CSE 331
Software Design & Implementation

Kevin Zatloukal
Summer 2016
Java Graphics and GUIs
(Based on slides by Mike Ernst, Dan Grossman, David Notkin, Hal Perkins, Zach Tatlock)
Review: how to create a GUI

1. Create a JFrame (window)

2. Add components to it
   - organize them on the screen using a layout manager

3. Add handlers on the components
   - one for each event you want to respond to
**JPanel – a general-purpose container**

In addition to all the uses we saw in lecture:

- Commonly used as a place for graphics

A particularly useful method:
  - `setPreferredSize(Dimension d)`
  - you may want to call this when using JPanel as a canvas
    - (don’t usually want to otherwise)
Graphics and drawing

What if we want to actually draw something?
- A map, an image, a path, ...?

Answer: Override method `paintComponent`
- Components like `JLabel` provide a suitable `paintComponent` that (in `JLabel`’s case) draws the label text
- Other components like `JPanel` typically inherit an empty `paintComponent` and can override it to draw things

Note: As we’ll see, we override `paintComponent` but we don’t call it
Example

SimplePaintMain.java
Graphics methods

Many methods to draw various lines, shapes, etc., ...

Can also draw images (pictures, etc.):
  - In the program (not in `paintComponent`):
    - Use AWT’s “Toolkit” to load an image:
      ```java
      Image pic =
      Toolkit.getDefaultToolkit().
      .getImage(file-name (with path));
      ```
    - Then in `paintComponent`:
      ```java
      g.drawImage(pic, ...);
      ```
**Graphics vs Graphics2D**

Class **Graphics** was part of the original Java AWT

- Has a procedural interface:
  
  ```java
  g.drawRect(...), g.fillOval(...), ...
  ```

Swing introduced **Graphics2D** (extends **Graphics**)

- Added an object interface – create instances of **Shape** like **Line2D**, **Rectangle2D**, etc., and add these to the **Graphics2D** object

Actual parameter to **paintComponent** is always a **Graphics2D**

- Can always cast this parameter from **Graphics** to **Graphics2D**
- **Graphics2D** supports both sets of graphics methods
- Use whichever you like for CSE 331
So who calls `paintComponent`? And when??

- Answer: the window manager calls `paintComponent` *whenever it wants!!!* (a callback!)
  - When the window is first made visible, and whenever after that some or all of it needs to be *repainted*
- Corollary: `paintComponent` must *always* be ready to repaint regardless of what else is going on
  - You have no control over when or how often
  - You must store enough information to repaint on demand
- If “you” want to redraw a window, call `repaint()` from the program (*not* from `paintComponent`)
  - Tells the window manager to schedule repainting
  - Window manager will call `paintComponent` when it decides to redraw (soon, but maybe not right away)
  - Window manager may combine several quick `repaint()` requests and call `paintComponent()` only once
Example

FaceMain.java
How repainting happens

It’s worse than it looks!
Your program and the window manager are running concurrently:

- Program thread
- User Interface thread

Do not attempt to mess around – follow the rules and nobody gets hurt!
Crucial rules for painting

- Always override `paintComponent(g)` if you want to draw on a component
- Always call `super.paintComponent(g)` first
- **NEVER, EVER, EVER** call `paintComponent` yourself
- Always paint the entire picture, from scratch
- Use `paintComponent`'s `Graphics` parameter to do all the drawing. **ONLY** use it for that. Don’t copy it, try to replace it, or mess with it. It is quick to anger.
- **DON’T** create new `Graphics` or `Graphics2D` objects

Fine print: Once you are a certified™ wizard, you may find reasons to do things differently, but that requires deeper understanding of the GUI library’s structure and specification
What’s next – and not

You’re on your own to explore all the wonderful widgets in Swing/AWT.
  – Have fun!!
  – (But don’t sink huge amounts of time into eye candy)
Reminder: UI thread

Recall that sometimes the program has additional threads, e.g.:
  – one thread is waiting for network data (“the network thread”)
  – another thread is displaying the UI (”the UI thread”)

All UI actions happen in the UI thread – *including callbacks* like `actionListener` or `paintComponent`, etc. defined in your code.

After event handling and related work, call `repaint()` if `paintComponent()` needs to run. **Don’t** try to draw anything from inside the event handler itself (as in *you must not do this!!!*)

Remember that `paintComponent` must be able to do its job whenever the window manager calls it – so any data it needs to render must be prepared in advance.
Event handling and repainting

Remember: your program and the window manager are running concurrently:

- Program thread
- User Interface thread

It’s ok to call `repaint` from an event handler, but **never call `paintComponent` yourself** from either thread.
Synchronization issues?

Yes, there can be synchronization problems
- (cf. CSE332, CSE451, CSE452, …)
Not generally an issue in well-behaved programs, but can happen
Advice:
- Keep event handling short
- Call repaint when data is ready, not when only partially updated
- Don’t update data in the UI and program threads at the same time (particularly for complex data)
- Never call paintComponent directly
  • (Have we mentioned you should never ever call paintComponent? And don’t create a new Graphics object either.)
If you are building industrial-strength UIs, learn more about threads and Swing and how to avoid potential problems (Swing tutorial, …)
Larger example – bouncing balls

A hand-crafted MVC application. Origin is somewhere back in the CSE142/3 mists. Illustrates how some swing GUI components can be put to use.

Disclaimers:

– Not the very best design (maybe not even particularly good)
– Unlikely to be directly appropriate for your project
– Use it for ideas and inspiration, and feel free to steal small bits if they really fit

Enjoy!