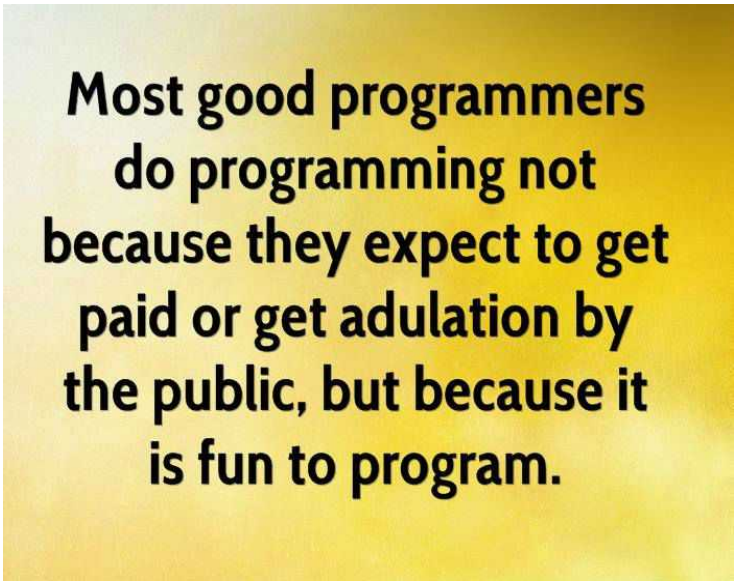
Keenan Knaur
Adjunct Lecturer

California State University, Los Angeles
Computer Science Department

Introduction to Programming



CS2011: Introduction to Programming I

Programming 101

So You Want to be a Programmer?!?

- Programmers are responsible for writing programs which make the computer hardware behave the way it does.
- Programmers are also responsible for testing, debugging, and maintaining the code that they write (or code that another person has written).
- Programmers are people who find creative solutions to real life problems.
- Programmers are "outside of the box" thinkers and people who are good at solving logic puzzles.

Learning to Program Takes Discipline

- Learning to be a good programmer is similar to learning how to play a musical instrument or learning how to play a sport.
  - What do both of these activities have in common?
    - PRACTICE, PRACTICE, PRACTICE
- Are you willing to spend countless hours working on computer code?
  - True Fact: Programmers spend most of their time debugging and testing code. The actual writing of the code is a very minimal fraction of the total time spent on any given programming project.
- Are you willing to put in time every day to get better?
  - This class will test your resolve and will prove to you whether or not you have the focus to become a good programmer.

Software and Programming Languages

Software (Programs)

- Instructions that tell the computer what to do.
  - Computers cannot think for themselves (yet).
  - Computers only do exactly what you tell them to do.
  - If the computer is behaving badly, it is because the code you wrote is bad!

Programming Langauges

- Software is written using ***computer languages***.
  - Languages which the computer can understand and which help to make writing programs easier.
  - Computers cannot understand human languages.
  - Programming in pure machine code is tedious.

Machine Langauge (Machine Code)

- Primitive instructions written in binary numbers.
  - 10101 101010101 111 1010101 101 01 010 1001010
  - This is the native language of all computers, and the only language that a computer can understand.

- Machine language is unique to each hardware configuration and must be tailored to that specific computer:
  - What works on one machine will not necessarily work on another.

- Imagine having to write computer programs using only 0's and 1's!

High-Level Languages

- First developed in the 1950s.
- ***Platform-independent***, you can write a program in a high-level language on one computer and run it on different types of machines.
- English-like, easier to learn and use.
- Instructions in a high-level programming language are called statements.
- Example: Compute the area of a circle with radius of 5:
  - area = 5 * 5 * 3.1415

Popular High-Level Languages

- Java (Obviously)**
- Python**
- C++**
- C#
- Kotlin
- Swift
- GoLang
- Any many many others.

- **NOTE: Java, Python, and C++ are what I call the **"Big Three"** languages. Master these languages before you graduate.

First Java Program

The Task

- Display the following message on the screen:

```
*** hello, world! ***
*** Welcome to CS 2011! ***
```

- Why “hello, world” ?
  - Common first time programming example for beginning programmers
  - See the article about Hello, World under the reading assignments for this week.

Writing and Editing a Java Program

- Basic Text Editor:
  - Notepad++
  - Sublime Text
  - Atom
  - Some Mac Equivalent Text Editor

- Required Software:
  - JDK Java Development Kit
    - Allows you to compile and execute Java programs.

How Do I Save My Files?

- All programs written in java are saved with the .java extension.
- The name of the file must exactly match the name of the class in the source code (more on this later).
- Save your files somewhere that is easy to navigate to:
  - Top level folder on a flash drive.
  - Top level folder on your c: or other harddrive.
  - Desktop (can be tricky to find).

The Source Code of Our Program

```
/*
  Author:  Keenan Knaur
  Purpose: Display a welcome message to the console.
*/

public class Hello {

  // method main(): application entry point
  public static void main(String[] args) {
    System.out.println("hello, world!");
    System.out.println("Welcome to CS 2011!");
  }

}
```

Comments

```
/*
  Author:  Keenan Knaur
  Purpose: Display a welcome message to the console.
*/

public class Hello {

  // method main(): application entry point
  public static void main(String[] args) {
    System.out.println("hello, world!");
    System.out.println("Welcome to CS 2011!");
  }

}
```

Comments

Comments

- Notes put into the source code by the programmer.
- Document what the program does and how sections of the program work.
- Ignored by the Java compiler
- Three types of comments:
  - **single-line comments**:
    - preceded by two slashes (//)
  - **multi-line comments**:
    - enclosed between /* and */ over one or multiple lines
  - **javadoc comments**:
    - Multi-line comment between /** and */.
    - used for documenting classes, data, and methods.
    - can be exported to a set of HTML files which make up the API of your program.

Class Header / Declaration

```
/*
  Author:  Keenan Knaur
  Purpose: Display a welcome message to the console.
*/
```

Class Header / Declaration

```
public class Hello {

  // method main(): application entry point
  public static void main(String[] args) {
    System.out.println("hello, world!");
    System.out.println("Welcome to CS 2011!");
  }

}
```

Class Body

Class Header / Declaration

- Generally there is one class per .java file.
- Every java program requires at least one class.

- **Class Header / Declaration Syntax**
  - **public**
    - this semester classes will always be declared public
  - **class**
    - required keyword used to identify where the class begins
  - **class name**
    - User specified
    - all classes must have a name
    - must be the same as the file name

- **Class Body**
  - Enclosed in a pair of { }

Class Name Rules

- Rules
  - Must start with a letter
  - Cannot conflict with any language keywords or symbols
  - Case-sensitive
- Conventions
  - Class names start with an upper-case letter
  - Multiple words are concatenated and every first letter of a word is capitalized.
  - The class name has to match exactly the name of the java file that contains the class
  - Example: I have a class called `WelcomeToJava` then it should be saved in a file called `WelcomeToJava.java`

The `main()` Method

```
/*
  Author:  Keenan Knaur
  Purpose: Display a welcome message to the console.
*/

public class Hello {

  // method main(): application entry point
  public static void main(String[] args) {
    System.out.println("hello, world!");
    System.out.println("Welcome to CS 2011!");
  }

}
```

main() Method Header

main() Method Body

The main() Method

- `main()` is a special method in Java

- Every Java program MUST have a main method inside one of its classes in order for the program to execute.
  - Programs can only have ONE main method

- The header of the main method is always written:
  - `public static void main(String[] args) { }`

Statements

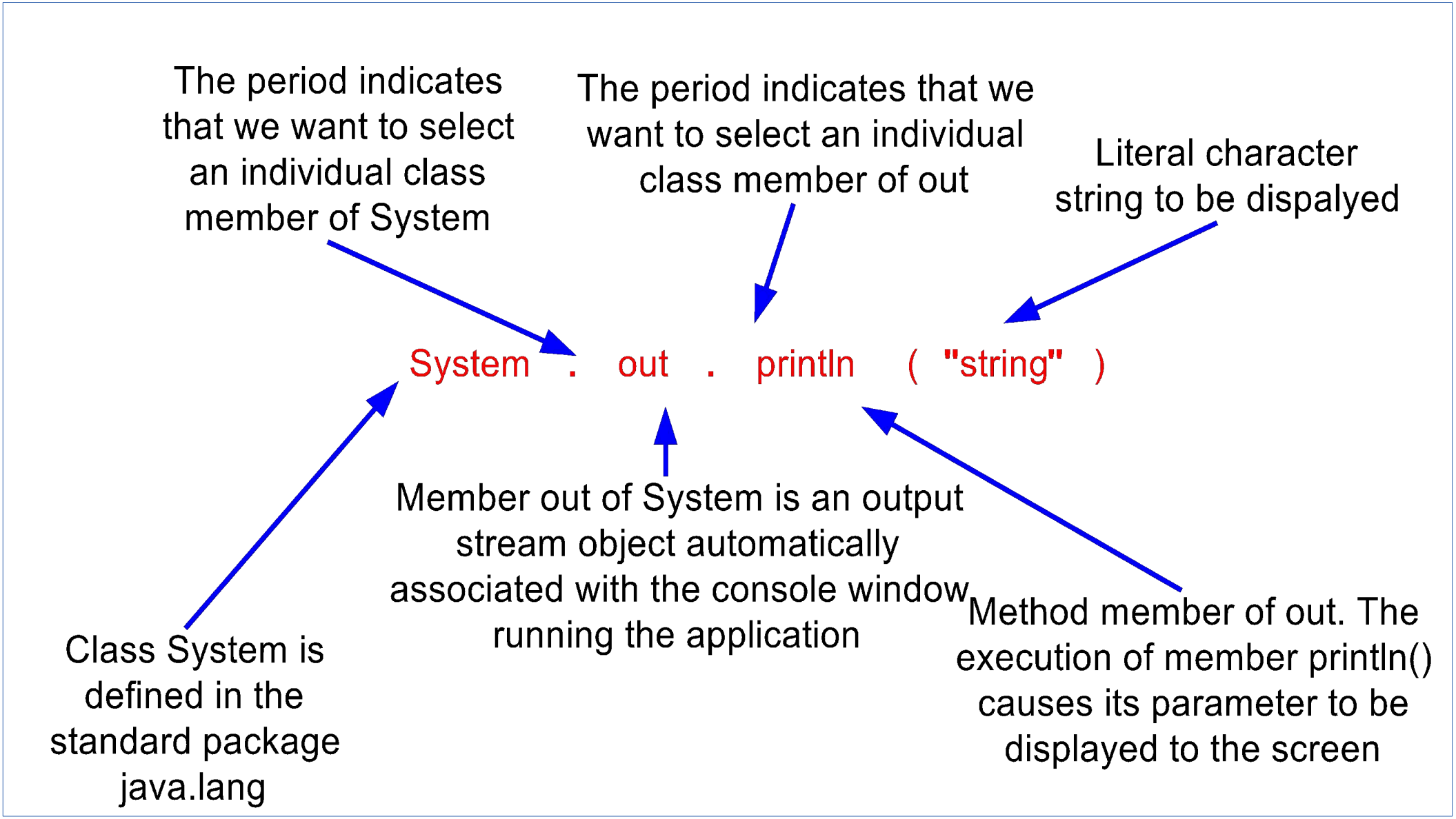
```
/*
  Author:  Keenan Knaur
  Purpose: Display a welcome message to the console.
*/

public class Hello {

  // method main(): application entry point
  public static void main(String[] args) {
    System.out.println("hello, world!");
    System.out.println("Welcome to CS 2011!");
  }

}
```

Statements

Statements

- Statements are the “sentences” in a programming language.
- They are an action or sequence of actions.
- Methods can contain many many statements.
- Ends in a semicolon (;) called the ***statement terminator***.

Printing To the Console



Reserved Words (Keyword)

```
/*
  Author:  Keenan Knaur
  Purpose: Display a welcome message to the console.
*/

public class Hello {

  // method main(): application entry point
  public static void main(String[] args) {
    System.out.println("hello, world!");
    System.out.println("Welcome to CS 2011!");
  }

}
```

Reserved Words

Reserved Words (Keywords)

- Words that have a specific meaning to the compiler.
- Cannot be used for any other purpose in the program.
- Are case sensitive:
  - i.e. it would be wrong to write `Public` instead of `public`.

Blocks of Code

```
/*
  Author:  Keenan Knaur
  Purpose: Display a welcome message to the console.
*/

public class Hello {

  // method main(): application entry point
  public static void main(String[] args) {
    System.out.println("hello, world!");
    System.out.println("Welcome to CS 2011!");
  }
}
```

Main Block

Class Block

Blocks of Code

- Pairs of curly braces form blocks of code that logically groups together parts of a program.
- A new block of code is always indented one more level to the right.
  - This is called ***nesting***.
- Every class has a class block that groups the data and methods of the class.
- Every method has a method block that groups the statements in a method.
- All opening braces MUST have a matching closing brace.

Brace Style

```
public class Test
{
  public static void main(String[] args)
  {
    System.out.println("next-line style");
  }
}
```

- next-line style:
  - aligns braces vertically making programs easy to read

```
public class Test {
  public static void main(String[] args) {
    System.out.println("end-of-line style");
  }
}
```

- end-of-line style:
  - saves space
  - may avoid some subtle programming errors.

NOPE NOPE NOPE

- Use whatever brace style you prefer
- But not this...
- Don't do this...
- Seek help instead of this...

```
public class Permuter                                    {
   private static void permutate(int n, char[] a)        {
      if (n == 0)                                        {
         System.out.println(String.valueOf(a))           ;}
      else                                               {
         for (int i = 0; i <= n; i++)                    {
            permutate(n-1, a)                            ;
            swap(a, n % 2 == 0 ? i : 0, n)               ;}}}
   private static void swap(char[] a, int i, int j)      {
      char saved = a[i]                                  ;
      a[i] = a[j]                                        ;
      a[j] = saved                                       ;}}
```

Proper Spacing and Indentation

- Good indentation makes a program easier to read, debug, and maintain.

- Proper spacing between statement components should be used to also make reading the program easier
  - `System.out.println(3+4*4);` is ok
  - `System.out.println(3 + 4 * 4);` is better

Translating Source Code to Machine Code

JVM Java Virtual Machine

- The Java Virtual Machine is what makes Java programs platform-independent

- Write Once, Run Anywhere:
  - easy to run the same Java code in UNIX/OSX/Windows/ etc.
  - reduces the amount of knowledge the programmer needs to have about the specific platform.
  - adds long-term robustness. Your Java code will still run in Windows 18.7 as long as there is a JVM that can run it.

Translating High-Level Langauges to Machine Language

- A high-level language program is called a ***source program*** or ***source code***.
- Source code is not directly understood by computers.
- Source code must be translated into machine code.

Translating High-Level Langauges to Machine Language

- Translation is done using one (or both) of the following techniques:
  - ***interpreting***:
    - performed by software called an interpreter
    - reads one statement at a time from the source code at a time, translates it into machine code then executes it right away.
  - ***compiling***:
    - performed by software called a compiler.
    - translates the entire source code into a machine-code file.
    - the machine-code file is then executed.

Translating Java to Machine Langauge

- Java uses both an ***interpreter*** and a ***compiler***.
- The Java source code is compiled before runtime into an intermediate language called Java ***bytecode***.
- The ***Java Virtual Machine*** interprets bytecode to machine language at runtime.
-
- Each platform (operating system or hardware type) requires a different type of JVM, but all forms of the JVM run the same bytecode.

Command-Line Compiling

Directories

- In DOS, folders are known as directories
  - directories hold files
  - you can create or delete them at will
- All directories are organized in a hierarchical way
- The topmost directory is called the root directory c:\>
- The root directory contains many sub-directories
- Each sub-directory may contain many sub-sub-directories

View the Contents of a Directory

- To view the contents of the current directory that you are in, use the **`dir`** command

Command Prompt

```
C:\tmp>dir
 Volume in drive C has no label.
 Volume Serial Number is 30B2-A70D

 Directory of C:\tmp

09/21/2006  05:18 PM    <DIR>          .
09/21/2006  05:18 PM    <DIR>          ..
09/14/2006  01:37 PM            56,832 mydoc.doc
09/21/2006  05:17 PM    <DIR>          myjava
               1 File(s)         56,832 bytes
               3 Dir(s)  18,055,290,880 bytes free

C:\tmp>
```

Changing Directories

- to change directories used the **`cd`** command

- change to a sub-directory of the current directory
  - Command: `cd sub-directory-name`
  - Example: `cd myjava`

- Change to the parent directory
  - Command: `cd ..`

- Change to the root directory from any directory
  - Command: `cd \`

Changing Directories

Command Prompt

```
C:\tmp>dir
 Volume in drive C has no label.
 Volume Serial Number is 30B2-A70D

 Directory of C:\tmp

09/21/2006  05:18 PM    <DIR>          .
09/21/2006  05:18 PM    <DIR>          ..
09/14/2006  01:37 PM            56,832 mydoc.doc
09/21/2006  05:17 PM    <DIR>          myjava
               1 File(s)         56,832 bytes
               3 Dir(s)  18,055,290,880 bytes free

C:\tmp>cd myjava

C:\tmp\myjava>cd ..

C:\tmp>cd \

C:\>cd tmp

C:\tmp>_
```

Compiling Your First Java Program

- Navigate to the directory (folder) where your java files are located.
  - For Example: lets say your directory structure looked something like this:

```
c:>
  my_programs
    MyProgram.java
```

- navigate into my_programs
  - `cd my_programs`
- once in the folder use the javac command to compile your program
  - `javac MyProgram.java`
  - notice that the extension must be part of the file name when you are using the javac command
- After you compile the program you will either see some errors, or the compiler will generate a .class file with the same name as your program i.e. `myProgram.class`.
- To run your program use the java command
  - `java MyProgram`
  - notice that you DO NOT add the file extension when running your program.

Programming Errors

Syntax Errors (Compile Errors)

- Errors detected by the compiler which result from incorrect code construction.

- Prevent the compiler from properly compiling the program and the program will not execute.

- Examples include:
  - mistyping a keyword,
  - omitting some necessary punctuation,
  - missing matching braces, etc.

Run-time Errors

- Errors that cause a program to terminate abnormally.

- The compiler is able to compile and execute the program, but during execution the program will stop unexpectedly if the JVM detects an operation that is impossible to carry out.

- Examples include
  - input mistakes
  - division by zero

Logic Errors

- Errors that occur when a program does not perform the way it was intended to.

- Can be hard to detect because the fault lies in the user's logic (thought processes) behind the program.

- For example: Suppose you write a program to calculate the area of a circle and instead of using the formula 2 * π * r you mistakenly use 3 * π * r. This would cause a logic error because the program will execute and finish and the compiler will give no errors but the logic behind the formula is faulty which results in the incorrect output.

References

- Liang, Chapter 01: Introduction to Computers Programs, and Java

- Braces joke from Ministry of Dev, PhD @UdellGames