

Lecture 11: UML Terminology and Additional Models and Notation

UML Terminology 1

- *Classifier*: actor, class, interface, component (physical piece of system), subsystem (package with spec implementation and identity), use case (static system: classifiers and their relationships)
- *Class* A set of objects with a state and a behavior, generalization of concept of a program class
- *Implementation Class* An actual software class

UML Terminology 2

- *Interface* Description of behavior of object with giving implementation or state information, operations but no attributes
- *Type* Like an interface except can also have attributes and associations. Is a specification of a class. Has no methods but can have method specifications (operations). (E.g. Concept class in domain modeling)
- *Data type* primitive values that lack identity e.g. numbers, characters, etc

UML Terminology 2

- *Operation* Description of a function/method
- *Method* Implementation of an operation

UML Diagrams

- Use Case
- Class
- Interaction Sequence
- Collaboration
- Package
- Activity
- Deployment
- State

UML Class Diagrams

- Generic, used for both Domain and Design Class Models
- Additional kinds of associations
 - Generalization
 - Aggregation

Generalization

- A general term in UML, also applicable to classifiers other than classes, such as actors, Use Cases, etc.
- Used to indicate that one kind of classifier is a generalization of another
- When applied to classes, similar to inheritance

Generalization and Types

- *Subtype* A subset of the instances of some classifier, having special properties of their own. All members of a subtype set are members of the supertype set
- *Supertype* More abstract classifier having common properties of subtypes. A supertype set contains all elements in subtype sets.

Rules for Creating SubTypes

- Correctness guidelines
 - When is a generalization correct?
- Usefulness guidelines
 - A generalization may be correct but not that useful
 - The goal is not to create as many classes as possible

Generalization Correctness Rules

- *is-a* An instance of a subtype is also an instance of the supertype. e.g. an administrator is a DS user
- *Substitutability* Suppose B is a subtype of A. It should be possible to substitute an instance of B any place that requires something of type A.
- *100% rule* All of the supertype's definition should also apply to the subtype (i.e. its attributes, associations)

Generalization Usefulness

Guidelines – SubType Creation

- Subtype has additional attributes of interest
- Subtype has additional associations of interest
- Subtype is operated on differently than other supertype members
- Subtype is animate and behaves differently in ways of interest

Generalization Usefulness

Guidelines – SuperType Creation

- Subtypes will be variations on a general theme
- Subtypes have common attributes that can be factored out and given to supertype
- Subtypes have common associations that can be factored out and given to supertype

Subtypes versus States

- States and subtypes: describe different kinds of possible behavior of a type of object
- The subtype of an object is fixed
- The state of an object can change

Aggregation

- A has an aggregation relationship with B and C if they are parts of A
- Fuzzy distinctions between
 - Aggregation
 - Composition
 - Association
- E.g. DS MemberData has a DateeData

Composition

- Strong form of aggregation
 - Parts only belong to one whole
 - If whole is deleted, parts get deleted
- E.g. DS: DateeData object can belong to one or more MemberData objects -> not composition
- E.g. a button object will only be part of one frame/dialog

Aggregation Correctness Rules

- *has-a* The composite object has an instance of each class that it aggregates
- Contrast with the is-a correctness rule for generalization

Aggregation Usefulness Rules

- Lifetime of part bound within lifetime of composite (if not, -> association only?)
- Whole-part physical or logical relationship
- Some properties of composite are automatically part of properties of parts (e.g. location)
- Some operations of composite are automatically applied to the parts

Generalization vs Aggregation 1

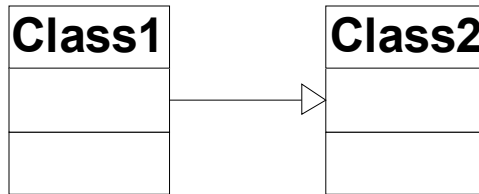
- Defining a new class, e.g. stack from vector
- Generalization
 - Subtype vector and define new stack class operations that use the inherited operations
- Aggregation
 - New stack class has a vector class variable
 - Use define stack operations using vector operations

Generalization vs Aggregation 2

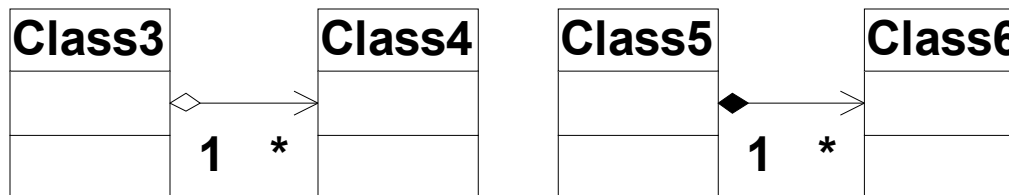
- Which is the better OO definition of a stack?
- Apply the is-a/substituability and has-a correctness rules
 - × Stack is-a vector? substitute a stack any place where you need a vector?
 - ✓ Stack has-a vector?

Notation

- Generalization

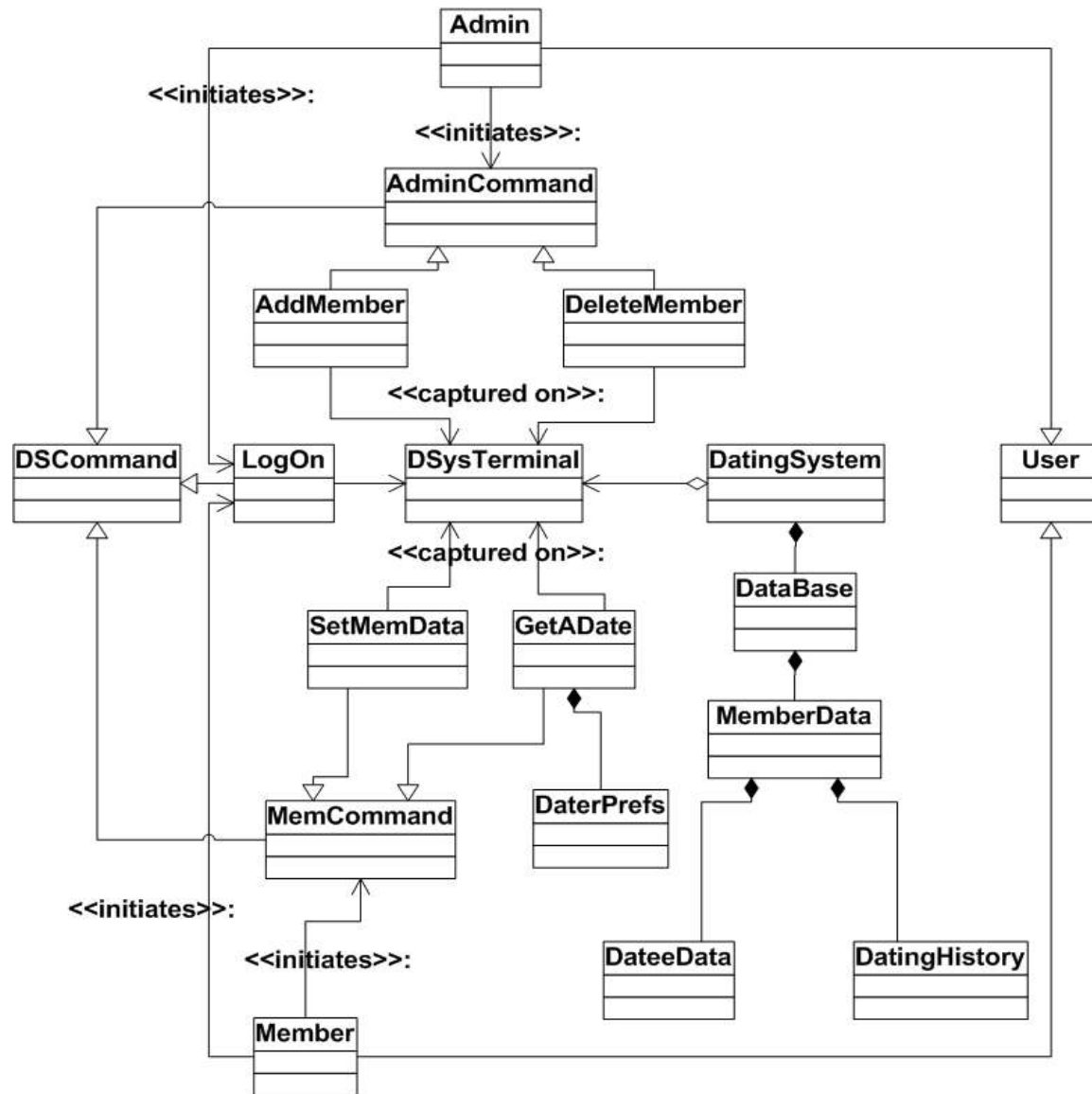


- Aggregation and Composition



Phase 2 DS Domain Model

- Includes use of generalization, aggregation and composition notation
- Leaves out class/concept attributes
- Assumes a single DS terminal



Application Guidelines Examples

- Generalization: AddMember and DeleteMember are shown as subtypes of Admin Command
- Aggregation: DatingSystem's relation to DatingTerminal
- Composition: DatingSystem's relation to DataBase

Generalization Acceptable?

Guideline	Admin. Com.	Add Mem.	Delete Mem.
<i>Correct</i>			
is-a		√	√
<i>Useful Subtype?</i>			
Operated on differently		√	√
<i>Useful Supertype?</i>			
Factored out common attributes	√ Both have a name field		
Factored out common associations	√ We could factor out “captured on”		

Aggregation Acceptable?

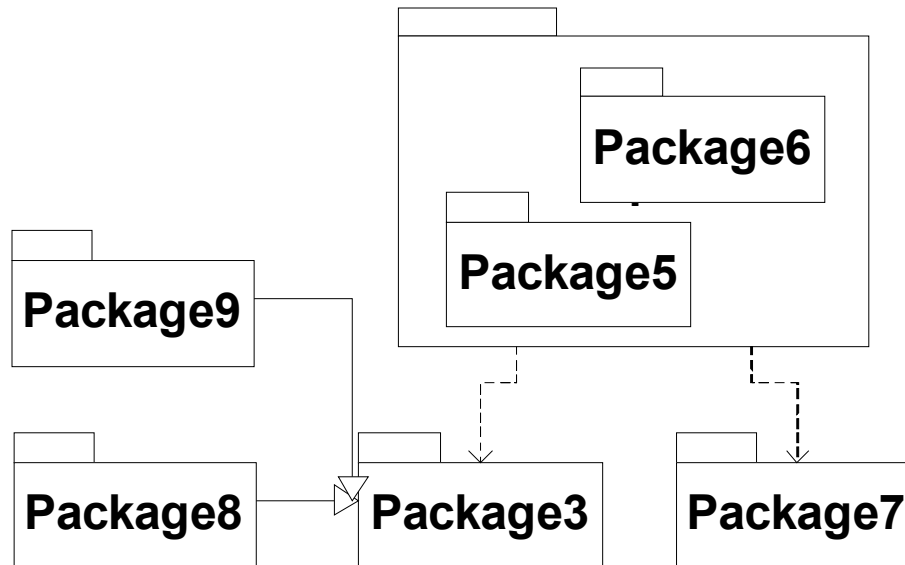
Guideline	Dating System	Data Base
	DS Terminal	MemberData
<i>Correct</i>		
has-a	√	√ (composition)
<i>Useful</i>		
Same lifetime	×	√
Whole-part	√	√
Propagated properties	×	×
Propagated operations	×	×

UML Package Diagrams – Package Relationships

- *Containment* In addition to classes, a package may contain other packages
- *Dependencies* If one package is dependent on others, changes to their classes/packages may require changes to it also
- *Generalization* A “subtype” package must conform to the interface for the more general package.

Relationship Notation

- Containment, Dependency, Generalization



DS Package Examples

- Containment: e.g. GUI package contains separate packages for the Member and Admin GUI Classes
- Dependencies: changes in one package cause changes in a depending class e.g. changes to classes in DL package/subsystem may require changes to classes in GUI package/subsystem
- Generalization: e.g. original DB has a proxy interface that can be implemented in different ways, with different classes of packages

UML Activity Diagrams

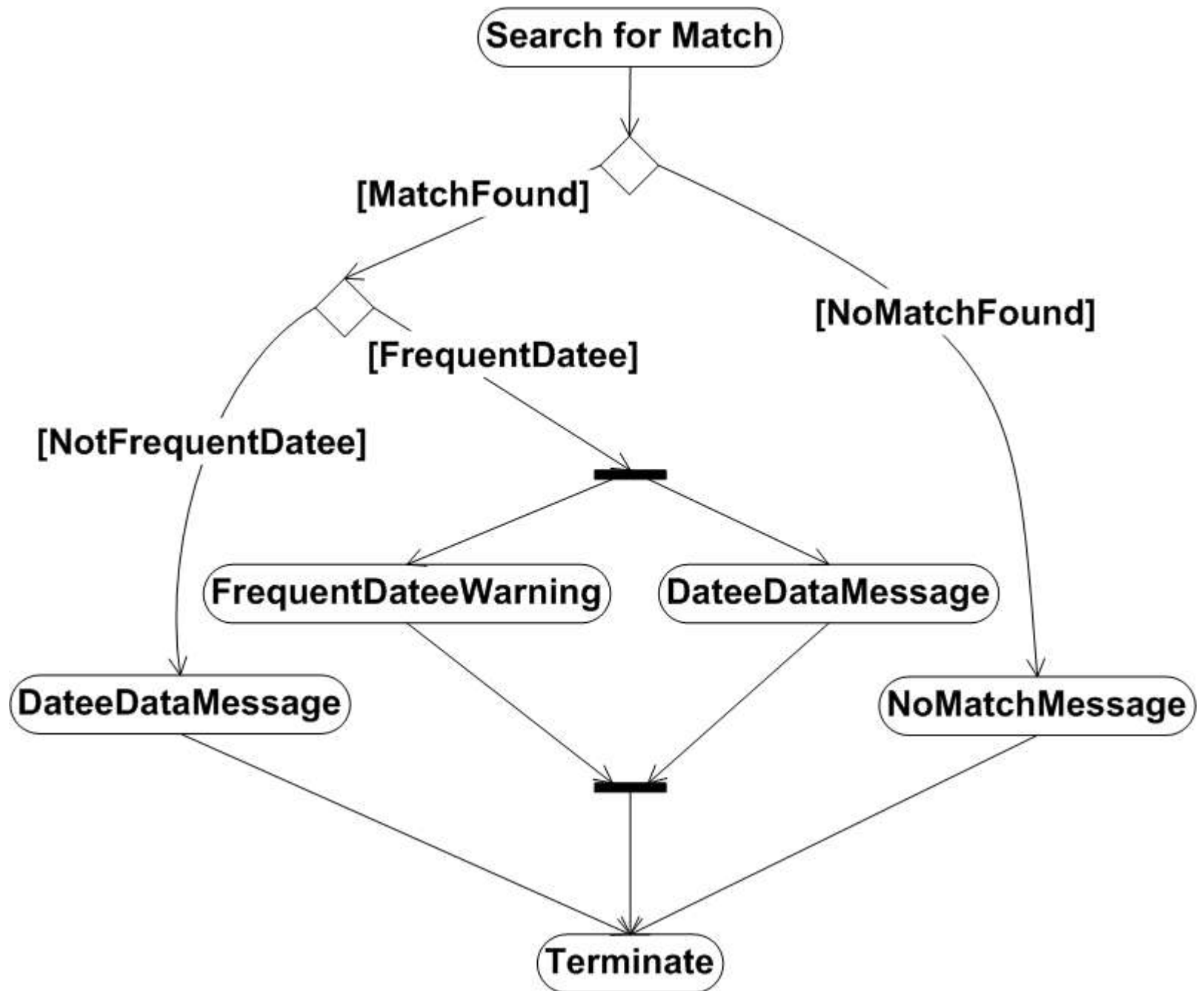
- Similar to flow charts but also allow parallel processes
- Components
 - Activity: task that needs to be done
 - Sequence flow: from one task to the next
 - Synchronization bar: splitting into and merging back for parallel flows
 - Decision: conditional branch

Activity Diagram Applications

- Use case documentation
- Activities whose sequencing is not determined
- Describing development process workflows

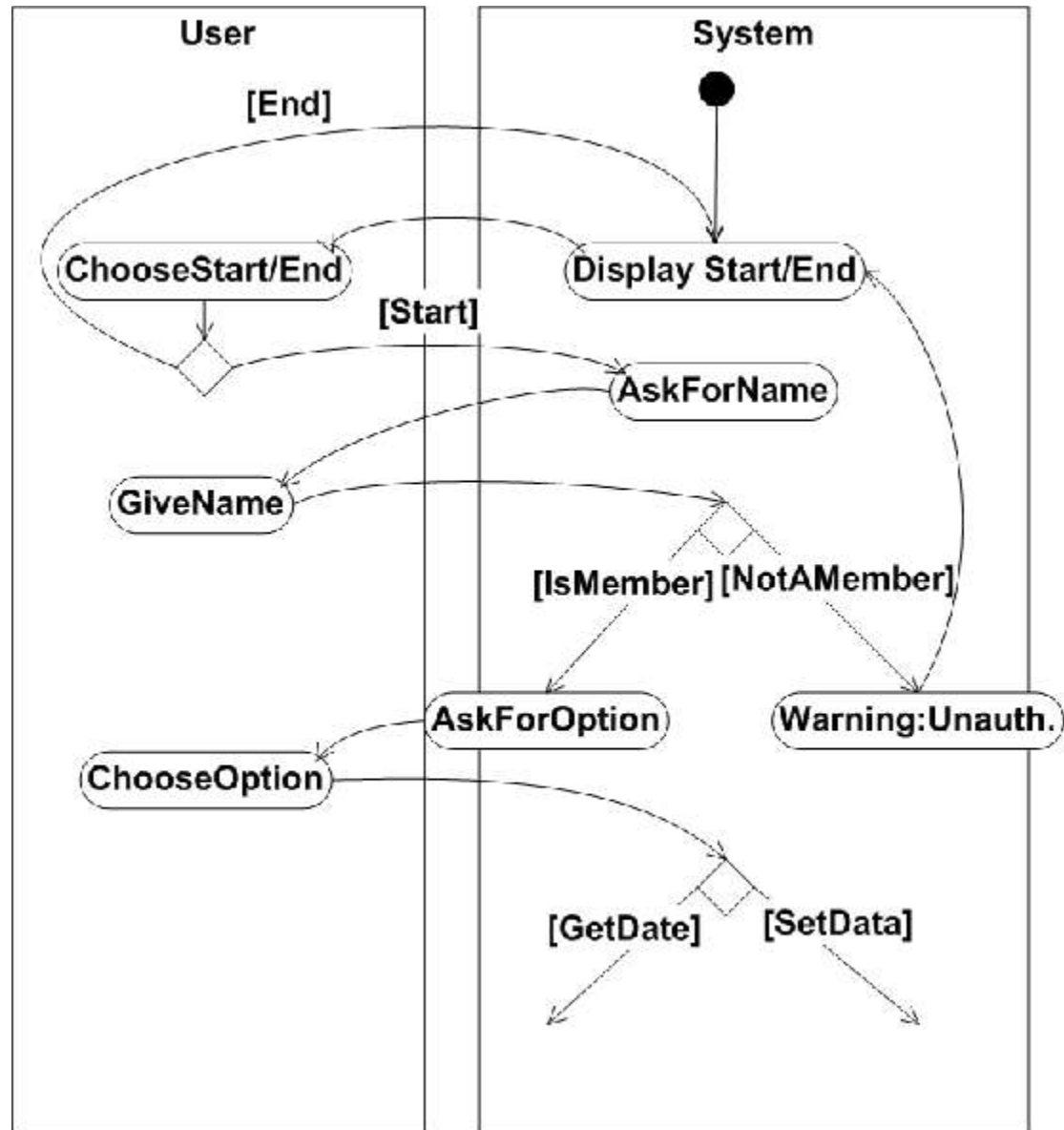
DS Activity Diagram Example

- Describes what the system must do when the user asks for a date
- High level abstraction before deciding how it will do these tasks
- E.g. FrequentDatee Warning and DateeDataMessage may be combined



OO Design and Activity Diagrams

- Activity diagrams are not OO oriented
- Compare with Interaction Sequence Diagrams
 - Focus on objects and messages between them
- Swimlanes: additional notation that can be used to make activity diagrams more OO
 - Divide up diagram into zones/columns associated with party responsible for zone tasks
 - E.g. example from DS



Activity Diagrams – Evaluation

- Used as flow charts for algorithms, with or without use of parallel processing
- For system level, useful if there are interacting parallel processes
- Not that effective for system level object oriented descriptions with non-parallel processing, as in previous example

UML Deployment Diagrams

- Purpose: show how software is distributed amongst hardware, where it runs
- Diagram components
 - Node: hardware of some kind, computer, sensor
 - Component: software modules/packages
 - Connections: communication paths between nodes
- E.g. Client-Server version of DS

