301AA - Advanced Programming

Lecturer: Andrea Corradini
andrea@di.unipi.it
http://pages.di.unipi.it/corradini/

Course pages:
http://pages.di.unipi.it/corradini/Didattica/AP-19/

AP-02: Motivations and Introduction
Software is Everywhere
Programming in the 21 century

- Software as complex as ever
- Command line interface not enough
- Data comes from multiple sources: structured (DB) and unstructured
- Single computer not enough
- Software development is a group activity
- Deployment on Web or mobile devices
Complexity Prompts for Innovation

- Object-Oriented Programming allows ever larger applications to be built
- But limited support for reuse
- OS + libraries not enough
- Reusable components are needed
- Multi-tier applications development increases the choices on how to build applications
Key Ingredients for Complex Software

• **Advanced features** extending programming languages
• **Component models** to ensure reusability
• **Frameworks** to support efficient development of (component based) applications
• **Execution environments** providing runtime support for ever dynamic software systems
The Software Architect

• A new role is needed: **Software Architect**
• to create, define or choose an **application framework**
• to create the component design according to a **component model**
• to structure a complex application into pieces
• to understand the interactions and dependencies among components
• to select the **execution environment / platform** based on cost/performance criteria
• to organize and supervise the development process
Course Objectives

• Understand programming language technology:
  – Execution Models
  – Run-time systems
• Analyze programming metaphors:
  – Objects
  – Components
  – Patterns
• Learn advanced programming techniques
• Present state-of-the-art frameworks incorporating these techniques
• Practice with all these concepts through small projects
Course Syllabus

• Programming Languages and Abstract Machines
• Run Time Systems and the JVM
• Component-based Programming
• Software Frameworks and Inversion of Control
• Polymorphism and Generic Programming
• Functional programming and Haskell
• Scripting Languages and Python
Programming Languages and Abstract Machines

- Syntax, Semantics and Pragmatics of PLs
- Programming languages and Abstract Machines
- Interpretation vs. Compilation vs. Mixed
- Examples of Virtual Machines
- Examples of Compilation Schemes
Run-Time Systems and the JVM

• RTSs provide a Virtual Execution Environment interfacing a program in execution with the OS.
• They support, among others:
  – Memory Management, Thread Management
  – Exception Handling and Security
  – AOT and JIT Compilation
  – Dynamic Link/Load
  – Debugging Support and Reflection
  – Verification
• A concrete example: the Java Virtual Machine
Component-based Programming

- Component models and frameworks, an Introduction
- Examples of component-based frameworks:
  - JavaBeans and NetBeans
  - Spring and Spring Beans
  - COM
  - CLR and .NET
  - OSGi and Eclipse
  - Hadoop Map/Reduce
Software Frameworks and Inversion of Control

• **Software Framework**: A collection of common code providing generic functionality that can be selectively overridden or specialized by user code providing specific functionality

• **Application Framework**: A software framework used to implement the standard structure of an application for a specific development environment
Framework Features

• Frameworks, like *software libraries*, provide *reusable abstractions* of code wrapped in a well-defined API

• But: **Inversion of control**
  – unlike in libraries, the overall program's flow of control is not dictated by the caller, but by the framework

• Helps solving recurring design problems

• Drives solution
  – Provides a default behavior
  – Dictates how to fill-in-the-blanks

• Non-modifiable framework code
  – Extensibility: usually by selective overriding
OO Software Framework

• Object-oriented programming frameworks consist of a set of abstract classes
• An application can be built simply inheriting from pre-existing classes in the framework
• Instantiation of a framework consists of composing and subclassing the existing classes
Examples of Frameworks

• General software frameworks
  – **.NET** – Windows platform. Provides language interoperability
  – **Android SDK** – Supports development of apps in Java (but does not use a JVM!)
  – **Spring** – Cross-platform, for Java applications
  – **Cocoa** – Apple’s native OO API for macOS. Includes C standard library and the Objective-C runtime.
  – **Eclipse** – Cross-platform, easily extensible IDE with plugins
Examples of Frameworks

• Frameworks for Application with GUI
  – **Gnome** – Written in C; mainly for Linux
  – **Qt** - Cross-platform; written in C++
Examples of Frameworks

• Web Application Frameworks [based on Model-View-Controller design pattern]
  – **ASP.NET** by Microsoft for web sites, web applications and web services
  – **GWT** - Google Web Toolkit (GWT)
  – **Rails** - Written in Ruby - Provides default structures for databases, web services and web pages.
Examples of Frameworks

• Concurrency
  – **Hadoop Map/Reduce** - software framework for applications which process big amounts of data in-parallel on large clusters (thousands of nodes) in a fault-tolerant manner.
    • **Map:** Takes input data and converts it into a set of tuples (key/value pairs).
    • **Reduce:** Takes the output from Map and combines the data tuples into a smaller set of tuples.
Framework Design

• Intellectual Challenging Task
• Requires a deep understanding of the problem domain
• Requires mastering of software (design) patterns, OO methods and polymorphism in particular
Polymorphism and Generic Programming

• A classification of Polymorphism
• Polymorphism in C++: inclusion polymorphism and templates
• Java Generics
• The Standard Template Library: an overview
• Generics and inheritance: invariance, covariance and contravariance
Functional programming and Haskell

• Introduction to Functional Programming
• Evaluation strategies (lambda-calculus)
• Haskell: main features
• Type Classes and overloading
• Monads
• Functional programming in Java
  – Lambdas and Stream API
Scripting Languages and Python

• Overview of scripting languages
• Main features of Python
• Imperative, functional and OO programming in Python
• Higher-order functions and Decorators
• On the implementation of Python: the Global Interpreter Lock
Selected Advanced Concepts in Programming Language

- Overloading and Type Classes in Haskell
- Closures vs Delegates in CLI
- Algebraic data types and Active patterns in F#
- Associative arrays in scripting languages
- Ownership and borrowing in Rust
- Extensions in Swift