Copyright 2009, Leutenegger

Processing Notes 7

Chapter 15: Timer/Stopwatch Class

For making a game a timer is often needed. Also, for any sort of simulation one often needs to time how long something takes.   Lets assume we want to create a stopwatch that when we hit “s” the counter starts, “p” and the counter pauses, and “c” it continues. If we hit “s”, then the screen shows a 0, a second later it chances to 1, a second later it chances to 2, and so on. Obviously we want to design a class to make this easy to use. So, lets start with code to use the class and then build the class. Here is some code to use the class:

```
Timer timer ;

void setup()
{
  size(400,400) ;
  timer = new Timer(10,60) ;   // make the display at location (10,60)
  timer.start() ;
}


void draw()
{
  background(200) ;
  timer.DisplayTime() ;
}

void keyReleased()
{
  if ((key == 's') || (key == 'S') )
  {
    timer.restart() ;
    println("reset") ;
  }
  if ((key == 'p') || (key == 'P') )
  {
    timer.pause() ;
    println("pause") ;
  }
  if ((key == 'c') || (key == 'C') )
  {
    timer.continueRunning() ;
```

```
    println("continue") ;
  }
}
```

In this example a new timer object is created in the setup( ) function. In the draw( ) function timer.DisplayTime( ) is called, which presumable displays the current time. Then, in the keyReleased( ) function we see calls to timer.restart( ), timer.pause( ), and timer.continueRunning( ). In our example we have decided restart( ) will reset the timer to zero and start it running, pause( ) will pause the timer so it stops incrementing, and continueRunning( ) will restart a paused timer. All that remains now is to develop a class that supports this interface including start( ), DisplayTime( ), restart( ), pause( ), and continueRunning( ) as shown below. Note, for this to work you need to create the font Arial-Black-48 as specified below:

```
class Timer
{
  long startTime ;   // time in msecs that timer started
  long timeSoFar ;   // use to hold total time of run so far, useful in
                  // conjunction with pause and continueRunning
  boolean running ;
  int x, y ;   // location of timer output

  Timer(int inX, int inY)
  {
    x = inX ;
    y = inY ;
    running = false ;
    timeSoFar = 0 ;
  }


  int currentTime()
  {
    if ( running )
      return (  (int)   ( (millis() - startTime) / 1000.0) ) ;
    else
      return ( (int) (timeSoFar / 1000.0) ) ;
  }

  void start()
  {
    running = true ;
    startTime = millis() ;
  }
```

```
    void restart()
    // reset the timer to zero and restart, identical to start
    {
      start() ;
    }

    void pause()
    {
      if (running)
      {
        timeSoFar =   millis() - startTime  ;
        running = false ;
      }
      // else do nothing, pause already called
    }

    void continueRunning()
    // called after stop to restart the timer running
    // no effect if already running
    {
      if (!running)
      {
        startTime = millis() - timeSoFar ;
        running = true ;
      }
    }

    void DisplayTime()
    {
     int theTime ;
     String output = "";

     theTime = currentTime() ;
     output = output + theTime ;

     // println("output = " + output) ;
     fill(150,0,200) ;
     PFont font ;
     font = loadFont("Arial-Black-48.vlw") ;
     textFont(font) ;
     text(output,x,y) ;
    }
  }
```

In Processing/java the call millis( ) returns the number of milliseconds (thousandths of a second) since the program started running. We use this call to build our timer. The data member “startTime” is used to hold the time at which the timer was started. Then, to get the current time, we just call millis( ) again and subtract the value in startTime. In the method currentTime( ) above, we have the line:

```
return (  (int)   ( (millis() - startTime) / 1000.0) ) ;
```

This gets the number of milliseconds elapsed, divides by 1000 to get seconds, and then converts to an integer to give a nice simple integer counter.

The method restart( ) just calls start again so the counter starts over at zero.

Methods pause( ) and continueRunning( ) are a bit more tricky. The idea is pause( ) will temporarily stop the counter, the number displayed will remain the same, until continueRunning( ) is called, at which time the counter continues on from the paused number. The code for pause( ) is:

```
if (running)
   {
    timeSoFar =   millis() - startTime  ;
    running = false ;
   }
```

We store the time accrued on the timer in the member “timeSoFar” and set running to false. The boolean variable “running” is used that that if the user of the class accidentally calls pause( ) multiple times, the erroneous additional calls have no ill effects.

When continueRunning( ) is called the following code is run:

```
if (!running)
   {
    startTime = millis() - timeSoFar ;
    running = true ;
   }
```

At first this looks a bit odd. What we are doing is setting the startTime to be “timeSoFar” milliseconds before the actual current time, this way it seems the timer has been running for “timeSoFar” milliseconds.

The final method is DisplayTime( ). This uses Processing’s PFont and text() constructs. PFont holds a font. A font is how text is graphically represented on the screen. There are many different fonts. In order to use a font in Processing you must create it first. Don’t worry, this is much easier than it sounds.

Inside the main Processing window, from the top, select “Tools” and then “Create Font...”. This will pop up a window. Find the font you want, select the size, and then click “okay”. The font will then be created for you and put in the Sketch’s data folder. Note, for this example, you would need to create font “Arial-Black-48”. Now consider the following lines of code in DisplayTime( ):

```
PFont font ;
font = loadFont("Arial-Black-48.vlw") ;
textFont(font) ;
text(output,x,y) ;
```

The fist creates a PFont variable. The second loads the font Arial-Black-48. Note, the .vlw is important. The third statement sets the font to be used to be this font. Finally, the command text( ), displays the string specified in the first parameter at the x-coordinate specified by the second parameter, and the y-coordinate specified by the third.

EXERCISE 15A
Modify the Timer class so that it is a countdown timer, in other words it counts dow like: 10, 9, 8,.... to zero. Modify the constructor so you can specify the starting value. Finally, make sure you pause( ) and continue( ) methods work.

Chapter 16: Strings and Input/Output

So far we have been using Strings implicitly without much discussion. Strings variable hold a sequence of zero or more characters. For example:

```
String s1 = “This is simple ” ;
String s2 = “This is silly” ;
println(s1) ;
String s3 = s1 + s2 ;
println(s3) ;
```

would create two String variables, s1 and s2, assign “This is simple” and “This is silly” to s1 and s2 respectively, and then print out the contents of s1. The “+” operator when applied to strings concatenates the two operands together. Thus, variable s3 is assigned “This is simple This is silly” and hence that gets printed out next.

Sometimes it is desireable to convert integer and float values to strings. Processing provides the str( ) function to make this easy:

```
int num1 = 55 ;
float num2 = 44.44 ;
String s1 = str(num1) ;   // convert from int to string
String s2 = str(num2) ;   // convert from float to string
String s3 = "string representation of " + num1 + " and " + num2 ;
String s3b = s1 + s2 ;   // concatenate without a space!
println(s3) ;
println(s3b) ;
```

The above will print out:

```
string representation of 55 and 44.44
5544.44
```

Note, because there is no space in the assignment to variable s3b the string representation of the numbers is blurred together. You can also easily go the other way, i.e. from strings to numbers as in the following example:

```
String s4 = "44" ;
int num4 = int(s4) ;
if (num4 == 44)
  println("Yep, an int equal to 44") ;

String s5 = "44.54" ;
float num5 = float(s5) ;
if (num5 == 44.54)
  println("Yep, a float equal to 44.54") ;
```

Which prints out:

```
Yep, an int equal to 44
Yep, a float equal to 44.54
```

Sometimes we may have a string of characters that we want to split apart into separate words. Consider the following code:

```
s1 = "31 44 56 77 99 121" ;
String[] numsAsStrings = split(s1," " ) ;   // the split token is space
```

```
println("Number elements = " + numsAsStrings.length) ;
for (int i = 0 ; i < numsAsStrings.length ; i++)
  println(numsAsStrings[i]) ;
```

Running this code would produce the following output:

```
Number elements = 6
31
44
56
77
99
121
```

The split function has two parameters, the first the string to be split apart, and the second the character to use as the split delimiter. A common delimeter is a space, another one is the ",". For example, the string:

"This,pig,big,laugh,wise"

Would make 5 elements if the split delimiter was the ",".

All this string manipulation becomes really handy when we are reading data from the user or from a file. First lets consider reading/writing data from/to files. The loadStrings( ) and saveStrings( ) functions allow us to read/write from/to files. Consider the following example:

```
String[] input = loadStrings("inputData.txt") ;   // the split token is space
println("Number of lines of input data = " + input.length) ;
for (int i = 0 ; i < input.length ; i++)
  println(input[i]) ;
```

This will read all of the text in the file inputData.txt and put each line of data into a separate element of the array named "input". If the sketch has a data folder, the loadStrings() command looks in the data folder for a file named "inputData.txt". If found, it loads the data from that file. If not found, or there is no data folder, then Processing looks for a file named "inputData.txt" in the same directory as the .pde file.

The above code will echo the data from the file to the Processing output window. For example, if the file contains:

```
If I were a rich man
Ya ha deedle deedle, bubba bubba
```

Copyright 2009, Leutenegger

Then those same two lines would be printed in the output window. The lines of text in the file “inputData.txt” are not broken apart into individual words. To do that we would need to use the split command:

```
String[] input ;
String[] temp ;
input = loadStrings("inputData.txt") ;
for (int i = 0 ; i < input.length ; i++)
{
  temp = split(input[i]," ") ;
  for (int j = 0 ; j < temp.length ; j++)
    println(temp[j]) ;
}
```

For the same data file it would print out:

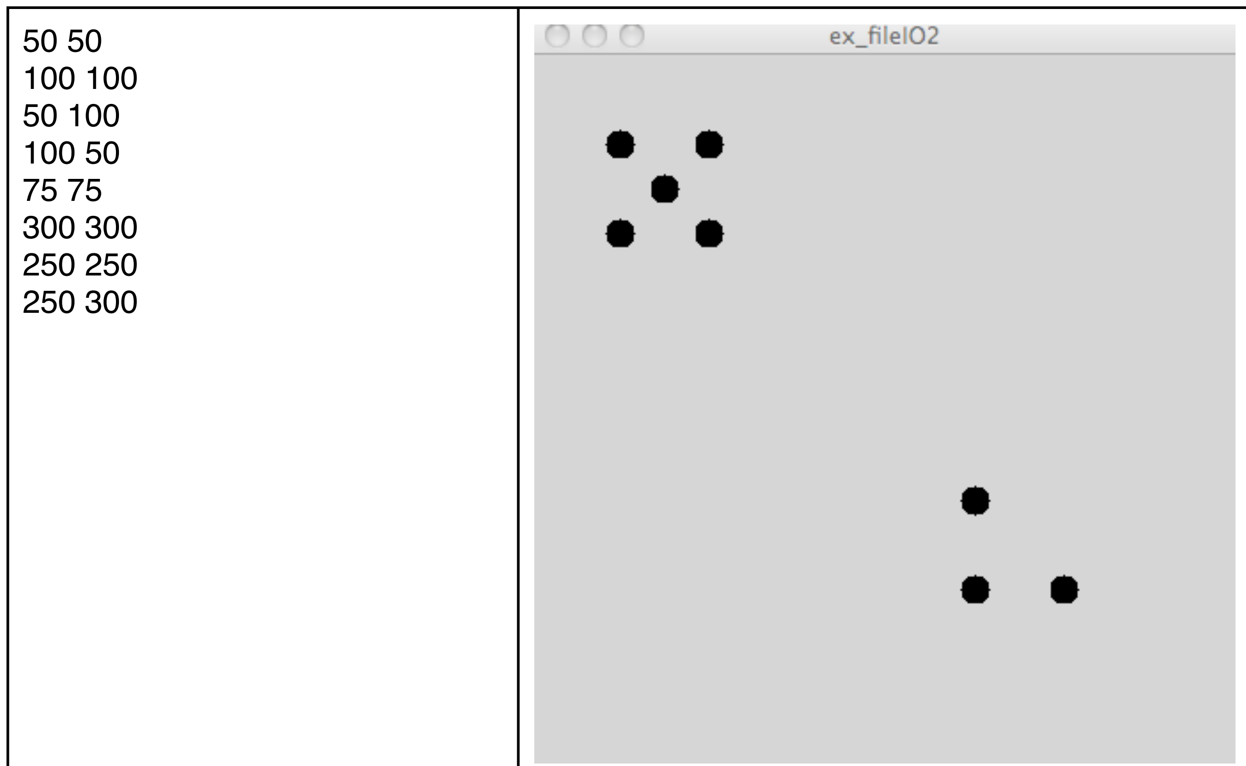
```
If
I
were
a
rich
man
Ya
ha
deedle
deedle,
bubba
bubba
```

Now that we can read in data, we can use that data in different ways. One simple way is to create a scatter-plot of data, i.e. a dot at every (x,y) pair specified in the input data file. For example, consider the following code:

```
void setup()
{
  size(400,400) ;

  String[] input;
  String[] temp ;
  int x, y ;

  fill(0) ;
  input = loadStrings("inputData.txt") ;
  for (int i = 0 ; i < input.length ; i++)
  {
```

```
    temp = split(input[i]," ") ;
    x = int(temp[0]) ;
    y = int(temp[1]) ;
    ellipse(x,y,15,15) ;
  }

}
```

If the data file contains the data is in the below left, the output would be the image in the below right:



Not only can we load data from a file, we can also save data to a file. We do this using the saveStrings(filename, StringArray) command, where the first argument is the name of the file you want to write, and StringArray is the name of a variable that holds an array of strings. Consider the following code:

```
String[] outStrings ;
outStrings = new String[5] ;
outStrings[0] = "My momma talkn to me" ;
outStrings[1] = "tried to tell me how to live." ;
outStrings[2] = "Da da da, da da da, DA DA!" ;
outStrings[3] = "But I don't listen to her" ;
```

```
 outStrings[4] = "cuz my head is like a SIEVE" ;
 saveStrings("outData.txt",outStrings) ;
```

The effect of this code is to create a file name “outData.txt” and put the file lines of text in the array outStrings into the file. Note, if the file already exists the existing file is over-written with this data.

In addition to reading/writing data from/to files, we can get data from the user. Consider the following code:

```
String inputFromScreen   ;
String[] history  ;
PFont font1 ;

void setup()
{
  size(400,400) ;
  fill(0) ;


  font1 = loadFont("Serif-24.vlw") ;
  history = new String[0] ;
  textFont(font1) ;
  inputFromScreen = "" ;


}

void keyReleased()
{
  char nextChar = key ;
  if (nextChar == '\n')
  {
    append(history,inputFromScreen) ;
    inputFromScreen = "" ;
  }
  else
    inputFromScreen = inputFromScreen + nextChar ;
}

void draw()
{
  background(100) ;
  text(inputFromScreen,20,20) ;
}
```

Every time the user releases a key, if the key was not the return key, the key appended to the string variable “inputFromString”. The draw function is constantly displaying the contents of this variable, hence, what the user types is echoed to the screen. When the user types the return key, the string so far is appended to an array of strings that keeps track of the history of what has been typed, and then the string is reset to the empty string.

Exercise 16A
Write a program that interacts with the user to read in keyboard input and echo it to the screen. When the user types the exclamation key, i.e. key == ‘!’, the history is written out to a file named history1 and the history string is reset to the empty string, i.e. it clears the output. Then next time the user hits ‘!’ the history is written out to a file named history2 and everything is reset, and so on. For example, if I type in: Little Bo Peep Had lost her sheep and ! doesn’t know where to find them.! There will be two files created, history1 and history2. File history1 will have three lines of text with the last line have the word “and “. File history 2 will have two lines of text with the first line being “doesn’t know” and the second being “where to find them.”

Exercise 16B
Write a program that collects the location of mouse-clicks, draws a line between each point of mouse click, and stores the (x,y) coordinates in an output data file one line per x,y pair.

Copyright 2009, Leutenegger

EXERCISE 16C

Write a program that reads a data file that represents rectangles, circles, and lines and draws the picture for that file. Here is an example data file:

```
r 10 50 20 20
c 100 100 20 20
c 150 150 30 30
l 10 50 100 100
l 100 100 150 150
```

For this datafile one rectangle will be drawn: rect(10,50,20,20) ; two circles will be drawn: ellipse(100,100,20,20), ellipse(150,150,30,30) ; and two lines will be drawn: line(10,50,100,100) , line(100,100,150,150)