C# PROGRAMMING 2007

Command Pattern
GDI +

Thomas J. Fuchs
Overview

Command Pattern
- Class Structure
- Collaboration
- Benefits
- Undo & Redo
- Practical Example

GDI+
- What is GDI+
- The Graphics Object
- Painting & Drawing
- Regions And Clipping
- Practical Example
The Command Pattern

“Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.”
Class Structure

- Client
- Invoker
- Command
  - Execute()
- Receiver
  - Action()
- ConcreteCommand
  - Execute()
  - state
  - receiver -> Action()
Classes

**Command**
declares an interface for executing an operation.

**ConcreteCommand** *(PasteCommand, OpenCommand)*
defines a binding between a Receiver object and an action.
implements Execute by invoking the corresponding operation(s) on Receiver.

**Client** *(Application)*
creates a ConcreteCommand object and sets its receiver.

**Invoker** *(MenuItem)*
asks the command to carry out the request.

**Receiver** *(Document, Application)*
knows how to perform the operations associated with carrying out a request. Any class may serve as a Receiver.
Collaborations

- Client creates a **ConcreteCommand** object and specifies its receiver.
- An **Invoker** object stores the ConcreteCommand object.
- The invoker issues a request by calling **Execute()** on the command.
- The ConcreteCommand object invokes operations on its receiver to carry out the request.
Command - Benefits

- Pattern replaces function pointers *(passing functions as parameters)*

- Pattern allows a class to call a receivers routine without knowledge of it. Gives high degree of decoupling between caller and callee.

- Security can be enhanced by logging user actions through centralized Command objects.
Command - Benefits

- Support logging changes so they can be reapplied in case of a system crash.
  - Simply augment the Command interface with **load** and **store** operations to keep a persistent log of changes.

- Structure a system around high-level operations built on primitives operations.
How-To Undo

- The Execute() operation can store the current state when it is called.

- An Unexecute() operation must be added so that the Command can restore the state to where it was before Execute() was called.

- Executed commands are stored in a history list, so unlimited-level undo and redo is possible by simply traversing the list.
**Undo & Redo**

Conceptually, the command history looks like the following graph. Each circle represents a command object.
Undo & Redo

Unexecute() present
Undo & Redo

present
Undo & Redo

past

future

present
Undo & Redo

execute()
Undo & Redo

Diagram showing the concept of past, present, and future.
And... The Consequences

- The object that invokes the operation and the object that knows how to perform it are decoupled.
- Commands are first-class objects. So they are able to be manipulated and extended.
- Commands can be assembled into a composite command. Example is the MacroCommand.
- It is easy to add new commands, because you don’t have to change existing classes.
PRACTICAL EXAMPLES ...
Overview

Command Pattern
- Class Structure
- Collaboration
- Benefits
- Undo & Redo
- Practical Example

GDI +
- What is GDI+
- The Graphics Object
- Painting & Drawing
- Regions And Clipping
- Practical Example
What is GDI+?

- Windows graphics design interface GDI under Win32
- GDI+ is an advanced version of GDI only available in the Common Language Runtime.
- Provides the basic tools necessary to be able to do graphics on controls
The Graphics Object

- The System.Drawing.Graphics object represents the drawing surface, or canvas, on which to draw.
- The graphics object can be obtained in three ways:
  - Directly, using this.CreateGraphics()
  - From a paint event handler object or,
  - From an image.
Painting On Controls

- Mostly override the standard paint handlers:
  - protected void OnPaint(PaintEventArgs PE);
  - protected void OnPaintBackground(PaintEventArgs PE);

- Double Buffering
  - Set styles UserPaint, AllPaintingInWmPaint, DoubleBuffer.
Painting on Controls Cont…

- Base Methods are called at the appropriate points
- Not always necessary to do this if you are doing a lot of heavy graphics routines yourself.
- If you are doing animation graphics, best to do them in only one of the paint methods. Usually OnPaint.
Types of drawing tools

- Pens
- Images
- Text
- Brushes
  - SolidBrush
  - HashBrush
  - TextureBrush
  - LinearGradientBrush
  - PathGradientBrush
Regions And Clipping

- You can define custom areas to draw using regions and setting clip areas.
- Regions affect the whole form.
  - `this.Region = new Region();`
- Clip areas affect the graphics object.
  - `g.SetClip();`
- Both are usually defined using Graphics paths.
PRACTICAL EXAMPLES ...
QUESTIONS?
GDI+ Material by Andrew Scott, 2001
Modified by Keith Mitchell, 2002
Modified by Thomas Fuchs, 2007
Resources

- Course Homepage:
  - se.inf.ethz.ch/teaching/ss2007/251-0290-00

- Exercise Material:
  - www.inf.ethz.ch/personal/thomas.fuchs