# Tools for GUI Programming

Toolkits

Imperative, declarative models

Programming languages & toolkits

# GUI Toolkits

- Operating systems (and windowing systems) include basic capabilities for **input**, **output**, and window management (*see previous lecture*).
- We often want a higher level of abstraction, or more capabilities than the base OS provides.
  - Classes that hide the underlying implementation
  - Wrappers to make the underlying functionality compatible with a particular programming language
- A **toolkit** is set of reusable classes or components for building applications.
  - Can be provided by OS vendors, programming language vendors, other sources.
  - Usually specific to a programming language + platform

## What functionality does a toolkit provide?

Toolkit are delivered as libraries that provide useful capabilities for handling:

- **Output**
  - Drawing graphics primitives (e.g. shapes, images).
  - Animating shapes, windows.
  - Video and animation playback.
- **Input**
  - Standard input (e.g. keyboard, mouse)
  - Camera, sensors etc.

Also, they often provide UI components or widgets that can be reused in any application (which we will discuss soon).


Widgets (Qt)

# Types of Toolkits

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, JavaFX, MFC, WTL.

# Examples of Toolkits

**Desktop**: platform-specific language + toolkit combination (“low-level”)

- Windows: UWP, WPF, Windows Forms, Win32 -- using C++ or C#
- Mac: Cocoa -- Swift or Objective C
- Linux: Gtk -- C ++

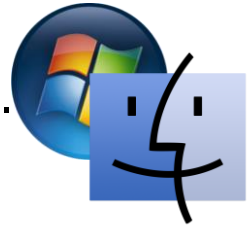
**Cross-Platform**: toolkit can be deployed to multiple platforms (“high-level”).

- Java Swing, JavaFX -- Java or Kotlin
- wxWidgets – C++, Python, Perl
- Qt -- C++, Java, Python
- Tk -- Tk and Python

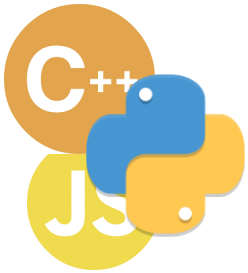
https://en.wikipedia.org/wiki/List_of_widget_toolkits

# Choosing Programming Language + Toolkit

You often have to consider multiple dimensions when deciding which technology stack to use for 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).

# How to build a User Interface?

After you've picked a set of technologies (programming language + toolkit), 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*.
   - The "traditional" approach, which treats the UI as a library to manipulate in code.
2. **Declarative**: separation between the layout and the code.
   - Often layout is described in a human-readable layout file. The toolkit has support for loading the layout file and applying it.
   - Layout can be compiled into a resource file (e.g. Objective C, NIB), or dynamically interpreted (e.g. HTML/CSS).

# 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).

pythonw

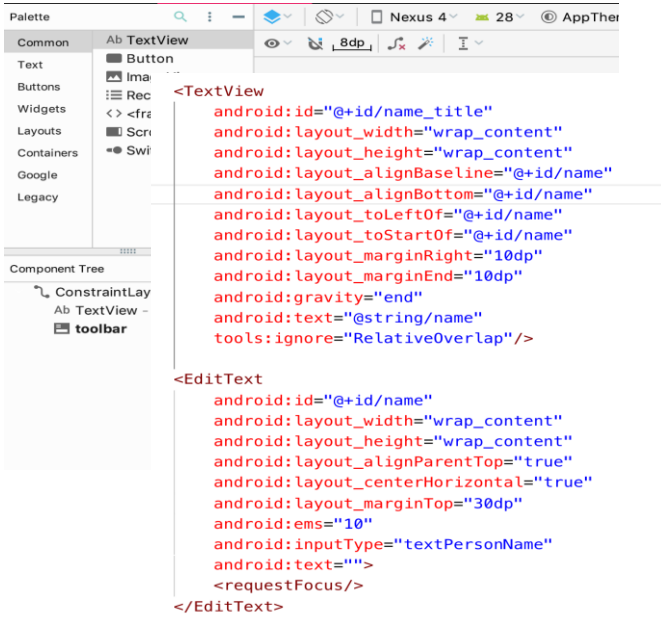
```
1  import sys
2  from PyQt4.QtCore import *
3  from PyQt4.QtGui import *
4  from PyQt4.Qsci import QsciScintilla, QsciLexerPython
5
6
7  class SimplePythonEditor(QsciScintilla):
8      ARROW_MARKER_NUM = 8
9
10     def __init__(self, parent=None):
11         super(SimplePythonEditor, self).__init__(parent)
12
13         # Set the default font
14         font = QFont()
15         font.setFamily('Courier')
16         font.setFixedPitch(True)
17         font.setPointSize(10)
18         self.setFont(font)
19         self.setMarginsFont(font)
20
21         # Margin 0 is used for line numbers
22         fontmetrics = QFontMetrics(font)
23         self.setMarginsFont(font)
24         self.setMarginWidth(0, fontmetrics.width("00000") + 6)
25         self.setMarginLineNumbers(0, True)
26         self.setMarginsBackgroundColor(QColor("#cccccc"))
```

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!

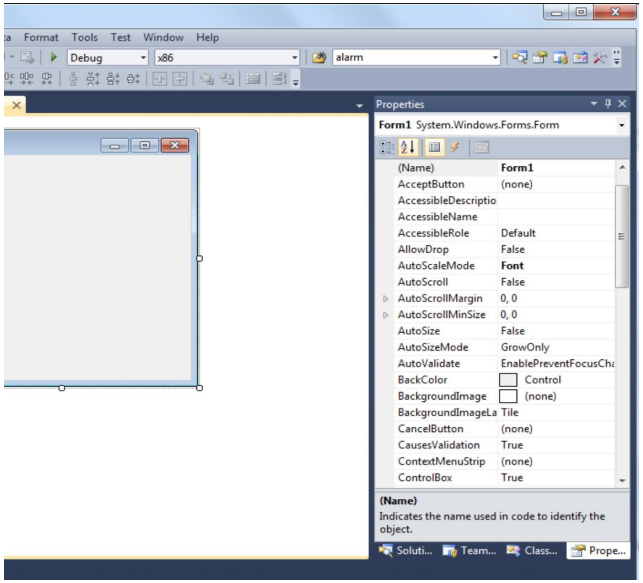
# Declarative (Markup)

- 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.




Android GUI builder and Layout

- 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.

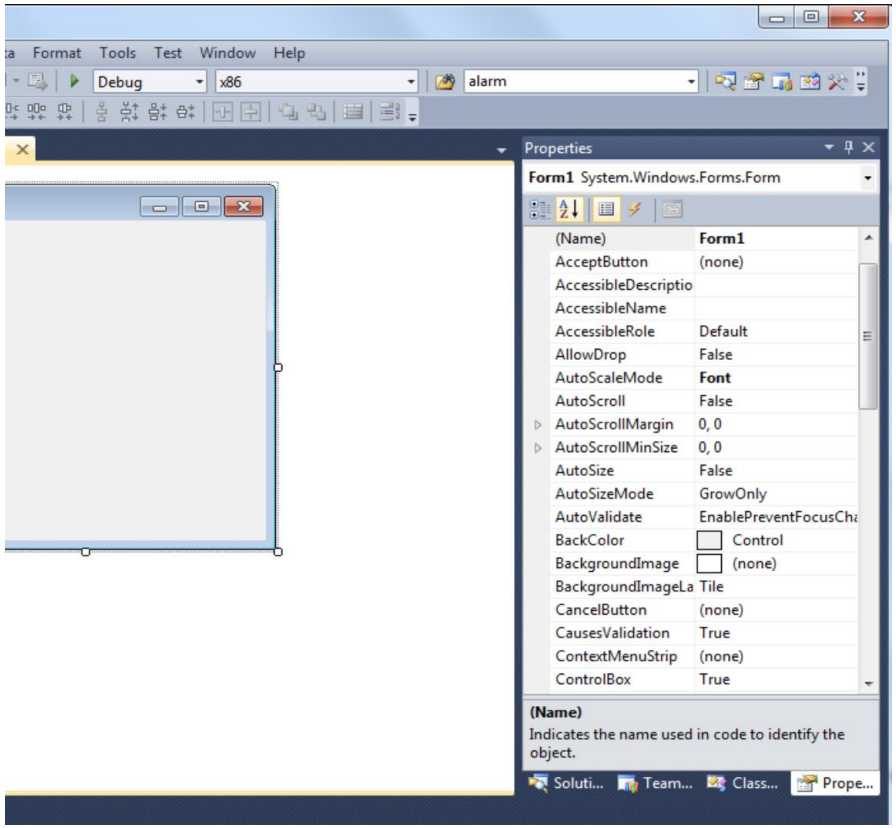



Visual Basic GUI Builder

# Declarative (Drag-and-Drop)

- The programmer uses a tool to build the UI and associates code with elements in the UI.
- Format is generally not human-readable aka **binary**. But not for every toolkit (e.g. JavaFx)
- Used to be common, but the lack of visibility into the GUI code limits it. E.g. Visual Basic, Delphi.

- Benefits:
  - Non-programmers can use it.
  - Supports GUI builders.
- Drawbacks:
  - Requires proprietary tools to generate or modify to the UI.
  - Binary -> can't diff the UI code!




Visual Basic GUI Builder

# Course Goals: Why Java?

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

# Java GUI Toolkits

| Toolkit | Description |
| --- | --- |
| AWT | • Low-level or “heavyweight”, used platform-specific widgets.<br>• AWT applications were limited to common-functionality that existed on all platforms. |
| **Swing** | • High-level or “lightweight”, full widget implementation using *imperative* model.<br>• Commonly used and deployed cross-platform. |
| Standard Window Toolkit / SWT | • ”Heavyweight” hybrid model: native, and tied to specific platform (Windows).<br>• Used in Eclipse. |
| **Java FX** | • Intended for rich desktop + mobile apps.<br>• *Declarative* programming model, designed to replace Swing. |