CIS 201 – Computer Science I
Laboratory Assignment 9

Introduction

In this lab you will create a game that reads its configuration from a file.

Reading a file

Consider the process of writing a pinball game. Given a screen and a ball that bounces, different levels or pinball games frequently differ in how their bumpers (and other things) are laid out.

What we really want, then, is to build a pinball *engine*, a program that can simulate any pinball machine described in an appropriate way. You will begin with a simulator that has ball physics but no balls or bumpers. Your job is to extend the program so that it prompts the user for the name of a file (format described below) and then creates the appropriate sprites in the machine before setting the ball(s) loose.

Copy the files `Pinball.java` and `PinballGame.java` into a directory `Lab09`. You can find the files to copy in the directory

```
/home/student/Classes/201/Labs/Lab09
```

Also in that directory is a file named `first.pnd`. We will be using the extension `.pnd` to stand for a "Pinball Description" file, an extension we coined. This file will have a format described later.

Download the Java and pinball files into your lab directory. Compile and run the pinball game. It should compile but it should not do anything because there is no description of a pinball game.

Look at the `PinballGame` class. Find all of the stub functions (they are the ones with little or no code in them). In this lab, you will replace the stubs with working code.

Modify `getFileName` so that it displays the prompt that is passed into it followed by a single space and then reads a string from the user, returning that value.

Reading information from the user's keyboard is somewhat like printing information to the user's screen. Code such as `System.out.print(...)` will display the expression to your screen based on the type of the expression – such as a `String`, an `int`, or whatever. Reading information from your keyboard uses `System.in`, which should not be surprising. But how will the program know the type of the value to be read? Here is where the notion of a `Scanner` comes in.

A `Scanner` is a Java object that takes raw input from the keyboard (or from a file, as we shall see) and interprets it as a string or integer or whatever, depending on what the program asks it to do.

How does one read a string from the keyboard, for example? Here is a snippet of code that would do so:

```
Scanner kbd = new Scanner(System.in); // tokenize raw input from the keyboard
String str = kbd.next(); // interpret the next token from the keyboard
```

For a `Scanner` object, the term "token" means a string of input characters that does not have any whitespace (such as spaces, tabs, or newlines). The `next()` method skips any whitespace until it encounters a non-whitespace character and then collects all of the following non-whitespace characters into a single string that it returns to the caller.

A `Scanner` can also return `int`s using `nextInt()` and `double`s using `nextDouble()`, for example. You should look at the `Scanner` class documentation to see all of the things that a `Scanner` can process.

Use a `Scanner` as described above to implement the `getFileName` method.

Checkpoint 1

Compile and run your program and show us your results.

The `ensureExtension` method

The `ensureExtension` method is passed a file name and returns that file name with a given extension. For example, if `ensureExtension` were passed the parameters `something` and `txt`, it would return the string `something.txt`. The method is called *ensure*Extension because it should only add the extension if the file name does not already end with it. Thus calling the method with parameters `something.txt` and `txt` should also return `something.txt`.

Before you implement the `ensureExtension` method, look at the documentation for the `String` class and study the following methods:

```
equals
endsWith
```

The `ensureExtension` method takes two `String` parameters. The first parameter is the file name (which you get using the `getFileName` method), and the second is the desired extension. If the file name does not end with a dot and the given extension, it needs fixing; otherwise return it

unchanged. To fix a file name, if it ends with a dot, just return file name plus the extension; otherwise return file name plus dot plus extension. Here are possible inputs and their corresponding return values:

fname	ext	return value
abcde	pnd	abcde.pnd
abcde.	pnd	abcde.pnd
abcde.x	pnd	abcde.x.pnd
abcde.pnd	pnd	abcde.pnd

After you get the file name (from `getFileName` above) but before passing it to `connectScannerToFile`, you should ensure that it has the extension for a pinball description file; something like this:

```
fname = ensureExtension(fname, "pnd");
```

Checkpoint 2

Show us your working code.

Reading the configuration file

Now you are prepared to read the whole configuration file by implementing the `readPNDFile`. You will write a loop that reads the next token from the file (a token is, in this case, a word).

Reading from a `Scanner` associated with a file using a loop normally looks something like this, where `in` is assumed to be a `Scanner` object (your mileage may vary):

```
while (in.hasNext()) {
    String obj = in.next(); // get the next string
    ...
}
```

This loop will continue to read strings from the file until there are no more strings to read – in other words, until all the contents of the file have been read and processed.

If the string that the `Scanner` sees is “Ball”, “Oval”, or “Rectangle” you, will call a routine that reads the rest of the information for that type of sprite and adds one to the scene.

That is, in `readPNDFile` you will use the provided code to prompt the user and get a `Scanner` hooked up to a file. Then, while there is another word to read, read it. If the word is “Ball”, call the `handlePinball` method (and so on for the other two types).

Lines in a PND file are of the form:

```
Ball <color> <x> <y> <w> <h> <dX> <dY>
Oval <color> <x> <y> <w> <h>
Rectangle <color> <x> <y> <w> <h>
```

When using a scanner, if you know what the next item from to be read will be, you can simply call the appropriate method such as `nextInt()` (for an `int`), `nextDouble()` (for a `double`), or just `next()` (for a `String`). For example, if you have a line that begins with `Oval`, you will call the `handleOval` method that will then get a `String` (the color) and four `double`s, create an `OvalSprite` object with the appropriate width and height, set its color to the given color value, locate it at the appropriate x and y values, and add it to the game canvas. (Note that you got to the `handleOval` method because you already saw the string `Oval` in the `readPNDFile`.) Similar remarks apply to lines that begin with `Rectangle`. For `Ball` lines, you need to create a `Pinball` object (look at `Pinball.java` to see what the constructor requires for this class!), adjust its color and location, add it to the game canvas, *and* add it to the `pinballs` list.

The word `<color>` can be passed to `getColor` to return a color and all the rest are double values. The variables `x` and `y` are the location of the bumper (or ball); `w` and `h` are width and height. The `dX` and `dY` are the velocity of the ball. You should use the `Scanner` variable `in` to do most of the work of reading all the fields of a given bumper or ball.

0.1 Checkpoint 3

Show us your code at this point.

Write your own PND file

Create a PND file of your own choosing that describes a pinball level with two balls (starting near the left and right edges of the screen heading toward the center of the screen) and two oval sprite eyes and one rectangle sprite “mouth” to make a “smiley face”. Your PND file should end with a `.pnd` extension.

Checkpoint 4

Run your `PinballGame` file and enter your PND file as the filename. Show us your results.

Are you done?

If you have finished with the checkpoints, start working on your Assignment 5.