GUI Programming

Toolkits

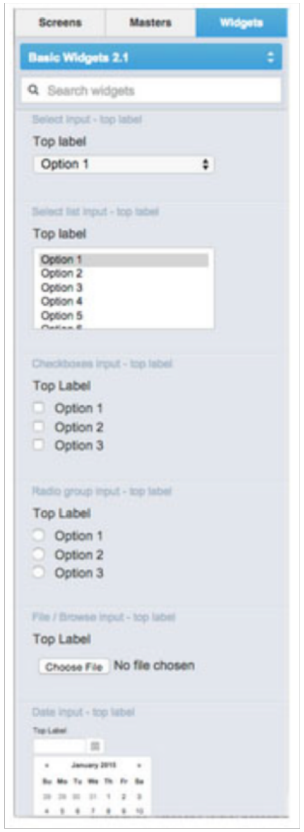
Imperative, declarative models

Programming languages & toolkits

GUI Toolkits

- Operating systems (and windowing systems) include basic capabilities for input, output, graphics.
  - Low-level capabilities are exposed to developers.
  - Suitable for building libraries, but not applications.
- We often need a higher level of abstraction for building applications
  - Classes that hide the underlying implementation
  - Wrappers to make the underlying functionality compatible with a particular programming language – language independence is helpful!
- A **toolkit** is set of reusable classes or components for building user-interfaces.
  - Often referred to as a **widget toolkit**, although that name overlooks a lot of other functionality that they provide.
  - Can be provided by OS vendors, programming language vendors, other sources.

What functionality does a toolkit provide?



Toolkit are delivered as libraries that provide abstractions for:

- **Graphics**
  - The ability to draw graphics primitives (e.g. simple shapes, images).
  - Possibly video and animation playback.
- **Collections of widgets**
  - Set of common UI components that can be reused in any application.

This functionality is often integrated with system services.

- Audio
  - The ability to integrate audio into your UI, to play back (and possibly record).
- Event handling
  - Events that are generated through hardware (e.g. mouse movement, key presses, timer ticks), and that can be passed to the user-interface.
- System services
  - Abstractions for any other functionality that may be needed (e.g. camera access, sensor access on a phone, network access).

Types of Toolkits

Two main categories, depending on how tightly the libraries are integrated with the underlying operating system:

1. **Low-level toolkits**
   - build into, or tightly integrated with, the underlying operating system (also called “native” or “heavyweight”).
   - e.g. Win32 on Windows, Cocoa on macOS, Xlib on Unix.
2. **High-level toolkits**
   - sit “above” the operating system, with no tight integration (“third-party” since they are not provided by OS vendor).
   - also called “lightweight” since they’re not tightly coupled.
   - may or may not have a “native” look-and-feel for a platform.
   - e.g. Qt, Gtk+, wxWidgets, Swing, MFC, WTL, (Cocoa).

UI Development Fragmentation

Native: platform-specific language + toolkit combination

- Windows: C++/UWP, C#/WPF, C#/Windows Forms, C++/Win32
- Mac: Objective-C/Cocoa, Swift/Cocoa
- Android: Java, Kotlin
- iOS: Swift, Kotlin

Cross-Platform: language, toolkit can be built and deployed to multiple platforms.

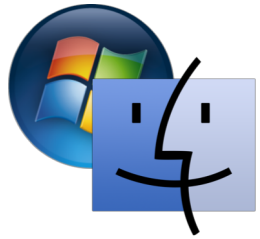
- Java/Swing, JavaFX (i.e. using JVM runtime on each platform)
- C++, Python using Qt (i.e. many supported platforms and language bindings)
- Kotlin: cross-platform language, binds to native toolkits e.g. JavaFX, iOS
- Web-native: desktop apps with web and native widgets e.g. React Native, Electron

Web Cross-Browser: build using common technologies, deployed to browsers

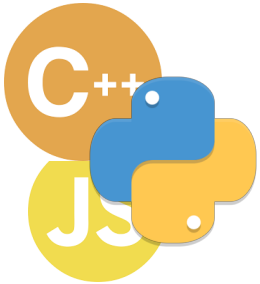
- JS/HTML/CSS, JQuery/Node.js, React

Choosing Programming Language + Toolkit

You often have to consider multiple dimensions when deciding which technology stack to use or your application.



- **Platform**: which operating systems do you wish to support (e.g. Windows, Mac, Linux, iOS, Android)?
- **Programming Language**: which programming language for building your application?



- **Toolkit**: which toolkit works on platform + language, and provides the capabilities that you need.

Vendors (Microsoft, Apple) are typically focused on promoting their own platform, less interested in other platforms.

- They typically provide libraries and support for their “favored” programming language (e.g. Swift on macOS, Java on Android).
- Cross-platform tools are rare (“holy grail” of UI development).

Current Languages + Platforms

The (incomplete) mess of current generation UI languages, toolkits and frameworks.

- "Native" support is intrinsically tied to a vendor language/platform.
- "Cross platform" (Java FX, .NET, Qt, JS/React) is uncommon on desktop.
- Web as a platform addresses cross-platform, but sandboxing in a browser is limiting

Vendors commonly build using different code-base, tools on different platforms.

- e.g. Java for Android, Swift for iOS

					Windows	macOS	Linux	Android	iOS	Web
				C++	GTK					
				C++	Qt					
		Kotlin	Java		Java FX (JVM)					
		Kotlin			JavaFX					
				C++	Win32					
	Swift					Cocoa				
		Kotlin						UITK		
	Swift								UIKit	
			Java		Swing (JVM)					
JS								Flutter		
JS										React

How to build a User Interface?

After you've picked a set of technologies, how do you use them to write code for interactive applications?

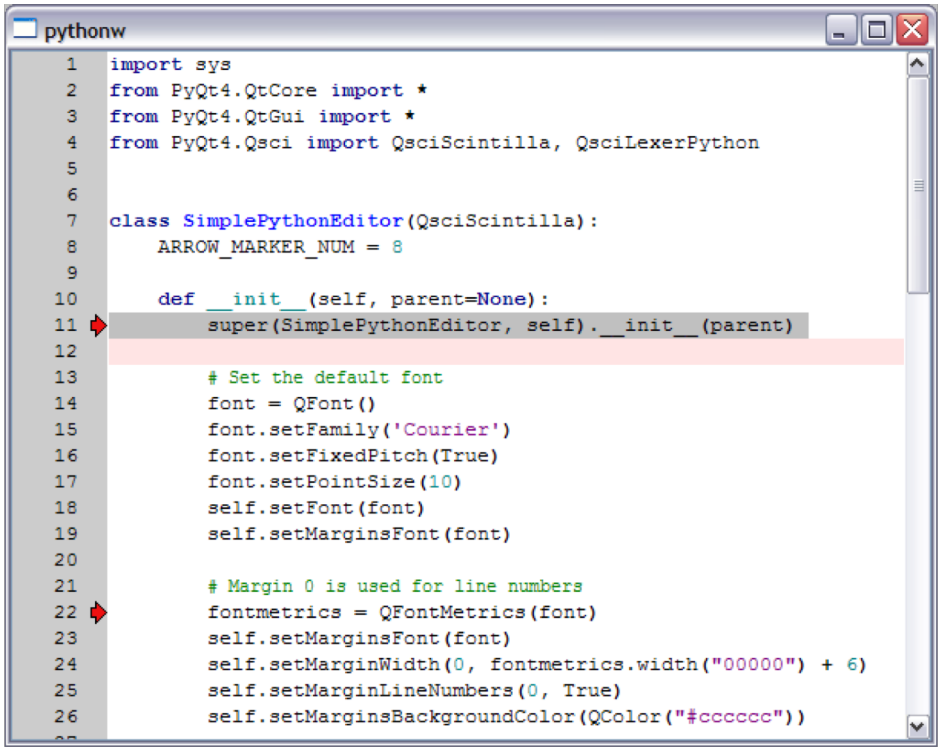
Two main approaches:

1. **Imperative**: the user-interface is manually constructed in code (also called *procedural*).
2. **Direct (Drag-and-Drop):** the user-interface is constructed with some GUI builder or tool, and there is a less clear separation between user-interface and code than declarative.
3. **Declarative**: the user-interface is described in some type of human-readable layout file. There is a separation between the layout and the code to control it.

Imperative Programming

Code is used to manually construct the view. Everything is manually controlled.

Virtually every programming environment offers some ability to do this (e.g. Java/Swing, C++/Qt, Python, Javascript/HTML).

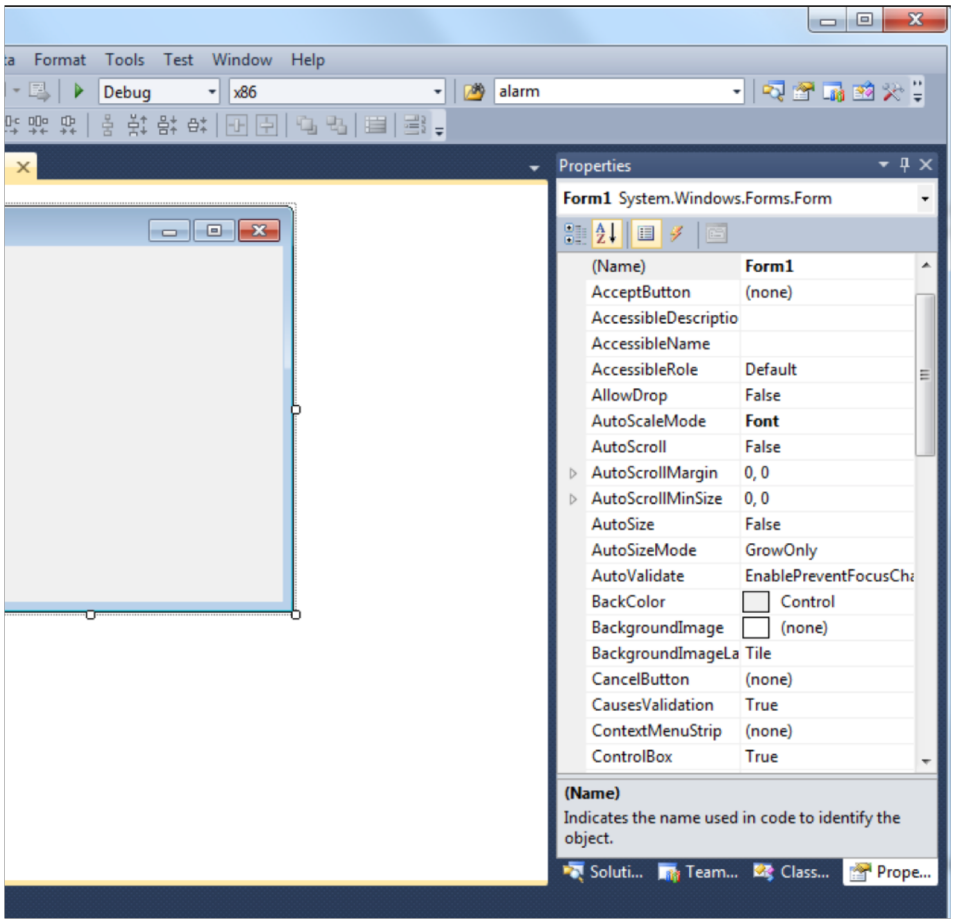


Python w. Qt toolkit

- Benefits:
  - You have complete control over how objects are created and managed.
- Drawbacks
  - Requires programming knowledge to create or change.
  - It's tedious to build a UI in this fashion!

Direct (Drag-and-Drop)

- The programmer uses a tool to build the UI and associates code with elements in the UI.
- Unlike declarative, the tool does NOT generate XML (or some declarative language). Format is generally non-readable or **binary**.
- Used to be common, but the lack of visibility into the GUI code limits it. E.g. Visual Basic, Delphi.

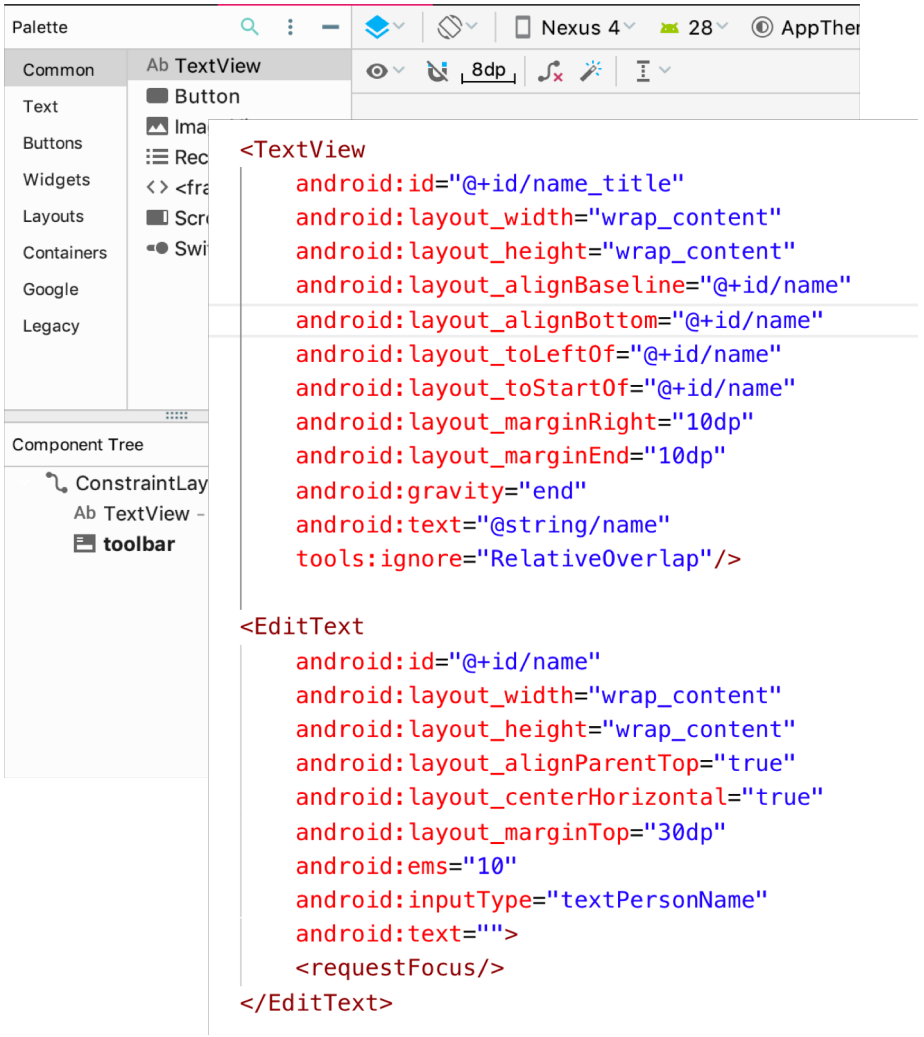


Visual Basic GUI Builder

- Benefits:
  - Non-programmers can use it.
  - Supports GUI builders (drag-drop).
- Drawbacks:
  - Impossible to debug.
  - Requires tools to generate; forever tied to that tool to support modifications to the UI.
  - Binary -> can't diff the UI code!

Declarative Programming

- The programmer (or designer) uses some markup language to describe how the View should be constructed.
  - e.g. XML
- At runtime, code “automatically” loads in this markup declaration and uses it to instantiate code.
  - e.g. Layout files on Android, Java FX.
- Benefits:
  - Non-programmers can work with it.
  - Human-readable, and “easy” to read.
  - Supports GUI builders (drag-drop)
- Drawbacks:
  - Syntax can be messy
  - Practically requires tools to generate.



Android GUI builder and Layout

Course Goals

- Build for desktop + mobile
- Cross-platform
  - Able to code + build + test on your own machine
  - Supports: Windows, Mac, Linux
- Robust language + framework
  - Support a range of applications, incl. graphics
  - Extensive libraries/frameworks, good tools support
  - Not going to be replaced anytime soon (Angular? React?)
- Imperative + Declarative
- Demonstrates best-practices
  - Reflect how we want to teach UI design and development

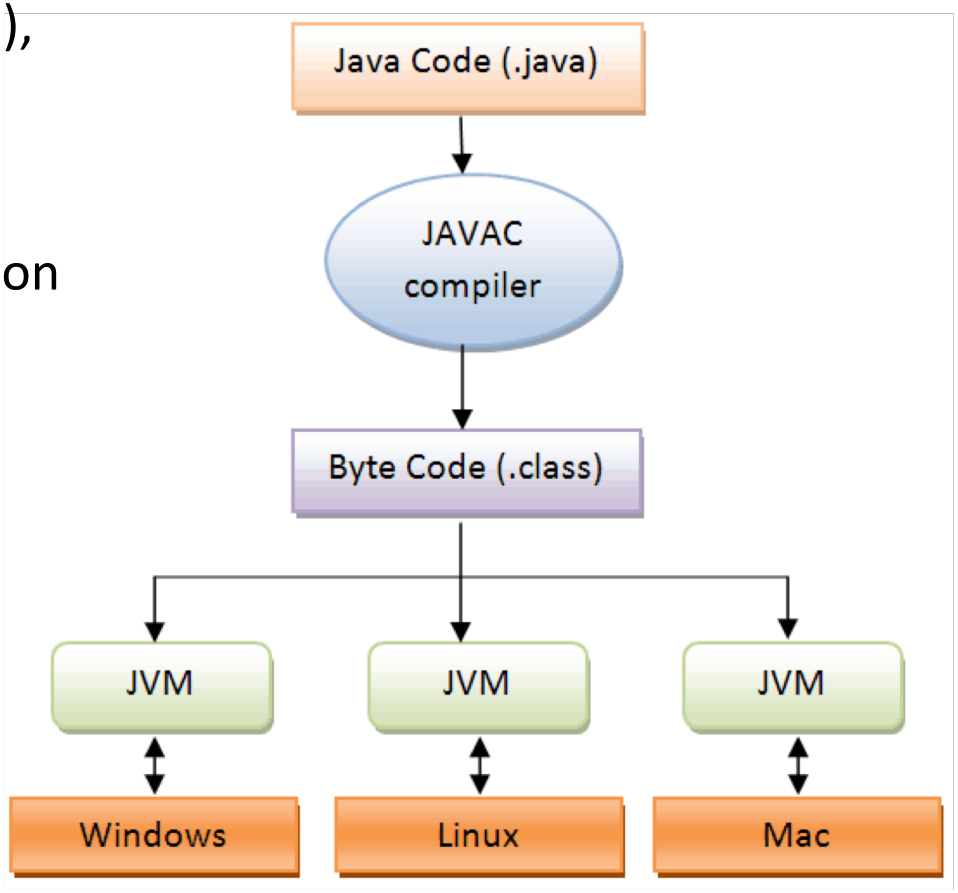
Java

- Open Source
  - Designed by James Gosling
  - released by Sun Microsystems in 1995
  - Made open source under GNU GPL in 2007
  - Sun and Java acquired by Oracle in 2010

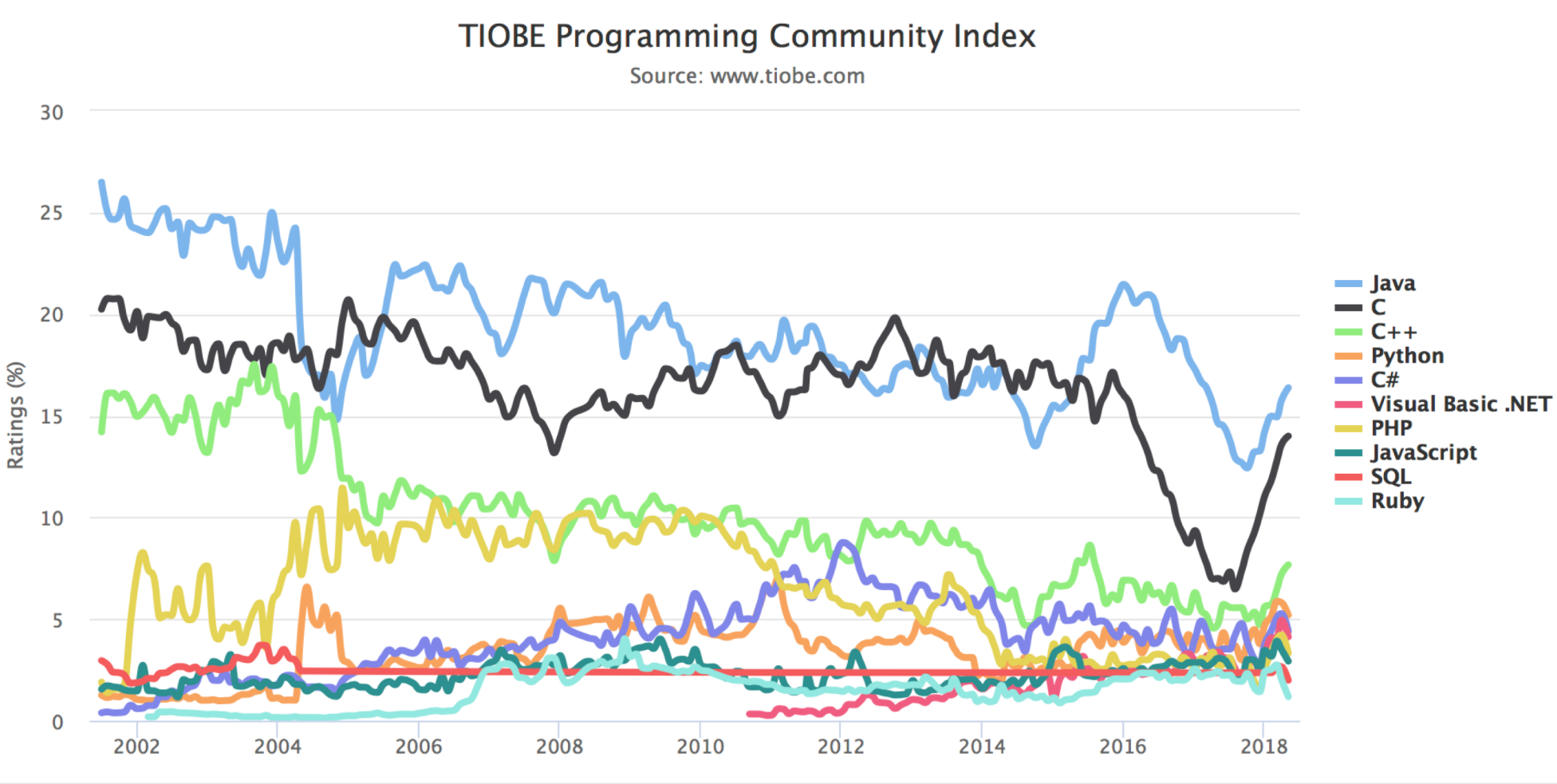
- Class-based, object-oriented design
  - C++ syntax
  - Strongly typed
  - Manages memory, garbage collection
- Extensive class libraries – robust
  - 2 main UI libraries (Swing + JavaFX)
- Primary language for Android – mobile + desktop
- Java Virtual Machine (JVM) – cross platform
- IntelliJ, Android Studio – imperative + declarative + GUI builder

Portability through Virtualization

- Java compiles to bytecode (.class file), which is executed by a Java Virtual Machine (JVM)
- Just-in-Time (JIT) bytecode compilation can give near-native performance.
- JVM can execute other languages that specifically target it (e.g. Jython, Scala, Kotlin)
- Java runs everywhere (Mac, Linux, Windows, Raspberry Pi, Arduino, car OS, toasters…)

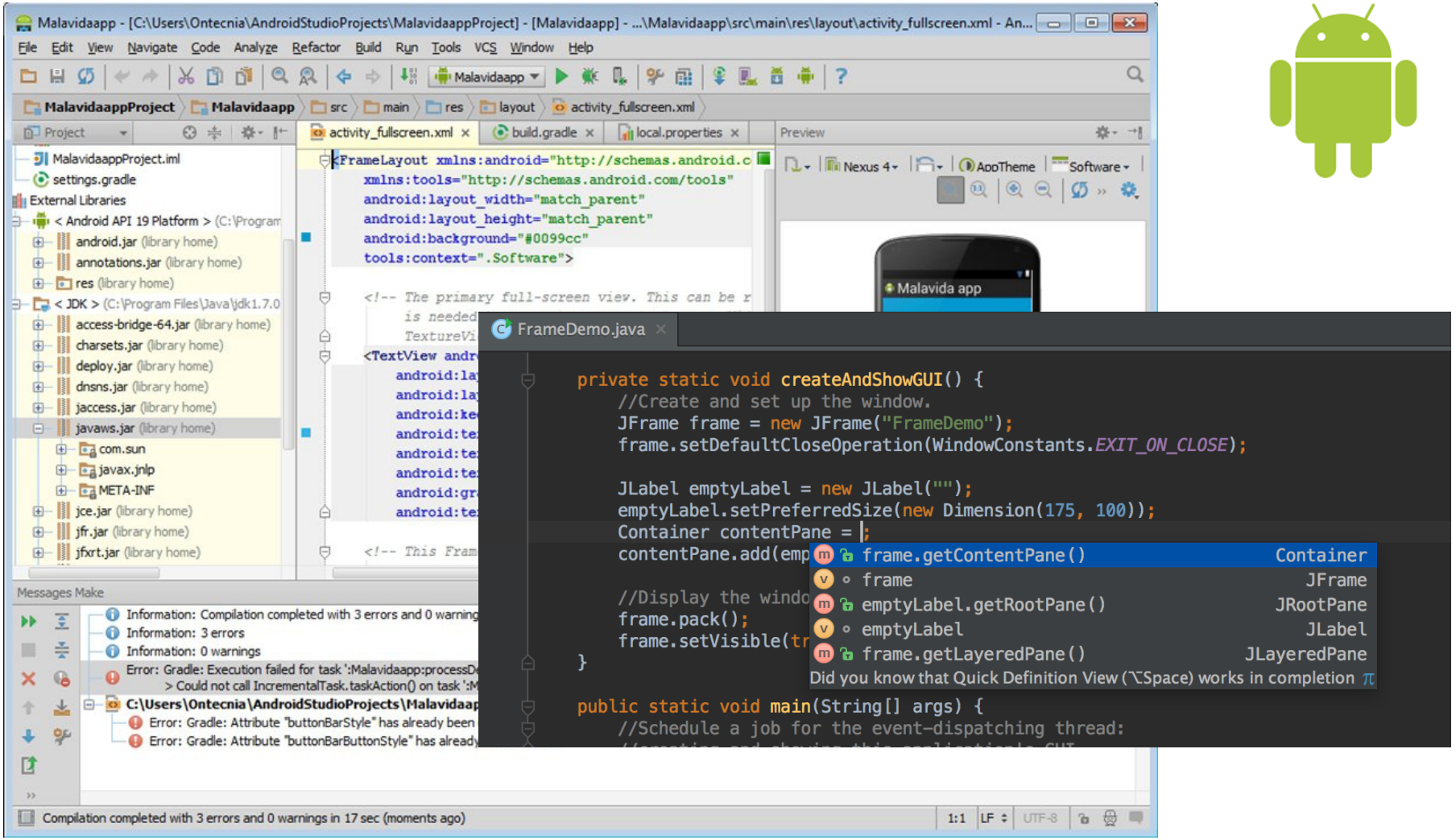


http://viralpatel.net/blogs/java-virtual-machine-an-inside-story/



May 2018	May 2017	Change	Programming Language	Ratings	Change
1	1		Java	16.380%	+1.74%
2	2		C	14.000%	+7.00%
3	3		C++	7.668%	+2.92%
4	4		Python	5.192%	+1.64%
5	5		C#	4.402%	+0.95%

Tools Support



- https://developer.android.com
- https://www.oracle.com/technetwork/java/javase
- https://www.jetbrains.com/student/