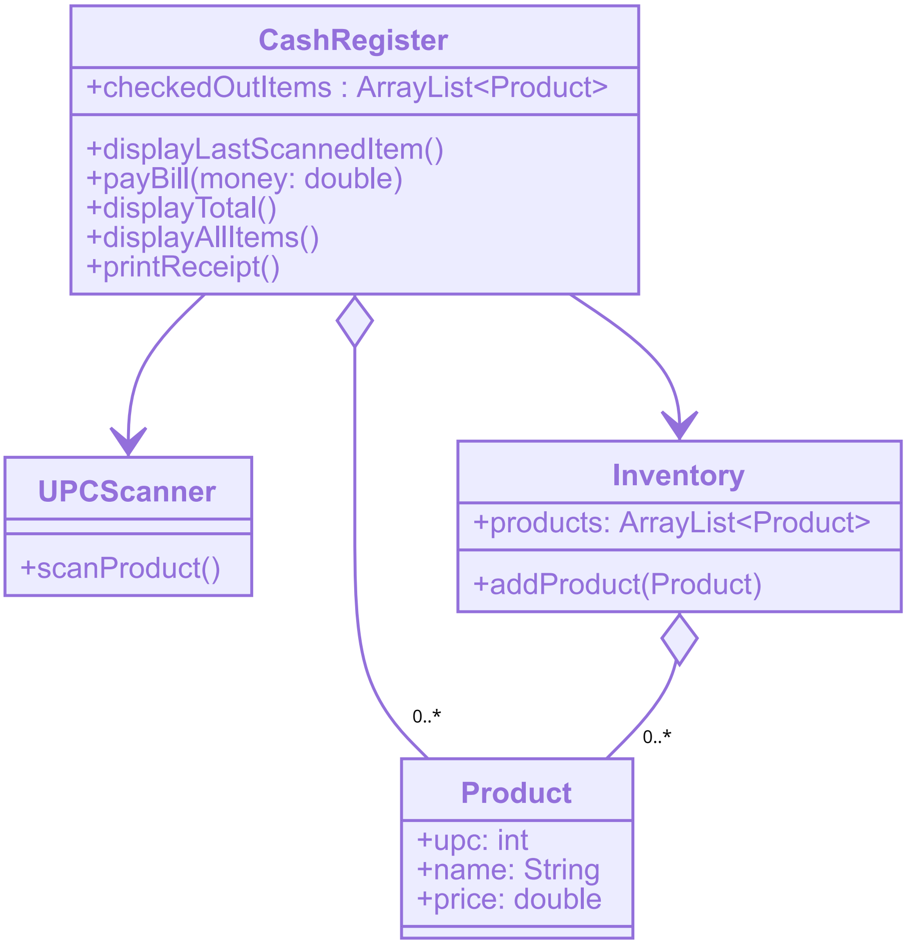
2.4

CRC cards

1. UPCScanner
   - Responsibilities
     - scan products
   - Collaborators
     - CashRegister
     - Cashier
2. Product
   - Responsibilities

        - just storing its own info
    - Collaborators
        - None
3. CashRegister
    - Responsibilities
        - Display last scanned item
        - Set system into payment mode
        - Display all items with their price and total price
        - pay bill
        - print receipt
    - Collaborators
        - Cashier
        - UPCScanner
4. Inventory
    - Responsibilities
        - storing all the products the store carries
    - Collaborators
        - CashRegister

UML



Code

q4

```
import java.util.Scanner;

class q4 {
    /**
     * This main method will load up the inventory
     * Then it will ask the user if they want to pay or scan an item
     * If the user incorrectly enters in an selection, it will ask the user to
retry
     * @param args
     */
    public static void main(String[] args) {
        CashRegister cashRegister = new CashRegister();

        cashRegister.inventory.add(new Product(123,"Candy", 5.99));
        cashRegister.inventory.add(new Product(111,"Apple", 2.99));
        cashRegister.inventory.add(new Product(222,"Water", 1.99));
        System.out.println("Welcome to the Store.");
        System.out.println("Press 1  - to scan an item");
        System.out.println("Press 2  - to pay");
        Scanner sc = new Scanner(System.in);
        while(true){
            String input = sc.nextLine();
            if(input.equals("1")){
                cashRegister.scanItem();
            }
            else if(input.equals("2")){
                if(cashRegister.getTotal() > 0.0) {
                    cashRegister.displayTotal();
                    cashRegister.displayAllItems();
                    while (true) {
                        try {
                            System.out.println("Enter payment");
                            input = sc.nextLine();
                            double cash = Double.parseDouble(input);
                            if (cashRegister.payBill(cash)) {
                                break;
                            }
                        } catch (Exception e) {
                            System.out.println("ERROR: Parsing double error.
Please enter in valid double");
                        }
                    }

                    break;
                }else {
                    System.out.println("You dont have any items to checkout");
                }
            }else{
                System.out.println("Incorrect entry.");
            }
            System.out.println("Press 1  - to scan an item");
            System.out.println("Press 2  - to pay");
        }
    }
}
```

CashRegister

```
import java.util.ArrayList;

/**
 * The Cash Register has access to the UPCScanner and the Inventory
 */
class CashRegister{
    ArrayList<Product> checkedOutItems = new ArrayList<>();
    Inventory inventory = new Inventory();
    UPCScanner upcScanner = new UPCScanner();
    public CashRegister(){

    }

    /**
     * This method will use the scanner to scan the object and look for the UPC in the inventory.
     * It will notify the user if the UPC was not found in the inventory
     * If the item was found, then it will display it to the screen
     */
    public void scanItem(){
        int newItem = upcScanner.scanProduct();
        boolean found = false;
        for(Product item : inventory.products){
            if(item.upc == newItem){
                checkedOutItems.add(item);
                displayLastScannedItem(item);
                found = true;
            }
        }

        if(!found){
            System.out.println("Item " + newItem + " not found");
        }

    }

    /**
     * @return total price of checked out items
     */
    public double getTotal(){
        double total = 0.0;
        for(Product product : this.checkedOutItems){
            total += product.price;
        }
        return total;
    }


    /**
     * This method will validate if the customer provided enough cash to pay the bill.
     * @param cash input amount from the customer
     * @return true == enough cash, false == short on cash
     */
    public boolean payBill(double cash){
        if(getTotal() > cash){
            System.out.println("Not enough cash. You provided " + cash + " where the balance was " + getTotal());
            return false;
        }
        else{
            double remainingBalance = cash - getTotal();
            printReceipt();
            System.out.println("Balance paid! Returning change: " + String.format("%3.2f", remainingBalance));
            return true;
        }
    }

    /**
     * @param product the item to display to the screen
     */
    public void displayLastScannedItem(Product product){
        System.out.println(product);
    }

    /**
     * Displays the total of the checked out items to the screen
     */
    public void displayTotal(){
        System.out.println("Total: " + String.format("%3.2f", getTotal()));
    }

    /**
     * Displays all the checked out items ot the screen
     */
    public void displayAllItems(){
        System.out.println("UPC\tItem\tPrice");
        for(Product product: checkedOutItems){
            System.out.println(product);
        }
    }

    /**
     * Displays all the items, total and that the bill has been paid
     */
    public void printReceipt(){
        System.out.println("--------------");
        displayAllItems();
        displayTotal();
        System.out.println("Paid!");
    }
}
```

Inventory

```
import java.util.ArrayList;

public class Inventory {
    public ArrayList<Product> products = new ArrayList<>();
    public Inventory(){
    }

    /**
     * Adds a new product to the inventory
     * @param product new product to be added to the inventory
     */
    public void add(Product product){
        products.add(product);
    }

}
```

Product

```
/**
 * The product class holds information related to the product
 */
class Product{
    public Integer upc = 0;
    public String name = "";
    public Double price = 0.0;

    public Product(Integer upc, String name, Double price){
        this.upc = upc;
        this.name = name;
        this.price = price;
    }

    /**
     * @return formatted string to display a product
     */
    public String toString(){
        return upc.toString() + "\t" + name + "\t$" + String.format("%3.2f",
price);
    }

}
```

UPCScanner

```
import java.util.Scanner;

class UPCScanner{
    public UPCScanner(){
    }

    /**
     * The method is used to capture the scanned UPC
     * @return UPC from user input
     * Exception: The scanner will keep trying if the user enters in an invalid
integer
     */
    public int scanProduct(){
        while(true) {
            try {
                System.out.print("Item UPC: ");
                Scanner sc = new Scanner(System.in);
                return Integer.parseInt(sc.nextLine());

            } catch (Exception e) {
                System.out.println("ERROR: Parsing int error. Please enter in
valid int for the UPC");
            }
        }
    }
}
```

2.5

a - Use Cases

Read Recent Posts

1. User logs into the system
2. System will display choices to edit profile, read posts, write new posts.
3. User will select read post
4. System will display all unread posts
5. User will acknowledge posts read

Edit Profile

1. User logs into the system
2. System will display choices to edit profile, read posts, write new posts.
3. User will select to want to edit profile
4. System will provide all items that can be edited
5. User will select and edit the items that need to be changed
6. System will take the modifications and update the User profile
7. System will acknowledge user that this has been complete.

Write Post

1. User logs into the system
2. System will display choices to edit profile, read posts, write new posts.
3. User will select to write a post
4. System will display an dialog to enter in text for the new post
5. User will enter in text and submit new post
6. System will receive the new post and store it in the server

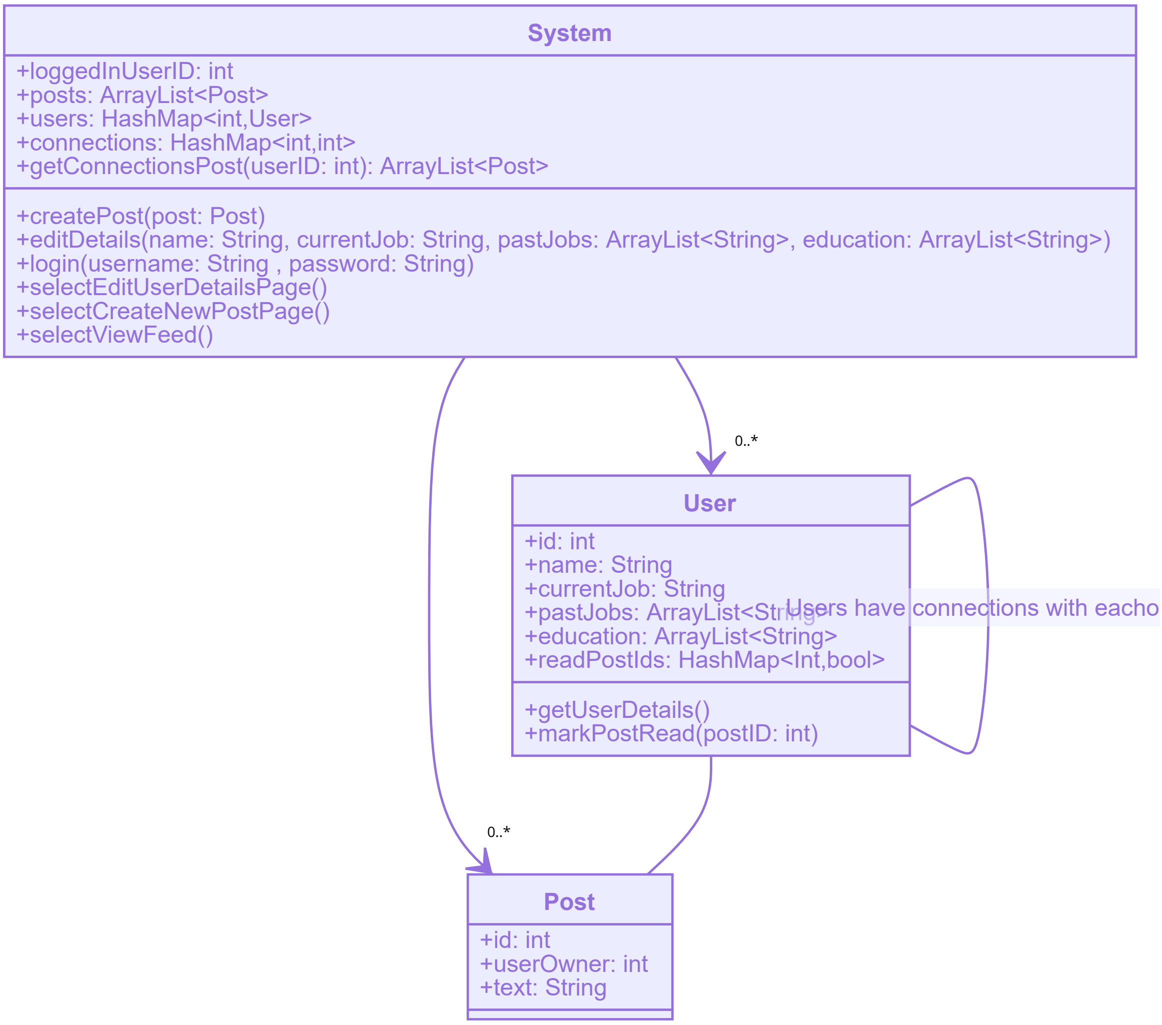
b - CRC cards

1. System
    - Responsibilities
        - Manage & display posts posts
        - Manage User accounts
        - Logging into accounts
        - Connections between users
    - Collaborators

        - Post
        - User

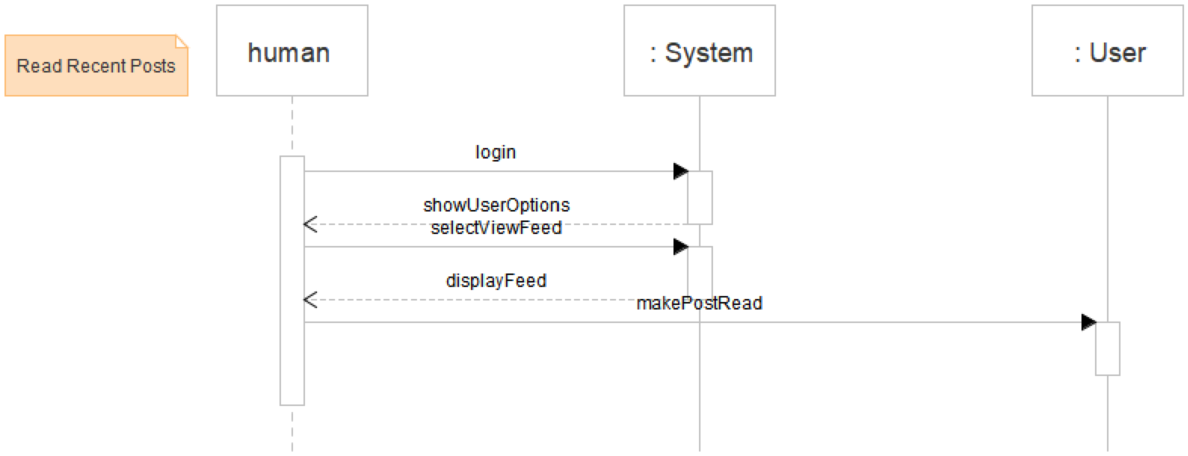
2. User
    - Responsibilities
        - Contain user info
        - Keep track of read posts
    - Collaborators
        - Post
3. Post
    - Responsibilities
        - Contain post text
    - Collaborators
        - Human creating post(no class needed)

c - UML Class Diagrams

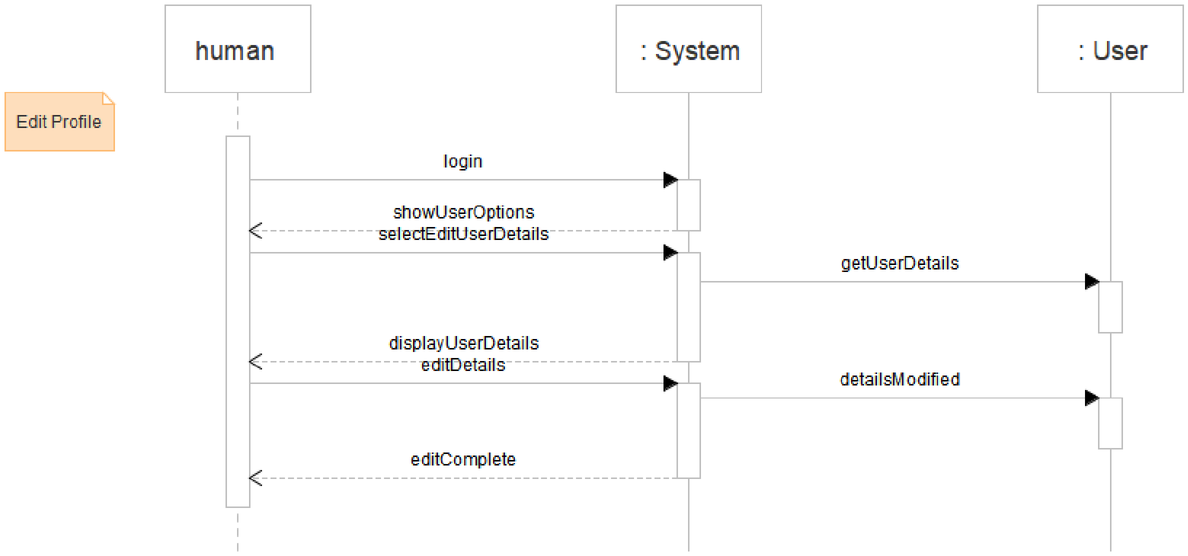

System
+loggedInUserID: int
+posts: ArrayList<Post>
+users: HashMap<int,User>
+connections: HashMap<int,int>
+getConnectionsPost(userID: int): ArrayList<Post>
+createPost(post: Post)
+editDetails(name: String, currentJob: String, pastJobs: ArrayList<String>, education: ArrayList<String>)
+login(username: String , password: String)
+selectEditUserDetailsPage()
+selectCreateNewPostPage()
+selectViewFeed()
0..*
User
+id: int
+name: String
+currentJob: String
+pastJobs: ArrayList<String>
+education: ArrayList<String>
+readPostIds: HashMap<Int,bool>
+getUserDetails()
+markPostRead(postID: int)
Users have connections with eacho
0..*
Post
+id: int
+userOwner: int
+text: String

d - UML Sequence Diagrams

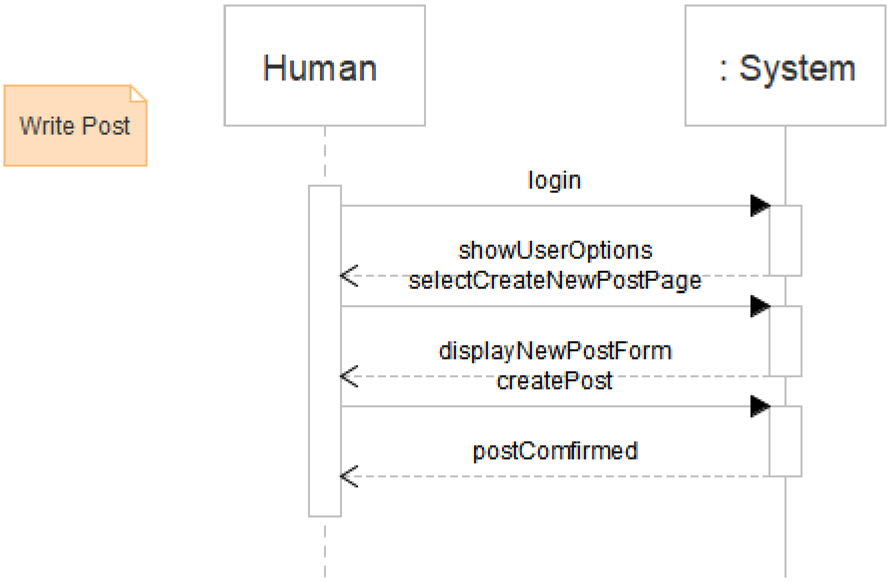
Read Recent Posts



Edit Profile



Write Post


Human
: System
Write Post
login
showUserOptions
selectCreateNewPostPage
displayNewPostForm
createPost
postComfirmed