# CSEN4174: Software Engineering Lab

## Assignment 3

**Team number: \_\_\_\_\_\_\_\_\_4\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Team name: \_\_\_\_\_\_\_\_\_\_Decrypt\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

*Before saving this file, please rename the file as 25-Odd-CSEN4174-Lab01-YYYYZZ-Name.docx, where “YYYY” is either AIML or DASC depending on whether you are AIML or Data Science students, “ZZ” is your team number and “Name” is the name of your team. Incorrectly named files will not be evaluated.*

### **Activity 3.1: Analysis classes**

Analysis aims at a conceptualization of a system’s components in terms of assigning responsibilities. The focus here is entirely on abstraction; without design or implementation concerns. *Analysis classes* are not classes in the strict object-oriented sense of the term. But “actual” classes can grow out of analysis classes. Analysis classes are of three types: boundary, entity, and control. Let us try to visualize how a software system serves users. First, the system should offer some way to accept user input and exhibit system output back to the user. Second, the system must be able to store and process the information users have supplied, or has been generated within the system, or gathered from other systems. Third, based on user inputs and the stored or processed information, the system must be able to take decisions towards fulfilling user needs. These three aspects map to the boundary, entity, and control classes respectively. Boundary classes facilitate information transfer in and out of the system. Entity classes help in the storage and retrieval of information within the system. Control classes govern the processing of information, and the system’s decision making. To reiterate an important point, analysis classes are the preliminary placeholders of functionality, and not the final components to be implemented in code. It is quite likely that one boundary or entity or control class identified during analysis spawns multiple components or becomes redundant during design.

How do we identify analysis classes and their collaborations? To get started, there is a simple rule of thumb. Given a particular use case in natural language, identify the nouns (or noun phrases). These are candidates for analysis classes. Of course, not all nouns are equally important in the system’s overall context. Which noun is important and which is not can be judged in the context of the system. Out of the nouns picked as analysis classes, some will be demoted to being merely attributes (data members) of classes during design. Once the significant nouns are selected, attention is turned to the verbs (or verb phrases). Verbs associating one noun with another indicate collaboration between the analysis classes. For example, let us analyse the statement: “an **accountholder** can **transfer money** between connected **accounts”**. The significant nouns are “account-holder”, “money”, and “account”. These are candidates for analysis classes, perhaps entity classes, since they are associated with some information uniquely identifying them. During design, we will decide whether or not to keep “money” as a separate class or make it an attribute of the “account” class. “Transfer” is the verb which connects “money” with “account” and “account holder”. The act of transferring money between accounts by an account-holder suggests collaboration between these entities, which is realized by the sending of messages, which translates to method calls between components. So we can think of a transfer() method called on the source account by an account-holder, passing as parameter the amount of money to be transferred and the destination account identifier.

*The above discussion has been adapted from Software Engineering: Concepts and Applications by Subhajit Datta; Oxford University Press (2010) - Chapter 14*

**Revisit the use cases you identified and elaborated for the first iteration (Assignment 2). Using the method described above, specify the analysis classes (at least 10), their type (boundary, entity, and control) identified from each use case and justify your choice.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sl. No.** | **Name of the analysis class** | **Type (boundary, entity, and control)** | **Derived from use case ID** | **Use case name** | **Justification of the choice of analysis class** |
| **1** | **Student** | **Entity** | **UC1**  **, UC2, UC3, UC4, UC5** | **Student Registration & Verification, Student Login, Rent a Locker, Access Locker, Return Locker** | **Represents the main user; stores personal info, verification status, and locker interaction history.** |
| **2** | **Registration Form** | **Boundary** | **UC1** | **Student Registration & Verification** | **Interface for students to enter registration details (ID, name, email).** |
| **3** | **OTP Service** | **Control** | **UC1, UC4** | **Student Registration & Verification, Access Locker** | **Generates and verifies OTPs for both registration and locker access.** |
| **4** | **Authentication Controller** | **Control** | **UC2** | **Student Login** | **Validates credentials, checks verification, issues JWT tokens.** |
| **5** | **Locker** | **Entity** | **UC3, UC4, UC5** | **Rent a Locker, Access Locker, Return Locker** | **Represents a physical locker, its status, and linkage to a student.** |
| **6** | **Locker Management Interface** | **Boundary** | **UC3, UC4, UC5** | **Rent a Locker, Access Locker, Return Locker** | **UI for viewing, renting, accessing, and returning lockers.** |
| **7** | **Rental Record** | **Entity** | **UC3, UC5** | **Rent a Locker, Return Locker** | **Stores details of locker rentals, including dates and student IDs.** |
| **8** | **Rental Controller** | **Control** | **UC3, UC5** | **Rent a Locker, Return Locker** | **Handles allocation, return processing, and penalty calculation.** |
| **9** | **Audit Log** | **Entity** | **UC3, UC4** | **Rent a Locker, Access Locker** | **Records important actions for accountability and system tracking.** |
| **10** | **Payment Processor** | **Control** | **UC3** | **Rent a Locker** | **Manages mock payment transactions for locker rentals.** |

**Activity 3.2: Class-Responsibility-Collaboration (CRC) cards**

Class-responsibility-collaboration (CRC) cards -- also known as class-responsibility-collaborator cards -- are a brainstorming tool for software design. The card is partitioned into three areas:

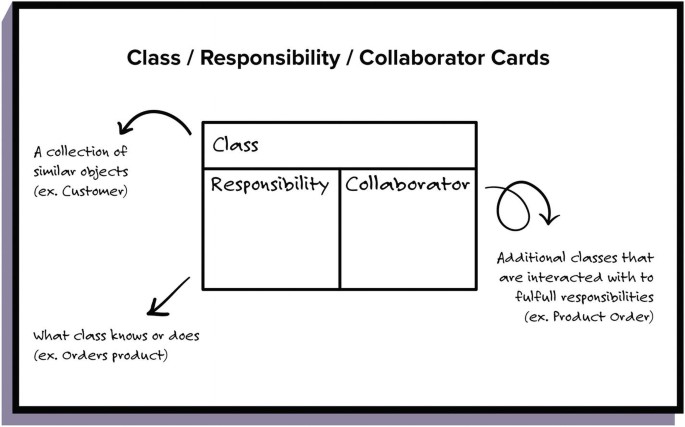
1. On top of the card, the class name
2. On the left, the responsibilities of the class
3. On the right, collaborators (other classes) with which the class interacts to fulfil its responsibilities

Using small cards minimizes the complexity of the design, reduces class responsibilities and keeps designers focused on the essentials of the classes without exploring implementation details.

*The above description is adapted from Class-responsibility-collaboration card:* [*https://en.wikipedia.org/wiki/Class-responsibility-collaboration\_card*](https://en.wikipedia.org/wiki/Class-responsibility-collaboration_card)

*You may refer to the original paper introducing CRC cards - A Laboratory For Teaching Object-Oriented Thinking:* [*http://c2.com/doc/oopsla89/paper.html*](http://c2.com/doc/oopsla89/paper.html)

Here is a typical structure of a CRC card:



<https://media.springernature.com/lw685/springer-static/image/chp%3A10.1007%2F978-1-4842-4206-3_23/MediaObjects/470826_1_En_23_Figa_HTML.jpg>

*You may read about an example scenario involving CRC cards at* [*https://www.math-cs.gordon.edu/local/courses/cs211/ATMExample/CRCCards.html*](https://www.math-cs.gordon.edu/local/courses/cs211/ATMExample/CRCCards.html)

**For each of the analysis classes you identified in the previous exercise, create a CRC card using the following template (there can be as many or as few responsibilities or collaborators):**

|  |  |
| --- | --- |
| **Class: Student** | |
| **Responsibility** | **Collaborators** |
| **• Provide personal details for registration** | **• Registration Form** |
| **• Login to access system features** | **• Authentication Controller** |
| **• Request locker rental, access, and return** | **• Locker Management Interface** |

|  |  |
| --- | --- |
| **Class: Registration Form** | |
| **Responsibility** | **Collaborators** |
| **• Collect student registration data** | **• Student** |
| **• Send registration request to OTP Service** | **• OTP Service** |

|  |  |
| --- | --- |
| **Class: OTP Service** | |
| **Responsibility** | **Collaborators** |
| **• Generate OTP** | **• Registration Form** |
| **• Verify OTP** | **• Locker Management Interface** |

|  |  |
| --- | --- |
| **Class: Authentication Controller** | |
| **Responsibility** | **Collaborators** |
| **• Validate credentials** | **• Student** |
| **• Issue JWT tokens** | **• Auth DB** |

|  |  |
| --- | --- |
| **Class: Locker** | |
| **Responsibility** | **Collaborators** |
| **• Maintain locker status** | **• Rental Record** |
| **• Link locker to rental record** | **• Rental Controller** |

|  |  |
| --- | --- |
| **Class: Locker Management Interface** | |
| **Responsibility** | **Collaborators** |
| **• Display available lockers** | **• Locker** |
| **• Provide access and return options** | **• Rental Controller** |

|  |  |
| --- | --- |
| **Class: Rental Record** | |
| **Responsibility** | **Collaborators** |
| **• Store rental start and end dates** | **• Locker** |
| **• Store student and locker IDs** | **• Student** |

|  |  |
| --- | --- |
| **Class: Rental Controller** | |
| **Responsibility** | **Collaborators** |
| **• Assign lockers** | **• Locker** |
| **• Process returns** | **• Rental Record** |
| **• Calculate penalties** | **• Payment Processor** |

|  |  |
| --- | --- |
| **Class: Audit Log** | |
| **Responsibility** | **Collaborators** |
| **• Record all important locker-related actions** | **• Locker Management Interface** |
|  | **• Rental Controller** |

|  |  |
| --- | --- |
| **Class: Payment Processor** | |
| **Responsibility** | **Collaborators** |
| **• Process locker rental payments** | **• Rental Controller** |
|  | **• Locker Management Interface** |