CSE 1720

Lecture 4

Aggregation, Graphics II

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

Today's Topics

- Java 2D API Concepts
- Collections, Collection Traversal
- Aggregations vs Composition

2D Graphics

The Java 2D API… A Basic Overview

- apps that use graphics must launch a window
  - thus, such apps involve the window manager, as discussed last lecture
- what if the app **wants to draw something**?
  - how can the app do so?
  - before answering, let's first emphasize **AGAIN** that the technically correct way to pose this question is:
    - what if the app **wants to specify something to be drawn**?
    - the window manager actually does the drawing

How to draw something…

- the app asks the display window to access its `Graphics2D` object
- the app uses the `Graphics2D` object to specify what is to be drawn

Coordinate spaces

From: http://docs.oracle.com/javase/tutorial/2d/overview/coordinate.html

- The Java 2D API maintains two coordinate spaces:
  - *User space* – The space in which graphics primitives are specified
  - *Device space* – The coordinate system of an output device such as a screen, window, or a printer
- User space is:
  - a device-independent logical coordinate system.
  - the coordinate space that your program uses.
- All geometries passed into Java 2D rendering routines are specified in user-space coordinates.
- When it is time to render the graphics, a transformation is applied to convert from user space to device space. The origin of user space is the upper-left corner of the component's drawing area.

The `Graphics2D` API

- a `Graphics2D` object encapsulates the drawing region (in *device space*) and a set of supported drawing operations on that drawing region (in *user space*)
- All methods can be divided into two groups:
  - Methods to draw a shape
  - Methods that affect rendering
- The state of a `Graphics2D` object is, in part, the current settings for the following attributes:
  - The stroke width, the way the strokes are joined together
  - The current translation, rotation, scaling, and shearing values
  - The paint color
  - The fill pattern

How to draw something… revisited

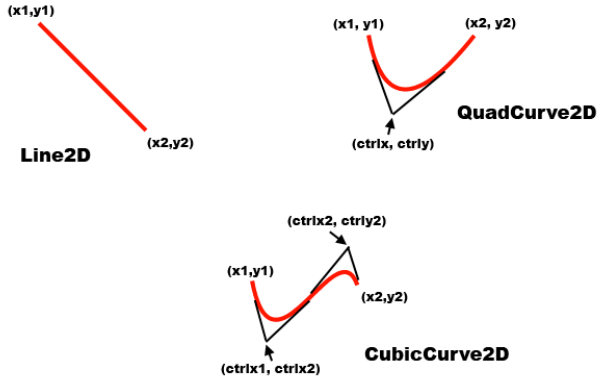
- the app asks the display window to access its `Graphics2D` object
- E.g.,

```
Graphics2D graphicsObj = myPict.getGraphics();
```

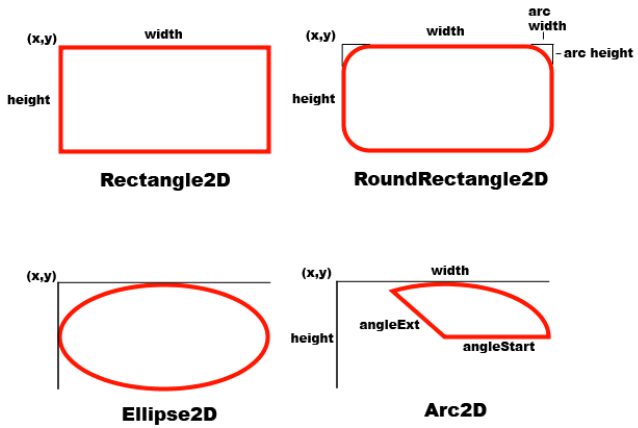
- the app uses the `Graphics2D` object to specify what is to be drawn
  - assign settings as desired
  - draw shape (shape will use settings)
  - reassign settings as desired
  - draw shape (shape will use updated settings)
  - … and so on…

How to draw…

ref: http://java.sun.com/developer/technicalArticles/GUI/java2d/java2dpart1.html



How to draw…



About the shapes…

- Classes for each of these seven shape primitives can be found in the API.
  - use `Rectangle2D`, not `Rectangle`
- The constructors can be found in the **nested classes**
- You can use the `Double` versions
- E.g.,

```
Rectangle2D shape1 =
      new Rectangle2D.Double(0, 0, 20, 50);
```


Construct a shape object…

```
Graphics2D graphicsObj = myPict.getGraphics();
Rectangle2D shape1 =
            new Rectangle2D.Double(0, 0, 20, 50);
graphicsObj.draw(shape1);
Rectangle2D shape2 =
            new Rectangle2D.Double(60, 60, 20, 50);
graphicsObj.fill(shape2);
```


Various settings

- Stroke :
  - The default is a solid line of width 1.0
  - to change:

```
BasicStroke newStroke = new BasicStroke(4.0f);
graphicsObj.setStroke(newStroke);
```

- Paint Color:
  - The default is Color.WHITE
  - to change:

```
graphicsObj.setColor(Color.BLUE);
```


Various settings

- Paint Color, better version:
  - to change:

```
graphicsObj.setPaint(Color.BLUE);
```

- also can do this (fancier):

```
Point p1 = new Point(0, 0);
Point p2 = new Point(50, 50);
GradientPaint paint1 =
       new GradientPaint(p1, Color.RED, p2, Color.MAGENTA, true);
graphicsObj.setPaint(paint1);
```

Collections

About Collections…

The course material that concerns collections (e.g., traversal, static/dynamic allocation, etc) will make a lot **more sense** if you have a crystal clear understanding about what a collection actually is

So what is a collection anyway?

Let's start with what a collection is **NOT**.

A collection is **NOT** a set.

- A set is, by definition, a collection that does not contain duplicate elements.

A collection is **NOT** a list.

- A list is, by definition, an ordered collection.

So what is a collection anyway?

In terms of what a collection **is**, all we really can say is:

- a collection is a thing that has elements.

In terms of what a collection **does**, we can say some more:

- a collection is a thing that lets you add/remove and traverse the elements.
- a collection is a thing that can report its size

The Forest Gump way of defining a collection

A collection **is** what a collection **does.**

Does it have elements that I can traverse?
Does it let me add elements?
Does it let me remove elements?
Does it tell me its size?

Then it is a collection.*

*a collection does a few other things, but we will talk about these later

The JBA way of defining a collection

A collection is an aggregate in which the multiplicity is variable and in which the aggregated parts are called elements.

What does “traverse” mean?

A traversal can be thought of as a trip that visits each element once and only once.

JBA, p.318

What this means:

- No element can be missed
- No element can be visited more than once.

What does “traverse” **NOT** mean?

That the elements will be visited in **any particular order**

- Even if you traverse a given collection several times, you should not assume that the elements will be visited in the same order.
- There is **no order** defined over the elements (even if the elements are things that you may commonly think of as having a “natural” order, such as numbers)

What does "traverse" **NOT** mean?

That you are able to do **"partial trips"**

- e.g., visit "every other element" or "the first half of the elements" or any other trip that is anything other than the complete traversal of all the elements
- A collection simply is not defined to provide this.
- This is just a variant of trying to **impose a particular order** on the elements of the collection.

Iterator-Based Traversal 8.2.4

If `collection` is a variable that refers to a collection object, then the following enhanced for loop will be provided:

```
for (ElementType e : collection) {
     // visit element e
}
```

OK, but what is `ElementType`?

…a collection is an aggregate, which means, by definition ***a class that has as one of its features an attribute that is non-primitive.***
What is this non-primitive type?
Don't know – let's just call it `ElementType` for the time being…

Iterator-Based Traversal 8.2.4

Another version of iterator-based traversal is…

```
while (collection.hasNext()) {
     ElementType e = collection.next();
}
```

It is equivalent to the enhanced for loop version…
(see the API of the Iterable interface)

Indexed Traversal 8.2.3

A sneaky bit of material that has the potential to confuse…

we just emphasized that traverse does not mean visiting the elements in any particular order… but…

Sometimes a collection may, *in addition to its requisite methods*, also support **an indexing scheme for its elements,** such as via this method:

```
public ElementType get(int index)
```

Indexed Traversal 8.2.3

If there is an indexing scheme, then we can implement full and even partial traversals…

```
for (int i = 0; i < collection.size(); i++) {
     ElementType e = collection.get(i);
}
```

Since collection is a collection, it must have a size() method

Where can I get me a collection?

- the `Portfolio` class implements a collection of `CreditCard` elements; use the static method `getRandom()` to get a randomly-populated collection
- the `Picture` class implements getPixels(), which returns an array of Pixel[]. The array is not a collection (technically speaking), but provides the identical behaviours

- If you want to create and populate your own collection, I recommend you wait until we cover section 9.3.3, Generics

Where can I get me a collection?

- the `Portfolio` class implements a collection of `CreditCard` elements; use the static method `getRandom()` to get a randomly-populated collection
- the `Picture` class implements getPixels(), which returns an array of Pixel[]. The array is not a collection (technically speaking), but provides the identical behaviours

Where can I get me a collection?

What if I want to create and populate my own collection:

1. use a constructor to create an empty collection
2. add the elements one by one

- its not possible* to create and populate a collection all in one step

*well, a collection may provide constructors that allow the client to specify the initial content of a collection by passing a reference to another collection; such constructors are for convenience only and are implemented using the two steps above anyway.

we will do this once we cover section 9.3.3, Generics

A design tradeoff

When a new, empty collection is created, a block of run-time memory will be allocated for this object.

How large should this block be?

- If the block is too small:
  - then the size will be quickly exceed. When this happens, a whole new empty collection will need to be created, using a larger block and all of the elements copied over.
- If the block is too large:
  - a significant amount of memory will sit empty and cannot be used for anything else

The extreme form of the **small block** version is to start with a block so small that is it not big enough to hold even a single element. Then the amount of memory that is used grows and shrinks as the collection grows and shrinks.

Static vs Dynamic

sec 8.2.1