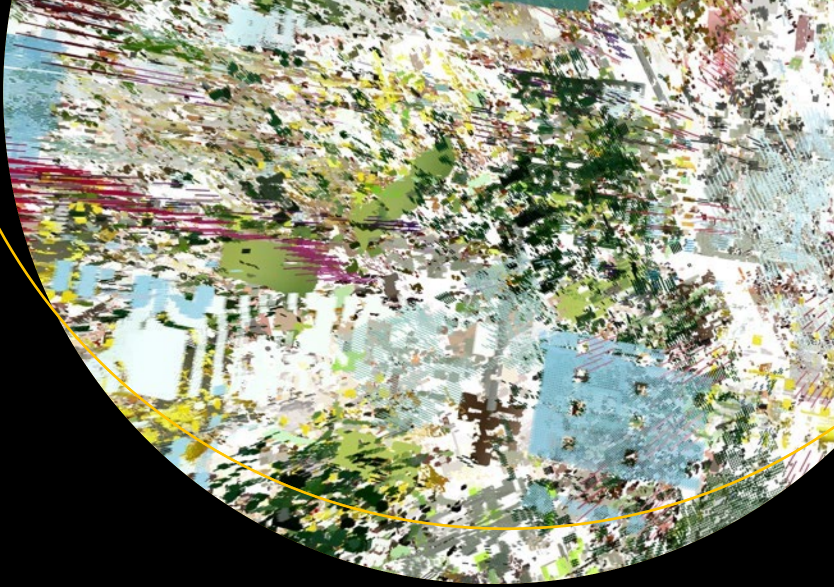


Processing for Android

Create Mobile, Sensor-Aware, and VR Applications Using Processing

—

Andrés Colubri

Apress®

Processing for Android

Create Mobile, Sensor-Aware, and VR Applications Using Processing

Andrés Colubri

Apress®

Processing for Android: Create Mobile, Sensor-Aware, and VR Applications Using Processing

Andrés Colubri
Cambridge, Massachusetts, USA

ISBN-13 (pbk): 978-1-4842-2718-3 ISBN-13 (electronic): 978-1-4842-2719-0
https://doi.org/10.1007/978-1-4842-2719-0

Library of Congress Control Number: 2017958640

Cover image designed by Freepik

Managing Director: Welmoed Spahr
Editorial Director: Todd Green
Acquisitions Editor: Natalie Pao
Development Editor: James Markham
Technical Reviewer: Anthony Tripoldi
Coordinating Editor: Jessica Vakili
Copy Editor: April Rondeau
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, email `orders-ny@springer-sbm.com`, or visit `www.springeronline.com`. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please email `rights@apress.com`, or visit `http://www.apress.com/rights-permissions`.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at `http://www.apress.com/bulk-sales`.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at `www.apress.com/978-1-4842-2718-3`. For more detailed information, please visit `http://www.apress.com/source-code`.

Printed on acid-free paper

To Jihyun, for her encouragement to take on new challenges.
To my family, for their support while being far away.

Contents at a Glance

Contents

About the Author

Andrés Colubri is a programmer, scientist, and artist, and long-time contributor to the Processing project. He originally obtained a doctoral degree in mathematics at the Universidad Nacional del Sur in Bahía Blanca, Argentina, and was a Burroughs Wellcome postdoctoral fellow at the University of Chicago, where he studied the protein folding problem. Later on, he completed an MFA from the Design Media Arts department at the University of California, Los Angeles. Andrés uses Processing to carry out research at the intersection between interaction, visualization, and computing. He currently works at the Broad Institute of Harvard and MIT, developing new methods and tools for analysis of biomedical data. `http://andrescolubri.net/`

About the Technical Reviewers

Jose Luis Garcia del Castillo is an architect, computational designer, and educator. He advocates for a future where programming and code are tools as natural to designers as paper and pencil. In his work, he explores creative opportunities at the intersection of design, technology, fabrication, data and art. His current research focuses on the development of digital frameworks that help democratize access to robotic technologies for designers and artists.

Jose Luis is a registered architect, and holds a Master in Architectural Technological Innovation from Universidad de Sevilla and a Master of Design Studies in Technology from the Harvard University Graduate School of Design. He has worked as a structural consultant for several international firms, such as OMA, Mecanoo, and Cesar Pelli, as well as data visualization architect at Fathom Information Design. He is also the co-founder of ParametricCamp, an international organization whose mission is to spread the knowledge of computational design among designers and architects.

Jose Luis currently pursues his Doctor of Design degree at the Material Processing and Systems group at the Harvard Graduate School of Design, works as Research Engineer in the Generative Design Team at Autodesk Inc., and teaches computational creativity in the Arts+Design Department at Northeastern University.

Anthony Tripaldi started programming in 2004, with a background in design and animation. By 2007 he went all in on Flash as the future of the web. Once realizing his devastating mistake, he pivoted, making Android apps and interactive installations with Processing for clients of all types. Once the ad industry had taken its toll, he found his way into Google Creative Lab, where he helped lead the creation of Android Experiments, a platform for artists and engineers alike to celebrate creativity on Android.

Gottfried Haider, born 1985 in Vienna, works as an artist, educator and software tool-maker. He received a degree in Digital Arts at University of Applied Arts Vienna in 2009. After receiving a Fulbright Scholarship in 2010, he pursued a MFA in Design Media Arts at University of California Los Angeles, which he finished in 2013. His artwork has been displayed in various venues and publications internationally.

Acknowledgments

During the past few months, I have learned that writing a book demands a significant personal effort, but, at the same time, it is the unequivocal confluence of the work, ideas, dreams, and passions of countless individuals. Since it would be impossible to track all the threads coming into the junction represented by this book, I will acknowledge the invaluable contributions of those who are most directly related to this project, knowing I will leave out many others.

I would like to start by extending my most earnest gratitude to Ben Fry, Casey Reas, and Daniel Shiffman, without whom this book would not have been possible. Their tireless work with Processing has enabled people from all over the world to use code in art and design, and made it possible for those coming from the sciences and engineering (like myself) to discover the possibilities of visual literacy within technology. Very special recognition goes to Jonathan Feinberg, who together with Ben, wrote the initial version of the Android mode the book is based upon, as well as to Pardis Sabeti, for supporting and encouraging my Processing work with her tireless enthusiasm. I also want to thank all the members of the Processing Foundation for putting forward a framework that promotes learning, diversity, and inclusion in the creative coding community and that empowers work such as mine.

Daniel Sauter and Jesus Duran made very important contributions to the Processing for Android project early on with their Ketai library, and also by organizing the Mobile Processing Conference at the School of Art and Design at the University of Illinois at Chicago, to which I was invited as speaker and workshop instructor in all editions from 2011 to 2013. These events were key to sustaining the initial momentum of the project into the present, and so my deepest recognition goes to them.

I would like to make a very special mention of all the Google Summer of Code students who worked on various Processing for Android projects during the past years: Sara Di Bartolomeo and Rupak Daas (GSoC 2017), Mohammad Umair (GSoC 2015), and Imil Ziyaztdinov (2014). Their contributions were absolutely crucial to the continued improvement and growth of Processing for Android. I am also very grateful for the existence of the GSoC program, which allowed these extraordinary coding and mentorship experiences with students of diverse backgrounds and origins.

Many thanks to Daniel Murphy, Shayan Amir-hosseini, Richard The, and Jen Kurtzman from Google Creative Lab in New York, who supported the renewed efforts behind the Processing for Android project during 2016, which led to this book.

As I mentioned at the beginning, a book like this is the result of the work of many people, but among them the technical reviewers played a critical role in ensuring that the book is correct, well structured, and easy to understand by its final audience. My reviewers, to whom I am deeply indebted, are Anthony Tripaldi, Kate Hollenbach, Jose Luis García del Castillo y López, and Gottfried Haider.

Likewise, I am very grateful to the Apress editors, Natalie Pao, Jessica Vakili, and James Markham, who, with their hard work and professionalism (and their patience with a novice writer), accompanied me along the way from draft idea to published book.

I would like to acknowledge Neil Zhao and Dean Kessey (on behalf of her late mother, Dr. Masumi Hayashi) for graciously allowing me to use reproductions of their work in the book.

Last, but not least, I thank the entire Processing community for their endless creativity, enthusiasm, and appreciation, which has been one of my most important motivations over the years.

Preface

By Ben Fry

The Android version of Processing got its start when I received an email from Andy Rubin back in February 2009. He was interested in having Processing be more tightly integrated with Android. The discussion led to initial funding, which helped us work on building an Android version of the project during the year or two that followed. For one of the test applications, I used some code developed by Casey Reas (the co-founder of the Processing project), and we were elated to see the first version of it up and running on the G1, the very first widely available Android device.

I was especially excited, and still am, about the Android platform as an incredible canvas to work from. You have mobile devices with a range of ways to communicate with the outside world (mobile network, wi-fi, Bluetooth, etc.), higher-performance CPUs than even the desktop machines we first used to develop Processing in 2001 (not to mention GPU performance better than the SGI workstations we were using around the same time), and a broad range of sensors—accelerometer, magnetometer, GPS, and gyroscope—that opened an amazing number of possibilities for applications and ideas. It's the platform I wish we'd had when we first started the project, with its focus on all the interesting use cases that come from openness and having the right set of tools and APIs to go along with it.

I'm especially excited, for instance, about possibilities like completely reinventing the mobile phone interface, because all those pieces are available to be customized, and you're just a few activities or fragments away from an entirely different end-user experience. Since the Handspring (later Palm) Treo, mobile phone interfaces really haven't changed much—a grid of applications as a home screen, and a phone "app" that essentially emulates the interface of a touch-tone phone. The Treo's interface dates to 2008 and built on the Palm Pilot interface from 1997, which itself referenced other organizer tools and too many icon-driven interfaces to mention from decades prior. The touch-tone phone, meanwhile, dates to at least 1968. Suffice to say, we're still working with a set of forms, interactions, and metaphors that have been heavily repurposed over the years, and I'm drawn to the idea of people experimenting with alternative interfaces and having the ability to rethink how those elements might work. Innovation comes from getting people the right tools to play with ideas, and while a new phone interface may not necessarily be around the corner, there's so much to be learned during the process itself.

This idea of experimentation and exploration is at the core of the Processing project. Projects are called "sketches" to keep people in the mindset of writing a small amount of code to get started, not overthinking it too much until they understand their problem better. This is not unlike scribbling something in a sketchbook and flipping to a new page—perhaps even tearing out the previous page and throwing it out. This approach is a little backward, as typical software engineering practice encourages figuring out structure first and filling in the details over time. However, we found that our iterative approach was helpful not just for our professional work, but also for teaching beginners how to get started. We built Processing in a way that allowed people to write a few lines of code that would make something happen—no knowledge of classes, threads, animation loops, or double-buffering necessary. Over time, as the user's experience grew and their ideas became more ambitious, they could learn the details of these more complex concepts. Interestingly, this mirrored things that we, as more seasoned developers, also wanted: why write a lot of the same boilerplate to do the same things? Why get RSI retyping the same handler and utility functions? Could we give people a toolbox that was useful enough on its own that starting a project didn't mean collecting the initial set of libraries that were required just to get things done?

A related idea was simply how one gets started. Most integrated development environments (IDEs) require a lot of background—even training—to use. Even though I had 20 years of coding experience, a friend had to sit me down to explain how to use Eclipse to set up a project. With Processing, it's a matter of downloading the software, writing a single line of code, and hitting the Run button. This gets even more important with a platform like Android, and we set out to make the Android version of Processing just as simple. We wanted users to be able to download the software, write a little code (or open an example), plug in their phone (or tablet, or who-knows-what), and hit the Run button to see something show up on the device. Once that works, it should be all set, and hopefully you're having enough fun that you lose the rest of the afternoon hacking away at your idea.

This book covers a wide range of ideas on how the Android platform can be used and how Processing can be a helpful bridge to creating everything from quick experiments to professionally developed applications. It's exciting to have Andrés writing it, as we want to see even more people building and playing with the platform, and also because you couldn't have a better expert in how Processing itself works. Andrés first started working as a core committer to the Processing project when we folded his popular OpenGL and Video libraries into the main project, which over the years led to his overseeing the 3D and Video portions of Processing. His experience with 3D got him deeply involved in the internals of how OpenGL works on Android, and his work there was the basis for his rebuilding the desktop version to move from the old-style fixed-function pipeline used in GL to the brave new world of shaders and highly multicore GPUs. It was through this experience, combined with his being an avid Android user, that Andrés became a core maintainer for the Android portion of the Processing project. He has helped shepherd it ever since, including everything from ongoing development to mentoring Google Summer of Code projects to, finally, writing this book. Suffice to say, you're in good hands.

I hope you'll enjoy this book, and we can't wait to see what you build with Processing!

PART I

First Steps with Processing for Android

CHAPTER 1

Getting Started with Android Mode

In this chapter, we will introduce the Processing software and Android mode, the community project behind them, and how we can begin using the mode to create apps for Android devices.

What Is the Processing Project?

The Processing project is a community initiative focused on sharing knowledge, fostering education, and promoting diversity in code-based art and design. The Processing software is a central part of this initiative, now guided by the Processing Foundation (`https://processingfoundation.org/`). The Processing software was created in 2001 by Casey Reas and Ben Fry at the MIT Media Lab as a teaching and production tool in computational arts and design, and has been evolving continuously since then. It is available for download at `https://processing.org/`, and its source code is released under free software licenses (GPL and LGPL). From now on, I will simply refer to Processing when talking about the Processing software.

Processing consists of two complementary pieces: the language and the development environment. Together, they form a "software sketchbook" designed to allow the expression of visual ideas quickly with code, while also providing enough room to let those ideas develop into full-blown projects. Processing has been used to create many beautiful and inspiring works in generative art, data visualization, and interactive installations, some of which are included in a curated list on the Processing site (`https://processing.org/exhibition/`).

The Processing Language

The Processing language comprises a set of functions for handling screen drawing, data input/output, and user interaction. A small team of volunteers behind the Processing project (`https://processing.org/people/`) has carefully constructed this set of functions, technically called an Application Program Interface or API, to simplify the development of graphical and interactive applications by means of a simple and consistent naming convention, unambiguous behavior, and a well-defined scope.

A. Colubri, *Processing for Android*, https://doi.org/10.1007/978-1-4842-2719-0_1

While originally implemented in Java, the Processing API is currently available in many programming languages, including Python, JavaScript, and R. However, it is the Java implementation of this API, together with some simplifications to the Java language, what defines the Processing language. Despite this distinction, throughout the book I will use the terms Processing language and API interchangeably, since in the context of Android, we will essentially be using the Java implementation of the Processing API.

In active development since 2001, the Processing language now encompasses around 300 items between not only functions, but also classes and constants (`https://processing.org/reference/`). One defining feature of this language is that it offers the possibility to create a program capable of displaying interactive graphics using very little code. As I mentioned, it also includes a number of simplifications with respect to the Java language, with the purpose of making it easier to teach to people who are not familiar with computer code. The following program exemplifies these features of the Processing language:

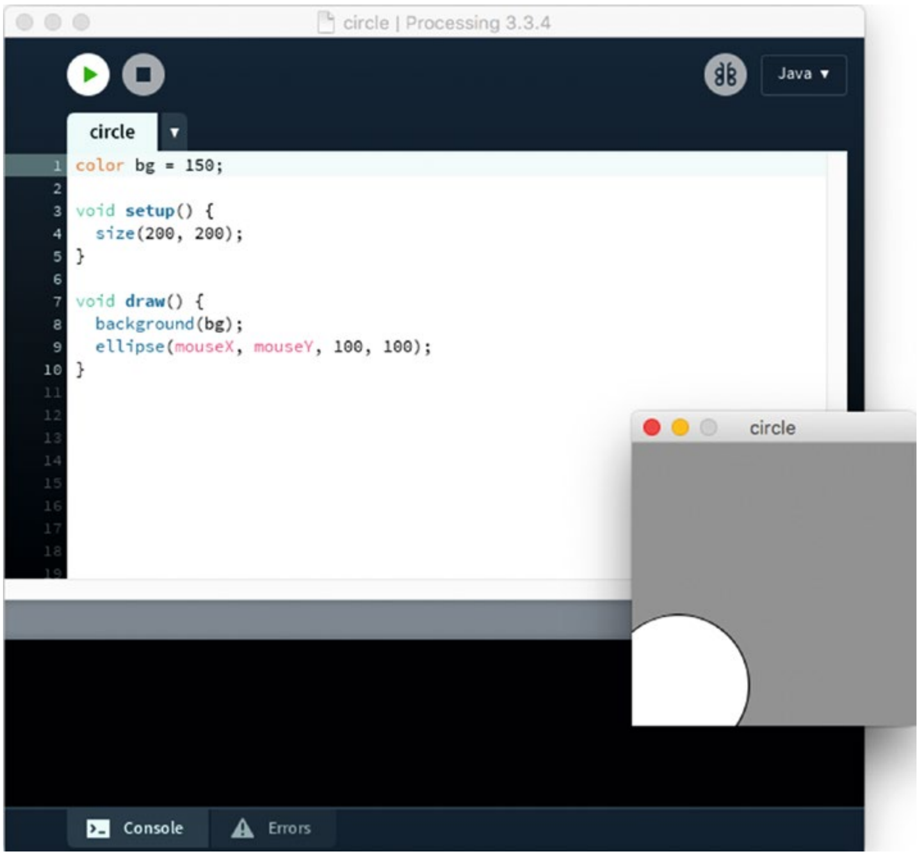
```
color bg = 150;

void setup() {
  size(200, 200);
}

void draw() {
  background(bg);
  ellipse(mouseX, mouseY, 100, 100);
}
```

The output of this program is a window of 200 by 200 pixels that contains a white circle that follows the movement of the mouse; the window has a gray background. The functions `setup()` and `draw()` are present in almost any Processing program and drive its "drawing loop." All the initialization of the program should take place in `setup()`, which is executed just once when the program starts up. The `draw()` function, which contains all the drawing instructions, is then called continuously several times per second (by default, 60 times) so that the graphical output of the program can be animated through time.

However, if you are familiar with Java, you have probably noticed that this code is not a valid Java program. For example, there is no explicit definition of a main class encapsulating all the code, nor additional instructions required in Java to initialize the "windowing toolkit" that handles the display and the user input. This program, as it is, needs to be run inside the Processing development environment, which applies a "preprocessing" step to the Processing code in order to convert it into a valid Java program. However, this transformation occurs behind the scenes, and the Processing user does not need to worry about it at all.

The Processing Development Environment

The Processing development environment (PDE) is the application that provides us with a simplified code editor to write, debug, and run Processing programs, called *sketches* (Figure 1-1). The PDE also incorporates an uncluttered user interface to handle all the sketches created with it and to add libraries and other external components that extend the core functionality of the PDE, such as p5.js, Python, or Android modes.



```
color bg = 150;

void setup() {
  size(200, 200);
}

void draw() {
  background(bg);
  ellipse(mouseX, mouseY, 100, 100);
}
```

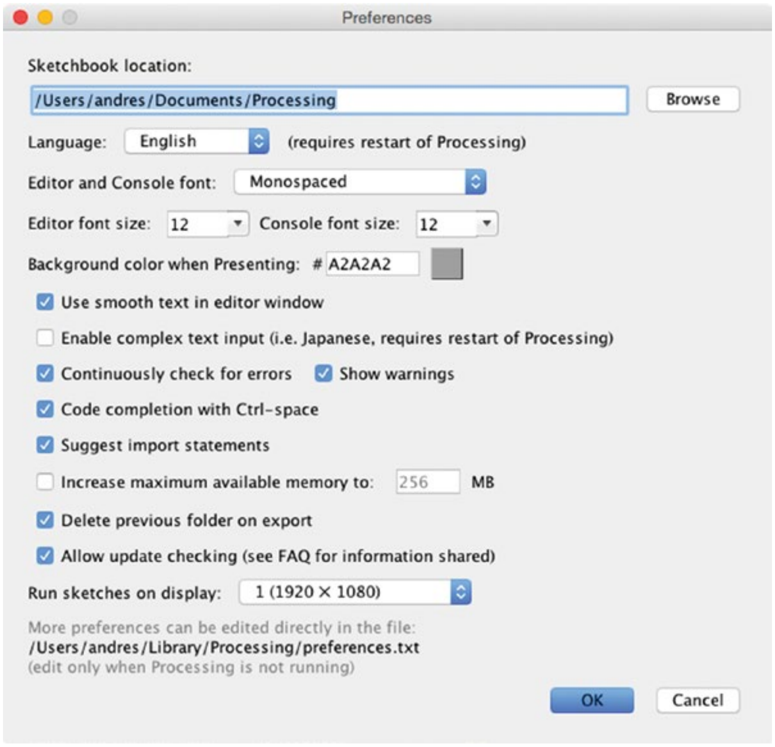
Figure 1-1. *The Processing development environment showing a running sketch in Java mode*

The simplicity and ease of use of the PDE and the Processing language are the key elements of this "code sketchbook." A stumbling block for many people wanting to start working with code is the complexity of a modern development environment, like Eclipse or IntelliJ, in terms of a lengthy installation and an overwhelming user interface. In contrast, the PDE addresses these issues by providing an easy install process and a minimal interface, while the simple structure of a Processing sketch enables users to obtain visual feedback rapidly. Processing's aim is to support an iterative development process analogous to sketching with pen and paper, where one can start with a simple idea and refine it through successive sketches.

■ **Note** The Processing API can be used outside of the PDE; for example, in a more advanced integrated development environment, or IDE, such as Eclipse, NetBeans, or IntelliJ. All of Processing's drawing, data, and interaction APIs are available when writing a program with any of these IDEs; however, the simplifications that the Processing language has with respect to Java will be lost.

We can download the latest version of Processing from the main website (`https://processing.org/download`). As pointed out in the previous paragraph, installation is fairly straightforward, only requiring the unpacking of the .zip (on Windows and Mac) or .tgz (on Linux) package that contains the PDE and all other core files. We should be able to then run the PDE without any additional steps from any location inside the `Home` or `Applications` folders.

The PDE organizes user sketches in a sketchbook folder. Each sketch is stored in a subfolder inside the sketchbook, which in turn contains one or more source-code files with the .pde extension. By default, Processing creates the sketchbook folder inside the `Documents` folder located in the user's account (for instance, `/Users/andres/Documents/Processing` on Mac), but this location can be changed by selecting the desired sketchbook folder in the Preferences window, available under the Processing menu on Mac and File menu on Windows and Linux (Figure 1-2). Notice the sketchbook location at the top.



Sketchbook location: /Users/andres/Documents/Processing

Language: English (requires restart of Processing)

Editor and Console font: Monospaced

Editor font size: 12 Console font size: 12

Background color when Presenting: # A2A2A2

☑ Use smooth text in editor window

☐ Enable complex text input (i.e. Japanese, requires restart of Processing)

☑ Continuously check for errors ☑ Show warnings

☑ Code completion with Ctrl-space

☑ Suggest import statements

☐ Increase maximum available memory to: 256 MB

☑ Delete previous folder on export

☑ Allow update checking (see FAQ for information shared)

Run sketches on display: 1 (1920 × 1080)

More preferences can be edited directly in the file:
/Users/andres/Library/Processing/preferences.txt
(edit only when Processing is not running)

Figure 1-2. *The Preferences window on Mac*

Extending Processing

As I mentioned at the beginning, the Processing project is not only the PDE or the language, but also, and very importantly, the community built around the use of the software and the goals of sharing, teaching, and inclusiveness. Thanks to Processing's open nature and modular architecture, many people have contributed improvements and extensions to the "core" software. These contributions fall within one of the following four categories:

Libraries: Modules (comprising one or more Java code files built into a jar package, and additional documentation and example files) that make it possible to access new functionality in the sketches. E.g., OpenCV library for computer vision (which is PC/Mac only), or Ketai for Android sensors (covered in Chapters 7 and 8).

Programming Modes: Alternative code editors and related PDE customizations that allow the use of an entire different language within the PDE. E.g., Android mode. We will see in the next sections of this chapter how to install Android mode.

Tools: Applications that can only run from Processing and provide specific functionality to aid in writing code, debugging, and testing the sketch. E.g., the color picker (discussed in Chapter 2).

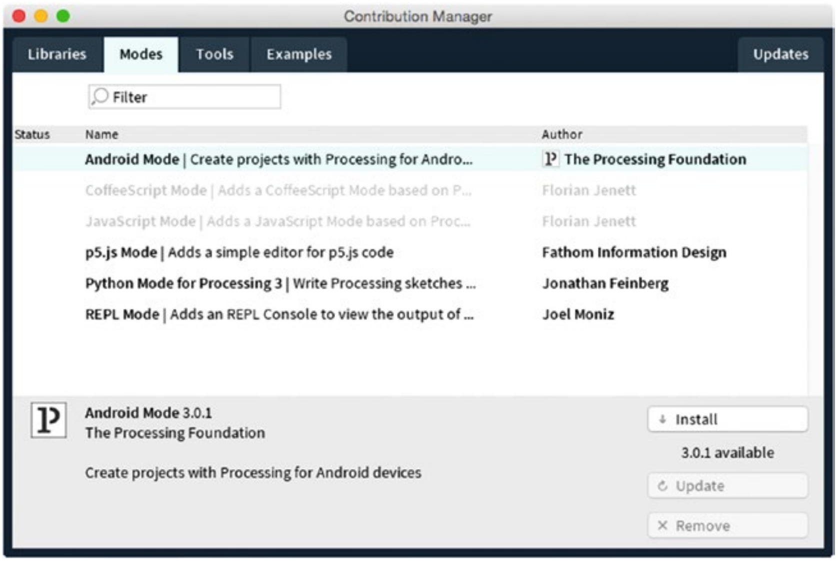
Examples: Packages of contributed code sketches that can be used as learning material or reference. E.g., the sketches from the book *Learning Processing* by Daniel Shiffman (`http://learningprocessing.com/`).

The extension of Processing through contributed libraries, modes, tools, and examples has enabled its growth into application domains that were not part of the original software, such as mobile apps, computer vision, and physical computing, while keeping the core functionality simple and accessible for new programmers.

The Contribution Manager

By default, Processing includes one default mode, Java, where we can write and run sketches on Windows, Mac, and Linux computers using the Java implementation of the Processing language. Processing also bundles several "core" libraries, some of which are OpenGL (for drawing hardware-accelerated 2D and 3D scenes), pdf (to export graphics as pdf files), and data (which allows the handling of data files in formats such as CSV and JSON).

To install additional contributions, we can use the Contribution Manager (CM), which makes the process seamless. A screenshot of the CM is shown in Figure 1-3. The CM has five tabs, the first four for each type of contribution—libraries, modes, tools, and examples—and the fifth for updates. All the contributions that are registered by their authors in a central repository are accessible through the CM and can also be updated through the CM when new versions become available.



Status	Name	Author
	Android Mode | Create projects with Processing for Andro...	The Processing Foundation
	CoffeeScript Mode | Adds a CoffeeScript Mode based on P...	Florian Jenett
	JavaScript Mode | Adds a JavaScript Mode based on Proc...	Florian Jenett
	p5.js Mode | Adds a simple editor for p5.js code	Fathom Information Design
	Python Mode for Processing 3 | Write Processing sketches ...	Jonathan Feinberg
	REPL Mode | Adds an REPL Console to view the output of ...	Joel Moniz

Android Mode 3.0.1
The Processing Foundation

Create projects with Processing for Android devices

3.0.1 available

Figure 1-3. *The Contribution Manager in Processing, showing the Modes tab*

■ **Note** Contributions that were not registered by their authors and hence are not available through the CM, can still be installed manually. We would need to download the package containing the library, mode, tool, or examples, typically in zip format, and extract it into the sketchbook folder. There are separate subfolders for libraries, modes, tools, and examples. See `https://processing.org/reference/libraries/` for more info.

Processing for Android

Processing for Android, not unlike the Processing software itself, is several things. Primarily, it is a community effort that started in 2009 with the purpose of supporting the development of Android apps using Processing, as well as translating some of the concepts of the project to the context of mobile apps: iterative sketching, simplicity, and accessibility.

From a software point of view, Processing for Android is composed of the processing-android library and the custom PDE programming mode itself. The library is the package that contains all the functions of the Processing API, but reimplemented for the Android platform. Android mode provides a customized version of the PDE that allows us to write Processing code and run it on an Android device or in the emulator. Android mode includes the processing-android library, which we need for our Processing code to run without errors. However, these distinctions are not critical at this point, since Processing will let us install and use Android mode without our having to worry about the processing-android library. This library would become more important for those among you who may be planning to use Processing for Android in more advanced applications.

■ **Note** The processing-android library can be imported from an IDE like Android Studio, allowing the use of all the Processing functionality in a regular Android app. This advanced use is covered in Appendix A.

Installing the Android mode

Once we have installed Processing on our computer, we should be able to open the PDE by running the Processing application, and then we can install the most recent release of Android mode through the CM. The mode also requires the Android Software Development Kit (SDK) in order to work. The Android SDK is the set of libraries, tools, documentation, and other supporting files provided by Google to develop and debug Android apps. So, to install Android mode and, if needed, the SDK, follow these steps:

1. Open the CM by clicking the “Add Mode...” option that appears in the drop-down menu in the upper-right corner of the PDE (Figure 1-4).

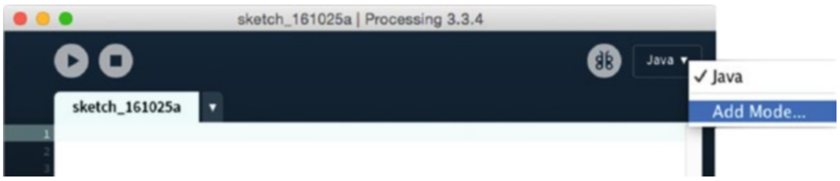


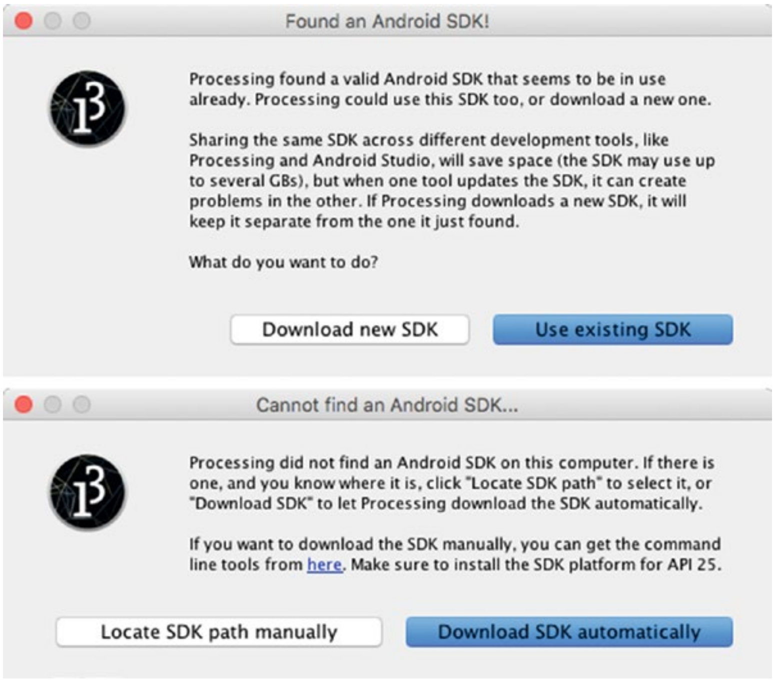
***Figure 1-4.** Opening the Contribution Manager to add a new mode*

2. Select the entry for Android mode in the Modes tab, then click the Install button.

3. After installation is complete, close the CM and switch to Android mode using the same drop-down menu from Figure 1-4.

   If a valid SDK is detected on the computer, Processing will ask if we want to use it or download a new one (Figure 1-5). Because the SDK is very large (up to several GBs), it can be a good idea to use the one that is already installed to save disk space. However, if that SDK is also used by another development tool, such as Android Studio, it may get updated outside Processing, which may lead to incompatibilities with the mode.

   If no valid Android SDK is detected, Processing will ask to either manually locate an SDK or automatically download one (Figure 1-5).



Processing found a valid Android SDK that seems to be in use already. Processing could use this SDK too, or download a new one.

Sharing the same SDK across different development tools, like Processing and Android Studio, will save space (the SDK may use up to several GBs), but when one tool updates the SDK, it can create problems in the other. If Processing downloads a new SDK, it will keep it separate from the one it just found.

What do you want to do?

Processing did not find an Android SDK on this computer. If there is one, and you know where it is, click "Locate SDK path" to select it, or "Download SDK" to let Processing download the SDK automatically.

If you want to download the SDK manually, you can get the command line tools from here. Make sure to install the SDK platform for API 25.

Figure 1-5. *Choosing between using an existing SDK or downloading a new one automatically (top), and between locating an SDK manually or downloading one automatically (bottom)*

■ **Note** Version 4.0 of Android mode requires Android version 8.0 (Oreo), corresponding to API level 26 (`https://source.android.com/source/build-numbers`). The mode's automatic SDK download will retrieve this version from the Google servers.

Pre-releases of Android mode, as well as older, unsupported versions, are no longer available through the CM, but rather are deposited on the GitHub releases page (`https://github.com/processing/processing-android/releases`) and can be installed manually by downloading the corresponding file and extracting it into the `Modes` folder in Processing's sketchbook.

Interface of Android Mode

The editor in Android mode is very similar to that of Java mode. The toolbar contains the Play and Stop buttons to launch a sketch and to stop its execution (on the device or in the emulator). Code autocompletion in the editor is available as well. However, version 4.0 of Android mode does not offer an integrated debugger. The main menu contains a number