CSE 1720

Lecture 3

Aggregation, Graphics

Announcements:

- labs this week:
  - preparation for labtest #1; sample problems/tasks
  - guided demo: gesture tracking (MaxMSP)
- labs next week:
  - labtest #1
  - given a description of some shape- and string-based images, implement the drawing using the services of `Graphics2D`
  - analogous to labtest #2 from cse1710 (which was based on pixel-based image modification)

Goals/To do:

- How to create, copy, and delegate to aggregates
  - example aggregates: `Pixel`, `Picture`, `Graphics2D`
- Create, modify, and iterate over collections
- Implement traversal over a collection
- Implement search within a collection
- Use services of `Graphics2D` for drawing

Goals/To understand:

- recognize aggregates from their APIs
- characterize and distinguish between two traversal techniques
- distinguish between aliases, shallow copies, and deep copies of aggregrates
- understand the characteristics of the "current settings" graphical model

This module:

- 2-3 lectures
- background material:
  - Ch 8, JBA
  - Excerpts from other sources
    - on website if/when they arise
  - CSE1710 F11 Notes: Lectures 8, 9

Quick Review: Basic Anatomy of a Class

RQ2.1-2.10

- a class has *members* (aka "features"):
  - methods
  - attributes
- features can be private or public
  - the attributes that clients can access are called fields
- method signatures must be unique
  - not only the method names, but also the parameter list
  - signature does not include the return
- attribute names must be unique
- When you use the services of a class, the compiler checks:
  - does the signature (or the attribute name) **match** what is in the class definition?

Quick Review: Method vs Attribute

RQ2.1-2.10

What is a method?

- performs some action
- has a signature and return

range of possibilities?

0 or more parameters, type compatibility must be assured

var.methodName( )
Classname.methodName( )

What is an attribute?

- holds data
- has a name and a type
- declared and initialized in the class defn

NO parameters

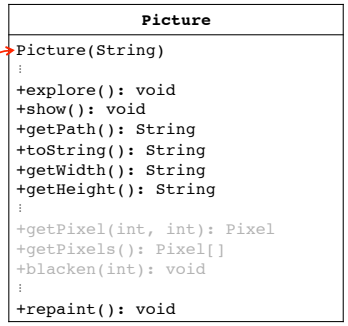
var.attributeName
Classname.attributeName

Recap:

- The `Picture` class encapsulates information about and operations on digital image files that contains pixel data
- `Picture` objects can be instantiated from files that contain pixel data (e.g., jpeg).
- We can use the services of `File` or `JFileChooser` so that our app can interface with the file system
- Graphical apps must work alongside the operating system's window manager and the platform's graphics hardware

Recap: The `Picture` class

The `string` parameter must correspond to a **pathname** that exists on the file system and that contains pixel data (e.g., a jpg file)



The `Picture` class encapsulates information about and operations on digital image files that contains pixel data

Recap: The `repaint()` method

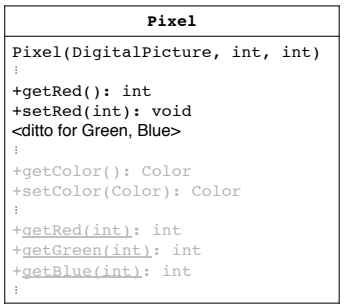
- consider `myPict.repaint()`
  - this marks the picture object as *being in need of being redrawn*
  - the method nor the app does NOT actually paint the picture itself.
- Rather, the `Picture` class's `repaint` method does the following:
  - marks the picture as being in need of being redrawn (think of a boolean flag!)
  - prompts the window manager to "survey" all of the windows.

Recap: The `repaint()` method

- The window manager, when encountering a window that is marked as being in need of being repainted, will repaint the window.
- It does so by consulting the window about *what* should be drawn.
- This is an example of abstraction – the implementation of **actual graphical rendering** and the **specificaion of what needs to be drawn** have been abstracted away from each other.
  - This design implements a separation of concerns
  - This design implements abstraction by delegation

Recap:

- To manipulate `Picture` objects, we modified the object's pixels.
- To manipulate a `Pixel` object, we modified the object's R, G, B values
- E.g., `L03App01`

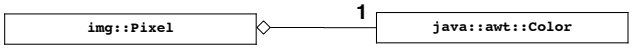


What is an aggregate?

- So `L03App01` demonstrated the manipulation of a `Picture` object, via the modification of the object's pixels.
- To manipulate a `Pixel` object, we modified the object's `Color`
- What do we take away from this?
  - …that any `Pixel` object has, as one of its **attributes**, a `Color` object
- The situation: *a class has as one of its features an attribute that is non-primitive*[*]
  - recall goal: "recognize aggregates from their APIs"
  - ** excluding strings*

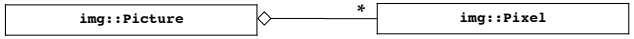
Illustrating Aggregation using UML

- The `Pixel` object HAS-A `Color` object (as one of its attribute values)
- It has one such `Color` object, hence the number 1 in the UML diagram below



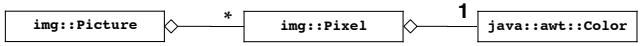
Illustrating Aggregation using UML

- To manipulate `Picture` objects, we modified the object's pixels.
- The `Picture` object HAS-A collection of `Pixel` objects (as one of its attribute values)
- Since we don't know the size of the collection, we use the asterisk * in the UML diagram below



Illustrating Aggregation using UML

- The HAS-A relationship can stand in a "chain"
- *multiplicity* is indicated



An object can "hide" its parts

- An aggregate (by definition) has (at least one) attribute that is a non-primitive value
- Is that class required to provide an accessor for that attribute?
  - No. It is up to the designer to define the methods of a class
  - However, it usually makes sense to provide some sort of access
- What if the class wants to provide "read-only" access to the aggregate part?
  - e.g., clients can examine the state of the aggregate, but should not be allowed to change the state.
- What is the strategy for this?
  - topic of next lecture. Read Chapter 8 prior to L04.

Traversal of Collections

- Indexed traversal
- Iterator-based traversal

Appendix: recap of WM

What is this *window manager* and why do I care?

– first, a more fundamental question:
  - what is the *desktop metaphor*?
    – a set of UI concepts that treat the computer display as if it were the user's real-world desktop
    – desktop items include: documents, folders, desk accessories (calculator, calendar)
    – the purity of metaphor now diluted and now includes things without real-world counterpart
      » menu bars, task bars, docks, trashcans,

– key feature: desktop items can ***overlap***

What is this *window manager* and why do I care?

– it is system software
  - operates computer hardware (the graphics card, in this case)
  - provides platform for running apps

– it provides display functionality for apps
  - controls placement and appearance of windows
    – open, close, minimize, maximize, move, resize
  - implements look and feel of window decorators
    – borders (decorative and functional), titlebar (title and buttons)

The window manager provides services to the VM

- **VM:** *Hi WM, I have this app that wants to draw something graphical on the display…*
- **WM:** *ok VM, here is some screen real estate.*
  - Your app can draw within that region, but not outside it. *(It can try, but I will never permit it to happen)*
  - I will decide what actually gets drawn. *(There may be overlapping windows, so your real estate may be occluded)*
  - I can't guarantee this region. *(The user may move the window, or resize or minimize it)*