# 03 User Interface & Object Composition

## Keyboard/Mouse Input

**Mouse variables** – Processing maintains the several useful global variables for tracking mouse position and the state of the mouse’s buttons: mouseX, mouseY, mousePressed, pmouseX, pmouseY, etc. This allows several useful constructs, e.g.

// Draw a line if the left mouse button is pressed

if (mousePressed && (mouseButton == LEFT))

line(pmouseX, pmouseY, mouseX, mouseY);

**Mouse events** – Processing defines special functions can be called if the mouse is clicked, moved or dragged, e.g.:

void mousePressed() {

println("mouse pressed");

}

**Keyboard variables/events** – similarly, global variables and functions such as keyDown and keyPressed() can allow keyboard input to be managed:

void keyPressed() {

println("a key was pressed, code "+keyCode);

}

## Designing UI Controls

**Control design** – controls such as buttons, lists and textboxes are perfect examples of objects. They have fields – e.g. state of the button (pressed/not pressed), contents of the textbox – and methods – e.g. click, type a character.

|  |  |  |
| --- | --- | --- |
|  | Button | Textbox |
| Sample Fields | text  position  size  textColor | text  position  size  focused  textColor  selected |
| Sample Methods | draw  clickStart  clickEnd | draw  keyTyped |

**Object interaction** – when designing classes its important to know how they will interact with the rest of the program. For example, buttons need to know when the user starts and ends a click so that the button can adjust its state accordingly. Conversely the rest of program needs to know when the button was successfully clicked so that things can happen, e.g. clicking a “Start” button begins a game.

Here is a sketch of a Button class:

class Button {

// Properties

public final int STATE\_NORMAL=0, STATE\_CLICKED=1;

private int state;

// Constructor

public Button(float x, float y, float w, float h) {

/\* … code… \*/

}

// Methods

public void draw(){

/\* … code… \*/

}

public void clickStart(float x, float y) {

/\* … code… \*/

}

public void clickEnd(float x, float y) {

/\* … code… \*/

}

}

**State field** – can be in one of two states, STATE\_NORMAL or STATE\_CLICKED (which occurs if the mouse button is pressed while mouse is over the button). The button gets informed about click events and updates itself whenever clickStart() and clickEnd() are called. The state properties also determines what the draw() method draws.

**Helper methods** – these are *private* methods. They are not meant to be called outside of the class. A useful helper method for this class is to determine if an arbitrary point (*x,y*) lies inside the rectangle of the button:

class Button {

/\* … code… \*/

private boolean pointInRegion(float x, float y) {

return x>=position.x-size.x/2

&& x<=position.x+size.x/2

&& y>=position.y-size.y/2

&& y<=position.y+size.y/2;

}

/\* … code… \*/

}

We can then use a helper method to determine when to change the state of the button to STATE\_CLICKED:

class Button {

/\* … code… \*/

public void clickStart(float x, float y) {

if (pointInRegion(x, y))

state=STATE\_CLICKED;

}

/\* … code… \*/

}

The clickEnd() method then communicates back to the main program when a click is completed, and resets the state to normal:

class Button {

/\* … code… \*/

public void clickEnd(float x, float y) {

if (state==STATE\_CLICKED)

exampleButtonClicked(); // Main program function

state=STATE\_NORMAL;

}

/\* … code… \*/

}

The main program must be then setup to (i) create the button object, (ii) send click messages at the right time, (iii) draw the button, and (iv) receive notifications when the button was clicked:

Button exampleButton;

void setup(){

exampleButton = new Button(…);

}

void draw(){

exampleButton.draw();

}

void mousePressed(){

exampleButton.clickStart(mouseX, mouseY);

}

void mouseReleased(){

exampleButton.clickEnd(mouseX, mouseY);

}

void exampleButtonClicked() {

println("example button clicked");

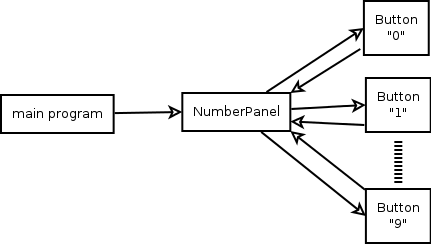
}

## Object Composition

**Object Composition** – a common design technique in which many simple objects are contained inside or compose one larger object. For example, my body composed of many smaller objects: arms, legs, head, torso, organs etc. But for most intents and purposes my body is a single object.

For example, a PIN number panel is composed of 10 buttons (0-9 inclusive). Each button is an individual object. Each button should respond to clicks, but in a slightly different way.

**Object interaction** – is slightly more complex now, as the constituent objects communicate back to their containing object, not the main program. This makes the main program appear to be very simple.



**this keyword** – is a special keyword that can be used inside an object and always evaluates to the current object. For example to set the properties and call methods of whatever object is executing now:

this.text=”something”;

this.draw();

**Number panel design** – main public field is the PIN so far, but hidden inside the object are the buttons. Calls to draw and click events pass straight through to the buttons. The buttons talk back to the panel when they are pressed.

**Setting up the two-way communication** – this can be achieved by using object reference fields inside the classes, e.g.:

class NumberPanel {

private ArrayList<Button> buttons;

}

class Button {

public NumberPanel panel;

}

Next, the objects themselves must be created and linked together. In this example, the linking is done inside the constructor of the number panel:

class NumberPanel {

private ArrayList<Button> buttons;

public NumberPanel() {

buttons = new ArrayList<Button>();

float size = width/3;

for (int index=0; index<10; index++) {

// Determine x,y position of the button

float x=(index%3)\*size+size/2;

float y=(index/3)\*size+size/2;

// Create the button

Button button = new Button(x, y, size, size);

button.text=""+index;

**button.panel=this; // Button to panel**

// Add the button to the collection

**buttons.add(button); // Panel to buttons**

}

}

}

**Panel to button communication** – access the buttons from the array list, e.g.

for (Button button: buttons)

button.draw();

**Button to panel communication** – call a method on the panel, passing this as a parameter so that the panel knows which button was clicked, e.g.

if (state==STATE\_CLICKED)

if (panel!=null)

panel.buttonClicked(this);

This requires a special method to be set up on the panel:

public void buttonClicked(Button whichOne) {

text+=whichOne.text;

if (text.length()>4)

text=text.substring(text.length()-4);

}