Revision:

Lecture 1 - 7

Topics

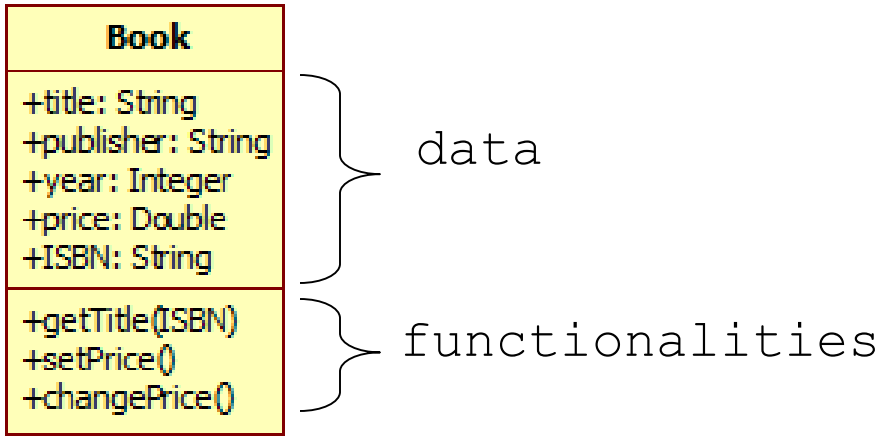
1. **Overview of Object-Oriented System Development**
   - Why Object-Oriented?
   - Object Oriented Methodology
   - Overview of the Unified Approach
   - OO Support tools
2. **Object-Oriented Concepts**
   - Object-Oriented Philosophy
   - Class, Object and Instance
   - Object Attributes
   - Object Behaviour and Methods
   - Encapsulation and Information Hiding
   - Class Hierarchy
   - Polymorphism
   - Object Relationships
3. **Object Oriented in C++**
   - Analysis of Class Concept
   - Class in C++
   - Member Functions
   - Constructor and Destructors
   - Member Function (Methods)
   - Access Control Rules
   - Friend Function
   - Derived Class in C++
   - Full Example Program:Constructor & Destructor
   - Analysis of Inheritance Concepts
   - Inheritance in C++
4. **Unified Modelling Language**
   - Static and Dynamic Models
   - The Unified Modeling Language
   - UML Diagrams
   - UML Class Diagram
   - Use-Case Diagram
   - UML Dynamic Modeling
   - Model Management
   - UML Extensibility
   - UML Meta-Model
5. **Unified Process (UP)**
   - Introduction to UP
   - UP is Use Case Driven
   - UP is Architecture Driven
   - Iterative and Incremental Development
6. **Software Life Cycle: Unified Process**
   - Inception
   - Elaboration
   - Construction
   - Transition
7. **Object-Oriented Analysis: Identifying Use Cases**
   - Analysis Problems
   - Understanding The Business Layer
   - Use-Case Driven Object Oriented Analysis
   - Business Process Modeling
   - Use-Case Model
   - Developing Effective Documentation

Introduction

- Object-Oriented (OO) systems development is a way to develop software by building self-contained modules that can be more easily:
  - Replaced
  - Modified
  - and Reused.

Object-Oriented Systems Development Methodology

- In an O-O environment, software is a collection of discrete objects.
- These objects encapsulate their data and functionalities to model real world "objects."

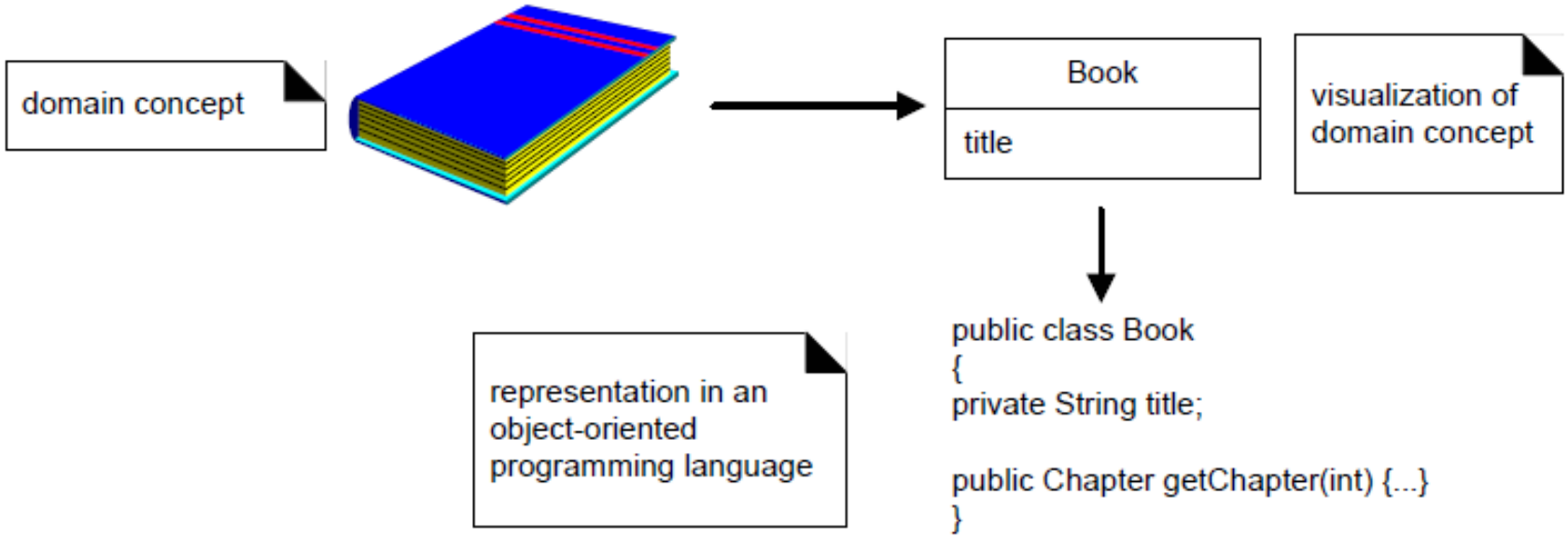


Benefits of Object Orientation

- Faster development,
- Reusability,
- Increased quality,
- Easier maintenance.

Examples

- Example class Car is known as Car in Analysis model, also in Design model, and Implementation model (e.g., car.java)



Concepts of object-orientation

i. Classes and objects
ii. Attributes
iii. Operations, methods and services
iv. Messages
v. Relationships : Inheritance, Association & Aggregation
vi. Encapsulation, and polymorphism.

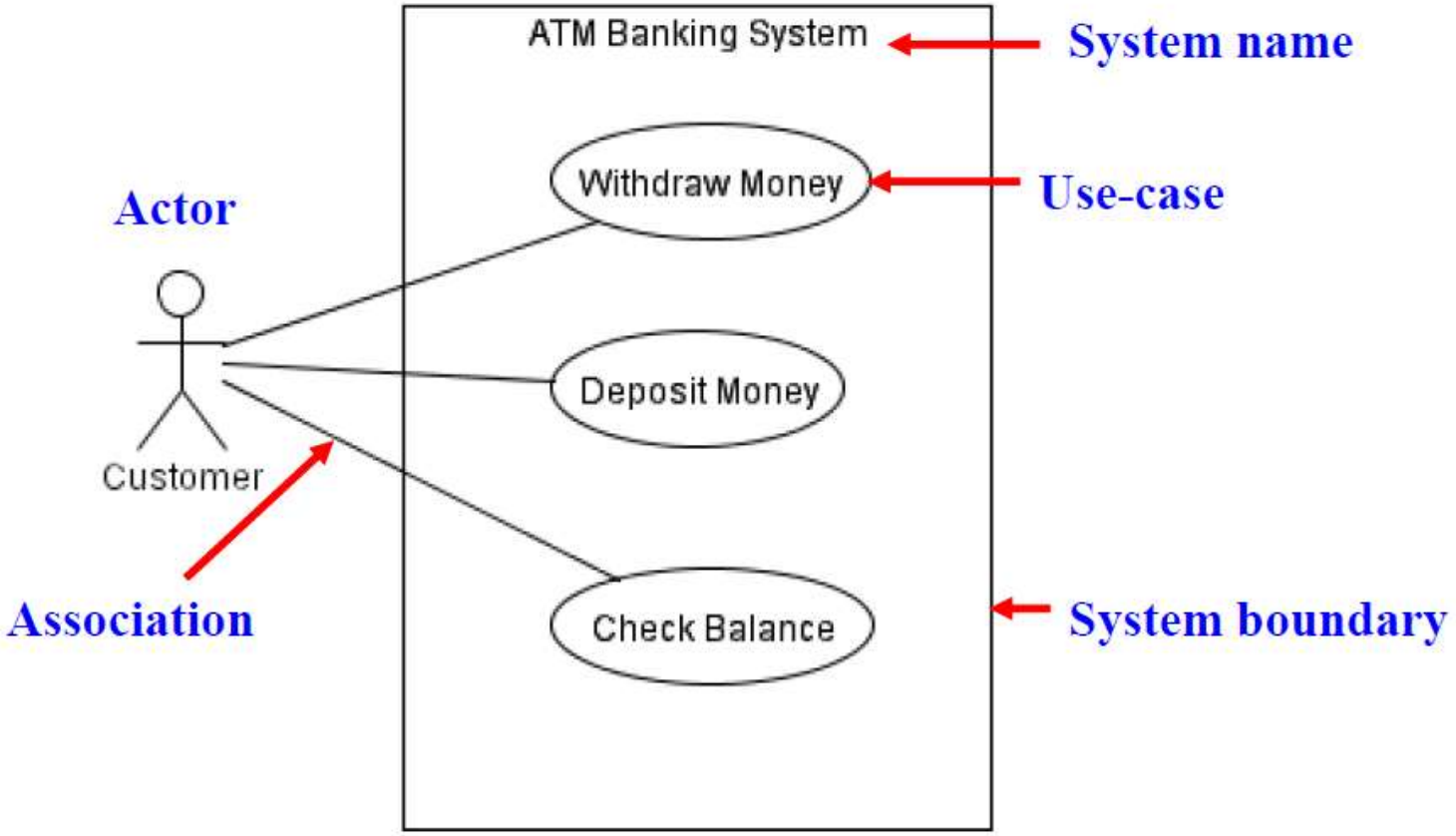
OO in C++

- Class in C++
- Member Functions
- Constructor and Destructors
- Access Control Rules
- Friend Function
- Inheritance in C++
- Polymorphism in C++

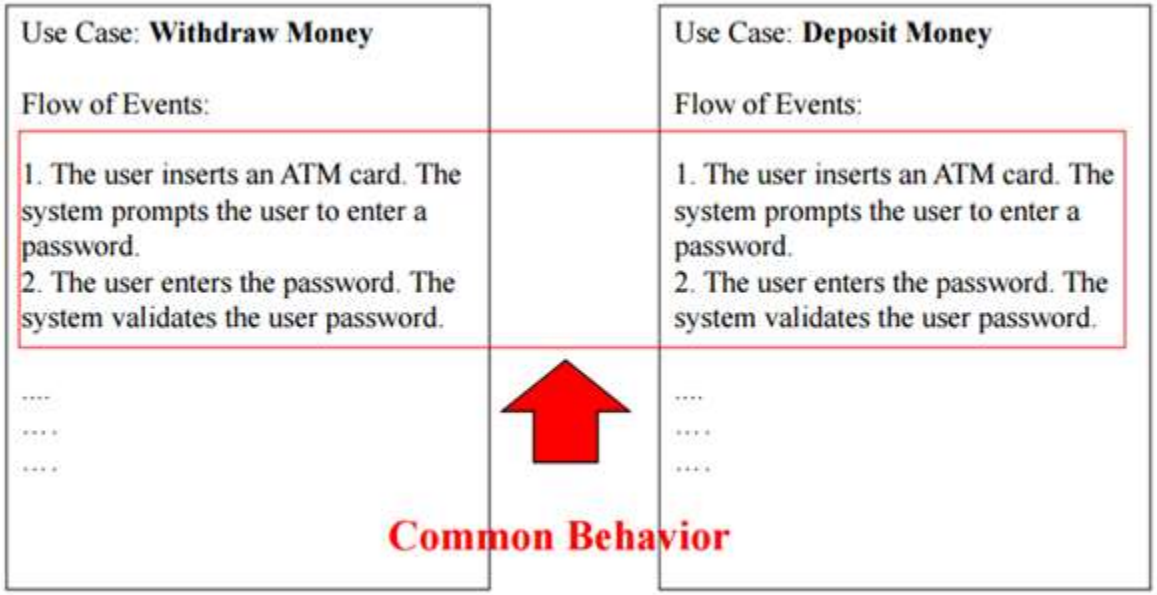
Unified modeling language

- Class diagram.
- Use case diagram.
- Interaction diagrams.
  - Sequence diagram.
  - Collaboration diagram.
- Statechart diagram.
- Activity diagram.
- Implementation diagrams.
  - Component diagram.
  - Deployment diagram.

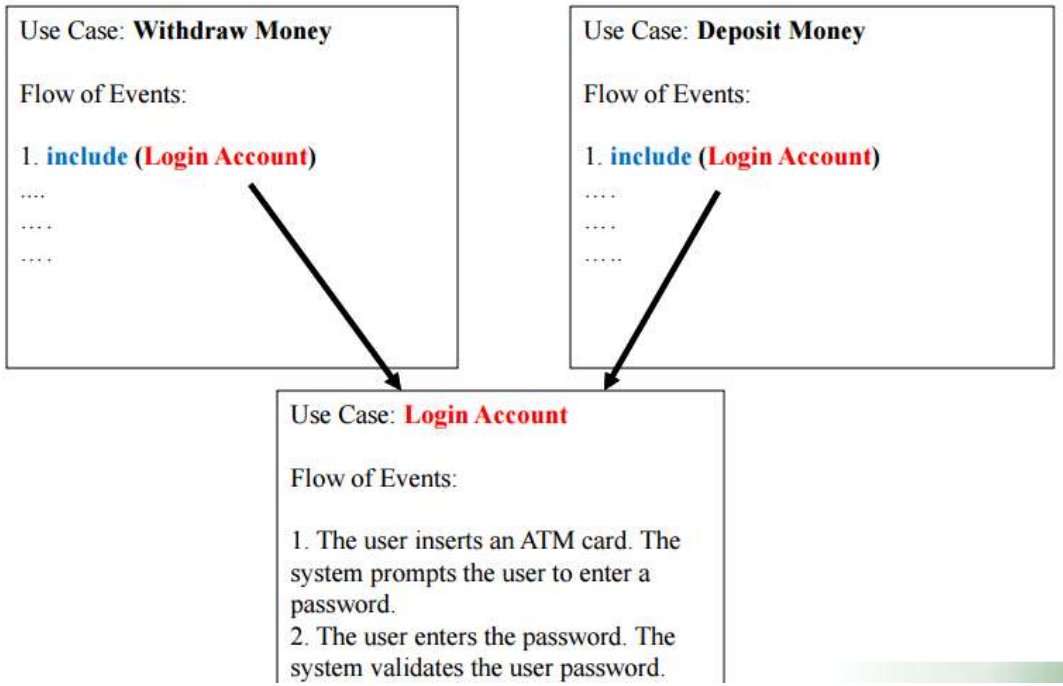
Use Case Diagram



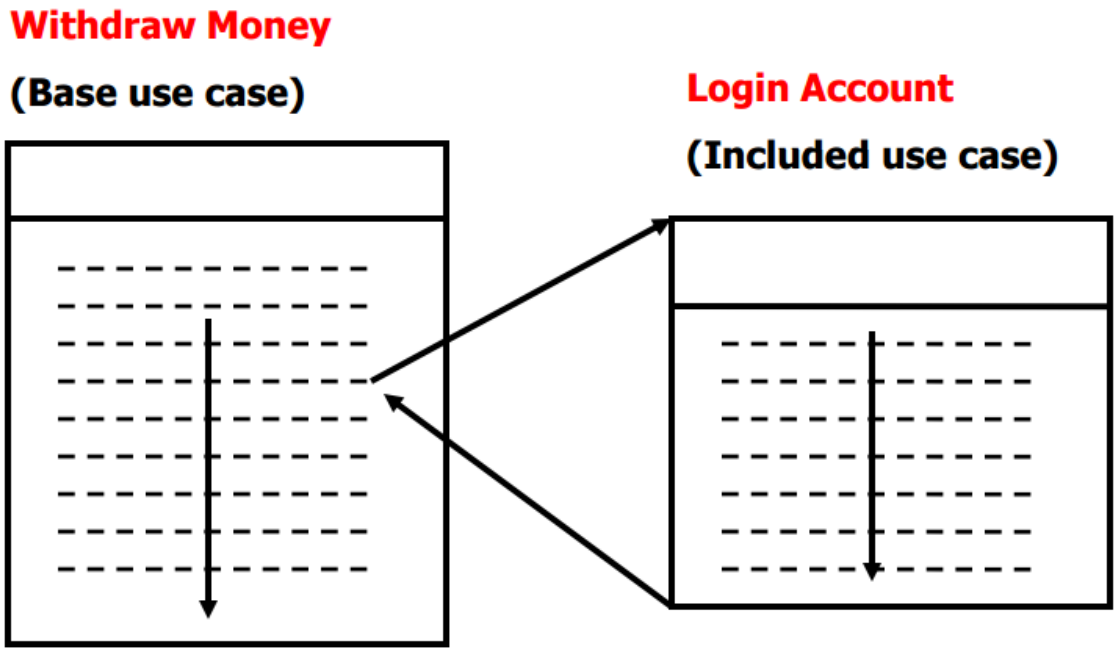
Include / Exclude Relationship ???



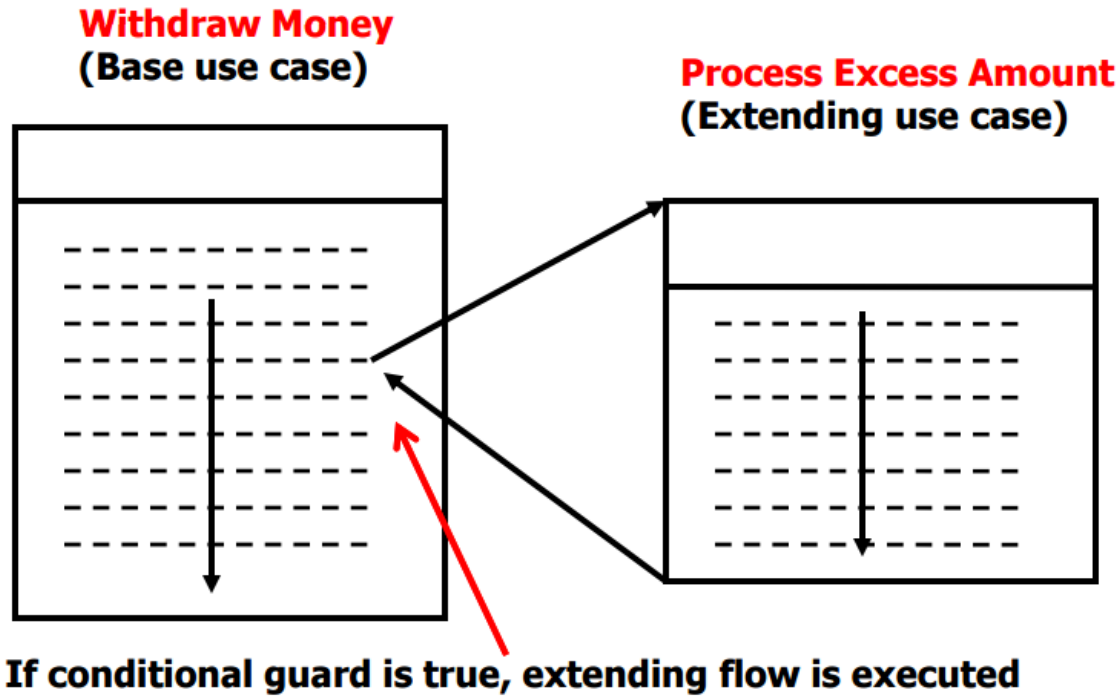
Structuring Use-cases with Relationships



The <<include>> Relationship



The <<extend>> Relationship



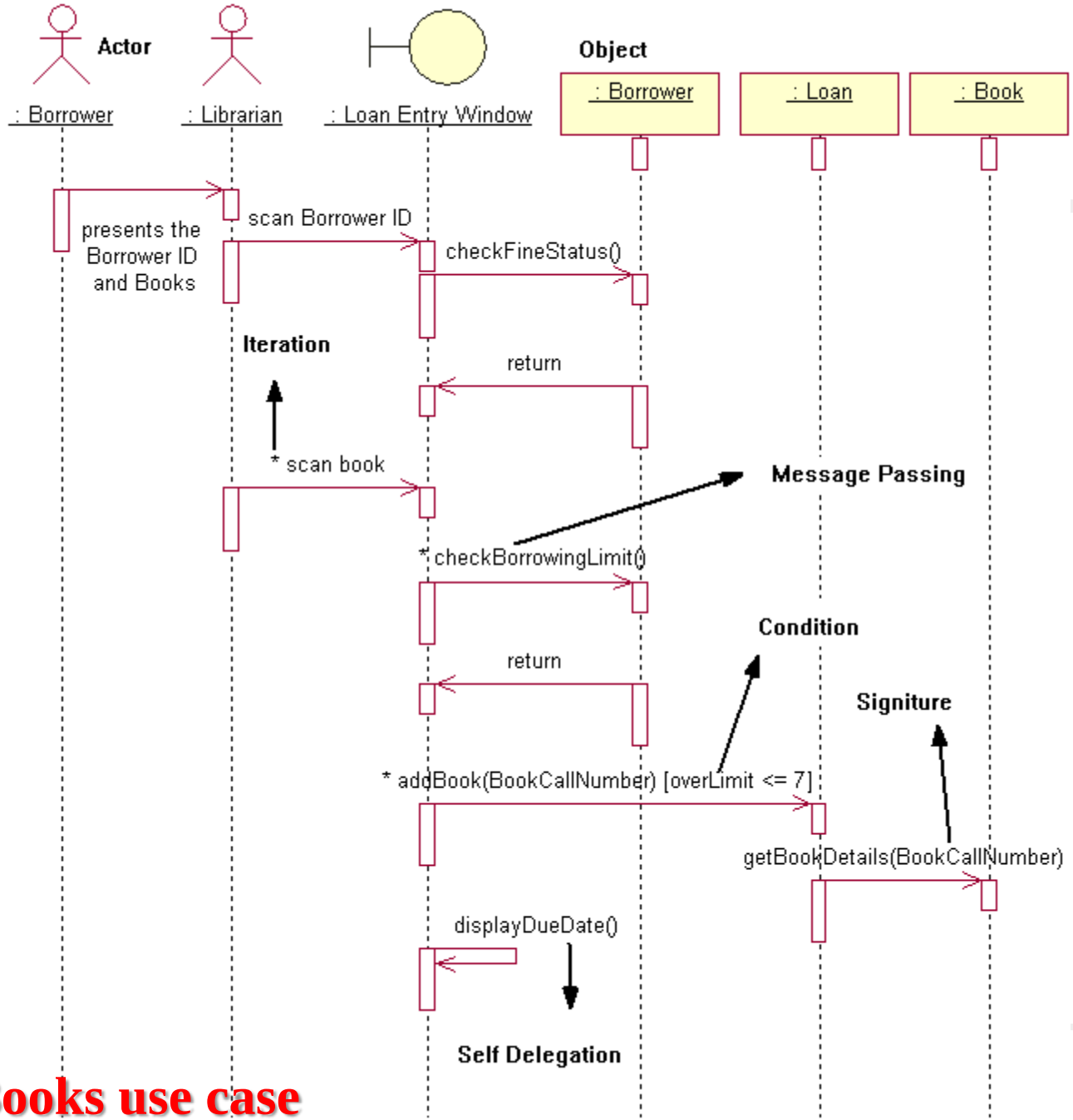
Sequence Diagrams

- **sequence diagram**: an "interaction diagram" that models a single scenario executing in the system

- relation of UML diagrams to other exercises:
  - CRC cards ➡ class diagram
  - use cases ➡ sequence diagrams

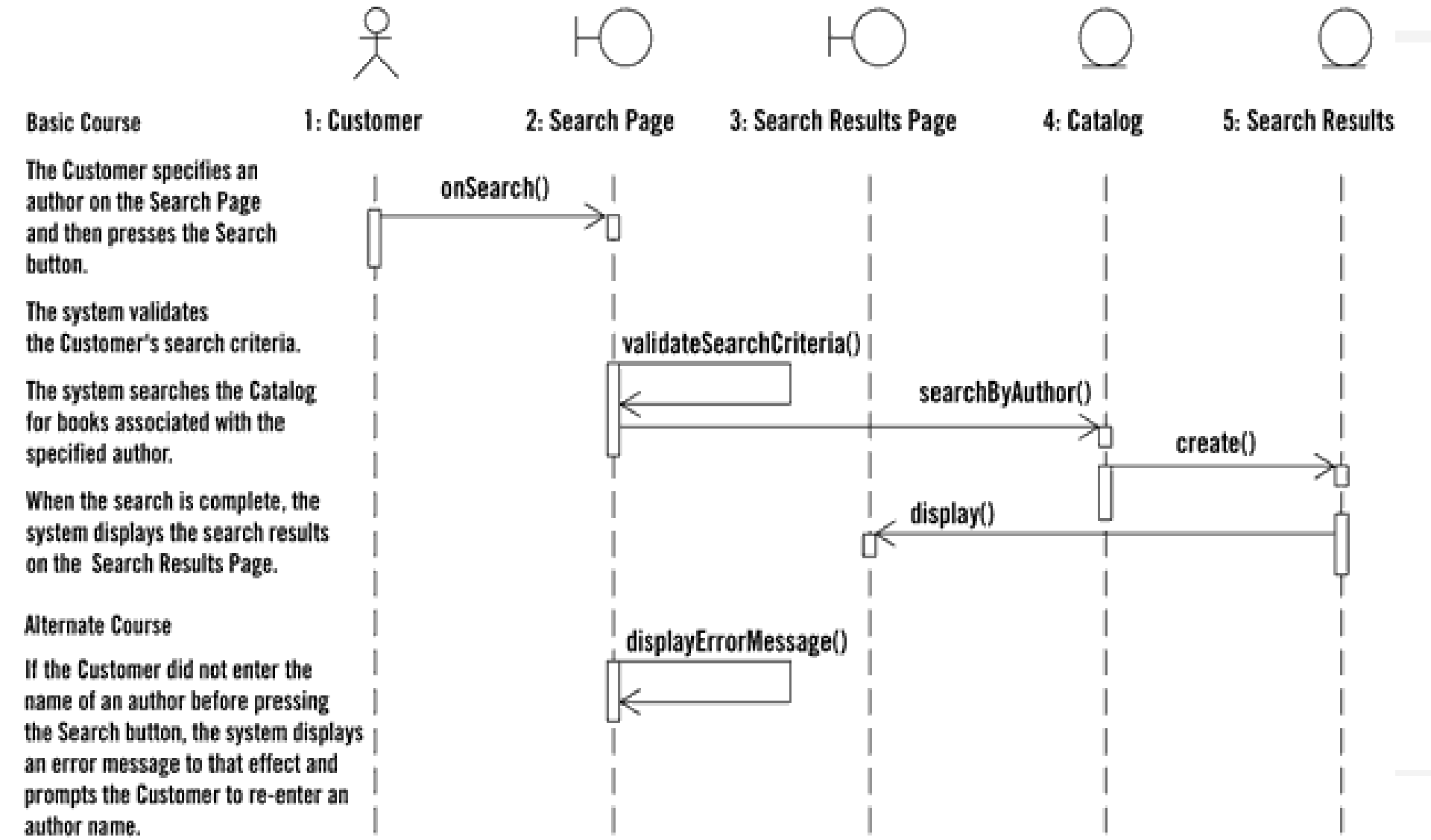
Key parts of a sequence diag.

- **participant**: an object or entity that acts in the sequence diagram
  - sequence diagram starts with an unattached "found message" arrow
- **message**: communication between participant objects
- the axes in a sequence diagram:
  - horizontal: which object/participant is acting
  - vertical: time (down -> forward in time)



Borrow Books use case

Sequence diagram from use case



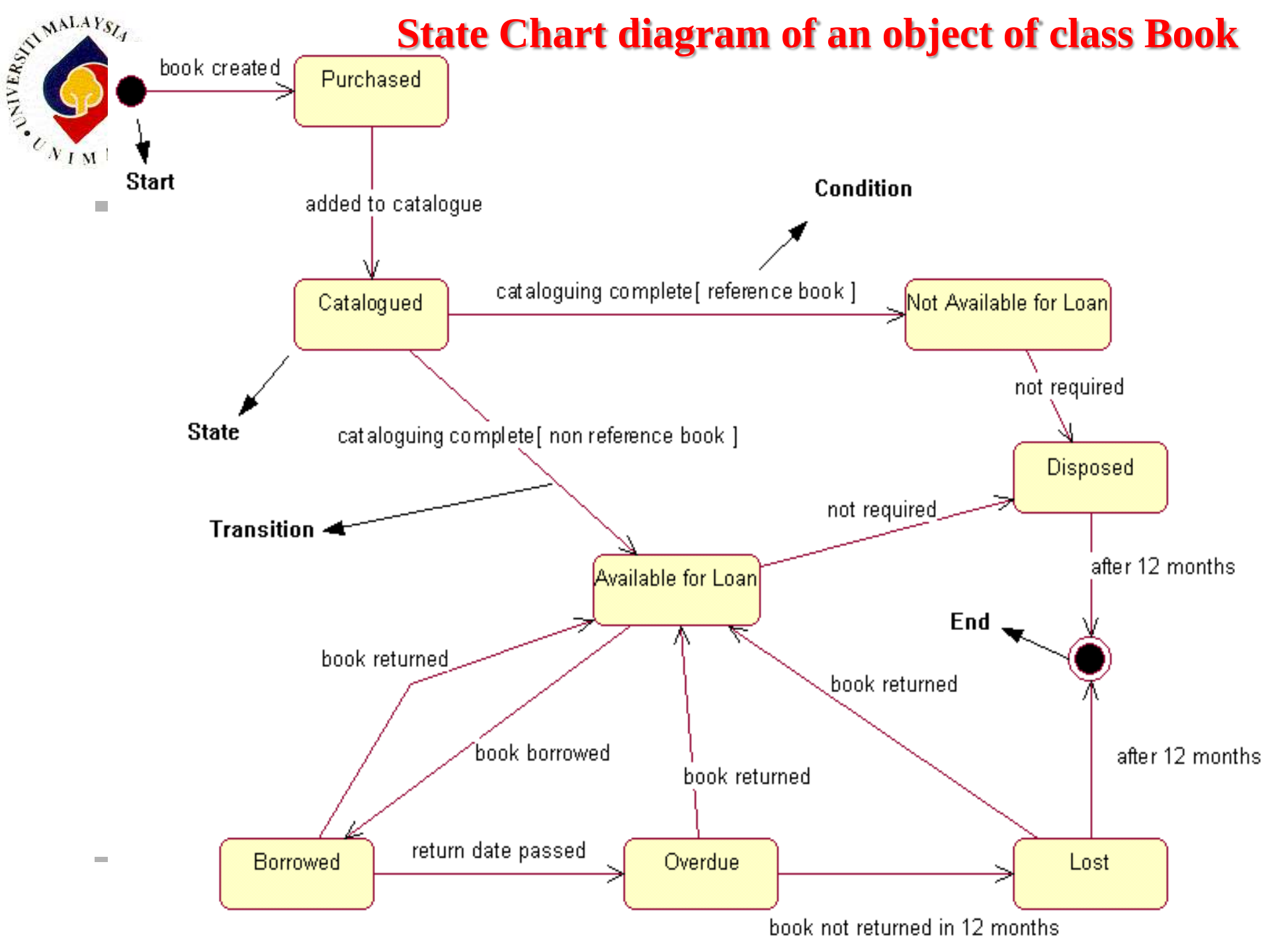
State Chart Diagram

- describe the **lifecycle of a given object of a class**.
- show all the possible **states** that the object can get into and the **transitions** which show how the object's state changes as a result of events that reach the object.
- useful to describe the behaviour **of an object** across several use cases.

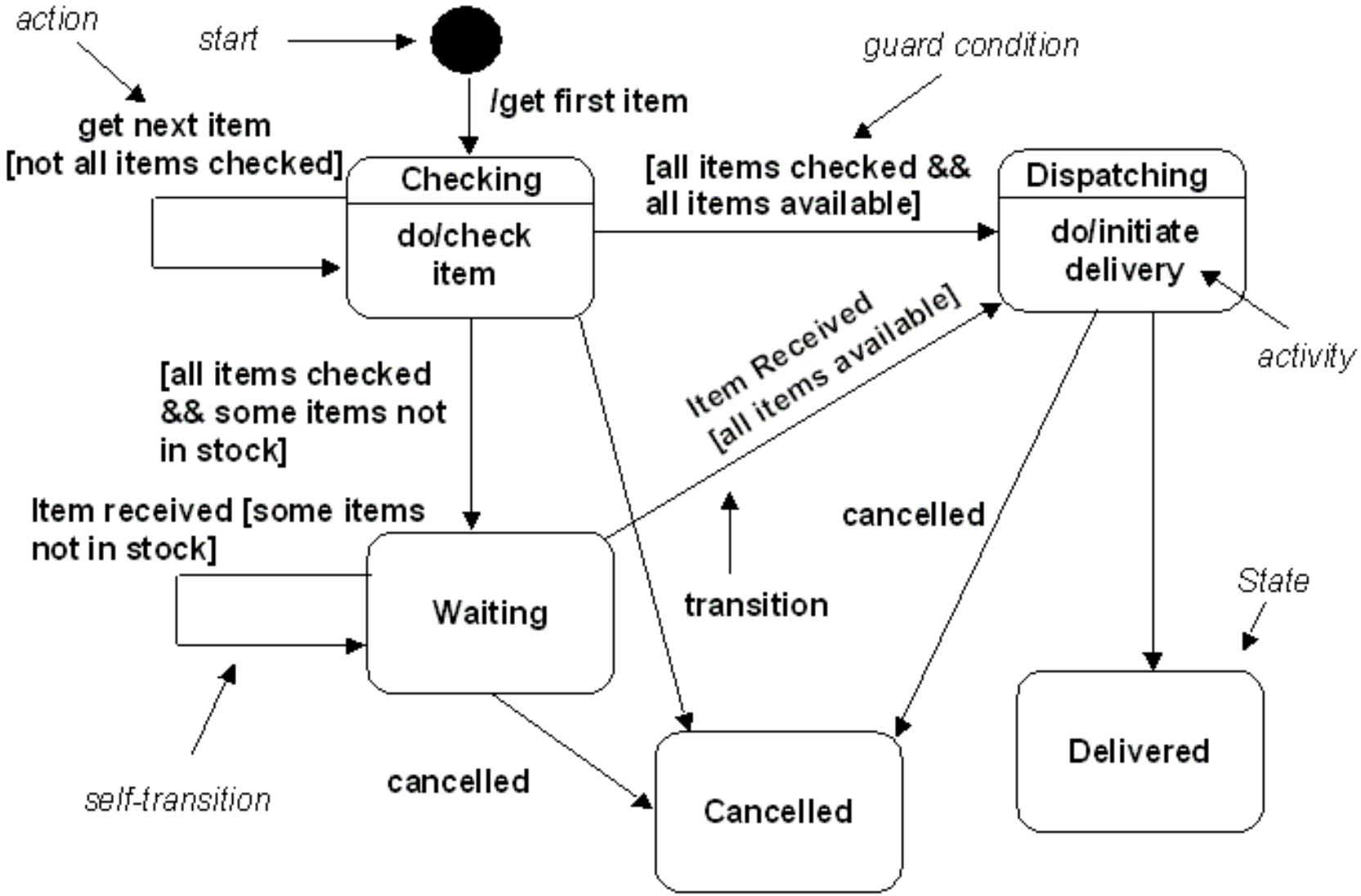
Scenario to generate State Chart

Librarians categorise the library books into loanable and non-loanable books. The non-loanable books are the reference books. However, the loanable books are the non-reference books. After cataloguing the books, the books are available for loan. Students who borrow the library books should return them back before the due date. Books that are 12 months over the due date would be considered as a lost state. However, if those books are found in the future, they must be returned back to the library. When the books are found not required in the library or have been damaged, the book would be disposed.

State Chart diagram of an object of class Book



Other example of State Chart



Classification: Approaches for Identifying Classes

- The noun phrase approach.
- The common class patterns approach.
- The use-case driven approach.
- The class responsibilities collaboration (CRC) approach.

Software Lifecycle phases

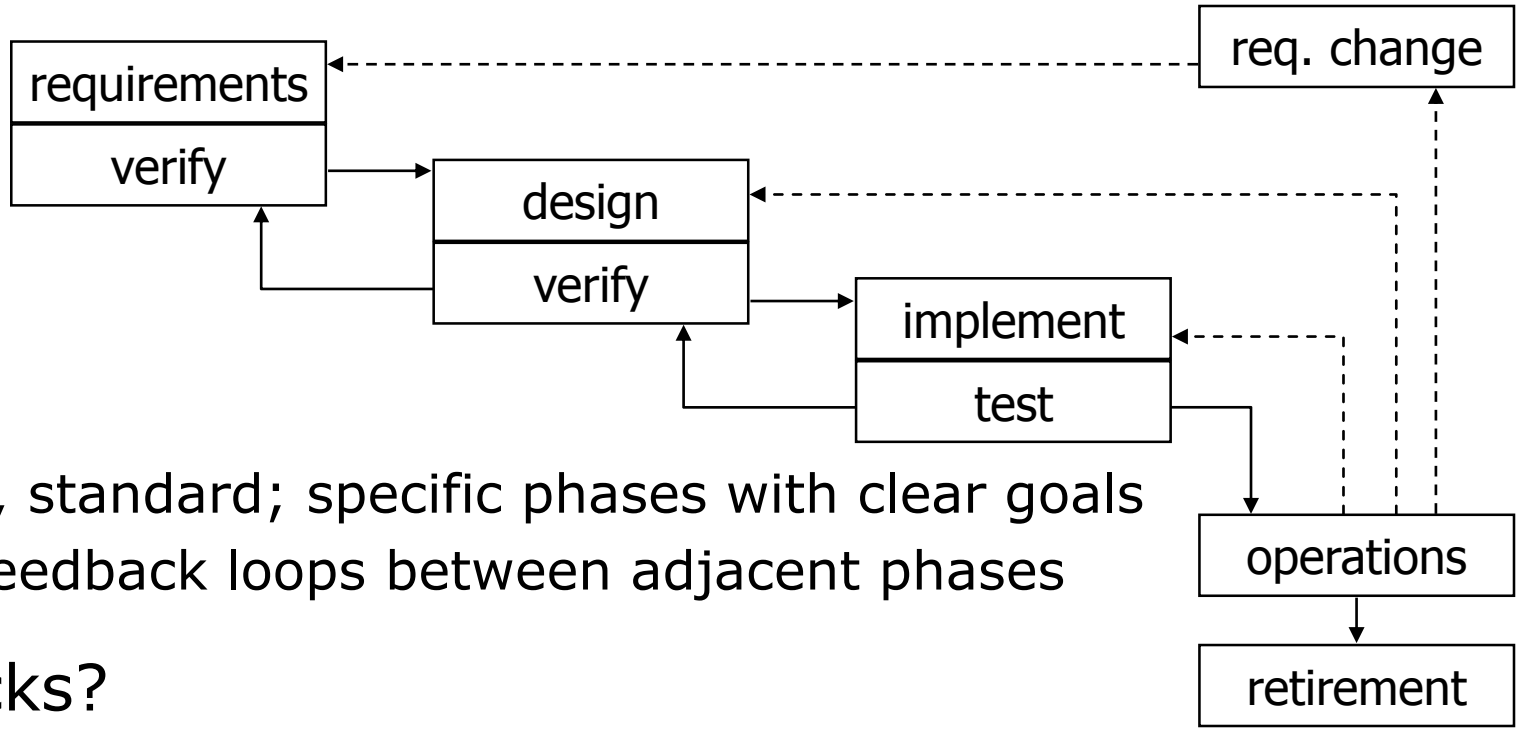
- **software lifecycle**: series of steps / phases
  - Requirements Analysis & Specification
  - High-level (Architectural) Design
  - Detailed (Object-oriented) Design
  - Implementation, Integration, Debugging
  - Testing, Profiling, Quality Assurance
  - Operation and Maintenance
  - other possibilities: Risk Assessment, Prototyping

  in each phase:
  - mark out a clear set of steps to perform
  - produce a tangible document or item
  - allow for review of work
  - specify actions to perform in the next phase

Life Cycle Modeling

- Waterfall model
- V-model
- Spiral model
- Unified approach/Unified Process (UP)
  - Inception phase
  - Elaboration phase
  - Construction phase
  - Transition phase

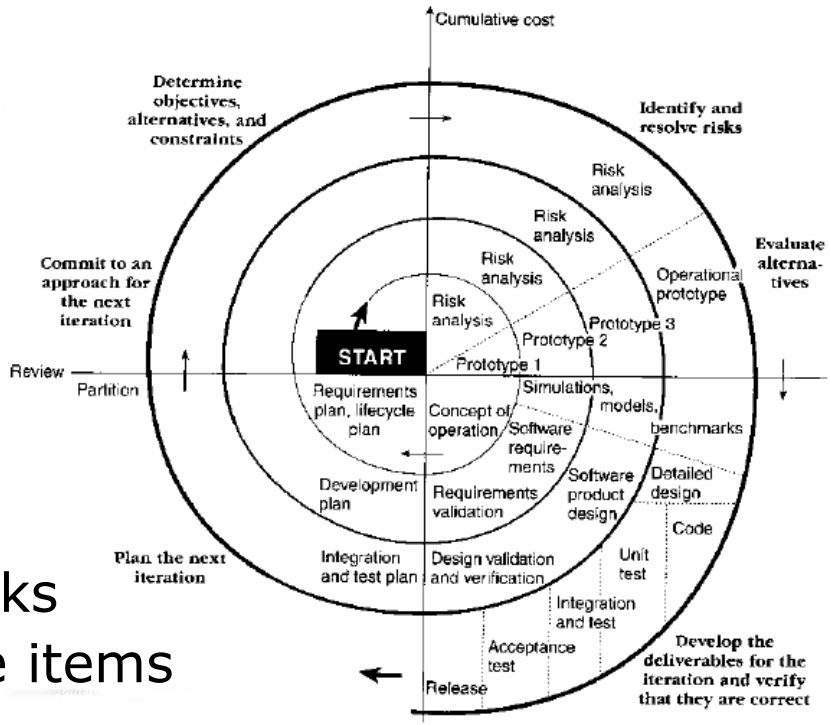
Waterfall



- benefits
  - formal, standard; specific phases with clear goals
  - good feedback loops between adjacent phases
- drawbacks?
  - assumes requirements will be clear and well-understood
  - requires a lot of planning up front (not always easy)
  - rigid, linear; not adaptable to change in the product
  - costly to "swim upstream" back to a previous phase
  - nothing to show until almost done ("we're 90% done I swear!")

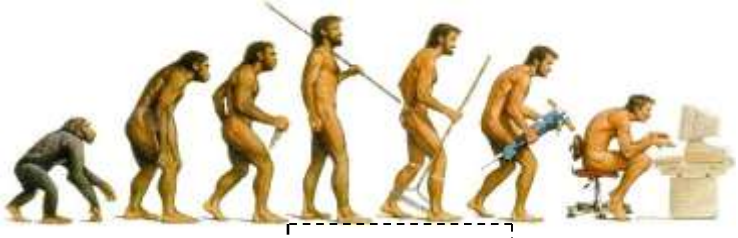
Spiral

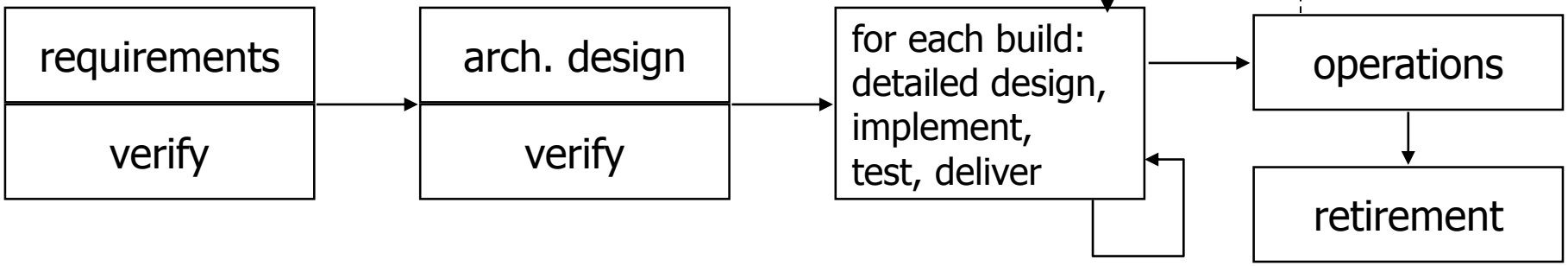
Barry Boehm, USC



- steps taken at each loop:
  - determine objectives, constraints
  - identify **risks**
  - evaluate options to resolve the risks
  - develop and verify any deliverable items
- benefits
  - provides early indication of unforeseen problems
  - always addresses the biggest risk first
  - accommodates changes, growth
  - eliminates errors and unattractive choices early
- drawbacks?
  - relies on developers to have risk-assessment expertise
  - complex; works poorly when bound to an inflexible contract

Evolutionary



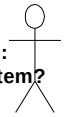


- build initial requirements, code it, "evolve" as needed
  - produces steady signs of progress, builds customer confidence
  - useful when requirements are not well known or change rapidly
  - customer involvement ("What do you think of this version?")
- drawbacks?
  - assumes user's initial spec will be flexible
  - fails for separate pieces that must then be integrated
  - temporary fixes become permanent constraints
  - bridging; new software trying to gradually replace old
  - unclear how many iterations will be needed to finish

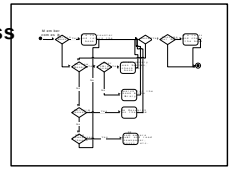
OOAD Phases

Analysis

1. Identify the users/actors (Chapter 6):
   Who is (or will be) using the system?



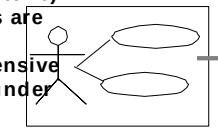
2. Develop a simple business process model



Business process modeling using activity diagram

The advantage of developing a business process model is that it familiarizes you with the system and therefore the user requirements

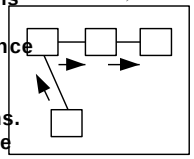
3. Develop the use case (Chapter 6):
   What are (or will be) the users are doing with the system?
   Use cases provide comprehensive documentation of the system under study



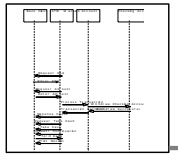
Use case diagrams

Use cases capture the goal of the users and the responsibility of the system to its users

4. Interaction diagrams (Chapter 7)
   4.1 Develop sequence diagrams
   4.2 Develop collaboration diagrams.
   4.3 Iterate and refine

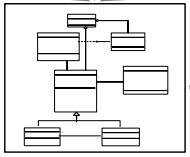


collaboration diagram



Sequence diagram

5. Classification (Chapter 8)
   5.1 Identify Classes
   5.2 Identify Relationships
   5.3 Identify Attributes
   5.4 Identify Methods
   5.5 Iterate and refine.

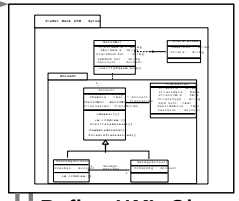


Class diagram

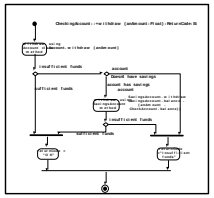
The process of creating sequence or collaboration diagrams is a systematic way to think about how a use case can take place, and by doing so, it forces you to think about objects involves in your application

Design

6. Apply design axioms to design classes, their attributes, methods, associations, structures, and protocols (Chapter 9)
   6.1. Refine and complete the static UML class diagram (object model) by adding details to the UML class diagram (Chapter 10)
      6.1.1 Refine attributes
      6.1.2 Design methods and protocols by utilizing UML activity diagram for representation of method's algorithm
      6.1.3 Refine (if required) associations between classes
      6.1.4 Refine (if required) class hierarchy and design with inheritance
   6.2 Iterate and refine (reapply Design axioms).

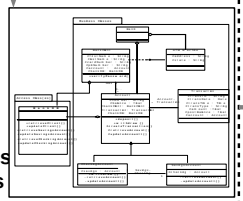


Refine UML Class diagram



Design methods by utilizing UML Activity Diagram

7.0 Design the access ayer (Chapter 11)
   7.1. Create maccess layer classes by mirroring the business classes
   7.2. Define relationships
   7.3. Simplify classes and structures
      7.3.1 Eliminate redundant class
      7.3.2 Eliminate method classes
   7.4 Iterate and refine

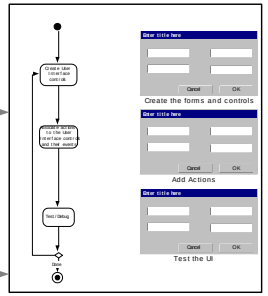


UML Class diagram with added access and view classes

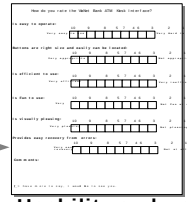
8. Designing view layer classes (Chapter 12)
   8.1 Macro-level UI design Process- Identifying View layer Objects
   8.2 Micro-level UI design activities:
      8.2.1 Designing the view layer objects by applying design axioms and corollaries
      8.2.2 Prototyping the view layer interface.
   8.3. Usability and user satisfaction testing (Chapter 14):
   8.4 Iterate and refine

9. Iterate and refine the design/analysis: If needed repeat the preceding steps

Prototyping and Testing



Prototype user interface



Usability and user satisfaction testing

Quiz

Question 1

1. Name ONE iterative software process model.

   - Agile Process
   - RAD
   - Unified Process

Question 2

2. _____ to extend which software module can be used in different applications.

- Reusability

Question 3

3. _____ is identified with the degree to which system, component or process satisfies specified requirements.

- Quality

Question 4

4. Name TWO quality factors.

- Correctness
- Reliability
- Maintainability
- Testability
- Efficiency
- Usability
- Integrity
- Portability
- Interoperability
- Reusability

Question 5

5. The _____ is a variation of waterfall model that makes explicit dependency between development activities and verification activities.

   - V-Model