

Overview of OO

- OO Modeling
- History
- UML Diagrams
- Problems with structural programming
- OO concepts
- OO relationships
- Problems with OO

72

OO Modeling

- A model is an abstraction of something for the purpose of understanding or analyzing it before building it
- Abstraction: omits nonessential details
- Reduce size and complexity
- Widely used technique

73

Why Modeling?

- Test a physical entity before build it
 - Cheaper
 - Accessible
 - Safe
- Communicate with customers
- Visualization
- Reduce complexity

74

OO Past and Present

- Early 1990s :
 - Booch method: “*Object-Oriented Design with Applications*”, Redwood city, 1991
 - OMT: “*Object-Oriented Modeling and Design*”, Prentice-Hall, Rumbaugh et al., 1991
 - OOSE: “*Object-Oriented Software Engineering: A Use Case Driven Approach*”, Addison-Wesley, Jacobson et al., 1993
- Late 1990s: Unified Modeling Language (UML)
 - *The Unified Modeling Language User Guide*
 - *The Unified Modeling Language Reference Manual*
 - <http://www.omg.org/uml/>

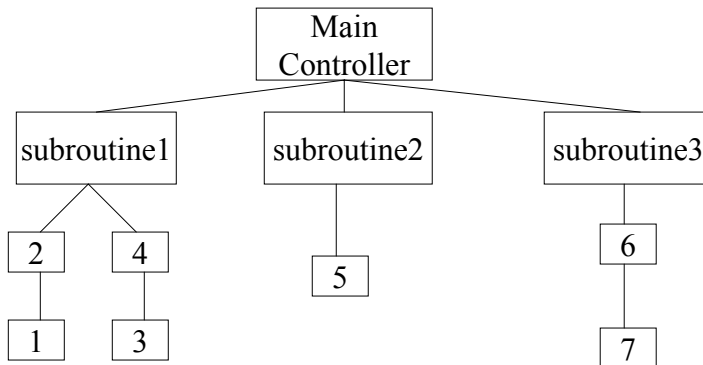
75

UML Diagrams

- Structural diagrams
 - Class diagram
 - Object diagram
 - Component diagram
 - Deployment diagram
- Behavioral diagrams
 - Use case diagram (for requirement gathering)
 - Sequence diagram
 - Activity diagram
 - Collaboration diagram
 - Statechart diagram
- Model Management diagrams
 - Packages,
 - Subsystems
 - Models

76

Main Program/Subroutine



77

Main Program/Subroutine

- Hierarchical decomposition
 - Based on definition-use relationship
- Single thread of control
 - Supported directly by programming languages
- Subsystem structure implicit
 - Subroutines typically aggregated into modules
- Hierarchical reasoning
 - Correctness of a subroutine depends on the correctness of the subroutines it calls

78

Problems

- Access to internal representation
 - Vulnerability of global (or widely shared) variables which can be manipulated in unexpected, undesired, and dangerous ways
 - Classical languages create sharing (block structure, global variables)
 - Nonlocality: if the way something is used depends on its implementation, you must find all uses to change it (e.g. Y2K)
- Inadvertent disclosure of structure
 - Exact location of field, linear vs linked representation
- Rippling design decisions
 - one change may affect many modules
- Dispersion of code related to a single decision
 - It may be hard to locate everything affected

79

Problems

- Families of related design
 - Related definitions de-localize decisions
- Forced distribution of knowledge
 - Non-uniform referents: syntax may reveal structure (If you export a data structure, how does its user iterate through it?)
- Coupling
 - Instance dependence: when multiple instances of a given structure are active, they must remain independent
- Families of definitions
 - Dynamic binding: if shared definitions involve type invariants, function variants must be chosen at run-time

80

OO Basic Concepts

- Object: an entity that has attributes and provides operations
 - Identity: data is quantized into discrete, distinguishable entities
 - Two objects are distinct even if all their attribute values are identical
 - Each object has a unique handle by which it can be uniquely referenced
- Class: description of the scheme of an object
 - Objects with the same data structure (attributes) and behavior (operations) are grouped into a class
 - An abstraction that describes properties important to an application and ignores the rest
- Subclass: class with additional properties
 - More restrictive than class

81

OO Basic Concepts

- Instance: object of a class
- Operation: an action or transformation that an object performs or is subject to
- Method: a specific implementation of an operation by a certain class
- Message: procedure call; request to execute method
- Abstract class: a class that has no direct instance but whose descendent classes have instances
 - Abstract operation defines the form of an operation for which each concrete subclass must provide its implementation

82

More OO Concepts

- Encapsulation (information hiding)
 - Restrict access to certain information
- Inheritance
 - Share one definition of shared data and functionality
- Polymorphism/Dynamic binding
 - Determine actual operation to call at run-time
- Management of many objects
 - Provide structure on large set of definitions
- Reuse and maintenance
 - Exploit encapsulation and locality

83

Encapsulation

- Parnas: hide secrets (not just representations)
 - Representation of data
 - Properties of a device, other than required properties
 - Implementation of models
 - Mechanisms that support policies
- Booch: object's behavior is characterized by actions that it suffers and that it requires
- Practically speaking:
 - Object has state and operations, but also has responsibility for the integrity of its state
 - Object is known by its interface
 - Object is probably instantiated from a template
 - Object has operations to access and alter state and perhaps generator
 - There are different kinds of objects (e.g., actor, agent, server)

84

Encapsulation

Automobile
Engine
Wheel
Door
Steering
Stop
Run
Reverse
Turn



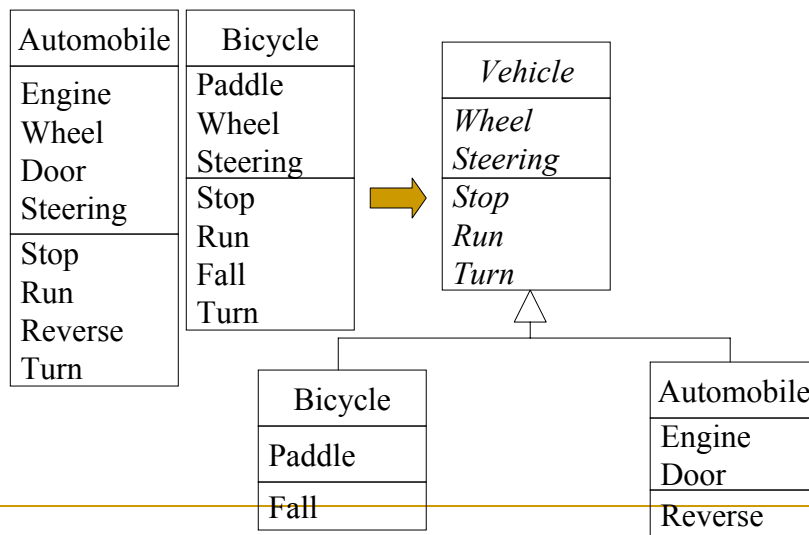
85

Inheritance/Generalization

- A relationship between a class and one or more refined versions of it
- Sharing of attributes and operations among classes based on a hierarchical relationship
- Define broadly and refine into finer subclasses
- Subclasses incorporate all properties of its superclass
- Subclasses can only add behavior
 - Instance variables and methods
- Transitive across an arbitrary number of levels
- Reuse

86

Inheritance/Generalization



87

Polymorphism/dynamic Binding

- The same operation may behave differently on different classes
- Operation: an action or transformation that an object performs or is subject to
- Method: a specific implementation of an operation by a certain class
- A polymorphic operation can have more than one method implementing it

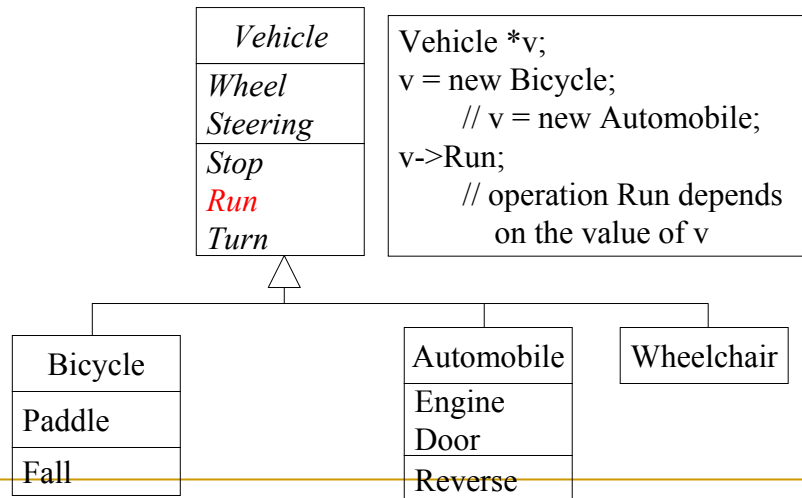
88

Polymorphism/dynamic Binding

- Real world: an operation is an abstraction of analogous behavior across different kinds of objects which know how to perform their own operations
- OOPL: automatically select the correct method to implement an operation based on
 - the name of the operation
 - the class of the object being operate on
- User: need not to be aware of how many methods exist to implement a given polymorphic operation
- Evolution: new classes can be added without changing existing code

89

Polymorphism/dynamic Binding



90

Relationships

- Generalization/Inheritance
- Association/Link
- Aggregation

91

Association/Link

- A link is a physical or conceptual connection between object instances
- An association describes a group of links with common structure and common semantics, e.g., a person *works-for* a company
- A link is an instance of an association
- Associations are bidirectional, e.g., *works-for* and *employs*
- Association are implemented as pointers from one objects to another, not necessarily in both directions
- Multiplicity: specifies how many instance of one class may related to a single instance of an associated class, e.g., "1", "1+", "2-6", "3,5,9"

92

Association/Link



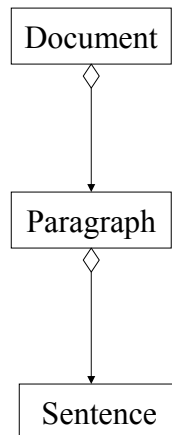
93

Aggregation

- Part-whole or a-part-of relationships
- Objects representing the components of something are associated with an object representing the entire assembly
- An assembly with many kinds of components corresponds to many aggregation relationships
- A special form of association
- Transitivity:
 - A is part of B, B is part of C, A is part of C
- Antisymmetry:
 - A is part of B => B is not part of A

94

Aggregation



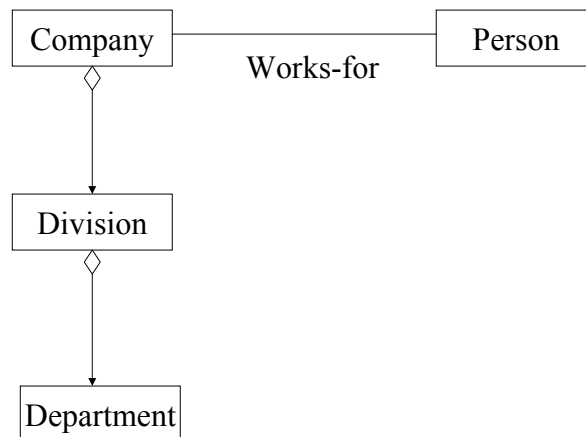
95

Aggregation vs Association

- Aggregation is a special form of association
- Two object
 - Part-whole => aggregation
 - Independent => association
- How to distinguish?
 - Do you use the phrase *part of*?
 - Are some operations on the whole automatically applied to its parts?
 - Are some attribute values propagated from the whole to all or some parts?
 - Is there an intrinsic asymmetry to the association, where one object class is subordinate to the other?

96

Aggregation vs Association



97

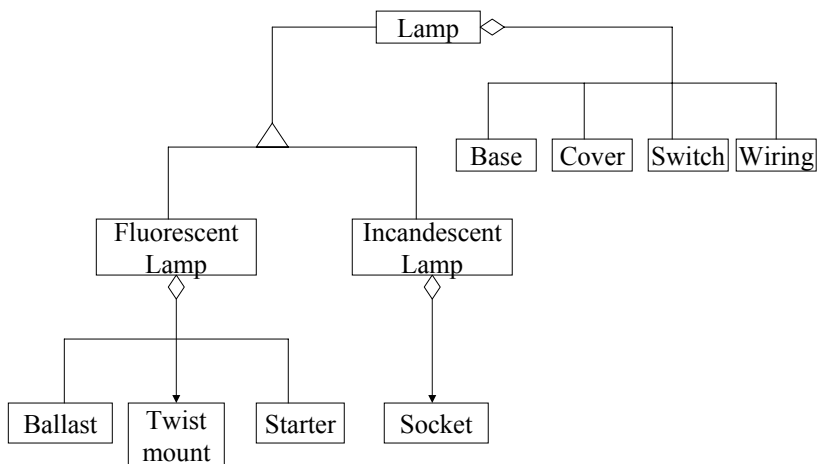
Aggregation vs Generalization

- A-part-of
- Relates to instances
- And-relationship
- A-kind-of (is-a)
- Relates to class
- Or-relationship

Both give rise to trees through transitive closure

98

Aggregation vs Generalization



99

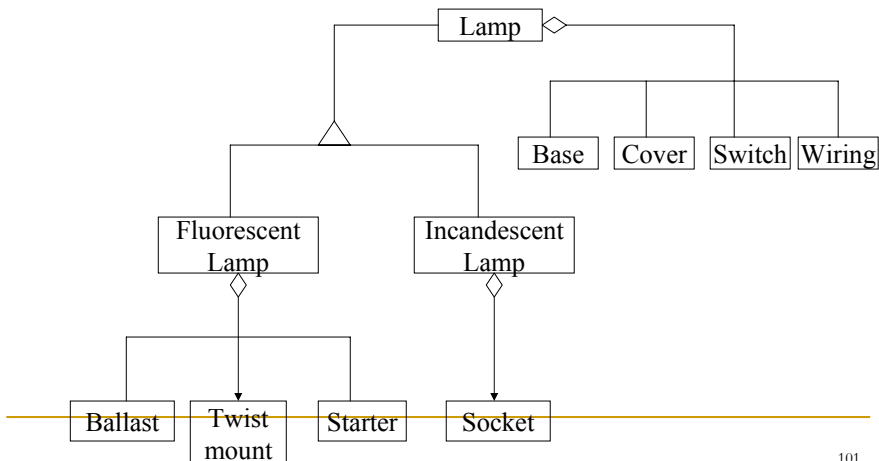
Kinds of Aggregation

- Fixed aggregate
- Variable aggregate
- Recursive aggregate

100

Fixed Aggregation

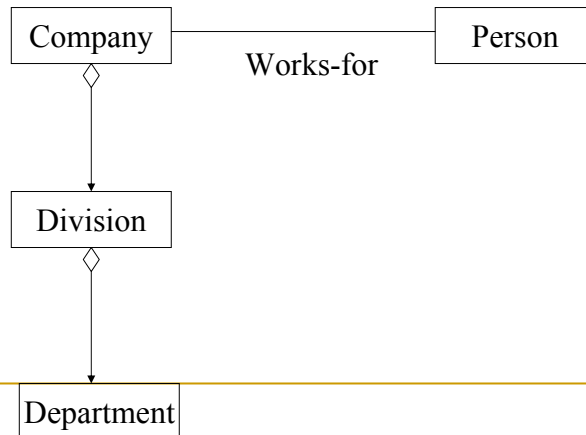
- The number and types of subparts are predefined



101

Variable Aggregation

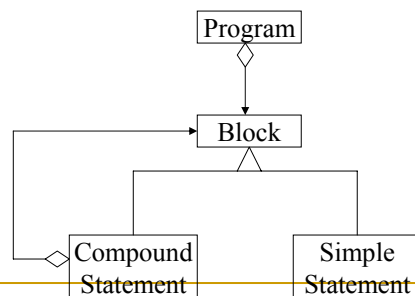
- The number of parts may vary, but a finite number of levels



102

Recursive Aggregation

- Contains an instance of the same kind of aggregate
- The number of potential level is unlimited



103

Evolution/Change

- Localize future change
 - Hide system details likely to change independently
 - Expose in interfaces assumptions unlikely to change
- Use functions to allow for change
 - They are easier to change than visible representation

104

Model the Real World

- It's intuitive if we understand the domain then we are led to a natural system structure based on the domain
- The real world doesn't change much, so systems that model it are unlikely to change much either
- Capture families of related designs through use of templates, and inheritance

105

Problems with Object Approach

- Managing many objects
 - Pure OO design leads to large flat systems with many objects
 - Same old problems can reappear
 - Hundreds of modules => hard to find things
 - Need a way to impose structure
 - Vast sea of objects requires additional structuring
 - Structuring options
 - Layers (which are not necessarily objects)
 - Supplemental index
 - Hierarchical decomposition: big objects and little objects

106

Problems with Object Approach

- Managing many interactions
 - Single interface can be limiting & unwieldy (friends)
 - Some languages/systems permit multiple interfaces
- Distributed responsibility for behavior
 - Make system hard to understand
 - Interaction diagrams now used in design
- Capturing families of related designs
 - Types/classes are often not enough
 - Design patterns as an emerging off-shoot

107