The Controller Pattern and Related Patterns

CMPT 371
Fall 2004
J.W. Benham
Controller

• **Problem:** Who should be responsible for handling an input system event?

• **Solution:** Assign to a class that represents either
  – The overall system, device, or subsystem
  – A use case scenario
    • `<use-case-name>Manager`
    • `<use-case-name>Controller`
    • `<use-case-name>Session`
Controller Example

- Who should handle input events for the TranslateSearchString use case?
- TranslateSearchStringManager – a new control class.
  - Manager gets list of supported search engines from active instance of TranslationEnvironment
  - Manager displays user interface with selection of supported search engines
  - User enters search string and selects search engines via user interface.
  - For each search engine selected, manager calls appropriate Translator.
  - Manager reports results via user interface.
Benefits of Controllers

• Increased potential for reuse
• Pluggable interfaces
• Reasoning about state of use case
  – Example – keep track of number of login attempts
  – Example – determine how to handle exceptions thrown by entity objects
Bloated controllers

• Controller directly handles too many areas of responsibility (low cohesion)

• Signs of a bloated controller
  – Single controller class for all system events of which there are many
  – Controller directly performs tasks required to respond to system event
  – Controller has many attributes to maintain knowledge about state of system
Curing bloated controllers

• Add more controllers
  – Possibly a separate controller for each use case

• Design controller to delegate more responsibilities to entity objects.
  – Calls for rethinking assignment of responsibilities
Command Pattern (GoF)

- **Problem:** How can an object (typically a UI object) issue a request without knowledge of the receiver of the request or the operation being requested?

- **Solution:** Turn the request itself into an object
  - extends (abstract) Command class (with abstract method `execute()`)
  - Implements Command interface (with unimplemented `execute()` method).
Command Pattern in General
Command Pattern Example

[Diagram showing the Command Pattern with classes and methods]
Façade Pattern (GoF)

• **Problem:** How can one minimize communication and dependencies between subsystems?
  – Also, how can one minimize communication dependencies with external actors?

• **Solution:** Create a *façade* class that provides the sole interface between the subsystem and the rest of the system.
General Façade Pattern (also illustrates Layers)
Façade Pattern Example (with implied lower layers)