

Java Language and Software Development

- Programming Languages
- Java Program Structure
- Problem Solving
- Object-Oriented Programming
- Reading for this class: L&L, 1.4-1.6

What is a program?

- It consists of two components:
 - Data (numbers, characters, true/false)
 - Steps
- A program goes through a number of steps with pieces of data to achieve a result:
 - Printing text to screen
 - Collecting information
 - Performing calculations
- Example: Long Division

Programming Languages

- Computer programmers write programs for computers using one or more programming languages
- Some languages are better for one type of program or one style of user interface than for others
- You may have heard of some programming languages: COBOL, Basic, Pascal, C/C++, Java, Assembly Language, and Others

“Hello, World” Versions

- Java

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

- Basic: 10 PRINT "HELLO WORLD"

- Fortran: PROGRAM HELLOWORLD

```
    10 FORMAT (1X,11HHELLO WORLD)  
    WRITE(6,10)  
    END "HELLO WORLD"
```

- Ruby: puts“Hello World”

- C

```
#include <stdio.h>  
#include <stdlib.h>  
  
int main(void)  
{  
    printf("Hello, world\n");  
    return EXIT_SUCCESS;  
}
```

- **Scheme:**

(display "Hello, World!")
(newline)

Programming Languages

- A *programming language* specifies the words and symbols that we can use to write a program
- A programming language employs a set of rules that dictate how the words and symbols can be put together to form valid *program statements*
- A programming language has both *syntax* and *semantics*

Syntax and Semantics

- The *syntax rules* of a language define how we can put together symbols, reserved words, and identifiers to make a valid program
- The *semantics* of a program statement define what that statement means (its purpose or role in a program)
- A program that is syntactically correct is not necessarily logically (semantically) correct
- A program will always do what we tell it to do, not what we meant to tell it to do

Syntax vs. Semantics

- **Everyday Language:**
 - Incorrect syntax: Ball the red is color the
 - Incorrect semantics: The ball is the color three
 - Alternative (not incorrect) syntax:
"Finish your training, you must" – Yoda
- **Programming:**
 - Correct: `int i = 34;`
 - Incorrect semantics: `int i = "foobar";`
 - Incorrect syntax: `i = 34`
**However, the last example would be correct in another language – e.g., Ruby*

Language Levels

- There are four programming language levels:
 - machine language
 - assembly language
 - high-level language
 - fourth-generation language
- Each type of CPU has its own specific *machine language*
- The other levels were created to make it easier for a human being to read and write programs

Machine Language



See
demo...

Programming Languages

- A program must be translated into machine language before it can be executed
- A *compiler* is a software tool which translates *source code* into a specific target language
- Often, that target language is the machine language for a particular type of CPU
- The Java approach is somewhat different: Incorporates an intermediate step

Java Translation

- The Java compiler translates Java source code into a new representation called *bytecode* in the `.class` file

FooBar.java → **Compiler** → **FooBar.class**

- A specific machine's *interpreter* program, called the Java Virtual Machine (JVM), reads bytecode and executes machine-comprehensible instructions

FooBar.class (bytecode) → JVM → Program runs!

- Java programs can be run on any machine that has a JVM, the latter of which fills the gap between high-level Java language and the computer's machine code.

Java Program Structure

- In the Java programming language:
 - A program is made up of one or more *classes*
 - A class contains zero or more *attributes*
 - A class contains one or more *methods*
 - A method contains program *statements*
- These terms will be explored in some detail throughout the course, less so in CS/IT114
- A Java application starts with a class containing a method called `main`
- See `Lincoln.java` (page 29)

Basic Definitions

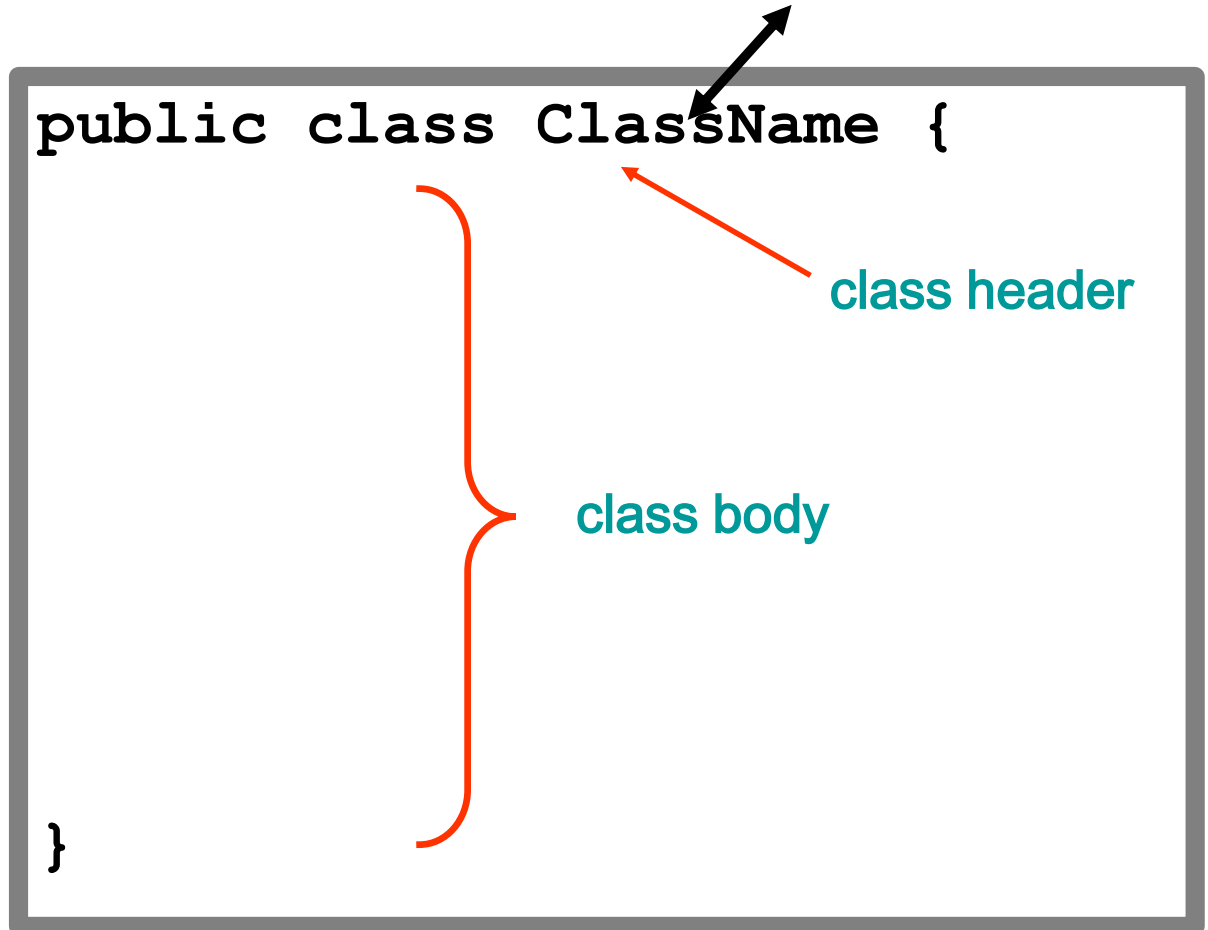
- *Class*: Primary building block of a Java program (oversimplified, but adequate for our purposes)
- *Attribute*: A piece of information about some Java entity.
- *Method*: A named set of instructions, relative to some Java entity, to accomplish a common goal. (We will be using methods, and writing some, to an extent)
- *Statement*: A piece of code representing a complete step in a program - usually, but not always, ends in a semicolon (;)

Java Program Structure

- For now, each program of ours will consist of a single class, which has a unique name – for example, ClassName
- It is represented by a file called ClassName.class
- We will obtain that file by compiling a program file that **we** write, called ClassName.java
- Writing these program files so that they compile successfully (and the program runs in the intended manner) will require ***meticulous*** attention to detail.

Java Program Structure

First, we have the outer code of ClassName.java



Java Program Structure


Then, we have the holder for your program code:

```
public class ClassName {
```

```
    public static void main (String[] args) {
```

```
    }
```

```
}
```



notice the
indentation
relative to the
class header?

Java Program Structure

Finally, we have the program code itself:

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

Comments

- Comments in a program are called *inline documentation*
- They should be included to explain the purpose of the program and describe processing steps
- They do not affect how a program works
- Java comments can take three forms:

```
// this comment runs to the end of the line
```

```
/* this comment runs to the terminating  
   symbol, even across line breaks */
```

```
/** this is a javadoc comment */  
(don't need this yet)
```

```
/*  
    This is the Hello code  
*/  
public class Hello { // class header  
    public /* ... */ static void main (String[] args) {  
        System.out.println("Hello, world!");  
    }  

```

But, don't comment out code
you want to keep!

```
System.out.// a statement println("Hello, world!");
```

Error!

Identifiers

- *Identifiers* are the words a programmer uses in a program
- Rules:
 - Can be made up of letters, digits, the underscore character (`_`), and the dollar sign
 - Identifiers cannot begin with a digit
 - *Case sensitive* - Total, total, and TOTAL are different identifiers
- By convention, programmers use different case styles for different types of identifiers:
 - *title case* for class names – Lincoln
 - *lower case* for object or other variable names – currentTemperature, limit
 - *upper case* for constants – MAXIMUM

Identifiers

- Sometimes we choose identifiers ourselves when writing a program (such as `Lincoln`)
- Sometimes we are using another programmer's code, so we use the identifiers that he or she chose (such as `println`)
- Often we use special identifiers called *reserved words* that already have a predefined meaning in the language
- A reserved word cannot be used in any other way

Reserved Words

- The Java reserved words:

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

Hello.java

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

Identifiers:

public	void	System
class	main	out
Hello	String	println
static	args	

White Space

- Spaces, blank lines, and tabs are called *white space*
- White space is used to separate words and symbols in a program. Extra white space is ignored
- A valid Java program can be formatted many ways
- Programs should be formatted to enhance readability, using consistent indentation
- See `Lincoln2.java` (page 34)
- See `Lincoln3.java` (page 35)


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

System.out.println("Hello, world!");    }    }
```

This would be valid Java code
and compile just fine...**but**
please don't! :-)

Formatting Poorly

```
public class Foo{public static void main(String[] args){String
  foo1="Barack Obama";String foo2="President";String foo3="The
  United States of America"; System.out.println(foo1+" is the "+foo2+"
  of "+foo3+".");}}
```

It compiles and runs fine, so what's wrong here?

- Hard to read (No use of spacing, indentation, tabs)
- Meaningless identifiers
- No commentary

Truly, a nightmare come true – for the next person who has to maintain this code!

Formatting Well

```
public class PoliticianPrinter {  
  
    public static void main(String[] args) {  
  
        // Intialize String variables with information  
        String personName = "Barack Obama";  
        String personTitle = "President";  
        String countryName="The United States of America";  
  
        // Print the string  
        System.out.println(personName + " is the " + personTitle + " of " +  
            countryName + ".");  
    }  
}
```

**"Always code as if the person who ends up
maintaining your code will be a violent
psychopath who knows where you live."**

-- Martin Golding

Grouping Statements

- Remember, `main` could have had as many statements as we wanted
- We grouped the statements informally, in the previous example, using white space
- Sometimes we may group them more formally, in the form of a method, which provides for two benefits:
 - Conciseness: An individual segment of code can be shorter because many commands are replaced with a single command
 - Reuse: Do not have to rewrite the same statements over and over

Example: DrawFigures

- Recall DrawFigures1
 - Many lines - "spaghetti code"
 - Tedious (and potentially confusing) to read
- Then, DrawFigures2
 - Better, because it breaks the code down into related chunks and places them into methods
 - The code inside main goes from 30 lines to 3!
- But, ***DrawFigures3*** is better still
 - Take repeated blocks of code to make even more methods
 - Thus, methods from DrawFigures2 are much simpler than before

Problem Solving

- The purpose of writing a program is to solve a problem
- Solving a problem consists of multiple activities:
 - Understand the problem
 - Design a solution
 - Consider alternatives and refine the solution
 - Implement the solution
 - Test the solution
- These activities are not purely linear – they overlap and interact (*for example, see the "iterative" diagram from the previous lecture*)

Problem Solving

- The key to designing a solution is breaking it down into manageable pieces
- When writing software, we design separate pieces that are responsible for certain parts of the solution
- *An object-oriented approach* lends itself to this kind of solution decomposition
- We will dissect our solutions into pieces called objects and classes

Object-Oriented Programming

- Java is an object-oriented programming language
- As the term implies, an *object* is a fundamental entity in a Java program
- Objects can be used effectively to represent real-world entities
- For instance, an object might represent a bank account
- Each bank account object handles the processing and data management related to that bank account

Objects

- An object has:
 - *state* - descriptive characteristics (variable values)
 - *behaviors* - what it can do (or what can be done to it)
- The state of a bank account includes its balance
- The behaviors associated with a bank account include the ability to get the balance, make deposits, and make withdrawals
- Note that the behavior of an object might change its state, e.g. making a deposit will increase the balance

Classes

Class (The Conceptual)

- Is like a blueprint for...
- Has *attributes* that...
- Has *methods* that...
- The class that contains the `main` method represents the starting point for a Java program
- The program can and usually does contain more classes than just the one that contains the `main` method

Object (The Concrete)

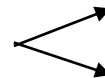
- An object
- Define the *state* of each object
- Define the *behavior* of each object

Objects and Classes

**A Class
(The Concept)**

BankAccount
- balance: float
+ getBalance(): float + deposit(float amount): bool + withdraw(float amount): bool

**Multiple objects
of the same class**



**Three objects
(Three Instances
of the Concept)**

John's Bank Account
Balance: \$5,257.51

Bill's Bank Account
Balance: \$1,245,069.89

Mary's Bank Account
Balance: \$16,833.27

Java Program Structure

```
public class BankAccount
{
    private float balance;           attribute definition

    public float getBalance()
    {
        method body
    }
    public boolean deposit(float amount)
    {
        method body
    }
    public boolean withdraw(float amount)
    {
        method body
    }
}
```