CS620
Structured Programming

Day 1

Java Language specifics

Java - The Platform

- A bit more about Programming Languages, Java and the things that make Java work

Programming Languages

- Many kinds of programming languages with diverse uses and traits
  - Different platforms
  - Various tradeoffs between use-friendliness and performance

- Languages can be broadly grouped according to their ‘level’
  - The more human-readable the code the higher its level.
  - The closer to writing processor-specific ‘machine code’, the lower its level

Programming Languages

- Languages can be broadly grouped according to their ‘level’
  - The more human-readable the code the higher its level.
  - The closer to writing processor-specific ‘machine code’, the lower its level
  - High-level languages are often built ‘on top of’ low-level languages
    - C/C++ compilers are usually written in Machine-Code/Assembly
      - GCC (A famous C/C++ compiler) is itself written in C and compiles itself!
    - Java compilers and JVMs can be written in C/C++

Programming Languages

- Languages by their level (roughly):
  - **Low-Level:** Binary & ‘Machine Code’
  - **Platform-Compiled:** C/C++
  - **VM / Managed:** Java, Microsoft .NET
  - **Scripting:** Python, PHP, Ruby, Lua

- What is Java?
  - A modern, portable, object-oriented language designed for programming cross-platform software to run on a ***Virtual Machine***.

Portability - Virtual Machines

- In Computing, a *Virtual Machine (VM)* is a virtual/imaginary computer running purely as software on an actual computer.

- Java uses a VM called the *Java Virtual Machine* (**JVM**).

- When you compile a program in Java, the program will run on the **JVM**, which is in turn running on the CPU of the computer.
  - The java program does not run directly on the CPU!

Compilation - Program Portability

• A program written in C, on the other hand, will run directly on the CPU of the computer.

• The C program needs to be recompiled for the specific OS and CPU architecture that it's intended to run on.

- If you write a program for Windows in C, you will need to recompile it for that program to work on Linux or Mac OS.
- If you write a program for an Intel CPU, you'll need to recompile to run it on an ARM CPU.
- This makes it harder to write 'portable' applications in a language like C.

Virtual Machines

- The **JVM** is written in a lower-level language such as *C++* and compiled specifically for that operating system and computer architecture.

- Programs written in Java running on the JVM don't care what kind of computer it is.

- All a Java program sees is the JVM and all it knows about the computer is what the JVM tells it.

What's the point of the JVM?

- The mantra of Java's design was *"Write Once, Run Anywhere"* - Meaning:
  - Write Java code
  - Compile Java code into an application
  - Run that application on any computer with a working JVM, regardless of its brand/architecture/design/operating system.

- This is why programs written in Java for Windows or Linux can be ported very quickly and easily to a very different system such as an *Android* Smartphone.
  - Note: *Google Android* is mostly comprised of Java programs and applications on top of a JVM written in C++ and a Linux-based OS written in C!

The JDK

• You can write Java source code on any machine you want as long as you have a text editor

- The simplest kind of text editor (such as Notepad) will do, but others exist that make programming easier

• To **compile** the source code you need a compiler

- You'll typically use the compiler in the JDK (Java Development Kit), called ***javac***

The JRE

• To **run** a java application, you don't need the JDK

• You only need the **JRE** - The *Java Runtime Environment*.

- The JRE contains a JVM tailored for that type of computer.

• This means that you can write a program in Java code, compile it with the JDK; and anyone who has the JRE (or a working JVM) can run your program.

Getting the JRE & JDK

- The simplest way to get the tools you need for Java development is to go to the site of the company that oversees Java development: ***Oracle***
  - ***Sun Microsystems*** originally developed Java, but Oracle bought them out a few years ago!

- Get both the JDK and JRE here:
- http://www.oracle.com/technetwork/java/javase/index.html

Java - The Language

- Back to the language itself and the different parts of its ‘grammar’ that you need to know

Keywords

- There are 50 keywords in Java that identify core parts of the language

- You can’t name your variables, classes or methods the same as any of these keywords.

- You don’t need to learn them off by heart, but you’ll become familiar with most of them over time just by working with Java.

Keywords

- Some commonly-encountered keywords:
  - break
  - case
  - switch
  - char
  - class
  - char / int / float / double
  - if / else
  - for / while
  - import
  - public/private/protected
  - this
  - void
  - new

Types

- Java is a ‘strongly typed’ language

- This means that it’s strict about what kind of data can be stored in different kinds of variables

- In practice, this means we need to declare special variables for different kinds of information

Types

- When we declare a variable, we tell the compiler what the variable's *type* is.

- The compiler converts that into an instruction for the computer to set aside a certain amount of space in memory for that variable.

Types

- Different *types* have different space requirements
  - We'll see this when we look at the different fundamental types more closely

- It's good practice for computational efficiency to choose the smallest *type* that will hold the information you want to store
  - For a simple program on a fast computer the difference will be entirely negligible
  - If your program is very complex and performance-intensive, or you're using a computer with limited performance, this matters much more!

Basic Types

- Integer
  - Holds **whole** numbers (only)
  - Keyword: ‘int’
  - Range: *-2,147,483,648* to *2,147,483,647*
  - If you try to store a decimal value in an integer it will be rounded to the nearest whole number
    - The rounding discards the decimal point and everything after it; it won’t round upwards!
    - Example:
      - `int x = 5 / 2`
      - The value of x will be 2, not 2.5!
  - Integers are suitable for basic counting and storing simple numbers

Basic Types

- Double
  - Holds **Decimal** numbers
  - Keyword: ‘double’
  - Range: **Huge**
    - The maximum range of a double-precision floating-point variable depends on the JVM and the architecture of the computer!
    - Suffice to say, a double is typically big enough to hold most decimal numbers that you’ll have to work with.
  - Doubles take up more space in the computer’s memory than ints
  - Generally speaking, use doubles any time you need to work with decimal numbers

Basic Types

- Float
  - Holds **Decimal** numbers, but smaller than a double
  - Keyword: ‘float’
  - Range: **Still Huge**
    - The maximum range of a floating-point variable depends on the JVM and the architecture of the computer!
    - Suffice to say, a float is typically big enough to hold most decimal numbers that you’ll have to work with.

  - Floats take up more space in the computer’s memory than *int*s, but less than *double*s

  - You would use floats in a very performance-intensive program where speed matters and where you know your decimals will be small enough to fit!

Basic Types

- Byte / Short / Long
  - Hold whole numbers
  - Similar to '*int*' but different ranges
  - Keywords: *byte*, *short*, *long*
  - Ranges:
    - Byte: -128 to +127
    - Short: -32,768 to 32'767
    - Long: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
      - (You don't need to know these numbers off by heart..)

Basic Types

- Boolean
  - Holds boolean values: true / false **only**
  - Keyword: *boolean*
  - Range:
    - *true* or *false*
  - Booleans are used to store the results of comparisons
  - Example:
    - 4 == 5 returns the boolean value: *false*
    - 3 < 6 returns the boolean value: *true*

Basic Types

- Char
  - Holds single characters (text)
  - Keyword: *char*
  - Chars are treated as symbols
  - If you store a number as a char, such as:
    - char x = 5
    - The computer will see this as the **symbol** ‘5’ or the **letter** ’5’, even
    - You cannot perform maths on *chars*!
  - You will rarely work with individual chars

Not-So Basic Types - Strings

- String
  - Holds multiple *chars*
  - Keyword: *String*
    - Note the capital S
      - This indicates that the String is not a ‘fundamental’ type
      - A String is a constructed type, made up of multiple *chars* stored as a string of text
  - Any piece of text longer than one character should be stored as a string
  - Typically, you’ll store even single characters in strings for simplicity, even though it is not as efficient!

Variables

- Setting-up, managing and working with variables

Variable Declaration

- Declaring a variable:

  - Specify its type, specify its identifier, end with a semicolon;

  - Examples:

    - int x;
    - int someNumber
    - char y;
    - char aSingleCharacter;
    - double z;
    - double someBigDecimalNumber;

  - It’s good practice to give your variables descriptive names

    - “int aNumberUsedToCountSomething” is better than “int x”

Variable Assignment

- Assigning values to variables:
  - Use the assignment operator: ‘=‘
  - In maths, equals is how we ‘get’ our answer
    - 5 + 5 = ?
      - Answer: 10 (Just in case you were wondering!)
  - In programming, equals is how we assign a value
    - x = 5 + 5
    - The value of x is now 10
  - This might take some getting used to
    - A handy way to think of it is that we put our answer on the left in programming:
    - Answer = expression

Variable Assignment

- Assigning values to variables:
  - If the variable doesn't already exist:
    - Declare it and give it a value at the same time
    - `int someNumber = 10;`

  - If the variable already exists (it's been declared already):
    - Just give it a value, don't re-specify the type
    - `someNumber = 10;`
    - If you specify the type, you'll either get an error claiming that the variable already exists; or you'll create a new variable with the same name somewhere else in memory!
    - This depends on 'scope', which we'll talk about later.

Variable Initialization

- It’s necessary to initialize your variables

- This means giving them a value to start out with, or making sure to assign them a value at least once before using them for anything else

- Uninitialized variables typically contain ‘junk’ data left over from other variables that used the same space in memory some point previously on the computer.

Variable Initialization

- We saw previously that we can declare a variable without a value such as:
  - `int someNumber;`

    - What happens when you try to do something with an uninitialized variable?

    - What's the result of "someNumber + 10;" ?

- Problems! Errors!

    - The Java Compiler won't let you compile a program that tries to use an uninitialized variable.

    - This prevents bugs creeping into code!

Operators

- Doing stuff with variables!

Operators

• Operators in java are generally the same as symbols/signs in mathematics

- ‘+’, ‘-’, ‘*’, ‘/’

  - Addition, Subtraction, Multiplication, Division

  - The asterisk and slash symbols are used in most programming languages instead of the traditional ‘ ’ for multiply and ‘ ’ (*obelus*) for divide, respectively

Operators

• Not all of the operators have the same meanings as you might be used to, however.

- ‘+’ Means addition
  - *‘+’ Also means ‘positive’*
  - ‘+’ Also means ‘string concatenation’
    - (i.e.; glue two strings together as one)
    - *“Some Words”* + *“Some other words*” in java terms gives: *“Some words Some other words”*
- *‘-’ Means subtraction*
  - *‘-’ Also means ‘negative’*

– *‘*’ Means multiplication*

- *‘/’ Means division*

Operators

- *Some operators you might not be familiar with:*

  - *‘%’ Means modulo* - To calculate the remainder

    - *Example:*
      - *10 % 5 gives remainder 0*
      - *11 % 5 gives remainder 1*

    - *Modulo is useful for checking if a number is odd or even:*
      - *If the remainder of a number divided by 2 is not 0, then the number must be odd!*

Operators

- *Some operators you might not be familiar with:*

  - '++' means 'increment'
    - `int x = 5;`
    - `x++;`
    - `Result: x is now 6`
    - This is the same as the code:
      - `int x = 5;`
      - `x = x + 1;`
      - `Result: x is now 6`

  - *'--' means 'decrement'*
    - `int x = 5;`
    - `x --;`
    - `Result: x is now 4`
    - This is the same as the code:
      - `int x = 5;`
      - `x = x - 1;`
      - `Result: x is now 4`

Logic Operators

- More operators:
  - '==' *(is Equal to)*
    - Used with conditionals
    - **NOT** the same as '='
    - More on this the next day!
  - '!=' *(is not Equal to)*
    - Used with conditionals
  - *> (Greater than)*
    - *>= (Greater than or equal to)*
  - *< (Less than)*
    - *<= (Less than or equal to)*

- We'll look at these operators in more detail when we deal with conditionals and boolean algebra later