Multiple Window Systems

- Want portability to different window systems
  - Similar to multiple look-and-feel problem, but different vendors will build widgets differently
- Solution:
  - Define abstract class Window, with basic window functionality (e.g., draw, iconify, move, resize, etc.)
  - Define concrete subclasses for specific types of windows (e.g., dialog, application, icon, etc.)
  - Define WindowImp hierarchy to handle window implementation by a vendor
Implementation

Bridge Pattern

• Name
  – Bridge or Handle or Body

• Applicability
  – Handles abstract concept with different implementations
  – Implementation may be switched at run-time
  – Implementation changes should not affect clients
  – Hide a class’s interface from clients

• Structure: use two hierarchies
  – Logical one for clients,
  – Physical one for different implementations
Bridge Pattern

- Consequences:
  - Decouple interface from implementation and representation
  - Change implementation at run-time
  - Improve extensibility
    - Logical classes and physical classes change independently
    - Hides implementation details from clients
      - Sharing implementation objects and associated reference counts
Supporting User Commands

• Support execution of Lexi commands
  – GUI doesn’t know
    • Who command is sent to
    • Command interface

• Complications
  – Different commands have different interfaces
  – Same command can be invoked in different ways
  – Undo and Redo for some, but not all, commands (print)

Supporting User Commands (cont’d)

• An improved solution
  – Create abstract “command” class
  – Create action-performing glyph subclass
  – Delegate action to command

• Key ideas
  – Pass an object, not a function
  – Pass context to the command function
  – Store command history
Command Objects

Command Pattern

- Name
  - Command or Action or Transaction

- Applicability
  - Parameterize objects by actions they perform
  - Specify, queue, and execute requests at different times
  - Support undo by storing context information
  - Support change log for recovery purposes
  - Support high-level operations
    - Macros
Command Pattern

• Consequences:
  – Decouple receiver and executor of requests
    • Lexi example: Different icons can be associated with the same command
  – Commands are first class objects
  – Easy to support undo and redo
    • Command must have method to check whether it’s reversible
    • Must add state information
  – Can create composite commands
    • Editor macros
  – Can extend commands more easily
Command Pattern

- Implementation notes
  - How much should command do itself?
  - Support undo and redo functionality
    - Operations must be reversible
    - May need to copy command objects
    - Don’t record commands that don’t change state
  - Avoid error accumulation in undo process

Comparing Objects

- Java has two designs for objects that can be (totally) ordered
  - These are things for which sorting makes sense
  - E.g., strings, integers, etc.
Comparable

```java
public interface Comparable {
    // Returns negative integer, zero, or a positive integer if this
    // object is less than, equal to, or greater than o.
    public int compareTo(Object o);
}
```

- Advantages and disadvantages?
  - Can only implement one compareTo operation
  - No extra levels of indirection; objects know how to compare themselves

Comparator

```java
public interface Comparable {
    int compare(Object o1, Object o2);
}
```

- Advantages and disadvantages?
  - Can have multiple comparison operations
  - An example of delegation
  - Comparable needs to know innards of your objects
    - Can make the Comparable implementer an inner class
  - Extra indirection; more objects floating around
Pattern Hype

- Patterns get a lot of hype and fanatical believers
  - We are going to have a design pattern reading group, and this week we are going to discuss the Singleton Pattern!

- Patterns are sometimes wrong (e.g., double-checked locking) or inappropriate for a particular language or environment
  - Patterns developed for C++ can have very different solutions in Smalltalk or Java