IAT 265

GUI in Processing

Today’s topics

- More on PFont
- Processing & Java Mode
- GUIs and Widgets
- Event-Driven Programming
- ControP5 as GUI Toolkit
- Create GUIs using ControlP5

More on PFont

- Review:
  - `Use Create Font tool to create a font`
  - `PFont loadFont()` – loads the font
- Alternative: create a font by calling
  - PFont createFont (String name, float size, boolean smooth)
  - Use PFont.list() to find the names for available fonts

Case study: add text to bug game

```
//Create a Texts class to handle text display
class Texts {
  PFont headFont;
  PFont overFont;

  Texts() {
    headFont = createFont("Arial",31,true);
    overFont = createFont("Arial",48,true);
  }

  //heads up display
  void headsUpDisplay(int score, int lives) {
    textAlign(CENTER);
    fill(255);
    textFont(headFont);
    text("Score: " + score, 100, 100);
    text("Lives: " + lives, width/2, 100);
  }
```

```
//game over screen
void gameOver(int score) {
    background(0);
    textFont(overFont,60);
    fill(255);
    textAlign(CENTER);
    text("GAME OVER", width/2, height/2);
    textFont(overFont, 20);
    text("Final Score: " + score, width/2,
              height/2 + 30);
  }
```

Case study: add text to bug game

```
//In the main sketch where setup() & draw() stay

//Text display variables
boolean gameOver = false; //flag for game over
Texts txt = new Texts();
int score = 0;          //score starts at 0
int lives = 3;          //player starts 3 lives

void draw() {
  background(255);
  fill(0, 200, 0);
  rect (gardenX, gardenY, gardenW, gardenH);

  if (gameOver)  txt.gameOver(score);
  else {
     //call playGame() to start the game
     playGame();
     //displays heads up display
     txt.headsUpDisplay(score, lives);
  }
}
```

We started with Processing in...

```
// any code here, no methods
line(0,0,2
```

```
// methods!
// global
int a;
// call-ba
void set
}
void dra
}
```

```
// …with classes
// (all of th
class Emo
//fields
//construct
//methods
}
```

```
// …and subclasses!
// (ALL of the above, and…)
class Happy extends Emotion {
//new fields
//constructor
//methods
}
```

Processing is actually a Java Class

```
// Java-Mode!!!
class MyClass extends PApplet {
// void setup() and void draw() inherited from PApplet
//methods
//classes and subclasses
}
```

Java Mode

- Allows you to program in pure Java
  - Can import classes that aren't normally imported into a Processing app
  - Importing means making a classes available to your program – the Java API docs tell you what classes are available
- In Java mode, create a class that extends PApplet
  - Normally, all Processing applets extend PApplet behind the scenes
- setup(), draw(), etc. are methods overriding methods inherited from PApplet – Polymorphism!!

A Java-mode program

```
class MyProgram extends PApplet {
    void setup() { ... }
    void draw() { ... }

    void myTopLevelMethod() { ... }

    class Text { // Text is just an example
        int xPos, yPos;
        String word;
        ...
    }
}
```

Notice that for Processing in Java-mode any classes you define are *inside* the top class. In Java, this is called nested classes, which you'll use only occasionally for some special situations, such as for event handlers

Why use Java-mode?

- Java-mode gives you access to the entire Java SDK
  - E.g. we need access to some SDK classes for HTML parsing that Processing doesn't make visible by default

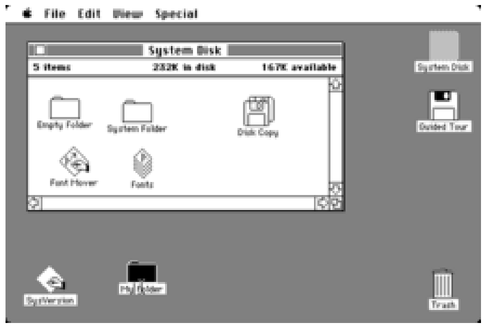
- Java-mode helps you understand how Processing is built on-top of Java
  - All those "magic" functions and variables are just methods and fields of PApplet that your program inherits

Libraries!

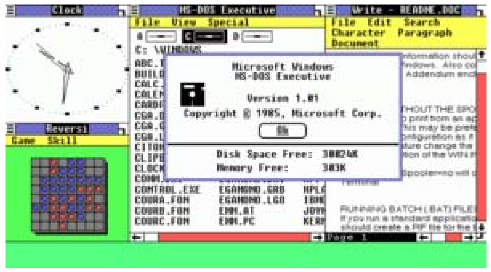
- Libraries are other classes (in .java or .jar files)
  - Use import nameOflibrary.nameOfPackage.nameOfClass; (e.g. import java.awt.*; import javax.swing.JFrame;)
- With Java-mode, you should also put your programs in multiple files
  - One file for each class (this is a MUST in real Java programming, and file name must be the same as class name)
  - In Processing, you do this by using the tab button at the upper right

GUI – Graphical User Interface

- A type of user interface that allows users to interact with programs in more ways than typing
  - A GUI offers graphical icons and visual indicators **(widgets)**
  - The actions are usually performed through direct manipulation of the **widgets**



Original 1984 Macintosh desktop



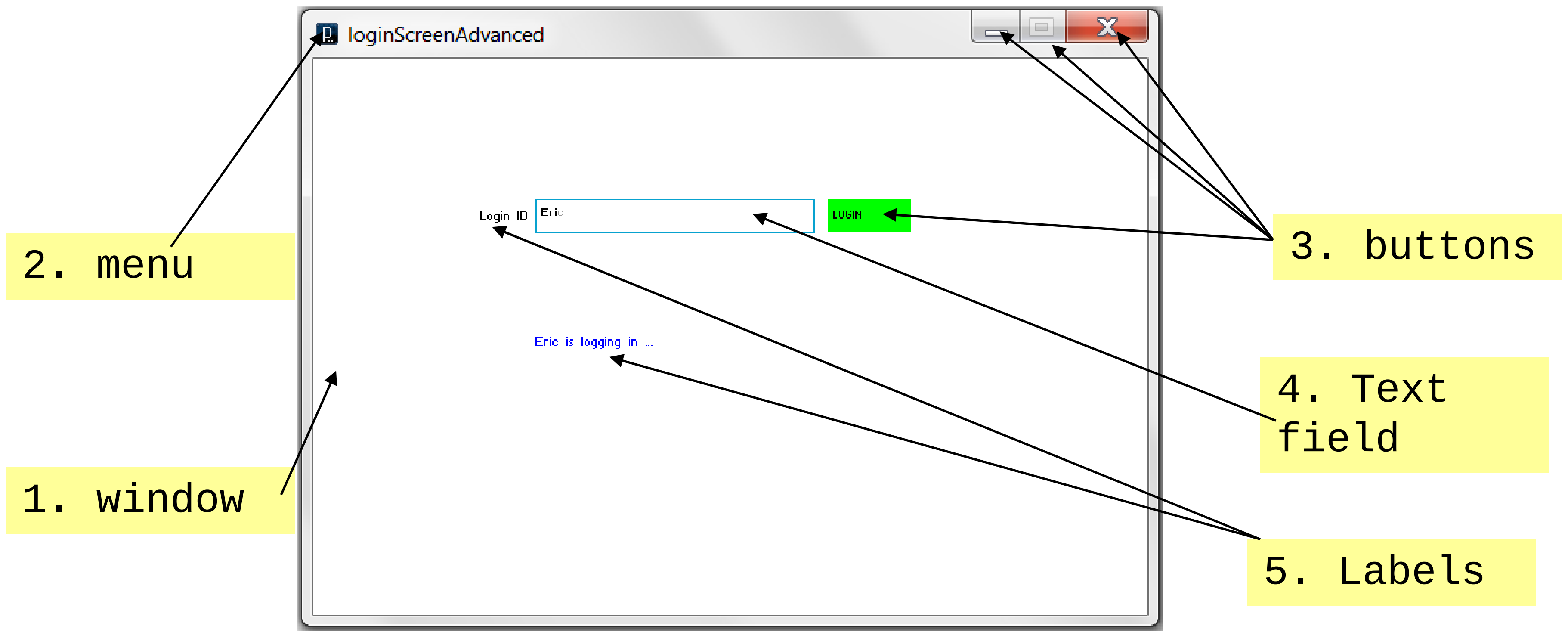
Windows 1.0, released in 1985

Typical GUI Widgets

- ***Window*** – a smaller screen within the screen
- ***Menu*** – list of alternatives offered to user
- ***Button*** – icon that can be pressed
- ***Label*** – display of descriptive caption
- ***Text field/area*** – text box for text I/O
- ***Slider/Knob/bang/toggle*** – value manipulators
- ***ScrollList/Radio*** – a list of selectable items
- ...

Examples of GUI widgets

- **What widgets do you find here?**



Challenge of GUI Programming

- The Challenge lies mainly in the need to dynamically change user interface (based on events) at runtime

- It can be tackled by a program design pattern named: **event-driven programming**

Event-Driven Programming

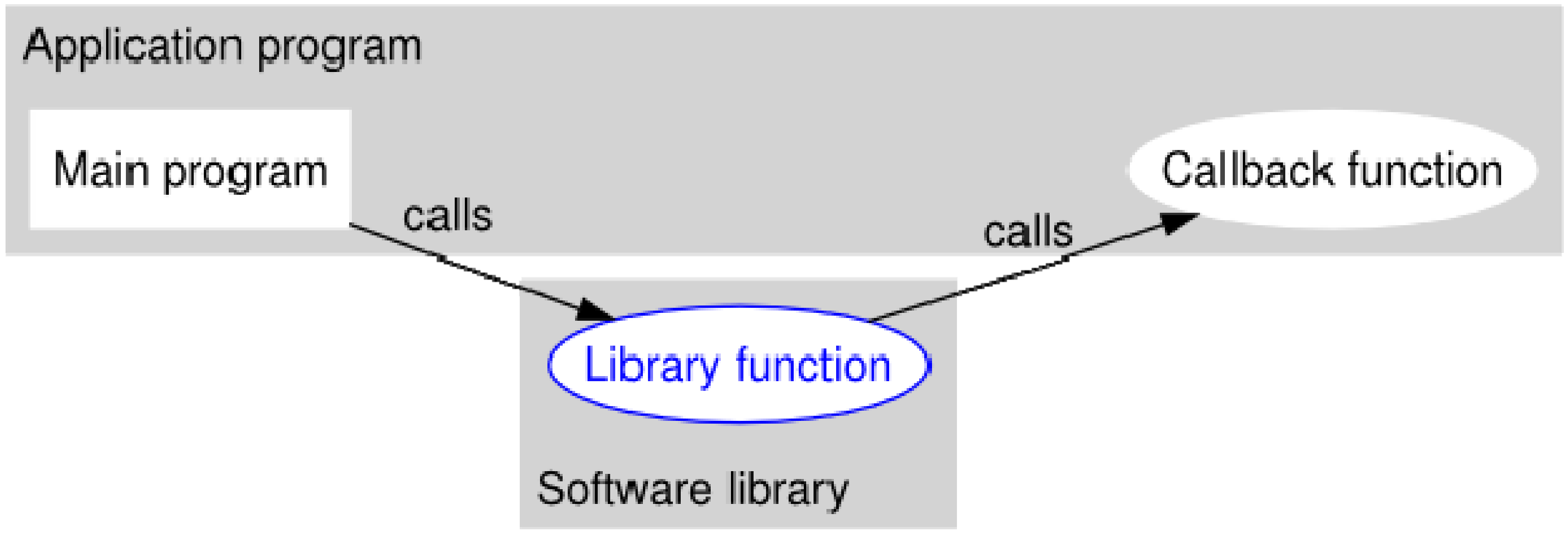
- The flow of the program is determined by ***events*** — e.g. user actions (mouse clicks, key presses), sensor inputs, or timer ticks

- Widely used in GUIs and adopted by most widget toolkits (aka libraries) as the model for interaction
  - E.g. Swing(Java), MFC (C++), Tk (Python), **Control5P** (Processing)

Event-Driven Programming: Architecture

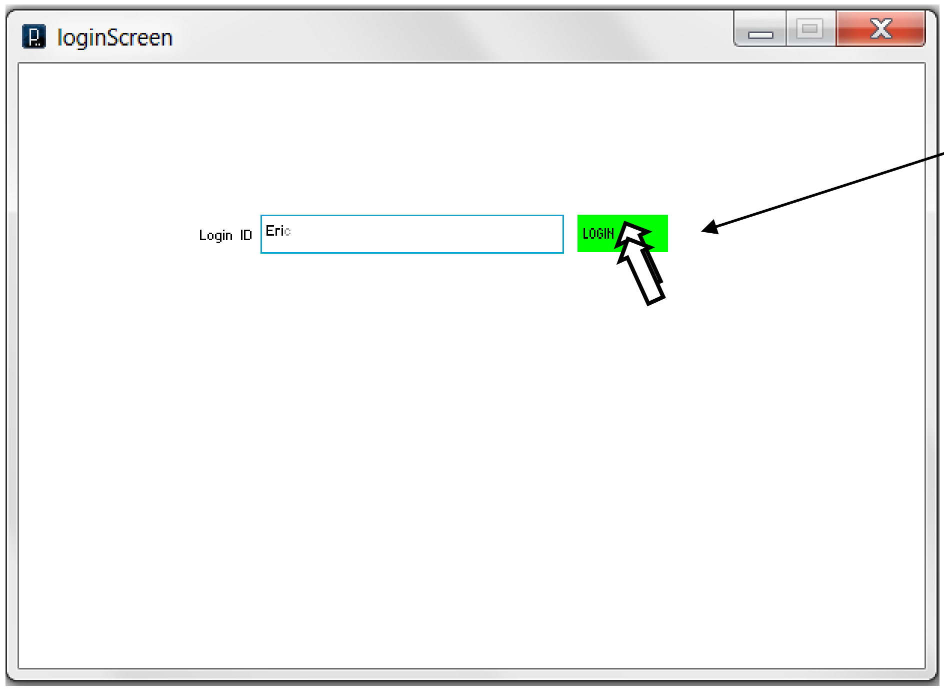
- Program waits for *events* to occur and then responds, which is broken down into three sections:
  - ***Event firing*** – objects/interactions generate events
  - ***Event detection*** – listeners check for events
    - Normally taken care of by programming framework
  - ***Event handling*** – functions respond to events
    - Programmers need to define these functions (aka event-handlers), typically they are ***callbacks***

Review:
callback mechanism



Example of the Architecture

3. Event handler responds



1. Button click fires an event

2. Listener catches the event behind the screen

Eric is logging in ...

GUI Toolkits for Processing

- GUI in Processing is done using either the **controlP5** or the **G4P**
  - controlP5 is better documented than G4P

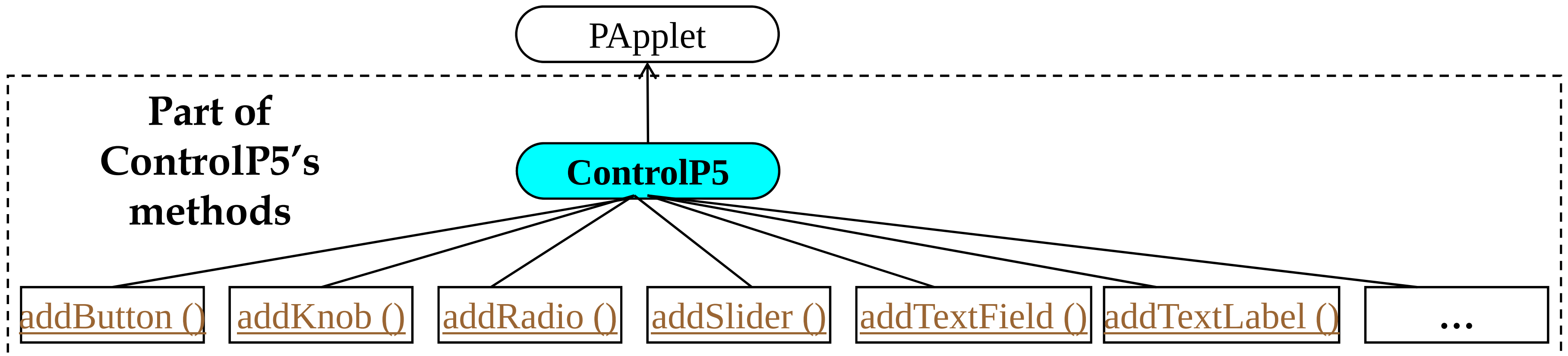
- They can be downloaded respectively at:
  - http://www.sojamo.de/libraries/controlP5
  - http://www.lagers.org.uk/g4p/index.html

ControlP5

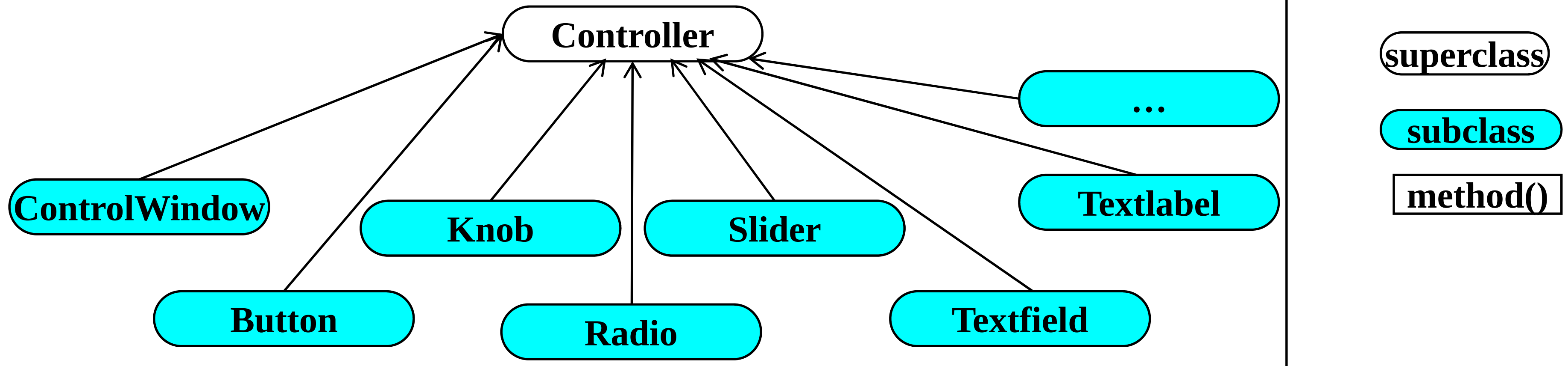
- ControlP5 is a GUI and controller library for processing that can be used in application and applet mode
  - Controllers here are actually GUI widgets such as Sliders, Buttons, Toggles, Knobs, Textfields, Radios, Checkboxes among others

ControlP5 Reference

- **Here is the link:**
  - http://www.sojamo.de/libraries/archive/controlP5-0-3-14/reference/
- It contains the doc for ControlP5's classes, their methods, and functions as well as examples
- Next slide gives you a brief overview of its structure (Note: it provides only a partial picture. Refer to the doc for all)



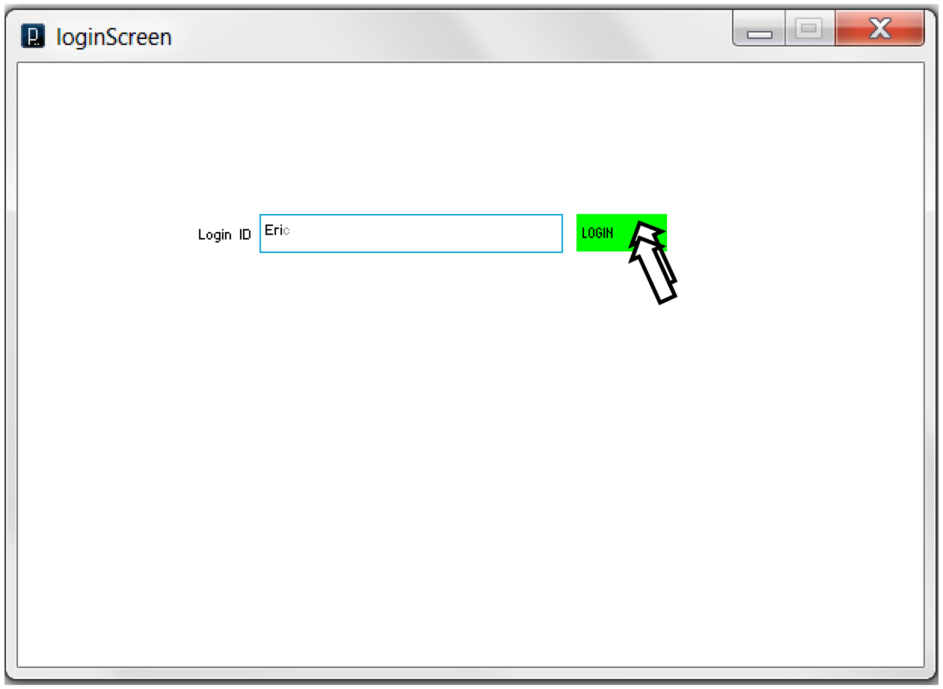
- You can call these methods of ControlP5 to add controllers to your sketch



• Controller is the superclass of all available controllers …

• You can call each controller's methods (including inherited) to set or get its properties such as label, value, color etc., or do other manipulations

How to implement such a GUI using *ControlP5* ?



```
Eric is logging in ...
```

Implementation

1. Set up window & Create ***ControlP5*** object

```
import controlP5.*;          //import the whole library
ControlP5 controlP5;         //declare variable of ControlP5

void setup() {
  size(600,400);
  frameRate(25);
  //instantiation with constructor: ControlP5(PApplet theParent)
  //Here this is used to refer to the PApplet container
  controlP5 = new ControlP5(this);
}

void draw() {
  //drawing background in white
  background(255);
}
```

Implementation

2. Add the **Login ID** label to the window

```
import controlP5.*;         //import the whole library
ControlP5 controlP5;        //declare variable of ControlP5
Textlabel logLabel;         //declare variable of TextLabel

void setup() {
  …    //whatever is done so far
  controlP5 = new ControlP5(this); //instantiation
 //parameters: name, text, x, y
 logLabel = controlP5.addTextlabel("loginID","Login ID",
      110, 110);

  //set its text color to black
  logLabel.setColorValue(color(0));
}
```

Implementation

3. Add the **login text field** as input box to the window

```
import controlP5.*;          //import the whole library
ControlP5 controlP5;         //declare variable of ControlP5
Textlabel logLabel;           //declare variable of Textlabel
Textfield logTextfield;       //declare variable of Textfield
void setup() {
  …    //whatever is done so far
 //parameters: name, x, y, width, height
 logTextfield =
      controlP5.addTextfield("logField",160,100,200,25);
  //set text color to black & field color to white
  logTextfield.setColorValue(color(0));
  logTextfield.setColorBackground(color(255));
  //Set up so that the field will always maintain
  //the focus at runtime
  logTextfield.setFocus(true);
  logTextfield.keepFocus(true);
}
```

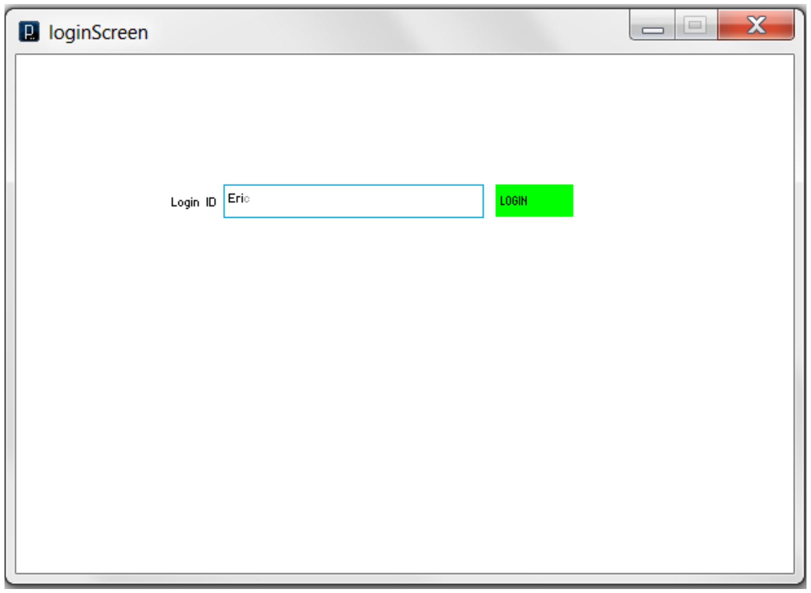
Implementation

4. Add the **login button** to the window

```
import controlP5.*;           //import the whole library
ControlP5 controlP5;          //declare variable of ControlP5
Textlabel logLabel;           //declare variable of Textlabel
Textfield logTextfield;       //declare variable of Textfield
Button logButton;             //declare variable of Button

void setup() {
  …    //whatever is done so far
  //parameters: name, value (float), x, y, width, height
  logButton = controlP5.addButton("Login",0,370,100,60,25);
  //set button’s foreground color to black & background
  //color to green
  logButton.setColorLabel(color(0));
  logButton.setColorBackground(color(0, 255, 0));
}
```

Problem …

- If you run the program, this is what you'll get:



- However, when you input some text and then press the button, nothing happens. Why?

Answer

- That's because we haven't created any code to handle the ***buttonPressed*** event yet
  - When you pressed the button, a *buttonPressed* event was fired, but it was ignored by our program

Implementation

5. Add the **event handler**, which is a callback function ***controlEvent(theEvent)***

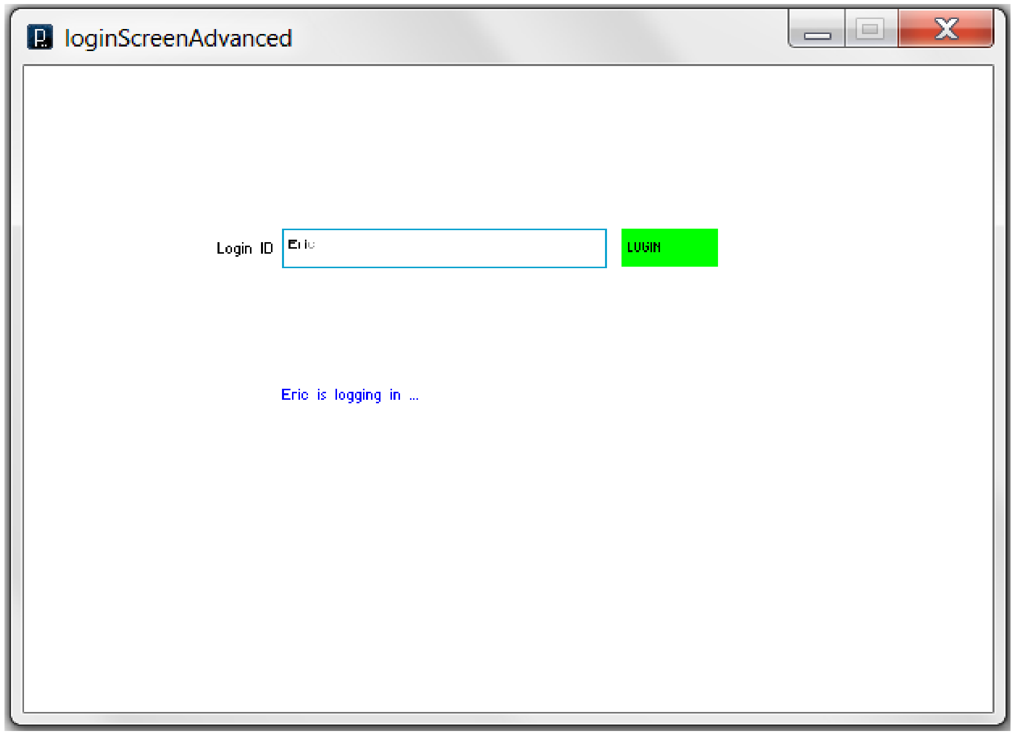
```
 …    //whatever is done so far

 /* events triggered by controllers are automatically
  * forwarded to the controlEvent method. By checking the
  * name of a controller you can distinguish which controller
  * has been changed.
*/
void controlEvent(ControlEvent theEvent) {
  if(theEvent.controller().name() =="Login") {
    println(logTextfield.getText() + " is logging in ...");

  }
}
```

check if the controller is the Login button, and then output a message to the console if so

If you had more controllers on the screen, you could add more *if-statements* to check them out

Now think about …

- How to display the response on the GUI, rather than to the console, as shown below?



Login ID Eric

Eric is logging in ...

Add a Label for Response

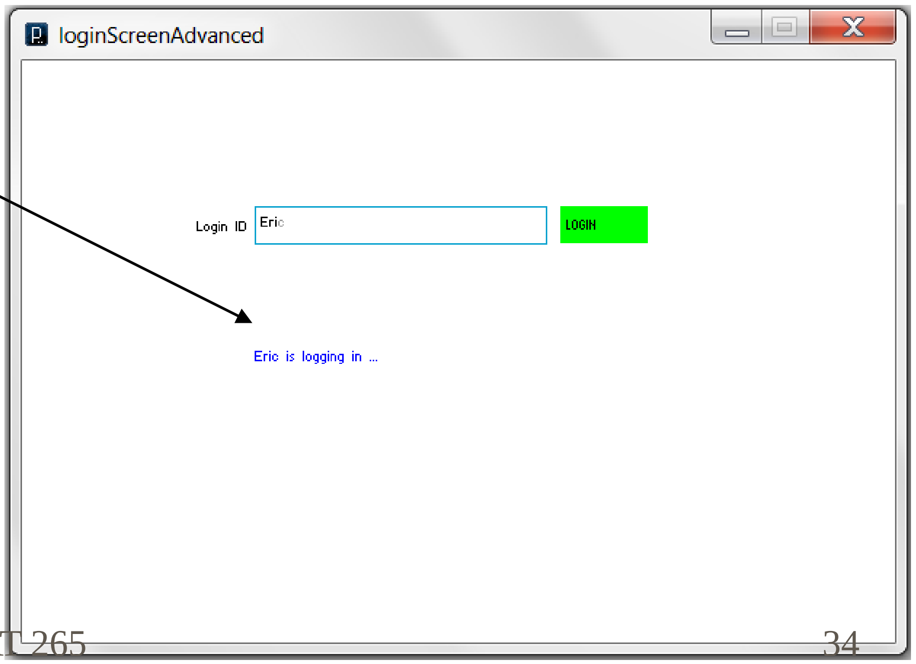
```
import controlP5.*;            //import the whole library
ControlP5 controlP5;           //declare variable of ControlP5
Textlabel logLabel, resLabel; //declare variables of TextLabel


void setup() {
  …    //whatever was done so far

  //parameters: name, text, x, y
  resLabel = controlP5.addTextlabel("resp","", 160, 200);

  //set its text color to blue
  resLabel.setColorValue(color(0, 0, 255));
}
```

Set Response to the Label

```
…     //whatever was done so far

void controlEvent(ControlEvent theEvent) {
  if(theEvent.controller().name() =="Login") {
    resLabel.setValue(logTextfield.getText() + " is logging
        in ...");
  }
}
```

This is what you get when you input an id & then press the button.

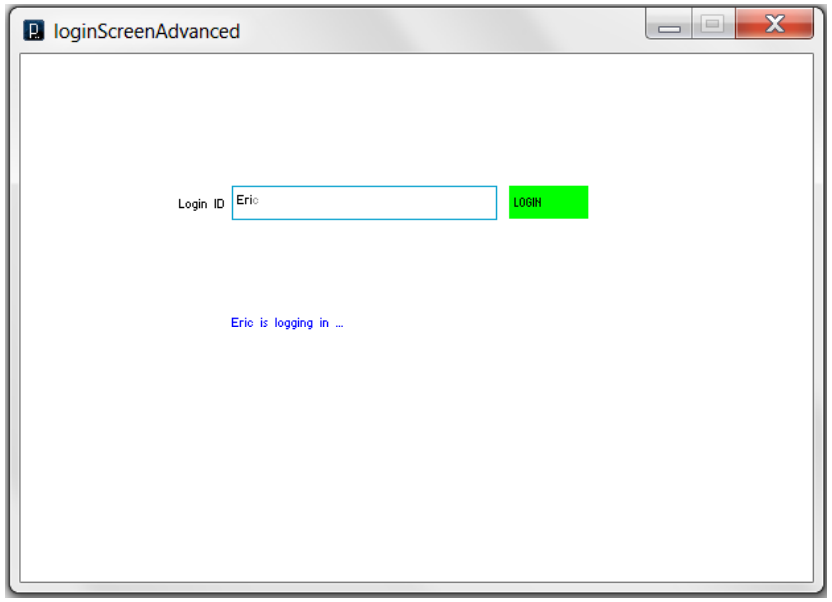
Are you happy with this GUI?



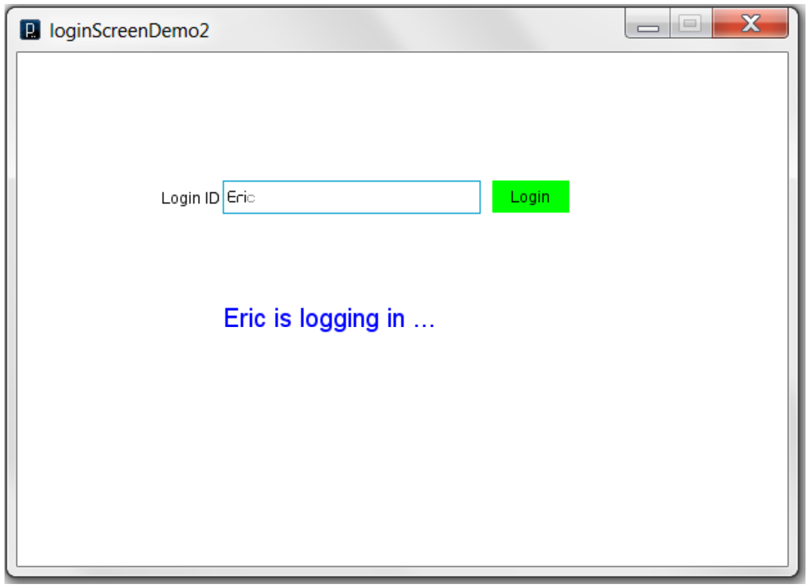
Eric is logging in ...

Working with Fonts on GUIs

- Apply different font types to controller captions or values (text inputted/outputted)
- Change controller captions' case (all upper by default)
- Adjust controller captions' location



→



Apply Fonts to Controller Caption

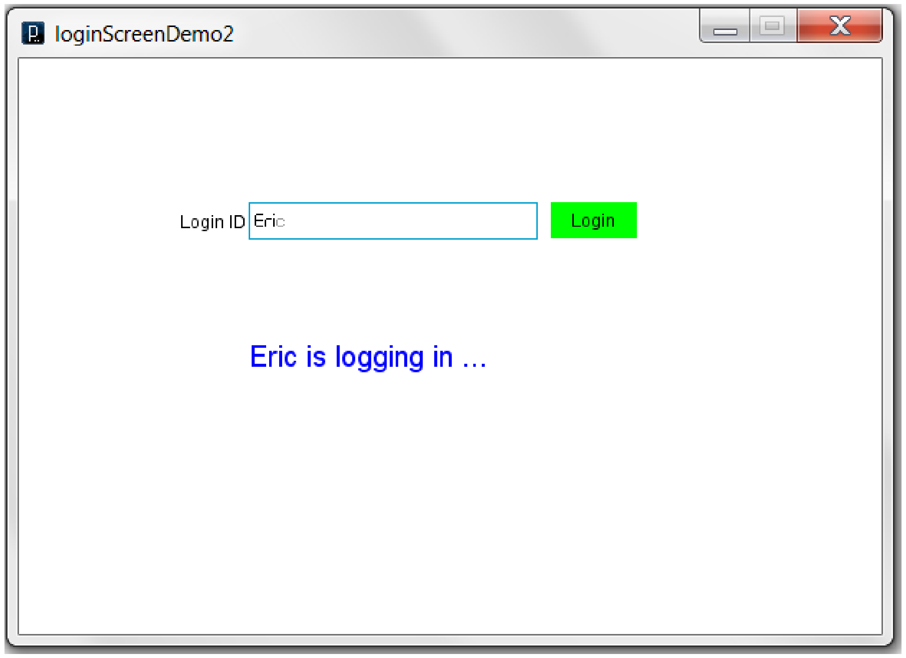
- Apply a font type to ***logLabel 's*** & ***logButton's*** captions

```
 …    //whatever was done so far
ControlFont font;  //ControlFont is a wrapper for PFont
                    //Its Constructor: Control(PFont pf)
void setup() {
  …   //whatever was done so far
 //First create a PFont object for ControlFont() argument
  PFont pfont = createFont("Arial",12,true);
  font = new ControlFont(pfont);
  logLabel = controlP5.addTextlabel("loginID","Login ID",
      110, 110);
  logLabel.setColorValue(color(0));
  logLabel.setControlFont(font);

  logButton = controlP5.addButton("Login",0,370,100,60,25);
  logButton.captionLabel().setControlFont(font);
}
```

Apply Font to Controller Caption

- Apply a different font type to ***resLabel's* caption**

```
void setup() {
  …    //whatever was done so far

  resLabel = controlP5.addTextlabel("resp","",160,200);
  resLabel.captionLabel().setControlFontSize(30);
  resLabel.setColorValue(color(0, 0, 255));
  resLabel.setControlFont(new ControlFont(createFont("Times",
       20, true)));

}
```



Eric is logging in ...

Apply Font to Controller’s Values

- Apply a different font type and size to ***logTextfield*’s value** (i.e. text inputted)

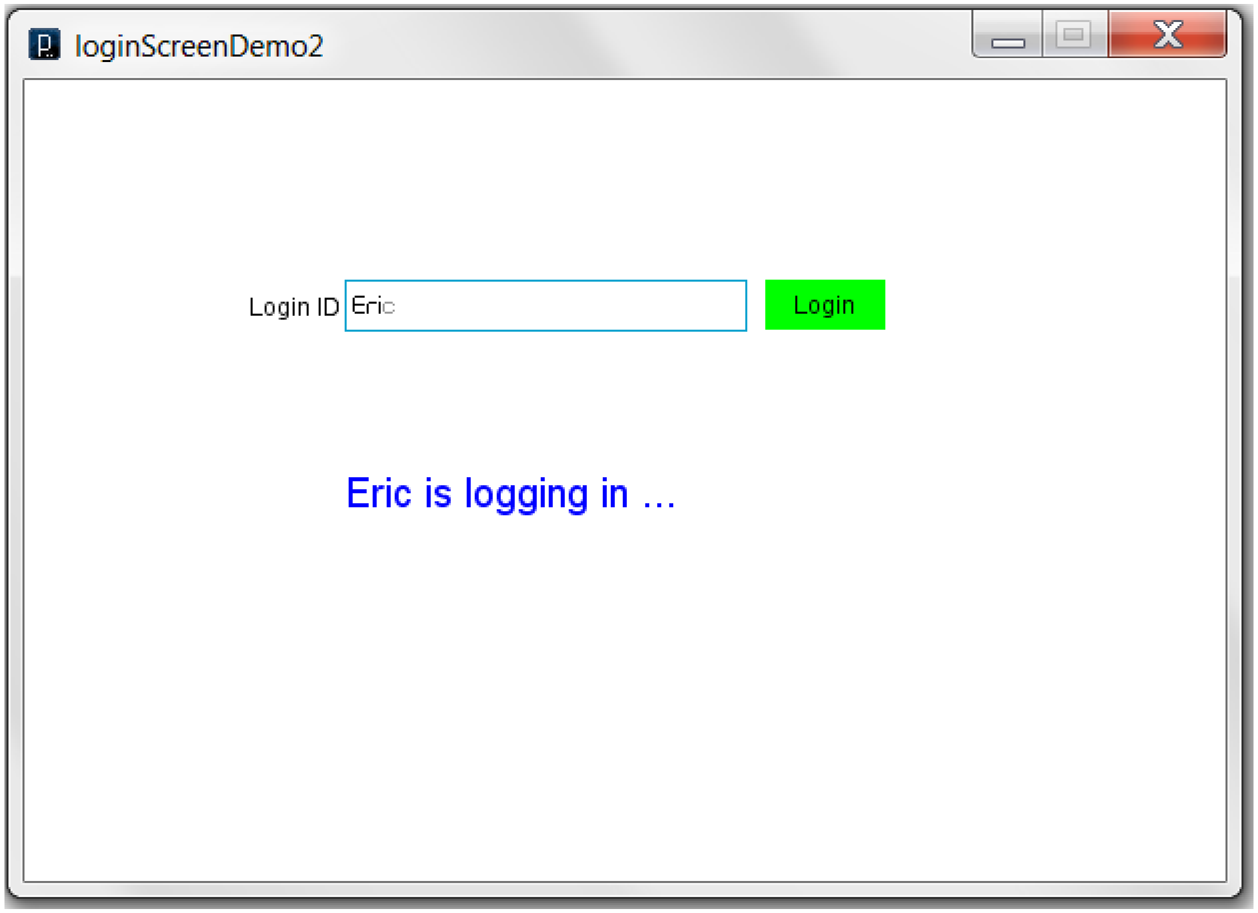
```
void setup() {
  …    //whatever was done so far

  logTextfield =  controlP5.addTextfield("logField",160,100,
       200,25);
  logTextfield.setColorValue(color(0));
  logTextfield.setColorBackground(color(255));

  //As per Sojamo (autor of ControlP5) Textarea and Textfield
  //don’t work with ControlFont at present. It’s on his todo
  //list though. For now  he suggests to use the slightly
  //bigger font 'ContrlP5.grixel' to walk around:
  logTextfield.valueLabel().setFont(ControlP5.grixel);

  …    //whatever was done so far
}
```

Adjust Controller Caption’s Cases & Location

- Decapitalize ***logButton*’s caption** (except for its initial) and adjust it’s ***location*** within the button

```
void setup() {
  …    //whatever was done so far

  logButton.captionLabel().setControlFont(font);
  //Change the letter case of caption for the button
  logButton.captionLabel().toUpperCase(false);
  logButton.captionLabel().set("Login");

  // Adjust the location of its caption using the
  // style property of a controller.
  logButton.captionLabel().style().marginLeft = 10;

  // you can also adjust its vertical location using
  // logButton.captionLabel().style().marginTop if necessary
}
```

This is what you end up with…



Eric

Eric is logging in …

Summary

- More on PFont
- Processing & Java Mode
- Processing & Java Mode
- GUIs and Widgets
- Event-Driven Programming
- Overview of ControP5 Architecture
- Create GUIs using ControlP5
  - Add controllers (buttons & text I/O widgets)
  - Handle events with callback method
  - Work with fonts on GUIs