
CSE 142

Relationships Between Classes
Introduction to Inheritance

4/9/2008

(c) 2001-3, University of Washington

1-1

A Design Exercise

- **Suppose we are asked to design a set of classes to represent the items in a library's collection**
 - Books
 - Magazines/Journals
 - CDs
 - Videos/DVDs
 - Etc.
- **Ignoring overlap between classes, what object properties and responsibilities would be needed/ appropriate for this?**

4/9/2008

(c) 2001-3, University of Washington

1-2

Your Design Here

4/9/2008

(c) 2001-3, University of Washington

1-3

Critique

- **What do these classes (objects) have in common?**
- **How do they differ?**
- **How do we capture the common parts of the design?**
 - **Want to describe/define these once, not repeatedly in every class**
- **How do we relate the specific classes to the common parts of the design?**

4/9/2008

(c) 2001-3, University of Washington

1-4

Notes

4/9/2008

(c) 2001-3, University of Washington

1-5

Relationships Between Classes

- We are long familiar with objects that *contain* references to objects of the same or other classes
 - “Contains” or “has-a” relationship
 - Example: a car “has-a” engine, 4 tires, steering wheel, etc.
 - In this case, the relationship is one object being a component of another object
- For the library collection, we’d like to capture a different notion
 - that a book or journal “is-a” specialized kind of item in the collection
 - New kind of relationship between classes, not “has-a”

4/9/2008

(c) 2001-3, University of Washington

1-6

Interfaces: Not Quite Enough

- Methods common to two classes can be captured in a Java interface
 - Example: Comparable: classes which have a *compareTo* method
- “is-a” is a much stronger relationship than just “has something in common”
 - Strings and BankAccounts are both Comparable, but...
 - A String is not a type of BankAccount
 - A BankAccount is not a type of String

4/9/2008

(c) 2001-3, University of Washington

1-7

Key Ideas of Inheritance



- A class can be defined as an *extension* of an existing class
 - Java Syntax

```
class Book extends LibraryItem { ... }
```
- Key idea:
 1. The extended class *inherits* all of the properties, capabilities, and responsibilities of the original class
 2. The extended class can add additional properties, capabilities and responsibilities

4/9/2008

(c) 2001-3, University of Washington

1-8

More About Inheritance

- Objects in the extended class have *all* of the state and methods of the original class
 - Allows us to factor properties/responsibilities common to several classes into a single class that can be extended
- Extended classes can define additional properties and responsibilities that are appropriate for it
- Extended classes can also redefine behavior which was already defined in the original class

4/9/2008

(c) 2001-3, University of Washington

T-9

Inheritance: Java Syntax

```
public class MyClass extends OldClass {  
  
    // add instance variables  
    // add new methods, if desired  
    // redefine (override) old methods, if desired  
  
}
```

4/9/2008

(c) 2001-3, University of Washington

T-10

Some Technical Terminology

- A **base** class (or **superclass**) defines properties/responsibilities shared by a set of related classes
- A **derived** class (or **subclass**) extends a base class
 - **Inherits** all of the properties/responsibilities of the base class
 - Can define additional properties/responsibilities

class A extends B {...

- Which class is which?

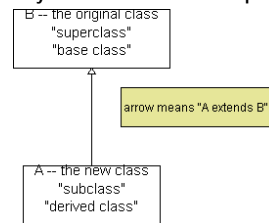
4/9/2008

(c) 2001-3, University of Washington

T-11

Drawing Inheritance Relationships

- As usual, there is a conventional way to draw it
 - Put the base class at the top
 - Similar to the style for interface relationships



4/9/2008

(c) 2001-3, University of Washington

T-12

Library Classes Revisited

- Re-work your design for the library collection classes to use inheritance
 - Define a single base class `LibraryItem` with properties and responsibilities common to Books, Journals, CDs, etc.
 - Define extended classes for each of the different kinds of collection items with additional properties/responsibilities appropriate for those classes but not for all `LibraryItems` in general

4/9/2008

(c) 2001-3, University of Washington

T-13

Design Using Inheritance

4/9/2008

(c) 2001-3, University of Washington

T-14

Inheritance in Java's GUI Libraries

- Modern applications usually have a Graphical User Interface (GUI)
- Windows, buttons, menus, icons, labels, text entry areas, pull-down lists, sliders, spinners, etc. etc.
- Java has two main packages for GUI support
 - AWT (older)
 - Swing (newer)
- Inheritance and interfaces are keys to understanding and using GUI libraries



4/9/2008

(c) 2001-3, University of Washington

T-15

Example: A GUI for The Game Of Life

- Goal: A rectangular board, with cells colored according to status, set in a window with a dark background
 - Board should be updated automatically as the game operates
- Stretch goal: be able to change cells by clicking on them

4/9/2008

(c) 2001-3, University of Washington

T-16

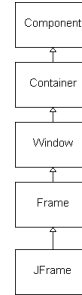
Basic Components for the GUI: JFrame

- Need a Window: has a title bar, min/max buttons, borders, and can contain other components
 - Swing class *JFrame* provides exactly this
- Look at Javadoc to find constructor and methods
- Constructor lets you put a title in the top border
- Problem: Javadoc shows no method for setting the background color
- Solution: look in the class which JFrame inherits from!
 - One of those classes has a *setBackground* method
 - Dozens or hundreds of methods are available!

4/9/2003

(c) 2001-3 University of Washington

T-17



4/9/2003

(c) 2001-3 University of Washington

T-18

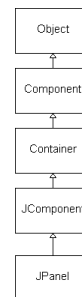
Basic Components for the GUI: JPanel

- The game board should be drawn on a subcomponent of the frame
 - Will not have its own title bar, etc.
 - Will consist of a number of lines and/or rectangles
- Swing component JPanel is suitable
 - A borderless area which can hold other components, or can be drawn on directly
 - Inherits many methods from the classes above it

4/9/2003

(c) 2001-3 University of Washington

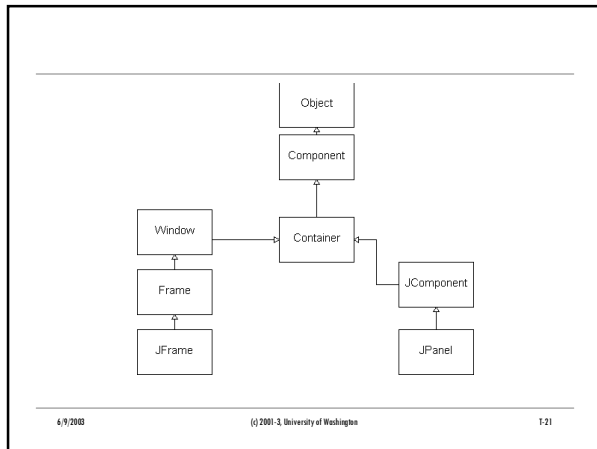
T-19



4/9/2003

(c) 2001-3 University of Washington

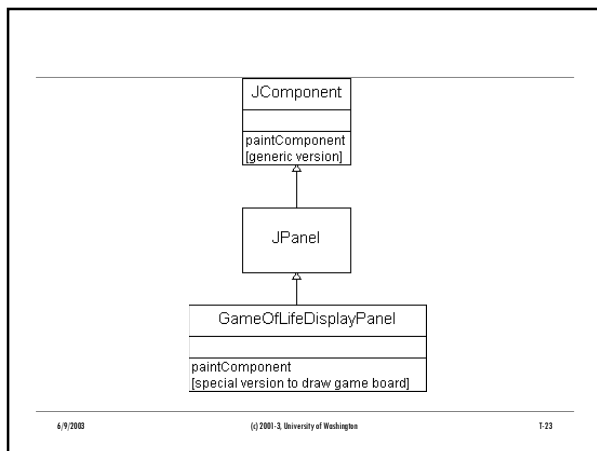
T-20



Problem: Drawing the Game Board

- Drawing or “painting” is done when the system needs to refresh the screen
- The *repaintComponent* method gets called in this case
- All components inherit a *repaintComponent* method
- Unfortunately, that method doesn’t draw a game board (surprise!)
- We have to *override* the method, i.e., supply our own version
 - This requires creating a new class which inherits from *JPanel*

4/9/2008 (c) 2001-3, University of Washington 1:22



Events in the GUI

- “Events” are things which happen, often unpredictably, often as a result of user input
 - mouse clicks, menu pull-downs, key pressed, etc.
- Swing responds to events by calling methods with pre-defined signatures (if there are such methods around):
 - Example: on a mouse click, the system will call `public void mouseClicked(MouseEvent e);`

4/9/2008 (c) 2001-3, University of Washington 1:24

Events and Interfaces

- Two objects are involved in event processing
 - One object “generates” the event
 - Another object “handles” the event
 - It is allowable for one object to play both roles
- If you want your object to “handle” an event, you implement the appropriate method
 - the methods are defined in interfaces
 - Thus, interfaces are heavily used in GUI programming
 - There are dozens of defined interfaces
- Example: mouseClicked is one of several event methods defined in the MouseListener interface

```
public class MyObject implements MouseListener {...
```

4/9/2008

(c) 2001-3, University of Washington

T-25

Inheritance (extends) vs Interface (implements)

- Interface: **A implements B**: A *must* implement every method defined by B
- Inheritance: **A extends B**: A *may* reimplement any method defined by B
- In both cases, you are free to add additional methods and instance variables
- *implements* can be thought of as a special kind of *extends*

4/9/2008

(c) 2001-3, University of Washington

T-26

Summary

- Inheritance is a key concept of object-oriented programming
- Inheritance models the “is-a” relationship
- Java keyword: *extends*
- Inheritance is heavily used in system modeling and design
 - E.g., library collection
- Inheritance is heavily used in standard library design
 - E.g. GUI libraries
- *Want to learn more? Stick around and take CSE143!*

4/9/2008

(c) 2001-3, University of Washington

T-27