Lecture 1:

Program Execution in the 21st Century

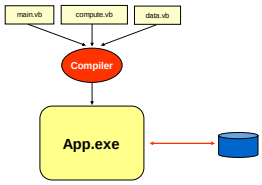
Objectives

"Modern computer applications are no longer self-contained, self-sufficient programs. Instead, they often require complex run-time environments that must be present in order for the program to run. Java requires the JVM (Java Virtual Machine), Microsoft .NET requires the CLR (Common Language Runtime)…"

- **Application designs**
- **Managed execution**
- **.NET execution model**

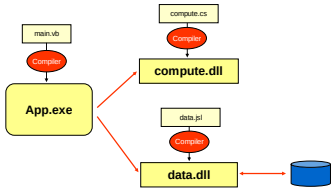
Monolithic applications

- **A monolithic app is where all source code compiled into .EXE**
  - less and less common today…



Component-based applications

- **A component-based app is built from many pieces**
  - a component is a logically-related set of source code files
  - a more common design today…

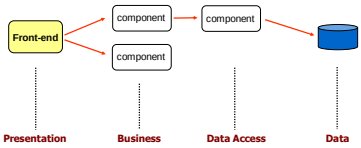


Why component-based?

- **Many motivations:**
  - team programming
  - multi-language development (I like VB, you like C#)
  - code reuse (e.g. across different .EXEs)
  - independent updating (can update just component X)

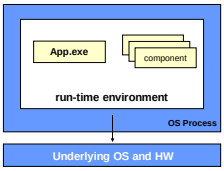
Example: n-tier design

- **Many applications are designed with N levels or "tiers"**
  - good separation of concerns
  - enables reuse of back-end tiers across varying FEs



Idea

- **Modern software executes within a run-time environment**
- **Why?**
  - portable execution: software can run anywhere environment is
  - safer execution: environment prevents unauthorized access

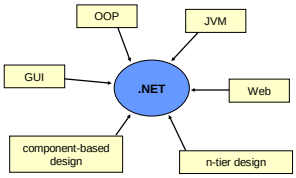


Disadvantages?

- **Run-time environment must be installed to run program**
- **Program may run slower**

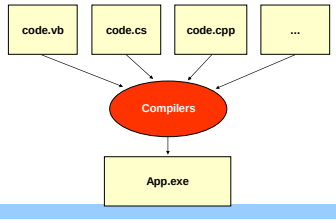
Influences

- **.NET is the result of many influences…**



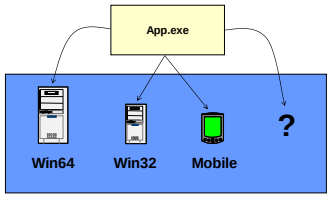
.NET is multi-language

- **.NET languages: VB, C# (C-sharp), C++, etc.**
- **Additional languages available from 3rd parties, academia, …**



.NET is cross-platform

- **Compiled .NET apps run on any *supported* platform**
  - i.e. platforms with appropriate run-time environment



How is cross-platform achieved?

- **Cross-platform execution realized in two ways:**
  1. apps are written against *Framework Class Library* (FCL), not underlying OS
  2. compilers generate generic assembly language which must be executed by the *Common Language Runtime* (CLR)
- **These two pieces represent the *.NET Framework***
  - i.e. FCL + CLR = *.NET Framework*

(1) FCL

- **Framework Class Library**
  - 1000's of predefined classes
  - networking, database access, XML processing, GUI, Web, etc.
  - common subset available across all platforms & languages

- **Goal?**
  - FCL approaching a portable operating system

(2) CLR

- **Common Language Runtime is .NET's run-time environment**