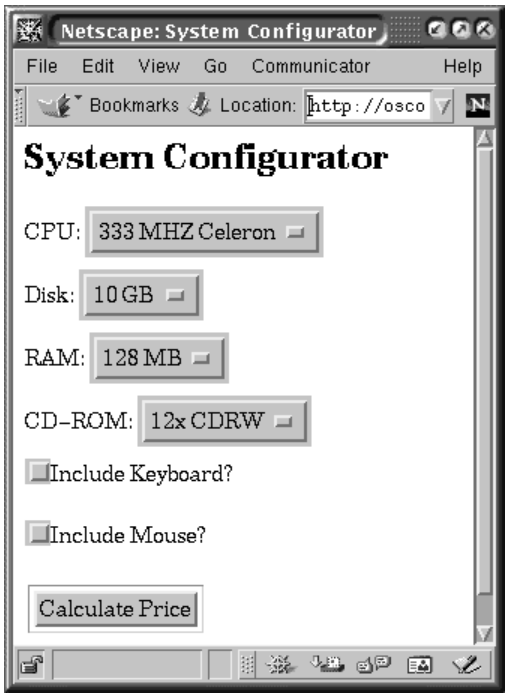
<<Form>>
BuildYourOwn

<<Select>> cpu_model
<<Select>> disk_size
<<Select>> ram_size
<<Select>> cdrom_model
<<Checkbox>> include_keyboard
<<Checkbox>> include_mouse
<<Submit>> show_price

make_popup(label, item)

<<Perl Module>>
BYOData

get_options(component)
get_price(component, option)




```
#!/usr/bin/perl

use BYOData;
use CGI qw(:standard);

# Headers and headings.
print header();
print start_html("System Configurator");
print "<H1>System Configurator</H1>";

# Display the form.
print start_form(-action => 'CalcPrice.cgi');

# Create menus.
make_popup('CPU:',    'cpu_model');
make_popup('Disk:',   'disk_size');
make_popup('RAM:',    'ram_size');
make_popup('CD-ROM:', 'cdrom_model');

# Display the keyboard and mouse checkboxes.
print checkbox(-name => 'keyboard',
               -label => 'Include Keyboard?');
print "<P>\n";
print checkbox(-name => 'mouse',
               -label => 'Include Mouse?');
print "<P>\n";
# Wrap up the form and HTML doc.
print submit("Calculate Price");
print end_form();
print end_html();

# Creates a popup menu.
sub make_popup {

  my $label = shift;  # The label.
  my $item = shift;   # The component name.

  my @values = BYOData::get_options($item);
  my $menu = popup_menu(-name => $item,
                        -values => \@values);

  print "$label $menu<BR>\n";
}
```


1-A

Exercise. Implement this in the language of your choice. Feel free to make up your own values for the options.

<<Form>>
CustomShoe

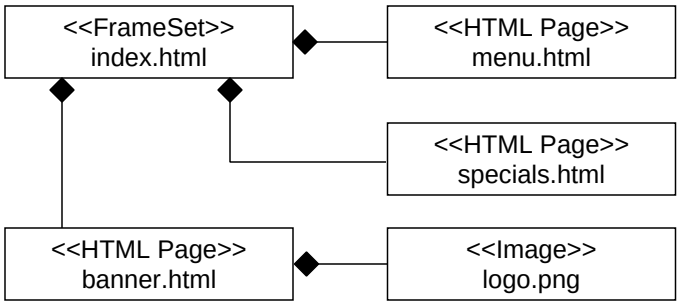
<<Option>> size
<<Option>> color
<<Option>> material
<<Option>> shipping_option
<<Checkbox>> is_gift
<<TextField>> gift_message
<<Submit>> show_price

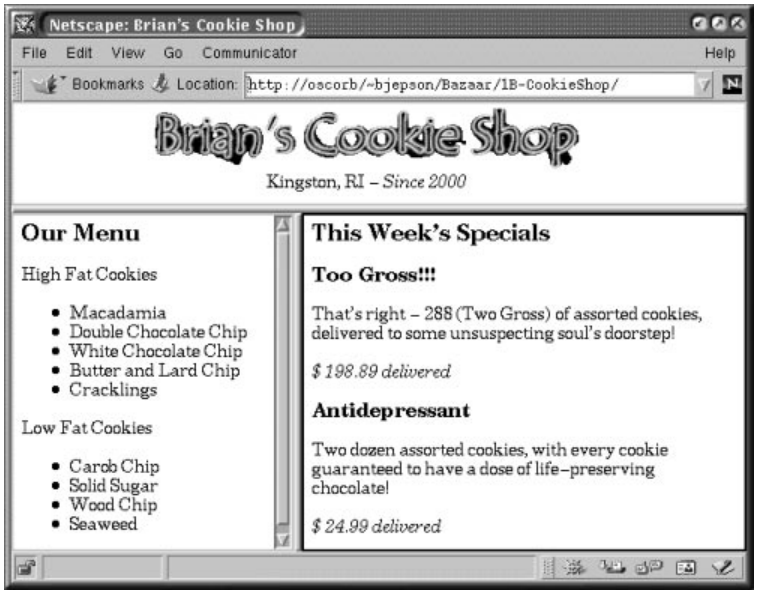
make_popup(label, item)

<<Perl Module>>
ShoeData

get_options(choice)
get_price(choice, option)

```
package BYOData;

%config = (
  cpu_model => {
      '333 MHZ Celeron' => 39,
      '400 MHZ Celeron' => 43,
      '466 MHZ Celeron' => 54,
  },
  disk_size => {
      '6 GB'   => 60,
      '8.4 GB' => 80,
      '10 GB'  => 99,
  },
  ram_size => {
      '32 MB'  => 50,
      '64 MB'  => 100,
      '128 MB' => 200,
  },
  cdrom_model => {
      '24x'      => 60,
      '44x'      => 70,
      '12x CDRW' => 129,
  },
  keyboard => 12,
  mouse    => 5,
);

# Return a list of options for
# a given component.
#
sub get_options {
  my $component = shift;
  return sort keys %{ $config{$component} };
}

# Return the price of an option.
#
sub get_price {
  my $component = shift;
  my $option = shift;
  return $config{$component}->{$option};
}
1;
```





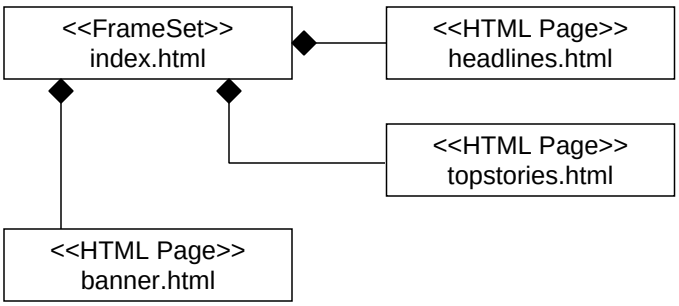

```
<HTML>
<HEAD>
<TITLE>Brian's Cookie Company</TITLE>
</HEAD>
<BODY>
<CENTER>
<IMG SRC="logo.gif" ALT="Company Logo"><BR>
Kingston, RI - <I>Since 2000</I>
</CENTER>
</BODY>
</HTML>
```

```
<HTML>
<HEAD>
<TITLE>Cookie Menu</TITLE>
</HEAD>
<BODY>
<H2>Our Menu</H2>
High Fat Cookies
<UL>
  <LI>Macadamia
  <LI>Double Chocolate Chip
  <LI>White Chocolate Chip
  <LI>Butter and Lard Chip
  <LI>Cracklings
</UL>
Low Fat Cookies
<UL>
  <LI>Carob Chip
  <LI>Solid Sugar
  <LI>Wood Chip
  <LI>Seaweed
</UL>
</BODY>
</HTML>
```

1-B

1-B

Exercise. Here is a frameset that describes the home page of an on-line news site, Little Rest News. Develop the HTML pages that make up this frameset.



Some possible headlines:

Mayor declares Tuesday to be free doughnut day.

Student uprising leads to frenzied spending spree.

Mayor concedes to health advocates, explains he meant doughnut-free day.

Main Street closed for paving project.

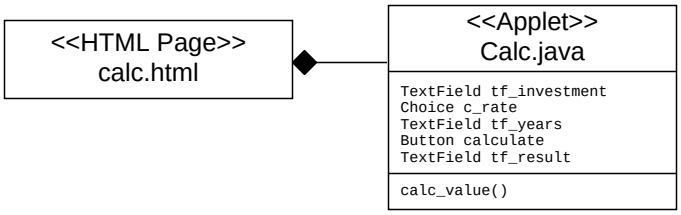
Pottery Shack in mortal feud with Pottery Kiosk.

Doughnuts in the air!

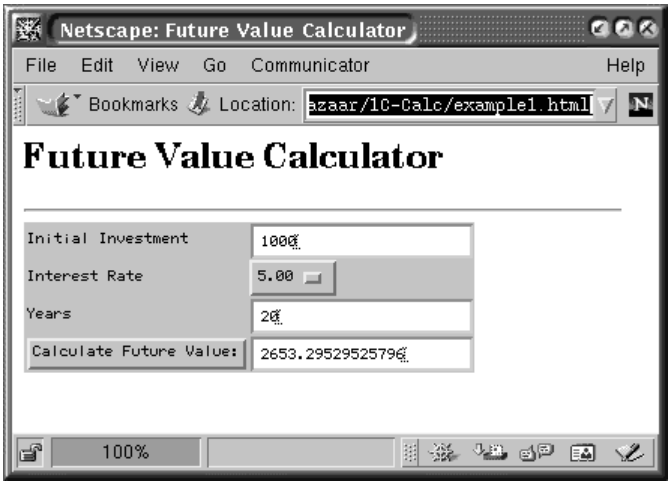
Short order cooks in short supply.

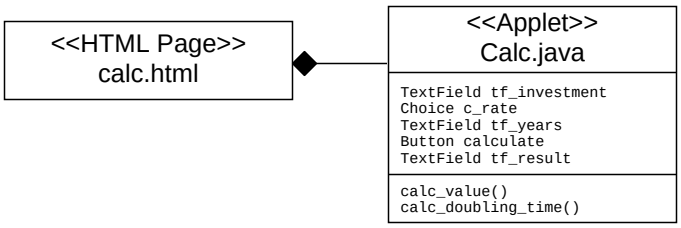
```
<HTML>
<HEAD>
<TITLE>Brian's Cookie Shop</TITLE>
</HEAD>
<FRAMESET ROWS="90,*">
    <FRAME SRC="banner.html">
    <FRAMESET COLS="240,*">
        <FRAME SRC="menu.html">
        <FRAME SRC="specials.html">
    </FRAMESET>
</FRAMESET>
<NOFRAMES>
    You may be using a browser that
    does not support frames.<P>
    <A HREF="menu.html">See our menu
    of cookies.</A><BR>
    <A HREF="specials.html">See our
    current specials.</A><BR>
</NOFRAMES>
</HTML>
```

```
<HTML>
<HEAD>
<TITLE>Specials</TITLE>
</HEAD>
<BODY>
<H2>This Week's Specials</H2>
<H3>Too Gross!!!</H3>
That's right - 288 (Two Gross) of
assorted cookies, delivered to some
unsuspecting soul's doorstep!<P>
<I>$ 198.89 delivered</I>
<H3>Antidepressant</H3>
Two dozen assorted cookies, with
every cookie guaranteed to have a
dose of life-preserving chocolate!<P>
<I>$ 24.99 delivered</I>
</BODY>
</HTML>
```




```
<HTML>

  <HEAD>
    <TITLE>Future Value Calculator</TITLE>
  </HEAD>

  <BODY>
    <h1>Future Value Calculator</h1>
    <hr>
    <applet codebase="." code="Calc.class" width=300 height=100>
    </applet>
  </BODY>

</HTML>
```





Exercise. Add a feature to the applet that calculates the doubling time of the investment.

The formula to calculate the doubling time (in years) is:

ln(2) / ln(1 + interest rate)

Here is an implementation in Java:

```
double time_to_double = Math.log(2) / Math.log(1 + rate);
```

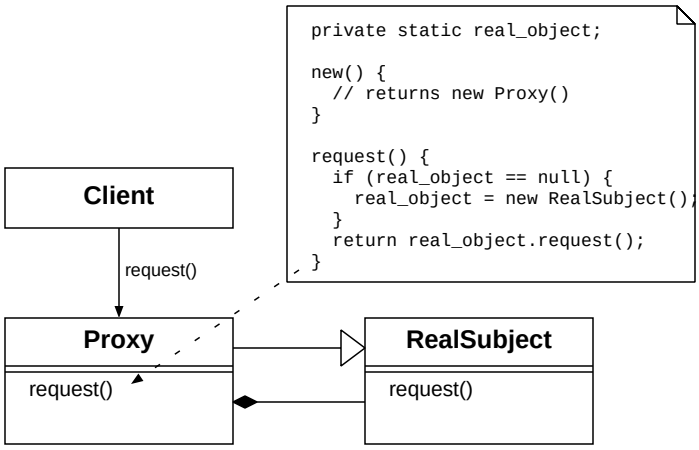
Check your work by feeding the projected doubling time into the years field of the applet.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class Calc extends Applet {

  TextField tf_investment, tf_years, tf_result;
  Choice c_rate;
  public void init() {

      setLayout( new GridLayout(4, 2) );

      add( new Label("Initial Investment") );
      add( tf_investment = new TextField(20) );

      add( new Label("Interest Rate") );
      add( c_rate = new Choice() );
      c_rate.add("5.00"); c_rate.add("5.25"); c_rate.add("5.75");

      add( new Label("Years") );
      add( tf_years = new TextField(20) );

      Button calculate = new Button("Calculate Future Value:");
      add(calculate);
      calculate.addActionListener( new ActionListener() {
          public void actionPerformed(ActionEvent e) {
                calc_value();
          }
      });

      // Result field.
      add( tf_result = new TextField(20) );
  }

  // Future value formula:
  //    FV = PV * [ (1 + rate) ** years ]
  void calc_value() {
      try {
          long pv    = Long.parseLong( tf_investment.getText() );
          long years = Long.parseLong( tf_years.getText() );
          String str_rate = c_rate.getSelectedItem();
          float rate = Float.valueOf( str_rate ).floatValue() / 100;
          // Get the future value and update the result.
          double fv = pv * Math.pow(1 + rate, years);
          tf_result.setText( String.valueOf(fv) );
      } catch (NumberFormatException e) {
          System.err.println(e.getMessage());
      }
  }
}
```

Proxy

Problem: Suppose we need to represent a large number of complex objects, but only a handful of them actually get used. However, we don't know which ones will get used until run-time.

How can we defer the cost of creating an object until it gets used?



Solution: Create a proxy object that stands in for the real object until it is needed. The proxy may be a subclass of the target, and it also may contain an instance of the target.

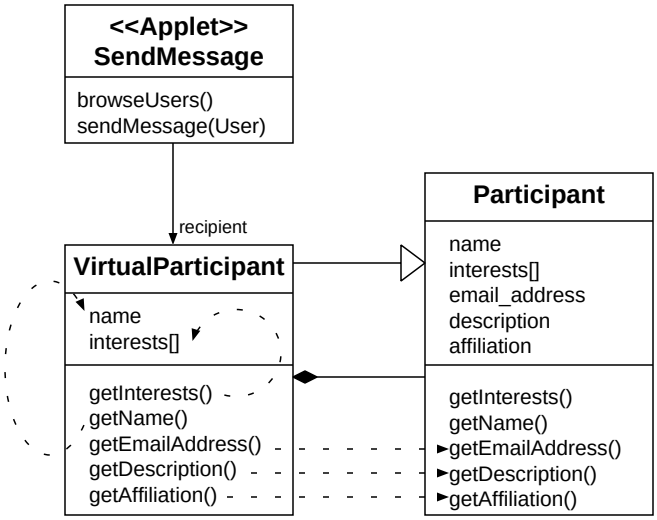
Assignment: Find three collaborations that implement Proxy

Source: Design Patterns, Elements of Reusable Object-Oriented Software. Gamma, et al.

Many Participants

Our web application is a real-time conferencing system. Our application maintains a pool of Participant objects that represent any of the thousands of users who interact with our site.

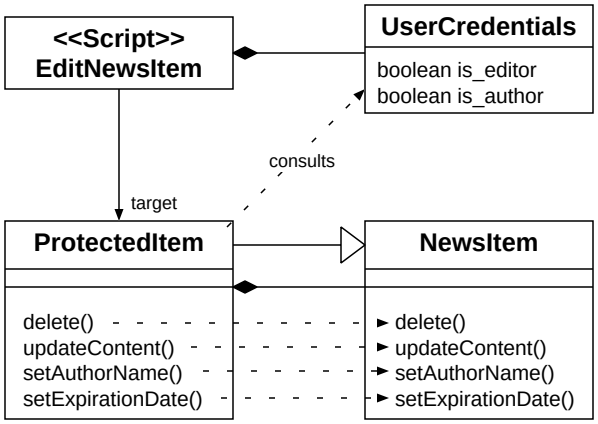
The most common interaction with a Participants object is inspection: one of the users looks at the complete list of participants, with their names and preferences. If the user decides to initiate a conversation with another participant, or send that participant a message, the system creates a real Participant on demand.



Access Levels

Our website presents news articles to our readers. We have several users who work on each article, and these people need to have varying levels of access to the article.

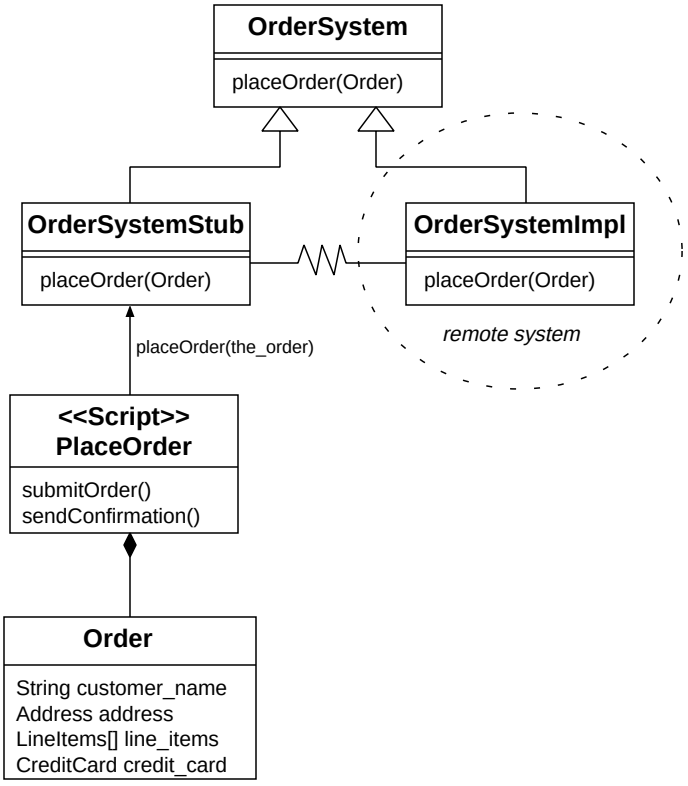
We didn't want to muddle up the NewsItem class by dumping a lot of security features in it, so we've created a wrapper class to control access. the wrapper class examines the current user's credentials before permitting or denying a specific action.



Remote Objects

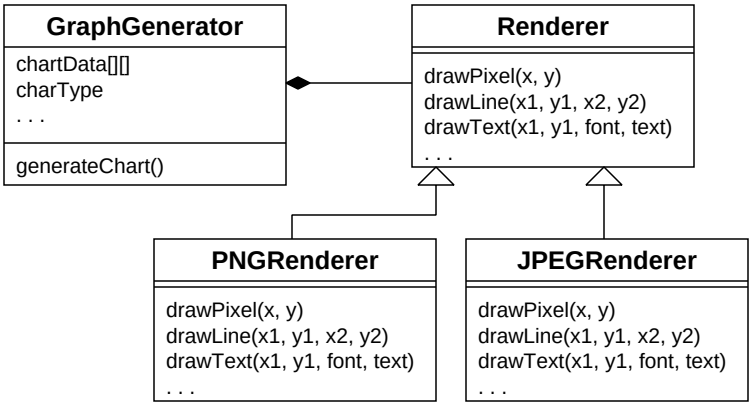
Our online ordering system needs to submit orders to a remote object that is running on a server somewhere across the network.

We are using CORBA to access this object remotely and invoke its methods.



Builder

Problem: Suppose that the same information needs to be rendered in multiple ways. Imagine a data graph that can be rendered in GIF, JPG, or PNG. How can we separate the construction of the image from the way we represent it internally?



Solution: The builder pattern suggests that we define a common interface for classes that generate images. Then, we create implementations for each of the graphics formats we want to support. A client program selects a different implementation depending on what image format we want to use

Assignment: Find three collaborations that implement Builder.

Source: Design Patterns, Elements of Reusable Object-Oriented Software. Gamma, et al.

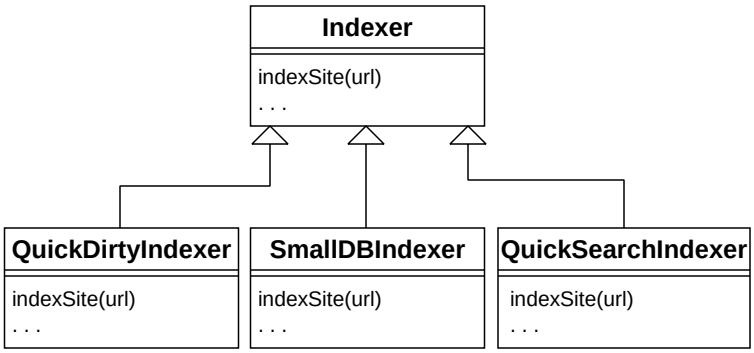
Site Management

Depending on a user's needs, our web site management suite can use one of three indexing algorithms:

Quick and Dirty: Generates indexes quickly, but the performance of searches is poor, especially for large sites.

Small Database: Generates the smallest index, but takes a long time to generate. Searches perform acceptably.

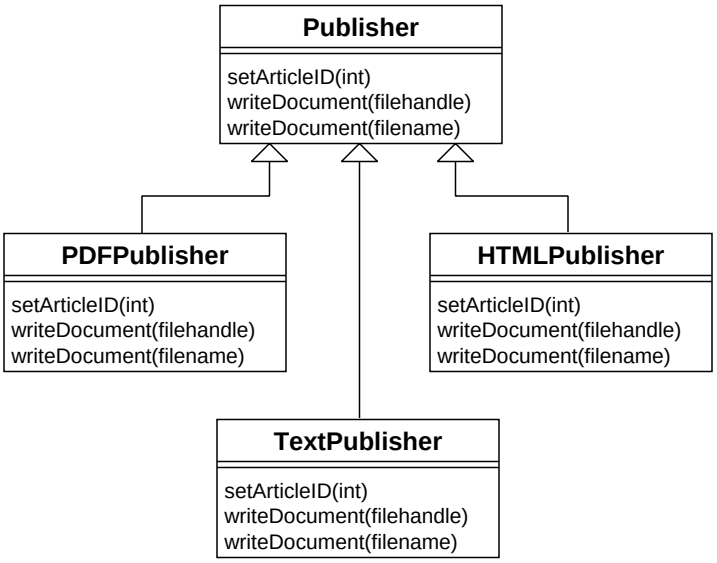
Quick Searches: Generates a large index, takes a long time to generate. Searches perform very quickly.



All the News that Fits

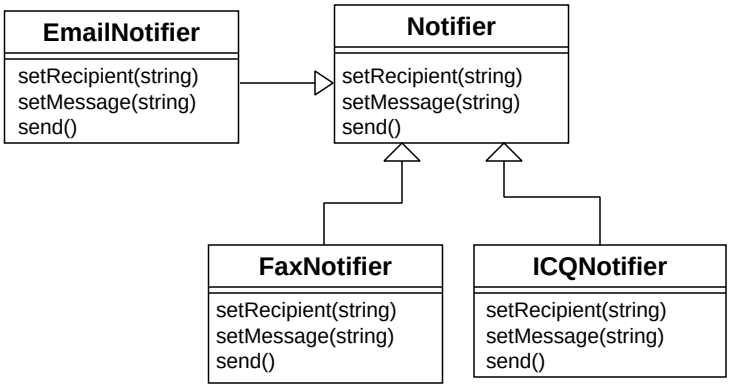
Our news delivery system stores news stories in an SQL database. Because it's stored in an easily accessible format, we can render the news stories in a variety of ways.

Our system will use a plug-in architecture to generate news stories in formats such as HTML, plain text, and PDF.



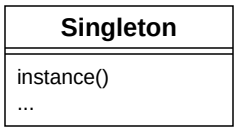
Automated Notifications

Our system needs a generic method to send notification messages regardless of the delivery method (email, fax, or ICQ).



Singleton

Problem: We want to make sure that a given class only has one instance, but we don't want to use a global variable, since it's kind of messy.




```
public class Singleton {

  static private Singleton the_object;

  // The real constructor.
  private Singleton() {
  }

  // Constructor from the Singleton pattern.
  static public Singleton instance() {
    if (the_object == null) {
      the_object = new Singleton();
    }
    return the_object;
  }

}
```

Solution: The Singleton pattern gives you an alternate constructor that always returns the same instance of the class.

Applications that use Singleton should not call the new method, but should instead call the instance() method. The instance() method only calls the new() method once, and stores the new instance into the_object. Next time the application invokes instance(), it gets the same object again.

Assignment: Find three collaborations that implement Singleton

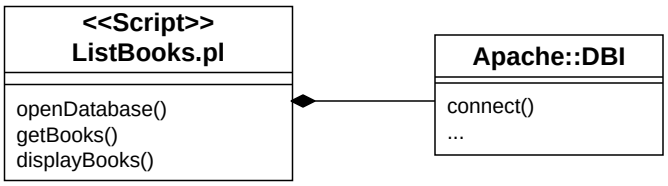
Source: Design Patterns, Elements of Reusable Object-Oriented Software. Gamma, et al.

Making the Connection

We're using Apache and mod_perl (perl.apache.org) in a public library system. We're also using the Perl DBI to connect to a database of books. Since scripts written for mod_perl remain in memory between invocations, it makes sense to use a persistent database connection, rather than making a new connection each time the script is run.

From the Apache::DBI man page:

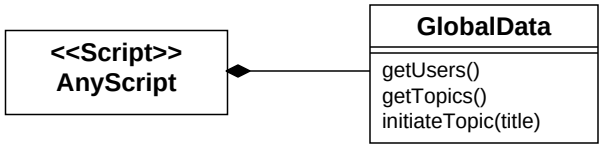
> ...every connect request will be forwarded to the Apache::DBI module. This looks if a database handle from a previous connect request is already stored and if this handle is still valid... If these two conditions are fulfilled it just returns the database handle... If there is no appropriate database handle or if the ping method fails, a new connection is established and the handle is stored for later re-use.



Sharing Things

Our web application needs to share certain global values between different users. The application is a real-time conferencing system, and users need to get a list of the other users who are logged in, the currently active topics, and other information.

We're using a GlobalData object to encapsulate the global application data.

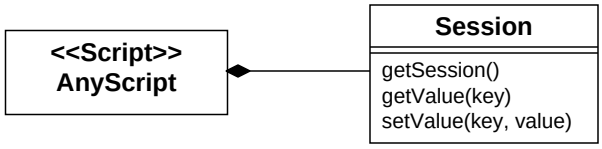


Remembering You

In a CGI application, scripts are terminated when they have finished running. As a result, all of the information stored in the script's variables is forgotten.

We're using a session object that uses HTTP cookies to store variables on the user's browser. That way, each time a script starts up, it can get an instance of that session object, and we have access to all the variables left there by the last script that the user ran!

In this way, we can share state information between scripts.



Use Case: Read News Articles

Summary: Our user visits our home page, and is presented with a list of current topics (with descriptions). He should select one or more topics, and then the system displays all the headlines for the selected topics. In addition, each article is rated by other users (a score of -1 or 1), and the total score is displayed next to each headline. The user can read the articles by choosing a headline.

User Action	System Response
1. Visit the home page.	2. Display a list of topics and topic descriptions.
3. Choose one or more topics	4. Display all the headlines with rating score, grouped by each topic.
5. Choose a headline.	6. Display the article that corresponds to the headline.

Assignment: find three concepts, then work together to find relationships and draw the conceptual model.

Use Case: Create User Profile

Summary: A user decides to create a personal profile for their future visits to our web site. As they do this, they select various preferences, and finally create their profile.

User Action	System Response
1. Initiates creation of a new profile.	2. Starts the new profile process, which begins by collecting the user's name and email address.
3. Enter name and email address.	4. Prompt user to choose one or more preferred musical genres.
5. Choose musical genre.	6. Prompt user for default shipping address.
7. Supply shipping address.	8. Prompt user to confirm information before creating profile.
9. Confirm that the information is correct.	10. Create the new profile and redirect user to the home page.

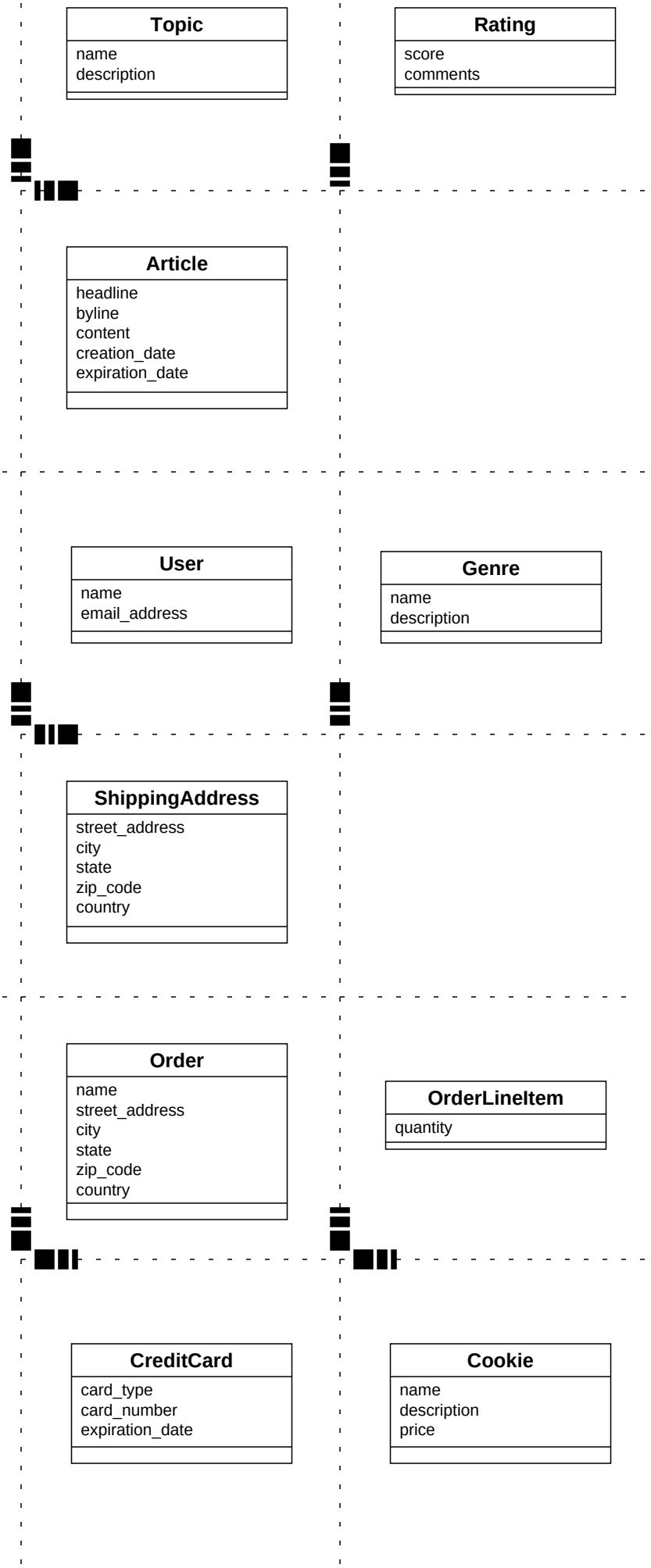
Assignment: find three concepts, then work together to find relationships and draw the conceptual model.

Use Case: Order Cookies

Summary: A user visits the cookie order form, selects some cookies for ordering. Next, the user supplies shipping and billing information to complete the order.

User Action	System Response
1. Visit the cookie order form.	2. Display a form of all cookies available, their price, and a quantity field next to each cookie.
3. Select a quantity for each cookie desired.	4. Prompt the user for name, shipping address and credit card information.
5. Supply requested information.	6. Prompt the user to confirm the information.
7. Confirm the information.	8. Process the order.

Assignment: find four concepts, then work together to find relationships and draw the conceptual model.



Exercises

Module One.

Exercise: Pass out the cards for module one. Ask the person with the class diagram to find the source code that implements their diagram. When they are done, they should perform the exercise as a group. If anyone is uncomfortable working in a group, I will give them a sheet with the classes and the diagram, and they can work on this on their own. They can just sit in their seat during the exercise, and I'll discreetly bring the sheet over to them.

Let the students know that there are only three different diagrams, and multiple copies, so it should be easier to find their matches.

Module Two.

Exercise: Pass out the cards for module two. Ask the person with the pattern to find class diagrams with implementations of the pattern. Note that the pattern can be used for problems other than the one stated on the pattern card, so think about the pattern in isolation from the problem, and consider how it might apply elsewhere.

When they are done, they should implement the class diagram in the language of their choice. No need to develop a full implementation - just write stub functions that print some message out and write a simple program to test out each class.

Module Three.

Exercise: Pass out the cards for module three. Ask the person with the use case to find concepts that belong in their conceptual model.

When they have collected all the matching concepts, draw the conceptual model. Then, implement stub implementations for the classes in the conceptual model, using their language of choice.

Module Four.

Exercise: With the use case from the previous exercise, develop a conceptual model that incorporates user interface elements.

Module Five.

Exercise: Develop a prototype for the use case from the previous module. Use the conceptual model you created in the previous step.

Use on-line pattern repositories or one of the books I have with me to find patterns that help you assign responsibility.

Module Six.

Choice One: Implement the remaining steps of the Select a Product use case.

Choice Two: Implement one of the use cases from the exercises.

Some (hopefully) Helpful Links

Brad Appleton's Object Orientation Links:
    http://www.enteract.com/~bradapp/links/oo-links.html

Open Directory Project: Patterns and Anti-Patterns:
    http://dmoz.org/Computers/Programming/Patterns_and_Anti-Patterns/

Design Patterns (Gamma, et al) converted to UML:
    http://www.tcm.hut.fi/~pnr/GoF-models/html/

Patterns Home Page:
    http://hillside.net/patterns/patterns.html

HTML Pattern Language:
    http://www.anamorph.com/docs/patterns/default.html

Patterns and Software: Essential Concepts and Terminology:
    http://www.enteract.com/~bradapp/docs/patterns-intro.html

Design Patterns in Java:
    http://www.ece.utexas.edu/~natrajan/Patterns.html

Lecture Notes from a Design class at SDSU:
    http://www.eli.sdsu.edu/courses/spring98/cs635/notes/index.html

- -

Some (hopefully) Helpful Links

Brad Appleton's Object Orientation Links:
    http://www.enteract.com/~bradapp/links/oo-links.html

Open Directory Project: Patterns and Anti-Patterns:
    http://dmoz.org/Computers/Programming/Patterns_and_Anti-Patterns/

Design Patterns (Gamma, et al) converted to UML:
    http://www.tcm.hut.fi/~pnr/GoF-models/html/

Patterns Home Page:
    http://hillside.net/patterns/patterns.html

HTML Pattern Language:
    http://www.anamorph.com/docs/patterns/default.html

Patterns and Software: Essential Concepts and Terminology:
    http://www.enteract.com/~bradapp/docs/patterns-intro.html

Design Patterns in Java:
    http://www.ece.utexas.edu/~natrajan/Patterns.html

Lecture Notes from a Design class at SDSU:
    http://www.eli.sdsu.edu/courses/spring98/cs635/notes/index.html