Towards Object Oriented System Development

A Lecture on System Development Course

Avinanta Tarigan

Magister of Information System Management
Gunadarma University

Outline

Outline

General System Theory

- Initiated by Ludwig von Bertalanffy

- Followed by Mead and Bateson

- The study of the nature of complex systems in nature, society, and science
- Interdiciplinary study: single organism, any organization or society, or any electro-mechanical or informational artifact
- Cybernetics: study of feedback $\rightarrow$ control systems

System of Objects

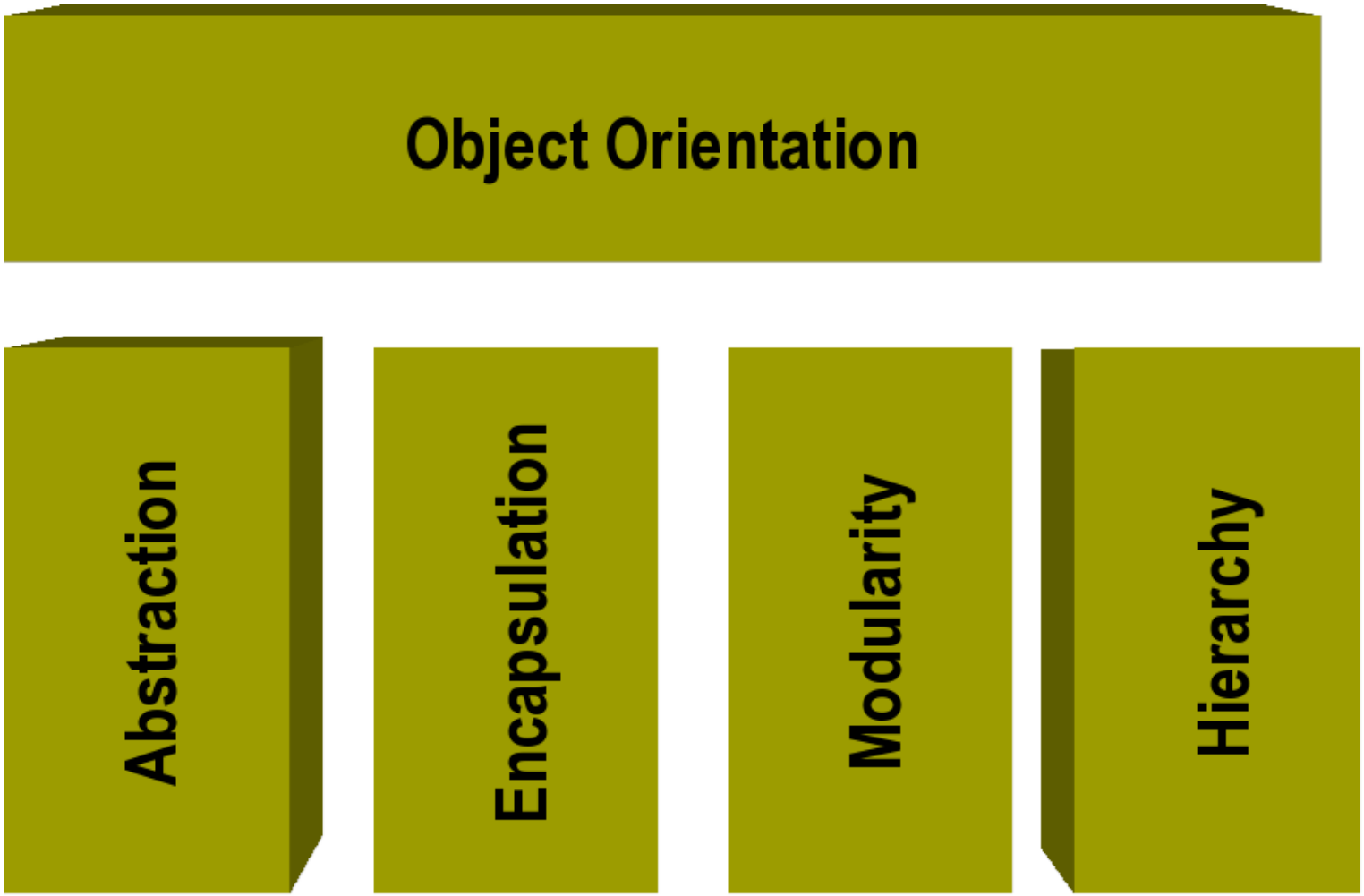
- System consists of objects
  - A group of MLM consists of members
  - An object is something that can be identified
- Every object has relationship(s) with others
  - Hierarchical structure of MLM group
- Objects engage constituting system's behavior
  - Every member is active and vigorous
  - The MLM group is becoming rich
- A state of system is overall state of the objects
  - The MLM group earned 100 million Rp last year
  - The MLM group earned 120 million Rp this year
- System behavior = sequence of states

Information System (viewed with GST)

- Objects in information system:
  - Data / Information about something
  - Algorithm / program that process the data
  - Persons / entity / operator that operate the program
  - Devices that interact with the system
- What is the state of the system ?
  - the value of data / information at certain time
  - operator's perception of the system state at certain time
- What drives the behavior of system ?
  - Algorithm implemented in program
  - Parameter (decision of operator, stored data, output from a device)

Outline

Principles of Object Oriented I



Principles of Object Oriented II

- Abstraction:
  - identification of classes of objects in the system
  - e.g. SalesPerson, Customer, Product
- Encapsulation
  - Hide implementation and internal structure from clients
  - Clients depend on interface
  - Manages complexity of the system
- Modularity
  - The breaking up of something complex into manageable pieces
- Hierarchy
  - Levels of abstraction

Concepts in Object Oriented I

- Object:
  - An object is a concept, abstraction, or thing with sharp boundaries and meaning for an application
  - An object is something that has:
    - State
    - Behavior
    - Identity
- Class
  - A class is a description of a group of objects with common properties (attributes), behavior (operations), relationships, and semantics
  - An object is an instance of a class
  - A class is an abstraction in that it: Emphasizes relevant characteristics Suppresses other characteristics
- Attribute

Concepts in Object Oriented II

- Property of the object / class that has value
- State of an object is overal values of attributes of the object at certain time

- Operation / Methods
  - What drives the change the state of the object
  - Procedure / function / algorithm
- Interface (Polymorphism)
  - The ability to hide many different implementations behind a single interface
  - Interfaces formalize polymorphism
  - Interfaces support “plug-and-play” architectures
- Component
  - A non-trivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture

Concepts in Object Oriented III

- A component may be :
  - A source code component
  - A run time components
  - An executable component
- Package
  - A package is a general purpose mechanism for organizing elements into groups
  - A model element which can contain other model elements
- Subsystem
  - A combination of a package (can contain other model elements) and a class (has behavior)
  - Realizes one or more interfaces which define its behavior
- Relationships
  - Relationship is defined through services provided by objects

Concepts in Object Oriented IV

- Interfaces: public methods + passing parameter
- Types / structure of relationships:
  - Association
    - Aggregation
    - Composition
  - Dependency
  - Generalization
  - Realization

Outline

System Behavior Analysis and Design

- Traditional Approach:
  1. Analysis that elicites requirements
  2. Logical DFD
  3. Physical DFD
  4. Program structure charts & process descriptions
  5. Data model (ERD) & physical database design
- Object Oriented Systems Development
  1. Analysis that elicites requirements
  2. Object relationship model with attributes and relationships
  3. Object relationship model with attributes, relationships, and methods
- Final joint of OOSD is the program / algorithm itself

Designing Object-Oriented System Behavior I

1. Analyze use cases to identify classes and behaviors
    1. Identify all classes (objects) in the system
    2. Identify behavior of objects (classes)
    3. Identify all relationships of the identified classes
2. Define methods and assign to object classes
    1. Define attributes
    2. Define automated behaviors (methods)
        1. Method used to create and break relationships
        2. Method that changes the state of an object
3. Extend the object relationship model to include methods, and create a class specification for each class in the model
    1. Define the business-name of the classes and all attributes / methods
    2. Define service interface - listing the public methods

Designing Object-Oriented System Behavior II

3. Define detailed methods (algorithm)
4. Reflect classes, methods, and messages in formal use cases
5. “Walk through” formal use cases to verify design

Criteria for Evaluating System Behavior Design

- Low Coupling
  - minimize interdependency of modules / classes
  - more lossely-coupled the objects $\rightarrow$ greater reusability
  - Guideliness:
    - No message should require more than three parameters
    - No messages should be sent from a subclass to access data defined in superclass (do this in inheritance)
    - Collaboration relationships should not be so strong that class *A* is useless without class *B*
- Cohesiveness
  - A cohesive design specifies only one function per-method
  - Function is clearly expressed as verb
- Clarity
  - High quality design speaks for itself
  - Minimize the “gap” of perception of the system between analyst, designer, programmer, implementer
- Simplicity

Outline

Object Oriented Design with UML

Please see example from Russell C. Bjork

and

Those from Maciaszek

UML is not the best ?

- From definition:
  - Unified/Universal Modeling Language (UML) is a standardized **visual specification** language for object modeling
- Visual specification contains error caused by wrong interpretation
- UML aids the design in OO manner but not as strong as formal method
- Designing System using Formal Method requires in-depth knowledge and skill in mathematics

Example of Formal Method

MODULE *AsynchInterface*

EXTENDS *Naturals*
CONSTANT *Data*
VARIABLES *val*, *rdy*, *ack*

$$TypeInvariant \triangleq \begin{array}{l} \land\ val \in Data \\ \land\ rdy \in \{0, 1\} \\ \land\ ack \in \{0, 1\} \end{array}$$

$$Init \triangleq \begin{array}{l} \land\ val \in Data \\ \land\ rdy \in \{0, 1\} \\ \land\ ack = rdy \end{array}$$

$$Send \triangleq \begin{array}{l} \land\ rdy = ack \\ \land\ val' \in Data \\ \land\ rdy' = 1 - rdy \\ \land\ \text{UNCHANGED}\ ack \end{array}$$

$$Rcv \triangleq \begin{array}{l} \land\ rdy \neq ack \\ \land\ ack' = 1 - ack \\ \land\ \text{UNCHANGED}\ \langle val, rdy \rangle \end{array}$$

$$Next \triangleq Send \lor Rcv$$

$$Spec \triangleq Init \land \Box[Next]_{\langle val, rdy, ack \rangle}$$

THEOREM $Spec \Rightarrow \Box TypeInvariant$

Why UML descents into darkness

1. Design by committee
   - not by community (lack of acceptance)
2. The obsessive focus on monetizing UML
   - Just draw the pictures and we generate the code for you
3. Tries to unify everything even the kitchen sink (UML specs>800 pages)
4. UML attempts to become a programming language
5. The need for expensive tools vs. just a text editor
6. Lack of solutions for real software design issues
7. Assumes you can know everything before writing the first line of code
8. Treat software development like manufacturing
9. UML tools focus on the wrong goal
10. etc

Thanks

The END