CISC 3120
Design & Implementation of Software Applications I

Lecture #2 – Programming Language Theory (Programming Language Analytics)

Professor: Matthew K. Meyer
Course Page:
http://www.sci.brooklyn.cuny.edu/~meyer/CISC3120
Content

1. Languages
   1. Definition & Reliability
   2. Syntax & Semantics
   3. Java Syntax & Semantics

2. Computer Languages
   1. Binary & Assembly
   2. Compilers & Interpreters
   3. Java Compilers & Interpreters and the Java Development Process

3. Paradigms
   1. Languages VS. Tasks
   2. Paradigms
      1. Imperative
      2. Procedural
      3. Object-Oriented
(1.1) Languages

For our purposes: A **language** is a structured means of information exchange.

- Spoken languages (English, Chinese)
- Written languages (Arabic, Latin)
- Visual languages (ASL, QE)
- Tactile languages (Braille)
Reliable Languages

- Can a painting or piece of music "speak to you"?
- Will it say the same thing to every person?
- Will it say the same thing to you, every time?
- Not all languages are "reliable" means of communication.
(1.2) Syntax & Semantics

To be useful and reliable, a language must have a well defined syntax and semantics.

- **Syntax**: The structure and punctuation of the language.
- **Semantics**: The meaning of objects (words) and combination of objects in the language.
(1.3) Java Syntax

- You are already familiar with the vast majority of Java Syntax (because of C++).
- Syntax rules include:
  1. Java is case sensitive (Age != age)
  2. Java statements end with a semicolon ' ; '
  3. Extra whitespace is ignored
  4. Blocks of code are delineated with curly braces - start with { and end with }.
  5. Comment are exactly like in C/C++
     - // Line Comment
     - /* Block Comment */
     - /** JavaDoc Comment */
(1.3) Java Semantics

• This is a partial list of the reserved words in Java. How many do you recognize:

- abstract
- assert
- boolean
- break
- byte
- case
- catch
- char
- class
- const
- continue
- default
- do
- double
- else
- enum
- extends
- final
- finally
- float
- for
- goto
- if
- implements
- import
- instanceof
- int
- interface
- long
- native
- new
- package
- private
- protected
- public
- return
- short
- static
- strictfp
- super
- synchronized
- this
- throw
- throws
- transient
- try
- void
- volatile
- while

* Not used
** - **** Recently added
(2.1) Computer Languages

- At the lowest level computers are exchanging information in binary format (0,1).
- Combination's of 0's and 1's can have different meanings though, depending on the "encoding scheme" (language) and/or the machine language being used.

- Assembly -> 00001001 -> (Use register 9)
- ASCII  -> 00001001  -> (Tab character)
(2.2) Compilers & Interpreters

- Having to write everything out in binary would be both difficult and time consuming (punch-cards).

- **Compilers and Interpreters** are programs that take code written in other languages and create the 0's and 1's (the binary) that a machine will actually understand.

- Thank you Admiral Hopper!
(2.3) Java Program Execution

- **Compilation** - Code entered is reduced to set of machine-specific instructions then saved as an executable file (C++).
- **Interpretation** - Code is saved "as is" and translated to instructions for the underlying machine at runtime. Code can be changed modified at run time but usually runs slower.
- Java uses a combination of these two techniques:
  - Compilation - to an intermediate (low-level) format
    - One format is byte-code -- similar to machine code
  - Interpretation of that low-level format on a virtual machine
    - Software simulation of CPU whose machine language is the byte-code.
- **This scheme allows for**
  - High degree of platform independence
  - Fast introduction to new platforms
(2.3) Java Virtual Machine

- Java source is compiled to *Java byte code*
- The byte code is then executed via:
  1. Interpreter on the JVM
  2. JIT - *Just-in-time* compilation
     - Java byte-code is compiled to native machine code and executed
  3. Direct execution - a Java chip is built-- a CPU who native machine language is Java byte code
(2.3) Java Platform & Architecture

- **JDK - Java Development Kit**
  - Software distribution containing tools (javac, java) for developing and running Java applications
  - Comes in a variety of flavors:
    - J2SE - Standard Edition
      - Desktop applications & applets (what we're going to use)
    - J2EE - Enterprise Edition
      - Server-based applications
    - J2ME - Micro Edition
      - Embedded systems with resource constraints (mobile devices)

- **JRE - Java Runtime Environment**
  - Software and classes for running Java applications
  - But no development tools
Paradigms

(hint, it's not 20 cents)
(3.1) Languages VS. Tasks

- We both speak the same language, therefore, you might think it would be easy for me to get you to complete a certain task or solve a specific problem.
  - Example 1: Give me $50.
  - Example 2: Land a plane.

- In example 1, you are going to want a reason, for 2, you are going to need instructions (if you don't already know how).
(3.1) Languages VS. Tasks

- Just because you and a computer have a means of communication (a programming language) doesn't mean that it will be easy for you to get the computer to do what you want.
  - Computers are dumb! Really, really dumb.
    - Maliciously, stupid.
  - Nothing can make horrible mistakes faster than a computer.

- Over time researchers have developed "programming paradigms" to help construct programming languages for working with (these really stupid) computers.
(3.2) Paradigms

- A paradigm is a structured approach to solving a problem.
  - Give me $50 dollars, please.
  - Give me $50 dollars, I will pay you back.
  - This is a stickup, give me all of your money.

- When we are writing a program, we aren’t just trying to "communicate" to with the computer, we are trying to get it to accomplish a specific task.
  - Our problem is getting the computer to do what we want.
  - NOTE: It’s no use threatening a computer.

- Programming paradigms are organized approaches to solving the problem of how to get a computer to accomplish a specific task.
(3.2) Paradigms

- The advantage to learning about programming paradigms is that once you understand a particular paradigm it is easier to learn other programming languages that also use that same paradigm.

- 3-paradigms in particular are ubiquitous to programming:
  - Imperative Paradigm
    - A "Smart List"
  - Procedural Paradigm
    - "Making phone calls"
  - Object-Oriented Paradigm
    - A program as set of "interacting objects"
(3) Paradigms

- **NOTE:** There are hundreds of other paradigms.
- Some other famous paradigms are:
  1. **Functional**
     - Views program as a mathematical equation
     - Examples: Haskel
  2. **Logical**
     - Views a program as a logic puzzle
     - Examples: Prolog
- But the most popular paradigm is the Imperative one.
  - Views a program as a "smart" list
(3.1) Imperative Paradigm

- An Imperative is a command, an obligation, or a requirement.
  - It is a moral imperative that you not cheat.
- In the imperative paradigm, we give the computer a specific set of commands that it must follow.
- The imperative paradigm views a program as a "list" of things the computer must do.
- You might want to remember it as a "smart list".
(3.1) "A Smart List"

Imperative languages need 3 things:

1. **Sequence** - A predefined order in which to process information.
2. **Selection** - The ability to make a choice. The "IF" statement.
3. **Repetition** - The ability to repeat an action. The "WHILE" statement.
Imperative Programming in Java

1. **Sequence**: After the compilation process the main() method is called and executed Left->Right, Top-> Bottom.

2. **Selection**:
   - if, if else
   - switch

3. **Repetition**:
   - for
   - while, do while

- Control structure syntax identical to C++
Java (Imperative) Example

// main method
public static void main(String[] args) {
    int i = 0;
    while( i < 10 ) {
        if( i % 2 == 0 ) {
            System.out.println( i + " even." );
        } else {
            System.out.println( i + " odd." );
        }
        i++;
    }
}
(3.2) Procedural Programming

- **Procedure Call**: the ability to "send a message" to another separate section of a program, and get information back from that section.

- **Function (procedure, method)**: A block of code with a name. To access the code simply use the name (instead of retyping the same code, over and over).

- **Types of procedural calls**:
  - Function Calls (MOST COMMON)
  - Message Passing
  - Event Handlers
  - RPC Calls, REST, etc.
Java (Procedural) Example

```java
public static void main(String[] args) {
    int i = 0;
    while( i < 10 ) {
        System.out.println( oddOrEven(i) );
        i++;
    }
}

public static String oddOrEven(int x) {
    if( x % 2 == 0 ) {
        return x + " is even."
    } else {
        return x + " is odd."
    }
}
```
(3.3) Object-Oriented Programming

- Some programs are hard to conceptualize as a list of instructions.

Example-> Tell me about chess:

1. There is a board:
   1. It has this shape, this pattern, these facts/properties.
2. There is a queen:
   1. It is a piece.
      1. It has this shape, these facts/properties.
      2. It can perform these actions, move, capture on the board.

- It is easier to describe some problems in terms of a series of interacting objects.
(3.3) Object-Oriented Programming

Basic Concepts:

1. Visualize a program as a set of interacting objects (easy to do with games).
2. Identify the associated facts (properties) and functions (methods) of these objects.
   1. Properties (Facts): Where is it?
   2. Methods (Functions): What can it do?
3. Create templates/patterns (classes) for these objects that can be reused.
   1. Game-Object -> paddle1, paddle2, ball
   2. Score-Object -> player1, player2
(3.3) Object-Oriented Programming

- **Class**: A class is a pattern (template, blueprint) that contains both the recipe for creating an object AND instructions regarding what that object can legally be allowed to do.

- Take a second and think about the data type integer. The computer knows what an integer is (whole number, comprised of 32 bits) and what it is allowed to do (i++).

- Classes allow us to create our own data-types and specify what they are allowed to do.
Java (OO) Example

public class StopLight {
    double latitude;
    double longitude;
    double direction;
    String color;
}

.... in another program ...

int x = 10;       // type-> int, name-> x, value-> 10
StopLight main = new StopLight();
/* type-> StopLight , name-> main,
   value-> a pointer to a place on the heap, where
   a collection of 4 variables can be found. */
main.color = brown;       // dot indicates ownership
Java (OO) Example

- The previous class example (StopLight) contained only "member variables", which are variables that will be part of the object when an object is created from the class pattern.

- The last line in the previous example was problematic. Brown is not a valid color for a streetlight (red, yellow, green are).

- We don't want to allow random changes to the color variable. In many cases we want to specifically limit a class so that only certain function can be performed on an object.

- Take a look at the next two example pages.
public class StopLight {
    public double latitude, longitude, direction;

    private String color;

    // Constructor (there is no return type)
    public StopLight(){
        currentColor = "red";
    }

    public void changeColor() {
        if( color.equals("green") ) {
            color = "yellow";
        } else if (color.equals("yellow") ) {
            color = "red";
        } else {
            color = "green";
        }
    }
}

Java (OO) Example
Java (OO) Example

.... in another program ...
int x = 10;
// type-> int, name-> x, value-> 10

StopLight main = new StopLight();
/*
   type-> StopLight ,
   name-> main,
   value-> a pointer to a place on the heap,
   where a collection of 4 variables can be found.
*/

main.change(); // stared as red, so is now green

main.color = brown; // ILLEGAL ACCESS, THROWS EXCEPTION (CRASH)
Java OO Practicalities

• **Everything** in Java is a class.
  ◦ Classes that have main() method will be treated as programs.

• **Access Specifiers** (control access... duh)
  ◦ There are several types but public and private will be good enough for our work.
    • Public -> Anyone, anything can call/use this class, method, or variable.
    • Private -> Only members of the current class, can call/use this class, method or variable.
A class is just a pattern (like a dress pattern).

A class tells us the facts and functions that are part of an object.

We can use a class as many times as we want!!!

The dot (dot syntax) indicates ownership or belonging.
Paradigms Review

• The advantage to learning about programming paradigms is that once you understand a particular paradigm it is easier to learn other programming languages that also use that same paradigm.

• 3-paradigms in particular are ubiquitous to programming:
  ◦ Imperative Paradigm
    • A "Smart List"
  ◦ Procedural Paradigm
    • "Making phone calls"
  ◦ Object-Oriented Paradigm
    • A program as set of "interacting objects" created from classes.
The End