EECS 1710
Programming for Digital Media

Lecture 24 :: Eclipse Toolset Introduction (& Processing plugin)

YORK UNIVERSITÉ UNIVERSITY

Next (final) lecture

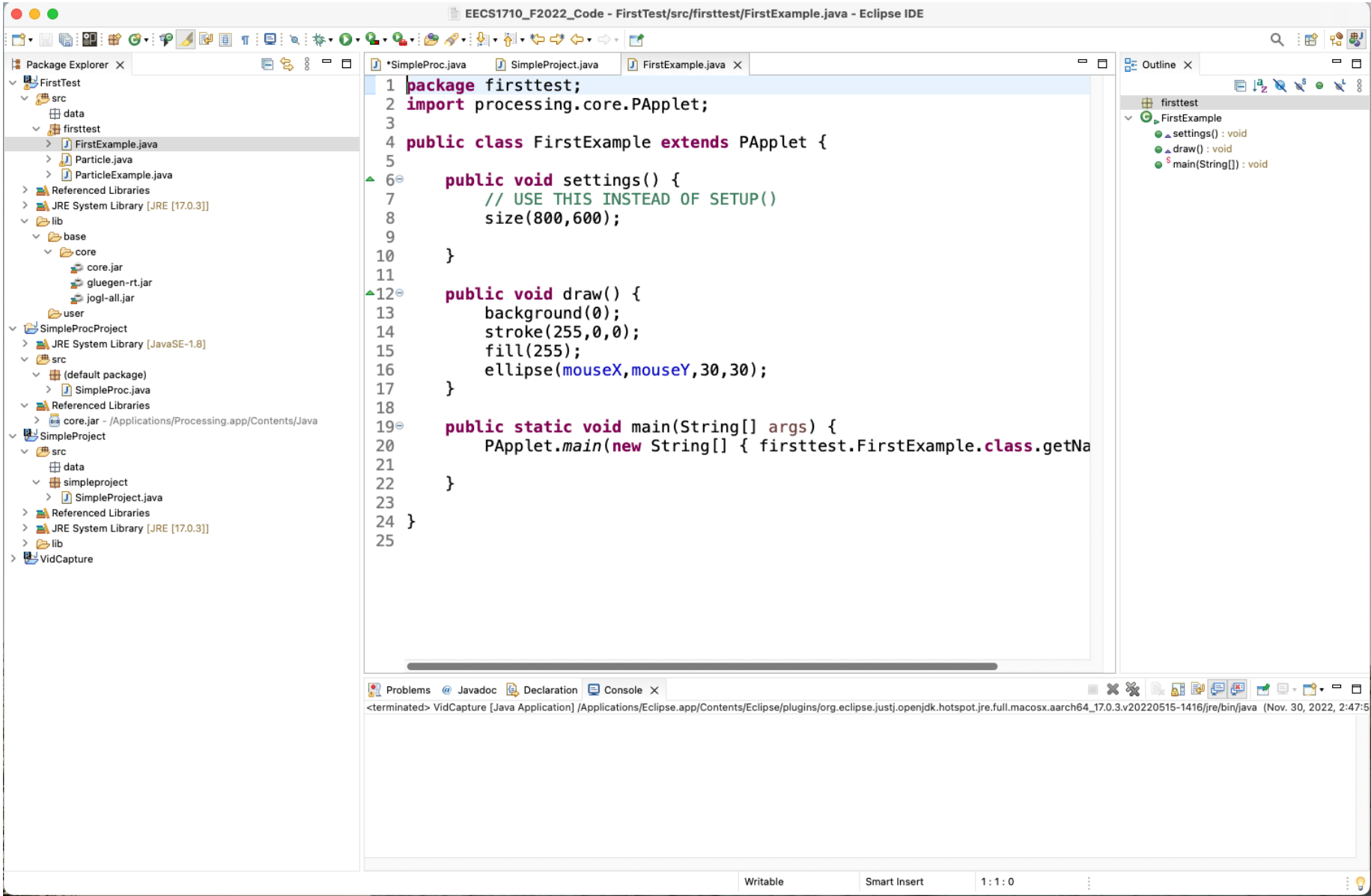
- Eclipse toolset
  - Installing Eclipse
  - Add Proclipsing to Eclipse (Eclipse plugin for Processing)
  - Basic Processing projects (porting from PDE)
- Java Anatomy Preview (non-processing version of things)
- Final Exam format (last slide – will talk more on Friday lecture about this)

1. Eclipse IDE

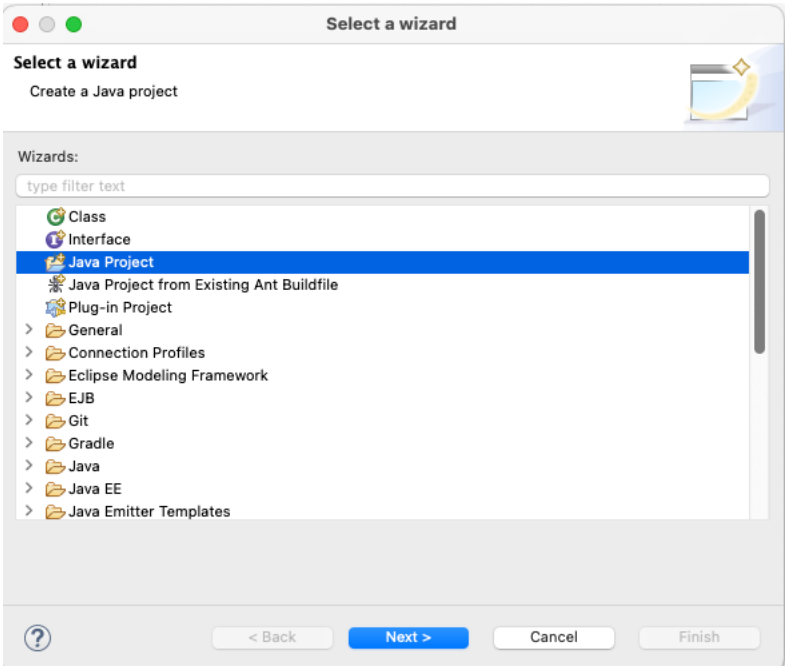
- https://www.eclipse.org/downloads/packages/

- Download “Eclipse IDE for Java Developers” for your operating system, and install

- (assumes you already have Processing installed… we will have to then tell Eclipse about where Processing is
  - Can add processing core as an external jar (java archive) file to a new java project
  - Easier way: install “Proclipsing” plugin (which allows us to open ”processing” projects – which include all the relevant jar files we need for processing)


EECS1710_F2022_Code - FirstTest/src/firsttest/FirstExample.java - Eclipse IDE
Package Explorer
FirstTest
src
data
firsttest
FirstExample.java
Particle.java
ParticleExample.java
Referenced Libraries
JRE System Library [JRE [17.0.3]]
lib
base
core
core.jar
gluegen-rt.jar
jogl-all.jar
user
SimpleProcProject
JRE System Library [JavaSE-1.8]
src
(default package)
SimpleProc.java
Referenced Libraries
core.jar - /Applications/Processing.app/Contents/Java
SimpleProject
src
data
simpleproject
SimpleProject.java
Referenced Libraries
JRE System Library [JRE [17.0.3]]
lib
VidCapture
*SimpleProc.java
SimpleProject.java
FirstExample.java
1 package firsttest;
2 import processing.core.PApplet;
3
4 public class FirstExample extends PApplet {
5
6 public void settings() {
7 // USE THIS INSTEAD OF SETUP()
8 size(800,600);
9
10 }
11
12 public void draw() {
13 background(0);
14 stroke(255,0,0);
15 fill(255);
16 ellipse(mouseX,mouseY,30,30);
17 }
18
19 public static void main(String[] args) {
20 PApplet.main(new String[] { firsttest.FirstExample.class.getNa
21
22 }
23
24 }
25
Outline
firsttest
FirstExample
settings() : void
draw() : void
main(String[]) : void
Problems
Javadoc
Declaration
Console
<terminated> VidCapture [Java Application] /Applications/Eclipse.app/Contents/Eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.aarch64_17.0.3.v20220515-1416/jre/bin/java (Nov. 30, 2022, 2:47:5
Writable
Smart Insert
1 : 1 : 0

Java project (with core.jar as external jar)



YORK UNIVERSITÉ UNIVERSITY

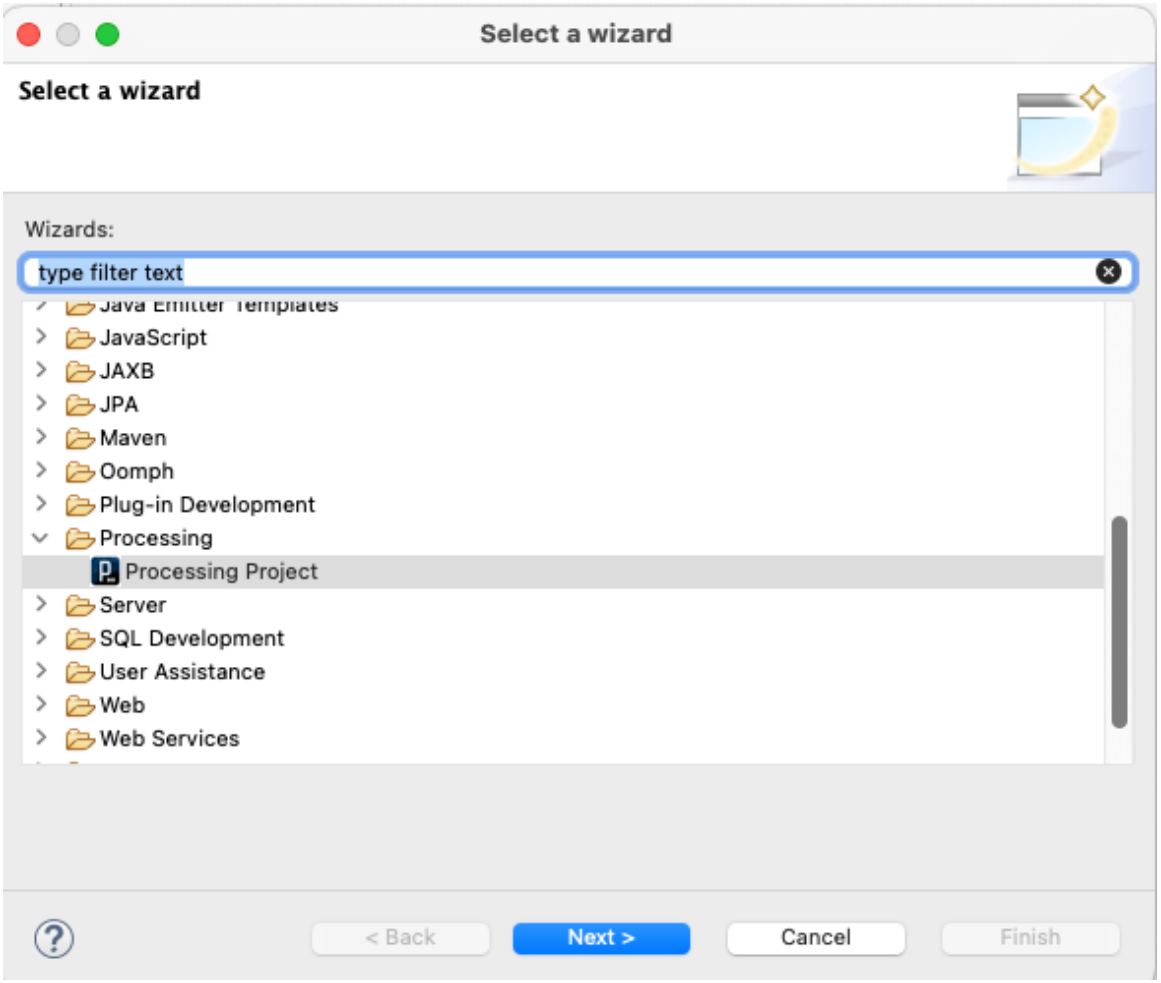
2. Proclipsing plugin for Eclipse

- https://github.com/ybakos/proclipsing

Installing

1. Download a release **.zip** file.
2. Unzip the file and place the folder in a convenient location.
3. In Eclipse, select the menu item *Help* > *Install New Software*. In the dialogue that appears, select the *Add...* button. Enter **Proclipsing** as the *Name:*, and select the *Local...* button. Navigate to the location of your Proclipsing folder, and within, select the **proclipsingSite** folder and select the *Open* button.
4. Back in the *Install* dialogue, select the *Select All* button, and then select *Next* >.
5. Accept the license agreement and select *Finish*. If you are prompted to restart Eclipse, do it.

After restarting, you should see this inside:



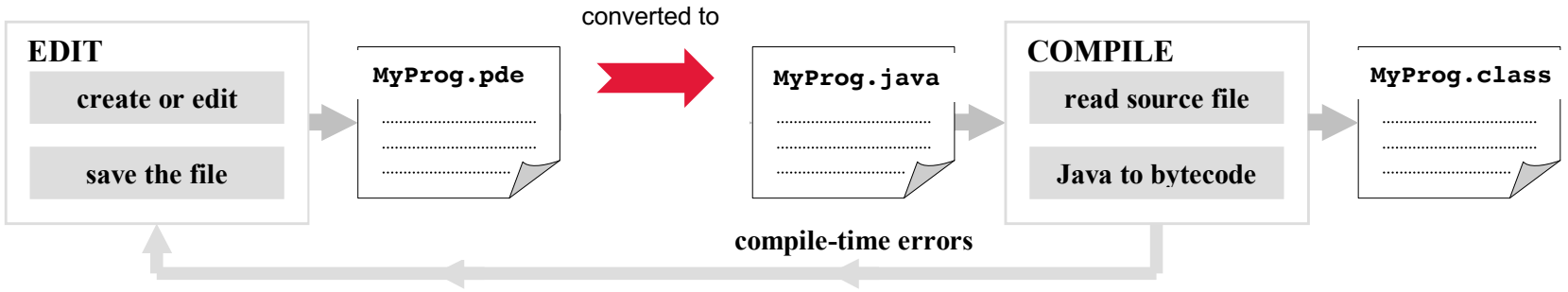
File→New→Project→Processing→Processing Project, open a new project

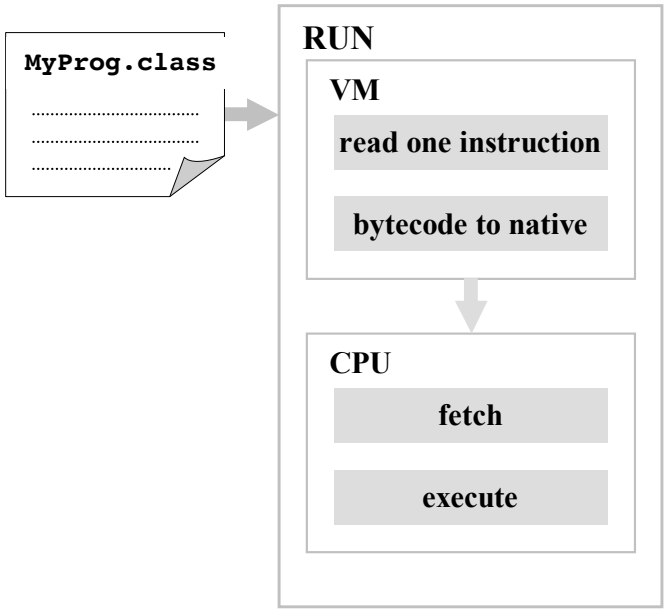
YORK UNIVERSITÉ UNIVERSITY

A couple of notes about pure Java (vs. Processing)

- In Java, every file is usually associated with a single class
  - It is possible to have classes within classes (for special circumstances, but generally only one class per file, and the class has to have the same name as the file)
- In Java, a project can have multiple files (i.e. multiple classes), but there must exist a class with a “main” method
  - This “main” method is always the entry (start) point of the program when run
  - If tracing a program, we always start our tracing from there

defining & executing a Processing sketch

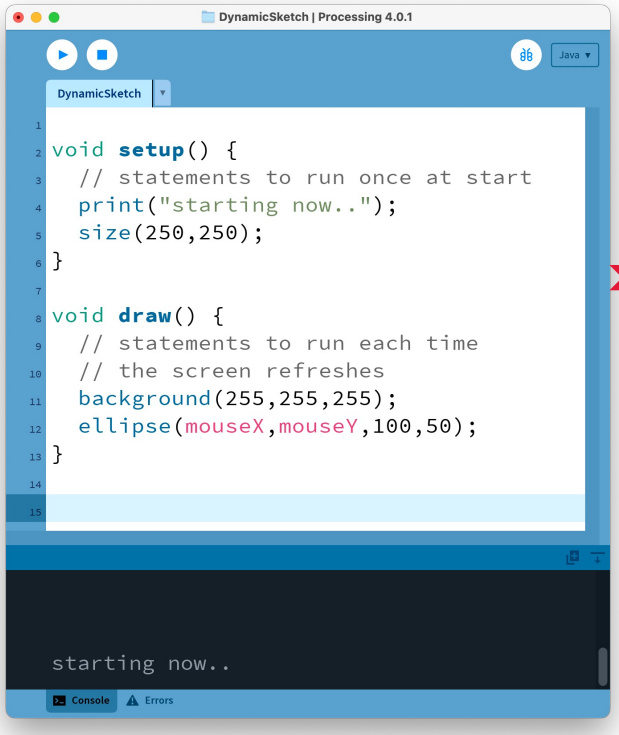




YORK UNIVERSITÉ UNIVERSITY

Source code

DynamicSketch.pde



Processing ~ Java (lite)

DynamicSketch.java

```
import processing.core.PApplet;

public class DynamicSketch extends PApplet {

      public void settings() {
       print("starting now..");
       size(250,250);
      }

      public void draw() {
       background(255,255,255);
       ellipse(mouseX,mouseY,100,50);
      }

      public static void main(String[] args) {
       String[] processingArgs = {"HelloSketch"};
       DynamicSketch mySketch = new DynamicSketch();
       PApplet.runSketch(processingArgs, mySketch);
      }
}
```

Java (full) – note the extra scaffolding!

DynamicSketch.class (Byte Code)

Recall: Anatomy of an API

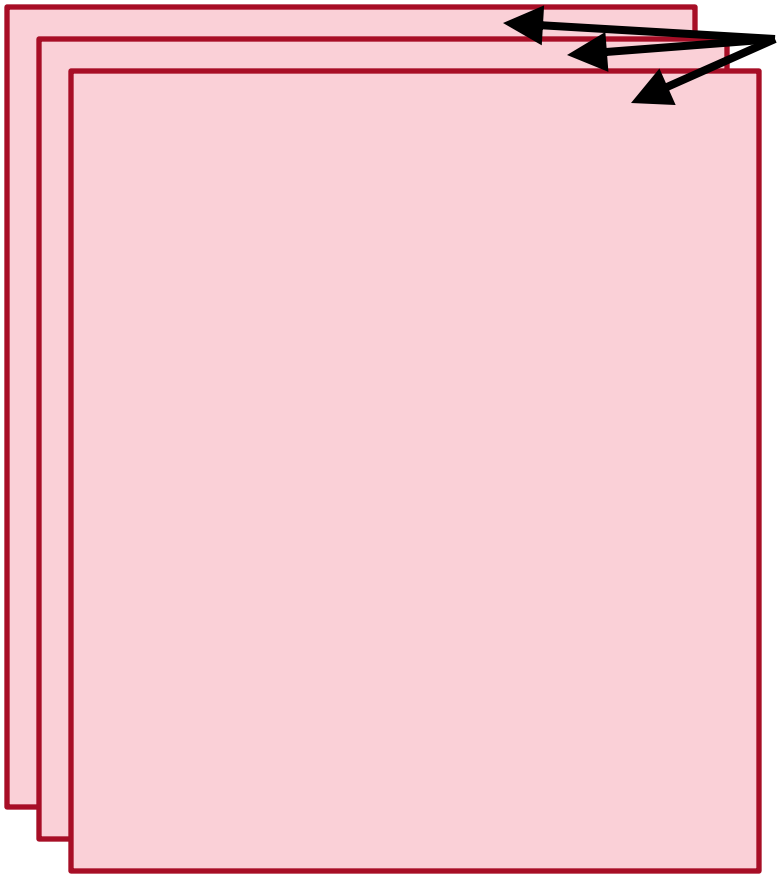
Packages	Details
	The Class section
	The Field section
Classes	The Constructor section
	The Method section

API ~ exposes public structure of a class
(how is this organized internally?)

Organization of a Java Program

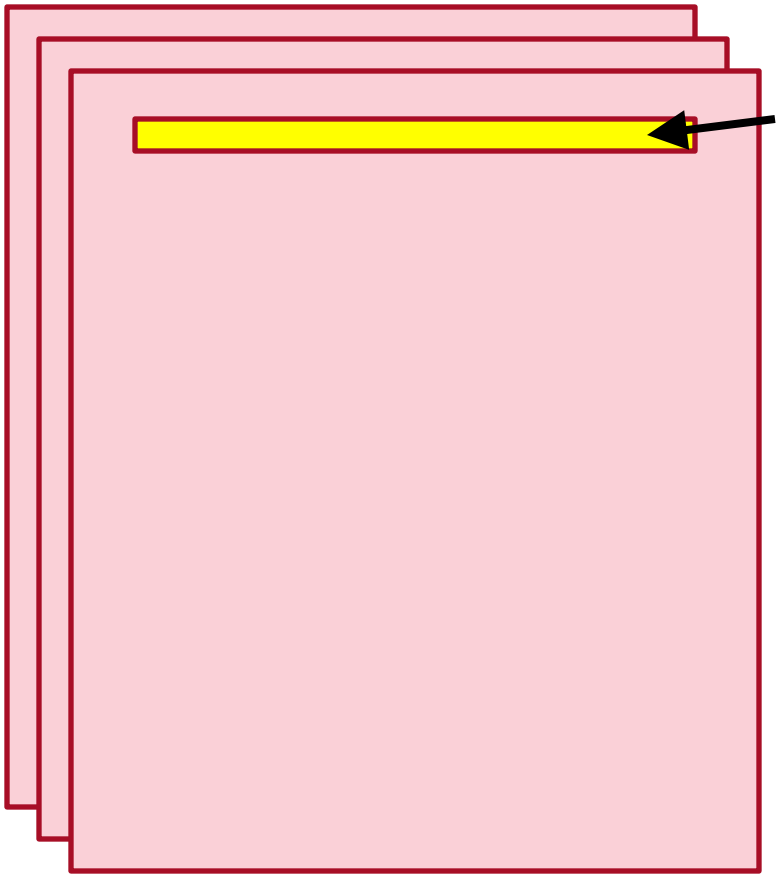
Packages, classes, fields, and methods

Organization of a Typical Java Program



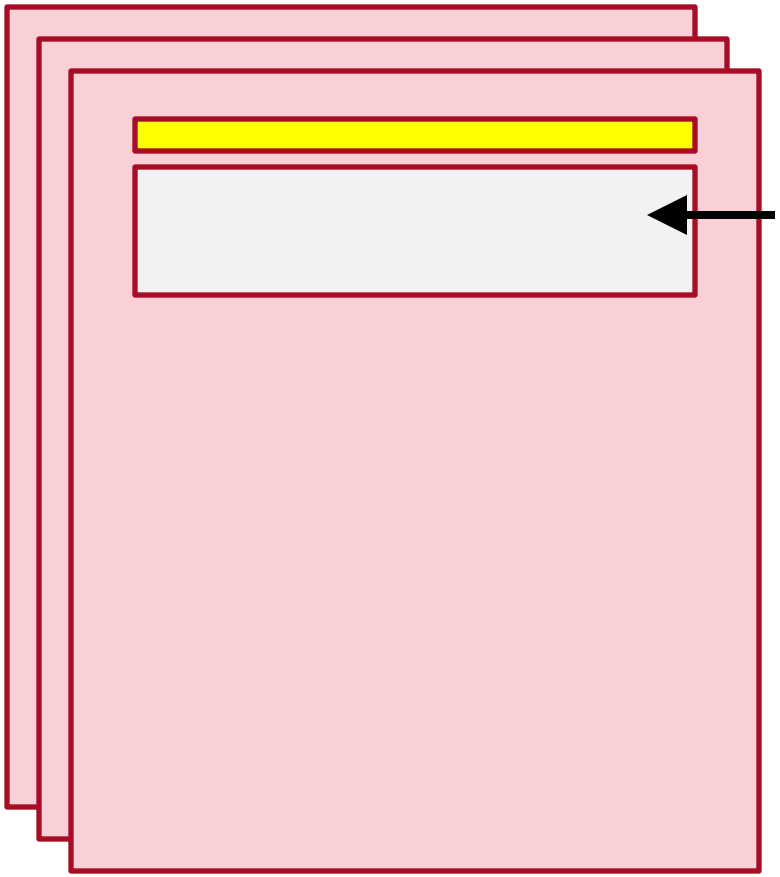
- one or more files

Organization of a Typical Java Program



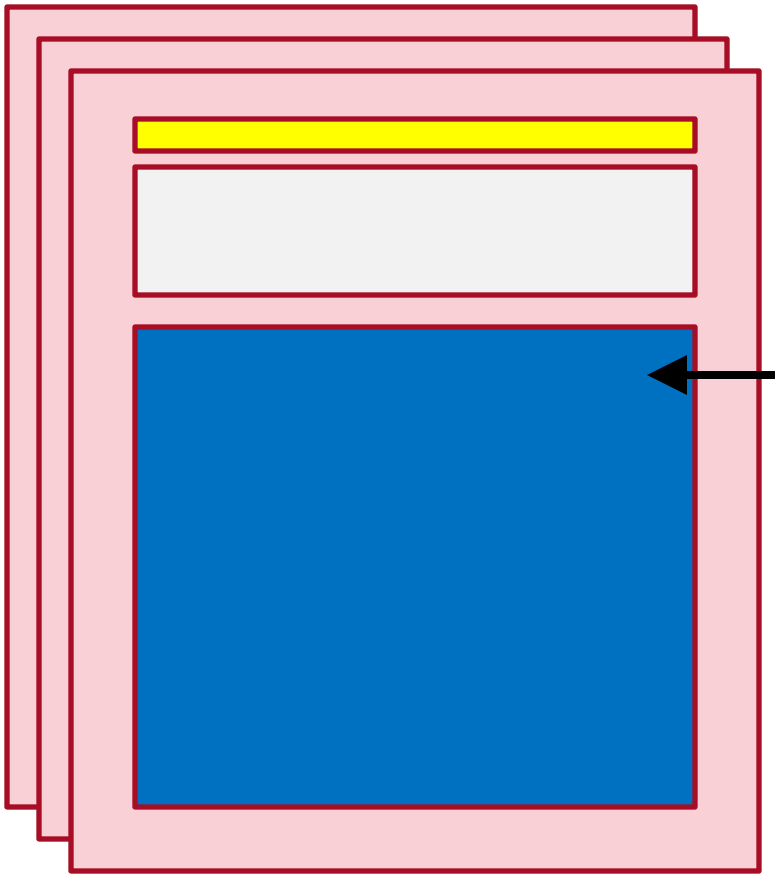
- one or more files
- zero or one package name

Organization of a Typical Java Program



- one or more files
- zero or one package name
- zero or more import statements

Organization of a Typical Java Program



- one or more files
- zero or one package name
- zero or more import statements
- one class

Opening a Processing Project

```
package simpleproject;
import processing.core.PApplet;

public class SimpleProject extends PApplet {

    public void setup() {

    }

    public void draw() {

    }

    public static void main(String _args[]) {
        PApplet.main(new String[] {
                simpleproject.SimpleProject.class.getName() });
    }
}
```

Opening a Processing Project

```
package simpleproject;
import processing.core.PApplet;

public class SimpleProject extends PApplet {

    public void settings() {

    }

    public void draw() {

    }

    public static void main(String _args[]) {
        PApplet.main(new String[] {
                simpleproject.SimpleProject.class.getName() });
    }
}
```

put setup() calls here

put draw() loop calls here

YORK UNIVERSITÉ UNIVERSITY

Example (simple draw app)

```
package firsttest;
import processing.core.PApplet;

public class FirstExample extends PApplet {

     public void settings() {
          // USE THIS INSTEAD OF SETUP()
          size(800,600);
     }

     public void draw() {
          background(0);
          stroke(255,0,0);
          fill(255);
          ellipse(mouseX,mouseY,30,30);
     }

     public static void main(String[] args) {
          PApplet.main(new String[] { firsttest.FirstExample.class.getName() });
     }
}
```

This is like your main.pde file (that is associated with the sketch folder)

PApplet is the “type” that Processing uses for a sketch

Keyword “public” ??

- Access modifier
- Tells program that this class/method/variable is fully accessible
- If not included, everything is assumed “public” – which is the default for Processing applications

- We will learn about access later (but for now, know there is a way to restrict and lock down parts of your code – so you can protect ip, or just make sure people who use your classes cant modify things in random ways (they are forced to use methods to modify things instead)

Projects with external dependencies?

e.g. using sound, etc

Example (external lib – sound)

```
package simpleproject;

import processing.core.PApplet;
import processing.sound.*;

public class SimpleProject extends PApplet {

     AudioSample sample;
     public void settings() {
          size(640, 360);
          int sampleRate = 44100;    // samples per sec = sample rate (SR)
          float freq = 440;          // frequency in Hz (cycles/sec)
          float[] sinewave = new float[sampleRate];

          for (int i = 0; i < sinewave.length; i++) {
          sinewave[i] = sin(TWO_PI*freq*i/sampleRate);
          }

          sample = new AudioSample(this, sinewave);
          sample.amp(0.5f);
          sample.play();

     }
     public void draw() {
     }
     public static void main(String _args[]) {
     PApplet.main(new String[] { simpleproject.SimpleProject.class.getName() });
     }
}
```

Porting some projects (e.g. Particle class)

How to access Processing methods etc from an additional class (not your main PApplet) – slightly more advanced use of Processing in Java

Porting some projects (e.g. Particle class)

- From last lecture…

- Recall, this class was in its own file :
  - a class called **Particle**… plus there was a second file:
  - **ParticleExample** → called from setup() & draw()

```
package firsttest;

public class Particle {

	final float GRAVITY = 9.8;
	final float DT = 0.1;
	PVector pos;
	PVector vel;
	int col;
	float radius;

	Particle(float x, float y, float dx, float dy, int c, float r) {

		pos = new PVector(x,y);
		vel = new PVector(dx,dy);
		col = c;
		radius = r;
	}

	void display() {
		fill(col);
		ellipseMode(RADIUS);
		circle(pos.x, pos.y, radius);
		stroke(0,0,0);
	}

	void move() {
		pos.x = pos.x + vel.x*DT;
		pos.y = pos.y + vel.y*DT;

		vel.y = vel.y + 0.5f*GRAVITY*DT*DT;  // includes acceleration term
	}

	public static void main(String[] args) {
	// TODO Auto-generated method stub
	}
}
```

```
package firsttest;
import processing.core.PVector;

public class Particle {

     final float GRAVITY = 9.8f;
     final float DT = 0.1f;
     PVector pos;
     PVector vel;
     int col;
     float radius;

     Particle( float x, float y, float dx, float dy, int c, float r) {
     pos = new PVector(x,y);
     vel = new PVector(dx,dy);
     col = c;
     radius = r;
     }

     void display(ParticleExample p5) {
         p5.fill(col);
         p5.ellipseMode(p5.RADIUS);
         p5.circle(pos.x, pos.y, radius);
         p5.stroke(0,0,0);
     }

     void move() {
         pos.x = pos.x + vel.x*DT;
         pos.y = pos.y + vel.y*DT;
         vel.y = vel.y + 0.5f*GRAVITY*DT*DT;  // includes acceleration term
     }

     public static void main(String[] args) {
     }
}
```

```
package firsttest;
import processing.core.PApplet;

public class ParticleExample extends PApplet {

     Particle bullet;
     Particle firework;

     public void settings() {

          size(800,600);

          bullet = new Particle(0,height/2,10,-10,
                         color(random(255),random(255),random(255)),
                         random(10)+10);

          firework = new Particle(width/4,height, 0,(float) (-10.0+random(10)-10.0),
                              color(random(255),random(255),random(255)),
                              random(10)+10);
     }

     public void draw() {
          background(255,255,255);
          bullet.display(this);
          bullet.move();
          firework.display(this);
          firework.move();

     }

     public static void main(String _args[]) {
          PApplet.main(new String[] { firsttest.ParticleExample.class.getName() });

     }

}
```

We will use some Processing

- But we will also explore other classes and types that are part of the core of the Java language

- If you want to practice with the Eclipse tools before 1720, try porting some of the code from 1710 into this Java form..
    - Note: easier for some of the earlier code (single *.pde files)
    - A little more challenging for code using multiple files, and/or external libraries
        - Make a note of any issues you come across and we will talk more about how to resolve in EECS1720

FINAL EXAM FORMAT

- More on this in Fridays lecture (virtual via ZOOM)
  - Format:
    - 40-50 multiple choice questions
      - Variables/objects, loops, conditionals, arrays, ArrayLists (IntList,FloatList, etc)
      - Looping on an array/ArrayList (you will have access to a datasheet if you need to remember methods)
      - Structure of a processing app
      - Types of errors, Style conventions, etc.
    - 2 short answer questions
      - 1 on audio
        - Understand the frequency/tone generation process
        - Understand how to use AudioSample and SoundFile
      - 1 on image/moving image
        - Creating an image (by setting pixels)
        - Blending images (or taking something from multiple images)
        - Flipping/manipulation/chromakey/etc.
    - You will be given example data sheet with reference info you need to answer questions (e.g. like Processing reference manual … i.e. has method description/signatures, etc)