Object Oriented Programming in C++

Nepal College of Information Technology

Course Instructor: Er. Rabina Chaudhary

Chapter 1:

Object Oriented Concepts

Object Oriented Programming

- Programming model which is based upon the concept of objects
- Programs are organized as cooperative collections of objects, each of which represents an instance of some class
- Importance is given to data rather than the algorithm
- Data and functions are tied together
- Data protection or security of data is achieved by data hiding using access specifiers
- Uses bottom up approach in program design

Characteristics of Object Oriented Programming

1. Classes
2. Object
3. Encapsulation
4. Inheritance
5. Polymorphism
6. Data Abstraction
7. Message Passing

Characteristics of Object Oriented Programming

1. Object :

- Real world entities
- Object is entity that has characteristics and behavior
- Object is instance of a class

-If Ram is a Student than

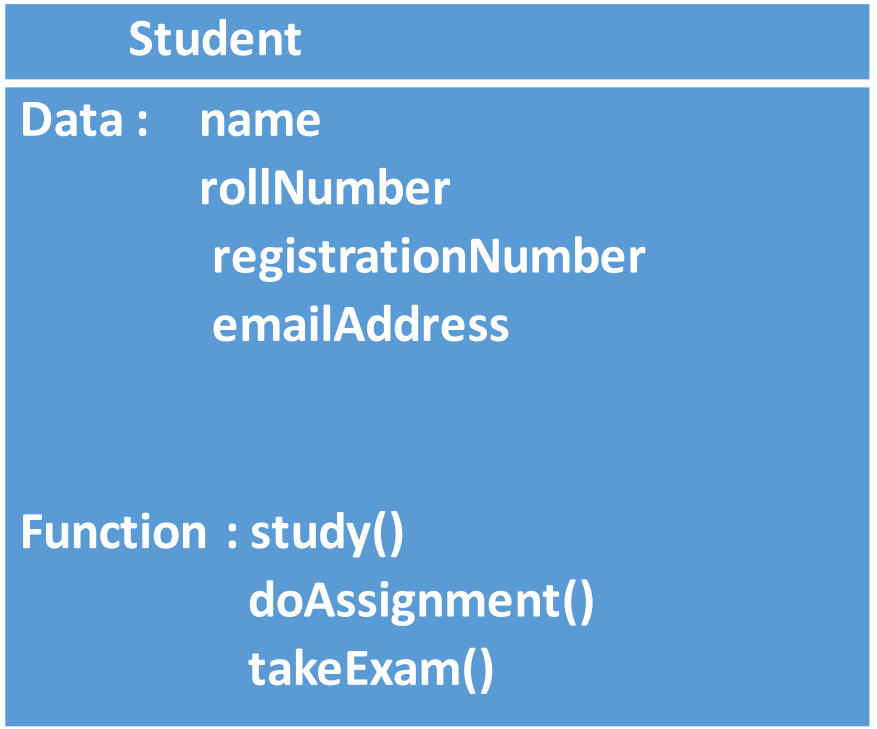
- He has characteristics like name, address, registrationNumber, marks
- He has behavior like studying, taking exam, doing assignment

Characteristics of Object Oriented Programming

2. Classes :

- A group of objects that share common characteristics and behavior

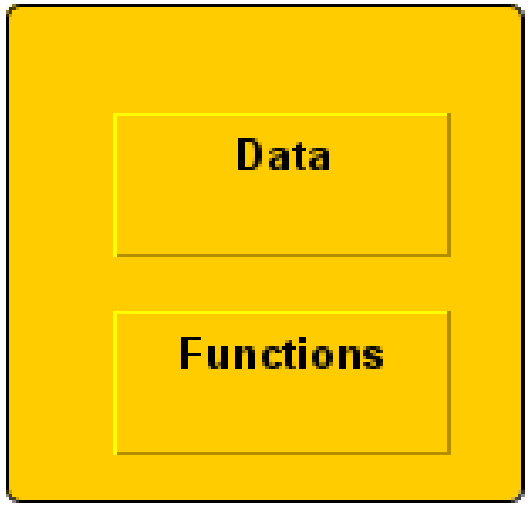
Example: Student



Characteristics of Object Oriented Programming

3. Encapsulation :

- Binding together the data and functions
- Prevents the data from unauthorized access



Characteristics of Object Oriented Programming

4. Inheritance :

- Ability to create new classes from the already existing class
- The existing class is called **base class** and the new class is called **derived class**
- Property of base class is inherited by derived class
- Derived class has data and functions of its parent class as well as its own data and functions
- Inheritance supports reusability

Characteristics of Object Oriented Programming

5. Data Abstraction:

- Abstraction refers representing only essential features or information and hiding background details

6. Polymorphism:

- Polymorphism means having many form
- Polymorphism can be achieved through function overloading and operator overloading

Characteristics of Object Oriented Programming

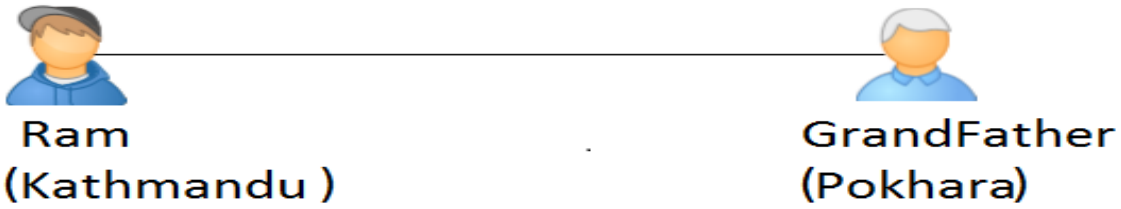
7. Message Passing :

- Objects communicate with each other by sending and receiving information to each other to perform certain task
- Message for an object is a request for execution of a procedure
- Invoke the function in receiving object that generates the desired result

Object.functionName(information)

A way of viewing the world:

Ram who lives in Kathmandu wants to send flowers to his grandfather who is in Pokhara.



A way of viewing the world:

- Ram logins to “FlowersNepal” application.
- Ram tells FlowersNepal kinds of flowers and address to which the flowers needs to be delivered
- Ram makes payment and can be assured that flowers will be delivered as expected

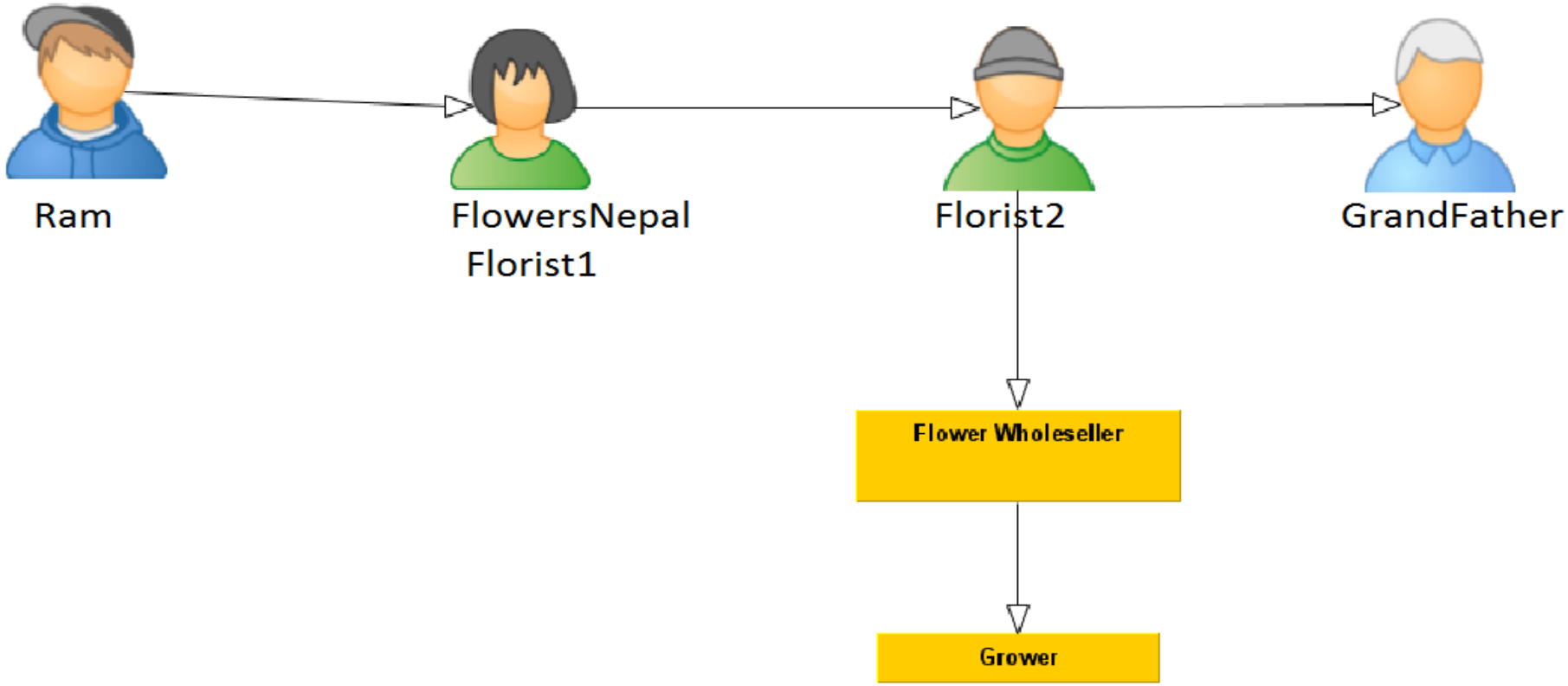
To solve the problem, the mechanism we used to was find an appropriate agent(FlowersNepal) and passed the agent a message containing request.

After that it is the responsibility of FlowersNepal to satisfy the request.

A way of viewing the world:

- FlowersNepal may have some sets of operations to do this. But Ram doesn’t need to know what methods or operations.
- This information is usually hidden
- Now, FlowersNepal will send a message to a florist in Pokhara with another message.
- Florist in Pokhara may have obtained the flowers from flower wholesaler who in turn had interactions with flower growers
- Florist in Pokhara passes the flowers to a delivery person with another message and so on.

Community of agents helping delivery of flowers



A way of viewing the world:

Message and methods:

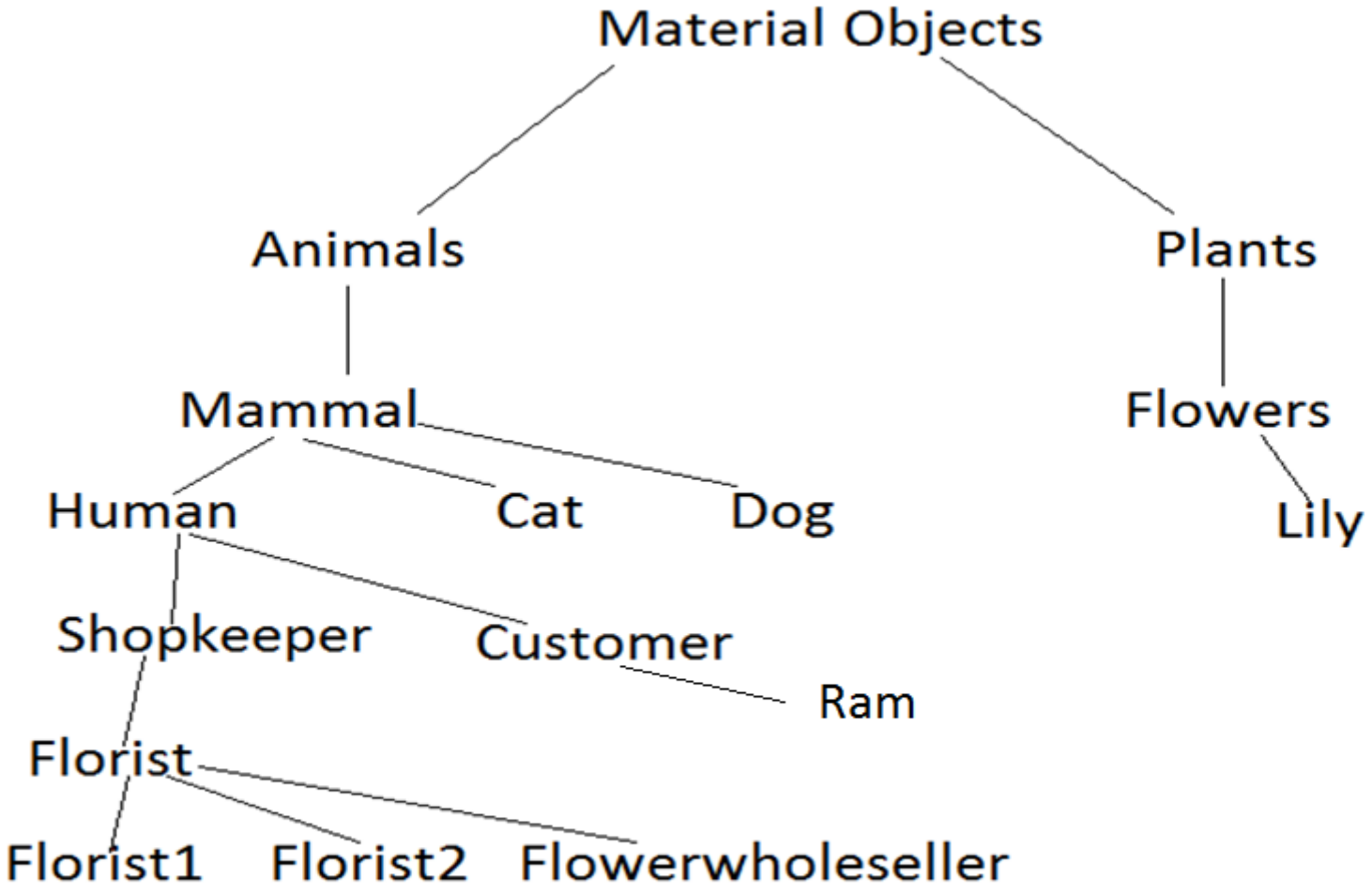
The chain of message that started from request given by Ram to “FlowersNepal” to solve Ram’s problem. This request lead to other requests which lead to more requests until flowers reached Grandfather.

Next principle of OOP is transmitting of a message to an agent(an object) responsible for action

A way of viewing the world:

- Ram requests for action and he only wants desired outcome that is flowers sent to GrandFather.
- FlowersNepal can choose any technique to achieve desired objective and it will not get any interference from Ram.
- By discussing the problem in terms of responsibilities, we increase the level of abstraction. This permits greater independence between objects which is important to solve complex problems

Class hierarchy



Alan Kay's Definition of Object Oriented Programming

1. Everything is an object.
2. Computation is performed by objects communicating with each other through message passing.
3. Each object has its own memory.
4. Every object is an instance of a class. A class simply represents a grouping of similar objects.
5. The class is the repository for behavior associated with an object. That is, all the objects that are instances of same class can perform the same actions.
6. Classes are organized into a singly rooted tree structure called inheritance hierarchy.

Computation as Simulation

- In traditional model, computer is a data manager following some pattern of instructions, wandering thru memory, pulling values of various memory address, transforming them in some manner and pushing the results back into other memory slots
- Values in the memory describes the state of the machine or results of computation
- This model doesn’t help us in understanding how to solve problems using computer

Computation as Simulation

- In object oriented model:
- We do not mention memory addresses, variables, assignments
- We mention objects, messages, responsibility for some action
- Programmer creates community of interacting objects that help programmer to solve the problem
- This view of programming is similar to style of computer simulation called Discrete Event Driven Simulation, where the user creates computer models of the various elements of simulation, describes how they will interact with one another and sets them moving
- This is almost identical to Object Oriented Programming in which the user describes what the various entities in the program are, how will they interact with one another and finally set them in motion
- Thus OOP we have the view that computation is simulation

Coping with Complexity
The non-linear behavior of complexity

A task that would take one programmer two months to complete could not be accomplished by two programmers working for one month.

- The reason for this non-linear behavior is complexity which are
    1. the interconnection between software components are complicated
    2. a large amount of information has to be communicated among various members of the programming team

Abstraction

- Abstraction is used to control the complexity
- Abstraction means showing only essential attributes and hiding unnecessary information
- It is hiding unwanted details and showing only most essential information

Abstraction Mechanism

1. Procedure and function
2. Block Scope
3. Modules
4. Abstract Data Type
5. Object

Abstraction Mechanism

1. Function

-one of the first abstraction mechanism used in programming language

- function allows to write a set of code to perform a task and when ever you need to perform that task, we can simply call the function rather than writing same code multiple times

-Libraries of function gave the first possibility of information hiding where one programmer could write a set of procedures which could be used by many others.

-The other programmer would only know the necessary interface and not know the exact details of implementation.

- A function abstraction works by allowing the programmer to call a function written by the another programmer without necessarily understanding how it is implemented.

1. Procedure and function

```
int a,b;
void input()
        {
        ……
        }
 void add()
        {
        ……
        }
```

Abstraction Mechanism

2. Block Scope:

- Block scope determines the visibility or accessibility of the block
- When we use { }, it is a block. So, if we declare variables in the block scope, it means those variables exist only within the corresponding block

2. Block Scope:

Nesting of function

- In nesting of function, one function is nested within another function

```
void add(int x,int y)
        {
                …
        }
 void input()
        {
                int a,b;
                add(a,b);
        }
```

Abstraction Mechanism

3. Modules :

- Large programs can be broken into modules
- The programmers can think about the implementation of a piece of a program without full knowledge of the rest of the programs
- The rest of the programs needs to be understood only abstractly

3. Modules :

David Parnas' principle for modules are

i. The development module must be provided with all information needed to complete the module and nothing more.

ii. The developer of a module must provide the intended user with all the information to use the module correctly and nothing more.

Abstraction Mechanism

4. Abstract Data Type:

- Abstract data type is user defined data type or class whose properties is defined by set of values and set of operations
- ADT helps us to group data members and member functions
- It can decide which data members will be visible to which parts of programs
- It prevents user from directly accessing the private members

Abstract data type

5. Object

- Objects provide abstraction by hiding internal implementation details
- We just need to know which methods of the object are available to call and which input parameters we need to pass do specific operation
- We don’t need to understand how the method is implemented and what kinds of operations the method has to perform to create the expected result
- It has message passing feature

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 and 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

- 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

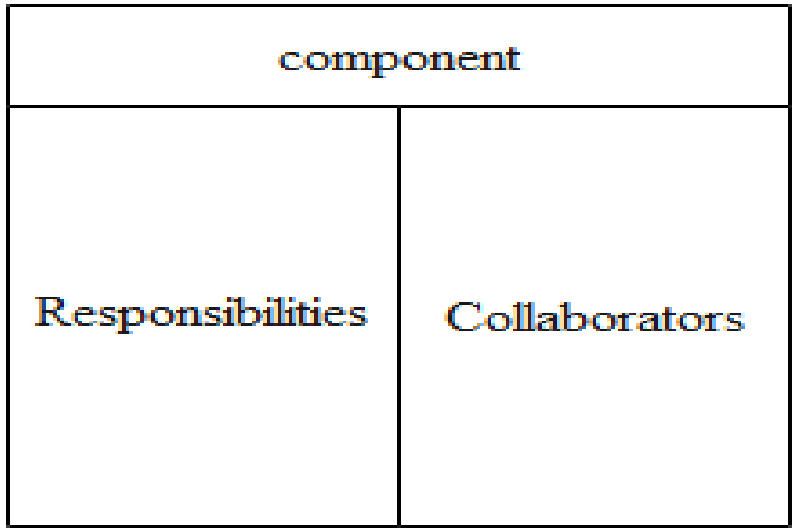
➢ List collaboration

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

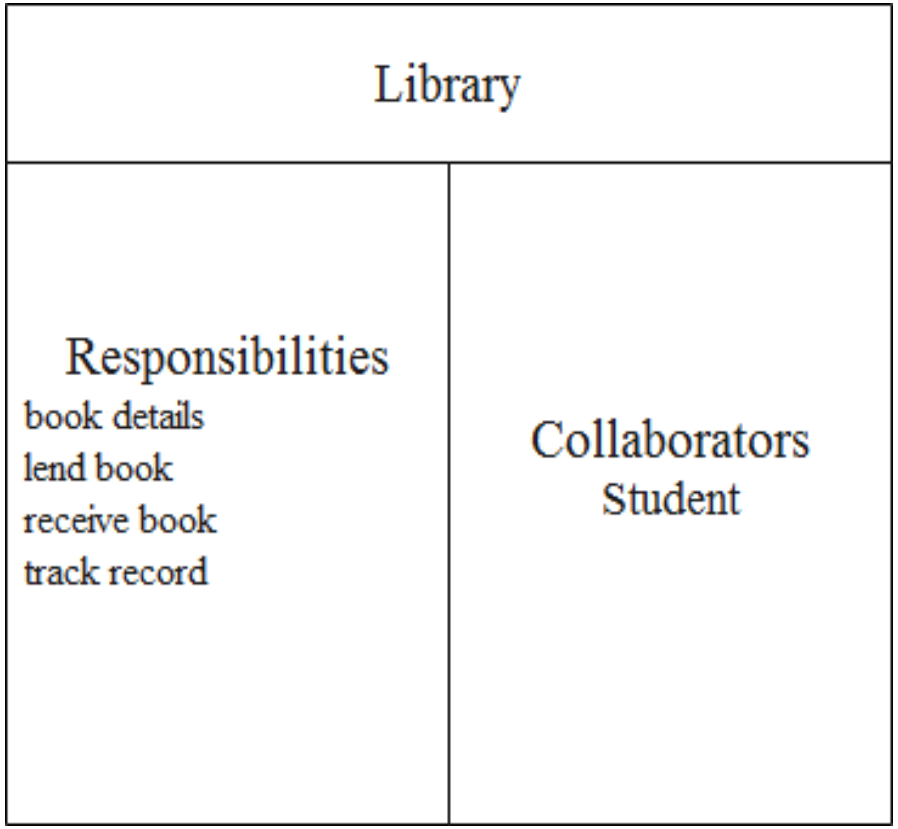
CRC card:



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.

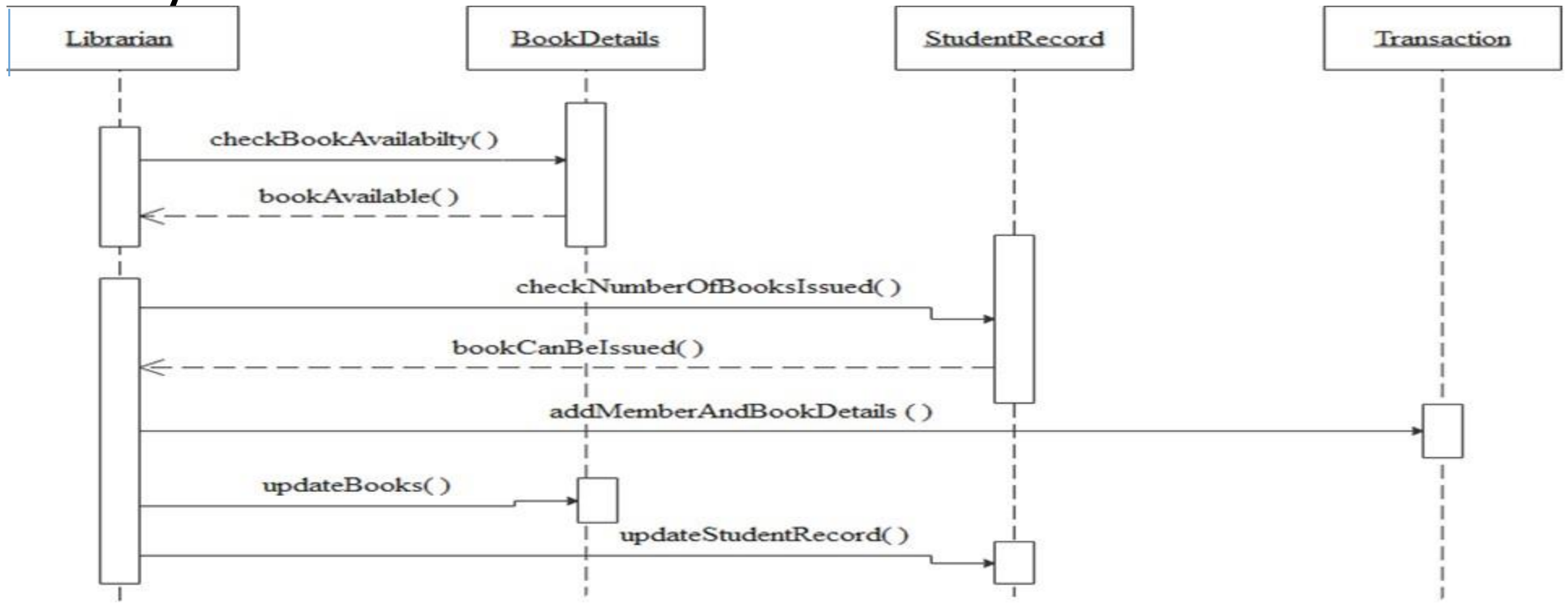
Sequence diagram:

- 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

Sequence diagram of borrowing book from library:



End of Chapter 1