ROYAL MELBOURNE INSTITUTE OF TECHNOLOGY

COSC 2440 - Further Programming
Assignment 1: Build a Console App

Student Name	**Phan Nam Nguyen**
Student Number	**3873792**
Lecturer	**Minh Vu Thanh**
Github Link	**COSC2440_A1**

I certify that this is all my own original work. If I took any parts from elsewhere, then they were non-essential parts of the assignment, and they are clearly attributed in my submission. I will show I agree to this honor code by typing "Yes": **Yes**.

Date of report **April 8th, 2024**

Table of Contents

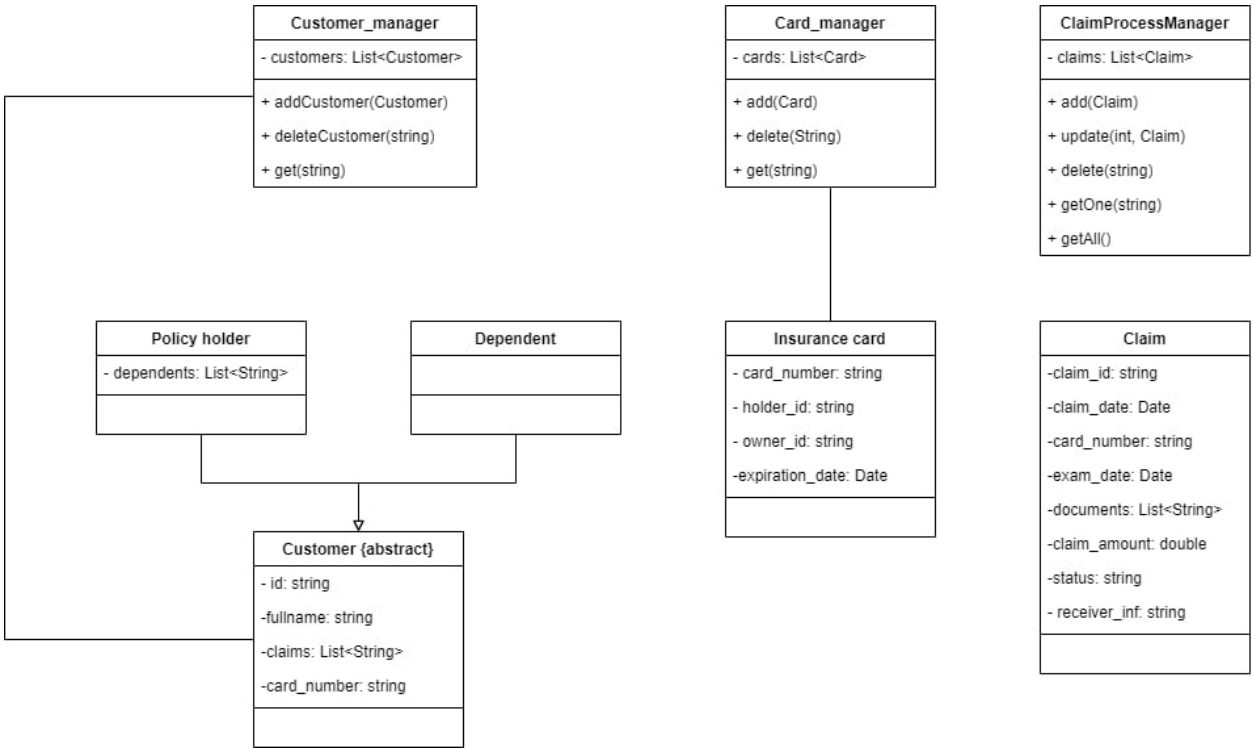
1. Application Description

This program is an insurance claims management system designed to streamline the process of managing, tracking, and processing claims. It caters to both policyholders and their dependents, ensuring efficient claim handling for all insured individuals.

- **Customer Management:**
  - Categorizes customers as policyholders or dependents.
  - Stores policyholder information including full name, ID (c-number format), and a list of dependents.
  - Stores dependent information including full name and ID (c-number format).
- **Insurance Card Management:**
  - Tracks insurance cards with details like card number (10 digits), card holder name, policy owner name, and expiration date.
  - Ensures a one-to-one relationship between a card and its holder.
- **Claim Management:**
  - Tracks claims with details like ID (f-number format), claim date (as Date object), insured person name, associated card number, exam date (as Date object), list of uploaded documents (with unique naming convention: ClaimId_CardNumber_DocumentName.pdf), claim amount, and current status (New, Processing, Done).
  - Allows uploading claim-related documents.
- **Reporting and Export:**
  - Enables sorting entities (customers, cards, claims) based on various criteria.
  - Generates reports displaying the sorted information in text files.
- **Data Persistence:**
  - Supports populating sample data for customers, cards, and claims from external files upon program initialization. (File format design is flexible as long as it allows storing at least 15 objects per file).

This system offers a comprehensive solution for insurance companies, allowing efficient claim handling, improved organization, and enhanced customer experience.

2. Class Diagram



- **Prevents Cyclic Dependency:** Without a two-way reference, I have eliminated the circular relationship between customers and insurance cards. This simplifies data manipulation and avoids potential issues during updates or deletions.
- **Data Normalization:** Storing related information (customer and card details) in separate entities promotes data normalization. This reduces data redundancy and ensures consistency across the system.
- **Improved Maintainability:** It becomes easier to modify or update customer or card information independently without affecting the other entity. Additionally, it allows for potential future enhancements where a customer might have multiple insurance cards.

3. API list (With brief description)

3.1. CardManager.

```
public class CardManager {
    private List<InsuranceCard> cards;
    private static CardManager instance;

    private CardManager() {…

    public static CardManager getInstance() {…

    public boolean containsCard(String cardNumber) {…

    public void addCard(InsuranceCard card) {…

    public void removeCard(String cardNumber) {…

    public List<InsuranceCard> getCards() {…

    public InsuranceCard getCardByNumber(String cardNumber) {…
}
```

- private CardManager(): Construct a new card manager object, can only be called once
- public static CardManager getInstance(): Get the singleton instance of this class, make sure this class produce only one object
- public boolean containsCard(String cardNumber): Check if a card exists
- public void addCard(InsuranceCard card): Add a new card to the list of cards
- public void removeCard(String cardNumber): Remove a card from the list of cards
- public List<InsuranceCard> getCards(): Return the list of cards
- public InsuranceCard getCardByNumber(String cardNumber): Return the card with the given card number

3.2. CustomerManager

```
public class CustomerManager {
    private List<Customer> customers;
    private static CustomerManager instance;

    private CustomerManager() {…

    public static CustomerManager getInstance() {…

    public boolean containsCustomer(String id) {…

    public void addCustomer(Customer customer) {…

    public void removeCustomer(String id) {…

    public Customer getCustomerByID(String id) {…

    public List<Customer> getCustomers() {…

    public List<Customer> getPolicyHolders() {…

    public List<Customer> getDependants() {…
}
```

- private CustomerManager(): Construct a new CustomerManager object
- public static CustomerManager getInstance(): Return the instance of the CustomerManager object, make sure that only one instance of the CustomerManager object is created
- public boolean containsCustomer(String id): Check if a customer with the given id exists
- public void addCustomer(Customer customer): Add a new customer

- public void removeCustomer(String id): Remove a customer with the given id
- public Customer getCustomerByID(String id): Return the customer with the given id
- public List<Customer> getCustomers(): Get all customers
- public List<Customer> getPolicyHolders(): Get all policy holders
- public List<Customer> getDependants(): Get all dependants

3.3. ClaimProcessManager.

```
public class ClaimProcessManager {
    private List<Claim> claims;
    private static ClaimProcessManager instance;

    private ClaimProcessManager() {…

    public static ClaimProcessManager getInstance() {…

    public boolean containsClaim(String claimID) {…

    public void addClaim(Claim claim) {…

    public void updateClaim(Claim newClaim) {…

    public void removeClaim(String claimID) {…

    public Claim getClaimByID(String claimID) {…

    public List<Claim> getClaims() {…
}
```

- private ClaimProcessManager(): Construct a new ClaimProcessManager object.
- public static ClaimProcessManager getInstance(): Return the singleton instance of the ClaimProcessManager class.
- public boolean containsClaim(String claimID): Check if the claim with the given claimID is in the list of claims

- public void addClaim(Claim claim): Add a new claim to the list of claims.
- public void updateClaim(Claim newClaim): Update the claim with the corresponding claim
- public void removeClaim(String claimID): Remove the claim with the given claimID from the list of claims.
- public Claim getClaimByID(String claimID): Return the claim with the given claimID.
- public List<Claim> getClaims(): Get all the claims

4. Any drawback and Future Work

4.1. Drawbacks:

- **Limited Scalability for Large Datasets:** The current implementation might not be optimized for handling very large datasets efficiently. As the number of customers, cards, and claims grows, the system's performance could potentially degrade.
- **Centralized Driver Code:** The driver code, currently residing primarily in the App.java file, might become cumbersome to maintain as the system evolves. Its centralized location could make it difficult to manage and understand the overall program flow.

4.2. Future Work:

- **Data Structure Optimization:** Implementing data structures like HashMap could significantly improve query performance and overall data management. HashMap allows for faster retrieval of specific objects based on key-value pairs, making it efficient for searching and filtering large datasets.
- **Modularization of Driver Code:** Breaking down the driver code in App.java into smaller, more focused classes and functions would enhance code readability, maintainability, and reusability. Each class or function could handle a specific aspect of the system, promoting better organization and easier future modifications.
- **Additional Features (Optional):**
    - Consider implementing functionalities like user authentication and authorization for secure access to claim information.
    - Develop a graphical user interface (GUI) to provide a more user-friendly experience for interacting with the system.

These improvements would address the current limitations and pave the way for a more scalable and robust claims management system.