Design 1

- Static Design.
  - Design Class Diagram.
  - Architecture Model.

1

Differences Between Analysis and Design

Analysis	**Design**
What	How
Problem Space	Solution Space
Few Choices	Many Choices

2

Design Details

- Control strategy.
- Connection to external subsystems (UI, persistence, secondary actors).
- Performance.
- Boundary conditions (e.g., errors).
- Concurrency.
- Language details (e.g., single inheritance, Java interfaces).
- Hardware and OS details.

3

Design Class Diagram (1 of 3)

- Most of the classes in the Analysis class diagram can be found in the Design class diagram as well.
- The next few pages list some of the differences between the analysis and the design class diagrams:

4

Design Class Diagram (2 of 3)

- We may introduce classes into the design:
  - controllers.
  - Objects to implement associations and aggregations.
  - Objects for performance, persistence, concurrency, interface to other subsystems, etc.
  - Collections for finding objects based on a key value (that comes from the UI, for example).

5

Design Class Diagram (3 of 3)

- Some of the classes in the analysis class diagram may not show up in the design.
  - They may be outside the system.
  - The may become attributes of other objects because they don’t have enough behavior of their own.
  - What looked like inheritance at analysis time might be better implemented another way (e.g., as flag attributes) in the design.

6

Creating the Design Class Diagram (1 of 3)

- Even though we use much of the same notation in the design class diagram that we did in the analysis model, it is a good idea to start fresh with the design class diagram rather than to try to evolve the analysis class diagram.
- Start with a clean drawing surface.

7

Creating the Design Class Diagram (2 of 3)

- You probably will need to develop the dynamic model at the same time as you are working on the static model. This is an iterative process. Neither part can be done in isolation.
- As we identify classes that belong in the design, we add them to the class diagram.

8

Creating the Design Class Diagram (3 of 3)

- Start with the classes in the analysis model.
  - Eliminate any objects that are simply attributes of another object.
  - Eliminate any objects that are outside the system.
  - Split any really complicated classes into multiple (more cohesive) parts.
  - Introduce coordinator and controller objects (needed by the dynamic models), depending on the control strategy selected.

9

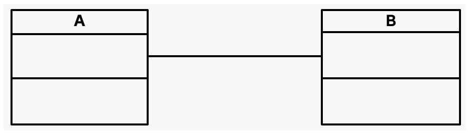
Implementing Relationships

- In the analysis class diagram, associations, aggregations and compositions represented “remembered” information.
- In the design, *all* information is stored as attributes.
- Thus, if we need to remember that object instances are linked, these links must be implemented as stored object references.
- In UML, we represent an object reference by an arrow.
- Use the dynamic models to determine which direction the reference points.

10

Object Reference Options

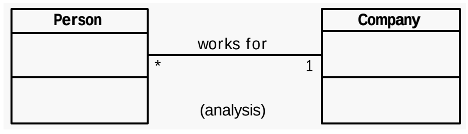
- Given an association in the analysis model, there are three basic choices to implement this in the design model.
  - A points to B.
  - B points to A.
  - Another object keeps track of the linkages.

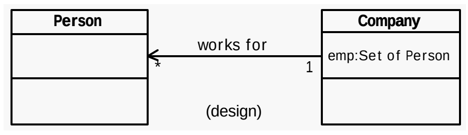


11

Implementing Multiplicities

- Multiplicity greater than one may require collection of references.

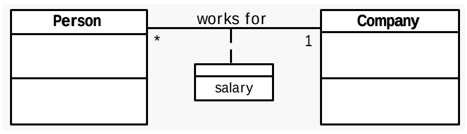




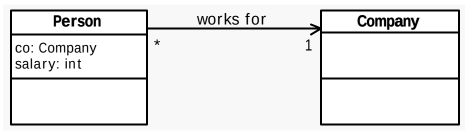
12

Implementing Association Attributes

- Whichever class stores the reference must also store the association attribute.
  - Analysis



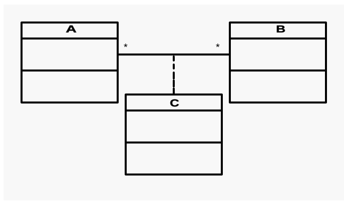
  - Design



13

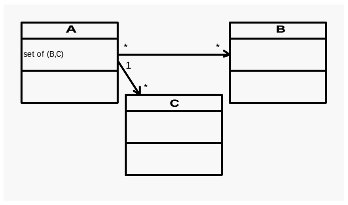
Implementing Association Classes (1 of 2)

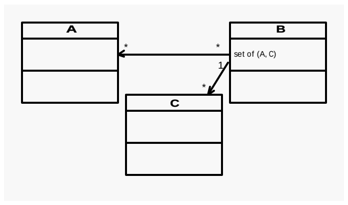
- There is a 1:1 correspondence between the following:
  - *Links* between object instances of classes A and B, and
  - Object instances of class C.
- This constraint must be managed by whichever object stores the link.

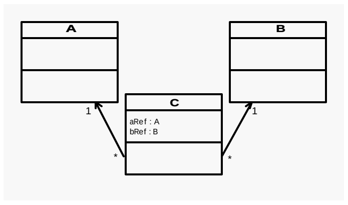


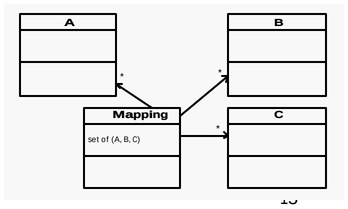
14

Implementing Association Classes (2 of 2)



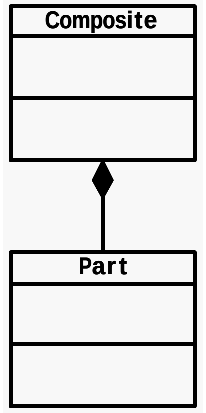






Implementing Compositions

- Recall the constraints from the analysis model.
  –
  –
  –
  –
- How should this be implemented?



16

Managing Communication With Parts (1 of 3)

- The aggregate is in charge of all communication with its parts.
- Suppose a client object needs to send a message to one or more of the parts within an aggregation.
- Two main schemes for managing this:
  - Opaque (usually for composition).
  - Transparent (usually for aggregation).

17

Managing Communication With Parts (2 of 3)

- Opaque communication:
  - The aggregator completely encapsulates the parts.
  - The client sends messages to the aggregator, which, in turn, sends messages to the parts.
  - The aggregator must have operations in its protocol to allow for this.

18

Managing Communication With Parts (3 of 3)

- Transparent communication:
  - The aggregator provides *get* and *put* operations.
    - *In situ* – the element remains in the collection; its reference is returned to the client.
    - Check-in, check-out – the part is removed from the collection, and its reference is returned to the client.

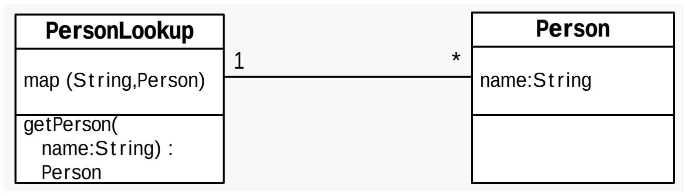
19

Introducing Lookup Objects (1 of 2)

- The input that comes in from the user interface usually consists of strings.
- If the string is a key value that refers to an object instance, we must map it to the object reference.
- We need a (singleton) lookup object.

20

Introducing Lookup Objects (2 of 2)



21

Implementing Explicit Constraints

- Check before updating state.
- Check after updating state.
- Defer to superclass or embedded class.
- Punt (publish restrictions on methods, and provide operations to check for these conditions).

22

Optimization

- Usually driven by dynamic model and the non-functional requirements.
- Some optimization techniques:
  - Redundant references.
  - Caching proxies.

23

Redundant Information

- Associations between two classes.
  - If A needs to link to B, but B also needs to link to A, then they may point to each other.
    - Watch out if the links are highly mutable.
- Derived Data.
- Derived Associations.

24

Caching Proxies

- Cache data that is frequently computed.
  - References to other objects.
  - Computed values.
- Watch out: You have to be aware when the cached data is no longer valid (because the data in underlying servers has changed.)

25

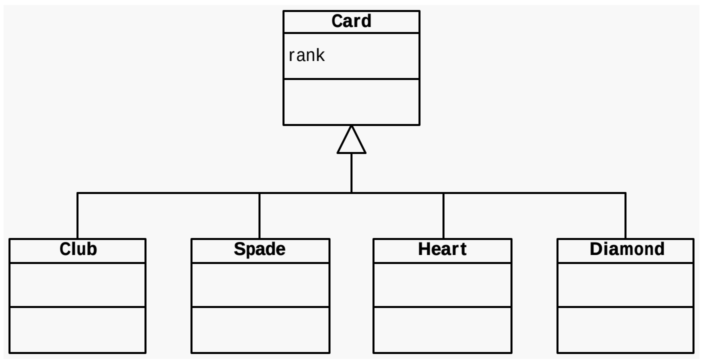
Dependency Relationships

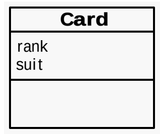
- In the dynamic model, the client object must establish visibility to the server object in order to send it a message.
- If the reference is computed (rather than stored), this is a dynamic link.
- It is indicated in UML by a dashed arrow.

26

Generalization-Specialization (1 of 3)

- Use flag attributes if there are no overridden operations.





27

Generalization-Specialization (2 of 3)

- How would you model the following?
  - There are two kinds of employees: blue collar (paid by the hour, get overtime pay), and white collar (have an annual salary, but do not get paid for overtime).
  - All employees have lots of attributes in common (name, address, hire date, etc.).
  - All employees have a *pay()* operation.
  - Blue collar employees can get promoted and become white collar employees.

28

Generalization-Specialization (3 of 3)

- Sometimes we can introduce inheritance at design time.
- During analysis, we identify classes in the problem.
- At design time, consider dividing such a class into general and specific parts.
  - The general becomes an abstract superclass.
  - The specific becomes a concrete subclass.
- More subclasses may easily be added in the future.

29

Architecture (1 of 2)

- Physical architecture:
  - Hardware, OS platform.
  - Middleware.
  - Tiered architectures.
  - Not purely a software issue.
  - Outside the scope of this course.
- Logical architecture:
  - Grouping related classes into packages.
  - Relationships among packages.

30

Architecture (2 of 2)

- Where do objects reside in the physical architecture?
- Persistence – What is the strategy for preserving the state of objects longer than the execution of the program?
- Communication – how do objects in different platforms communicate?
- Synchronization – What if an object receives messages from multiple clients at the same time?

31

Packages

- In UML, a package is a collection of related classes.
- There is tighter coupling among the classes in a packages than there is between classes in separate packages.
- A package should be cohesive.
- Different packages may reside on different platforms.
- Usually, all classes within a package reside on the same platform.

32

Package Relationships

- The relationships are actually between *classes* in the packages.
- All relationships are (directed) dependencies:
  - Static dependencies (stored references).
  - Dynamic dependencies (client-server calls).
  - Inheritance.
- Dependencies are not transitive.
  - If package A depends on package B, and B depends on C, A does not necessarily depend on C.
- Try to avoid cyclic dependencies among packages.

33

Minimizing Coupling

- Proxy pattern.
- Façade Pattern.

34