

OBJECT ORIENTED DESIGN

127

5 Criteria in OO Design

- Decomposition: approach?
- Integration: usability of modules by different systems
- Understandability: of an object in isolation
- Continuity: changes should affect only a few modules
- Protection: Structural property to prevent side-effects to propagate to other modules.

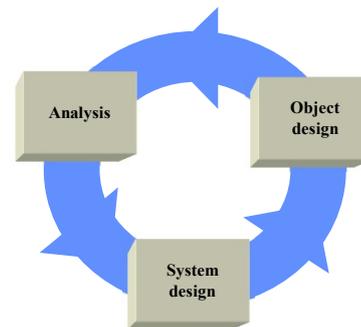
128

Modularity in OO Design

- Language structures to support modularity
- Little interface
- Reduced information flow through interface
- Understandability of an interface in isolation (without having to refer to global structures)
- Information hiding

129

Design in OO methodologies



130

Booch

- Structural Planning
 - Gather similar objects
 - Separate objects with respect to abstraction levels
 - Define scenarios
 - Create design prototype
 - Test the prototype with usage scenarios
- Tactical design
 - Definition of the rules for the usage of attributes and operations
 - Definition of the rules for foundations such as memory management and error messages
 - Development of a scenario to understand the rules
 - A prototype for each rule (policy)
 - Refining of the prototype
 - Revision of every policy for the structural vision

131

Booch - II

- Version Planning
 - Priorities are assigned for the scenarios developed in analysis
 - Structural versions are assigned to scenarios
 - Structural versions are sequentially designed and developed
 - Goals and dates for the sequential versions will be adjusted as required

132

Coad and Yourdon

- **Problem domain component**
 - Group all the classes defining the domain
 - Create hierarchies for application classes
 - Simplify inheritance as appropriate
 - Apply design for efficiency
 - Add and refine lower-level objects as required
 - Review design and question the additions for the analysis
- **Human interaction component**
 - Define the users
 - Develop the task scenarios
 - Define the hierarchy for user commands
 - Refine the user interaction ordering
 - Design related classes and their hierarchies
 - Integrate with Graphical User Interface

133

Coad and Yourdon - II

- **Task management component**
 - Define task types (such as event or clock triggered)
 - Determine priorities
 - Assign a task as a manager for another
 - Design appropriate objects for each task
- **Data management component**
 - Design data structures
 - Design required service operations for data management
 - Define tools for data management
 - Design appropriate classes and hierarchies

134

Jacobson

- Adjust the analysis model for matching the real-world
- Define blocks as fundamental design objects
 - Define a block for related analysis objects
 - Define interface, entity and control blocks
 - Explain how blocks will communicate in run-time
 - Define the signals and their sequence among the blocks

135

Jacobson - II

- Draw interaction diagrams to show the signals among blocks
- Organize blocks under subsystems
- Review the design

136

Rumbaugh

- **System Design**
 - Decompose the analysis model to subsystems
 - Define the synchronizations the system requires
 - Distribute subsystems among processors and tasks
 - Select a main strategy for data management
 - Common resources and structures for their accesses are defined
 - A suitable control mechanism for the system is defined
 - Consider the boundary conditions
 - Review the racing conditions

137

Rumbaugh - II

- **Object Design**
 - Select operations from the analysis model
 - Define the algorithm for each operation
 - Define suitable data structures for the algorithms
 - Define the internal classes
 - Review the object arrangement for data access and computation efficiency
 - Design the class attributes
- Applying the control mechanisms for the System design
- Adjust class structures to strengthen inheritance
- Message design for object relations
- Collect classes under modules

138

Fusion

- Inheritance and class structures
 - Defined
 - Refined
- Objects and messages are defined
- Dynamic behavior specification if required

139

UML

- Refine Class and object structures
- Message specifications
- Refine state specifications for classes

140

Booch / OMT / UML

- Comments



- Subsystems



- Objects



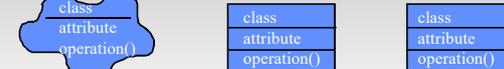
141

Booch / OMT / UML - II

- Classes



- Attributes and operations



- Visibility

public n/a	+ public	+ public
protected	# protected	# protected
private	- private	- private
implementation		

142

Code: access to private member of an object of the same class?

```
class student {
    private int grade;
    void get-others-grade (student * other);
};
```

```
int student:: get-others-grade (student * other) {
    return other* grade;
};
```

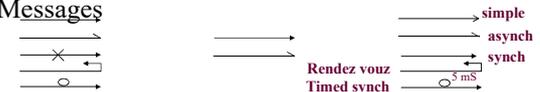
```
int main {
    student s1, s2
    cout << s2.get-others-grade (@s1);
    ....
}
```

// do not declare getter or setter functions. This is all you should use !!!
Also write the answer as one word (YES/NO) if it can access, on one line.

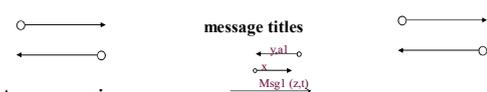
143

Booch / OMT / UML - III

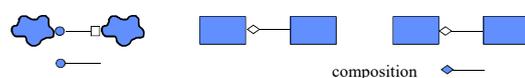
- Messages



- Data flows



- Aggregation



144

Ada language and asynchronous messages

```
a := 3 + b;  
wait (senderPck, msgName, par1);  
b := par1 + 3;
```

```
When Begin  
  msg1: do this function;  
  msg2: do other;  
  msg 3 & msg4 | msg1 : do something else;  
End
```

145

UML Design Diagrams

- Most diagrams are used in requirements also
 - only Use Case diagrams are strictly for requirements
- Trend (dynamic modeling): Activity diagrams for requirements, Interaction diagrams for design
- Strictly design: Component, Deployment

Views: use case, logical, implementation, deployment, ...

- Different types of diagrams can support one view.

146