Object Oriented Programming in C++

Nepal College of Information Technology

Course Instructor: Er. Rabina Chaudhary

Chapter 7 : Object Oriented Design

Responsibility implies non-interference:

- Responsibility implies a degree of independence or non-inference
- Example: when you send flowers to your grandfather at Pokhara, it was not necessary to think how the request would be served. The florist, having taken the responsibility for this service, is free to operate without inference from the customer

- In Conventional programming one portion of code is dependent on many other portions of the code of system
- Such dependencies arise through use of global variables, pointer variables or inappropriate use of and dependencies of implementation details on other portion of code

- Responsibility driven design make the portion of code as independent as possible
- Responsibility driven design support information hiding ans it becomes more important when one move from programming in small to programming in large

Programming in Small:

- Code is developed by a single programmer or very small number of programmers
- A single individual can understand all the aspects of a project

Programming in large:

- Software system is developed by a large team
- Team consists of people with many different skills
- A single individual may not necessarily understand all aspects of the project
- Major problem in software development is team management and communication of information between different parts of the project

Role of behavior in OOP: why begin with behavior

- Object oriented design process begins with the analysis of behavior
- Behavior can be described from the moment an idea is conceived and can be described in terms meaning to both the programmers and the client

Responsibility Driven Design:

- Developed by Rebecca Wirfs- Brock
- It is an object oriented design technique that is driven by a emphasis on behavior at all levels of development
- In object oriented design, first we establish who is responsible to for which action of the system
- The responsible class to carry out an action may communicate with other classes to complete its responsibility

- Responsibility is anything that a class knows or does

- Responsibility is anything that a class knows or does
- Example:

class Student might have knowledge for its data members : registration number, roll number, name, address, faculty

and it may perform actions like attend class, take exam, view result, pay fee

Collaborator:

- Collaborator is another class that is used to interact with a class
- Collaborator for a class is also a class which helps to perform first class’s action

Class Responsibility Collaborator:

- CRC card:

➢Identify class

➢Identify list of responsibility

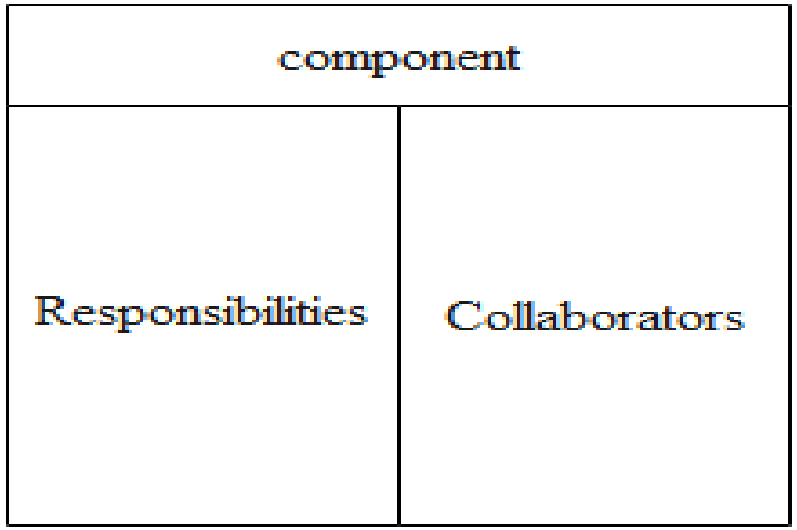
➢List collaboration

Class Responsibility Collaborator card:

- Consists of three sections:

1. Class : a class represents a collection of similar objects
2. Responsibility: it is something that a class knows or does
3. Collaborator: another class that is a class interacts with to fulfill its responsibilities

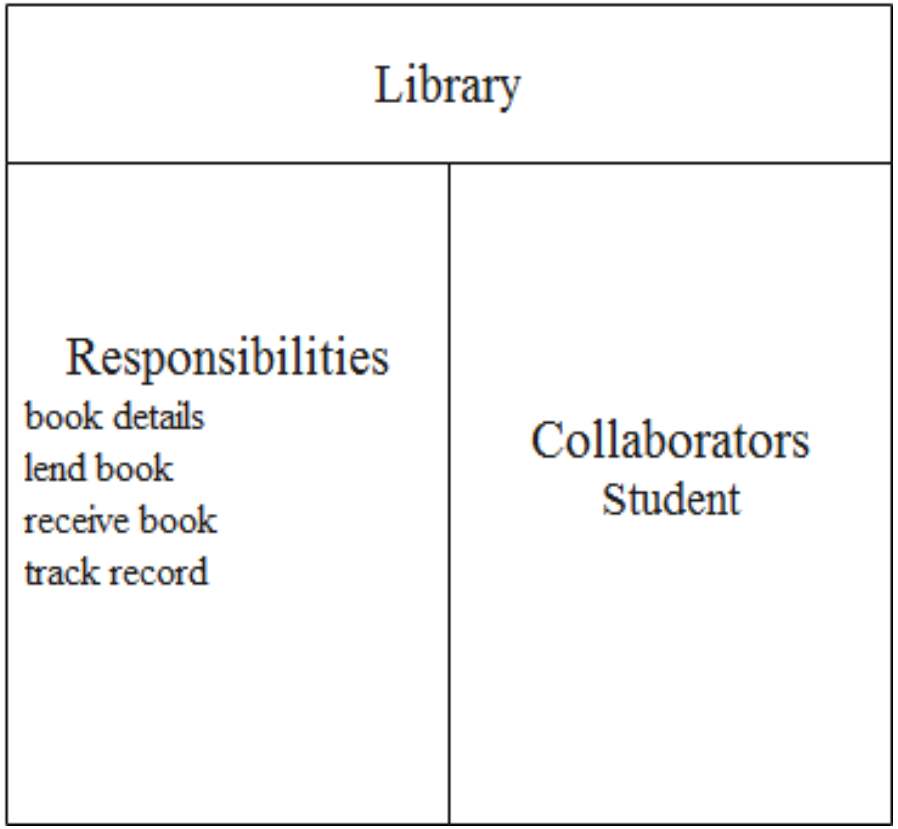
CRC card:



CRC card for Student:

Student	
Responsibilities name roll number registration number attend class fill examination form take exam request for transcript pay fee borrow book return book	**Collaborators** ExamDepartment AccountDepartment Library

CRC card for Library:



Identification of Component:

- The complex system is decomposed into smaller units
- A component is simply an entity that can perform tasks which fulfills some responsibilities
- A component can be function, structures or class or collection of other components
- The identification of component takes place during the process of imagining the execution of a working system

Documentation:

- Two document that are essential part of any software system
    1. user manual
    2. system design documentation

Coupling and Cohesion:

- If any services provided by component is done within the component then it is cohesive
- If the component cannot do all the work itself and interacts with other components to do the work then it is coupling
- It is desirable to reduce the amount of coupling as much as possible.
- Coupling is not good because of dependency in the components, if problem in one, the problem occurs in another too

Interface and Implementation:

- It is possible for one programmer to know how to use a component developed by another programmer, without needing to know how the component is implemented
- The interface view is the side seen by other programmers. It describes what a software component can perform
- The implementation view is the side seen by the programmer working on a particular component. It describes how a component is performing a task

Formalize the interface:

- Component with only one behavior can be made into function
- Components with many tasks are implemented as classes
- Names given to each of the responsibilities identified on the CRC card for each component are mapped into function names along with function name, the argument to be passed to the function is identified
- If the component requires some data to perform a specific task, the source of the data, either through argument or global value or maintained internally by the component, must be clearly identified

Coping with names:

- The selection of useful name is important
- Names should be meaning and according to the context of the problem
- Use pronounceable name
- Use capitalization or underscore to mark beginning of a new word within a name

    use BankAccount, Bank_account rather than bankaccount

- Avoid names with multiple interpretations
- Avoid digits within a name (0 can be misread as O, 2 can be misread as Z, 5 can be misread as S]
- Name functions and variables that yield Boolean values so they describe the interpretation of true and false value

  StudentIsPresent clearly indicates that a true value and student is present

  StudentStatus does not give clear indication

- Once the names have been developed for each activity, the CRC cards for each component are redrawn with the names and formal arguments

Designing the Representation:

- The design teams are divided into groups, where each group is responsible for one or more software components

- The description of behavior must be transformed into algorithm

Implementing Components:

- After designing each software subsystem, the next step is to implement the desired activities in a computer language

Integration of Components:

- Once the software subsystem are individually designed and tested, they can be integrated into a final product

Sequence diagram:

- Sequence diagrams is used to model the interactions between objects in a single use case
- They illustrate how the different parts of a system interact with each other to carry out a function.

Sequence diagram:

- **Lifeline**

A lifeline represents an individual participant in the Interaction

- **Activations**

A thin rectangle on a lifeline) represents the period during which an element is performing an operation

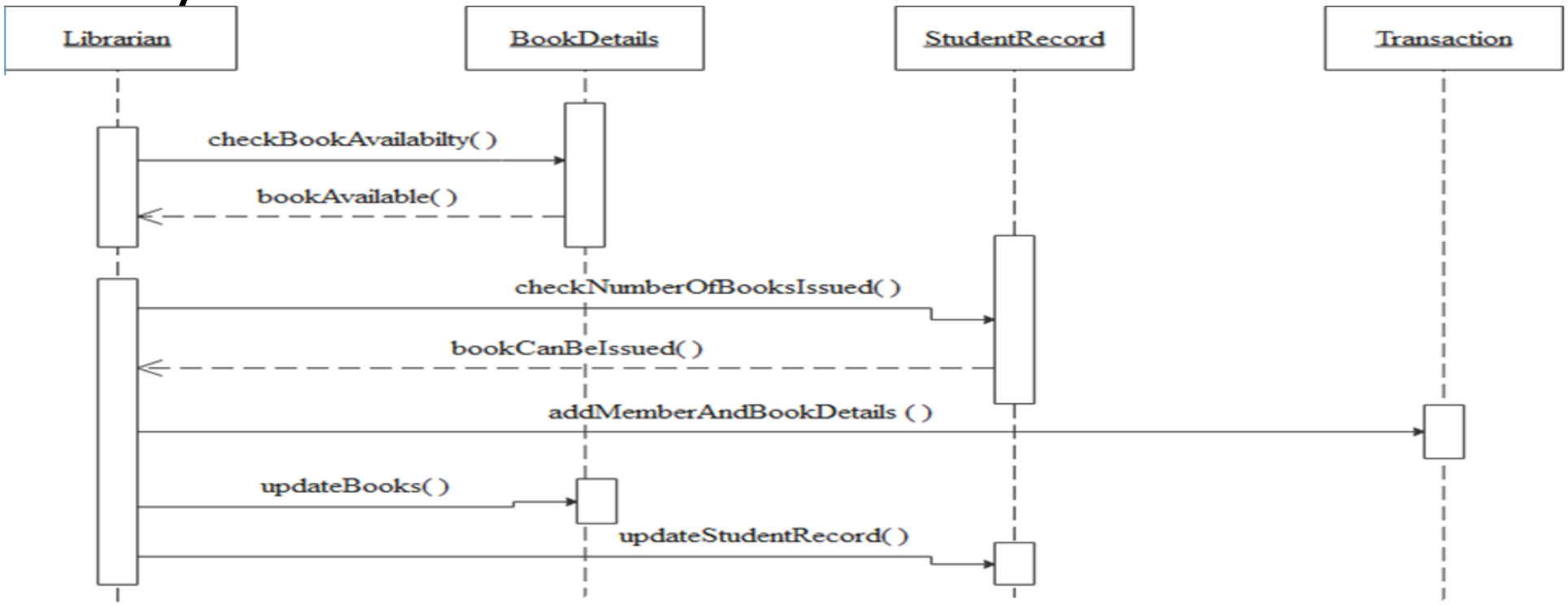
- **Call Message**

A message defines a particular communication between Lifelines of an Interaction

- **Return Message**

Return message is a kind of message that represents the pass of information back to the caller of a corresponded former message

Sequence diagram of borrowing book from library:



Practice:

Q1. Design a CRC card of ATM card reader

Q2. Design a sequence diagram to withdraw money from ATM