Workshop: How to Design Class Hierarchies

Viera K. Proulx

July 13, 2003

How to design class hierarchies: an introduction to object-oriented programming
Matthias Felleisen, Matthew Flatt, Robert Findler, Kathy Gray, Shriram Krishnamurthi, Viera K. Proulx
p. cm.
Includes index.
ISBN 0-262-06218-6 (hc.: alk. paper)
1. Computer Programming. 2. Electronic data processing.

QA76.6 .H697 2001
005.1'2—dc21 00-048169

1 Lab Monday am:

Do the following sets of problems from HtDP:

- Functions: 2.3.3, 3.1.1, 3.1.2, 3.1.3, and 3.3.2, 3.3.3, 3.3.4

Exercise 2.3.3 An old-style movie theater has a simple pro£t function. Each customer pays $5 per ticket. Every performance costs the theater $20, plus $.50 per attendee. Develop the function *total-pro£t*. It consumes the number of attendees (of a show) and produces how much income the attendees produce. ▮

Exercise 3.1.1 The next step is to make up examples for each of the functions. Determine how many attendees can afford a show at a ticket price of $3.00, $4.00, and $5.00. Use the examples to formulate a general rule that shows how to compute the number of attendees from the ticket price. Make up more examples if needed. ▮

Exercise 3.1.2 Use the results of exercise 3.1.1 to determine how much it costs to run a show at $3.00, $4.00, and $5.00. Also determine how much revenue each show produces at those prices. Finally, £gure out how much pro£t the monopolistic movie owner can make with each show. Which is the best price (of these three) for maximizing the pro£t? ▮

Exercise 3.1.3 Determine the pro£t that the movie owner makes at $3.00, $4.00, and $5.00 using the program de£nitions in both columns. Make sure that the results are the same as those predicted in exercise 3.1.2. ▮

Exercise 3.3.2 Develop the program *volume-cylinder*. It consumes the radius of a cylinder's base disk and its height; it computes the volume of the cylinder. ▮

Exercise 3.3.3 Develop *area-cylinder*. The program consumes the radius of the cylinder's base disk and its height. Its result is the surface area of the cylinder. ▮

Exercise 3.3.4 Develop the function *area-pipe*. It computes the surface area of a pipe, which is an open cylinder. The program consumes three values: the pipe's inner radius, its length, and the thickness of its wall.

Develop two versions: a program that consists of a single de£nition and a program that consists of several function de£nitions. Which one evokes more con£dence? ▮

- Structures: 6.3.1, 6.3.2, 6.4.1, 6.5.1

Exercise 6.3.1 Consider the following structure de£nitions:

1. (*de£ne-struct movie* (*title producer*))
2. (*de£ne-struct boyfriend* (*name hair eyes phone*))
3. (*de£ne-struct cheerleader* (*name number*))
4. (*de£ne-struct CD* (*artist title price*))
5. (*de£ne-struct sweater* (*material size producer*))

What are the names of the constructors and the selectors that each of them adds to Scheme? Draw box representations for each of these structures. ▮

Exercise 6.3.2 Consider the following structure de£nition

(*de£ne-struct movie* (*title producer*))

and evaluate the following expressions:

1. (*movie-title* (*make-movie* 'ThePhantomMenace 'Lucas))
2. (*movie-producer* (*make-movie* 'TheEmpireStrikesBack 'Lucas))

Now evaluate the following expressions, assuming x and y stand for arbitrary symbols:

1. (*movie-title* (*make-movie* x y))
2. (*movie-producer* (*make-movie* x y))

Formulate equations that state general laws concerning the relationships of *movie-title* and *movie-producer* and *make-movie*. ▮

Exercise 6.4.1 Provide data de£nitions for the following structure definitions:

1. (*de£ne-struct movie* (*title producer*))
2. (*de£ne-struct boyfriend* (*name hair eyes phone*))
3. (*de£ne-struct cheerleader* (*name number*))
4. (*de£ne-struct CD* (*artist title price*))
5. (*de£ne-struct sweater* (*material size producer*))

Make appropriate assumptions about what data goes with which £eld. ∎

Exercise 6.5.1 Develop templates for functions that consume the following structures:

1. (*de£ne-struct movie* (*title producer*))
2. (*de£ne-struct boyfriend* (*name hair eyes phone*))
3. (*de£ne-struct cheerleader* (*name number*))
4. (*de£ne-struct CD* (*artist title price*))
5. (*de£ne-struct sweater* (*material size producer*)) . ∎

- Unions: 7.2.1

Exercise 7.2.1 Develop structure and data de£nitions for a collection of zoo animals. The collection includes

spiders, whose relevant attributes are the number of remaining legs (we assume that spiders can lose legs in accidents) and the space they need in case of transport;

elephants, whose only attributes are the space they need in case of transport;

monkeys, whose attributes are intelligence and space needed for transportation.

Then develop a template for functions that consume zoo animals.

Develop the function *£ts?*. The function consumes a zoo animal and the volume of a cage. It determines whether the cage is large enough for the animal. ∎

- : Lists: 9.3.3, 9.5.1, 9.5.2, 9.5.3, 9.5.4

Exercise 9.3.3 Develop the function *contains?*, which consumes a symbol and a list of symbols and determines whether or not the symbol occurs in the list. ∎

Exercise 9.5.1 Use DrScheme to test the de£nition of *sum* on the following sample lists of numbers:

empty
(*cons* 1.00 *empty*)
(*cons* 17.05 (*cons* 1.22 (*cons* 2.59 *empty*)))

Compare the results with our speci£cations. Then apply *sum* to the following examples:

empty
(*cons* 2.59 *empty*)
(*cons* 1.22 (*cons* 2.59 *empty*))

First determine what the result *should* be; then use DrScheme to evaluate the expressions. ∎

Exercise 9.5.2 Develop the function *how-many-symbols*, which consumes a list of symbols and produces the number of items in the list.

Develop the function *how-many-numbers*, which counts how many numbers are in a list of numbers. How do *how-many-symbols* and *how-many-numbers* differ? ∎

Exercise 9.5.3 Develop the function *dollar-store?*, which consumes a list of prices (numbers) and checks whether all of the prices are below 1.

For example, the following expressions should evaluate to true:

(*dollar-store? empty*)

(*not* (*dollar-store?* (*cons* .75 (*cons* 1.95 (*cons* .25 *empty*))))) (*dollar-store?* (*cons* .75 (*cons* .95 (*cons* .25 *empty*))))

Generalize the function so that it consumes a list of prices (numbers) and a threshold price (number) and checks that all prices in the list are below the threshold. ∎

Exercise 9.5.4 Develop the function *check-range1*, which consumes a list of temperature measurements and checks whether all measurements are between 5^oC and 95^oC.

Generalize the function to *check-range*, which consumes a list of temperature measurements and a legal interval and checks whether all measurements are within the legal interval. ∎

- **Exercise:** Produce a list of all positive numbers form a given list of numbers.

2 Lab Monday pm:

Goals

Learn to make data de£nitions
Learn to use ProfJ

Simple classes

(2.1.1, 2.1.2, 2.1.3, 2.1.4) - do two of these

Exercise 2.1.1 Translate the three examples of information from the GPS problem into instances of *GPSLocation*.

Also interpret **new** *GPSLocation*(50.288,0.11) in this context. ∎

Exercise 2.1.2 Take a look at this problem statement:

> Develop a program that assists a bookstore manager. The program should keep a record for each book. The record must include its book, the author's name, its price, and its publication year.

Develop an appropriate data de£nition and implement the de£nition with a class. Create instances of the class to represent these three books:

1. Daniel Defoe, *Robinson Crusoe*, $15.50, 1719;
2. Joseph Conrad, *Heart of Darkness*, $12.80, 1902;
3. Pat Conroy, *Beach Music*, $9.50, 1996. ∎

Exercise 2.1.3 Study this class de£nition:

```
class Image {
  int height /* pixels */;
  int width /* pixels */;
  String source /* £le name */;
  String quality /* informal */;
  Image(int height, int width, String source, String quality) {
    this.height = height;
    this.width  = width ;
    this.source = source;
    this.quality = quality;
  }
}
```

Draw the class diagram.

The class de£nition was developed in response to this problem statement:

> Develop a program that creates a gallery from image descriptions. Those specify the height, the width, the source of the images, and, for aesthetic reasons, some informal information about its quality.

Interpret the following three instances in this context:

```
new Image(5, 10, "small.gif", "low");
new Image(120, 200, "med.gif", "low");
new Image(1200, 1000, "large.gif", "high");
```
∎

```
+------------------------+
| Car                    |
+------------------------+
| String model           |
| int price /* dollars */ |
| double milage          |
| boolean used           |
+------------------------+
```

Figure 1: A class diagram for cars

Exercise 2.1.4 Translate the class diagram in £gure 1 into a class de£nition. Also create instances of the class. ∎

```
+-----------------------------+
| WeatherRecord               |
+-----------------------------+
| Date d                      |-----------------+
| TemperatureRange today      |--+              |
| TemperatureRange normal     |--+              |
| TemperatureRange record     |--+              |
| double precipitation        |  |              |
+-----------------------------+  |              |
                                 v              v
              +---------------------+   +------------+
              | TemperatureRange    |   | Date       |
              +---------------------+   +------------+
              | int high            |   | int day    |
              | int low             |   | int month  |
              +---------------------+   | int year   |
                                        +------------+
```

Figure 2: A class diagram for weather records

Classes with containment

(3.1.1, 3.1.2, 3.1.3) - do two of these

Exercise 3.1.1 Develop a data de£nition and Java classes for this problem:

> Develop a "real estate assistant" program. The "assistant" helps the real estate agent locate houses of interest for clients. The information about a house includes its kind, the number of rooms, its address, and the asking price. An address consists of a street number, a street name, and a city.

Represent the following examples using your classes:

1. Ranch, 7 rooms, $375,000, 23 Maple Street, Brookline;
2. Colonial, 9 rooms, $450,000, 5 Joye Road, Newton; and
3. Cape, 6 rooms, $235,000, 83 Winslow Road, Waltham. ∎

Exercise 3.1.2 Take a look at the data de£nition in £gure 2. Translate it into a collection of classes. Also create examples of weather record information and translate them into instances of the matching class. ∎

Exercise 3.1.3 Revise the data representation for the book store assistant in exercise 2.1.2 so that the program keeps track of an author's year of birth in addition to its name. Modify class diagram, the class de£nition, and the examples. ∎

Union

(4.1.2, 4.1.3, 4.1.4) - do two of these

Exercise 4.1.1 Consider a revision of the problem statement in exercise 2.1.3:

> Develop a program that creates a gallery from three different kinds of records: images (gif), texts (txt), and sounds (mp3). All have names for source £les and sizes (number of bytes). Images also include information about the height, the width, and the quality of the image. Texts specify the number of lines needed for visual representation. Sounds include information about the playing time of the recording, given in seconds.

Develop a data de£nition and Java classes for representing these records. Then represent these three examples with Java objects:

1. an image, stored in `flower.gif`; size: 57,234 bytes; width: 100 pixels; height: 50 pixels; quality: medium;
2. a text, stored in `welcome.txt`; size: 5,312 bytes; 830 lines;
3. a music piece, stored in `theme.mp3`; size: 40960 bytes, playing time 3 minutes and 20 seconds. ∎

Exercise 4.1.2 Take a look at the data de£nition in £gure 3. Translate it into a collection of classes. Also create instances of each class. ∎

Exercise 4.1.3 Draw a UML diagram for the classes in £gure 4. ∎

```
                +---------------------+
                | ATaxiVehicle        |
                +---------------------+
                | int idNum           |
                | int passangers      |
                | int pricePerMile    |
                +---------------------+
                        / \
                        ---
                         |
     +----------------+--+-----------------+
     |                |                    |
+--------+     +---------------+   +----------------+
| Cab    |     | Limo          |   | Van            |
+--------+     +---------------+   +----------------+
|        |     | int minRental |   | boolean access |
+--------+     +---------------+   +----------------+
```

Figure 3: A class diagram for taxis

```
abstract class AMuseTicket {
Date d;
int price;
}
```

```
class MuseAdm
  extends AMuseTicket {
  MuseAdm(Date d,
             int price) {
    this.d = d;
    this.price = price;
  }
}
anewline
anewline
anewline
anewline
anewline
anewline
anewline
anewline
anewline
anewline
```

```
class OmniMax
  extends AMuseTicket {
  ClockTime t;
  String title;
anewline
  OmniMax(Date d,
             int price,
             ClockTime t,
             String title) {
    this.d = d;
    this.price = price;
    this.t = t;
    this.title = title;
  }
}
anewline
anewline
anewline
```

```
class LaserShow
  extends AMuseTicket {
  ClockTime t;
  String row;
  int seat;
anewline
  LaserShow(Date d,
             int price,
             ClockTime t,
             String row,
             int seat) {
    this.d = d;
    this.price = price;
    this.t = t;
    this.row = row;
    this.seat = seat;
  }
}
```

Figure 4: Some classes

3 Lab Tuesday am:

Designing classes that represent lists and trees

Containment in Union — Lists

(5.1.2, 5.1.3, 5.1.4) - do two of these

Exercise 5.1.2 Consider a revision of the problem in exercise 3.1.1:

> Develop a program that assists real estate agents. The program deals with listings of available houses. . . .

Make examples. Develop a data de£nitions for listings of houses. Implement the de£nition with Java classes. Translate the examples into Java instances. ∎

Exercise 5.1.3 Consider a revision of the problem in exercise 2.1.2:

> Develop a program that assists a bookstore manager with reading lists. . . .

The diagram in £gure 5 represents the data de£nitions for classes that represent reading lists. Implement the de£nitions with classes. Create two book lists that contain at least one of the books in exercise 2.1.2 plus one or more of your favorite books. ∎

Exercise 5.1.4 Take a look at £gure 6, which contains the data de£nition for weather reports. A weather report is a sequence of weather records (see exercise 3.1.2). Translate the diagram into a collection of classes. Also represent two (made-up) two weather reports, one for your home town and one for your college town, in Java. ∎

Containment in Union — Trees

(5.3.1, 5.3.2) - do at leat one of these

Exercise 5.3.1 Consider the following problem:

```
       +----------------+
       | AReadingList   |<-----------------+
       +----------------+                  |
              / \                          |
              ---                          |
               |                           |
               |                           |
     +---------+---------+                 |
     |                   |                 |
+---------+          +------------------+  |
| MTLoB   |          | ConsLoB          |  |
+---------+          +------------------+  |
             +-------| Book fst         |  |
             |       | AReadingList rst |--+
             |       +------------------+
             v
     +-----------------+
     | Book            |
     +-----------------+
     | String author   |
     | String title    |
     | int price       |
     | int year        |
     +-----------------+
```

Figure 5: A class diagram for reading lists

> Develop program that helps with recording a person's ancestry tree. Speci£cally, for each person we wish to remember the person's name and year of birth, in addition to the ancestry on the father's and the mother's side, if it is available.

See £gure 7 for an example of the relevant information.

Develop the class diagram and the Java class hierarchy that represents the information in an ancestry tree. Then translate the sample tree into Java code. Also draw your family's ancestor tree as far as known and represent it as a Java object. ∎

```
          +----------------+
          | AWR            |<----------------+
          +----------------+                |
                 / \                        |
                 ---                        |
                  |                         |
                  |                         |
        +---------+---------+               |
        |                   |               |
   +---------+       +-------------------+  |
   | MTWR    |       | ConsWR            |  |
   +---------+       +-------------------+  |
                +----| WeatherRecord fst |  |
                |    | AWR rst           |--+
                |    +-------------------+
                v
+-----------------------------+
| WeatherRecord               |
+-----------------------------+
| Date d                      |-----------------+
| TemperatureRange today      |--+              |
| TemperatureRange normal     |--+              |
| TemperatureRange record     |--+              |
| double precipitation        |  |              |
+-----------------------------+  |              |
                                 |              |
                                 v              v
              +---------------------+   +------------+
              | TemperatureRange    |   | Date       |
              +---------------------+   +------------+
              | int high            |   | int day    |
              | int low             |   | int month  |
              +---------------------+   | int year   |
                                        +------------+
```

Figure 6: A class diagram for weather reports

Exercise 5.3.2 Take a look at the class diagram for a program that manages a phone tree (like those for a soccer team). To inform the team about rain-outs and schedule changes, the league calls the coach, who in turn calls the

```
                        Bob
                        1917
                        |
+--+--+  +--+--+    +--+--+
   |        |          |
Angela   Robert     Annie
 1936      1935     1938
   |        |          |
   +----+----+         +----+----+
        |                   |
     Janet              Paul
     1958               1956
        |                   |
       +---------+---------+
                 |
               Peter
               1980
```

Figure 7: A family tree

team captain. Each player then calls at most two other players.

Translate the following examples into pictures of phone trees:

```
Player coach = new Player("Bob", 5432345);
Player p1 = new Player("Jan", 5432356);
Player p2 = new Player("Kerry", 5435421);
Player p3 = new Player("Ryan", 5436571);
Player p4 = new Player("Erin", 5437762);
Player p5 = new Player("Pat", 5437789);

APT empty = new MTTeam();

APT pt =
    new PhoneTree(
        p2,
        new PhoneTree(p3, empty, empty),
        new PhoneTree(
            p4,
            new PhoneTree(p5, empty, empty),
            new PhoneTree(p1, empty, empty)));
```

```
       +--------------+
       | Coach        |
       +--------------+
+------| Player p     |
|      | APT players  |-+  +---------------+
|      +--------------+ |  |  +---------+  |
|                       |  |  |         |  |
|                       v  v  v         |  |
|                 +-------------+       |  |
|                 | APT         |       |  |
|                 +-------------+       |  |
|                      / \              |  |
|                      ---              |  |
|                       |               |  |
|           +-----------+-----+         |  |
|           |                 |         |  |
|    +------------+  +--------------+   |  |
|    | MTTeam     |  | PhoneTree    |   |  |
|    +------------+  +--------------+   |  |
|           +--------| Player p     |   |  |
+------+    |        | APT call1    |---+  |
       |    |        | APT call2    |------+
       v    v        +--------------+
  +--------------+
  | Player       |
  +--------------+
  | String name  |
  | int phone    |
  +--------------+
```

Figure 8: A class diagram for a phone tree

Coach ch = **new** *Coach*(*coach*, *pt*);

Now develop Java code that corresponds to the given data de£nition. ▮

4 Lab Tuesday pm:

Design methods, build a test suite, introduction to templates

Methods for simple classes and classes with containment

(9.1.2, 9.1.3, 9.1,4, 9.1.5)

Exercise 9.1.2 Draw the complete diagram for *Coffee* with all the method signatures included in the method compartment. ∎

Exercise 9.1.3 Remember the class *Image* from exercise 2.1.3 for creating Web pages. Develop the following methods for this class:

1. *isPortrait*, which determines whether the image is taller than wider;
2. *size*, which computes how many pixels the image contains;
3. *isLarger*, which determines whether one image contains more picture than some other image. ∎

Exercise 9.1.4 Develop the following methods for the class *House* from exercise 3.1.1:

1. *isBigger*, which determines whether one house has more rooms than some other house;
2. *thisCity*, which checks whether the advertised house is in some given city (assume we give the method a city name);
3. *sameCity*, which determines whether one house is in the same city as some other house.

Don't forget to test these methods. ∎

Exercise 9.1.5 Here is a revision of the problem of managing a runner's log (see £gure **??**):

> Develop a program that manages a runner's training log. Every day the runner enters one entry concerning the day's run. . . . For each entry, the program should compute how fast the runner ran. . . .

Develop a method that computes the pace for a daily entry. ∎

5 Lab Wednesday am:

Design recipes and methods for composition

Methods for composition

(10.1.1, 10.1.2)

Exercise 10.1.1 Recall the problem of writing a program that assists a book store manager (see exercise 3.1.3). Develop the following methods for this class:

- *currentBook* that checks whether the book was published in 2003 or 2002;
- *currentAuthor* that determines whether a book was written by a current author (born after 1940);
- *thisAuthor* that determines whether a book was written by the speci£ed author;
- *sameAuthor* that ddetermines whether one book was written by the same author as some other book;
- *sameGeneration* that determines whether two books were written by two authors born less than 10 year apart. ∎

Exercise 10.1.2 Exercise 3.1.2 provides the data de£nition for a weather recording program. Develop the following methods:

1. *withinRange* that determines whether today's *high* and *low* were within the normal range;
2. *rainyDay* that determines whether the precipitation is higher than some given value;
3. *recordDay* that determines whether the temperature broke either the high or the low record;
4. *warmerThan* that determines whether one day was wormer than another day;
5. *lowerRecord* that determines whether the record low for one day was lower than for some other day. ∎

6 Lab Wednesday pm:

Methods for Unions

(12.1.1, 12.1.2, 12.1.3, 12.1.4) - do two of these

Exercise 12.1.1 Recall problem 4.1.1: that described a revision of the problem in exercise 2.1.3:

> **Problem** Develop class de£nitions for a collection of classes that represent several different kinds of £les. All £les have name and size given in bytes. Image £les (gif) also include information about the height and width of the image, and the quality of the image. Text £les (txt) include information about the number of characters, words, and lines. Sound £les (mp3) include information about the playing time of the recording, given in seconds.

1. Develop the method *timeToDownload* that computes how long it takes to download a £le at some network connection speed, typically given in bytes per second. The size of the £le is given in bytes.
2. Develop the method *smallerThan* that determines whether the £le is smaller than some maximum size that can be mailed as an attachment.
3. Develop the method *sameName* that determines whether the name of a £le is the same as some speci£ed name. ▮

Exercise 12.1.2 Recall the problem **??**:

> **Problem** Develop a program that keeps track of the items in the grocery store. For now, we assume that the store deals only with ice cream, coffee, and juice. Each of the items is speci£ed by its brand name, weight and price. Each coffee is also labelled as either regular or decaffeinated. Juice items come in different ¤avors, and can be packaged as frozen, fresh, bottled, or canned. Each package of ice cream speci£es its ¤avor and whether this is a sorbet, a frozen yogurt, or regular ice cream.

1. Develop the method *unitPrice* that computes the unit price of some grocery item. The unit price is given in cents per unit of weight.

2. Develop the method *lowerPrice* that determines whether the unit price of some grocery item is lower than some amount.

3. Develop the method *cheaperThan* that determines whether one grocery item is cheaper than some other one, in terms of the unit cost. ▮

Exercise 12.1.3 Recall the exercise 4.1.2: Take a look at the data de£nition in £gure 9. Translate it into a collection of classes. Also create instances of

```
                    +---------------------+
                    | ATaxiVehicle        |
                    +---------------------+
                    | int idNum           |
                    | int passangers      |
                    | int pricePerMile    |
                    +---------------------+
                            / \
                            ---
                             |
         +----------------+--+----------------+
         |                |                   |
+--------+     +---------------+   +----------------+
| Cab    |     | Limo          |   | Van            |
+--------+     +---------------+   +----------------+
|        |     | int minRental |   | boolean access |
+--------+     +---------------+   +----------------+
```

Figure 9: A class diagram for taxis

each class.

1. Develop the method *fare* that computes the fare in a given vehicle, based on the number of miles travelled, and using the following formulas for different vehicles.

   (a) passengers in a cab just pay ¤at fee per mile

   (b) passengers in a limo must pay at least the minimum rental fee, otherwise they pay by the mile

   (c) passengers in a van pay $1.00 extra for each passenger

2. Develop the method *lowerPrice* that determines whether the fare for a given number of miles is lower than some amount.
3. Develop the method *cheaperThan* that determines whether the fare in one vehicle is lower than the fare in another vehicle for the same number of miles. ∎

Exercise 12.1.4 Consider the following revision of the problem in exercise 2.1.2:

> **Problem** Develop a program that assists a bookstore manager in a discount bookstore. The program should keep a record for each book. The record must include its title, the author's name, its price, and its publication year. In addition, the books There are three kinds of books with different pricing policy. The hardcover books are sold at 20% off. The sale books are sold at 50% off. The paperbacks are sold at the list price.

1. Develop the class hierarchy to represent books in the discount bookstore.
2. Develop the method *salePrice* that computes the sale price of each book.
3. Develop the method *cheaperThan* that determines whether on book is cheaper than another book.
4. Develop the method *sameAuthor* that determines whether some book was written by the speci£ed author. ∎

7 Lab Thursday am:

Methods for lists

(14.1.1, 14.1.2, 14.1.3), (optional 14.1.4)

Exercise 14.1.1 Develop the data de£nition for a list of grocery items selected among those described in 12.1.2.

1. Develop the method *total* that computes the total price of the purchase.
2. Develop the method *brandlist* that produces a list of all items with the speci£ed brand name.
3. Develop the method *highestPrice* that determines the highest unit price among all items in the shopping list. ∎

Exercise 14.1.2 Develop the data de£nition for a shopping list of grocery items selected among those described in 12.1.2. This time, the list also records how many of each item do we need.

1. Develop the data de£nition for this class hierarchy.
2. Develop the method *total* that computes the total price of the purchase.
3. Develop the method *saletotal* that computes the total price of the purchase when all items with the speci£ed brand name are on sale at 20% off. ∎

Exercise 14.1.3 Develop a program that assists a bookstore manager in a discount bookstore (see exercise 12.1.4.

1. Develop the class hierarchy to represent a list of books in the discount bookstore.
2. Describe in English examples of three book lists and represent them as objects in this class hierarchy.
3. develop the method *price* that computes the total for the sale, based on the sale price of each book.

4. Develop the method *thisAuthor* that produces a list of all books by this author in the bookstore list. ∎

Exercise 14.1.4 Develop the following additional methods for the river system.

1. Develop the method *maxlength* that computes the length of the longest river segment.

2. Develop the method *con¤uences* that counts the number of con¤uences in the river system.

3. Develop the method *locations* that produces a list of all locations on this river - the sources, the mouths, and the con¤uences. ∎

Methods for trees and similar structures

- Refer to the problem 5.3.1: develop methods to count the number of ancestors, to determine whether there is a person with some name in the tree, etc.

- Refer to the problem 5.3.2: Develop methods to £nd whether a given player is in the list, count the players, etc.

8 Lab Thursday pm:

Methods for cyclic class hierarchies

> **Problem** Develop a program that assists a bookstore manager. The manager's program should keep a record for each book. The record must include information about the author, the book's title, its price, and its publication year. The information about the author includes author's name, year of birth, and a list of books written by this author.

Write programs that help the bookstore manager solve the following problems:

- Find out how many books did some writer publish.
- Produce a list of all books in his store written by modern authors - those born after 1940.
- Find out which one of the two authors wrote more books.
- Produce a list of all books some author published during the past three years.
- Produce a list of all authors who published a book this year, if you have a list of all authors.
- Produce a list of all authors who published books in the bookstore manager's list.

9 Lab Friday am:

Implementing interfaces - Lists of Objects

Exercise
Return to the exercises 14.1.1, 14.1.2, and 14.1.3.

- Follow the design recipe for abstractions: highlight the differences in the class de£nitions, and observe the similarities.
- Design the interface *IPriced* that contains the method *price*(). Assume the price is given in cents as a whole number.
- Rewrite the classes in exercises 14.1.1, 14.1.2, and 14.1.3, so that they now implement the *IPriced* interface.
- Develop the class hierarchy to represent a list of objects from classes that implement the *IPriced* interface.
- Develop the method *totalPrice* that computes the total price of all items in this list.
- Rewrite the examples from exercises 14.1.1, refex:methods-shoppinglist, and 14.1.3, using the list of *Object*s.
- Design the methods *lowerPrice* and *cheaperThan* in the classes that represent the list of *IPriced* objects.
- Design the necessary abstractions to implement the *sort* method in the classes that represent a list of *Object*s. You can use the *price* method, or design and implement an additional interface.

10 Lab Friday pm:

Inner classes; Free time

Note: This is a preliminary version of this exercise.

> **Problem** University keeps records for all students and instructors at the university. For each person it records the name and id number. Each instructor also has a title, and is a member of some department (for now, we just know the name of the department). Each student has a major and grade point average (GPA).

De£ne the class hierarchy to represent students and instructors at the university, as well as a list of all people, all students, and all instructors.

De£ne the following methods for these classes:

- *sortStName* that sorts the students in a list of *Student*s by their name.
- *sortStGPA* that sorts the students in a list of *Student*s by their GPA.
- *sortInsName* that sorts the instructors in a list of *Instructor*s by their name.
- *sortInsDept* that sorts the instructors in a list of *Instructor*s by their department.

Work on refactoring this code as follows:

- Study the Java *Comparator* interface.
- In the class *Person* de£ne a *Comparator* object that compares the persons by their name.
- In the class *Student* de£ne a *Comparator* object that compares the students by their GPA.
- In the class *Instructor* de£ne a *Comparator* object that compares the persons by their department.
- Refactor your code to use a list of *Object*s and rewrite the *sort* method so that it receives as a parameter an object in the class that implements *Comparator* interface.
- Add method to £nd whether an object is in the list.