

## Deeper A

- Why, if we decided we needed a class, would we choose to make it private?
  - (a) *so we don't have to worry about maintaining the stability of its interfaces.*
  - (b) *private partnerships pose less of a maintainability problem than public ones, because the number of partners is small and they are all centralized in one place.*

10/14/2012

Component Design

35

## Deeper A

- Why would I use routines to encapsulate global data structures?
  - To reduce the number of pieces of code that actually touch the global data, and thus the potential for interaction side-effects ... which are a major problem with global data.*
  - To provide a single portal through which all operations are forced to go.*
  - To ensure that all updates and references are correct, honoring all appropriate data and transactional assertions.*

10/14/2012

Component Design

36

## Deeper A

- Why would I want to ensure there was only one copy of a commonly used code sequence?
  - (a) *to simplify the use of that sequence by the routines that need it.*
  - (b) *to encapsulate the details of that code sequence.*
  - (c) *to make it possible to maintain only a single copy of that code sequence.*
  - (d) *to make it possible to instrument or intercept that code sequence.*

10/14/2012

Component Design

37

## Deeper B

- What makes pseudo-code prose a good form for representing a design?
  - When the flow of control is not complex and the primary focus is on the activities to be performed, indented pseudo-code (with its easily adaptable level of abstraction) is fast to type, easily understood, and easily maintained.*

10/14/2012

Component Design

38

## Deeper B

- What makes UML activity or state diagrams a good form for representing designs?
  - Graphical representations are often more quickly understood than textual ones.*
  - UML state diagrams are an obvious representation for designs that easily fit into a state/transition model.*
  - UML activity diagrams may be a more obvious representation for parallel and alternative code paths than indented pseudo-code.*

10/14/2012

Component Design

39

## Deeper B

- Why would we choose to represent a design in a tabular form?
  - When a key element of the design is the enumeration of cases to be handled, and the specification of correct behavior for each case, a table of cases may be the simplest, clearest, and most easily maintainable form for representing that description.*
  - It is easy to see what cases are and are not included, and to compare the handling of related cases.*

10/14/2012

Component Design

40

## Deeper

- Why would we choose to represent a design in some formal language?

*The most likely reason would be that we had access to design automation tools (e.g. browsers, validation suites, code generators, run time checkers, test case generators, etc) that worked off of that language.*

*The added power of such tools often more than compensates for the added complexity of writing designs in a more formal language.*

10/14/2012

Component Design

41

## Useful Concepts

- What is a synchronous interaction

*One where the sender waits until the recipient finishes processing the request. Remote procedure calls are the most common form of a synchronous interaction.*

- What is an asynchronous interaction

*One where the sender sends a message to the recipient, and then continues processing with no further interest in what the recipient might decide to do about the message.*

10/14/2012

Component Design

42

## Useful Concepts

- What is meant by thread creation and destruction?

*This is a popular concept in client/server computing. It refers to an independent thread of execution (something like a separately executing program) that is created just to process a request, and ends as soon as the request completes.*

*There are actually a great many different thread models. Take my OS class if you would like to learn more about these.*

10/14/2012

Component Design

43

## Deeper

- Why would you use a swim-lane vs. an interaction diagram?

*To show not only interactions, but the computations that drive them.*

- Why would you use a swim-lane vs. an activity diagram?

*If the algorithm of interest would be executed in distinct objects/threads/hosts.*

*To show the interactions of the algorithms running in the different objects/threads.*

10/14/2012

Component Design

44

## Deeper

- Why is it awkward to describe a data flow model in UML?

*UML was designed to describe OO systems. It is a language for describing hierarchies of objects, algorithm to implement their methods, and messages to mediate their interactions.*

*Data flow models follow data streams through a succession of transformations and routings.*

*These represent very different perspectives.*

10/14/2012

Component Design

45

## Answer

- Choose the model that is best suited to the aspects of the system we are trying to describe:

- user/system or inter-component interactions
- algorithms or distributed computations
- system state transitions
- data transformations

- You don't have to choose one

- You can use different diagrams to describe different aspects of system behavior

10/14/2012

Component Design

46

## Deeper ☹️

- Why are UML interaction diagrams poorly suited for expressing algorithms?

*Basic UML interaction diagram do not seem to have a reasonable way of describing decisions or iteration.*

*Actually, UML does define ways of representing iteration and decision in an interaction diagram ... but it is painfully awkward.*

*This problem is solved by UML activity diagrams.*