Aalto University
School of Arts, Design
and Architecture

Programming for Visual Artists

2024/2025
Department of Art and Media

Some things to check

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

Recap

IN CASE YOU
MISSED IT

- Functions

  Just this was a lot! ☺

Today…

Welcome

Recap

Today's Goals

Object-Oriented Programming (OOP)

What is?

Usage in art

Creating objects

Arrays

Particle Systems

Examples

BREAK (10:30–10:45)

Coding tasks

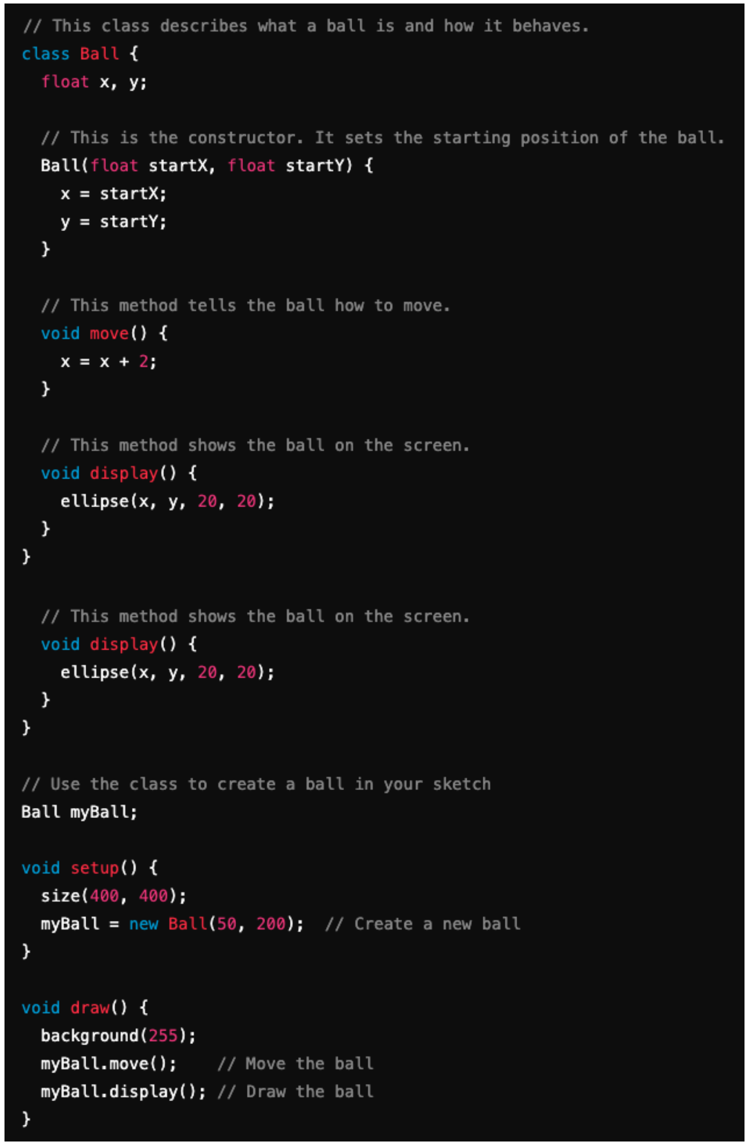
3 bursts of particles

Particles & Perlin Noise

Q&A

For tomorrow…

Object-Oriented Programming (OOP)



```
// This class describes what a ball is and how it behaves.
class Ball {
  float x, y;

  // This is the constructor. It sets the starting position of the ball.
  Ball(float startX, float startY) {
    x = startX;
    y = startY;
  }

  // This method tells the ball how to move.
  void move() {
    x = x + 2;
  }

  // This method shows the ball on the screen.
  void display() {
    ellipse(x, y, 20, 20);
  }
}

  // This method shows the ball on the screen.
  void display() {
    ellipse(x, y, 20, 20);
  }
}

// Use the class to create a ball in your sketch
Ball myBall;

void setup() {
  size(400, 400);
  myBall = new Ball(50, 200);  // Create a new ball
}

void draw() {
  background(255);
  myBall.move();    // Move the ball
  myBall.display(); // Draw the ball
}
```

Is way of writing code that organizes everything into "objects." You can think of objects like real-life items that have their own features and actions.

Classes: A class is like a recipe or blueprint. It tells you what an object will look like and what it can do.

Objects: An object is a specific item created from that recipe. For example, if you have a class for a ball, each ball you create is an object with its own position, speed, etc.

Definition: Encapsulation means keeping all the details (data and functions) that belong together in one place (a class). This makes it easier to manage and change your code.

Your class holds the ball's position and the functions that move and draw the ball, keeping everything related to the ball together.

In short, OOP in Processing makes your coding more organized and easier to work with, especially when you start building larger projects.

Examples of OOP for Art

Shapes as Objects: Imagine each shape, like a circle, square, or line, as its own object. You can create a class for each type of shape. Each object can have properties such as size, colour, and position, and methods that determine how it moves or changes.

A "Particle" class could represent a small moving dot.

When you create many particles, each one acts independently, creating dynamic, evolving patterns.

Interactivity: With OOP, you can easily add interactivity. For example, you might create objects that respond to mouse clicks, change colour when hovered over, or interact with each other in interesting ways.

User Input: Using objects, you can have different elements that react differently to user input, making it “feel alive” and interactive.

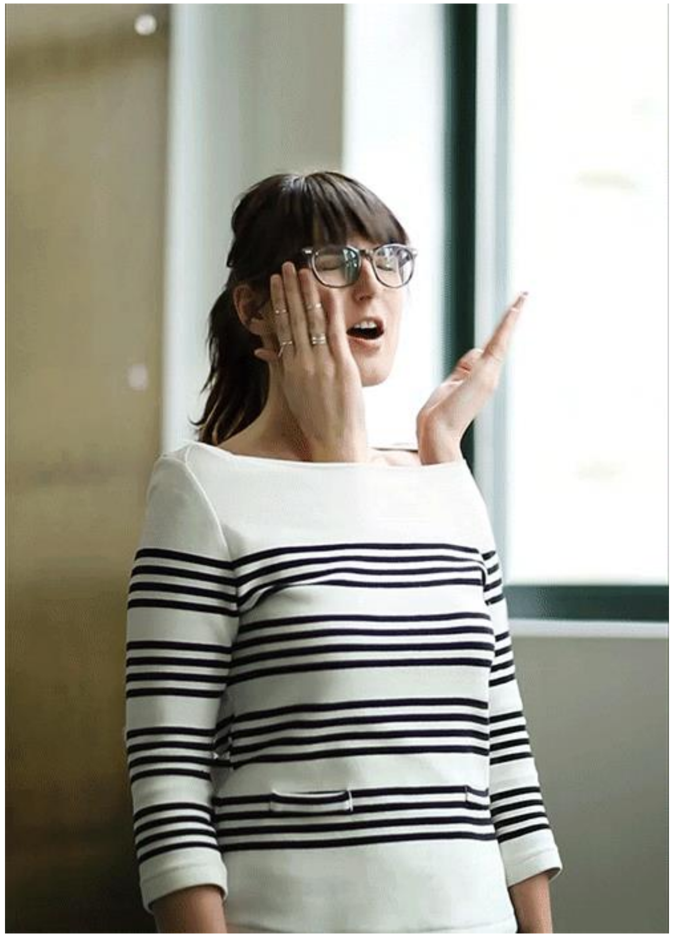
Modularity: OOP helps break down a complex artwork into manageable pieces. Instead of writing one long piece of code, you organize your art into separate classes, each handling a part of the scene.

Reusability: Once you write a class for a visual element, you can reuse it across different projects or parts of the same project, saving time and making your code cleaner.

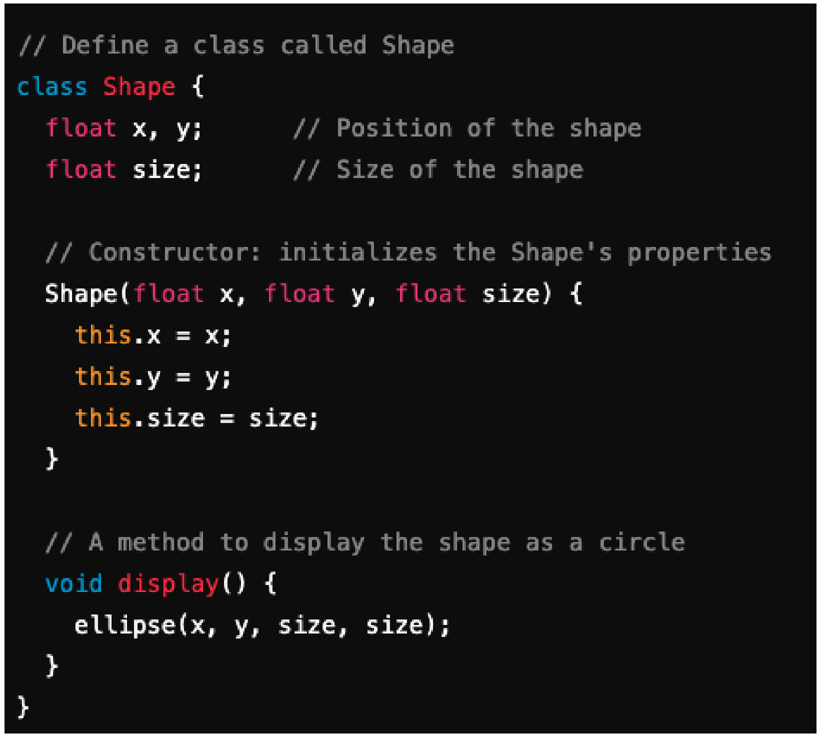
Example code OOP

Check on github – exampleOOP1

https://github.com/ptiagomp/aalto-programming-visual-artists-24-25/tree/main/Session-06_11032025



OOP – creating objects

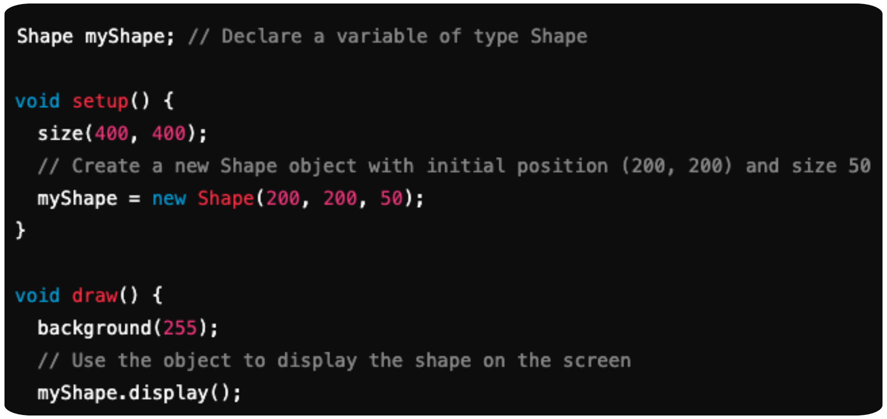


```
// Define a class called Shape
class Shape {
  float x, y;       // Position of the shape
  float size;       // Size of the shape

  // Constructor: initializes the Shape's properties
  Shape(float x, float y, float size) {
    this.x = x;
    this.y = y;
    this.size = size;
  }

  // A method to display the shape as a circle
  void display() {
    ellipse(x, y, size, size);
  }
}
```

Creating an object in Processing is as simple as using the new keyword along with a class's constructor. An object is an instance of a class, meaning it has all the properties and behaviours (methods) defined by that class.

In the example, consider a simple class called **Shape.**

OOP – creating objects



```
Shape myShape; // Declare a variable of type Shape

void setup() {
  size(400, 400);
  // Create a new Shape object with initial position (200, 200) and size 50
  myShape = new Shape(200, 200, 50);
}

void draw() {
  background(255);
  // Use the object to display the shape on the screen
  myShape.display();
```

To create an object (an instance of the **Shape** class), you use the new keyword along with the class's constructor. Here's how you do it:

Shape myShape; declares a variable that will hold a **Shape** object.

Inside setup(), we create the object using the constructor of the **Shape** class.

The new keyword creates a new instance.

The parameters (200, 200, 50) are passed to the constructor, initializing the **x, y, and size** properties.

In draw(), we call the display() method on **myShape**. This tells the object to draw itself (in this case, as an ellipse) on the screen.

Arrays

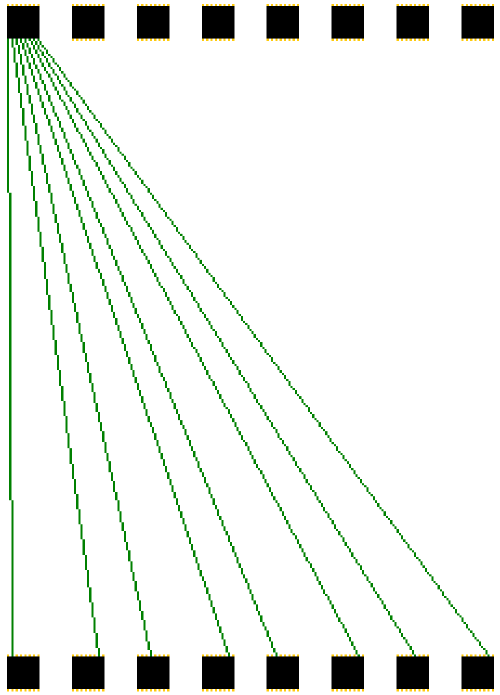
Arrays in Processing are like lists that let you store multiple values or objects under one name. Each value in an array has an index (its position in the list), starting at 0. This is useful when you want to work with many items without having to create a separate variable for each one.

Single Type: An array holds items of the same type (like numbers, strings, or objects).

Indexing: Items are accessed by their index, where the first item is at index 0.

Fixed Size: Once created, the array's size is fixed (although there are dynamic structures like ArrayList if you need more flexibility).

While traditional arrays have a fixed size, you can use dynamic structures like **ArrayLists** to allow your collection to grow or shrink during runtime.



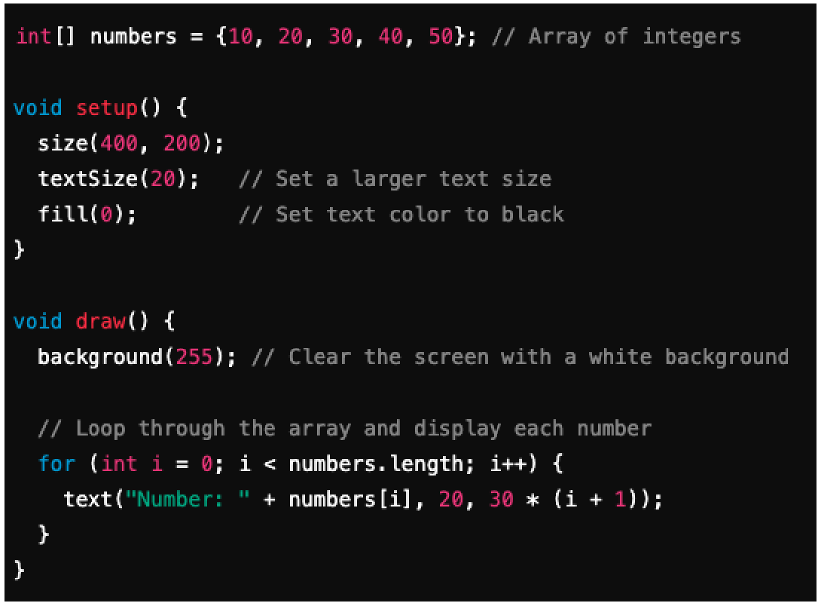
Arrays - static

In this example, we create an array of numbers and display them on the screen.

The line **int[] numbers = {10, 20, 30, 40, 50};** creates an array named numbers that holds five integers.

The for loop goes through each element in the array using numbers.length to know how many elements there are. Inside the loop, we display each number using the text() function.

The numbers are drawn on the screen at different vertical positions, so they don't overlap.



```
int[] numbers = {10, 20, 30, 40, 50}; // Array of integers

void setup() {
  size(400, 200);
  textSize(20);   // Set a larger text size
  fill(0);        // Set text color to black
}

void draw() {
  background(255); // Clear the screen with a white background

  // Loop through the array and display each number
  for (int i = 0; i < numbers.length; i++) {
    text("Number: " + numbers[i], 20, 30 * (i + 1));
  }
}
```

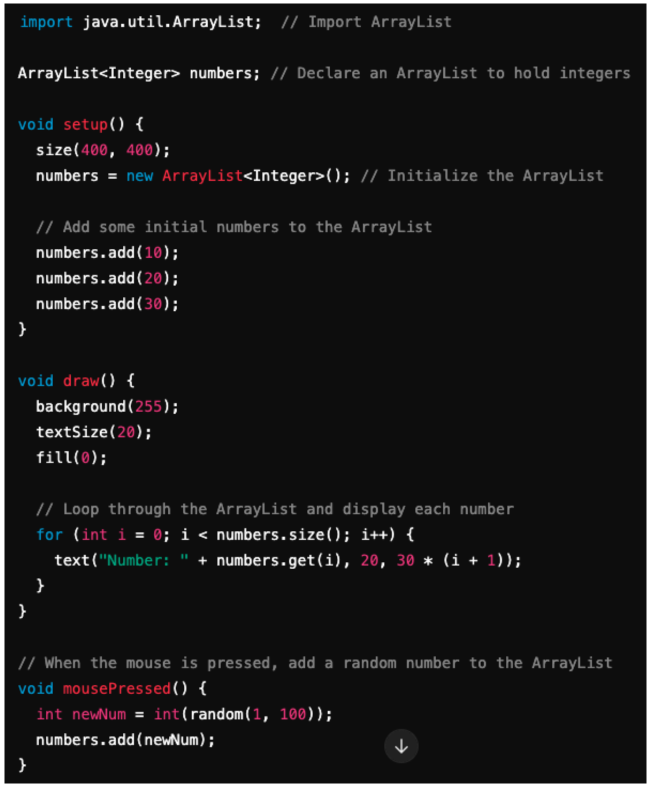
Arrays - dynamic

This example uses an ArrayList (a dynamic array) to store and display numbers. Each time you **click the mouse**; a new random number is added to the list.

We declare numbers as an ArrayList that stores integers and then initialize it in the setup() function.

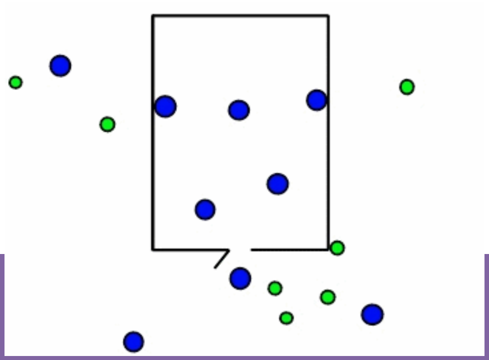
Three numbers (10, 20, 30) are added to the ArrayList initially. Each time you click the mouse; a new random number is added to the end of the list.

In the draw() function, we loop through the ArrayList and use text() to display each number on the screen.



```
import java.util.ArrayList;  // Import ArrayList

ArrayList<Integer> numbers; // Declare an ArrayList to hold integers

void setup() {
  size(400, 400);
  numbers = new ArrayList<Integer>(); // Initialize the ArrayList

  // Add some initial numbers to the ArrayList
  numbers.add(10);
  numbers.add(20);
  numbers.add(30);
}

void draw() {
  background(255);
  textSize(20);
  fill(0);

  // Loop through the ArrayList and display each number
  for (int i = 0; i < numbers.size(); i++) {
    text("Number: " + numbers.get(i), 20, 30 * (i + 1));
  }
}

// When the mouse is pressed, add a random number to the ArrayList
void mousePressed() {
  int newNum = int(random(1, 100));
  numbers.add(newNum);
}
```

Particle systems



A particle system is a technique used to simulate fuzzy, dynamic phenomena like smoke, fire, rain, or sparks by managing many small objects called particles. Each particle behaves on its own, but together they create complex, organic visuals.

Concepts:

- **Particles:**
  These are the individual elements in the system. Each particle usually has properties such as position, velocity, size, and lifespan.
- **Emitter:**
  This is the source that creates and releases particles. An emitter can be a point, a line, or an area.
- **Behaviour:**
  Each particle can have simple behaviours like moving, fading out, or growing/shrinking over time. Gravity, wind, or other forces can also be applied.
- **Lifespan:**
  Particles often have a limited lifespan. Once a particle reaches the end of its life, it is removed from the system to make room for new particles.

Particle systems

Check on GitHub: https://github.com/ptiagomp/aalto-programming-visual-artists-24-25/tree/main/Session-06_11032025

In the example, particles are created at the mouse position, move with random velocities, and fade out over time.

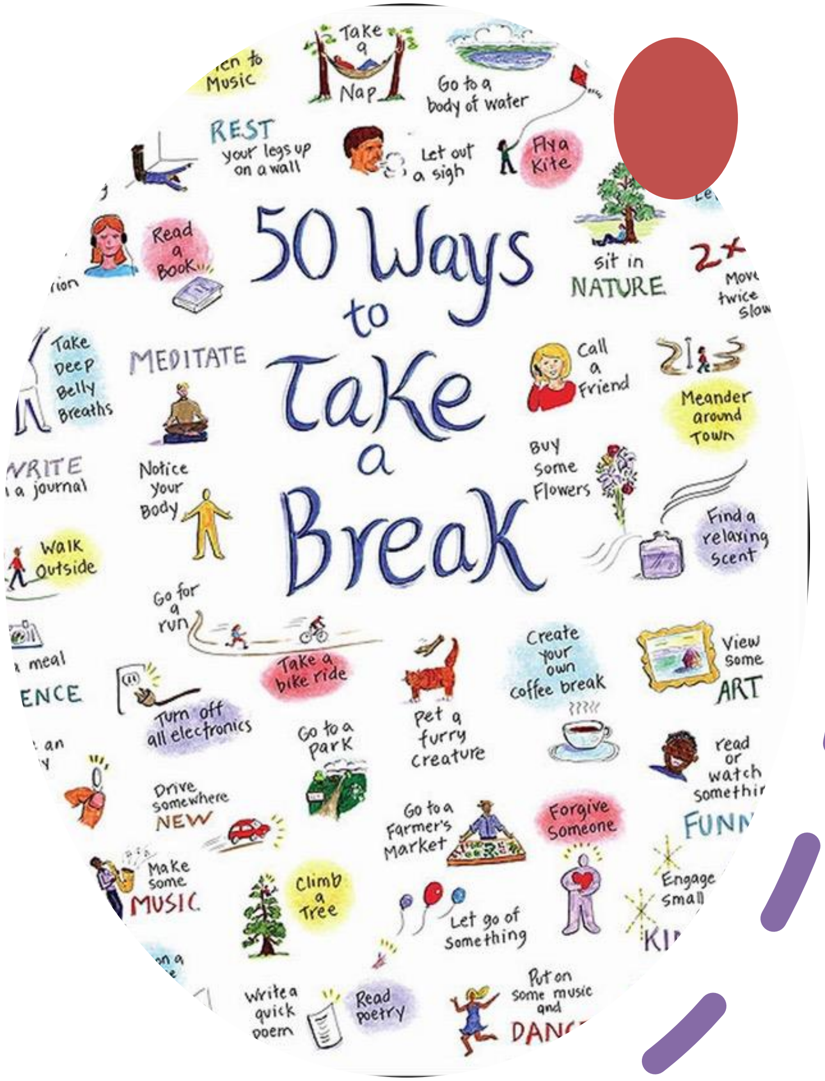
- Each particle is represented as an instance of the **Particle** class. It stores its own **position (pos), velocity (vel), and lifespan**.
- The update() method changes the particle's position and decreases its lifespan. The display() method draws the particle with an opacity proportional to its remaining life.
- In the draw() loop, new particles are added when the mouse is pressed. As particles update, they gradually fade away and are removed once their lifespan ends.

Particle systems allow you to create realistic and visually effects by managing lots of small, independent elements. This approach is especially popular in interactive art and games because it can produce complex behaviour from simple rules.

Break

15 min.!

Please don't be late!



Hands-On Exercise!

Objectives:

- Understand how a classes work.
- Understand how arrays work.
- Understand the basics of particles.
- Experiment! Try out things!

Get an example from here:
https://github.com/ptiagomp/aalto-programming-visual-artists-24-25/tree/main/Session-06_11032025

(if bored, check for the "extra" files!)

Discussion & Q&A

Share your feedback!
Am I going too fast or too slow? Is this too easy or too hard?

Next week's topics:

- Probably go through this all again!
- P5.JS
- Web-based generative work!

Don't forget the assignments!