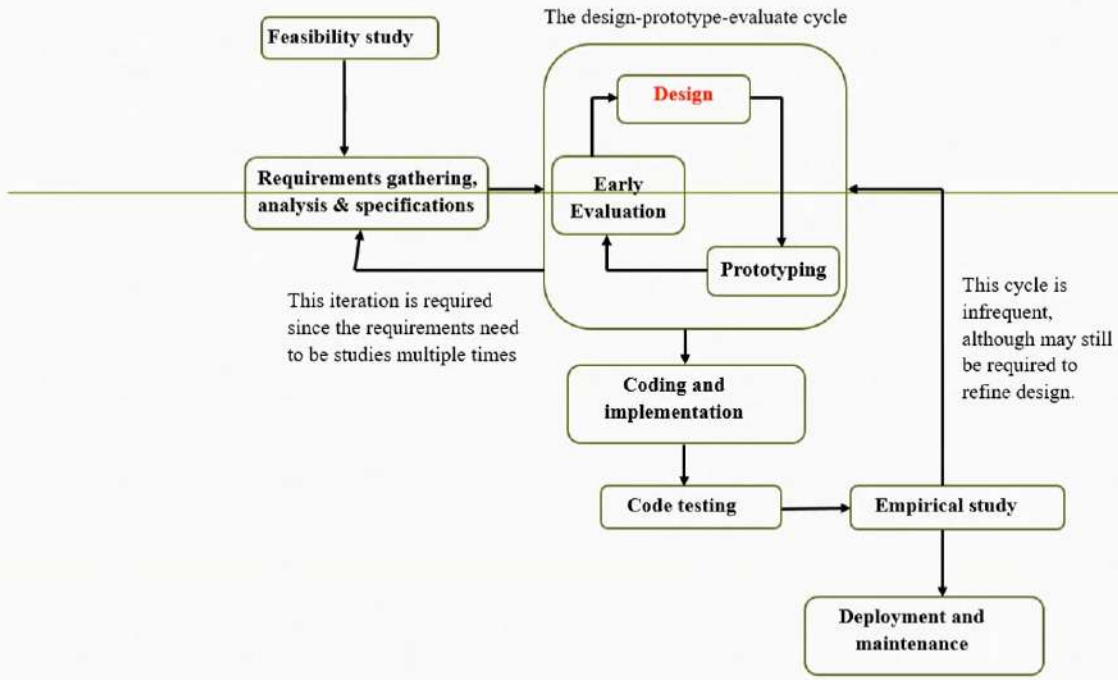
Object-Oriented Design

Dr Samit Bhattacharya
Computer Science and Engineering
IIT Guwahati



Basic Design Approaches

- Function oriented – basic abstractions are functions [use DFD to represent design]
- Object oriented – basic abstractions are objects (instantiation of class; similar objects refer to "class") [use UML to represent design]

OOD

Object Oriented Design Approach

- Different way of thinking about a problem vis-à-vis procedural approach
- Ex – library management system
  - Procedural – design consists of a set of functions (register, issue book, check availability etc)
  - OO – collection of objects and their behavior (member, book, book register etc and interaction between them)

Basic Concepts: Object & Class

- Class – generic definition of objects (contains attributes & operations (also called methods/member functions)
- Object - instantiation of class (assigning values to attributes and initialization of operations)

Example

- Name: **AnyInteger**
  - Check naming convention: lower+upper case
- Attribute: **integerValue**
  - Optionally data type, initial value etc
- Operation/member function: **convertToCharacter**
  - Input = **its** integer value, output = Unicode character value

Relevant Terms

- OOA – object oriented analysis (SRS with objects instead of functions)
- OOD – object-oriented design
- OOP – object oriented programming

UML


+80
YK
GR
NS
NALABOLU SANDEEP
SK
SHIVANGI KUMAR
AC
ARYAN CHAUHAN
SHIVRAJ AHIRWAR
ANIRAJ KUMAR
CHAPPA CHINMAI ANANDH
SB
Samit Bhattacharya

UML (Unified Modeling Language)

- Like DFD, we need a "language" to represent OOD
  - UML provides that
- May be used to visualize, specify, construct, and document artifacts of a software system

UML (Unified Modeling Language)

- Provides a set of notations (e.g. rectangles, lines, ellipses, etc.) to create a visual/graphical model of the system
  - Like any other language, UML has its own syntax (symbols and sentence formation rules) and semantics (meanings of symbols and sentences)
- UML not a design methodology - language to express object-oriented design obtained using some methodology

Views & Diagrams

- NINE diagrams to capture FIVE views of the system
  - —User view (use case diagrams)
  - —Behavioral view (sequence diagram, collaboration diagram, state chart diagram, activity diagram)
  - —Structural view (class diagram, object diagram)
  - —Implementation view (component diagram)
  - —Environment view (deployment diagram)

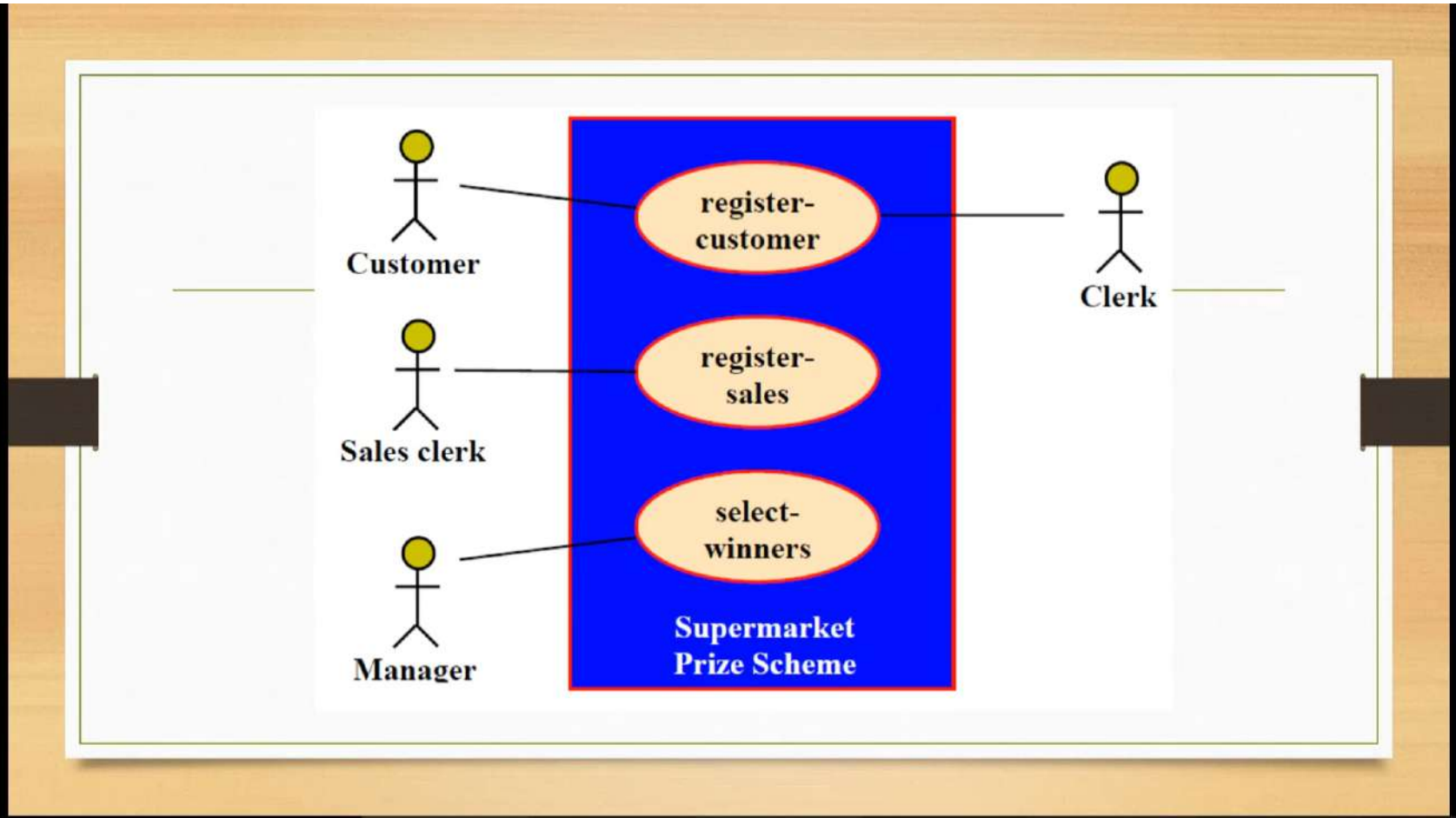
User View

Use Case Diagram (User View)

- Idea: to **represent user perception** of the system (with **dialog/conversation** between user and system to **express 'interaction'**)

Use Case Diagram (User View)

- Basic components
  - Actor (stick figure)
  - System boundary
  - Use case (ellipse)
  - Mainline sequence & alternative sequence



U1: register-customer: Using this use case, the customer can register himself by providing the necessary details.

Scenario 1: Mainline sequence

1. Customer: select register customer option.
2. System: display prompt to enter name, address, and telephone number.
3. Customer: enter the necessary values.
4. System: display the generated id and the message that the customer has been successfully registered.

Scenario 2: at step 4 of mainline sequence

1. System: displays the message that the customer has already registered.

Scenario 2: at step 4 of mainline sequence

1. System: displays the message that some input information has not been entered. The system display a prompt to enter the missing value.

Behavioral View

After Use Case

- What to do after use case identification?
  - Identify objects and classes and how they interact
- How to do that?
  - Experience, intuition, domain knowledge …

After Use Case

- What to do after use case identification?
  - Identify objects and classes and how they interact
- How to do that?
  - Experience, intuition, domain knowledge …
  - Some systematic approach!

Domain Modeling

- A simple and systematic approach to determine classes and objects
- Idea - identify obvious **domain objects** and their relations

Domain Objects

- Boundary objects – those with which the actors interact (screen, menu etc); can be identified from use cases
- Controller objects – coordinate activities between boundary and entity objects
- Entity objects – normally hold information (e.g., book, book registers etc)

Note

- Key thing to do – identify domain objects for **each use case**
  - There may be many similar objects

Interaction Diagram

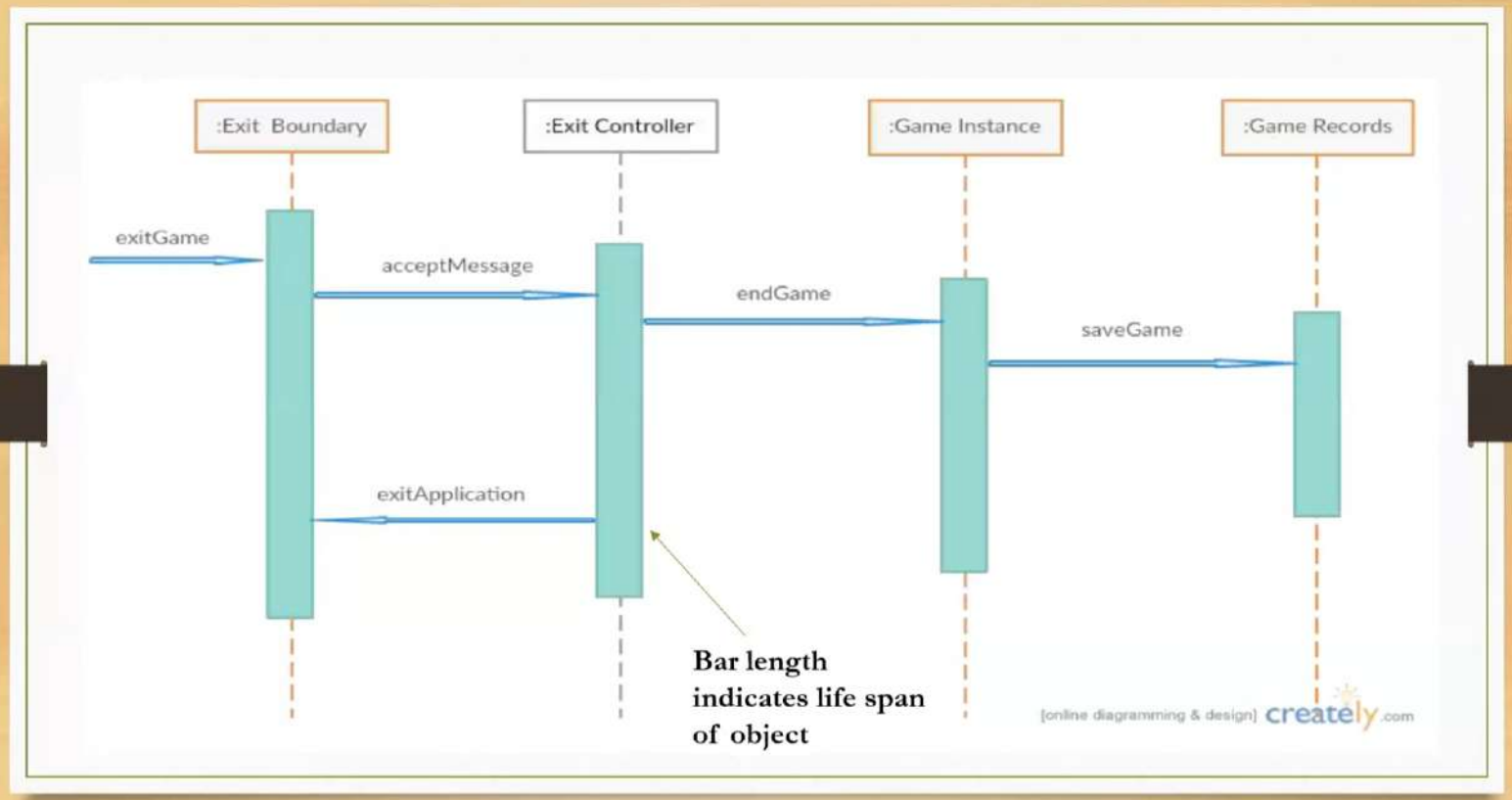
- Next create an **interaction diagram** for each use case execution scenario
  - To refine list of objects
  - Also to identify classes by clubbing together similar objects

Interaction Diagram

- Captures dynamic system behavior - typically one diagram per use case
- Many types
  - —Sequence diagram
  - —Collaboration diagram
  - —Activity diagram …

Interaction Diagram

- Captures dynamic system behavior - typically one diagram per use case
- Many types
  - —Sequence diagram
  - —Collaboration diagram
  - —Activity diagram …



Recap

- OOD sequence: Use case analysis → domain modeling → sequence diagram
  - Help to identify objects, classes and their behavior (over time)

Recap

- OOD sequence: Use case analysis → domain modeling → sequence diagram
  - Help to identify objects, classes and their behavior (over time)
  - Needs many iterations and experience
  - Easier said than done!

What's Next?

- We need to represent "structure" of the design
  - Classes and their relationship

Structural View

Class Diagram

- Describes **static** system structure
- Comprised of classes and their relationship

Class Representation

- Name: **AnyInteger**
- Attribute: **integerValue**
- Operation/member function: convertToCharacter [input = integer, output = Unicode value]

AnyInteger **[class name to be put here]**
integerValue: integer **[attribute names in this block]**
Number convertToCharacter(number) **[member functions listed here]**

Class Relationships

- Association
- Aggregation (whole-part relationship)
- Composition (a stricter form of aggregation)

'Association' Relationship

- Describes connection between classes
- **Links** – instances of association between two objects
  - Ex - Ramesh borrowed book "Operating System" - **borrowed** is the link between objects Ramesh (instance of class 'student') and Operating System (instance of class 'book')
- Association describes a **group of links** with **common structure and common semantics**

‘Association’ Relationship

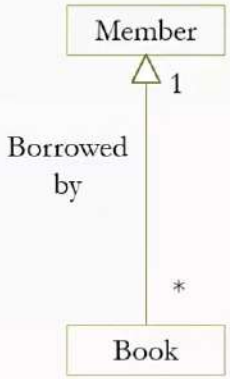
- Usually, association is a binary relation (between two classes)
  - However, 3/more classes can be involved in an association

‘Association’ Relationship

- Usually, association is a binary relation (between two classes)
  - However, 3/more classes can be involved in an association
- A class can have an association relationship with itself (**recursive association**)
  - Assumption - two different objects of the class are linked by the association relationship

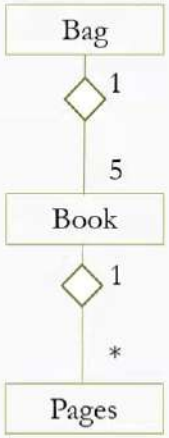
Representing 'Association'

- By a straight line between classes - name written along side line
- An arrowhead may be placed to indicate direction of association
- On each side of line, **multiplicity** is noted
  - Indicates no of instances of one class associated with other
  - Value ranges noted by specifying minimum and maximum, separated by two dots (e.g., 2..4)
  - An asterisk is a wild card and means many (zero or more)



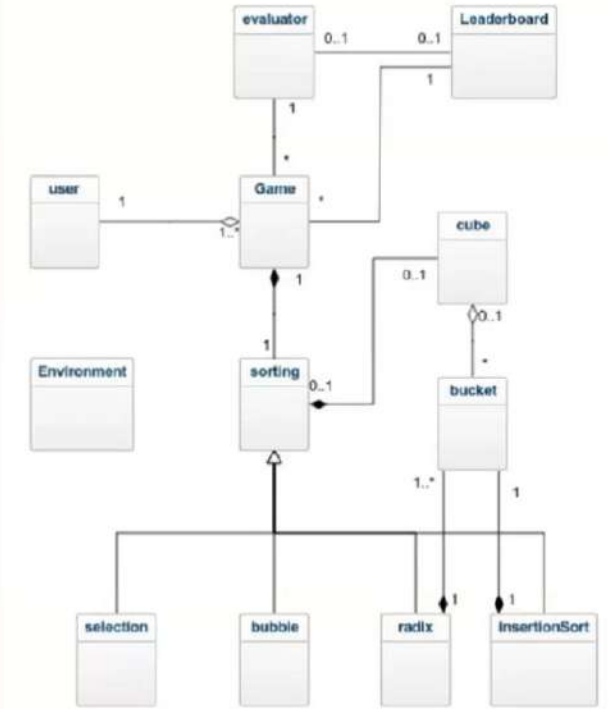
‘Aggregation’ Relationship

- A special type of association - involved classes represent a **whole-part** relationship
- Represented by diamond symbol at the ‘whole’ end of a relationship



‘Composition’ Relationship

- A **stricter** form of aggregation - parts are **existence-dependent** on the whole
  - When whole is created, parts are created and when whole is destroyed, parts are destroyed
- Represented as a filled diamond drawn at the ‘whole’ end



4.1 Game

4.1.1 Game –Data members

1. Players
2. Score
3. createdAt
4. sorting
5. duration

4.1.2 Game – Methods

1. getPlayer()
   - Parameters : NA
   - Return Value : Returns an object of the User Class
   - Description : getPlayer() method is used to obtain the entire details of the user which are associated to the object of the GameClass.
   - Called By : method is called in the main program/activity.
   - Calls : method calls getName() and getAge() method of the User Class.

Putting Everything Together: OOD Process