Lecture 4

Design Patterns

Announcement

- Project proposal is due tomorrow at 8 pm.
- Presentation sign-up sheet is available on the blackboard.

Announcement

- Students in Software Engineering (SSE)
  - http://www.edge.utexas.edu/sse/
- Software Engineering Reading Group (SERG)
  - https://users.ece.utexas.edu/~miryung/teaching/SE-Seminar.Spring09.html

Announcement

- Don't forget to put a header [EE382V] when emailing me.
- Please cc TA when you send me an email for _all_ your correspondences.

Announcement

- Question: Can you see your reading assignment grades on the blackboard?
- Please do not attach a document.

Today's Presentation

- Skeptic: Jason Vanfickell

Reprise: What is "Engineering"?

Definitions abound. They have in common:

Creating cost-effective solutions ...

... to practical problems ...

... by applying scientific knowledge ...

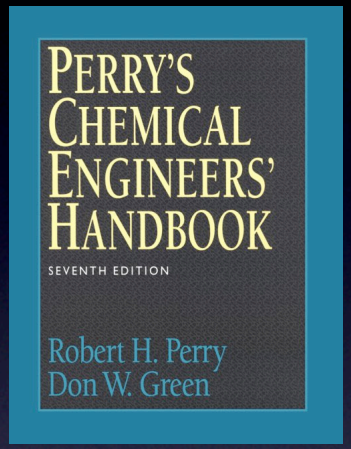
... building things ...

... in the service of mankind

Engineering enables ordinary people
to do things that formerly required virtuosos

Slide from Mary Shaw @ CMU

Example handbook



- Contents
  - Chemical and physical property data
  - Fundamentals (e.g. thermodynamics)
  - Processes (the bulk of the book)
    - heat transfer operations
    - distillation
    - kinetics
    - liquid–liquid
    - liquid–solid
    - etc.
  - Materials of construction
  - Waste management

Slide from Rob DeLine @ Microsoft Research

Other Precedents

- Polya's How to Solve It
  - Catalogs techniques for solving mathematical (geometry) problems
  - Two categories: problems to prove, problems to find/construct
- Christopher Alexander's books, e.g. A Pattern Language
  - Saw building architecture/urban design as recurring patterns
  - Gives 253 patterns as: name; example; context; problem; solution
- Pattern languages as engineering handbooks
  - Hype aside, it's about recording known solutions to problems
  - Pattern languages exist for many problems, but we'll look at design
  - Best known: Gamma, Helm, Johnson, Vlissides ("Gang of four")
    Design Patterns: Elements of reusable object-oriented software
  - Notice the subtitle: here, design is about objects and their interactions

Slide from Rob DeLine @ Microsoft Research

What type of paper is Gamma et al.?

- Concept/ idea paper
- Survey paper
-

What type of paper is Gamma et al.?

- idea paper
- survey paper
- evaluation?
  - case studies / experience reports

Reconciling review guidelines for a survey / idea paper?

- What is the discussed problem?
- What are main ideas?
- Why the proposed ideas are novel?
- What are the limitations & strengths of the proposed ideas & framework?
- What are future directions?

Why do we need Design Patterns?

- To reuse proven expert design
- To communicate design easily with other engineers
- To simplify design
- To leverage existing design template
- To reorganize / refactor design
- To allow composing software out of building blocks
- To help novice engineers understand software

Why do we need Design Patterns?

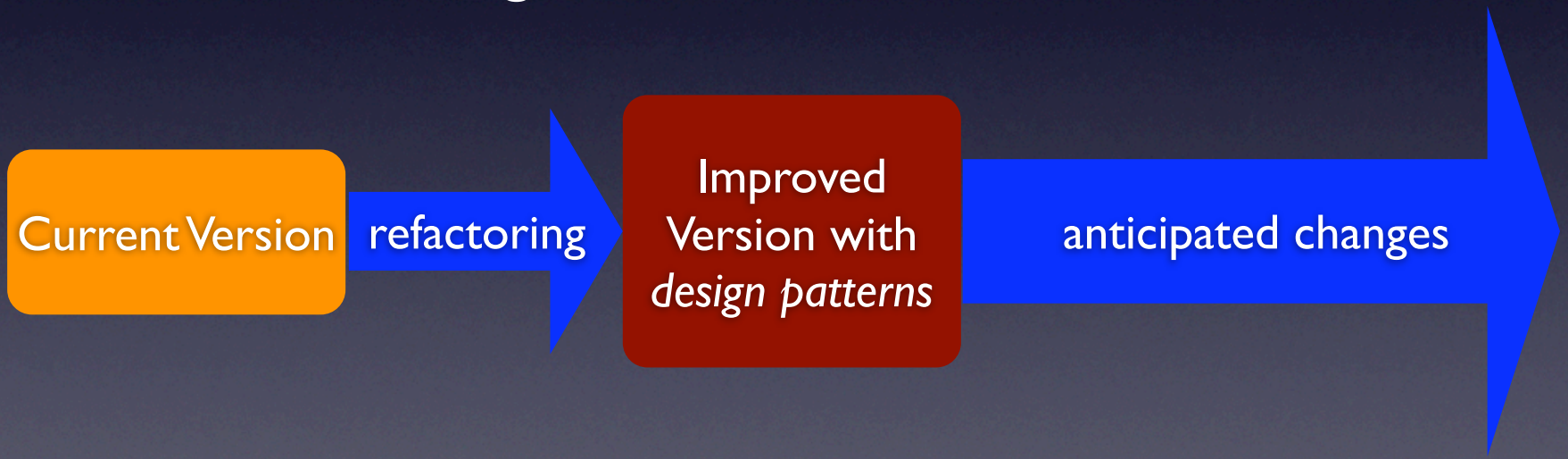
1. Abstract design experience => a reusable base of experience
2. Provide common vocabulary for discussing design
3. Reduce system complexity by naming abstractions => reduce the learning time for a class library / program comprehension

Why do we need Design Patterns?

4. Provide a target for the reorganization or refactoring of class hierarchies

Why do we need Design Patterns?

4. Provide a target for the reorganization or refactoring of class hierarchies



Which aspects of design does *Gang Of Four* discuss?

- a class or object collaboration and its structure

Categorizing Design Patterns

	purpose			
		creational	structural	behavioral
scope	class	factory method	adapter (class)	interpreter
				template method
	object	abstract factory	adapter (object)	chain of responsibility
		builder	bridge	command
		prototype	composite	iterator
		singleton	decorator	mediator
			facade	memento
			flyweight	observer
			proxy	state
				strategy
				visitor

WHY Categorizing Design Patterns?

- to refer to families of related patterns
- to learn and organize patterns in the catalog
- *to find new patterns*

Design Patterns

- Intent
- Motivation
- Applicability
- Participants
- Collaborations
- Diagrams
- Consequences
- Implementation
- Examples
- See Also

Problem / Goal

Solution

What types of changes are easier to implement due to this design

Case studies

Example: Abstract Factory

Problem / Goal
: Having an explicit dependencies on concrete product classes makes it difficult to change product types or add a new product type.

Example: Abstract Factory

Typical OOP program hard-codes type choices

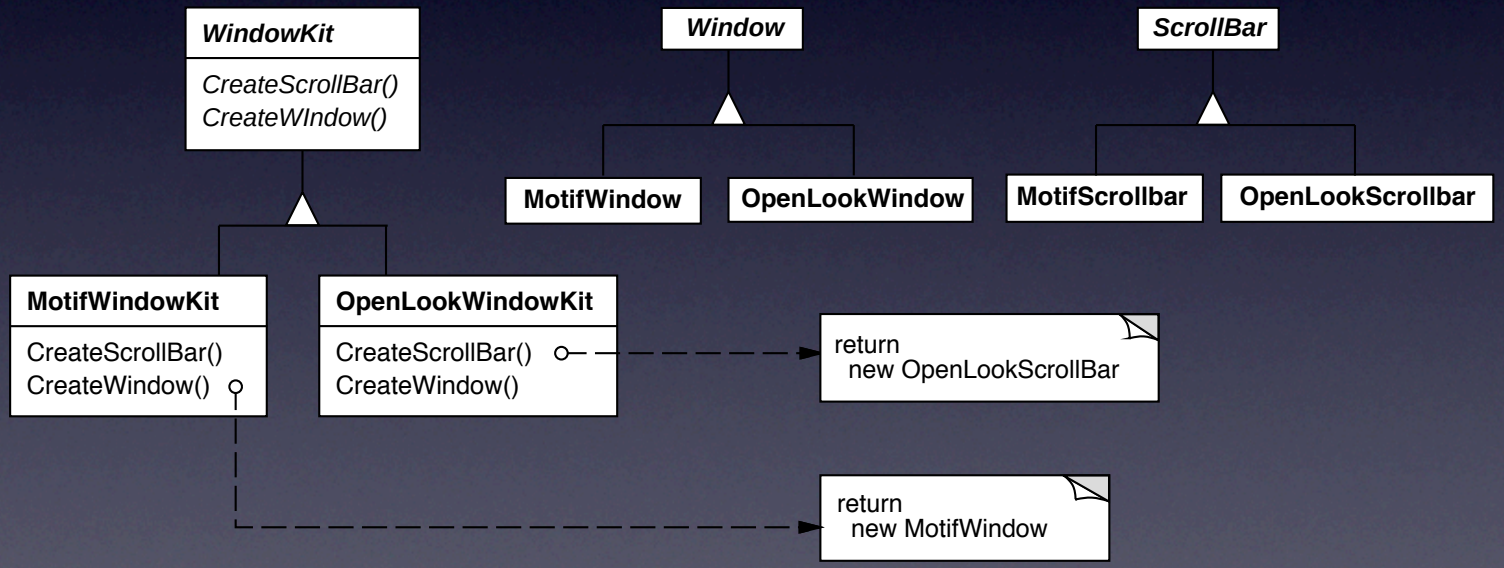
```
void AppInit () {
#if Motif
  Window w = new MotifWindow(...);
  ScrollBar b = new MotifScrollBar(...);
#else if OpenLook
  Window w = new OpenLookWindow(...);
  ScrollBar b = new OpenLookScrollBar
  (...);
#endif
  w.Add(b);
}
```

We want to easily change the app's "look and feel", which means calling different constructors.

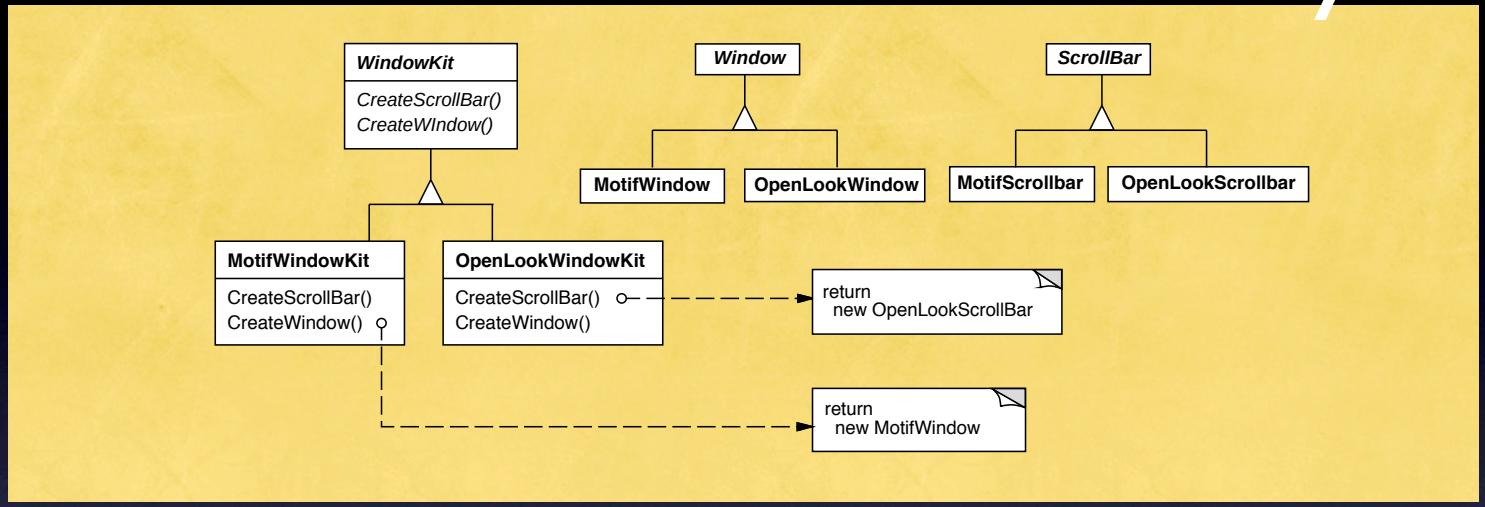
Solution: Abstract Factory

Solution
: Wrap the constructors in factory methods

Solution: Abstract Factory



Solution: Abstract Factory



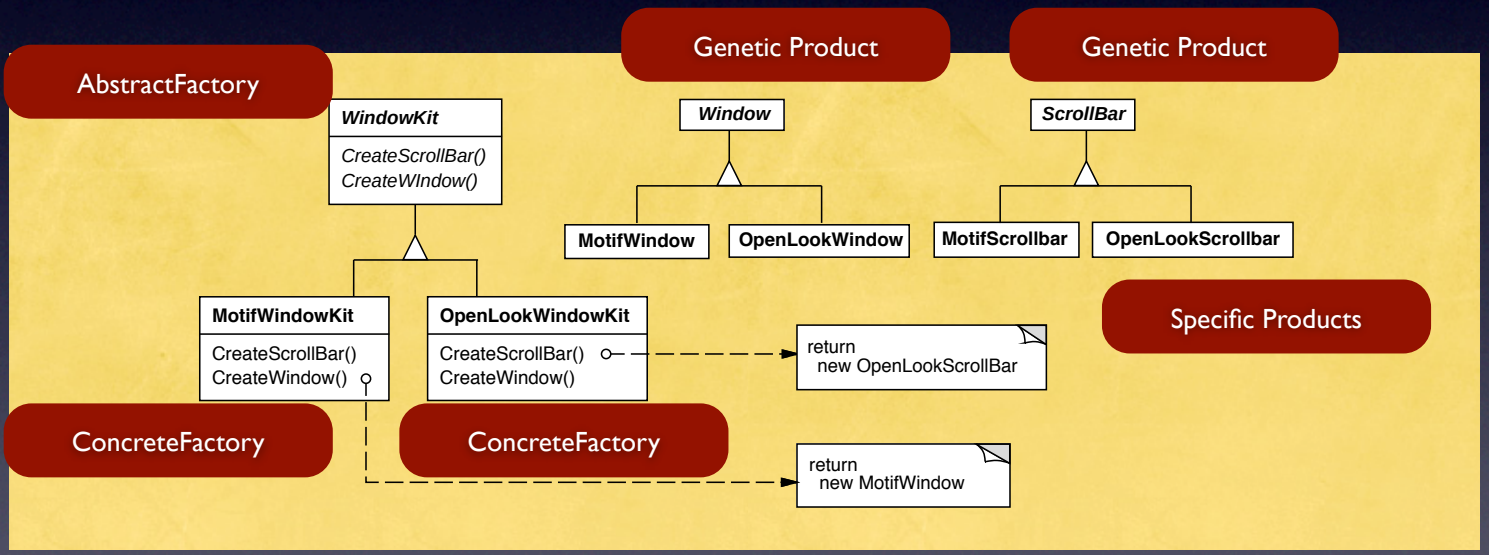
Client Code

```
WindowKit kit = new MotifWindowKit();
kit.appInit();
```

```
class WindowKit {
  WindowKit ();
  Window CreateWindow (...);
  ScrollBar CreateScrollBar (...);


void appInit () {
  Window w = CreateWindow(...);
  ScrollBar b = CreateScrollBar(...);
  w.Add(b);
}
```

Participants



What types of changes can you anticipate?

- What happens if we have multiple types of windows?
- What happens if we need different types of windows that take different arguments?
- What happens if we want to define a window as combination of window, scroll bar and button

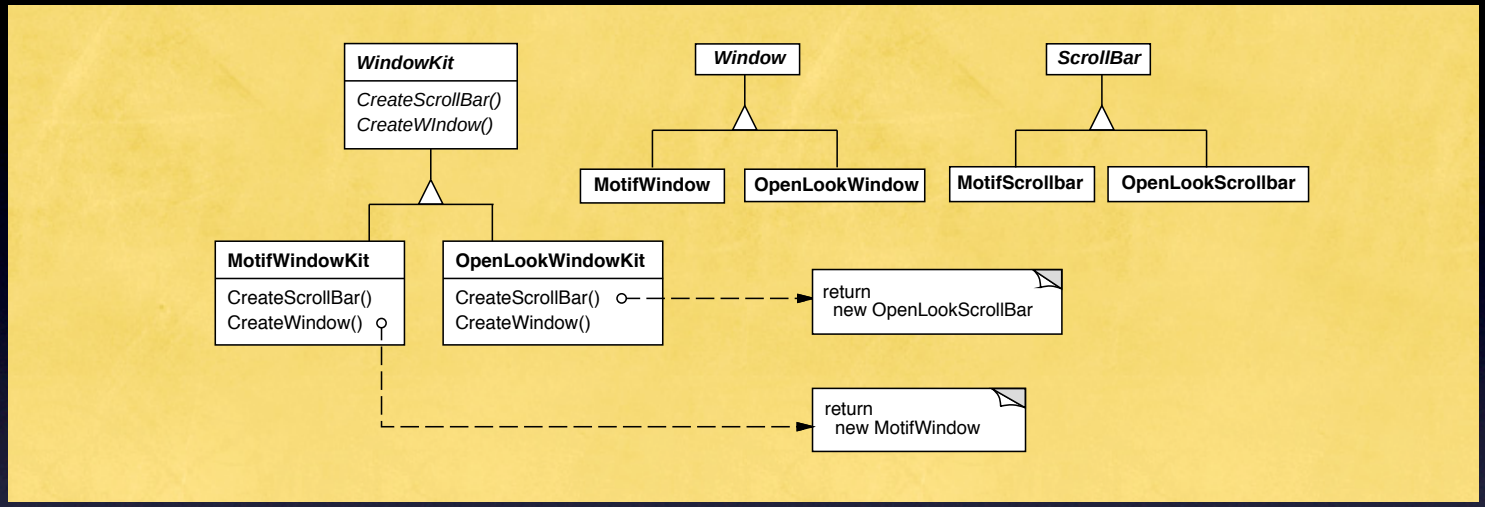
What types of changes can you anticipate?

- Adding a different look and feel such as MacWindowKit
- Adding a new type of object such as a button as a part of WindowKit

How about adding a different look and feel such as MacWindowKit?

Name:

UT EID:



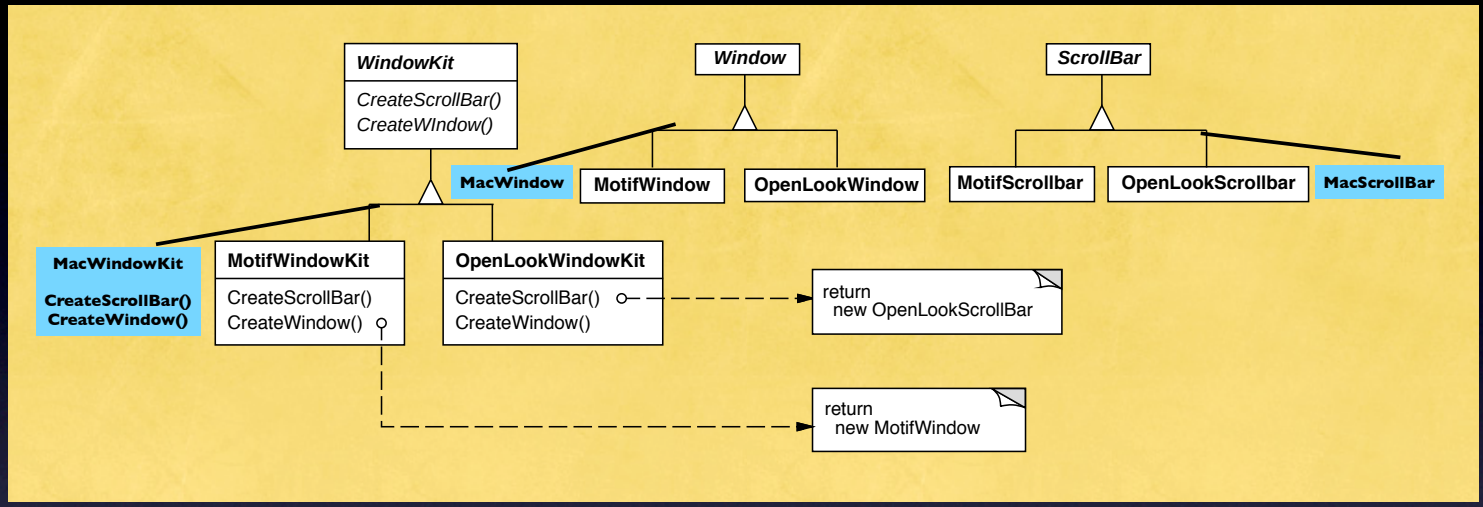
Client Code

```
WindowKit kit = new MotifWindowKit();
kit.appInit();
```

```
class WindowKit {
  WindowKit ();
  Window CreateWindow (...);
  ScrollBar CreateScrollBar (...);


void appInit () {
  Window w = CreateWindow(...);
  ScrollBar b = CreateScrollBar(...);
  w.Add(b);
}
```

How about adding a different look and feel such as MacWindowKit?



Client Code

```
WindowKit kit = new MacWindowKit();
kit.appInit();
```

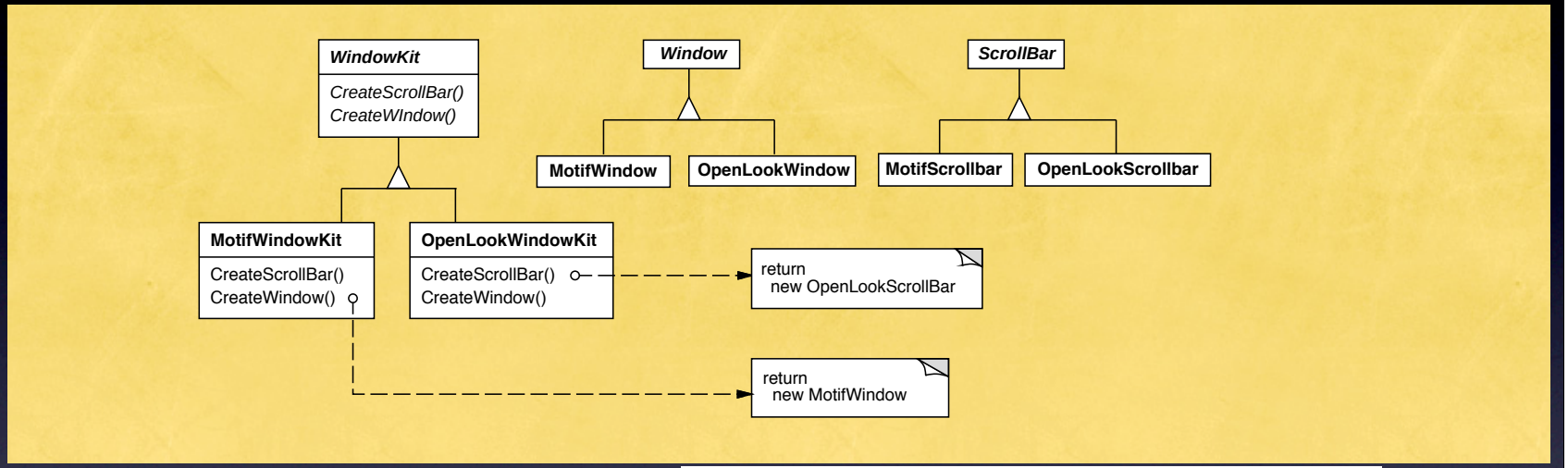
```
class WindowKit {
  WindowKit ();
  Window CreateWindow (...);
  ScrollBar CreateScrollBar (...);


void appInit () {
  Window w = CreateWindow(...);
  ScrollBar b = CreateScrollBar(...);
  w.Add(b);
}
```

How about adding a new type of object such as as button?

Name:

UT EID:



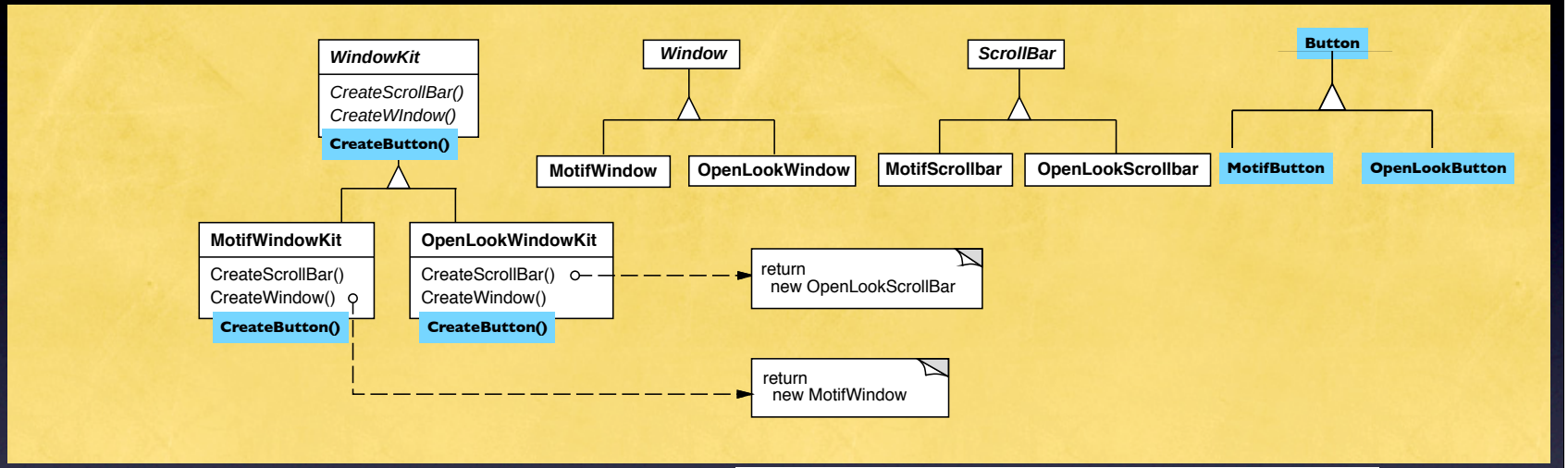
Client Code

```
WindowKit kit = new MotifWindowKit();
kit.appInit();
```

```
class WindowKit {
  WindowKit ();
  Window CreateWindow (...);
  ScrollBar CreateScrollBar (...);


void appInit () {
  Window w = CreateWindow(...);
  ScrollBar b = CreateScrollBar(...);
  w.Add(b);
}
```

How about adding a new type of object such as a button?



Client Code

```
WindowKit kit = new MotifWindowKit();
kit.appInit();
```

```
class WindowKit {
   WindowKit ();
   Window CreateWindow (...);
   ScrollBar CreateScrollBar (...);
   Button CreateButton(...);

void appInit () {
   Window w = CreateWindow(...);
   ScrollBar b = CreateScrollBar(...);
   Button bt = CreateButton(...);
   w.Add(b);
   w.Add(bt);
}
```

Problem: uniformly access sequential data

We don't want to make data access specific to the interface

```
class List {
  List ();
  void Add (object element);
  void Remove (object element);
  object Head ();
  List Rest ();
}
...
for (List scan = myList; scan != null; scan = scan.Rest()) ...
```

We also want multiple simultaneous traversals, different traversal orders

Solution: iterator

Put the traversal data in its own class

```
class List { ...
  ListIterator GetIterator();
}

class ListIterator {
  object GetCurrent ();
  bool Done ();
  void MoveToNext ();
}

for (ListIterator scan = myList.GetIterator();
        ! scan.Done();
        scan.MoveToNext() ) ...
```

Problem: We need many instances

For page layout, we need many instances of letters

```
class Glyph {
  Page myPage;
  Row myRow;
  Column myColumn;
  void PrintMe ();
}
```

For a 10-page doc, with 1000 glyphs per page, representation needs 3*32*10*1000 = 1 MB !

Solution: Flyweight

Use a single object per unique glyph

- move all state outside the object itself
- treat the "object" as a mathematical value (functional programming)

```
class Glyph {
  void PrintMe (Page page, Row row, Column col);
}
```

Problem: Too many object interconnections

In asynchronous programs, events have many reactions

```
void FontSelectCallback (FontDisplay fontList) {
  Font font = fontList.GetSelectedItem();
  if ( ! font.SupportsBold())
        boldSelector.Deactivate();
  if ( ! font.SupportsItalic())
        italicSelector.Deactivate();
  ...
}
```

If this dialog box gets a new element, we must update many classes

Solution: mediator

Centralize the interconnection code

```
void FontSelectCallback (FontDisplay fontList) {
  fontList.mediator.FontChanged(fontList.GetSelectedItem
  ());
}

class FontMediator {
  void FontChanged (Font newFont) {
        if ( ! font.SupportsBold())
              boldSelector.Deactivate();
        if ( ! font.SupportsItalic())
              italicSelector.Deactivate();
  }
}
```

From your reviews

- " They do leave out the implementation complexities of the patterns and admit that ***some of the patterns would not be clear to use until after the first implementation of a design was complete.*** Nor do they claim that design patterns can be used in every part of software implementation. The technique, however, has been and will continue to be an important part of Object-Oriented design. "

From your reviews

- It is debatable if the "class" jurisdiction is unique enough to be classified as a domain separate from "Object."

From your reviews

- I think that the idea of design patterns is an adequate idea as prelude to software architecture, but the mature expression of this concept is software architecture. In my opinion, there is even more utility in application of general architectural tools. Software architecture offers more reusability because ***it includes not only design but implementation issues.***

From your reviews

- First, this technique is strongly linked to object-oriented programming. Because of the verbose nature of object- oriented naming conventions, this may be less dramatic of a step towards common meta-language than it would be if it were applied to environments such as LISP.

From your reviews

- However, there is also a drawback to employing design patterns: ***it makes a design more complicated as more abstractions and indirections are often introduced by design patterns. These extra additions to the "core" design structures clutter the software's design and its implementation.***

Take away message

- Design patterns are reusable solutions for well-known problems.
- Design patterns are often a target for refactoring.
- Design patterns make it easier to add ***particular types of changes.***