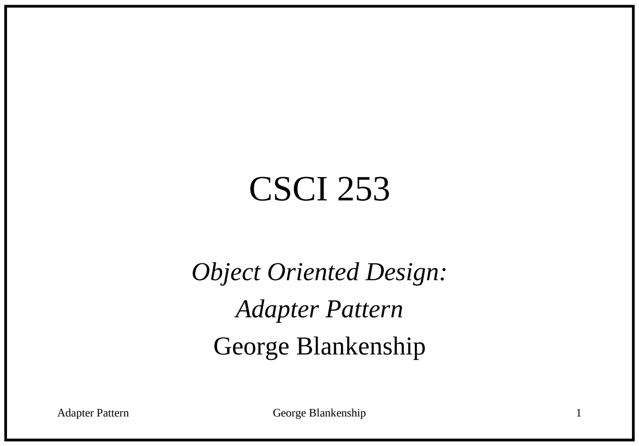
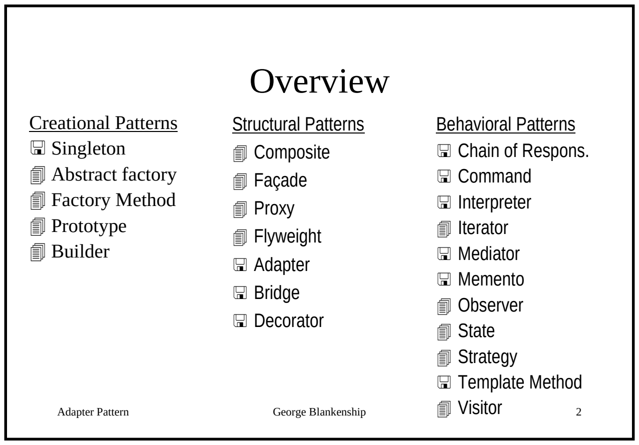


CSCI 253

Object Oriented Design:

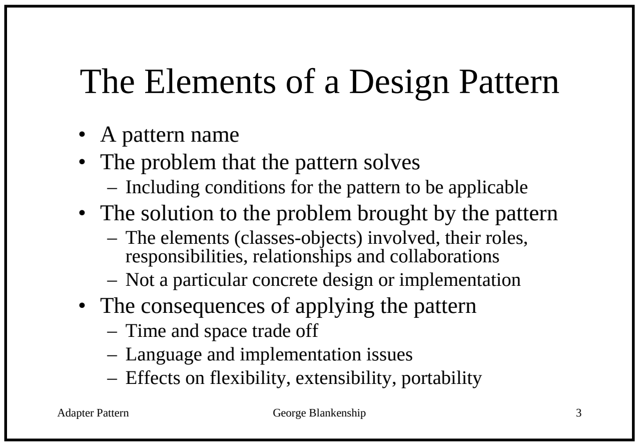
Adapter Pattern

George Blankenship



Overview

Creational Patterns	Structural Patterns	Behavioral Patterns
🖫 Singleton	🗐 Composite	🖫 Chain of Respons.
🗐 Abstract factory	🗐 Façade	🖫 Command
🗐 Factory Method	🗐 Proxy	🖫 Interpreter
🗐 Prototype	🗐 Flyweight	🗐 Iterator
🗐 Builder	🖫 Adapter	🖫 Mediator
	🖫 Bridge	🖫 Memento
	🖫 Decorator	🗐 Observer
		🗐 State
		🗐 Strategy
		🖫 Template Method
		🗐 Visitor

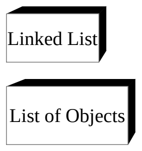


The Elements of a Design Pattern

- A pattern name
- The problem that the pattern solves
  - Including conditions for the pattern to be applicable
- The solution to the problem brought by the pattern
  - The elements (classes-objects) involved, their roles, responsibilities, relationships and collaborations
  - Not a particular concrete design or implementation
- The consequences of applying the pattern
  - Time and space trade off
  - Language and implementation issues
  - Effects on flexibility, extensibility, portability

The Adapter Pattern: The Problem

Provide an Interface for to remap the interface of an existing class to expected interface of another existing class



- Linked List – each object is a list entry (extends ListEntry)
- List of objects – none of the object are required to extend LinkEntry

Also known as : Wrapper

Motivation

- Sometimes a toolkit or class library can not be used because its interface is incompatible with the interface required by an application
- We can not change the library interface, since we may not have its source code
- Even if we did have the source code, we probably should not change the library for each domain-specific application

Link List

- Two classes to create and maintain linked lists
  - ListHead (control object for a linked list)
  - ListEntry (entries that can be placed on list)
  - Objects that extend ListEntry can be placed on any linked list
  - Objects that do not, cannot
- Should all classes extend LinkList?
- Should objects be allowed to be on two linked lists?
- Two classes to create link lists of arbitrary entities
  - ListObjectHead (extends ListHead)
  - ListObject (extends ListEntry)

ListHead

- getCount() - returns the count of entries on the list
- getFirst() - returns first entry on the list
- getLast() - returns last entry on the list
- setHead(entry) - places entry on the head of the list
- getHead() - returns (and removes) the head of the list
- setTail(entry) - places entry on the tail of the list
- getTail() - returns (and removes) the tail of the list
- remove(entry) - removes the entry from the list
- All require that the entries extend ListEntry

ListObjectHead

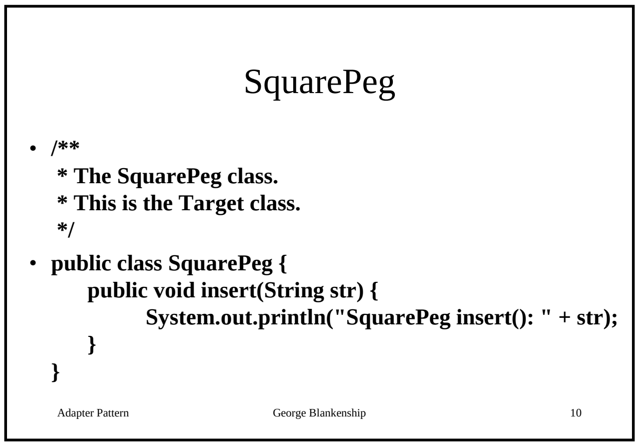
- ListObject
  - Extends ListEntry
  - Constructor creates a ListEntry containing an Object
- ListObjectHead extends ListHead
- Overloaded methods to use Object objects, not ListEntry objects
- setHead(Object o) {

```
        ListEntry e = ListObject(o);
        super.SetHead(e);
}
```

- getNextItem(Object o)
  - introduced to allow user to not be aware of ListEntry encapsulation
  - Returns the next object in the linked list

Square Peg/Round Peg Example

- Two existing tool kits
  - Square peg kit is able to orient and manipulate square pegs
  - Round peg kit is able to recognize and manipulate round pegs
- Application needs to be able to deal with pegs

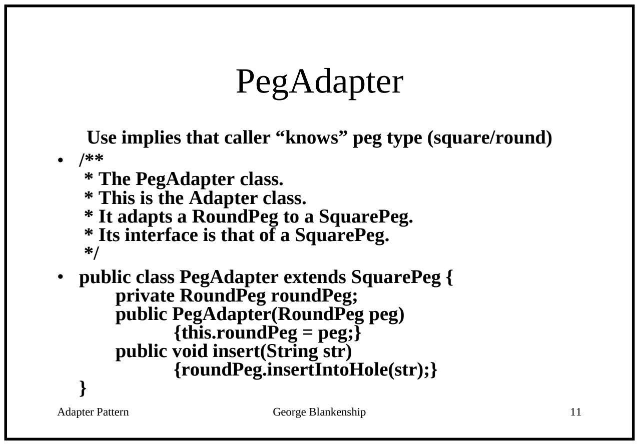


SquarePeg

- **/****
  *** The SquarePeg class.**
  *** This is the Target class.**
  ***/**
- **public class SquarePeg {**

```
public class SquarePeg {
      public void insert(String str) {
               System.out.println("SquarePeg insert(): " + str);
      }
}
```

Adapter Pattern George Blankenship 10



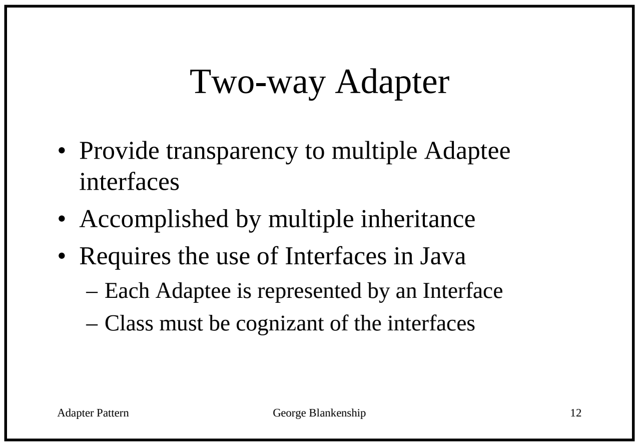
PegAdapter

Use implies that caller “knows” peg type (square/round)

```
/**
 * The PegAdapter class.
 * This is the Adapter class.
 * It adapts a RoundPeg to a SquarePeg.
 * Its interface is that of a SquarePeg.
 */
```

```
public class PegAdapter extends SquarePeg {
      private RoundPeg roundPeg;
      public PegAdapter(RoundPeg peg)
               {this.roundPeg = peg;}
      public void insert(String str)
               {roundPeg.insertIntoHole(str);}
}
```

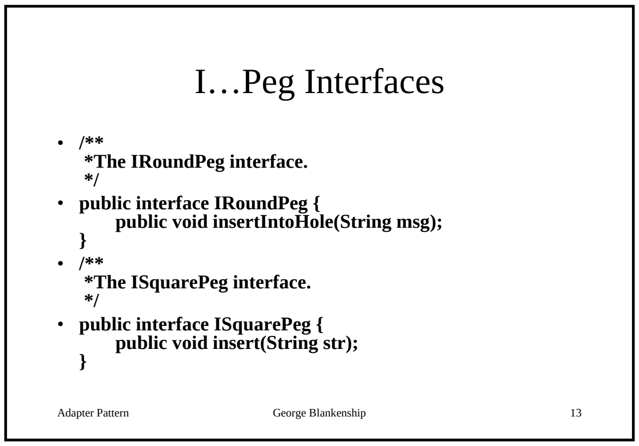
Adapter Pattern George Blankenship 11



Two-way Adapter

- Provide transparency to multiple Adaptee interfaces
- Accomplished by multiple inheritance
- Requires the use of Interfaces in Java
  - Each Adaptee is represented by an Interface
  - Class must be cognizant of the interfaces

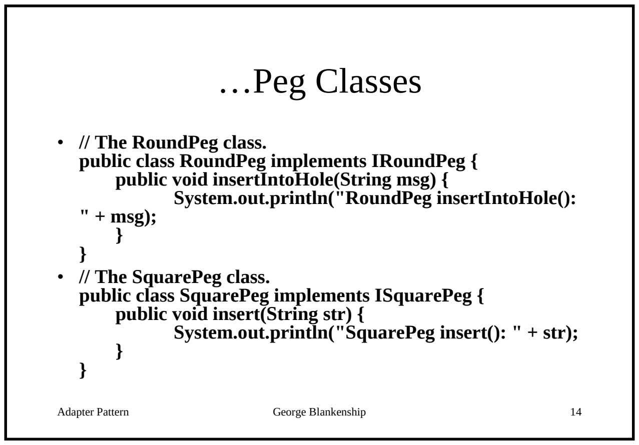
Adapter Pattern George Blankenship 12



I…Peg Interfaces

```
• /**
   *The IRoundPeg interface.
   */
• public interface IRoundPeg {
        public void insertIntoHole(String msg);
  }
• /**
   *The ISquarePeg interface.
   */
• public interface ISquarePeg {
        public void insert(String str);
  }
```

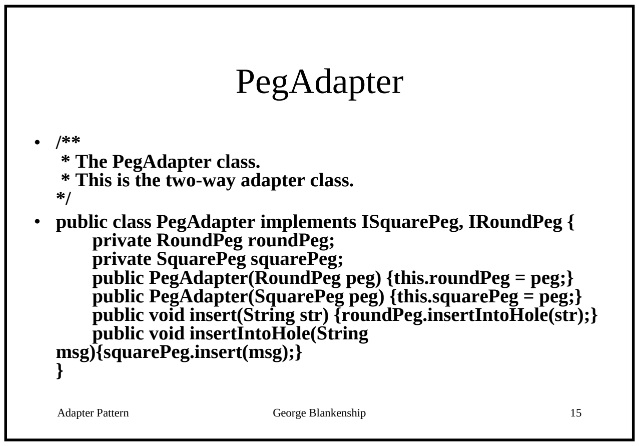
Adapter Pattern George Blankenship 13



…Peg Classes

```
• // The RoundPeg class.
  public class RoundPeg implements IRoundPeg {
        public void insertIntoHole(String msg) {
                System.out.println("RoundPeg insertIntoHole():
  " + msg);
        }
  }
• // The SquarePeg class.
  public class SquarePeg implements ISquarePeg {
        public void insert(String str) {
                System.out.println("SquarePeg insert(): " + str);
        }
  }
```

Adapter Pattern George Blankenship 14



PegAdapter

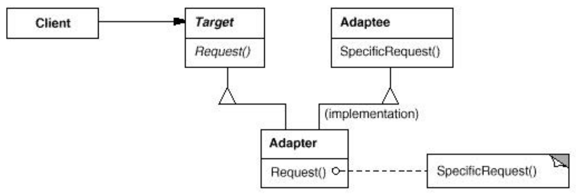
```
• /**
   * The PegAdapter class.
   * This is the two-way adapter class.
  */
• public class PegAdapter implements ISquarePeg, IRoundPeg {
        private RoundPeg roundPeg;
        private SquarePeg squarePeg;
        public PegAdapter(RoundPeg peg) {this.roundPeg = peg;}
        public PegAdapter(SquarePeg peg) {this.squarePeg = peg;}
        public void insert(String str) {roundPeg.insertIntoHole(str);}
        public void insertIntoHole(String
  msg){squarePeg.insert(msg);}
  }
```

Adapter Pattern George Blankenship 15

PegAdapter Example

```
// Test program for Pegs.
public class TestPegs {
        public static void main(String args[]) {
// Create some pegs.
                RoundPeg roundPeg = new RoundPeg();
                SquarePeg squarePeg = new SquarePeg();
// Do an insert using the square peg.
                squarePeg.insert("Inserting square peg...");
 // Do an insert using the round peg.
                roundPeg.insertIntoHole("Inserting round peg...");
// Create a two-way adapter and do an insert with it.
                ISquarePeg roundToSquare = new
PegAdapter(roundPeg);
                roundToSquare.insert("Inserting round peg...");
// Create a two-way adapter and do an insert with it.
                IRoundPeg squareToRound = new
PegAdapter(squarePeg);
                squareToRound.insertIntoHole("Inserting square peg...");
        }
}
```

The Adapter Pattern: Structure



The Adapter Pattern: Participants

- *Target*: domain-specific interface
- *Client*: collaborates with objects using the Target interface
- *Adaptee*: existing interface that needs adapting for use by Client
- *Adapter*: adapts the Target interface to the Adaptee interface

The Adapter Pattern: Collaboration

- Clients invoke operations of an Adapter instance
- Adapter instance invoke operation of an Adaptee instance
- Client is not aware of the Adaptee instance

Adapter Pattern George Blankenship 19

The Adapter Pattern: Consequences

- Class adapter
  - Concrete Adapter class
  - Unknown Adaptee subclasses might cause problem
  - Overloads Adaptee behavior
  - Introduces only one object
- Object adapter
  - Adapter can service many different Adaptees
  - May require the creation of Adaptee subclasses and referencing those objects

Adapter Pattern George Blankenship 20

The Adapter Pattern: Implementation

- Adapter should be subtype of Target
- Pluggable adapters should use the narrowest definition
  - Abstract operations to minimize exposed interface
  - Delegated objects to localize behavior
  - Parameterized processing avoids subclasses of adaptee

Adapter Pattern George Blankenship 21