Requirements Specification

- Due Feb 29, 12PM EST
- Submit 1 per team on moodle

1

Architecture

MIT Stata Center by Frank Gehry

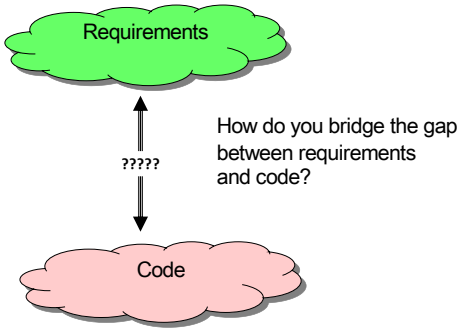
2

Why architecture?

"Good software architecture makes the rest of the project easy."
Steve McConnell, Survival Guide

3

The basic problem

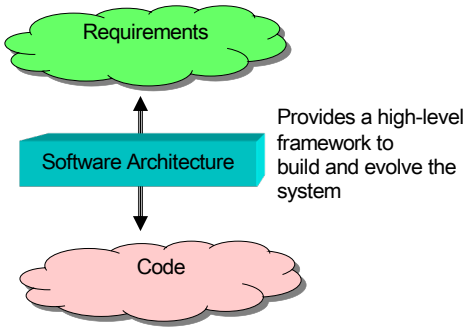


4

One answer

5

A better answer

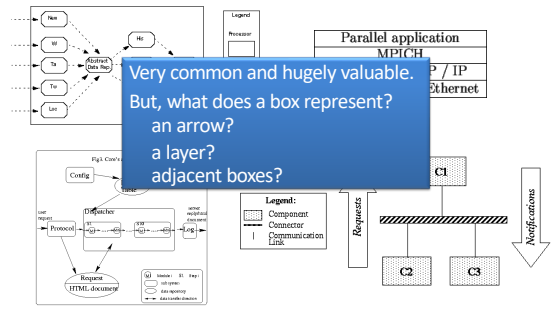


6

What does an architecture look like?

7

Box-and-arrow diagrams



Very common and hugely valuable.

But, what does a box represent?
- an arrow?
- a layer?
- adjacent boxes?

8

An architecture: components and connectors

- *Components* define the basic computations comprising the system and their behaviors
  - abstract data types, filters, etc.
- *Connectors* define the interconnections between components
  - procedure call, event announcement, asynchronous message sends, etc.
- The line between them may be fuzzy at times
  - Ex: A connector might (de)serialize data, but can it perform other, richer computations?

9

A good architecture

- Satisfies functional and performance requirements
- Manages complexity
- Accommodates future change
- Is concerned with
  - reliability, safety, understandability, compatibility, robustness, …

10

10

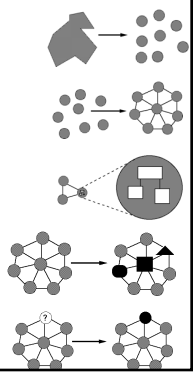
Divide and conquer

- Benefits of decomposition:
  - Decrease size of tasks
  - Support independent testing and analysis
  - Separate work assignments
  - Ease understanding
- Use of abstraction leads to modularity
  - Implementation techniques: information hiding, interfaces
- To achieve modularity, you need:
  - Strong cohesion within a component
  - Loose coupling between components
  - And these properties should be true at each level

11

Qualities of modular software

- decomposable
  - can be broken down into pieces
- composable
  - pieces are useful and can be combined
- understandable
  - one piece can be examined in isolation
- has continuity
  - change in reqs affects few modules
- protected / safe
  - an error affects few other modules



12

12

Interface and implementation

- **public interface**: data and behavior of the object that can be seen and executed externally by "client" code
- **private implementation**: internal data and methods in the object, used to help implement the public interface, but cannot be directly accessed
- **client**: code that uses your class/subsystem

Example: *radio*

- public interface: the speaker, volume buttons, station dial
- private implementation: the guts of the radio; the transistors, capacitors, voltage readings, frequencies, etc. that user should not see

13

13

UML diagrams

- UML = universal modeling language
- A standardized way to describe (draw) architecture
- Widely used in industry

14

Properties of architecture

- Coupling
- Cohesion
- Style conformity
- Matching
- Errosion

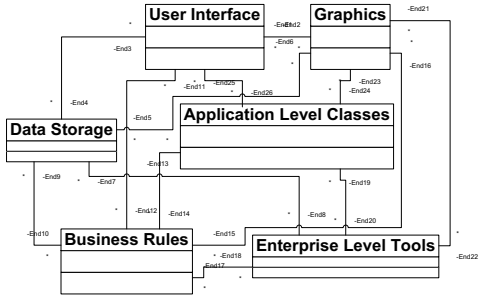
15

Loose coupling

- *coupling* assesses the kind and quantity of interconnections among modules
- Modules that are loosely coupled (or uncoupled) are better than those that are tightly coupled
- The more tightly coupled two modules are, the harder it is to work with them separately

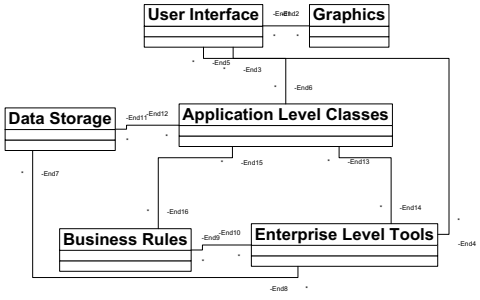
16

Tightly or loosely coupled?



17

Tightly or loosely coupled?



18

Strong cohesion

- *cohesion* refers to how closely the operations in a module are related
- Tight relationships improve clarity and understanding
- Classes with good abstraction usually have strong cohension
- No schizophrenic classes!

19

Strong or weak cohesion?

```
class Employee {

public:
  …
  FullName GetName() const;
  Address GetAddress() const;
  PhoneNumber GetWorkPhone() const;
  …
  bool IsJobClassificationValid(JobClassification jobClass);
  bool IsZipCodeValid (Address address);
  bool IsPhoneNumberValid (PhoneNumber phoneNumber);
  …
  SqlQuery GetQueryToCreateNewEmployee() const;
  SqlQuery GetQueryToModifyEmployee() const;
  SqlQuery GetQueryToRetrieveEmployee() const;
  …
}
```

20

An architecture helps with

- System understanding: interactions between modules
- Reuse: high-level view shows opportunity for reuse
- Construction: breaks development down into work items; provides a path from requirements to code
- Evolution: high-level view shows evolution path
- Management: helps understand work items and track progress
- Communication: provides vocabulary; pictures say 10^3 words

21

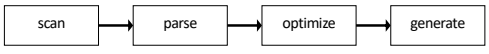
Architectural style

- Defines the vocabulary of components and connectors for a family (style)
- Constraints on the elements and their combination
  - Topological constraints (no cycles, register/announce relationships, etc.)
  - Execution constraints (timing, etc.)
- By choosing a style, one gets all the known properties of that style (for any architecture in that style)
  - Ex: performance, lack of deadlock, ease of making particular classes of changes, etc.

22

Styles are not just boxes and arrows

- Consider pipes & filters, for example (Garlan and Shaw)
  - Pipes must compute local transformations
  - Filters must not share state with other filters
  - There must be no cycles
- If these constraints are violated, it's not a pipe & filter system
  - One can't tell this from a picture
  - One can formalize these constraints



23

The design and the reality

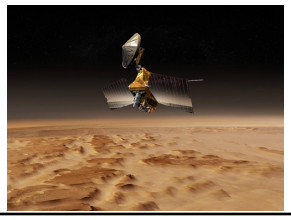
- The code is often less clean than the design
- The design is still useful
  - communication among team members
  - selected deviations can be explained more concisely and with clearer reasoning

24

Architectural mismatch

- Mars orbiter loss

  NASA lost a 125 million Mars orbiter because one engineering team used metric units while another used English units for a key spacecraft operation



25

Views

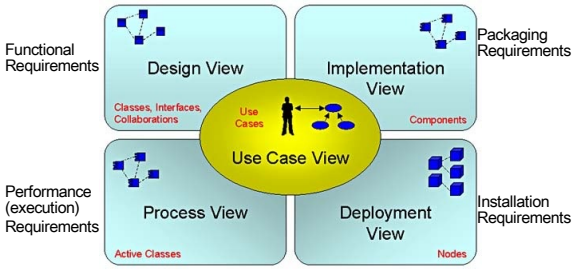
A **view** illuminates a set of top-level design decisions

- how the system is composed of interacting parts
- where are the main pathways of interaction
- key properties of the parts
- information to allow high-level analysis and appraisal

27

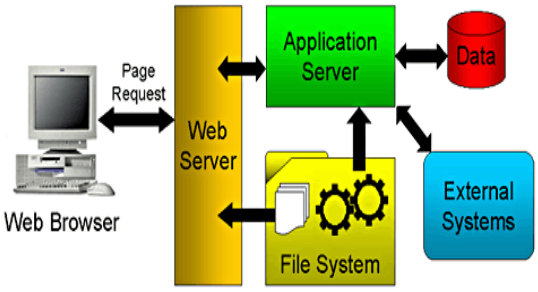
Importance of views

Multiple views are needed to understand the different dimensions of systems



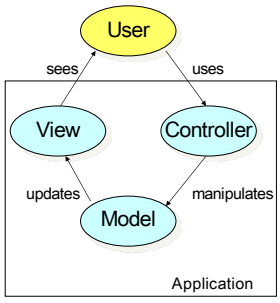
28

Web application (client-server)



29

Model-View-Controller

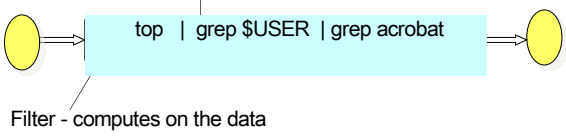


Separates the application object (model) from the way it is represented to the user (view) from the way in which the user controls it (controller).

30

Pipe and filter

Pipe – passes the data



Filter - computes on the data

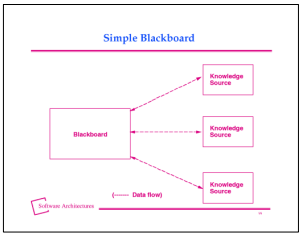
Each stage of the pipeline acts independently of the others.

Can you think of a system based on this architecture?

31

Blackboard architectures

- *The knowledge sources*: separate, independent units of application dependent knowledge. No direct interaction among knowledge sources
- *The blackboard data structure*: problem-solving state data. Knowledge sources make changes to the blackboard that lead incrementally to a solution to the problem.
- *Control*: driven entirely by state of blackboard. Knowledge sources respond opportunistically to changes in the blackboard.

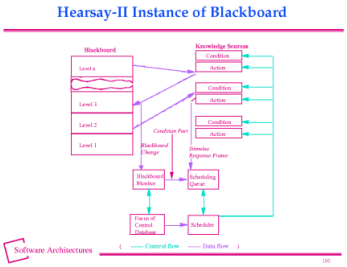


Blackboard systems have traditionally been used for applications requiring complex interpretations of signal processing, such as speech and pattern recognition.

32

32

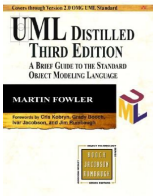
Hearsay-II: blackboard



33

33

UML



Design and UML Diagrams

34

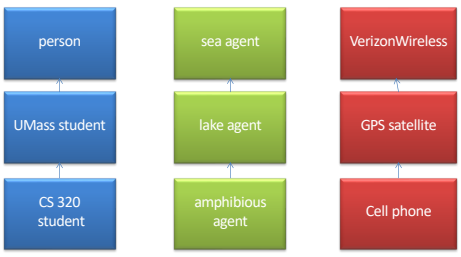
34

How do people draw / write down software architecture?

35

35

Example architectures



36

36

Big questions

- What is UML?
  - Why should I bother? Do people really use UML?
- What is a UML class diagram?
  - What kind of information goes into it?
  - How do I create it?
  - When should I create it?

37

37

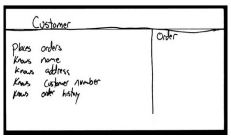
Design phase

- **design**: specifying the structure of how a software system will be written and function, without actually writing the complete implementation
- a transition from "what" the system must do, to "how" the system will do it
  - What classes will we need to implement a system that meets our requirements?
  - What fields and methods will each class have?
  - How will the classes interact with each other?

38

How do we design classes?

- class identification from project spec / requirements
  - nouns are potential classes, objects, fields
  - verbs are potential methods or responsibilities of a class
- CRC card exercises
  - write down classes' names on index cards
  - next to each class, list the following:
    - **responsibilities**: problems to be solved; short verb phrases
    - **collaborators**: other classes that are sent messages by this class (asymmetric)
- UML diagrams
  - class diagrams
  - sequence diagrams
  - ...



39

UML

In an effort to promote Object Oriented designs, three leading object oriented programming researchers joined ranks to combine their languages:

- Grady Booch (BOOCH)
- Jim Rumbaugh (OML: object modeling technique)
- Ivar Jacobsen (OOSE: object oriented software eng)

and come up with an industry standard [mid 1990's].

40

UML – Unified Modeling Language

- The result is large (as one might expect)
  - Union of all Modeling Languages
    - Use case diagrams
    - Class diagrams
    - Object diagrams
    - Sequence diagrams
    - Collaboration diagrams
    - Statechart diagrams
    - Activity diagrams
    - Component diagrams
    - Deployment diagrams
    - ....
  - But it's a nice standard that has been embraced by the industry.

41

Introduction to UML

- UML: pictures of an OO system
  - programming languages are not abstract enough for OO design
  - UML is an open standard; lots of companies use it
- What is legal UML?
  - a *descriptive* language: rigid formal syntax (like programming)
  - a *prescriptive* language: shaped by usage and convention
  - it's okay to omit things from UML diagrams if they aren't needed by team/supervisor/instructor

42

Uses for UML

- as a sketch: to communicate aspects of system
  - forward design: doing UML before coding
  - backward design: doing UML after coding as documentation
  - often done on whiteboard or paper
  - used to get rough selective ideas
- as a blueprint: a complete design to be implemented
  - sometimes done with CASE (Computer-Aided Software Engineering) tools
- as a programming language: with the right tools, code can be auto-generated and executed from UML
  - only good if this is faster than coding in a "real" language

43

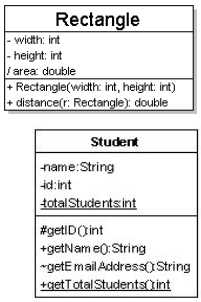
UML class diagrams

- What is a UML class diagram?
- **UML class diagram**: a picture of
  - the classes in an OO system
  - their fields and methods
  - connections between the classes
    - that interact or inherit from each other
- What are some things that are not represented in a UML class diagram?
- details of how the classes interact with each other
- algorithmic details
- how a particular behavior is implemented

44

Diagram of one class

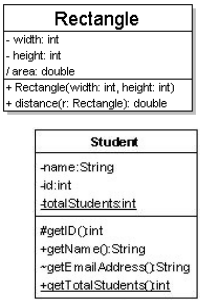
- class name in top of box
  - write <<interface>> on top of interfaces' names
  - use *italics* for an *abstract class* name
- attributes (optional)
  - should include all fields of the object
- operations / methods (optional)
  - may omit trivial (get/set) methods
    - but don't omit any methods from an interface!
  - should not include inherited methods



45

Class attributes

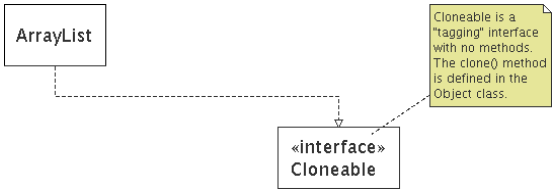
- attributes (fields, instance variables)
  - *visibility name* : *type* [*count*] = *default_value*
  - visibility: + public, # protected, - private, ~ package (default), / derived
  - underline static attributes
  - **derived attribute**: not stored, but can be computed from other attribute values
  - attribute example: - balance : double = 0.00



46

Comments

- represented as a folded note, attached to the appropriate class/method/etc by a dashed line



47

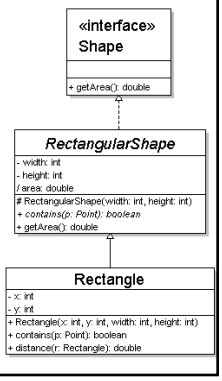
Relationships between classes

- **generalization**: an inheritance relationship
  - inheritance between classes
  - interface implementation
- **association**: a usage relationship
  - dependency
  - aggregation
  - composition

48

Generalization relationships

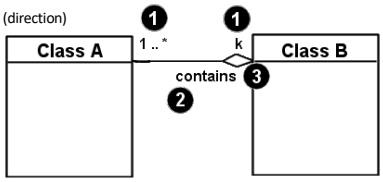
- generalization (inheritance) relationships
  - hierarchies drawn top-down with arrows pointing upward to parent
  - line/arrow styles differ, based on whether parent is a(n):
    - class: solid line, black arrow
    - abstract class: solid line, white arrow
    - interface: dashed line, white arrow
  - we often don't draw trivial / obvious generalization relationships, such as drawing the Object class as a parent



49

Associational relationships

- associational (usage) relationships
  1. multiplicity (how many are used)
     - * ⇒ 0, 1, or more
     - 1 ⇒ 1 exactly
     - 2..4 ⇒ between 2 and 4, inclusive
     - 3..* ⇒ 3 or more
  2. name (what relationship the objects have)
  3. navigability (direction)

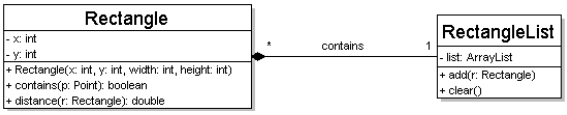


50

Multiplicity of associations

- one-to-one
  - each student must carry exactly one ID card

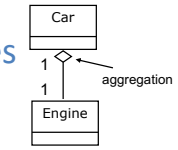
- one-to-many
  - one rectangle list can contain many rectangles

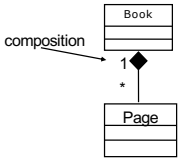


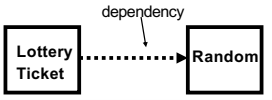
51

Association types

- **aggregation**: "is part of"
  - symbolized by a clear white diamond
- **composition**: "is entirely made of"
  - stronger version of aggregation
  - the parts live and die with the whole
  - symbolized by a black diamond
- **dependency**: "uses temporarily"
  - symbolized by dotted line
  - often is an implementation detail, not an intrinsic part of that object's state

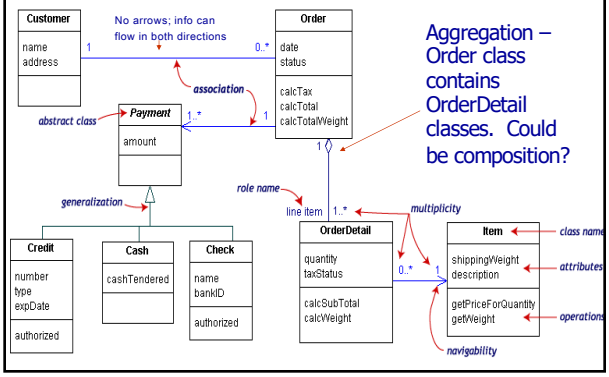






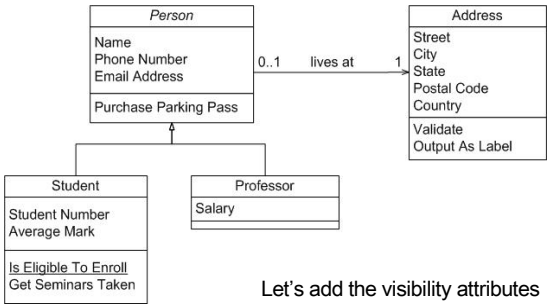
52

Class diagram example



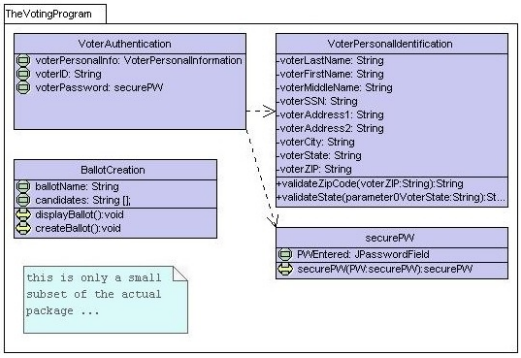
53

UML example: people



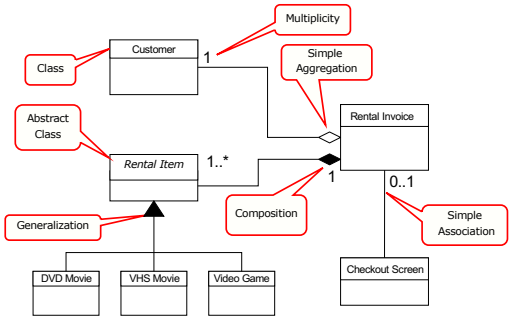
54

Class diagram: voters



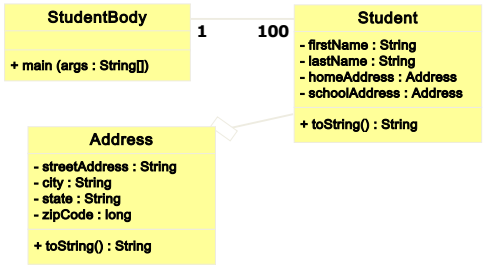
55

Class diagram example: video store



56

Class diagram example: student



57

Tools for creating UML diagrams

- Violet (free)
  - http://horstmann.com/violet/
- Rational Rose (trial)
  - https://www.ibm.com/support/pages/node/3064772?mhsrc=ibmsearch_a&mhq=rational%20rose
- Visual Paradigm UML Suite (trial)
  - http://www.visual-paradigm.com/
  - direct download link: https://www.visual-paradigm.com/download/

(there are many others, but many are commercial)

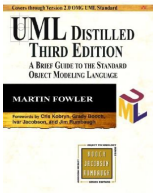
58

Class design exercise

- Consider this Texas Hold 'em poker game system:
  - 2 to 8 human or computer players
  - Each player has a name and stack of chips
  - Computer players have a difficulty setting: easy, medium, hard
  - Summary of each round:
    - Dealer collects ante from appropriate players, shuffles the deck, and deals each player a hand of 2 cards from the deck.
    - A betting round occurs, followed by dealing 3 shared cards from the deck.
    - As shared cards are dealt, more betting rounds occur, where each player can fold, check, or raise.
    - At the end of a round, if more than one player is remaining, players' hands are compared, and the best hand wins the pot of all chips bet so far.
  - What classes are in this system?
    What are their responsibilities?
    Which classes collaborate?
  - Draw a class diagram for this system. Include relationships between classes (generalization and associational).

59

UML



UML Sequence Diagrams

60

UML sequence diagrams

- **sequence diagram**: an "interaction diagram" that models a single scenario executing in the system
  - perhaps 2nd most used UML diagram (behind class diagram)
- relation of UML diagrams to other exercises:
  - CRC cards -> class diagram
  - use cases -> sequence diagrams

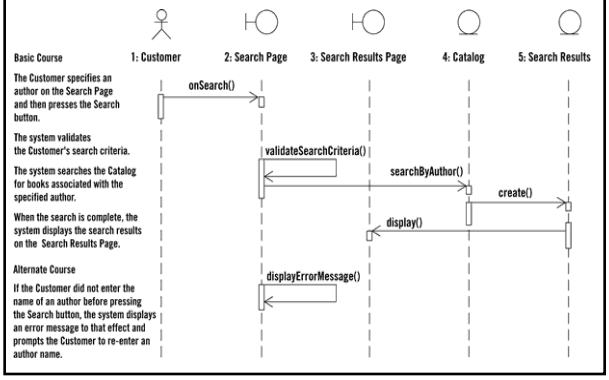
61

Key parts of a sequence diagram

- **participant**: an object or an entity; the sequence diagram actor
  - sequence diagram starts with an unattached "found message" arrow
- **message**: communication between objects
- the axes in a sequence diagram:
  - horizontal: which object/participant is acting
  - vertical: time (↓ forward in time)

62

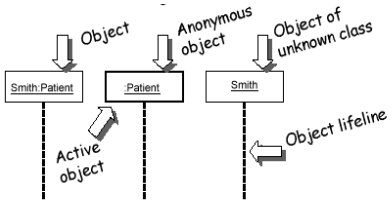
Sequence diagram from use case



63

Representing objects

- An object: a square with object type, optionally preceded by object name and colon
  - write object's name if it clarifies the diagram
  - object's "life line" represented by dashed vert. line

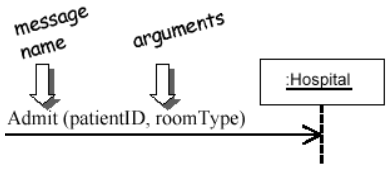


Name syntax: <objectname>:<classname>

64

Messages between objects

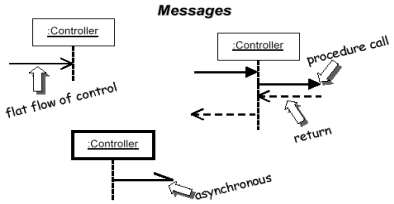
- message (method call): horizontal arrow to other object
  - write message name and arguments above arrow



65

Different types of messages

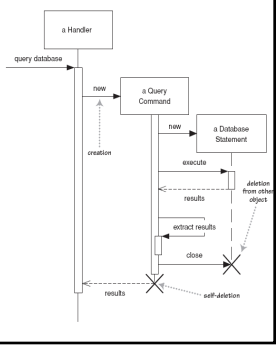
- Type of arrow indicates types of messages
  - dashed arrow back indicates return
  - different arrowheads for normal / concurrent (asynchronous) methods



66

Lifetime of objects

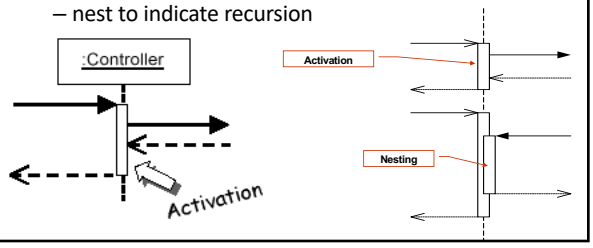
- *creation*: arrow with 'new' written above it
  - an object created after the start of the scenario appears lower than the others
- *deletion*: an X at bottom of object's lifeline
  - Java doesn't explicitly delete objects; they fall out of scope and are garbage collected



67

Indicating method calls

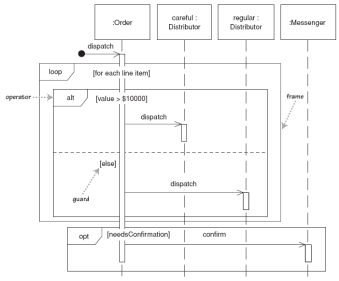
- **activation**: thick box over object's life line
  - Either: that object is running its code or it is on the stack waiting for another object's method
  - nest to indicate recursion



68

Selection and loops

- frame: box around part of a sequence diagram to indicate selection or loop
  - `if` -> (opt) [condition]
  - `if/else` -> (alt) [condition], separated by horizontal dashed line
  - loop -> (loop) [condition or items to loop over]

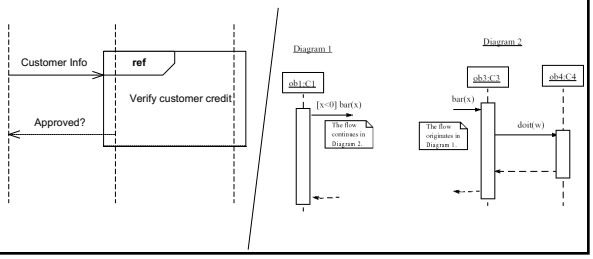


69

Linking sequence diagrams

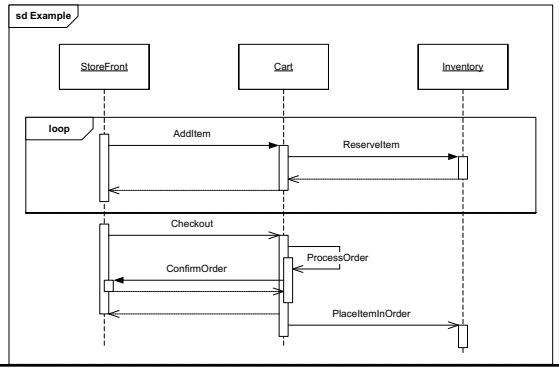
If one sequence diagram is too large or refers to another diagram:

- an unfinished arrow and comment
- a "ref" frame that names the other diagram



70

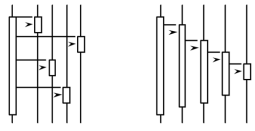
Example sequence diagram

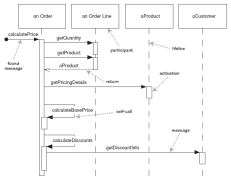


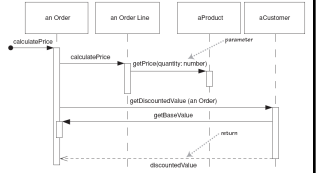
71

Forms of system control

- What can you say about the control flow of each of the following systems?
  - Is it centralized?
  - Is it distributed?







72

Why not just code it?

- Sequence diagrams can be somewhat close to the code level. So why not just code up that algorithm rather than drawing it as a sequence diagram?
  - a good sequence diagram is still a bit above the level of the real code (not all code is drawn on diagram)
  - sequence diagrams are language-agnostic (can be implemented in many different languages
  - non-coders can do sequence diagrams
  - easier to do sequence diagrams as a team
  - can see many objects/classes at a time on same page (visual bandwidth)

73

Poker sequence diagram exercise

The scenario begins when the player chooses to start a new round in the UI. The UI asks whether any new players want to join the round; if so, the new players are added using the UI.

All players' hands are emptied into the deck, which is then shuffled. The player left of the dealer supplies a blind bet of the proper amount. Next, each player is dealt a hand of two cards from the deck in a round-robin fashion; one card to each player. Then the second card.

If the player left of the dealer doesn't have enough money for his/her blind, he/she is removed from the game and the next player supplies the blind. If that player also cannot afford the blind, this cycle continues until a rich-enough player is found or all players are removed.

74

Calendar sequence diagram exercise

The user chooses to add a new appointment in the UI. The UI notices which part of the calendar is active and pops up an Add Appointment window for that date and time.

The user enters the necessary information about the appointment's name, location, start and end times. The UI will prevent the user from entering an appointment that has invalid information, such as an empty name or negative duration. The calendar records the new appointment in the user's list of appointments. Any reminder selected by the user is added to the list of reminders.

If the user already has an appointment at that time, the user is shown a warning message and asked to choose an available time or replace the previous appointment. If the user enters an appointment with the same name and duration as an existing group meeting, the calendar asks the user whether he/she intended to join that group meeting instead. If so, the user is added to that group meeting's list of participants.

75