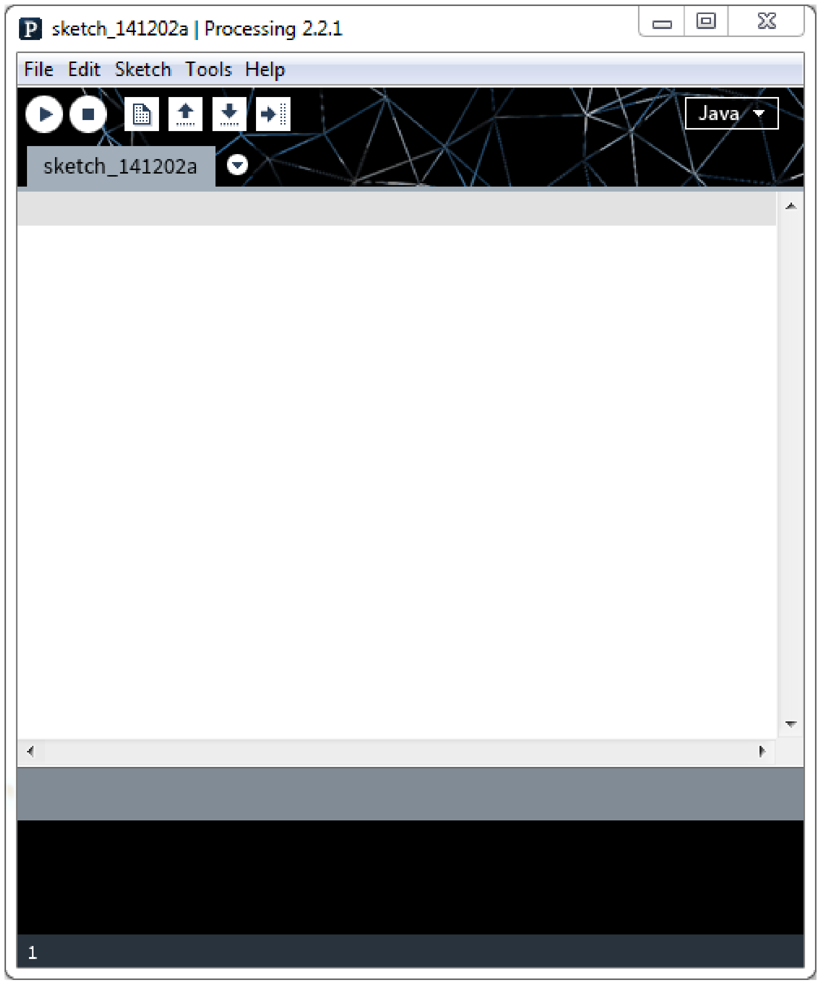
Event Driven Programming

Processing

Model-View-Controller

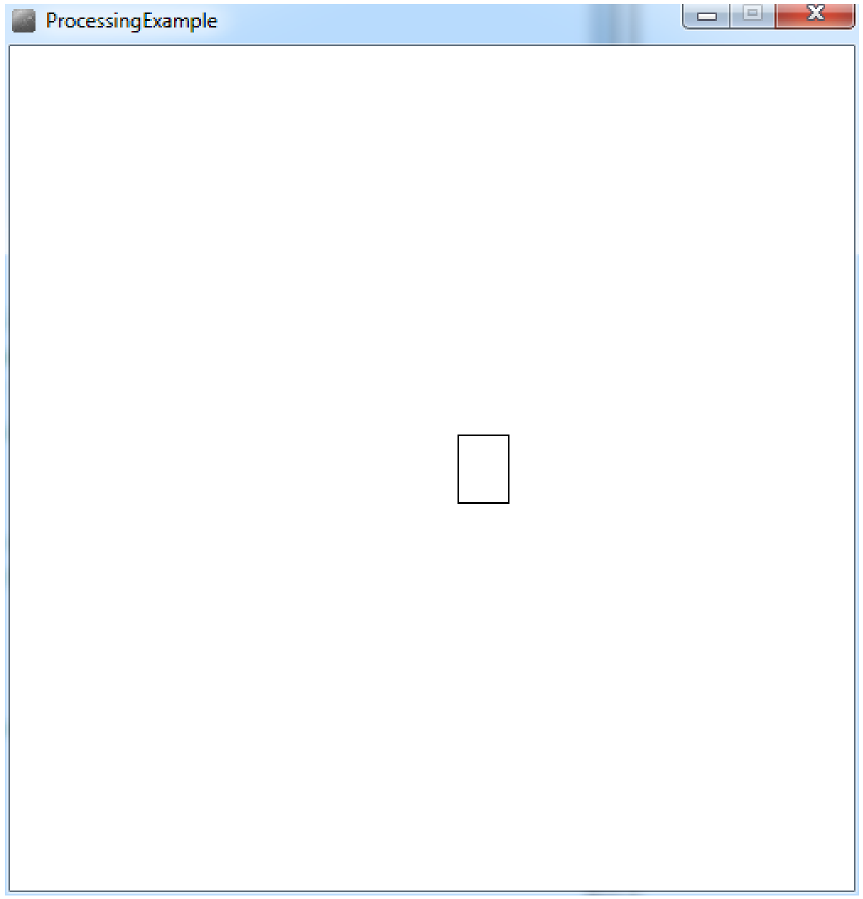
Reacting to User-driven Events

Processing


sketch_141202a | Processing 2.2.1
File Edit Sketch Tools Help
Java
sketch_141202a
1

https://processing.org/
https://processing.org/reference/

Using Processing in Java



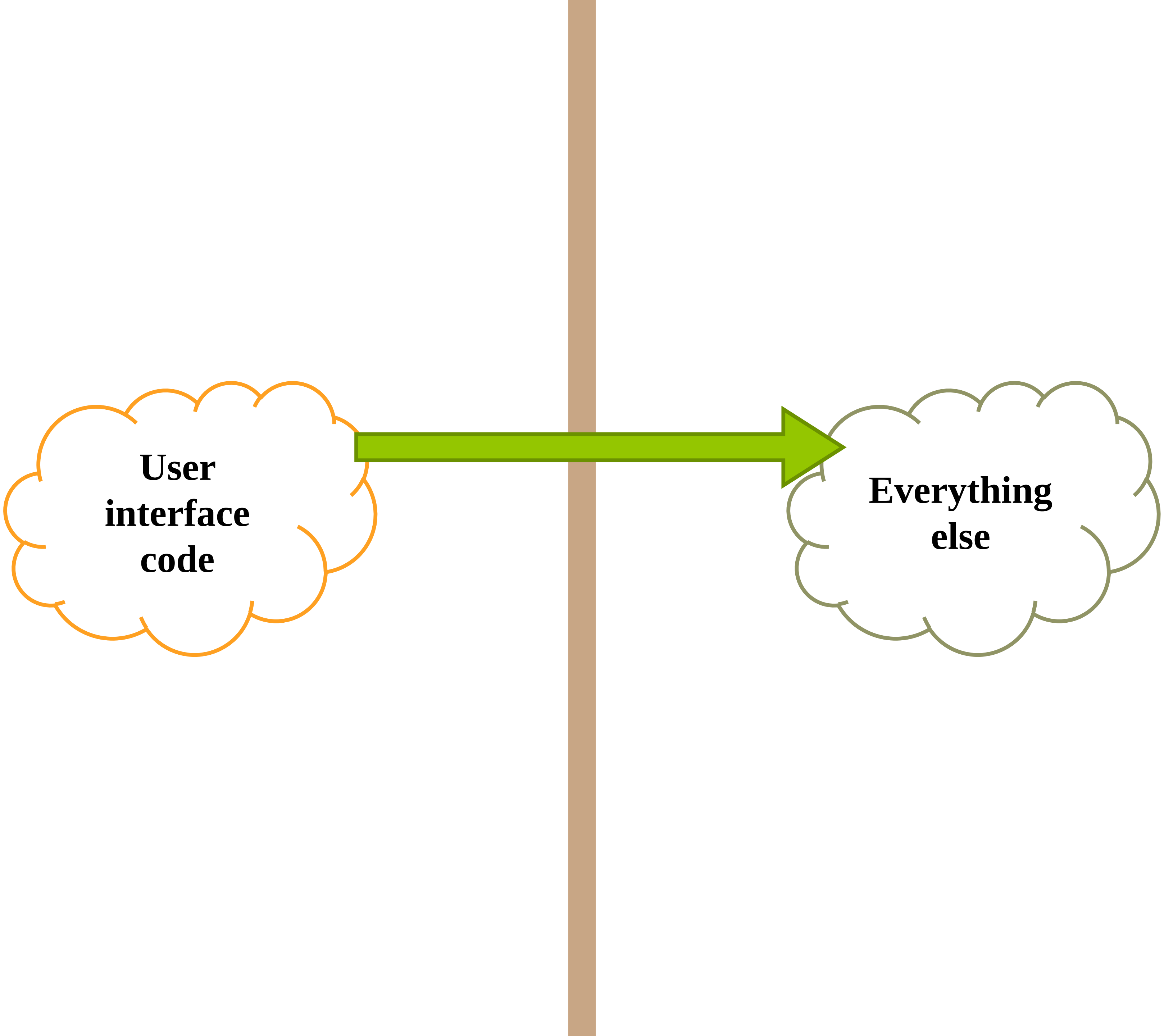
ProcessingExample.java

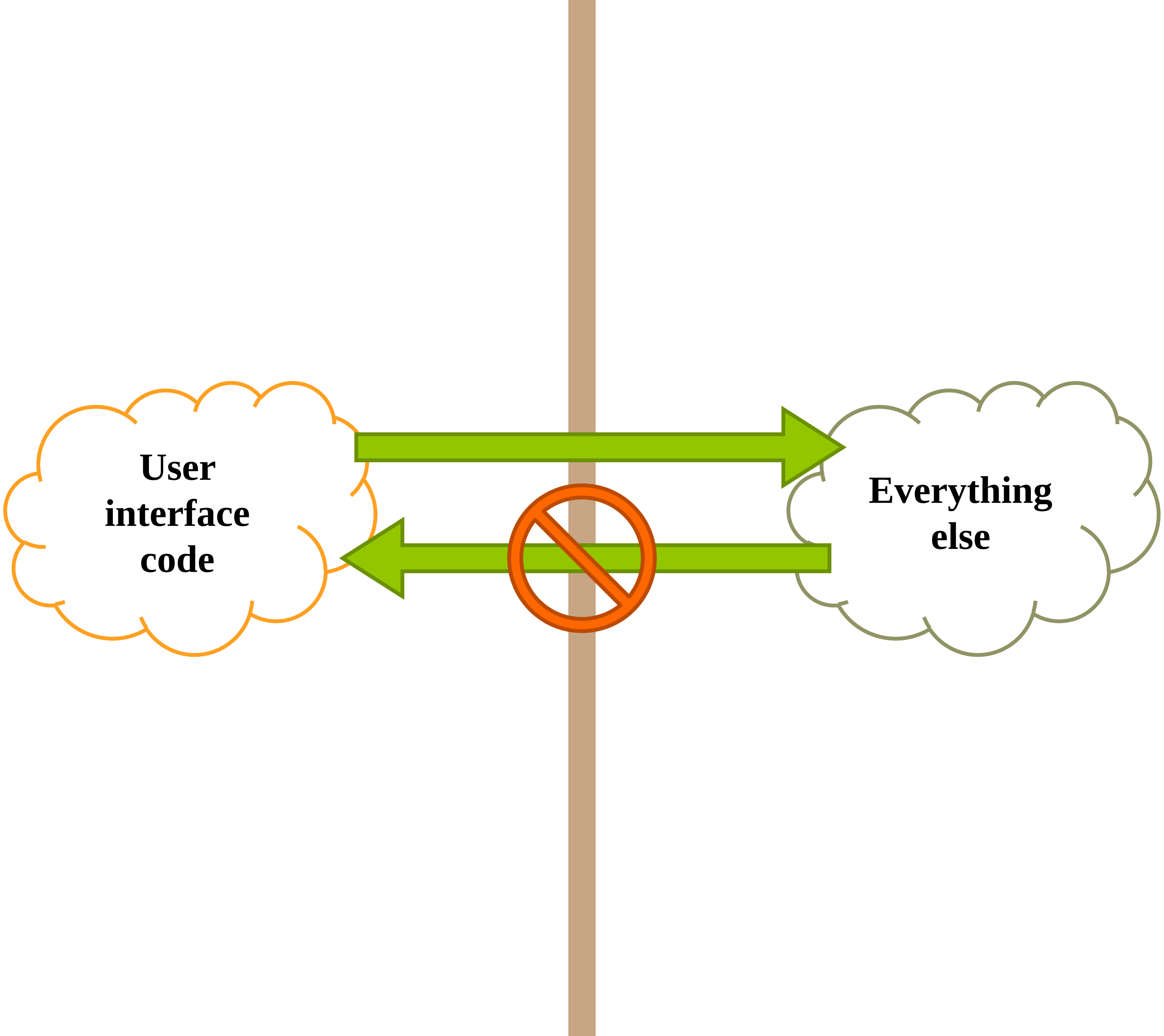
Model-View-Controller

Recall separated presentation design…

User interface code

Everything else


User interface code
Everything else


User interface code
Everything else

We can also call the two parts the “model” and the “user interface”

Model:
all classes that represent the "business logic" part of the application ... the underlying system on which a user interface is attached

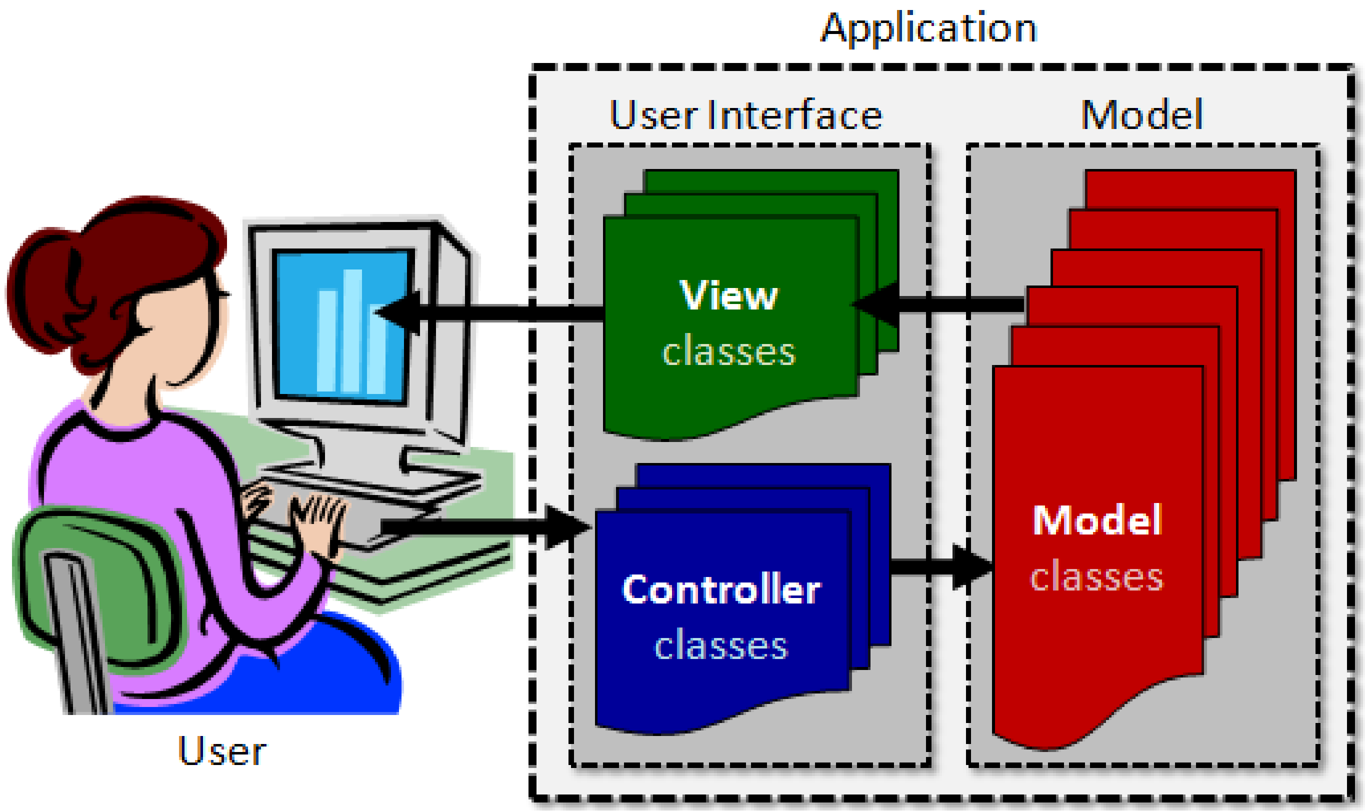
User Interface:
attached to the model, handles interaction with the user and does NOT deal with the business logic

We can further separate the user interface into the view and a controller…

View:
displays the necessary information from the model into a form suitable for interaction, typically a user interface element

Model:
all classes that represent the "business logic" part of the application ... the underlying system on which a user interface is attached

Controller:
accepts input from the user, modifies the model accordingly


Application
User Interface
Model
View
classes
Controller
classes
Model
classes
User

Why Use MVC?

Decouples models and view

Reduces **complexity** of overall architectural design

Increases **flexibility and maintainability** of code

Model-View-Controller with a Simple Processing App

Model:	Data that represents items that will eventually be drawn to the screen.
View:	Code that uses the model to put the items onto the screen using Processing commands.
Controller:	Subclass of PApplet that sets up the model and view as well as handles user events.

Structure of a PApplet Subclass

```
public class ClassName extends PApplet
{
  // References to model and view

  public void setup()
  {
    // Code to run once at beginning
  }

  public void draw()
  {
    // Code to run every frame
  }

  public static void main(String[] args)
  {
    PApplet.main(ClassName.class.getCanonicalName());
  }
}
```

Structure of the Draw Method

```
public void draw()
{
  // Clear the screen so we can draw a whole new
  // frame of the animation

  // Make updates to the model that should occur
  // every frame (such as movement)

  // Ask the user interface class to draw the model
  // according to its current state
}
```

Structure of a View Class

```
public class ViewName
{
  // Reference to instances of PApplet subclass
  // (usually called parent) and model

  // Constructor that takes instance of PApplet
  // subclass and model

  public void drawStuff()
  {
    // Use model to display things on the PApplet
    // instance using Processing commands
  }
}
```

Reacting to User-Driven Events

Event: something that happens in the program based on some kind of triggering input which is typically caused (i.e., **generated**) by user interaction

Event Handler: a procedure that specifies the code to be executed when a specific type of event occurs in the program

Handling Events in Processing

Simply override the method that corresponds to the event you want to handle:

```
public void mouseClicked() { … }
public void mouseDragged() { … }
public void mouseWheel() { … }
public void keyPress() { … }
```

(etc)

Handling Events in Processing

Use the attributes you have access to from `PApplet` to get details on the event

```
mouseX, mouseY
key
```
etc