CSE 1720

Lecture 10

Graphics and Fonts

Topics

- fonts, string formatting

The Basics

- `draw(Shape)`
  - our go-to method for drawing shape primitives
- `drawString(String, int, int)`
- `drawString(AttributedCharacterIterator, int, int)`
  - these will be our go-method for drawing strings
  - the version using `AttributedCharacterIterator` will be covered, time permitting
  - E.g., `L10App01`

How is Font information Encapsulated…

- e.g., `L10App02`
  - information about **font face**, **font style**, **font size** is encapsulated in the Font object
  - refer to the `Font` API

Font Concepts

- *font family* or *typeface* refers to a typographic design across several faces, e.g., Helvetica, Courier
  - members within the family share a common design by vary in terms of weight (bold/not bold), orientation (italics/non italics), width (condensed/non condensed)
- *font* refers to a specific member of a font family
  - Times Italic
  - once upon a time (in the era of metal types), font also referred to *point size*
  - the Font API refers to this as *font name*
- *font style* indicates plain, bold, italic, bold+italic
- *font size* is more complicated…

Font Concepts

- point size refers to the size of the font,
  - 1 pt is approx 1/72 of an inch; a 72 pt font is approx 1" high
- what does point size actually mean, given that the characters are all different heights?
- for this, we need some basic font terminology

Typography Concepts

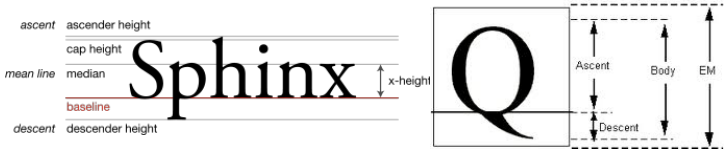
- *character* refers to the smallest semantic unit of a language
- *glyph* refers to the specific form characters can take on in a font face
- e.g.,
  character: the unicode character \u0041
  corresponding glyphs: 𝘈 **A** 𝒜 A
- to do typographic layout, glyphs for the characters of a given string must be selected and positioned
  - in older times, this was done manually
  - nowadays, this is done automatically or semiautomatically

Font Concepts

- all fonts have a *baseline*, *midline* (or mean line), and *cap height*
- *cap height* is the height of the capital letters that have "flat" tops, as opposed to capital letters that have "round" tops.
  - Round capital letters may overshoot the cap height
- *midline* is halfway from the baseline to the cap height
- *x-height* is the **distance** between the baseline and the midline
  - usually the height of an 'x' character

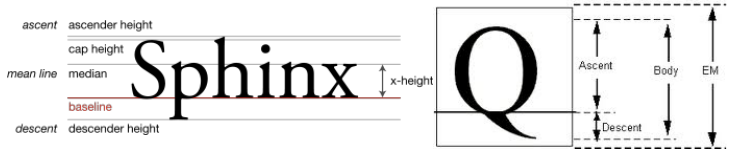
Font Concepts

- *ascenders* are the portion of a glyph that extends above the midline
- *descenders* are the portion of a glyph that extends below the baseline



Font Concepts

- the *em square* is an invisible box which is typically a bit larger than the distance from the tallest ascender to the lowest descender
- *point size* indicates the size of the *em square*



Pointing out an interesting phenomenon

- In `L10App03`
  - a duplicate of `L10App02`
    - except we exchange the order of the `show()` and `getGraphics2D()` method invocations
  - observe that we can show the `Picture` object either before or after we mutate the `graphics2D` object
  - the rendered text has a different appearance, yet the font object is the same
  - what explains this?

Rendering Hints Impact the Appearance of Fonts

- In `L10App04` we extract all of the rendering hints
  - this requires us to use a Map abstract data type
  - how is a Map different from a Collection? (foreshadow to Ch 10)

getGraphics2D() ***before*** pict is shown

```
Fractional metrics enable key: Default fractional text metrics mode
Global rendering quality key: Default rendering methods
Global antialiasing enable key: Default antialiasing rendering mode
Stroke normalization control key: Default stroke normalization
Text-specific LCD contrast key: 140
Text-specific antialiasing enable key: Default antialiasing text mode
```

getGraphics2D() ***after*** pict is shown

```
Fractional metrics enable key: Default fractional text metrics mode
Global rendering quality key: Default rendering methods
Global antialiasing enable key: Default antialiasing rendering mode
Stroke normalization control key: Default stroke normalization
Text-specific LCD contrast key: 140
Text-specific antialiasing enable key: Antialiased text mode
```


Graphics2D and Antialiasing

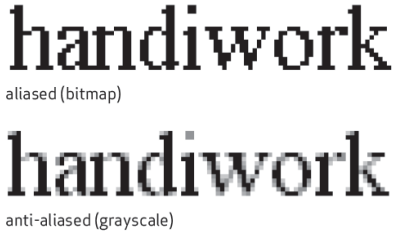
- Graphics2D has various rendering hints
  - one of these hints concerns the type of antialiasing
  - there are various versions available
- the difference is the visual presentation was because different antialiasing schemes were in effect
  - when we obtain the Graphics2D object after the Picture object is shown, then one of the Graphics2D object's rendering hints gets modified
- so what is the antialiasing?

Aliasing, Antialiasing

- up to a certain point in time, most computers were only capable of displaying 'aliased' (bitmapped) text on screen
- pixels either on or off, making type look very jagged on the screen

Aliasing, Antialiasing

- Anti-aliasing is a method to give the illusion of smooth outlines through the use of varying levels of gray surrounding the outline of the character
- Works well at larger sizes, not so well at smaller sizes.
  - smaller sizes → fuzzy outline, results in eyestrain and fatigue
  - sometimes turned off below a preferred point size



Fonts

- suppose a client wants to specify a different font
- refer to Font constructor

```
Font(String name, int style, int size)
```

font face name or a font family name

- so what are the possible font faces?
  - physical fonts
  - logical fonts

Possible Font Faces

- physical fonts
  - actual font libraries containing glyph data and tables to map from character sequences to glyph sequences
  - font libraries need to make use of a *font technology*, such as TrueType or PostScript Type 1
- logical fonts:
  - defined with Java SE
  - available on any Java platform
  - can be thought of as aliases for some underlying font that has the properties implied by its name.
  - five are defined: Dialog, DialogInput, Monospaced, Serif, SansSerif

Possible Font Faces

- which fonts are installed on a particular platform?
  - see `L10App05`

Defining and Using Another Font Face

- see `L10App06`

- other slides…

Typography Concepts

- consider the following task: given some characters, which glyphs should be employed to represent them?
- This is accomplished by typesetting software (e.g., LaTeX) or desktop publishing/wordprocessor)
- A goal is to achieve a visually pleasing appearance and to avoiding situations that are unattractive or reduce legibility
  - there are many possible problems
  - to sample these, let us consider three common remedies:
    - kerning
    - tracking
    - use of ligatures

Draw a box around a string

- see `L10App07`

Graphics Device

- screens, printers or image buffers can be the destination of Graphics2D drawing methods.
- Each graphics device has one or more GraphicsConfiguration objects associated with it.
  - different drawing modes or drawing capabilities (such as different resolutions or colour depths)
- These objects specify the different configurations in which the GraphicsDevice can be used.
- which fonts are installed on a particular platform?
  - installed means that the VM can make use of them

Graphics Environment

- a conceptualization used by Java to represent the collection of GraphicsDevice objects and Font objects available to a Java(tm) application on a particular platform.

- a which fonts are installed on a particular platform?
  - installed means that the VM can make use of them