CS 635 Advanced Object-Oriented Design & Programming
Fall Semester, 2021
Doc 12 Prototype, Factory Method, Abstract Factory
Nov 30, 2021

Copyright ©, All rights reserved. 2021 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/opl.shtml) license defines the copyright on this document.

Prototype

Prototype

Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype

Applicability

Use the Prototype pattern when

- A system should be independent of how its products are created, composed, and represented; and
- When the classes to instantiate are specified at run-time; or
- To avoid building a class hierarchy of factories that parallels the class hierarchy of products; or
- When instances of a class can have one of only a few different combinations of state.

Insurance Example

Insurance agents start with a standard policy and customize it

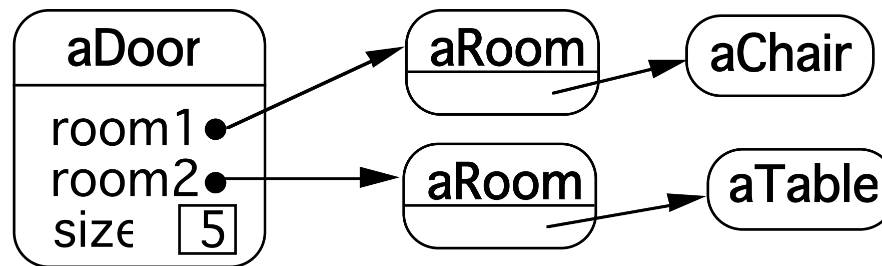
Two basic strategies:

- Copy the original and edit the copy
- Store only the differences between original and the customize version in a decorator

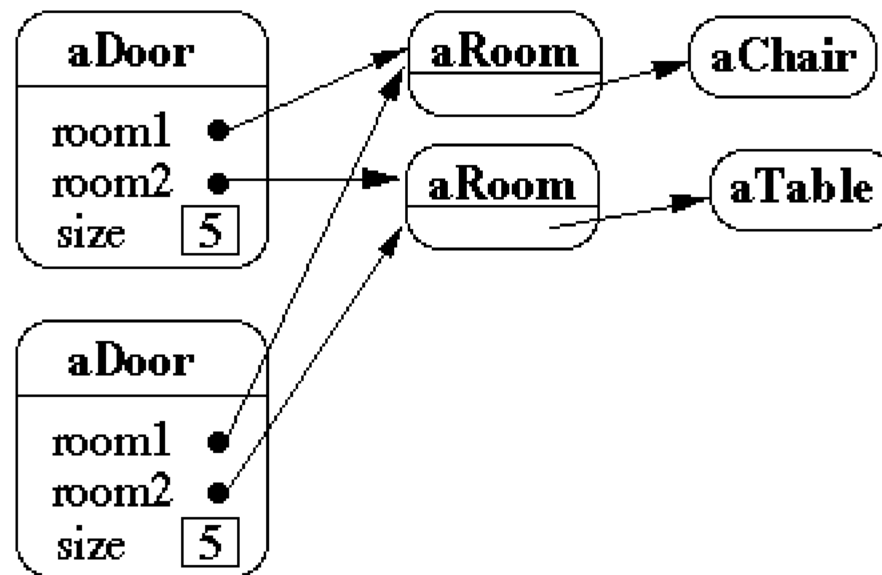
Copying Issues

Shallow Copy Verse Deep Copy

Original Objects

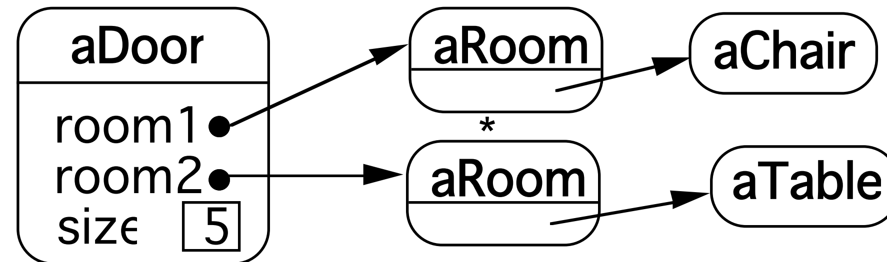


Shallow Copy

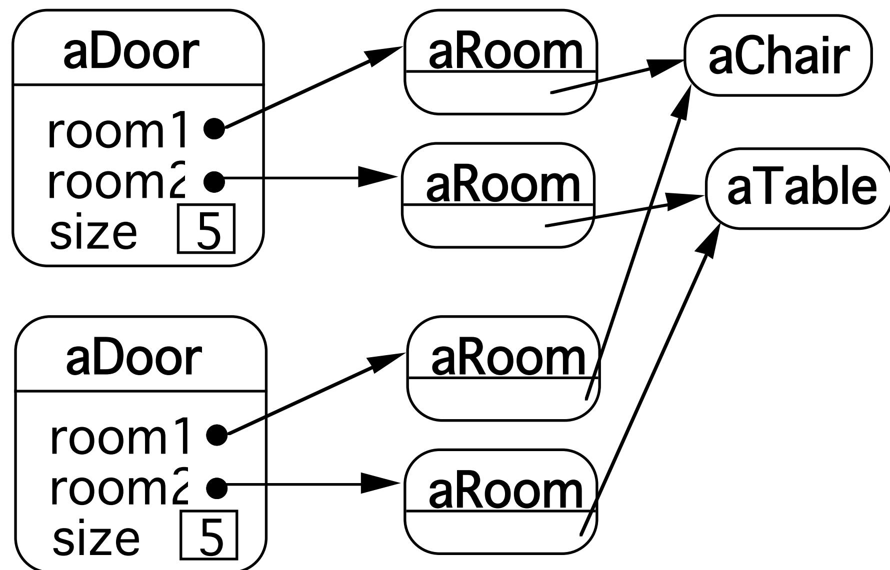


Shallow Copy Verse Deep Copy

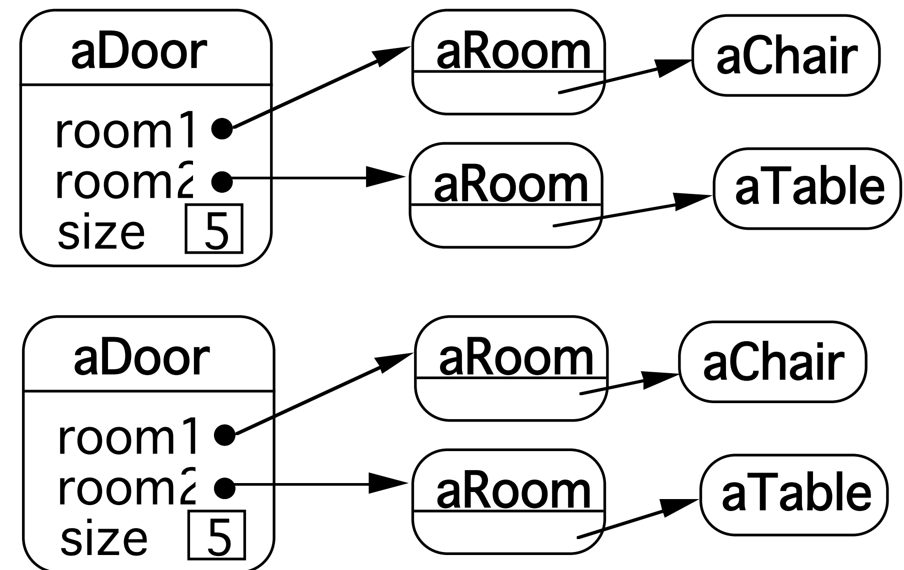
Original Objects



Deep Copy



Deeper Copy



Prototype-based Languages

No classes

Behavior reuse (inheritance)

- Cloning existing objects which serve as prototypes

Some Prototype-based languages

- Self
- JavaScript
- Squeak (eToys)
- Perl with Class::Prototyped module

JavaScript - Copying

```
var Animal = {
  type: 'Invertebrates',
  displayType: function() {
    console.log(this.type);
  }
};
```

Animal is the Prototype

```
var animal1 = Object.create(Animal);
animal1.displayType();                // Output:Invertebrates


var fish = Object.create(Animal);
fish.type = 'Fishes';
fish.displayType();                   // Output:Fishes

var copy = {...Animal};
```

JavaScript Prototype

Every object has a prototype

When looking for a property or method in an object
- Look in the object if not there
- Look in prototype, if not there
- Look in the prototype's prototype, if not there
- Continue looking in the prototype chain until find it or reach the end

A parent class's prototype is added to the subclasses prototype chain

But prototype chains are dynamic
- They can be changed at runtime

JavaScript Class

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }

}
```

```
const q = new Rectangle(10, 45);
q.height;
q['height'];
```

Prototype Example

```
class Cat {
  constructor(name) {
    this.name = name;
  }
}

class Dog {
  constructor(name) {
    this.name = name;
  }
}
```

```
let fluffy = new Cat("Fluffy");
let spot = new Dog('Spot');
fluffy['food'];                //undefined

Object.color = 'red';
Object.prototype.food = "mouse";
Cat.prototype.size = 'small';

fluffy['food'];                // 'mouse'
fluffy.size;                   // 'small'
fluffy.color;                  // undefined
spot.food;                     // 'mouse'

Dog.prototype.food = 'rabbit';
spot.food;                     // 'rabbit'
fluffy.food;                   // 'mouse'
```

JavaScript Classes are Functions

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
}

console.log(typeof Rectangle);                         // function

console.log(Object.getOwnPropertyNames(Rectangle));    // [ 'length', 'prototype', 'name' ]

console.log(Rectangle.name);                           // Rectangle

console.log(Rectangle.length);                         // 2
```

Most Things are Objects in JavaScript

Object
- Function
- Array
- Date
- RegExp

```
function adder(x, y) {
  return x + y;
}

console.log(Object.getOwnPropertyNames(adder))

 [ 'length', 'name', 'arguments', 'caller', 'prototype' ]
```

Factory Method

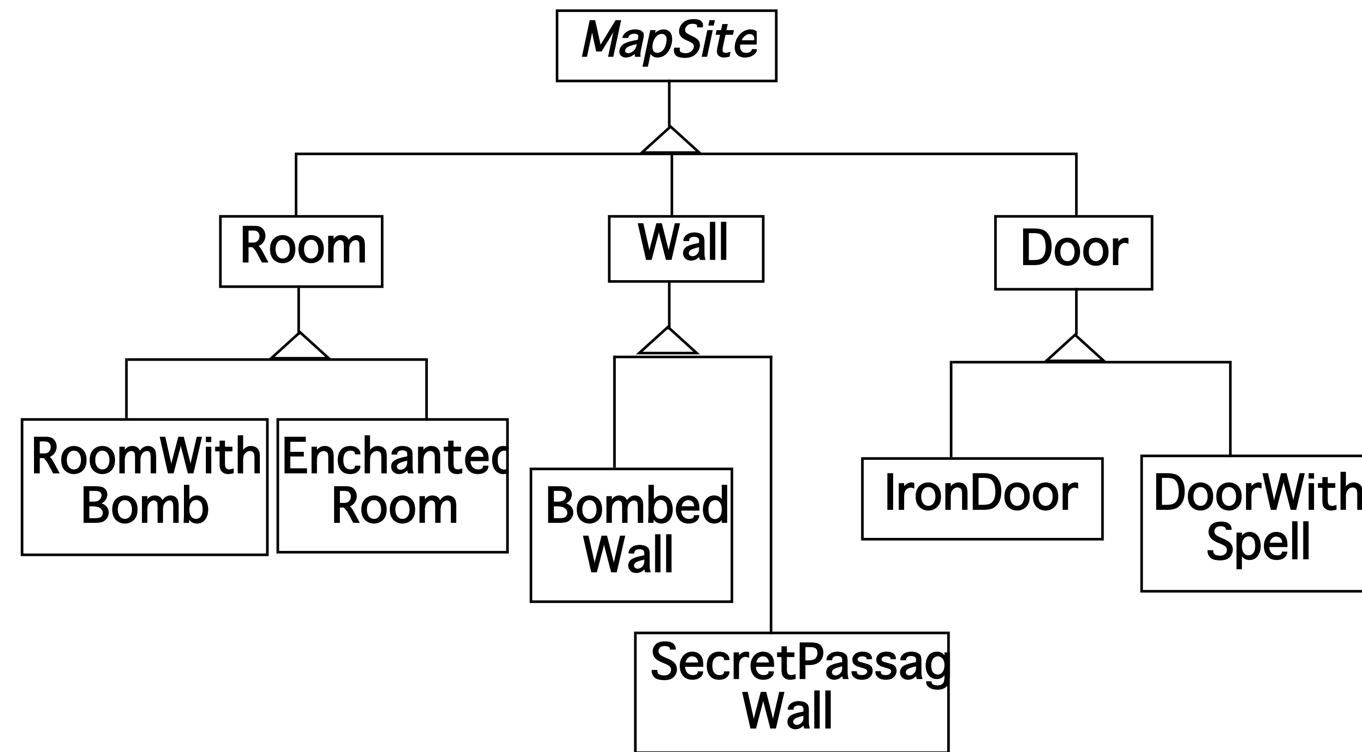
Factory Method

A template method for creating objects

```
public class Example {
   protected Bar bar() { return new Bar(); }

    public void foo() {
      blah
      Bar soap = bar();
      blah;
   }
```

Maze Game Example



Maze Game Example

```
class MazeGame{
   public Maze makeMaze() { return new Maze(); }
   public Room makeRoom(int n ) { return new Room( n ); }
   public Wall makeWall() { return new Wall(); }
   public Door makeDoor() { return new Door(); }

   public Maze CreateMaze(){
      Maze aMaze = makeMaze();
      Room r1 = makeRoom( 1 );
      Room r2 = makeRoom( 2 );
      Door theDoor = makeDoor( r1, r2);

      aMaze.addRoom( r1 );
      aMaze.addRoom( r2 );
      etc

      return aMaze;
      }
   }
```

```
class BombedMazeGame extends MazeGame {

   public Room makeRoom(int n )  {
      return new RoomWithABomb( n );
   }

   public Wall makeWall() {
      return new BombedWall();
   }
}
```

Don't repeat your self

```
public class LinkedList extends Collection {
  public OrderedLinkedList() {
    this(defaultOrder());
  }

  public LinkedList(Order listOrder ) {
    this(listOrder, new OrderedCollection());
  }

  public LinkedList(Collection items) {
    this(defaultOrder(), items);
  }

  protected Order defaultOrder() {
    return new RandomOrder();
  }

  public LinkedList(Order listOrder, Collection items) {
    blah
```

Implementation Variation

```
class Hershey {

   public Candy makeChocolateStuff( CandyType id ) {
      if ( id == MarsBars ) return new MarsBars();
      if ( id == M&Ms ) return new M&Ms();
      if ( id == SpecialRich ) return new SpecialRich();

      return new PureChocolate();
   }

class GenericBrand extends Hershey {
   public Candy makeChocolateStuff( CandyType id ) {
      if ( id == M&Ms ) return new Flupps();
      if ( id == Milk ) return new MilkChocolate();
      return super.makeChocolateStuff(id);
   }
}
```

Smalltalk Variant

Return the class, caller creates an object

```
chocolateStuff
	^SpecialRich
```

```
some code
candy := (self chocolateStuff) new
mode code
```

Use Factory Method When

A class can't anticipate the class of objects it must create

A class wants its subclasses to specify the objects it creates

You want to localize the knowledge of which help classes is used in a class

But when is this?

CS 580 Example - Testing a Server

```
public class SDTwitterServer {
   public void run(int port) throws IOException {
      ServerSocket input = new ServerSocket( port );

      while (true) {
          Socket client = input.accept();
          processRequest(
              client.getInputStream(),
              client.getOutputStream());
          client.close();
      }
   }

   void processRequest(InputStream in,OutputStream out) {
      do a bunch of stuff
   }

   etc.
```

Using Factory Method

```
public class SDTwitterServer {
   public void run(int port) throws IOException {
      ServerSocket input = this.serverSocket( port );

      while (true) {
          Socket client = input.accept();
          processRequest(
              client.getInputStream(),
              client.getOutputStream());
          client.close();
      }
   }

   ServerSocket serverSocket( int port) {
      return new ServerSocket(port);
   }

   etc.
```

TestServer

```
public class TestServer extends SDTwitterServer {
   MockServerSocket testSocket;

   ServerSocket serverSocket( int port) {
      return testSocket;
   }
}
```

Other than using a different type of socket it performs the operations as the parent class

```
public class Tests extends Testcase {
   public void testLogin() {
      TestServer server = new TestServer();
      server.testSocket = new MockServerSocket("client command to login");
      server.run();
      assertTrue(server.testSocket.serverResponse() = "the correct response here");
   }
}
```

MockServerSocket

Returns a fake (Mock) client connection

Fakes client connection

- Does not use network
- Contains fixed requests
- Records server responses

Dependency Injection

```
public class SDTwitterServer {
    ServerSocket input;
    public SDTwitterServer(ServerSocket input) {
          this.input = input;
    }

    public void run(int port) throws IOException {

       while (true) {
           Socket client = input.accept();
           processRequest(
               client.getInputStream(),
               client.getOutputStream());
           client.close();
       }
    }
```

Dependency Injection

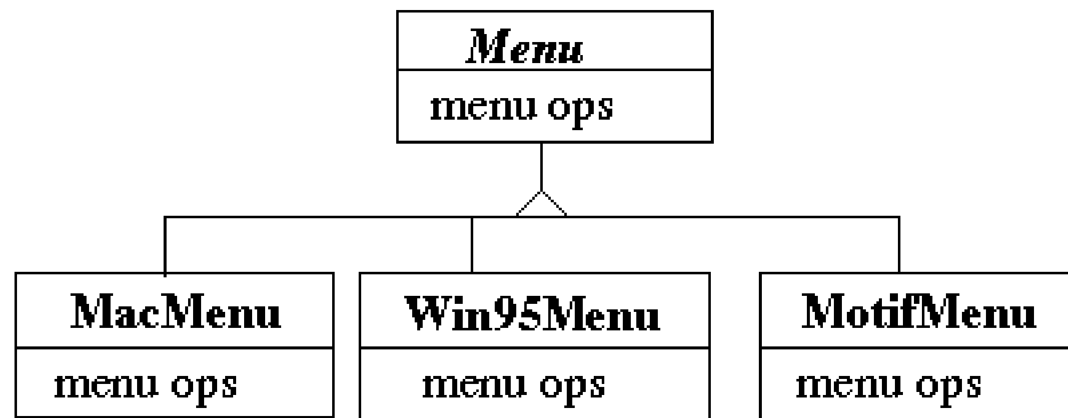
"One object (or static method) supplies the dependencies of another object"
Wikipedia

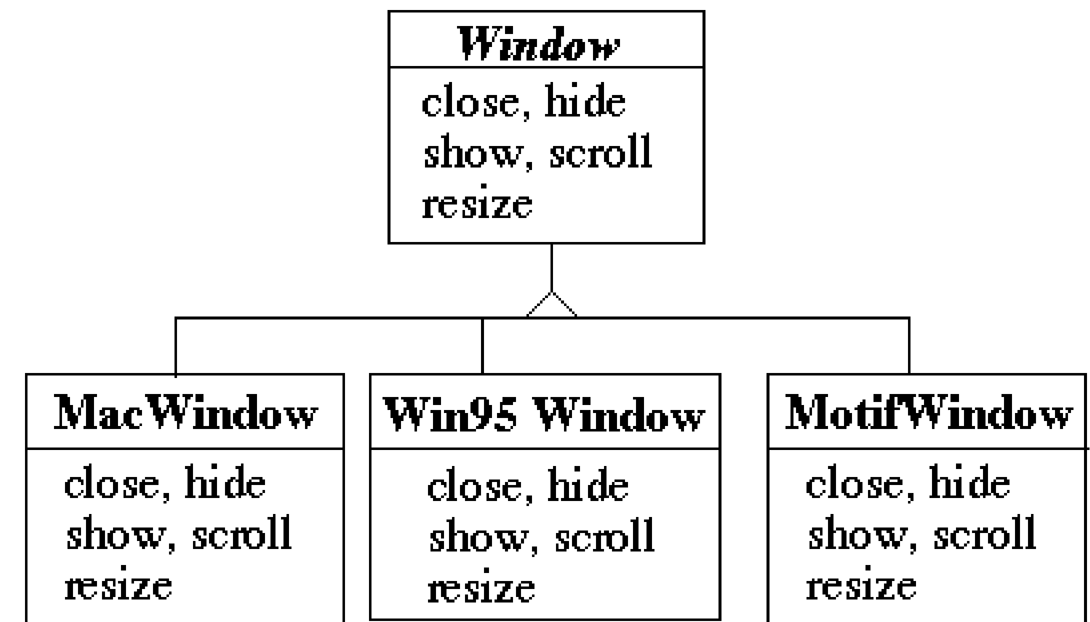
Constructor injection
Setter injection
Interface injection

Abstract Factory

Abstract Factory

Write a cross platform window toolkit





Bad Code Dependencies

```
public void installDisneyMenu()
   {
   Menu disney = new MacMenu();
   disney.addItem( "Disney World" );
   disney.addItem( "Donald Duck" );
   disney.addItem( "Mickey Mouse" );
   disney.addGrayBar( );
   disney.addItem( "Minnie Mouse" );
   disney.addItem( "Pluto" );
   etc.
   }
```

Use Abstract Factory

```
abstract class WidgetFactory {
    public Window createWindow();
    public Menu createMenu();
    public Button createButton();
    }
```

```
class MacWidgetFactory extends WidgetFactory {
    public Window createWindow()
        { code to create a mac window }

    public Menu createMenu()
        { code to create a mac Menu }

    public Button createButton()
        { code to create a mac button }
}

class Win95WidgetFactory extends WidgetFactory {
    public Window createWindow()
        { code to create a Win95 window }

    public Menu createMenu()
        { code to create a Win95 Menu }

    public Button createButton()
        { code to create a Win95 button }
}
```

Use one Factory per Application

```
public void installDisneyMenu(WidgetFactory myFactory)
   {
   Menu disney = myFactory.createMenu();
   disney.addItem( "Disney World" );
   disney.addItem( "Donald Duck" );
   disney.addItem( "Mickey Mouse" );
   disney.addGrayBar( );
   disney.addItem( "Minnie Mouse" );
   disney.addItem( "Pluto" );
   etc.
   }
```

Abstract Factory

Encapsulate a group of individual factories that have a common theme

Separates the details of implementation of a set of objects
from its general usage

How Do Abstract Factories create Things?

Use Subclass Factory Method

```
abstract class WidgetFactory
   {
   public Window createWindow();
   public Menu createMenu();
   public Button createButton();
   }

class MacWidgetFactory extends WidgetFactory
   {
   public Window createWindow()
      { return new MacWidow() }

   public Menu createMenu()
      { return new MacMenu() }

   public Button createButton()
      { return new MacButton()  }
   }
```

Use Widget Factory Method

```
abstract class WidgetFactory {
    private Window windowFactory;
    private Menu menuFactory;
    private Button buttonFactory;

    public Window createWindow()
        { return windowFactory.createWindow() }

    public Menu createMenu();
        { return menuFactory.createMenu() }

    public Button createButton()
        { return buttonFactory.createMenu() }
}
```

```
class MacWidgetFactory extends WidgetFactory {
    public MacWidgetFactory() {
        windowFactory = new MacWindow();
        menuFactory = new MacMenu();
        buttonFactory = new MacButton();
    }
}

class MacWindow extends Window {
    public Window createWindow() { blah }
    etc.
```

Why Widget Factory Method?

```
abstract class WidgetFactory {
    private Window windowFactory;
    private Menu menuFactory;
    private Button buttonFactory;

    public Window createWindow()
        { return windowFactory.createWindow() }

    public Window createWindow( Rectangle size)
        { return windowFactory.createWindow( size ) }

    public Window createWindow( Rectangle size, String title)
        { return windowFactory.createWindow( size, title) }

    public Window createFancyWindow()
        { return windowFactory.createFancyWindow() }

    public Window createPlainWindow()
        { return windowFactory.createPlainWindow() }
```

Multiple ways to create Widget

Use Prototype

```
class WidgetFactory{
    private Window windowPrototype;
    private Menu menuPrototype;
    private Button buttonPrototype;

    public WidgetFactory( Window windowPrototype,
                          Menu menuPrototype,
                          Button buttonPrototype)
        {
        this.windowPrototype = windowPrototype;
        this.menuPrototype = menuPrototype;
        this.buttonPrototype = buttonPrototype;
        }

    public Window createWindow()
        { return windowPrototype.createWindow() }

    public Window createWindow( Rectangle size)
        { return windowPrototype.createWindow( size ) }

    public Window              ()
        { return menuPrototype.createMenu() }
    etc.
```

How to prevent Cheating?

```
public void installDisneyMenu(WidgetFactory myFactory)
   {
   // We ship next week, I can't get the stupid generic Menu
   // to do the fancy Mac menu stuff
   // Windows version won't ship for 6 months
   // Will fix this later

   MacMenu disney = (MacMenu) myFactory.createMenu();
   disney.addItem( "Disney World" );
   disney.addItem( "Donald Duck" );
   disney.addItem( "Mickey Mouse" );
   disney.addMacGrayBar( );
   disney.addItem( "Minnie Mouse" );
   disney.addItem( "Pluto" );
   etc.
   }
```