# 03 User Interface

## 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.

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

**Interaction design** – when designing classes its important to know how they will interact with the rest of the program. For example, buttons need to know when they should draw themselves and then the user starts and ends a click. Conversely the program needs to know when the button was clicked.

Here is a sketch of a Button class:

class Button {

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

private int state;

/\* … properties … \*/

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

/\* … code… \*/

}

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 (mouse button is pressed while mouse is over the button). This needs to be updated by clickStart() and clickEnd(). The state 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) sent 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");

}

## Collections of UI Objects