Java Program Style and Design - Continued

- Program Design Concepts:
  - Design Patterns
  - Refinement
  - Abstractions - Program Conceptualization
    - Object-oriented programming and abstraction
  - Pseudo-code: abstraction and refinement

Today:

- JavaDoc
- Software Lifecycle
- UML - Unified Modeling Language

JavaDoc - Java API Documentation Generator

javadoc <sourcefiles.java> *

- Parses the declarations and documentation comments in a set of Java source files
- Produces a set of HTML pages describing public and protected classes, interfaces, constructors, methods, and fields.
  - One .html file for each .java file and each package
  - Class hierarchy (tree.html) & index of members (AllNames.html)
  - Includes class and member signatures.
  - You can add further documentation by including doc `/**` comments in the source code - may include html tags

```
/**
 * This is a <b>doc</b> comment.
 */
```

  - Doc comments only recognized immediately before **class**, **interface**, **constructor**, **method**, or **field** declarations.

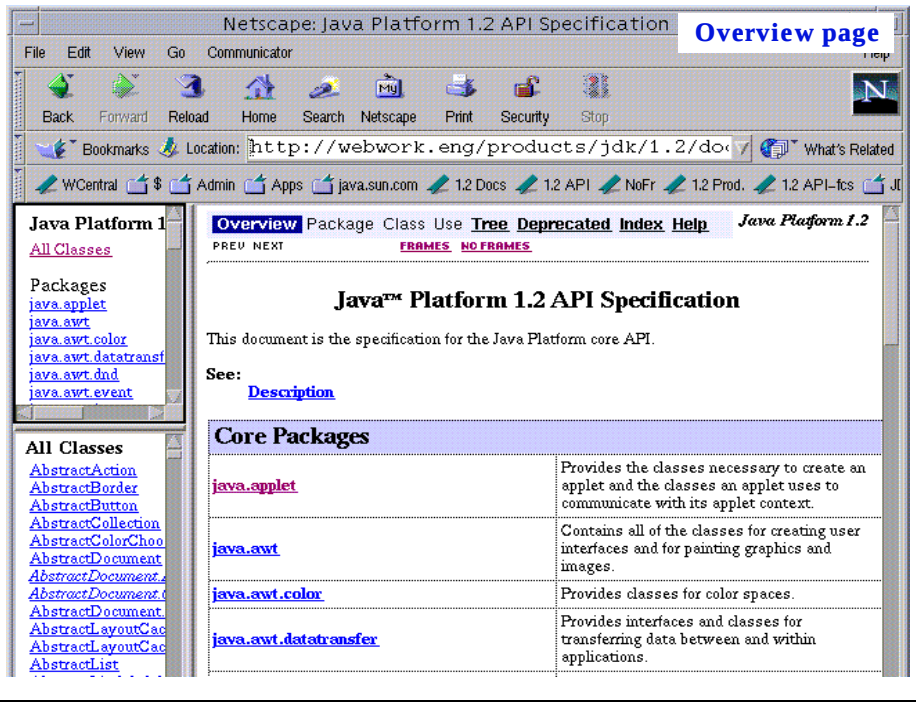
JavaDoc - 2

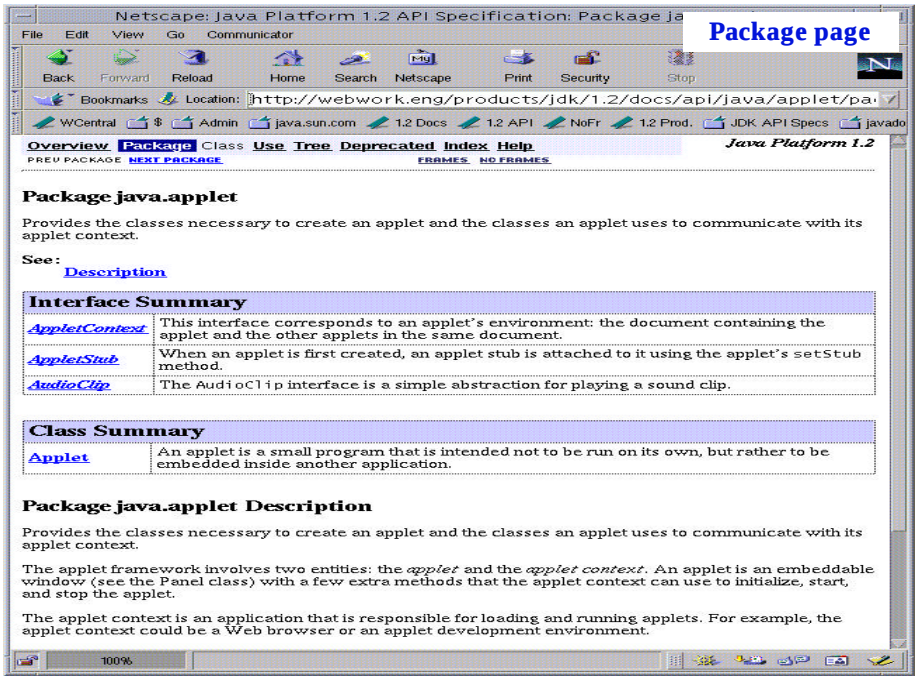
- *First* sentence of each doc comment should be a *summary* sentence - for method summary at top of html file.
- **javadoc** parses *special tags* within a Javadoc comment. These doc tags enable you to autogenerate a complete, well-formatted API from your source code. The tags start with "at" sign (@):
  - **`@param` *`parameter-name description`***
  - **`@return` *`description`***
  - **`@exception` *`fully-qualified-class-name description`***
  - **`@author` *`name-text`***
  - **`@version` *`version-text`***
  - **`@see` *`classname:`***
    `Adds a hyperlinked "See Also" entry to the class.`
- Class, method, and Field doc comments supported.
- http://java.sun.com/products/jdk/1.1/docs/tooldocs/win32/javadoc.html

An example of a method doc comment

```
/**
   * Returns the character at the specified index. An index
   * ranges from <code>0</code> to <code>length() - 1</code>.
   *
   * @param     index  the index of the desired character.
   * @return    the desired character.
   * @exception StringIndexOutOfRangeException
   *            if the index is not in the range <code>0</code>
   *            to <code>length()-1</code>.
   * @see       java.lang.Character#charValue()
   */
  public char charAt(int index) {
       ...
    }
```

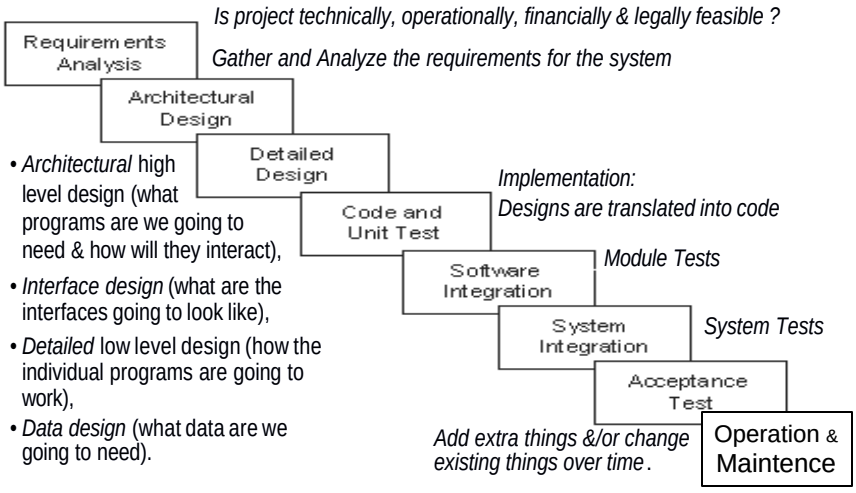
For Java API - JavaDoc *output* see:
http://java.sun.com/products/jdk/1.2/docs/api/index.html


Overview page
Netscape: Java Platform 1.2 API Specification
File Edit View Go Communicator
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://webwork.eng/products/jdk/1.2/do
What's Related
WCentral $ Admin Apps java.sun.com 1.2 Docs 1.2 API NoFr 1.2 Prod. 1.2 API-fcs
Java Platform 1
All Classes
Packages
java.applet
java.awt
java.awt.color
java.awt.datatransf
java.awt.dnd
java.awt.event
All Classes
AbstractAction
AbstractBorder
AbstractButton
AbstractCollection
AbstractColorChoo
AbstractDocument
AbstractDocument.
AbstractDocument.
AbstractDocument.
AbstractLayoutCac
AbstractLayoutCac
AbstractList
Overview Package Class Use Tree Deprecated Index Help
Java Platform 1.2
PREV NEXT
FRAMES NO FRAMES
Java™ Platform 1.2 API Specification
This document is the specification for the Java Platform core API.
See:
Description
Core Packages
java.applet
Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt
Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color
Provides classes for color spaces.
java.awt.datatransfer
Provides interfaces and classes for transferring data between and within applications.


Package page
Netscape: Java Platform 1.2 API Specification: Package ja
File Edit View Go Communicator
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://webwork.eng/products/jdk/1.2/docs/api/java/applet/pa
WCentral $ Admin java.sun.com 1.2 Docs 1.2 API NoFr 1.2 Prod. JDK API Specs javado
Overview Package Class Use Tree Deprecated Index Help
Java Platform 1.2
PREV PACKAGE NEXT PACKAGE
FRAMES NO FRAMES
Package java.applet
Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
See:
Description
Interface Summary
AppletContext
This interface corresponds to an applet's environment: the document containing the applet and the other applets in the same document.
AppletStub
When an applet is first created, an applet stub is attached to it using the applet's setStub method.
AudioClip
The AudioClip interface is a simple abstraction for playing a sound clip.
Class Summary
Applet
An applet is a small program that is intended not to be run on its own, but rather to be embedded inside another application.
Package java.applet Description
Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
The applet framework involves two entities: the applet and the applet context. An applet is an embeddable window (see the Panel class) with a few extra methods that the applet context can use to initialize, start, and stop the applet.
The applet context is an application that is responsible for loading and running applets. For example, the applet context could be a Web browser or an applet development environment.
100%

Classic 'Waterfall' Lifecycle

Waterfall model stages in the software development process are seen to cascade down from one to another.



Is project technically, operationally, financially & legally feasible ?

Gather and Analyze the requirements for the system

- *Architectural* high level design (what programs are we going to need & how will they interact),
- *Interface design* (what are the interfaces going to look like),
- *Detailed* low level design (how the individual programs are going to work),
- *Data design* (what data are we going to need).

Implementation:
Designs are translated into code

Module Tests

System Tests

Add extra things &/or change existing things over time.

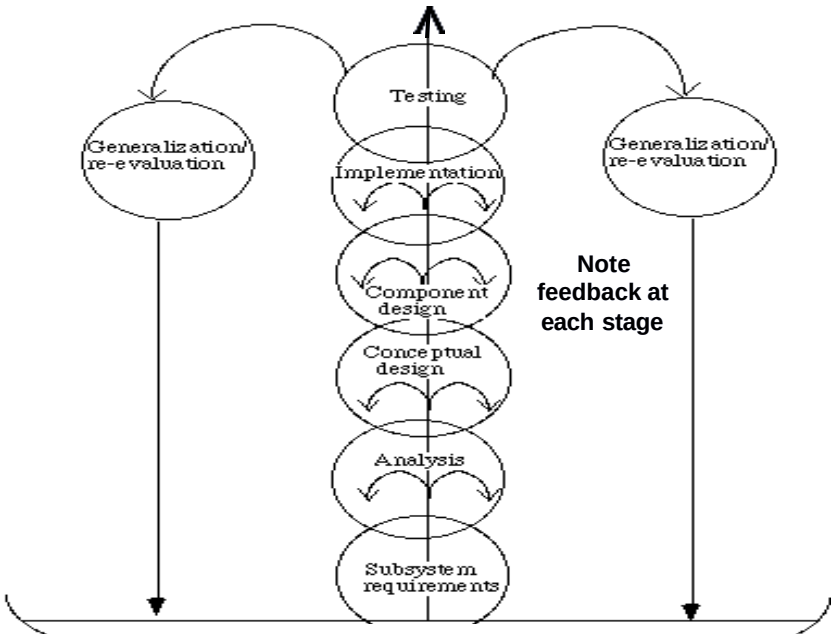
Features of waterfall model

- Systematic and linear approach towards software development
- each phase is distinct .
- Design and implementation phase only after analysis is over
- proper feedback, to minimize the rework

Weaknesses of waterfall model

- Difficult for the customer to state all the requirements in advance
- difficult to estimate the resources , with the limited information
- actual feedback is always after the system is delivered
- changes are not anticipated

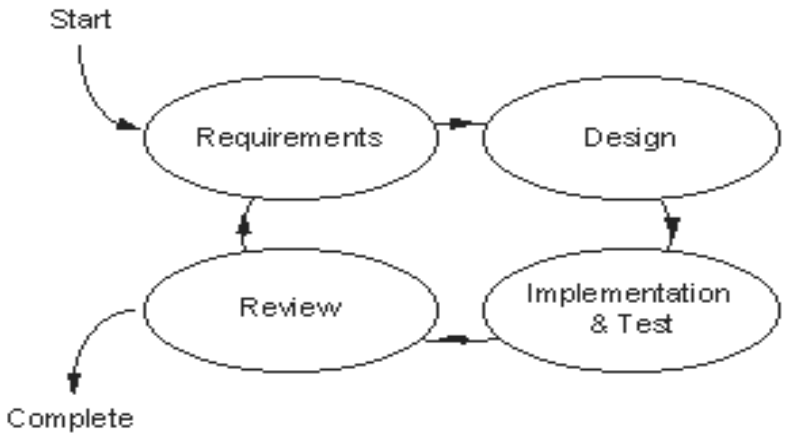
Matthew Morgenstern 9 CS211 Class - Sept. 7, 2000



The FOUNTAIN Model for an individual class or a subsystem

Iterative Lifecycle Model

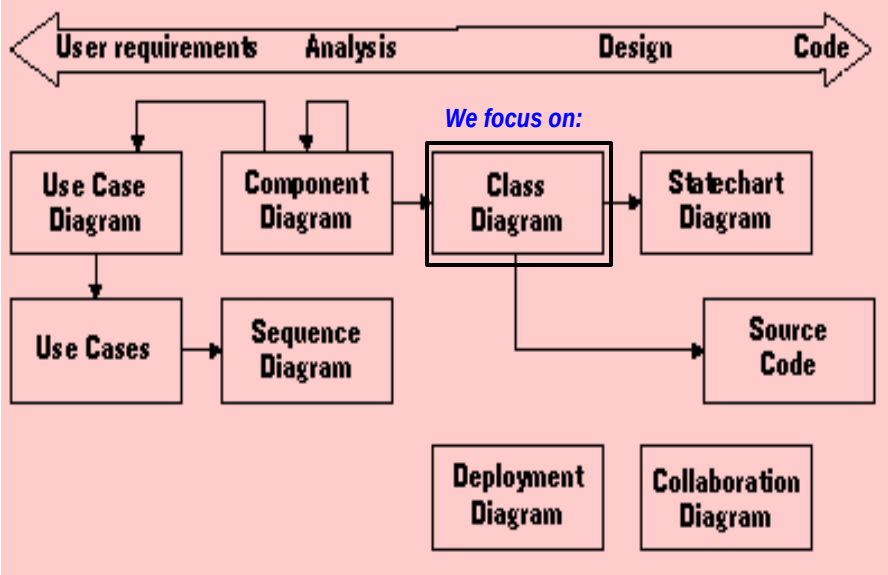
"Producing software by successive refinement"
Spiral Development emphasizes iterative phases



Unified Modeling Language (UML)

- One of the main goals of system modeling is to
  - partition a system into cohesive components
  - that have stable interfaces, creating a
  - core that need not change in response to subsystem-level changes
- UML was developed by Grady Booch, James Rumbaugh, and Ivar Jacobson

UML Modeling Techniques for Development Process

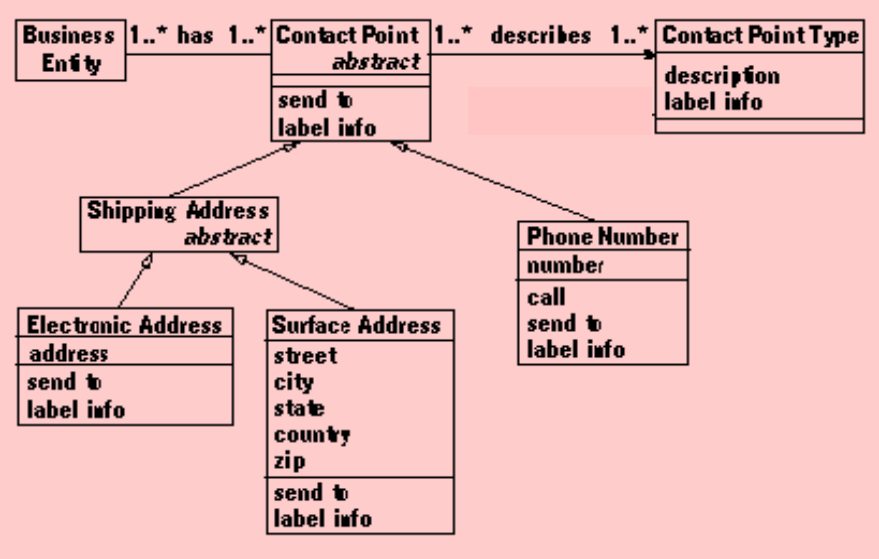


UML Modelling Diagrams

- **Use case diagrams**: external interaction with system:
  - Enroll students in courses; • Produce student transcripts.
- **Class diagrams**: object models, classes & their interrelationships:
  - including inheritance, aggregation, and associations, • Object diagrams
- **Sequence / object-interaction diagrams**: event-trace diagram:
  - defines the logic of how objects interact; e.g. in a use case scenario.
- **Component diagrams:** show the software components used:
  - shows their dependencies, interfaces, and interrelationships.
- **Deployment diagrams**: configuration of run-time processing units
  - including the hardware and software.
- **Statechart diagrams**: represent states in the behavior of an object:
  - shows events which cause transitions betwn stages & resulting actions.
- **Collaboration diagrams**: show the message flow between objects:
  - implies the basic associations between objects in an O-O application.

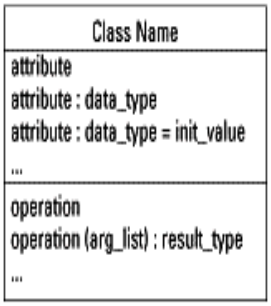
http://www.sdmagazine.com/uml/focus.ambler.htm , http://www.rational.com/uml/index.jsp

UML Class Diagram

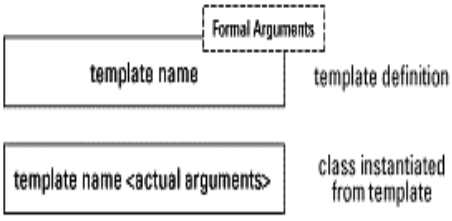


CLASS DIAGRAM Shows the existence of classes and their relationships in the logical view of a system

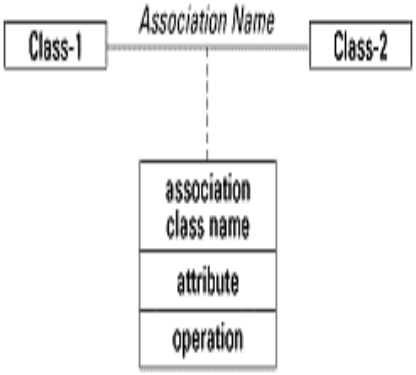
Class



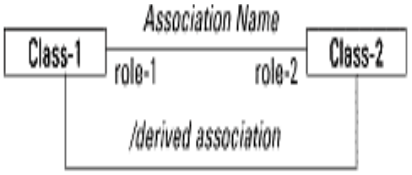
Parameterized class



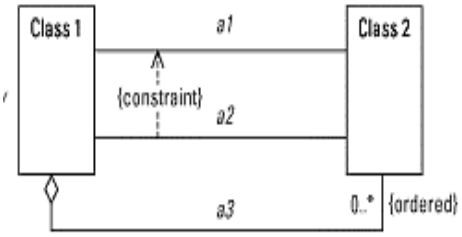
Association classes



Role names & derived association

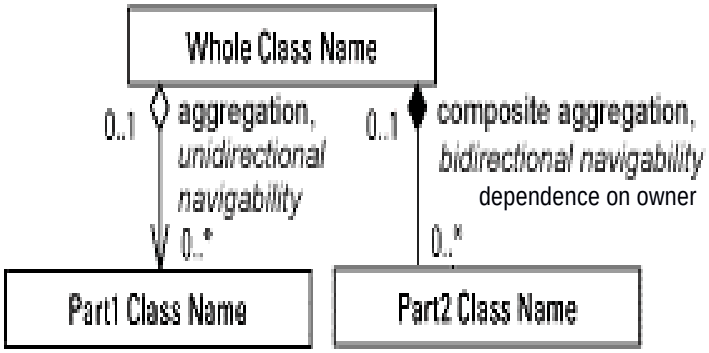


Constraint

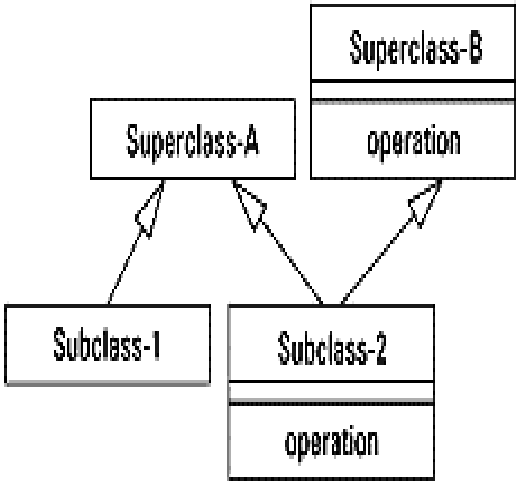


Qualified Association

Aggregation, navigability, and multiplicity



Generalization/specialization



"You can model 80 percent of most problems by using about 20 percent of the UML."-- Grady Booch

Visibility and properties

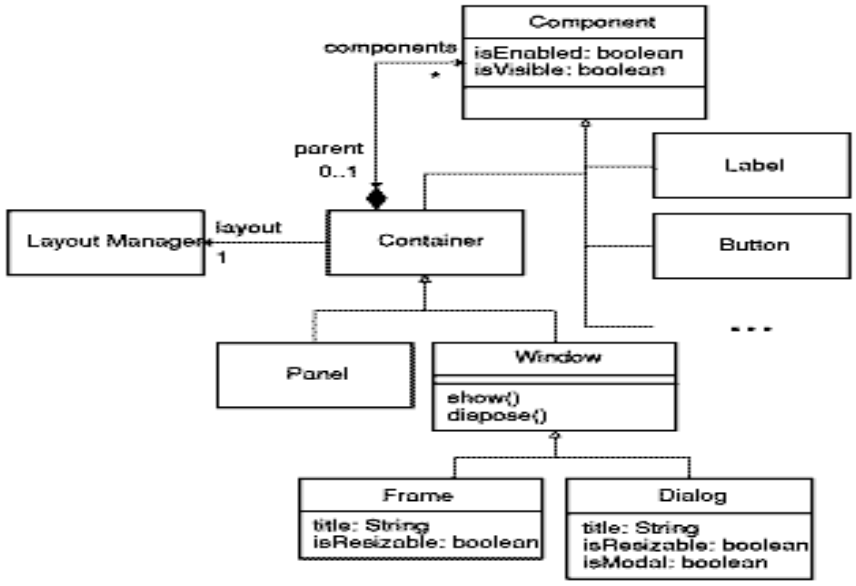
Class
- private attribute # protected attribute /- private derived attribute +$class public attribute
+ public operation # protected operation - private operation +$class public operation

Optional visibility icons

Attributes

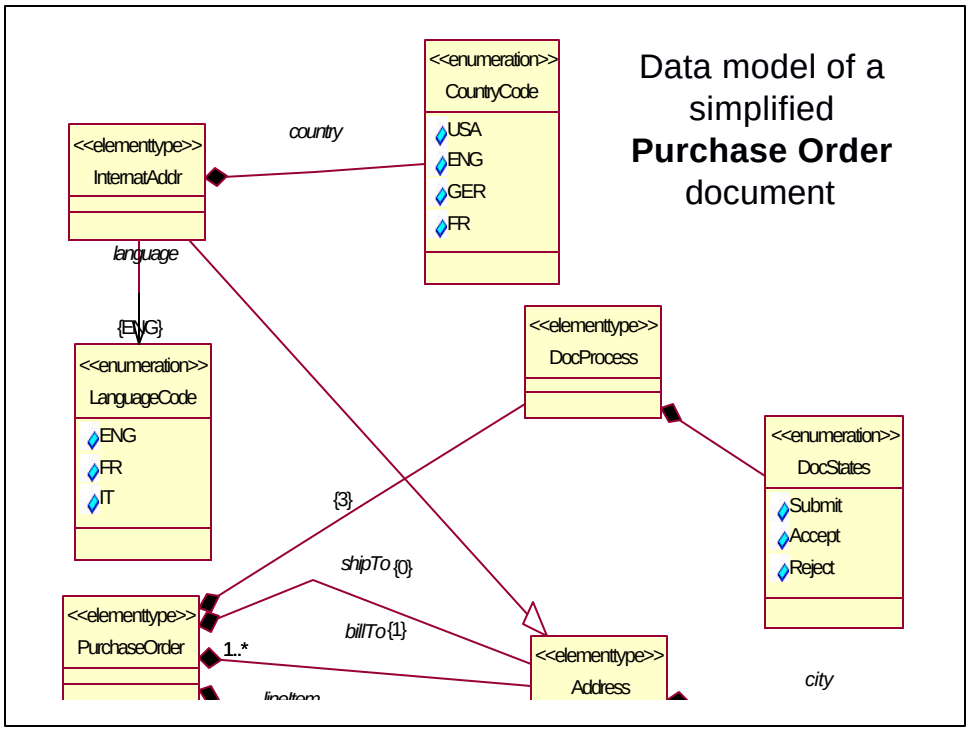
Operations

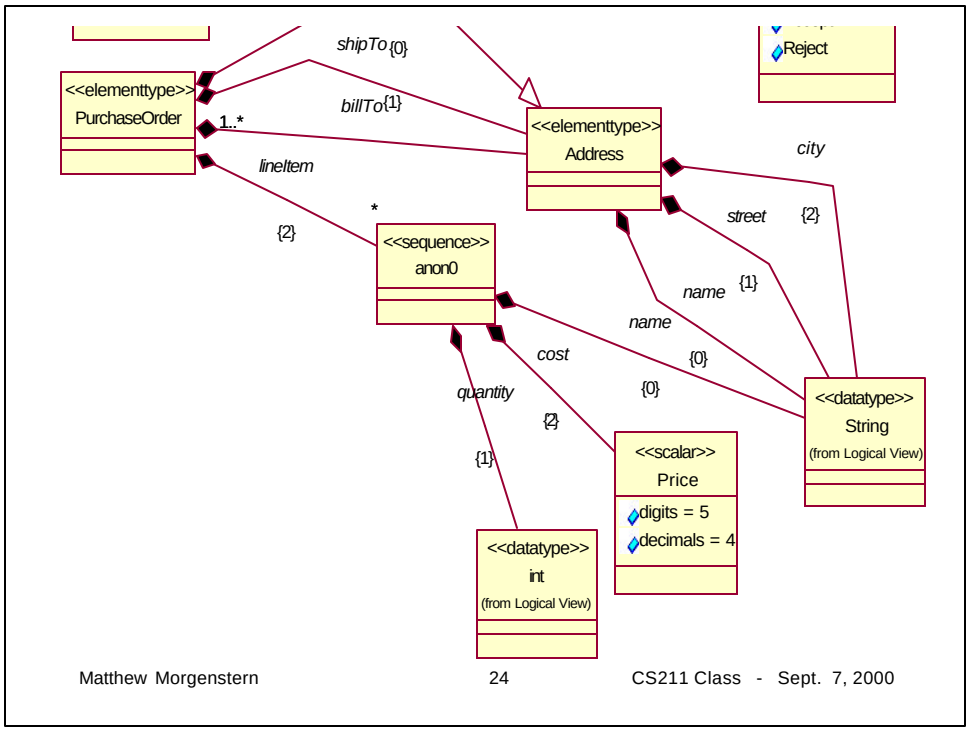
UML for Java's AWT Container Classes



From UML of Java's AWT Container Classes

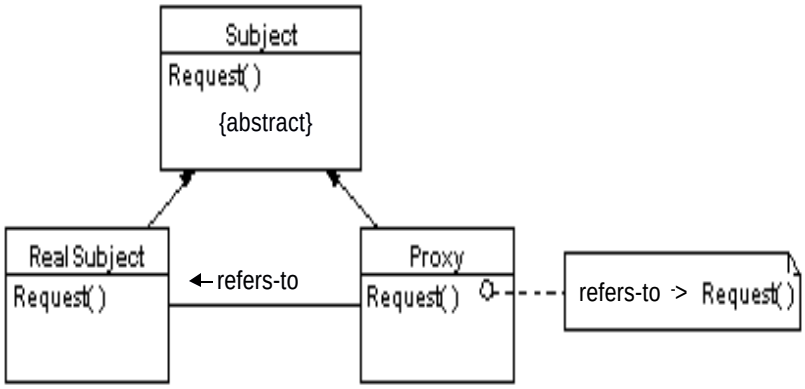
- UML highlights selected aspects the Container class in Java's AWT.
- Container is a subtype of Component.
- Components can be made visible or active
- Other kinds of components include labels, buttons, etc.
- Containers include components (which may be other containers) and also have a layout manager.
  - From a component can find its container.
  - Not all components need a container.
- Subtypes of container include panels and windows.
- Windows can show and dispose themselves.
- Window has subclasses frame and dialog.
- Frames and dialogs have titles and can be set to resize or not.
  - Although both subclasses of window do this, this behavior is not part of window itself.
- Dialogs can be marked as modal, but frames cannot.


Data model of a simplified **Purchase Order** document
<<enumeration>>
CountryCode
USA
ENG
GER
FR
country
<<elementtype>>
InternatAddr
language
{ENG}
<<enumeration>>
LanguageCode
ENG
FR
IT
<<elementtype>>
DocProcess
<<enumeration>>
DocStates
Submit
Accept
Reject
{3}
shipTo {0}
billTo {1}
<<elementtype>>
PurchaseOrder
1..*
<<elementtype>>
Address
city


shipTo {0}
billTo {1}
<<elementtype>>
PurchaseOrder
1..*
lineItem
{2}
*
<<sequence>>
anon0
<<elementtype>>
Address
city
street
{2}
name
{1}
name
cost
{0}
quantity
{0}
{2}
{1}
<<scalar>>
Price
digits = 5
decimals = 4
<<datatype>>
int
(from Logical View)
<<datatype>>
String
(from Logical View)
Reject
Matthew Morgenstern
24
CS211 Class - Sept. 7, 2000

Proxy - Design Pattern

Intent: This pattern makes the clients of a component communicate with a representative (a surrogate) rather than to the component itself



Proxy Server - Design Pattern

Motivating Problems:

- Different clients need access to a remote server
- Need optimization for access data: startup, batching
- Inappropriate to access component directly
- Do not want to hard code physical location (e.g. TCPIP address)
- Need security / access control
- Access need to be transparent and simple for the clients

Structure:

- Use a representative (Proxy) between client and component
- Offer interface of the component but performs additional pre and post-processing such as access-control, caching, logging, etc

Java Implementation of **Proxy** Design Pattern

```
public class Proxy extends Subject
{
  RealSubject refersTo;
  public void Request()
  {
     if (refersTo == null)
          refersTo = new RealSubject();
     refersTo.Request()
  }
}
```


Applications of Proxy Pattern

- **Remote Proxy**: clients or remote components should be shielded from network addresses and inter-process communication protocols
- **Protection Proxy**: Components under access control
- **Cache Proxy**: Multiple clients share read-only results from remote components
- **Synchronization Proxy**: Multiple simultaneous accesses to a component must be synchronized
- **Counting Proxy**: Audit trail for component counts
- **Firewall Proxy**: Protect local client from outside world