2 FunJava: Understanding Data

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).

We will work in a professional *Integrated Development Environment (IDE)* **Eclipse** using the language *FunJava*.

2.1 Eclipse IDE and the tester library

Goals

In the first part of this lab you will learn how to work in a commercial level integrated development environment IDE Eclipse, using the standard Java programming language. The environment provides an editor, allows you to organize your work into several files that together comprise a project, and has a compiler so you can run your programs. Several projects form a workspace. You can probably keep all the work till the end of the semester in one workspace, with one project for each programming problem or a lab problem.

There are several step in getting started:

1. Learn to set up your workspace and launch an Eclipse project.
2. Learn to manage your files and save your work.
3. Learn how to edit *FunJava* programs and run them, using the *tester* library.

Learn to set up your workspace.

Start working on two adjacent computers, so that you can use one for looking at the documentation and the other one to do the work. Find the web page on the *documentation* computer:

http://www.ccs.neu.edu/howto/howto-windows-n-unix-homedirs.html

and follow the instructions to log into your Windows/Unix account on the *work* computer.

Next, set up a workspace folder in your home directory where you will keep all your Java files. This should be in

```
z:\\...\EclipseWorkspace
```

Note that `z:` is the drive that Windows binds your UNIX home directory.

Next, set up another folder in your home directory where you will keep all your Java library files. This should be in

```
z:\\...\EclipseJars
```

We will refer to these two folders as *EclipseWorkspace* and *EclipseJars*. Make sure the two folders *EclipseWorkspace* and *EclipseJars* are in the same folder.

Start the Eclipse application.

DO NOT check the box that asks if you want to make this the default workspace for Eclipse if you are working on the lab computer. If you are working at home or using your laptop, you may want to make the selected workspace to be your default.

Working at home: If your home computer does not have Java compiler installed, please, consult the wiki, or ask one of the TAs or tutors to help you.

The First Project

1. Download the libraries we will use. The libraries you will need are available at a public web site at:

   http://www.ccs.neu.edu/javalib/

   Go to the *Downloads* folder and download the following libraries into your *EclipseJars* folder:

   - tester.jar
   - draw.jar
   - geometry.jar
   - colors.jar

When saving the downloaded file, the dialog asks you *Do you want to open or save this file*. Choose *save*. It then comes up with a *Save as* window. Browse to find your *EclipseJars* folder and on the bottom where it says **Save as type** instead of *WINRAR archive* choose *All Files*. Do this for all Java libraries, otherwise Windows messes up the file.

2. You will also need the library `funjava.jar` that handles the *FunJava* language. The link is

   http://www.ccs.neu.edu/home/vkp/2510-sp10/Labs/Lab2/funjava.jar

3. Create a project.

   - In the *File* menu select *New* then *Java Project*. In the window that appears in the *Project layout* section select *Create separate folders for sources and class files* and select *Next*. We assume you have named it *MyProject*.
   - In the *Java Settings* pane select the *Libraries* tab.
   - On the right click on *Add External JARs...*
   - You will get a chooser window. Navigate to your *EclipseJars* folder and select all *jar* files you have downloaded.
   - Hit *Finish*.

4. Add the *Shapes.java* file to your project.

   - Download the file *Shapes.java* to a temporary directory or the desktop.
   - In Eclipse highlight the *src* box under the *MyProject* in the *Package Explorer* pane.

     Note: If the pane is not visible, go to *Window* menu, select *Show View...* then *Package Explorer*. You should also select *Show View... Outline*.
   - In the *File* menu select *Import...*.
   - Choose the *General* tab, within that *File System* and click on *Next*.
   - Browse to the temporary directory that contains your *Shapes.java* file.
   - Click on the directory on the left, then select the *Shapes.java* file in the right pane and hit *Finish*.

5. View and edit a FunJava file Shapes.java.

   - Click on the *src* block under *MyProject* in the *Pacakage Explorer* pane. It will reveal *default package* block.
   - Click on the *default package* block. It will reveal *Shapes.java*.
   - Double click on *Shapes.java*. The file should open in the main pane of *Eclipse*. You can now edit it in the usual way. Notice that the *Outline* pane lists all classes defined in this file as well as all fields and methods. It is almost as if someone was building our templates for us.
   - The TAs will guide you through setting that will convert all tabs into spaces, and will show you how to set the editor to show you the line numbers for all lines in the code.
   - Add one new example of data for each class: `Circle`, `Rect`, and `Combo`.

6. Set up the run configuration and run the program.

   - Highlight *MyProject* in the *Package Explorer* pane.
   - In the *Run* menu select *Run Configurations....*
   - In the top left corner of the inner pane click on the leftmost item. When you mouse over it should show *New launch configuration*.
   - Select the name for this configuration - usually the same as the name of your project.
   - In the *Main class:* click on *Search....*
   - Among *Matching items* select *FunJava (default package)* and hit *OK*.
   - Click on the tab `(x)= Arguments`. In the *Program arguments* text field enter ”src\Shapes.java” (if you are running Mac/OS or Linux, use ”src/Shapes.java”).

     Later, when you define your own program, you will use your file name instead of *Shapes.java*. **Make sure your file name does not have spaces in it.**
   - At the bottom of the *Run Configurations* select *Apply* then *Run*.
   - Next time you want to run the same project, make sure *Shapes.java* is shown in the main pane, then hit the green circle with the white triangle on the top left side of the main menu.

2.2 Data Definitions in FunJava

Look at the following data definitions in the *Beginner Student HtDP* language:

```
;; Sample data definitions -- simple classes of data

;; to represent a pet
;; A Pet is (make-pet String Num String)
(define-struct pet (name weight owner))

;; Examples of pets:
(define kitty (make-pet "Kitty" 15 "Pete"))
(define spot (make-pet "Spot" 20 "Jane"))
```

1. Draw the class diagram for this data definition.

2. Create a new project in Eclipse, create a file *Pet.java* and convert the data definition to the *FunJava* language — including the examples of data.

3. Create a new *Configuration* and run the examples.

   If you are comfortable with this material, you may omit the next two questions.

4. Convert the following class diagram into *FunJava* language:

```
+--------------+
| Restaurant   |
+--------------+
| String name  |
| String kind  |
| int avgPrice |
+--------------+
```

5. Convert the following information to data examples for your `Restaurant` class.

   - Chinese restaurant Blue Moon with average price per dinner $15
   - Japanese restaurant Kaimo with average price per dinner $20
   - Mexican restaurant Cordita with average price per dinner $12

2.3 Understanding Data: Classes with Containment

Look at the following data definitions in the *Beginner Student HtDP* language:

```
;; to represent a pet
;; A Pet2 is (make-pet String Num Person)
(define-struct pet2 (name weight owner))

;; to represent a person - a pet’s owner
;; A Person is (make-person String Num Boolean)
(define-struct person (name age male?))

;; Examples of person data:
(define pete (make-person "Pete" 15 true))
(define jane (make-person "Jane" 19 false))

;; Examples of pet2 data:
(define kitty2 (make-pet "Kitty" 15 pete))
(define spot2 (make-pet "Spot" 20 jane))
```

1. Draw the class diagram for this data definition.

2. Convert the data definition to the *FunJava* language — including the examples of data.

   If you are comfortable with this material, you may omit the next two questions.

3. Convert the following class diagram into *FunJava* language:

```
+--------------+
| Restaurant2  |
+--------------+
| String name  |
| String kind  |
| int avgPrice |
| CartPt loc   |--+
+--------------+  |
                  v
             +--------+
             | CartPt |
             +--------+
             | int x  |
             | int y  |
             +--------+
```

4. Make new examples for your `Restaurant2` class.

2.4 Understanding Data: Unions of Classes

The class of data that represent pets in the first part is not really sufficient. We have no idea what kind of pet the animal is. We would like to distinguish between the following kinds of pets:

- **cats** where we record whether it is a short-hair cat of a long-hair cat
- **dogs** where we record the breed (e.g. Husky, Labrador, etc., or Mutt — describing an unknown breed)
- **gerbils** where we need to know whether it is a male of female

We need a data definition for pets that covers all these options. Of course, we still keep track of the name of the animal and of its owner.

1. Make examples (in English words) of at least one of each kind of pets.
2. Draw a class diagram for the class hierarchy that represents this information about pets.
3. Design data definitions for this data in the *FunJava* language.
4. Convert your examples to data.

2.5 Representing Self-Referential Data

We want to trace your ancestry. Write down the name of your mother and your father, for each of them the name of their mother and father - as far as you can trace your ancestors. Write *unknown* when you no longer know the names. Organize your ancestor information into a tree-like structure - you are the root, your parents are the two branches, and each set of parents represents the two branches above their child. (You do not need to use actual names — feel free to make up the names of your ancestors — but go back to at least one great-grandparent.)

Design data definition that can be used to represent this information and then convert the information about your ancestry into data.

Follow the DESIGN RECIPE for data definitions:

- Is the information simple enough to be represented by a primitive data type?
- Are there several pieces of information that represent one entity? — if yes, design a class of data with a field for each piece of information.
- Is any of the fields itself a complex piece of data? — if yes, deal with designing classes for that field as a separate task.
- Are there several variants of data that should be known by a common name? — if yes, define an *interface* and have each variant implement this *interface*.
- Make sure you write down a comment explaining what each class of data (or each interface) represents.
- Make sure you make examples of every class you design.