2 Understanding Data: Simple Classes

In this lab we will focus on understanding data definitions, the distinction between information and data, how information can be represented as data, how to interpret the information that some instance of data represents, and learn to encode the data definitions, as well as construct instances of data in a class based language (like Java).

2.1 Data Definitions in HtDP

1. Open in DrScheme the file from the previous lab that dealt with the radio show and the ads for the show. Save a new copy, and delete any function definitions from the file. Make sure a data definition for each class of data includes a purpose statement that explains what information is represented by the instances of data in this class of data.

2. Make sure your program includes examples of the following information:

   - an iPod ad that lasts 2 minutes and brings in $100 profit.
   - an MS ad that lasts 1 minute and brings in $500 profit.
   - an XBox ad that lasts 2 minutes and brings in $300 profit.
   - a news show that runs the ads for iPod, MS, iPod again, and XBox and lasts 60 minutes.
   - a game show that lasts 120 minutes and runs the ads for iPod, MS, iPod and MS again, then XBox.

Additional example

3. Each of the following pieces of information represents a file in a computer directory. Design the Scheme data definitions necessary to represent this information and convert the information into Scheme data.

   - Picture of a river (jpeg) that is 3456 pixels wide and 2304 pixels high, using up 3,614,571 bytes.
   - Picture of a mountain (jpeg) that is 2448 pixels wide and 3264 pixels high, using up 1,276,114 bytes.

- Picture of a group of people (gif) that is 545 pixels wide and 641 pixels high, using up 13,760 bytes.
- Picture of a plt icon (bmp) that is 16 pixels wide and 16 pixels high, using up 1334 bytes.

2.2 Data Definitions in ProfessorJ

Open a new Tab and set the language to *Beginner ProfessorJ*.

1. Design the Java class to represent an ad in a radio show. Name the class *Ad*. Create a class *Examples* at the end of your *Definitions* as follows:

```
class Examples {
  Examples() {}

  // examples of ad data
  Ad ipodAd = ...
  Ad msAd = ...
  Ad xboxAd = ...
}
```

   You will be defining all your sample data in the *Examples* class. Later, it will also be a place where you put all your tests.

   Include in the *Examples* class definitions of all data that you have defined in the HtDP Beginner language.

2. Once you are done, run the program. You should see an instance of the *Examples* class in the *Interactions* window.

Additional example

3. Design the Java class to represent an iPhoto picture. Name the class *Photo*. Create a class *Examples* at the end of your *Definitions* as follows:

```
class Examples {
  Examples() {}

  // examples of photo data
  Photo p1 = ...
  Photo p2 = ...
}
```

4. Again, once you are done, run the program. You should see an instance of the *Examples* class in the *Interactions* window.

2.3 Data With Containment; Using the Wizard

Now that you have done the tedious typing, but understand what are the parts of a Java class definition, you can rewrite all your data definitions (except for the *Examples* class) using the *Wizard*. Observe first the similarities between the data definitions in *Beginner HtDP* and data definitions in *ProfessorJ* languages. To define a *ProfessorJ* class we only need to know the name of the class, its purpose, and for each field its *type* and its name. This information completely specifies what will be included in the class definition.

1. Start by looking at the classes in the textbook that represent the the clock time. The *ClockTime* class is defined on page 21.

   Open a new Tab using the *Beginner ProfessorJ* language. In the *Special* menu select *Insert Java Class*. Type in the class name *ClockTime*, add the purpose statement and the field information. Check the *add class diagram* checkbox. The code and the diagram is generated for you. (Do not check the *add toString method* checkbox!)

   Now define the *Examples* class and make examples of *ClockTime* data.

2. Now define the class *Show* that represents a radio show in a manner similar to the earlier Scheme definition. However, at this point we will ignore the ads, but will keep track of both the start time for the show and the end time.

   Work out the data definition and draw the class diagram for this class.

3. Make examples of two radio shows with the information specified above.

4. Use the *Insert Java Class* wizard to add the data definition for a *Show* to your program and translate your examples into data inthe *Examples* class.

5. Combine the class diagrams for the classes *Show* and *ClockTime* using the *containment* arrows where appropriate.

Additional example

6. The photo file information also includes the time when the photo file was created. Design a new *Photo* class that includes this information.

Again, use the *Insert Java Class* wizard to generate the class diagram and the Java code.

7. Adjust the class diagram by including the diagrams for the *ClockTime* class with the appropriate *containment* arrows.

8. Make sure to include examples of the instances of the new *Photo* class in your *Examples* class.

9. Write down the data definitions for these classes of data in the *Beginner HtDP* language as well.

2.4 Unions of Data

The FCC requires that the radio runs not only commercially profitable ads, but also public service ads. Public service ads do not generate any profit, but the station records the code for the ad (a *String*) as well as the duration of the ad.

1. Adjust the *Beginner HtDP* data definition for *Ad* so that we can now represent two variants of ads. Make examples of data as well.

Additional example

You now have a new camera that allows you to take not only still pictures, but also short videos. You want to keep the two kinds of files together. Here is a data definition for the data you may want - written in the *Beginner HtDP* language:

```
;; Data to represent a camera shot:
;; either a still photo or a video clip

;; A Shot is one of
;; – Photo
;; – Video

;; we omit the definition of the Photo
;; as you already have done that
```

```
;; A Video is (make-video String String Number Number Boolean)
(define-struct video (name kind size duration sound?))
;; Interpretation:
;; kind of video may be either QuickTime or RealPlayer
;; size is measured in bytes
;; duration is measured in seconds
;; sound? indicated whether or not this video has sound
```

2. Define the Java classes that correspond to the given data definition by sketching the class diagram on a paper. Also, make examples of data - by hand.

3. Add the field that represents the time when the video was created to the class Video. Now use the *Insert Java Union* wizard to convert the data definitions to class diagrams and Java code.

4. **Do we have to remind you to make examples of data???**

2.5 Lists of Data

Recall the data definition for a radio show we have done in Lab 1. It included a list of ads.

1. Represent these data definitions as a class diagram on a paper.

2. Observe the data definition for the class *ListofAds*. It defines a union. What happens to the class of data that represents an empty list of ads? How many examples can you make of data that belongs to this class of data? What are the names of the two fields for the *ConsLoAds* class? What are their types?

3. You can now convert this data definition into Java data definitions. Do so.

4. **Do we have to remind you to make examples of data???**

Additional example

We continue with the the theme of the photo images. Our collections of images surely contains more than one picture. We will first define a list of images.

Once we have the data that represents our pictures, we would like to be able to ask questions about the data. To do so, we design methods in the classes that represent our data.

5. The HtDP data definition for a list of photos is:

   ```
   ;; A ListOfPhotos is one of
   ;; — empty
   ;; — (cons Photo ListOfPhotos)
   ```

   Design the Java classes to represent a list of *Photo*s. Remember to make examples of data.

   Note: This is the last time we will remind you to make examples of data.

Save all your work — the next lab will build on the work you have done here!

If you have some time left, work on the *Portfolio* part of the homework.