Intro. To Java Programming Language (I)

- **I assume you know Java basics.**

- References:

  - Text book : Appendixes
  - Sun Java Tutorial : http://download.oracle.com/javase/tutorial/index.html

1. Overview

- **What is Java?**

  - Developed by Sun Microsystems
  - A general-purpose object-oriented language
  - Based on C/C++
  - Designed for easy Web/Internet applications
  - Widespread acceptance

- **Main Features**

  - **Simple :** no pointers, automatic garbage collection, rich predefined class library etc

  - **Object oriented:** all functions are associated with objects; almost all datatypes are objects (files, strings, etc.)

  - **Interpreted & portable:**

    - java compiler, *javac*, generates byte-codes, not native machine code.
    - The compiled byte-codes are platform-independent.
    - They are translated to machine readable instructions in runtime by Java Virtual Machine. Example: *java* launches JVM

  - **Others**: reliable, secure, multithreaded, etc

- **Environment Setup & version:**
  - thecity.sfsu.edu – Java 1.6
  - libra.sfsu.edu – Java 1.5
  - My PC : Java 1.6
  - You should use Java 1.6 or later
  - Download & Install Java (JDK) to your PC :
    http://www.java.com/en/download/

2. Concepts and a simple Java application program

- Highlight few important concepts:
  - Objects & Classes:
    - An object is a program construct that contains data (fields) and can perform certain actions (methods).
    - A class is a blueprint or prototype from which objects are created.
    - All objects in the same class have the same types of data and the same methods.
    - Example: String str1= “hello”; String str2 = new String(“world”);
  - Interfaces:
    - An interface is a contract between a class and the outside world.
    - When a class implements an interface, it promises to provide the behavior published by that interface.
  - Packages:
    - A package is a namespace that organizes a set of related classes and interfaces.
    - Conceptually you can think of packages as being similar to different folders on your computer. A package is a folder which contains a collection of classes.
    - The Java platform provides an enormous class library (a set of packages) suitable for use in your own applications. This library is known as the **"Application Programming Interface" (API).**
  - Applets & Application:
    - An applet is program which runs inside a browser (see more explanation in text book).
    - An application program is a program that runs on your computer.
    - In this course, we cover only application programs.

- **A simple Java application program**
  - **Step 1: Create the source file**
    - open a text editor, type in the code which defines a class (*HelloWorldApp*) and then save it in a file (*HelloWorldApp.java*) file
    - class names are case sensitive and must be matched exactly (except the .java part).

```
/**
 * The HelloWorldApp class implements an application
 * that displays "Hello World!" to the standard output
*/

public class HelloWorldApp {
        public static void main(String[] args) {
                System.out.println("Hello World!");
        }
}
```

    - Comments: Three kinds :

/* *text* */

/** *documentation* */

This indicates a documentation comment (*doc comment*, for short). The javadoc tool uses doc comments when preparing automatically generated documentation.

// *text – to end of line*

- Define a class “HelloWorldApp”. One of most basic form of a class definition. Will cover more on classes later

  Every application begins with a class definition

  In the Java programming language, every application must contain a main method (just like C/C++) whose signature is:

  ```
  public static void main(String[] args)
  // or static public, commonly argument name is “args” or “argv”
  ```

  Finally, the line:

  ```
  System.out.println("Hello World!");
  ```

  uses the System class from the Java core library. System.out is an object within class System and println() is a method within the object which prints the "Hello World!" message to standard output (screen).

- **Step 2: Compile the Source File into a .class File**
  - compile program HelloWorldApp.java

    javac HelloWorldApp.java

  - it generates a bytecode file named HelloWorldApp.class in the same directory as HelloWorldApp.java

  - Note: You may need to use full path to javac OR set PATH environment variable. Some common compilation problems: see http://download-llnw.oracle.com/javase/tutorial/getStarted/problems/

  - In this course, we will also use IDE (Integrated Development Environment) for software development. More on this later.

- **Step 3: Run the program**

  - run the code through:

    java HelloWorldApp

  - Note that the command is java, not javac, and you refer toHelloWorldApp, not HelloWorldApp.java or HelloWorldApp.class

  - if you see any error, see common problems link as given previously.

- More on Java program files: A Java programmer deals with *source* files and *bytecode* files (no *executable* files).

  - Source Files:
    - use extension .java and may have multiple classes
    - each source file can contain at most one *public* class
    - if there is a public class, then the class name and file name must match

    - Examples :

      If a source file contains the following 3 classes :

```
public class Test { ... }
class Foo { ... }
class Bar {... }
```

      then it must be in a file named Test.java

      If a source file contains the following:

```
class Test { ... }
class Foo { ... }
class Bar {... }
```

      then it can be in any ".java" file

    - Notes:

      Every function must be part of a class.

      Every class is part of a package

      A *public* class can be used in *any* package.

      A non-public class can only be used in its own package.

- Bytecode Files:
  - Are created by the Java compiler. They are ready to be "executed" (actually, really interpreted -- by the Java interpreter)
  - Foreach class in a source file (both public and non-public classes),the compiler creates one ".class" file, where the file name is the same as the class name
  - Example:

    If a source file contains the following:

    ```
    public class Test { ... }
    class Foo { ... }
    class Bar {... }
    ```

    then after compiling you will have three files:

    ```
    Test.class
    Foo.class
    Bar.class
    ```

- Execute Java Program:
  - The starting program name (class name) must contain a main() method.
  - The execution starts at the program's main() method.
  - Other classes that are used in this program may also have main() methods.
  - Those other main() methods are ignored during the program execution.

3. A quick overview of Java language basics

- Data Types
  - There are two "categories" of types: *primitive types* and *reference* types:
  - Primitive Types :

boolean	Boolean value true or false
char	holds one character (16-bit Unicode)
byte	8-bit signed integer
short	16-bit signed integer
int	32-bit signed integer
long	64-bit signed integer
float	floating-point number (32-bit)
double	floating-point number (64-bit)

  - Reference Types:
    arrays
    classes

  - Notes:

    There are no struct, union, unsigned, typedef (as in C/C++)

    arrays and classes are really pointers!! (but no pointer notations)

- Unicode vs ASCII code:

  - ASCII: a 7 bits encoding scheme for representing lower/upper case letters, digits, punctuation marks and control chars
  - Unicode: a 16 bits encoding scheme ‘\u0000’ to ‘\uFFFF’. First 128 chars ‘\u0000’ to ‘\u007F’ corresponding to ASCII chars
  - Note: do not cover Unicode

- Data fields in a class are assigned default values after declaration
  reference variables are initialized to "null"
  example:

```
int x;     x=5;              // default is 0
char c = 'A';                // default is '\u0000'
boolean b = true;            // default is false

int[] intArray;              // declare an array (pointer), initial value is null
intArray = new int[20]; // allocate storage, initial values in entries is 0
intArray[0] = 12;            // assign value to first entry

// determine current length of arrays
intArray.length  // → return 20

// declare with initial values, fixed number of entries
int [] A = {1, 222, 0};

int [][] A;                  //  Declare a two-dimensional array
A = new int[4][];            // A now has 4 rows, but no columns yet
A[0] = new int [1];          // A's first row has 1 column
A[1] = new int [3];          // A's second row has 3 columns
A[2] = new int [3];          // A's third row has 3 columns
A[3] = new int [5];          // A's fourth row has 5 columns
A[2][1] = 10;                // adding a value to 3rd row, 2nd column

int [][] B = new int[4][3];  // B has 4 rows, each row has 3 cols

// C has 3 rows, each row has 2 columns
double[][] C = {{1.0, 0.0},{0.0, 1.0} ,{2.1, 2.5}};

// one way to create or instantiate a String object from String class
// Will cover other ways later.
String myString;                      // declare a variable of String object
myString = new String("hello");   // create storage to hold a String object
```

- Local variables (in methods, blocks, etc) are not initialized. Warning msgs will be printed by compiler.

- Scope : variable scope begins in its declaration and end at closing brace of the pair of braces that enclose the variable's declaration

  Example:

```
{       …
        int tmp1=1;  // tmp1 available here
        …
        {       …
                int tmp2=5;  // tmp1 & tmp2 available
                …
        }       // end of scope for tmp2
        …       // tmp1 available
} // end of scope for tmp1
```

Notes: special case for "static variables"
will cover more later.

Intro to type conversion

- Booleans cannot be converted to other types.
- For the other primitive types, there are two kinds of conversion: *implicit* and *explicit* (type casting)
- Implicit conversions: In general, you can assign a value of any type to a variable of any type that appears further down on the list

  byte → short → int → long → float → double

  A value char can also be assigned to int variable

  Example:

  ```
  char c = 'a';
  int k = c;
  long x = k;
  float y = c;
  ```

  May want to use "type casting" to explicitly change type

- Explicit conversions: Explicit conversions are done via *casting*: the name of the type to which you want a value converted is given, in parentheses, in front of the value.

  Example :

  ```
  double d = 5.6;
  int k = (int) d;
  short s = (short)(d * 2.0);
  ```

- More conversions & examples in :
  http://java.sun.com/docs/books/jls/third_edition/html/conversions.html
  http://www.rgagnon.com/javadetails/java-0004.html

- Use type conversion carefully!
- Object type compatibility will be covered later!

- Named Constants:
  - Example :

```
// final – cannot change value of MYVALUE
// Usually, use uppercase letter for constants
// commonly, declare this at the beginning of class

        final double MYVALUE = 3.14159;

// access constant PI (type: double) in Math library

        Math.PI
```

- Precedence rules (partial listing): order from higher to lower
  - Parentheses ()
  - Unary operators: ++,--,+,-, !, (type casting)
  - Binary operators: *,/,%
  - Binary operators: +,-
  - Relational operators: <,<=,>,>=
  - Relational operators: !=, ==
  - Logical operators: &&
  - Logical operators: ||
  - Condition operators: ?:
  - Assignment operators: =, +=, -= etc

  Note: do not include bitwise and bit shift operators

- Increment and decrement operator:
  - ++a OR --a // pre: do an operation, then return value
  - a++ OR a-- // post: return value, then do an operation

- Enumerations : a group/set of named constants

  Example:

  ```
  enum Suit { CLUBS, DIAMONDS, HEARTS, SPADES }
  Suit card1 = Suit.SPADES;
  ```

- Flow of control

  - if, if-else, if-else if

    Example:

    ```
    if (testscore >= 90) {
       grade = 'A';
    } else if (testscore >= 80) {
       grade = 'B';
    } else if (testscore >= 70) {
       grade = 'C';
    } else if (testscore >= 60) {
       grade = 'D';
    } else {
       grade = 'F';
    }
    System.out.println("Grade = " + grade);
    ```

- switch & conditional operator

  Example:

```
max = (n1 > n2)? n1:n2;   // true → max=n1, false → max=n2

// need "break;" at the end of each case
// may use optional "default" for all other cases
// In general, switch works with the byte, short, char, and int, enum
switch (month) {
        case 1:  System.out.println("January"); break;
        case 2:  System.out.println("February"); break;
        case 3:  System.out.println("March"); break;
        case 4:  System.out.println("April"); break;
        case 5:  System.out.println("May"); break;
        case 6:  System.out.println("June"); break;
        case 7:  System.out.println("July"); break;
        case 8:  System.out.println("August"); break;
        case 9:  System.out.println("September"); break;
        case 10: System.out.println("October"); break;
        case 11: System.out.println("November"); break;
        case 12: System.out.println("December"); break;
        default: System.out.println("Invalid month.");break;
    }
```

- Loop : for, while, do-while

  Example:

```
int count = 1;
 while (count < 11) {
     System.out.println("Count is: " + count);
     count++;
 }


int count = 1;
 do {
     System.out.println("Count is: " + count);
     count++;
 } while (count <= 11);


for(int i=1; i<11; i++) {
       System.out.println("Count is: " + i);
 }


int[] numbers = {1,2,3,4,5,6,7,8,9,10};

// meaning: for each of value in array numbers[]
// can also use with enum and collections
 for (int item : numbers) {
     System.out.println("Count is: " + item);
 }
```

- Break & continue
  - Use “break” to terminate innermost switch or loop.
  - May use labeled break to terminate outer switch, Example:
  - Example:

```
search:
        for (i = 0; i < arrayOfInts.length; i++) {
            for (j = 0; j < arrayOfInts[i].length; j++) {
                if (arrayOfInts[i][j] == searchValue) {
                    // this terminates labeled “search” statements
                    // which is a nested for loop. Continue at print statement
                    break search;
                }
            }
        }
System.out.println("Stop at " + i + ", " + j);
```

  - The continue statement skips the current iteration of a loop. May also use labeled continue.

- Java Package and Java API
  - A package is a directory that contains a group of functionality related classes and interfaces.
  - To create a Java package

    Create a directory which is the package name. Example : mytestpackage

    Each file (classes or interfaces) must include “package mytestpackage;” in the first line. Compile and store in the directory

- Must make sure that CLASSPATH environment variable is set to point to any directories that contain Java packages that you want to import

- Use *import* statement to access packages.

  Example : import mytestpackage.*;

- Java API provides an extensive collection of libraries (or packages)

- Few common Java libraries:

  - java.lang provides classes that are fundamental to the design of the Java programming language
  - java.util ** this contains collections that we will study soon
  - java.net provides classes for network applications
  - java.awt contains all of the classes for creating user interfaces and for painting graphics and images
  - javax.swing provides popular GUI related classes

- Three ways to import java packages

```
import java.util.*;       // 1. make all classes in package java.util visible
import java.util.Stack; // 2. make only Stack class visible
Stack s = new Stack(); // usage : create a stack s

// 3. no import, use fully qualified class name
java.util.Stack s = new java.util.Stack();
```

- Special case: No import statement is necessary for using package java.lang

- Next, we summary several basic common classes.

- For complete specification, refer to : http://download.oracle.com/javase/6/docs/api/overview-summary.html

- The class Math
  - Define in package java.lang
  - Provides a number of standard math functions as static mathods (will cover static method later)
  - To invoke a static method, use *class name.method name()*
  - Some common functions :

    Math.ceil(x), Math.pow(x,y), Math.abs(x)

    Math.PI // static constant value 3.14159

    Math.E // static constant value 2.72 the base of natural logarithm e
  - http://download.oracle.com/javase/6/docs/api/java/lang/Math.html

- The class String
  - Define in package java.lang
  - Many ways to create strings

```
String greeting = "Hello world!";  // direct way to create a string

char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.'};
// using new operation to create string
String str1  = new String(helloArray);  // from array
String str2  = new String("hello world"); // from constant string
```

  - The String class is immutable, so that once it is created a String object cannot be changed.

  - The String class has a number of methods.

```
int len = str2.length();          // return length of a string

// concatenate two strings. Useful for long or multi-line strings
String str3 =  str1 + str2;
String str4 =  "hello" + 42;      // cast 42, become "hello42"

char c1 = str1.charAt(2);         // return third char in str1
strY = strX.trim();               // trims off leading and trailing white spaces
strY = strX.toLowerCase();        // convert all chars to lower case
str1.compareTo(str2);             // compare two strings, return <0,0,>0
str1.equals(str2);                // compare two string, return true or false
```

  - Escape chars : use "backslash" before special chars in strings

```
\"      // char double quote
\\      // char backslash
\n      // new line
\t      // tab
```

Example : String str4 = "special \"line\" \n next line";

- Class StringBuilder
  - Define in package java.lang
  - String objects cannot be altered.
    Sometime you like to modify it, example: change certain chars in a string

    String str1= “hello world”;
    str1= “Hello world”; // actually, discard old string, create a new string
  - StringBuilder allows several methods to modify current string :
    x.append(), x.delete(), x.insert(), x.replace, x.setCharAt()
  - Examples:

    ```
    //Create a StringBuilder object
    StringBuilder builder = new StringBuilder(“Line 1\n”);

    //Append text to the end of the buffer
    builder.append(“Line 3\n”);        // “Line 1\nLine 3\n”

    //Now we want to add text in between line 1 and line 3
    String lineToInsert = “Line 2\n”;
    int index = builder.indexOf(“Line 3”);
    builder.insert(index, lineToInsert);  // “Line 1\nLine 2\nLine 3\n”
    ```
  - More info :

    http://download.oracle.com/javase/6/docs/api/java/lang/StringBuilder.html

    http://download.oracle.com/javase/6/docs/api/java/lang/String.html

- Simple I/O : screen output and keyboard input

  - System class is defined in package java.lang

  - Examples : printing to standard out (screen) using System.out

```
System.out.println("hello");    // print with newline char at the end
System.out.print(16);           // no newline char, print an integer
System.out.print(5.5 * .2);     // print a floating-point number
```

  - The + operator can be useful when printing.

    It is overloaded to work on Strings as follows: If either operand is a String, it converts the other operand to a String or creates a new String by concatenating both operands .

    Example:

```
int x = 20, y = 10;
System.out.println("x: " + x + "\ny: " + y);  // print a string
System.out.println(x + y);   // print an int value
```

    The output is:

```
x: 20
y: 10
30
```

  - Scanner class is defined in package java.util

```
// create a scanner object. Input from standard in (screen)
Scanner sc = new Scanner(System.in);
x = sc.nextInt();          // read next integer
y = sc.nextDoublel();  // read next real number
str1 = sc.nextLine();    // read the string that appear next in the input data

// read the next group of contiguous chars that are no white space
str2 = sc.next();          // good for read a word
```

- Using Scanner object on a String

```
import java.util.*;
public class Hello {
    public static void main(String[] args) {
            String str1= "one   two       hello     world";
            // create a scanner object. Input from str1
            Scanner sc = new Scanner(str1);
            System.out.println(sc.next());       // print “one”
            System.out.println(sc.next());       // print “two”
            System.out.println(sc.next());       // print “hello”
            System.out.println(sc.next());       // print “world”
            System.out.println(sc.next());       // exception!
    }
}
```

- Note: use may change delimiters using method useDelimiter()

- Ref: http://download.oracle.com/javase/6/docs/api/java/util/Scanner.html

- Wrapper Classes: also in java.lang.*

  - Primitive data types (int, double, char, etc) are not class types

  - Java provides a wrapper class for each of the primitive data types

  - Example:

```
Integer myInt1= new Integer(10);     // construct Integer object
Integer myInt2= new Integer(“52”); // may also construct this way
myInt2.doubleValue();                // return a double 52.0
myInt1.equals(myInt2);               // compare two objects
```

  - Static constants and method

```
Integer.MAX_VALUE; Integer.MIN_VALUE;  // max and min of int
Integer.toString(42);   // return string “42”
```

  - Other wrapper classes : Double, Float, Character, Long, Boolean, etc

- Command line arguments

  public static void main(String[] args) // starting point of a program

  args : an array of Strings that contain the arguments

  example : A sample program which prints all input arguments.

```
jwong@thecity $ javac CmdLineArgs.java
jwong@thecity $ cat CmdLineArgs.java

        public class CmdLineArgs {
                public static void main(String[] args) {

                        // arguments are in array args
                        // print each argument in array args
                        for(int i=0; i < args.length; i++)
                                System.out.println( args[i] );
                }
        }

// example with three input arguments
jwong@thecity $ java CmdLineArgs hello world 123

hello
world
123

// example with four input arguments
jwong@thecity $  java CmdLineArgs abc def ghi 455

abc
def
ghi
455
```

- Simple GUI Program

  Example: For reading user input from keyboard and writing output to screen

```
// use Java Swing package
import javax.swing.JOptionPane;

public class SimpleIO {
        public static void main( String [] args) {

                // Display an input Dialog box and read input to readStr
                String readStr = JOptionPane.showInputDialog("Enter
                something:");

                // Display msg box
                JOptionPane.showMessageDialog(null,readStr);
        }
}
```