04 Testing and Debugging

Types of Error

Syntactic – basic error that prevents your program from compiling, e.g. missing semicolon; typically easy to fix.

Semantic/Logical – error that occurs after compilation and while your program is running:

- may result in a run-time error, e.g. null pointer reference exception
- alternatively your program may run without crashing but not produce the correct output.

Typically harder to fix.

Code and Test Approach Minimizing Errors

- Develop your program from the simplest class to the most complex class.
    - **Simple class**=one that does not depend on any other class
    - **Complex class**=one that depends on other classes
- Develop each class **one property/method at a time** – write code & test your program each time you add a new property/method
- Test the entire class thoroughly when finished

Objective is to spend about 50% coding and 50% testing – this will save 200% of your time fixing bugs later.

Example: Slider class

A slider is a GUI control that lets the user slide a marker in order to select a value in a range between min and max.

What are the properties?
minValue, maxValue, size/position on screen, colour, currentValue

What are methods?

- must be able to draw itself
- must respond to mouse click interactions (click, drag, etc)

How to develop this class?

- first focus on the appearance – code properties and a draw() method so that the slider can draw itself
- next focus on the user interaction – add properties and methods to handle the user interaction

Method Design

A method or class can be used in two main ways:

Normal behavior – occurs when the method/class is used correctly, e.g. sqrt(9)
Erroneous behaviour – occurs when the method/class is used incorrectly, e.g. sqrt(-1)

We can also add a third type of use:

Boundary – occurs when the method/class is used at the boundary between normal and erroneous behaviour, e.g. sqrt(0), sqrt(-0.000000001), sqrt(0.000001)

What should your method do in each situation?

Testing

After deciding what your method should do, write the method, then *test* it by repeatedly running your program with different inputs.

Example: the setValue() method of the Slider class:

Normal case 1:

```
Slider slider = new Slider();
slider.minValue = 0;
slider.maxValue = 100;
slider.setValue(50);
```

Normal case 2:

```
Slider slider = new Slider();
slider.minValue = -10;
slider.maxValue = 20;
slider.setValue(1);
```

Normal case 3:

```
Slider slider = new Slider();
slider.minValue = 0;
slider.maxValue = 255;
slider.setValue(254);
```

Erroneous case 1:

```
Slider slider = new Slider();
slider.minValue = 0;
slider.maxValue = 255;
slider.setValue(288);
```

Erroneous case 2:

```
Slider slider = new Slider();
slider.minValue = -100;
slider.maxValue = 100;
slider.setValue(-36535353);
```

Boundary case 1:

```
Slider slider = new Slider();
slider.minValue = -100;
slider.maxValue = 100;
slider.setValue(-100);
```

Boundary case 2:

```
Slider slider = new Slider();
slider.minValue = 0;
slider.maxValue = 255;
slider.setValue(-255);
```

Does the method work in all situations as expected?

Additional boundary cases -- values just above our below the boundary between correct/erroneous behaviour also qualify as boundary cases and should be tested, e.g. if maxValue=1000 then good boundary cases are setValue(999), setValue(1000) and setValue(1001).

FYI here is the implementation of setValue():

```
public void setValue(float val) {
    if (val<minValue) currentValue = minValue;
    else if (val>maxValue) currentValue = maxValue;
    else currentValue=val;
  }
```

What does setValue() do for each of the cases?

Testing Processing's Builtin Functions

rect() function

- normal behaviour occurs when width>=0 and height>=0;
- erroneous case occurs when width or height are negative;
- boundary case is where width or height is zero.

Processing draws rectangles when the width or height are negative, but this behaviour is not documented in the online reference:
http://processing.org/reference/rect_.html
Not clear what happens!

PVector's add() method

- normal behaviour occurs when add() is passed a created PVector
- erroneous behaviour (null pointer exception) occurs when add() is passed null

Debugging

Having found some incorrect behavior, how to locate the error(s)?

Strategy 1 – if there are more than one error, target the *first* error. Why? The first error may be causing the rest of the errors.

Strategy 2 – add a debug flag to your program.

```
final boolean DEBUG = true;

void setup() {
        /* do stuff */
}
```

when you want to check something by printing it to the console, use if statements, e.g.:

```
public void setValue(float val) {
    if (val<minValue) currentValue = minValue;
    else if (val>maxValue) currentValue = maxValue;
    else currentValue=val;
    if (DEBUG) {
       println("setValue() called with val="+val+
               ", resulting in currentValue set"+
               " to "+currentValue);
    }
  }
```

which results in, for example:

```
setValue() called with val=50, resulting in currentValue
being set to 50

setValue() called with val=2000, resulting in
currentValue being set to 100

setValue() called with val=-10000000, resulting in
currentValue being set to 0
```

Strategy 3 – add a toString() method to your class and use it.

Example:

```
class Slider{
  /* stuff */
  public String toString() {
      String result = "This is a Slider";
      result+="\nminValue = "+minValue;
      result+="\nmaxValue = "+maxValue;
      result+="\ncurrentValue = "+currentValue;
      result+="\nx = "+x;
      result+="\ny = "+y;
      result+="\nsize = "+size;
      result+="\ncolour = "+colour;
      return result;
   }
}
```

In the setup():

```
redSlider=createSlider(150,200,#FF0000);
if (DEBUG) println( redSlider );
```

resulting in this console output:

```
This is a Slider
minValue = 0.0
maxValue = 255.0
currentValue = 0.0
x = 150.0
y = 200.0
size = 200.0
colour = -65536
```

Strategy 4 – use a HUD

Processing calls draw() thirty times a second which may produce too much console output using the other methods. An alternative is to draw your debug output instead of printing it:

```
// A HUD method to assist debugging
void drawHUD() {
  textAlign(CENTER, CENTER);
  fill(255);
  text("red="+redSlider.currentValue, 100, 500);
  text("blue="+blueSlider.currentValue, 300, 500);
  text("green="+greenSlider.currentValue, 500, 500);
}

void draw(){
  /* draw everything */
  // Draw the HUD
  drawHUD();
}
```