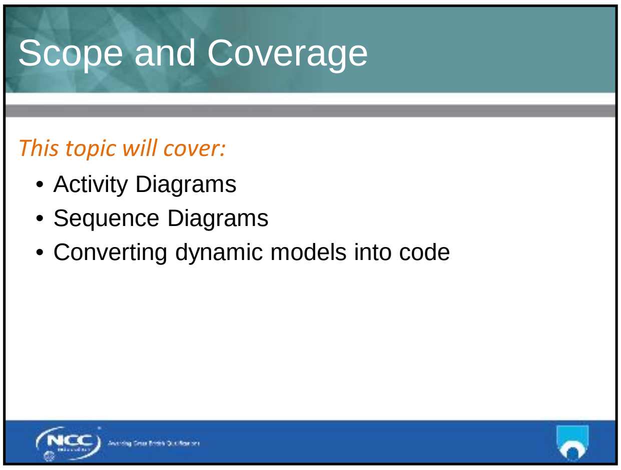
Analysis, Design and Implementation

Topic 6:

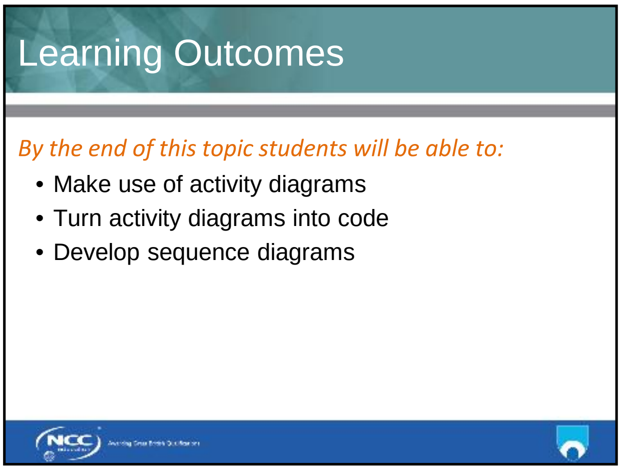
Dynamic Analysis and Design



Scope and Coverage

This topic will cover:

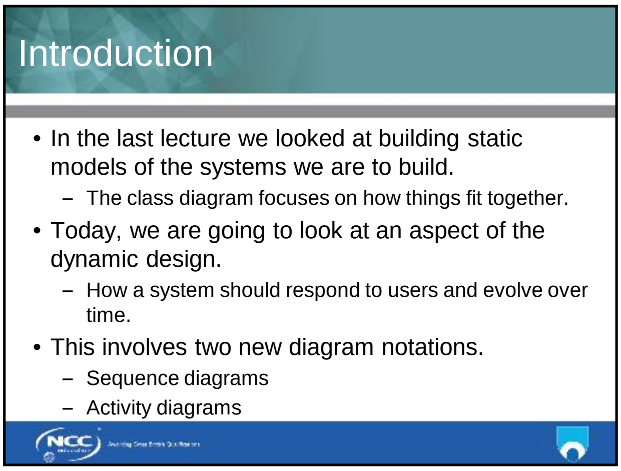
- Activity Diagrams
- Sequence Diagrams
- Converting dynamic models into code



Learning Outcomes

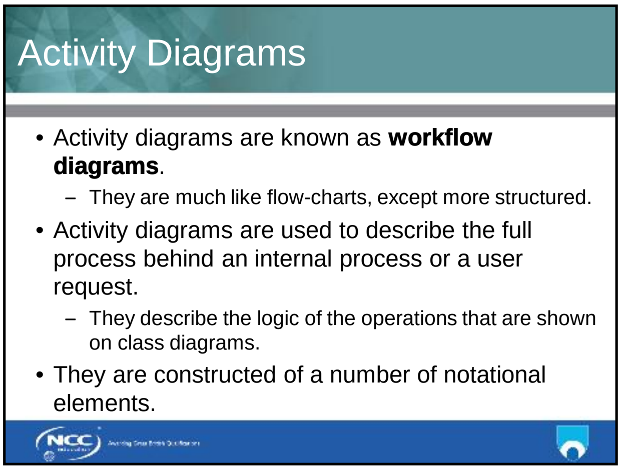
By the end of this topic students will be able to:

- Make use of activity diagrams
- Turn activity diagrams into code
- Develop sequence diagrams



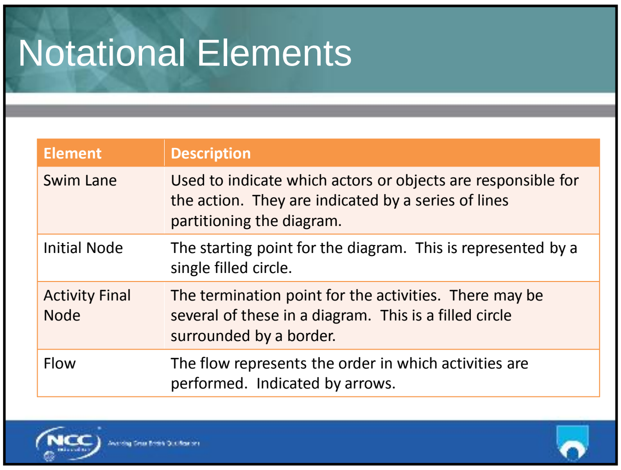
Introduction

- In the last lecture we looked at building static models of the systems we are to build.
  - The class diagram focuses on how things fit together.
- Today, we are going to look at an aspect of the dynamic design.
  - How a system should respond to users and evolve over time.
- This involves two new diagram notations.
  - Sequence diagrams
  - Activity diagrams



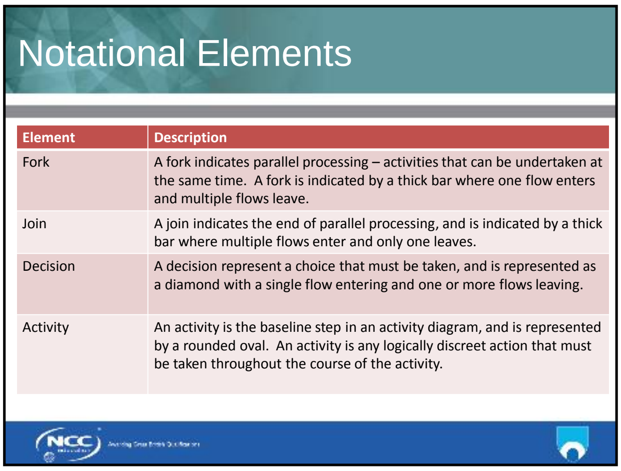
Activity Diagrams

- Activity diagrams are known as **workflow diagrams**.
  - They are much like flow-charts, except more structured.
- Activity diagrams are used to describe the full process behind an internal process or a user request.
  - They describe the logic of the operations that are shown on class diagrams.
- They are constructed of a number of notational elements.



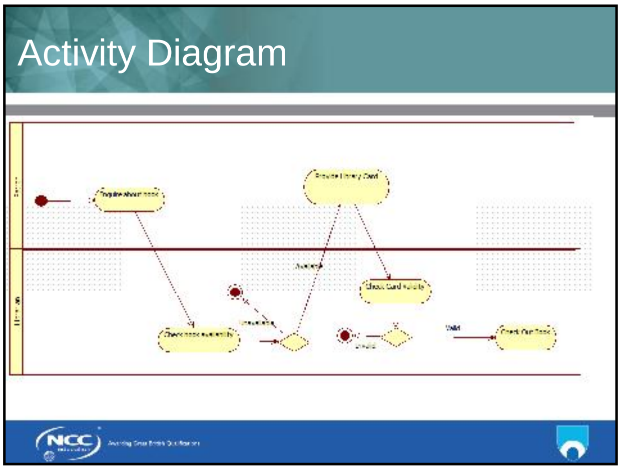
Notational Elements

Element	Description
Swim Lane	Used to indicate which actors or objects are responsible for the action. They are indicated by a series of lines partitioning the diagram.
Initial Node	The starting point for the diagram. This is represented by a single filled circle.
Activity Final Node	The termination point for the activities. There may be several of these in a diagram. This is a filled circle surrounded by a border.
Flow	The flow represents the order in which activities are performed. Indicated by arrows.

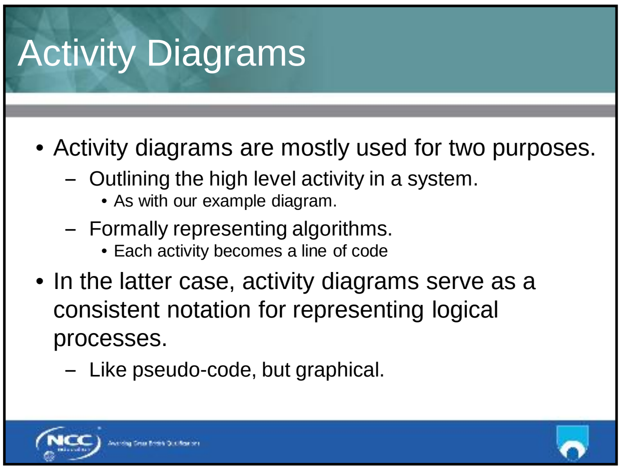


Notational Elements

Element	Description
Fork	A fork indicates parallel processing – activities that can be undertaken at the same time. A fork is indicated by a thick bar where one flow enters and multiple flows leave.
Join	A join indicates the end of parallel processing, and is indicated by a thick bar where multiple flows enter and only one leaves.
Decision	A decision represent a choice that must be taken, and is represented as a diamond with a single flow entering and one or more flows leaving.
Activity	An activity is the baseline step in an activity diagram, and is represented by a rounded oval. An activity is any logically discreet action that must be taken throughout the course of the activity.

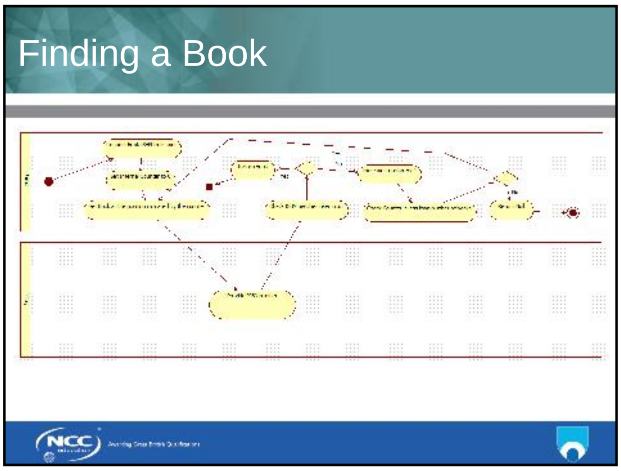


Activity Diagram

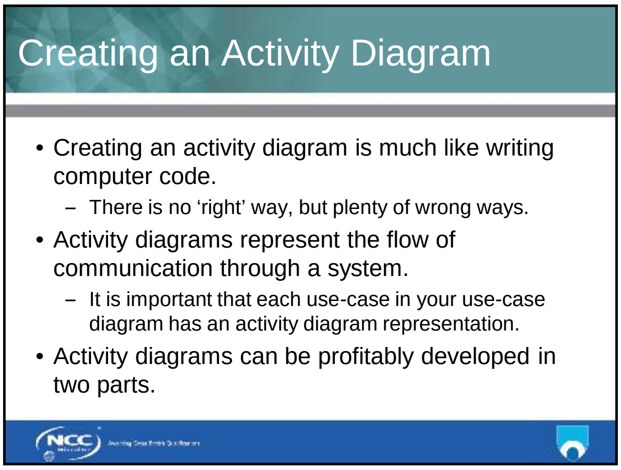


Activity Diagrams

- Activity diagrams are mostly used for two purposes.
  - Outlining the high level activity in a system.
    - As with our example diagram.
  - Formally representing algorithms.
    - Each activity becomes a line of code
- In the latter case, activity diagrams serve as a consistent notation for representing logical processes.
  - Like pseudo-code, but graphical.

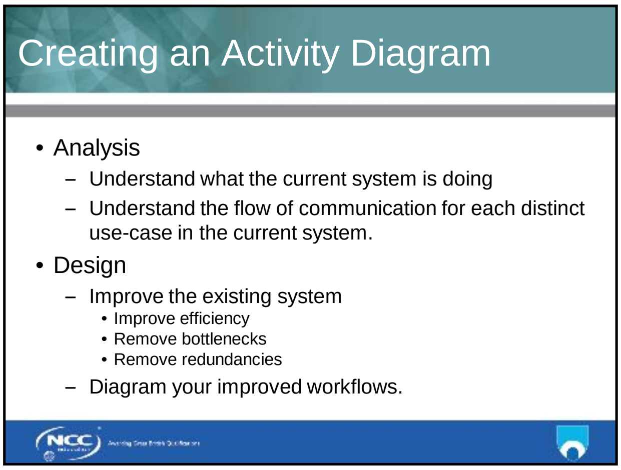


Finding a Book



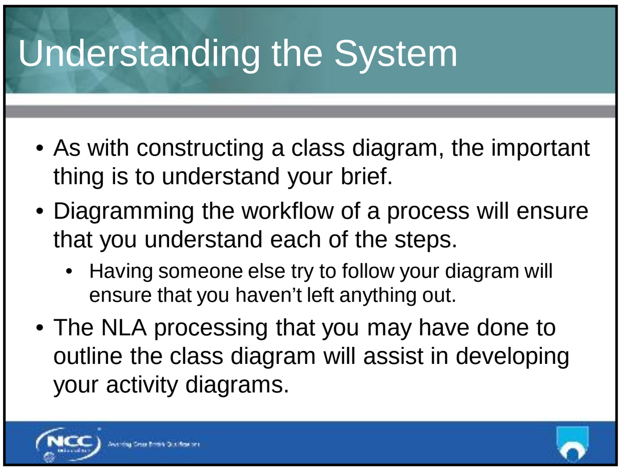
Creating an Activity Diagram

- Creating an activity diagram is much like writing computer code.
  - There is no ‘right’ way, but plenty of wrong ways.
- Activity diagrams represent the flow of communication through a system.
  - It is important that each use-case in your use-case diagram has an activity diagram representation.
- Activity diagrams can be profitably developed in two parts.



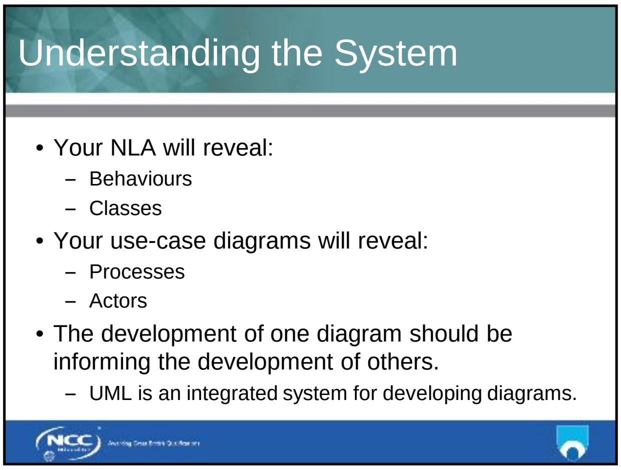
Creating an Activity Diagram

- Analysis
  - Understand what the current system is doing
  - Understand the flow of communication for each distinct use-case in the current system.
- Design
  - Improve the existing system
    - Improve efficiency
    - Remove bottlenecks
    - Remove redundancies
  - Diagram your improved workflows.



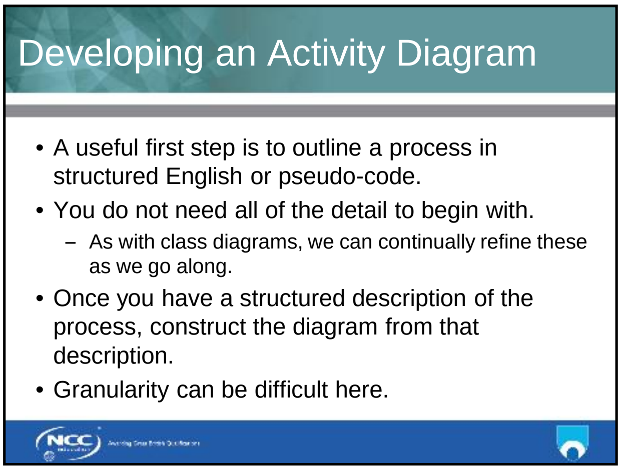
Understanding the System

- As with constructing a class diagram, the important thing is to understand your brief.
- Diagramming the workflow of a process will ensure that you understand each of the steps.
  - Having someone else try to follow your diagram will ensure that you haven't left anything out.
- The NLA processing that you may have done to outline the class diagram will assist in developing your activity diagrams.



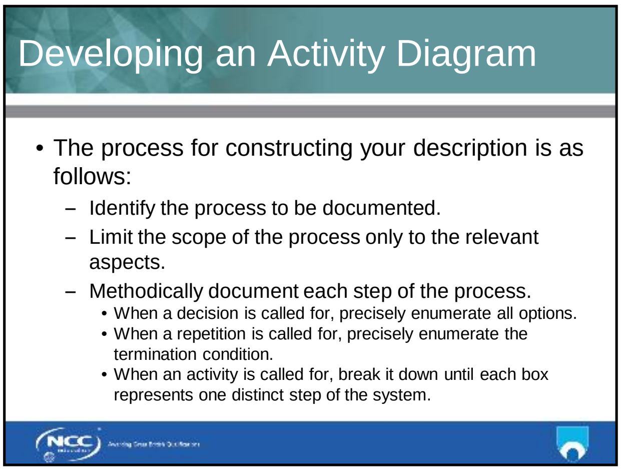
Understanding the System

- Your NLA will reveal:
  - Behaviours
  - Classes
- Your use-case diagrams will reveal:
  - Processes
  - Actors
- The development of one diagram should be informing the development of others.
  - UML is an integrated system for developing diagrams.



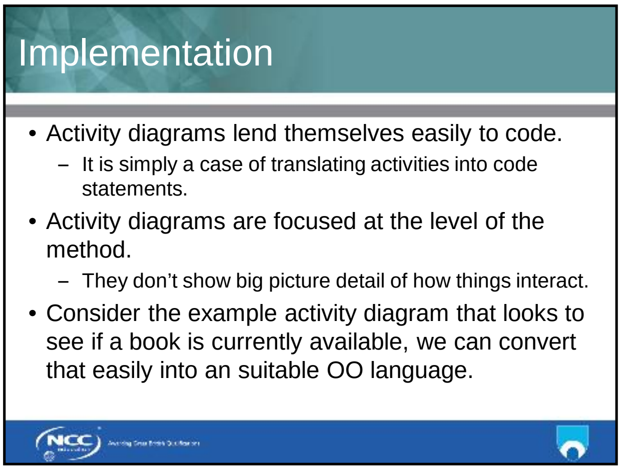
Developing an Activity Diagram

- A useful first step is to outline a process in structured English or pseudo-code.
- You do not need all of the detail to begin with.
  - As with class diagrams, we can continually refine these as we go along.
- Once you have a structured description of the process, construct the diagram from that description.
- Granularity can be difficult here.



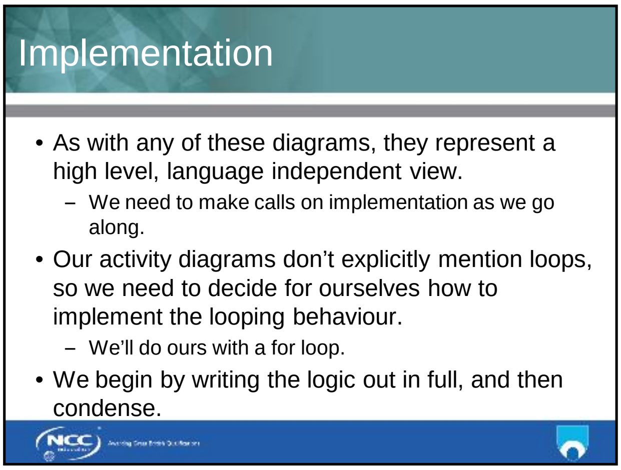
Developing an Activity Diagram

- The process for constructing your description is as follows:
  - Identify the process to be documented.
  - Limit the scope of the process only to the relevant aspects.
  - Methodically document each step of the process.
    - When a decision is called for, precisely enumerate all options.
    - When a repetition is called for, precisely enumerate the termination condition.
    - When an activity is called for, break it down until each box represents one distinct step of the system.



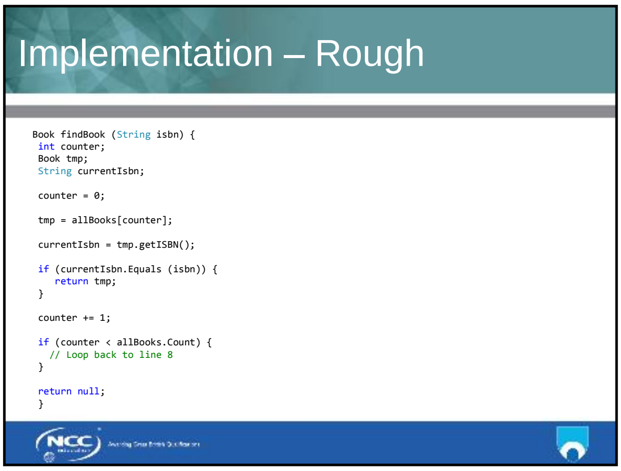
Implementation

- Activity diagrams lend themselves easily to code.
  - It is simply a case of translating activities into code statements.
- Activity diagrams are focused at the level of the method.
  - They don’t show big picture detail of how things interact.
- Consider the example activity diagram that looks to see if a book is currently available, we can convert that easily into an suitable OO language.



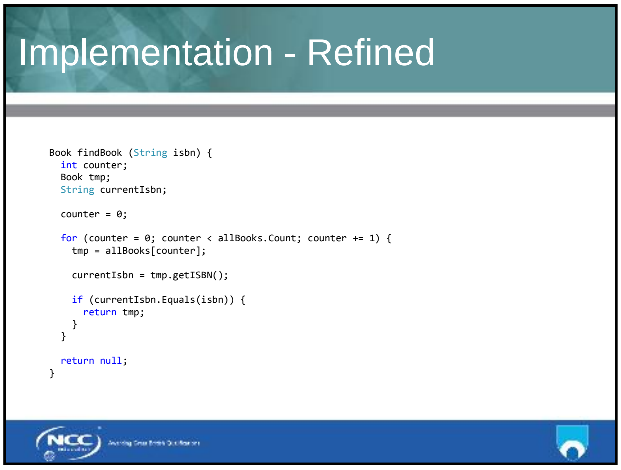
Implementation

- As with any of these diagrams, they represent a high level, language independent view.
  - We need to make calls on implementation as we go along.
- Our activity diagrams don’t explicitly mention loops, so we need to decide for ourselves how to implement the looping behaviour.
  - We’ll do ours with a for loop.
- We begin by writing the logic out in full, and then condense.



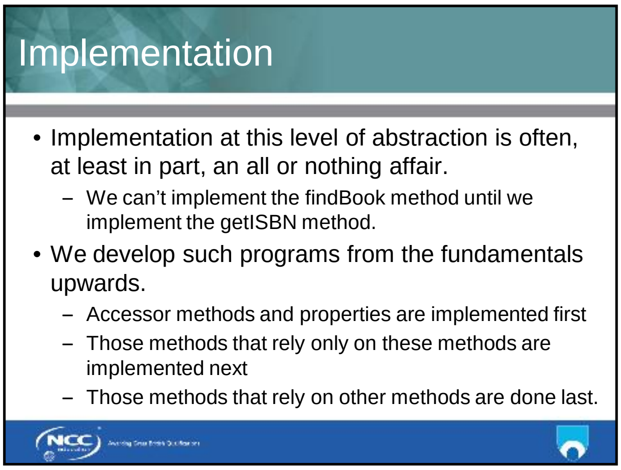
Implementation – Rough

```
Book findBook (String isbn) {
 int counter;
 Book tmp;
 String currentIsbn;

 counter = 0;

 tmp = allBooks[counter];

 currentIsbn = tmp.getISBN();

 if (currentIsbn.Equals (isbn)) {
    return tmp;
 }

 counter += 1;

 if (counter < allBooks.Count) {
   // Loop back to line 8
 }

 return null;
 }
```



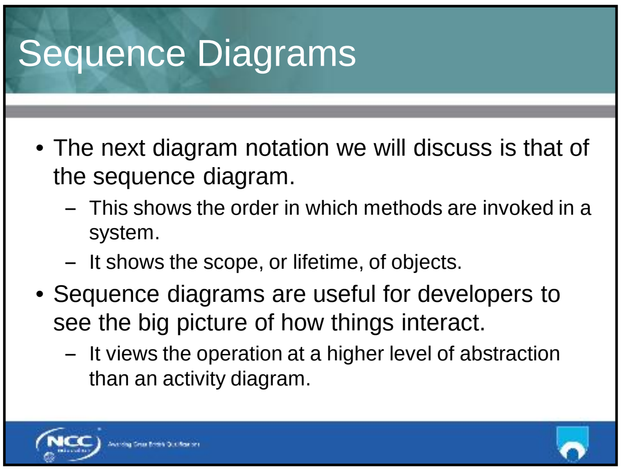
Implementation - Refined

```
Book findBook (String isbn) {
  int counter;
  Book tmp;
  String currentIsbn;

  counter = 0;

  for (counter = 0; counter < allBooks.Count; counter += 1) {
    tmp = allBooks[counter];

    currentIsbn = tmp.getISBN();

    if (currentIsbn.Equals(isbn)) {
      return tmp;
    }
  }

  return null;
}
```



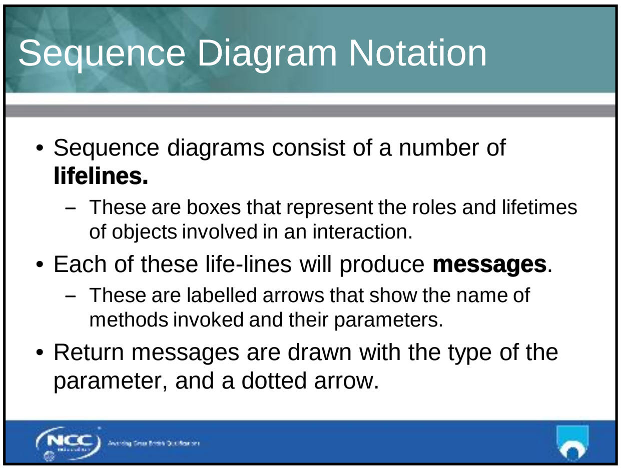
Implementation

- Implementation at this level of abstraction is often, at least in part, an all or nothing affair.
  - We can't implement the findBook method until we implement the getISBN method.
- We develop such programs from the fundamentals upwards.
  - Accessor methods and properties are implemented first
  - Those methods that rely only on these methods are implemented next
  - Those methods that rely on other methods are done last.



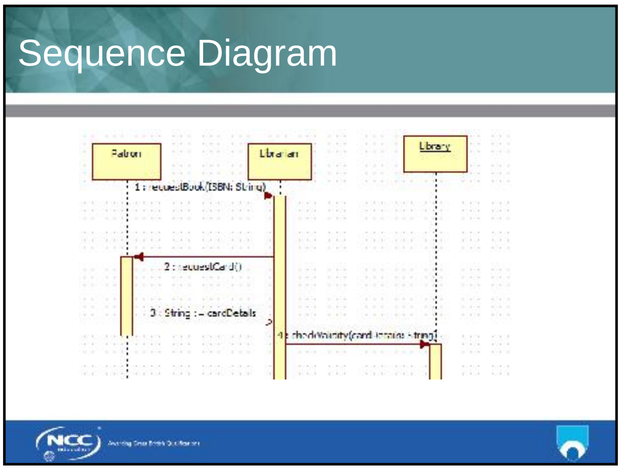
Sequence Diagrams

- The next diagram notation we will discuss is that of the sequence diagram.
  - This shows the order in which methods are invoked in a system.
  - It shows the scope, or lifetime, of objects.
- Sequence diagrams are useful for developers to see the big picture of how things interact.
  - It views the operation at a higher level of abstraction than an activity diagram.



Sequence Diagram Notation

- Sequence diagrams consist of a number of **lifelines.**
  - These are boxes that represent the roles and lifetimes of objects involved in an interaction.
- Each of these life-lines will produce **messages**.
  - These are labelled arrows that show the name of methods invoked and their parameters.
- Return messages are drawn with the type of the parameter, and a dotted arrow.



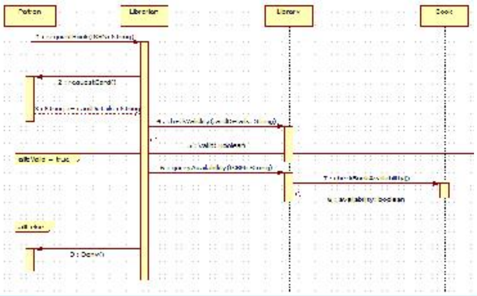
Sequence Diagram

Patron
Librarian
Library
1 : requestBook(ISBN: String)
2 : requestCard()
3 : String :– cardDetails
4 : checkValidity(cardDetails : String)

Guards and Alternates

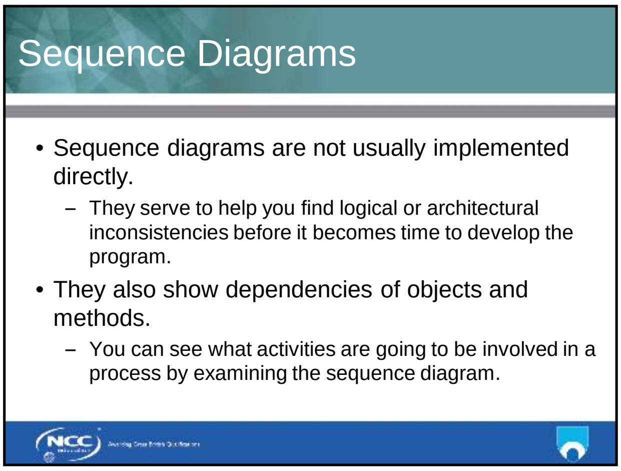
- The flow of logic through a sequence diagram is often dependent on the state of returned values.
  - Card validity
  - Book availability
- We represent these in a sequence diagram through the use of a **frame.**
  - This allows us to provide if/else strutures in our diagrams.
  - We place a **guard** condition on the frame which determines whether a frame should be executed.,

Guards and Alternates



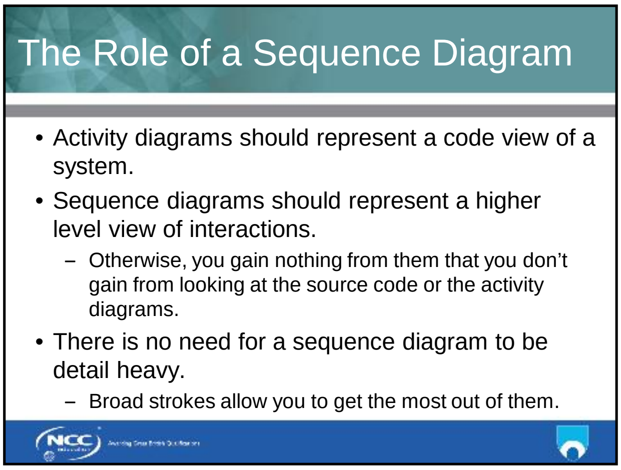
Objects and Classes

- In a sequence diagram, the boxes at the top of a lifeline represent objects.
  - Not classes.
- As such, they should properly be named and typed.
  - Names are of secondary importance unless we can be sure of a particular context.
- We name them anyway so that we can distinguish between instances of a class and potentially static operations (in which case, we have the type only).



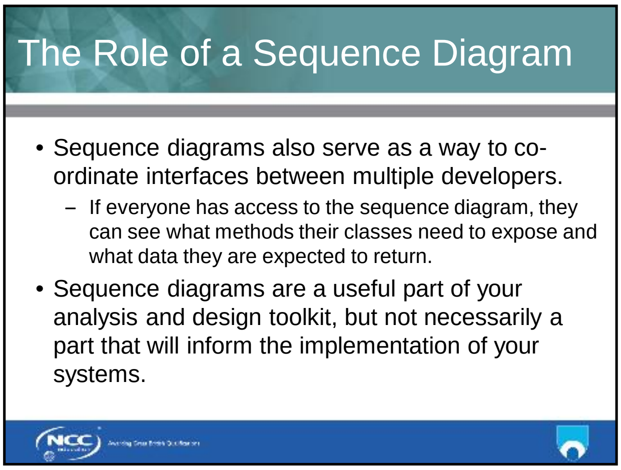
Sequence Diagrams

- Sequence diagrams are not usually implemented directly.
  - They serve to help you find logical or architectural inconsistencies before it becomes time to develop the program.
- They also show dependencies of objects and methods.
  - You can see what activities are going to be involved in a process by examining the sequence diagram.



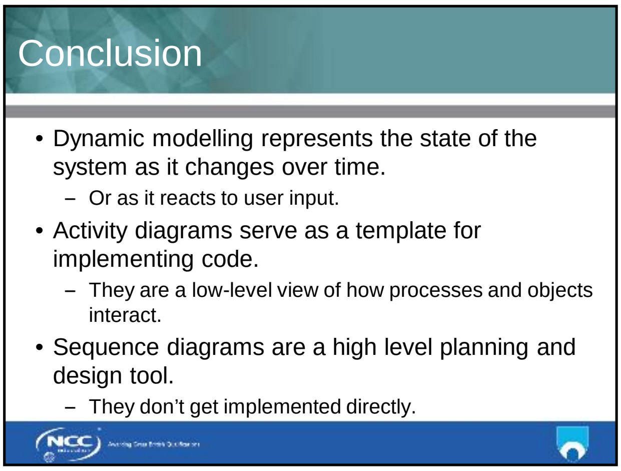
The Role of a Sequence Diagram

- Activity diagrams should represent a code view of a system.
- Sequence diagrams should represent a higher level view of interactions.
  - Otherwise, you gain nothing from them that you don't gain from looking at the source code or the activity diagrams.
- There is no need for a sequence diagram to be detail heavy.
  - Broad strokes allow you to get the most out of them.



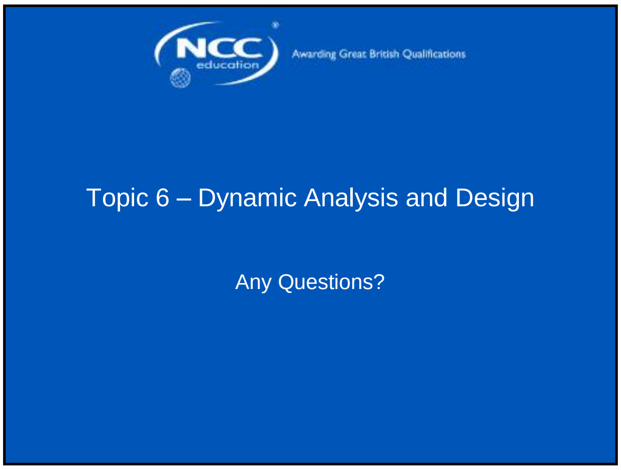
The Role of a Sequence Diagram

- Sequence diagrams also serve as a way to co-ordinate interfaces between multiple developers.
  - If everyone has access to the sequence diagram, they can see what methods their classes need to expose and what data they are expected to return.
- Sequence diagrams are a useful part of your analysis and design toolkit, but not necessarily a part that will inform the implementation of your systems.



Conclusion

- Dynamic modelling represents the state of the system as it changes over time.
  - Or as it reacts to user input.
- Activity diagrams serve as a template for implementing code.
  - They are a low-level view of how processes and objects interact.
- Sequence diagrams are a high level planning and design tool.
  - They don’t get implemented directly.



Awarding Great British Qualifications

Topic 6 – Dynamic Analysis and Design

Any Questions?