Interactr

Iteration 3

Jelle de Coninck, Hannes De Smet, Bruno Vandekerkhove, Shani Vanlerberghe

May 12, 2018

KULeuven

Table of contents

Design

Domain Model

- `Low representational gap`
- Main classes : `Diagram`, `Message`, `Party`
- Business logic completely separate from UI

Domain Model

Put diagram of domain model here

Entry Point - Controller

Everything starts in `Controller` :

→ creates a `Window` (extends `CanvasWindow`)
→ associates the window with `PaintBoard` (draw events)
→ associates the window with `EventHandler` (key/mouse events)

Both support `Protected Variations`. Both encapsulate *awt* package.

Controller - Underlying Structure

The `Controller` keeps track of a list of `SubWindows`.

A `SubWindow` can be a `DiagramWindow` or a `DialogBox`.

Each `SubWindow` has a frame.

Each `SubWindow` is '*activated*' by putting it at the front of the window list.

SubWindows - Diagram Windows & Dialog Boxes

A `DiagramWindow` has two `DiagramViews`.
A `DialogBox` has a list of `Controls`.

A `Diagram` is directly altered by these classes.

Synchronization is done with the `Observer` pattern.
Since `Diagram` knows when it changes, it keeps a list of `DiagramObservers` and notifies them (`Information Expert`).
$\rightarrow$ disadvantage : coordinates aren't synchronised

Event Handling - How it all Starts

→ Every mouse or key event goes from `Window` → `EventHandler`.

→ `EventHandler` interprets the events and transforms them into `Command` instances.

→ `Command` is forwarded to `Controller`.

Note : if event is a *mouse press*, `EventHandler` asks the `Controller` to activate the `SubWindow` at that coordinate first.

Event Handling - Executing Commands

Precondition : `SubWindow` is active

→ Every `Command` can be handled by a `CommandHandler`.

→ `DiagramWindow`, `DiagramView` and `DialogBox` extend the`CommandHandler` class.

→ The first `CommandHandler` is the active `SubWindow`.

→ Every `CommandHandler` either deals with the `Command` or - on failure - passes it to the next `CommandHandler` in its chain.

Patterns used : `Command`, `Visitor` and `Chain of Responsibility`.

Drawing - Paint Board

The `PaintBoard` encapsulates the *awt* package. It is passed along to any class responsible for drawing. Drawing is done '*on*' the paint board.

- manages clip rect (to prevent overflowing)
- allows color changes
- ...

$\rightarrow$ it is a `Facade`

Drawing - Displaying Elements

Each class that displays something draws itself (`Information Expert`) :

- `SubWindow` draws title bar, close button, frame.
- `DiagramView` draws diagram with messages, parties, ...
  $\rightarrow$ use of `Visitor` pattern to generate figures (representations) for these diagram components.
  $\rightarrow$ these figures are `Flyweights`.
  $\rightarrow$ separation of UI / domain logic without use of type checking (i.e. `instanceof`).
- `Control` draws itself, eg. text for label.

Extensibility

Event Handling

`Command` pattern eases undoing, allows delayed processing, use by menu items, ... When using new types of events (eg. speech recognition) only `EventHandler` has to change.

Small disadvantages ;

- many new types of `CommandHandler`
  $\rightarrow$ bloated `Command` abstract class (`Visitor` pattern discussion)
- if `Chain of Responsibility` is long
  $\rightarrow$ wasted processing

Testing

Testing

Coverage here!

Project Management

Time management

- weekly meeting with assistant
- +- 40 hours per person
- designed, refactored, tested, redesigned, ... in no particular order

Demonstration