SWT

Olivier Cailloux

LAMSADE, Université Paris-Dauphine

Version of 22nd May, 2019

LAMSADE
UMR CNRS 7243
laboratoire d'analyse et modélisation de systèmes pour l'aide à la décision

DAUPHINE
UNIVERSITÉ PARIS

PSL
RESEARCH
UNIVERSITY

What's a GUI?

- GUI?
- *An* interface between the user and your program
- Assembles graphical components
- Components are provided
- By the OS or language-specific
- You can design your own components
- Own components: rare!

What's a GUI?

- GUI? Graphical User Interface
- *An* interface between the user and your program
- Assembles graphical components
- Components are provided
- By the OS or language-specific
- You can design your own components
- Own components: rare!

Control in GUIs

- IoC?
- AKA The Hollywood principle: "Don't call us, we'll call you"
- Usually: you control the application flow
- From `main`, go to this method, ...
- With a GUI, the GUI controls the flow
- Most of the time: do "nothing"
  - Meaning: nothing you program
  - Refresh components when necessary, ...
  - Done by the GUI library
- Your program acts when the user clicks a button, ...
- Called an event

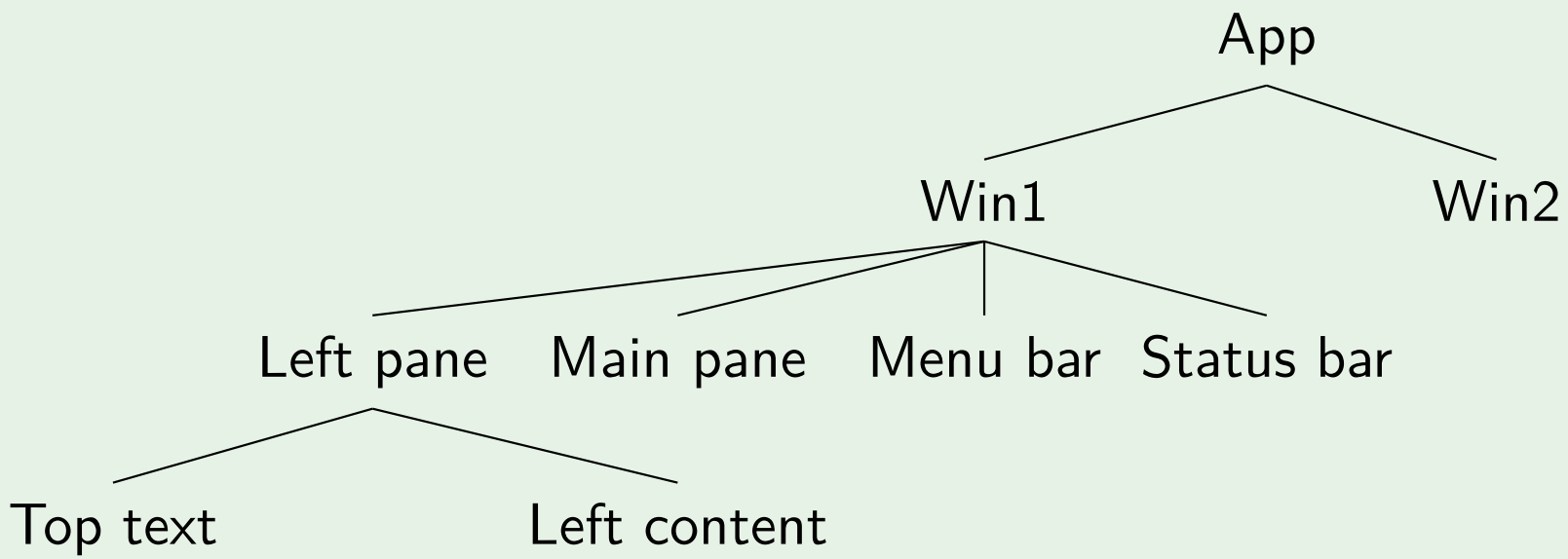
Control in GUIs

- IoC? Inversion of control
- AKA The Hollywood principle: "Don't call us, we'll call you"
- Usually: you control the application flow
- From `main`, go to this method, ...
- With a GUI, the GUI controls the flow
- Most of the time: do "nothing"
  - Meaning: nothing you program
  - Refresh components when necessary, ...
  - Done by the GUI library
- Your program acts when the user clicks a button, ...
- Called an event

Hierarchical design

Components are typically assembled into a tree

Example (A tree of GUI components)



Observer situation

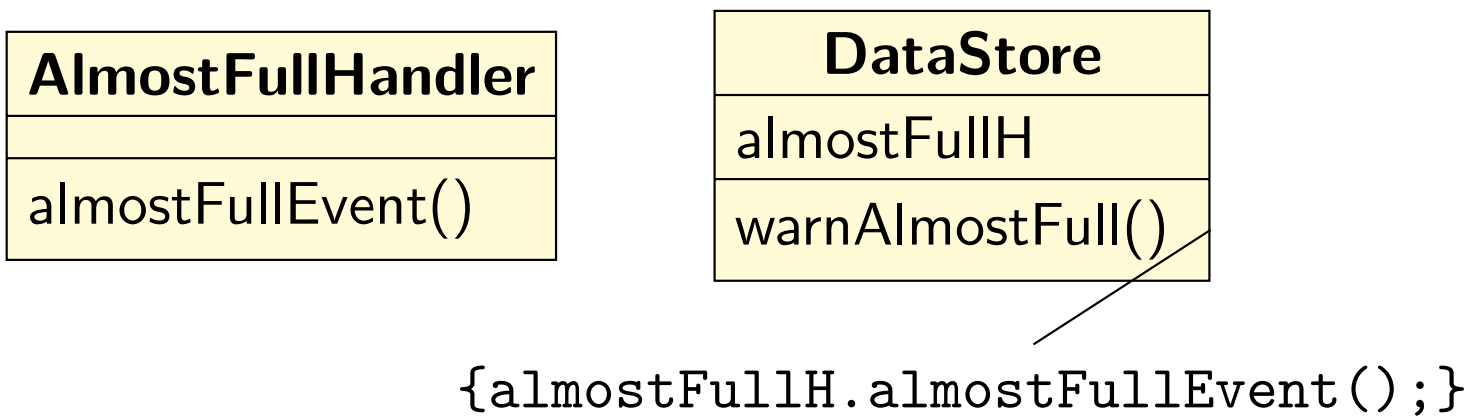
Situation: One object must advertise events to other objects

Example (Advertiser)

- A button advertises that it has been pressed
- A data store warns that it has almost no space left

Simplest approach:

- The advertiser knows the interested objects and calls them

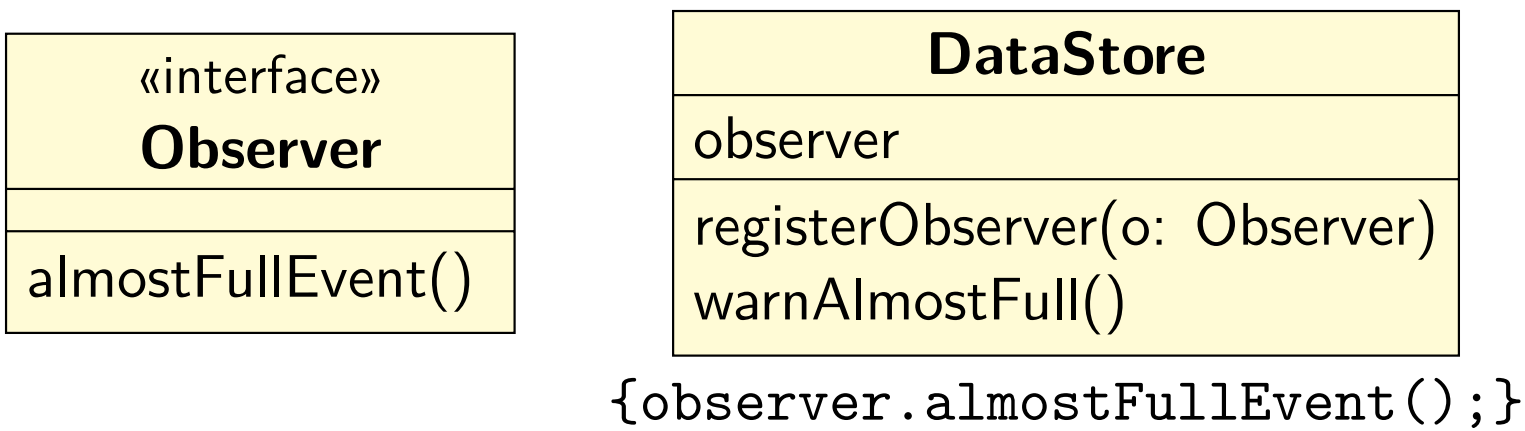


Observer pattern

Situation:

- You want one object to advertise events to observers
- You do not know all possible observers at development time
- You want loose coupling between objects

- The observed object calls any observer object
- Observer is an interface
- Observers must register to the observed object



Programming a GUI

Simple approach:

- Assemble your graphical components into a tree
- Register observers on components
- Start the GUI event loop

GUIs in Java

- Abstract Window Toolkit (AWT): oldest and most basic
- Uses native GUI components
- Swing: higher-level controls
- Reuses part of AWT to plug into OS
- Paints its own controls (consistent look and feel across OSes)
- JavaFX: replacer of Swing
- Declarative language for definition of the GUI; animations; CSS styles
- Paints its own controls
- AWT, Swing, JavaFX standardized in Java
- Standard Widget Toolkit (SWT): Eclipse's alternative to AWT/Swing
- JFace: Eclipse's advanced library that supplements SWT

Characteristics

- A lie in the name: ?
- JavaFX is newer and (possibly) more popular, and (really) standard
- SWT uses native graphical components (like AWT)
- JFace/SWT has high-level components (like Swing or JavaFX)

$\Rightarrow$ Native look and feel with high-level components

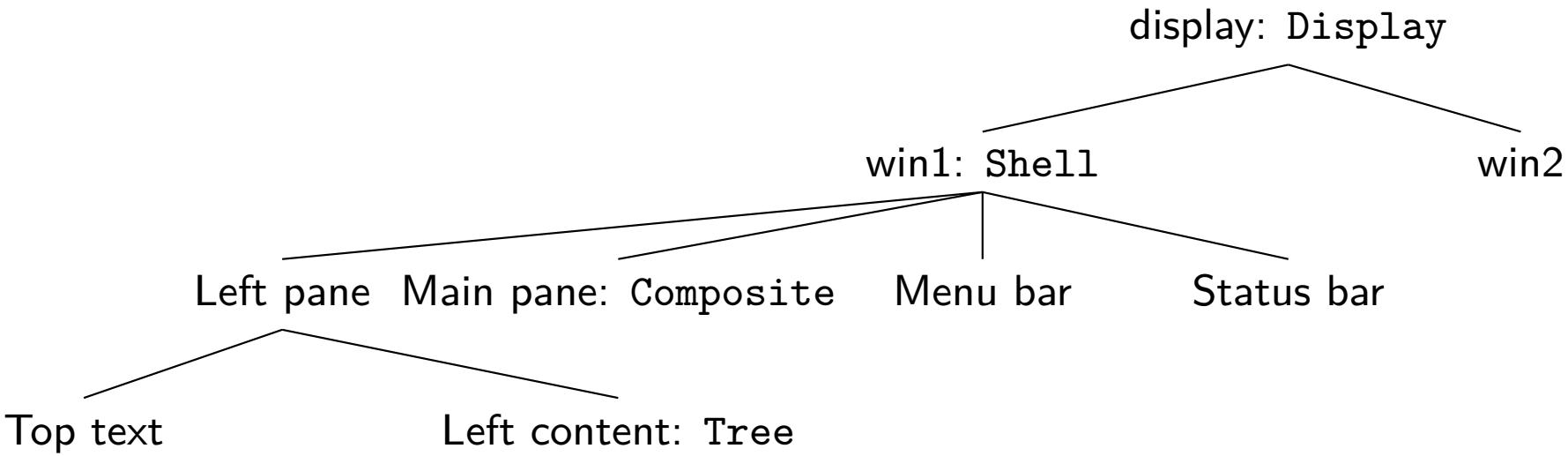
Characteristics

- A lie in the name: it's not standard!
- JavaFX is newer and (possibly) more popular, and (really) standard
- SWT uses native graphical components (like AWT)
- JFace/SWT has high-level components (like Swing or JavaFX)

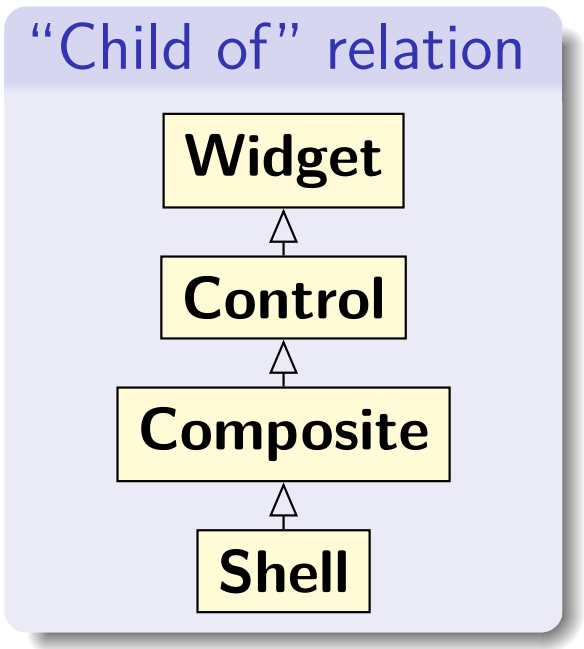
$\Rightarrow$ Native look and feel with high-level components

Hierarchical structure

Example:



Core concepts



- `Widget`: GUI component; can send events; must be disposed
- `Control`: may be put into a composite; has bounds
- `Composite`: can contain child controls
- `Shell`: top-level window connected to a display
- `Display`: represents the connection to the native display

General structure

- Create the display
- Create a shell bound to the display
- Put the desired controls into the shell
- Run the event loop

```
Display display = new Display();
Shell shell = new Shell(display);
// populate shell...
shell.open();
while (!shell.isDisposed()) {
  if (!display.readAndDispatch()) {
    display.sleep();
  }
}
display.dispose();
```

Create and position a control

- Use its constructor
- Provide the parent composite, and the style bits
- `new Button(shell, SWT.PUSH | SWT.WRAP);`
- A control has a position (relative to its parent) and a size
- Bounds of the control: rectangle that contains it entirely ($>$ client area)
- `control.setBounds(10, 0, 50, 20);` (location: (10, 0); size: 50×20)
- Hard to determine the right size! Depends on text content, ...
- Compute preferred size: `control.computeSize(...)`
- Set to preferred size: `control.pack();`

Listen to widgets

- Implement the right *Listener interface
- Example: `SelectionListener`
- Add your implementer to the observers of the widget
- `button.addSelectionListener(myListener);`
- When the widget fires the corresponding event, your listener is called

Dispose resources

- Widgets relate to native resources
- Must be explicitly disposed of
- Rule: if you create it, you must dispose of it
- `composite.dispose()` $\Rightarrow$ disposes of all the children

Items in controls

- Some controls must be populated with items
- Example: `TableItem` represents a row of a `Table`
- `Item`s are widgets but not controls: their position is partly controlled by their parent

Widgets in practice

- These are only general guidelines
- Each widget has its own specificities
- Choose your widget using the screenshots (see doc)
- Then check the javadoc or snippets
- Then check what JFace provides to help

Practicalities

- SWT through Maven: some trick required, see doc
- UI Thread: the thread that created the display
- Access GUI widgets: only through the UI thread
- Event loop must be run from the UI thread

Licence