Learning Objectives

- Knowledgeable in how sound software engineering principles for component-based design are manifested in a current popular programming language
  - SE principles: Resolve
  - Programming language: Java
- Proficient at Java programming
- Proficient at use of industrial-strength software development tools
- Informed in good programming practices
Pre- and Post-requisites

- **Required background: CSE 321**
  - Typed imperative programming paradigm
    - Control flow, types, variables, arrays
  - Encapsulation and information hiding
    - Client view vs implementation view
    - Abstract vs concrete templates/instances
  - Behavioral specifications
    - Mathematical model and constraints
    - Abstraction correspondence and conventions
    - Requires, ensures, and alters clauses

- **Preparation for: CSE 560**
  - Practical programming patterns
  - Tool support for software development

Course Content

- **Language**
- **Tools**
- **Good programming practices**
Course Content 1: Language

- Core syntax and features
  - Declarations, assignment, control flow
  - Methods, objects, classes, interfaces
  - Inheritance, polymorphism
  - Generics, exceptions
- Packages (ie Java component catalogs)
  - Collections (eg Map, Set, Queue, List...)
  - Logging, IO, Swing for GUIs

Course Content 2: Tools

- Eclipse
  - Industrial-strength open source IDE
  - Many (free) extensions available
- Javadoc
  - Industry-standard documentation utility for Java programs
- JUnit
  - Industry-standard library for unit testing programs
- CVS/SVN
  - Widely-adopted versioning systems for coordinating team development
Course Content 3: Good Practices

- **Problem:**
  - Complex language mechanisms make it easy to produce code that is wrong, brittle, inextensible, and hard to maintain

- **“Solution”:**
  - Good programming practices form a discipline that helps (but does not guarantee) developers write better code

- **Simple syntactic idioms**
  - Naming conventions, coding conventions
  - Decoupling by “programming to the interface”

- **Complex design patterns**
  - Single-point of control (eg factories, MVC)
  - Maintaining an invariant (eg immutable, singleton)

What is Java?

- Developed by Sun Microsystems (now Oracle)
  - James Gosling
  - Birth: 1994 (progenesis from Oak)

- Based on C/C++
  - Similar syntax, control, data structures
  - Imperative, object-oriented

- Originally designed for building Web/Internet applications
  - Now often viewed as a “general purpose” programming language

- Currently enjoys wide-spread acceptance
  - Had immediate impact, then continued success
Major Java Myths

1. Java is a small, simple language
   - True initially, but every revision to the language has added functionality and complexity

2. Java does not have pointers
   - References (ie pointers) are ubiquitous

3. Once I start using Java, I can forget all that Resolve/C++ stuff
   - Understanding sound principles for component-based software is even more important
Resources

- On line tutorials from Sun (“trails”)
  - http://java.sun.com/docs/books/tutorial
- On line API documentation
  - http://java.sun.com/javase/6/docs/api
- Class website
  - Handouts, lecture notes, lab assignments
  - Pointers to more resources
- Piazza (www.piazza.com)
  - Discussion forum, news, announcements
- Carmen
  - Lab submission (in “dropbox”), grades

The Java Virtual Machine (JVM)

- An abstract computer architecture
  - The software that executes Java programs
  - Part of Java Runtime Environment (JRE)
- Java program compiled into bytecode
- Java bytecode then interpreted by JVM
Implications of JVM

- **Portability**
  - Sun slogan: “Write once, run anywhere”
  - JVM is ubiquitous

- **Environment configuration**
  - path variable
    - for shell to find java / javac executables
  - classpath variable
    - for JVM to find bytecode at execution time

- **Dynamic extensibility**
  - JVM can find bytecode on-the-fly

- **Performance**
  - Extra layer comes at (small) penalty in performance

Environment Setup: JDK 6

- **Version 6 == Java 2 version 1.6**
- **Lab: CL 112 (& Baker 310 if available)**
  - [http://www.cse.ohio-state.edu/cs/labs.shtml](http://www.cse.ohio-state.edu/cs/labs.shtml)
- **CSE login server: stdlinux.cse.ohio-state.edu**
  - Red Hat Enterprise Linux 6
  - Solaris servers (ie stdsun) will not work for 421
  - See class web page, Resources, then “Eclipse use at OSU” for instructions on logging in to stdlinux
    - X-Win32 or VNC or ssh

- **Confirm set-up**
  - `$ java -version`
  - `java version "1.6.0_25"`
  - ...
Install Java Platform at Home

- Can be installed on different platforms:
  - Solaris, Windows, Linux, ...
- Trail: Getting Started > “Hello World!”
  - Download OS-specific Java Development Kit (JDK)
    - Tools for program development (eg javac)
    - JRE
  - Create simple program (with a text editor)
  - Compile (with javac)
  - Run (with java)
- Make sure to download:
  - J2SE JDK (not J2EE, not JRE, not bundles)
  - Version 6 (latest: 1.6.0_27, ie update 27)

Getting Started:
1. Creating Source File

- Using any text editor:
  - Create a file HelloWorldApp.java
  - Copy the following code into this file:
    ```java
    public class HelloWorldApp {
        public static void main(String[] args) {
            // Display "Hello World!"
            System.out.println("Hello World!");
        }
    }
    ```
- Note:
  - Class name must match file name
  - Java is CASE SENSITIVE!
Getting Started:
2. Compiling the Program

- Compile using javac
  
  \$ javac HelloWorldApp.java

- Generates a file named HelloWorldApp.class
  
  \$ ls
  
  HelloWorldApp.class  HelloWorldApp.java

- Problem
  
  javac: command not found

- Cause
  
  Shell can not find javac executable

- Solutions
  
  - Use full path on command line
    
    \$ /usr/local/jdk1.6.0_25/bin/javac HelloWorldApp.java
  
  - Set path environment variable
    
    \$ export PATH=$PATH:/usr/local/jdk1.6.0_25/bin/

Getting Started:
3. Running the Program

- From same directory, run using java
  
  \$ java HelloWorldApp
  
  Hello World!

- Note:
  
  - argument is HelloWorldApp, not a file (.java or .class)

- Problem
  
  Exception in thread "main" java.lang.NoClassDefFoundError: HelloWorldApp

- Cause
  
  JVM can not find HelloWorldApp bytecode (ie .class file)

- Solutions
  
  - Explicitly set classpath on command line
    
    \$ java -classpath ~/421/example HelloWorldApp
  
  - Set classpath using environment variable
    
    \$ export CLASSPATH=.:~/421/example
Language Basics: Statements

- Similar to C/C++
- Control flow:
  - if, if-else, if-else if
  - switch
  - for, while, do-while
  - break
  - continue
- Statements
  - Separation with ;
  - Blocks with { . . . }
- Comments with // or /* . . . */
- Operators
  - arithmetic: + - * / % ++ -- ...
  - logical (for booleans): & | ^ ! && ||
  - bit (for integer types): & | ^ ~ << >> >>>
  - relational: == != < > <= >=

Good Practice: Single-Statement Conditionals

- Always include body of if-else in braces, even if it is a single statement
- The following is correct, but discouraged:
  ```java
  if (!isDone)
    retry = true;
  ```
- Instead, write:
  ```java
  if (!isDone) {
    retry = true;
  }
  ```
Supplemental Reading

- Sun trails
  - Getting Started
  - Learning the Java Language > Language Basics
- Java overview white paper
  - http://java.sun.com/docs/white/langenv/
- Another walk-through of simple application
  - “Essentials of the Java Programming Language, Part 1”
  - Lessons 1 and 2

Summary

- Main course learning objective
  - Applying solid SE principles in Java programming
- Course content
  - Language, tools, good practices
- JVM
  - .java (source) vs .class (bytecode)
  - javac (compiler) vs java (interpreter)
- Environment configuration
  - Setting class and classpath
- Sample program: Hello World