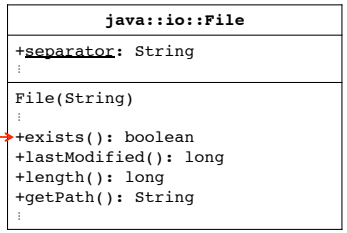
CSE 1710

Lecture 9

Working with Images II

Recap: the `File` class

just because an object is instantiated for a pathname doesn't necessarily mean that there is an actual file corresponding to that pathname



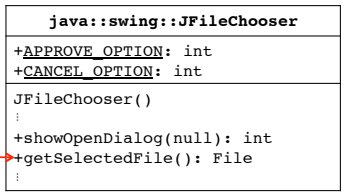
The `File` class encapsulates information about and operations on either potentially-existing files and already-existing files.

Recap: we know how to...

- distinguish between a directory and a "normal file"
  - both are considered to be *files*
- obtain a `File` object using a constructor, given a pathname
- determine whether a file object corresponds to an actual on the file system

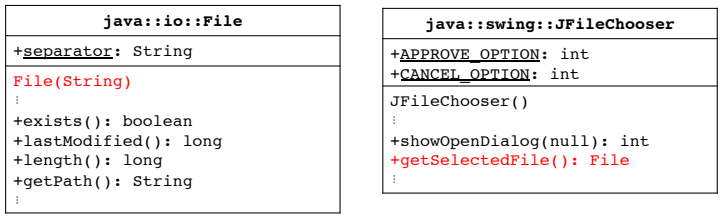
Recap: the `JFileChooser` class

`getSelectedFile()` will always return *something*, even before the dialog is **even opened**



`JFileChooser` encapsulates information about and operations on a file choice dialogue.

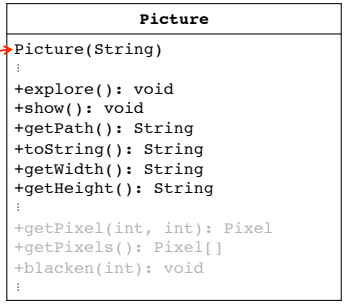
Recap: Using the `File` or the `JFileChooser` classes



either approach will get you a reference to a **file object** and we need this reference in order to…

Recap: The `Picture` class

The `string` pathname must correspond to a file that exists on the file system and that contains pixel data



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

Recap: we know how to...

- construct a `Picture` object from a pathname
- invoke several methods on the `Picture` object
  - show it, explore it
  - what is the file from which it has been rendered?
  - what is its width and height (in pixels)

About `picture.show()`

- the VM needs to make use of the services of the window manager for the image to actually appear on the display
  - what does the window manager do?
  - why does the VM need to use its services?

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: paper documents, folders, desk accessories (calculator, calendar)
- the purity of metaphor is now diluted
  - it now includes things without real-world counterparts
    - menu bars, task bars, docks, trashcans,
- key issues:
  - the "desktop" real estate is limited
  - desktop items need to ***overlap***

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

- it is system software (not app 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 aspects)
    - titlebars (titles and/or functional aspects)

The window manager provides services to the VM

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

foreshadow to next lecture…

- **VM:** *Hi WM, I have this app that wants to draw something graphical on the display…*
  - *glyphs, circles, lines, arcs, etc*
- **WM:** *ok VM, here is some screen real estate.*
  - You better use the services of a class that will encapsulate all of the complexity of device-dependent graphics display
    - e.g. (for the purpose of illustration)
      - » how are pixels are addressed on the display device?
      - » what voltage to apply to illuminate a pixel at the its maximum level?
  - The class that encapsulates all of this is `Graphics2D`
    - provides methods for drawing
    - hides away device-dependent details

The take-away point

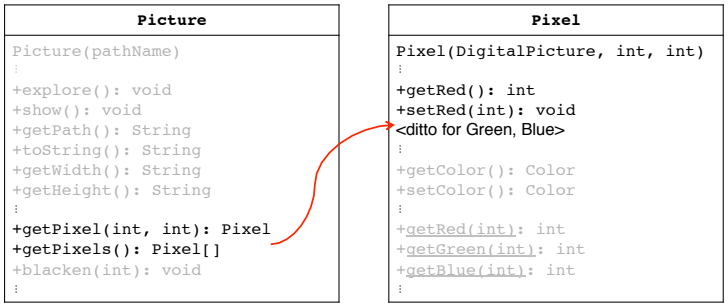
- `picture.show()` does not actually directly "commandeer" the display and draw on it
- the WM gives the app some real-estate, but that real-estate is subject to events outside of the app's control
  - e.g., user moves or occludes the window

How can an image be modified?

1. transformation of size
   - proportional increase/decrease
   - stretching
2. transformation of pixels
   - change values in an absolute or relative way
   - relocation
3. other?

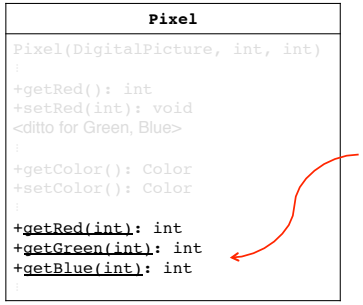
What is possible is determined by the services offered by the class you are using to encapsulate your image.

Intro: The `Pixel` class



The `Pixel` class encapsulates information about and operations on a pixel in a digital image

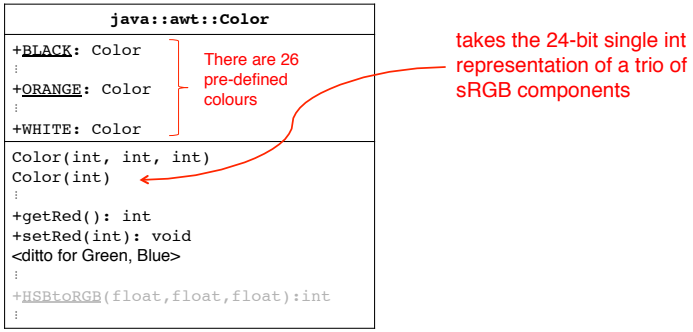
Elaboration: The `Pixel` class



a trio of sRGB (standard RGB) components can be represented as a 24-bit integer:
bits 0-7 ➔ blue
bits 8-15 ➔ green
bits 16-23 ➔ red

These static methods can extract the color components from the RGB int representation

Intro: The `Color` class



The `Color` class encapsulates information about and operations on the representation of color

Possible image modifications

1. for a particular pixel
   - change values in an absolute way
2. for a particular row or column
   - change values in an absolute way
   - change values in a relative way (e.g., reverse the image)
3. for all pixels in the image
4. for all pixels in a region of the image

0. Reference to a particular pixel

– for a particular pixel
- requires reference to a *picture object*
- accessor

```
myPict.getPixel(10, 10);
```

- constructor

```
new Pixel(myPict, 10, 10);
```

both produce a reference to ***the same pixel***

1. Modification of a particular pixel

– for a particular pixel
- change a colour component
- change the colour (use of `Color` class)

- Egs

```
thePixel.setColor(Color.RED);
p.setColor(new Color(255, 0, 0));
p.setRed(MAX_VALUE);
```

2. Modification of a particular row or column

– need expression to indicate targeted pixels
  • iterate over a column or row, using coordinates

```
for (int rowIdx=0; rowIdx< MAX_ROWS; rowIdx++) {
…
}
```

– convenience method to blacken row

```
myPict.blacken(ROW_INDEX + 25);
```


3. Modification of all pixels

– introduce notion of an array
– easy way to *iterate* over all elements

```
Pixel[] allPixels = myPict.getPixels();
for (Pixel p : allPixels) {
…
}
```

- possibilities
  • simulate sunset
  • negative colour image
  • implement color-greyscale conversion