Readings
Your textbook is *Big Java* (3rd Ed).

This Week’s Reading: Ch 2.1-2.5, Ch 4.1-4.2.

Labs and Tutorials
Labs and tutorials start this week!

You MUST be enrolled in a lab section.

If the lab sections you need are full, go to ICICS Room 201 and see a Computer Science undergrad advisor!

Learning Goals
By the end of the next several lectures you will be able to...

- Apply with basic competence simple programming constructs such as sequential execution, variable typing and declaration, naming, algebraic operations, operation precedence.
- Create programs which translate explicit English problem statements (an algorithm) into short series of sequential Java instructions.

Learning Goals
By the end of class today you will be able to...

- Describe the Java rules for identifiers and reserved words. Recognize legal and illegal identifiers.
- Explain what a variable is, and what a variable declaration does.
- Declare Java variables of type `int`.
- Recognize and write simple assignment statements.
- Trace the execution of a sequence of simple assignment statements.

Last Time...
- We looked at how Java programs get compiled (javac) into a .class file, which we then interpret (java).
- We looked at the building blocks of a Java program: white space, comments, strings, identifiers, numbers, curly braces, operators.
- We ran and experimented with a simple Java program. We saw how Java executes statements one-by-one, in order.
- We started studying rules for identifiers.
Identifiers

- Identifier must
  - Start with a letter and be followed by
  - Zero or more letters and/or digits
    - Digits are 0 through 9.
    - Letters are
      - the 26 lowercase letters in English alphabet
      - the 26 uppercase letters in English alphabet
      - plus the $ and _
      - also alphabetic characters from other languages

Which of the following are not valid identifiers?

- userName
- user_name
- $cash
- 2ndName
- firstName
- user.age
- _note_
- note2

Identifiers

- Java is case sensitive
- Oreo
- oreo
- OREO
- oreo

are all different identifiers, so be careful

common source of errors in programming

Identifiers

- Creating identifiers in your Java programs
  - Remember other people read what you create
  - Make identifiers meaningful and descriptive for both you and them
  - No limit to how many characters you can put in your identifiers
  - but don’t get carried away

```java
public class ReallyLongNamesWillDriveYouCrazyIfYouSoOverboard {
    public static void main (String[] args) {
        System.out.println("Enough already!");
    }
}
```
Laws versus Customs/Conventions

- In real life, some things are illegal: murder, shoplifting, running stop signs, etc.
- Other things are legal, but social norms dictate that we don’t do them, because life is better for everyone if we agree to follow the rules: cutting in line, talking on your cell phone in class/movie/concert, being rude, etc.

Java has conventions, too.

- Rules for identifiers are like laws: If you break them, the compiler won’t let your program compile:
  - E.g.: `3am` is not a legal identifier.
- Java also has conventions. These are the normal way Java programmers do things. It’s good to follow them. E.g.:
  - CamelCase: Makes it easier to read.
  - Don’t start identifiers with `$` or `_`
  - We’ll learn more as the course goes on…

Who cares about identifiers?

- Identifiers are used in Java to name things (to “identify” them).
- Why do we name things?

Who cares about identifiers?

- Identifiers are used in Java to name things (to “identify” them).
- Why do we name things?
  - To make it easy to refer to them.
  
- In computer programming, a fundamental trick is to name a chunk of memory, so you can store data there and find it easily.

Memory and Identifiers

- Example of a high-level instruction
  - `a = b + c;`
  - Tells computer to
    - go to main memory and find value stored in location called `b`
    - go to main memory and find value stored in location called `c`
    - add those two values together
    - store result in memory in location called `a`
  - Great! But... in reality, locations in memory are not actually called things like `a`, `b`, and `c`.

Memory Recap

- Memory: series of locations, each having a unique address, used to store programs and data
- When data is stored in a memory location, previously stored data is overwritten and destroyed
- Each memory location stores one byte (8 bits) of data

*For total accuracy, these addresses should be binary numbers, but you get the idea, no?*
Memory and Identifiers

- So what’s with the a, b, and c?
- Machine language uses actual addresses for memory locations
- High-level languages easier
  - Avoid having to remember actual addresses
  - Invent meaningful identifiers giving names to memory locations where important information is stored
- payRate and hoursWorked vs. 5802 and 5806
- Easier to remember and a whole lot less confusing!

Memory and Identifiers: Variables

- **Variable**: name for location in memory where data is stored
  - A little like variables in algebra class
- payRate, hoursWorked, a, b, and c are all variables
- Variable names begin with lower case letters
  - Java convention, not compiler/syntax requirement
- Variable may be name of single byte in memory or may refer to a block of many bytes
  - More about this later...

Programming With Variables

```java
public class Test
{
    public static void main (String[] args)
    {
        a = b + c;
        System.out.print ("The answer is ");
        System.out.println (a);
    }
}
```

Programming With Variables (Shorthand)

```java
public class Test
{
    public static void main (String[] args)
    {
        a = b + c;
        System.out.println ("The answer is " + a);
    }
}
```

Programming With Variables

```java
public class Test
{
    public static void main (String[] args)
    {
        a = b + c;
        System.out.println ("The answer is " + a);
    }
}
```

Programming With Variables: Take 2

```java
public class Test2
{
    public static void main (String[] args)
    {
        b = 3;
        c = 5;
        a = b + c;
        System.out.println ("The answer is " + a);
    }
}
```

Let’s give it a try...
- b and c cannot be found!
- need to assign values
Data Types

- Java requires that we tell it what kind of data it is working with.
- For every variable, we have to declare a data type.
- Java language provides eight primitive data types:
  - number (short, int, long, float, double)
  - boolean
  - character
- For more complicated things, can use data types created by others provided to us through the Java libraries.
- We want \( a \), \( b \), and \( c \) to be integers. Here's how we do it...

Variable Declaration

- variable declaration is instruction to compiler
  - reserve block of main memory large enough to store data type specified in declaration.
- variable name is specified by identifier.
- syntax:
  - \( \text{typeName variableName}; \)

Assignment

- \( a \), \( b \), \( c \): initializations and assignments
- \( a \) = \( b \) + \( c \)
Assignment Statements
- Assignment statement assigns value to variable
- Assignment statement is
  - identifier
  - followed by assignment operator (=)
  - followed by expression
  - followed by semicolon (;)

b = 3;
c = 8;
a = b + c;
weekly_pay = pay_rate * hours_worked;

Note that = is not a test for equality!

---

Assignment Statements
- Java first computes value on right side
- Then assigns value to variable given on left side

x = 4 + 7;   // what's in x?

Old value will be overwritten if variable was
assigned before

x = 2 + 1;   // what's in x now?

---

Assignment Statements
- Note that = is not a test for equality!
- Assignment is a new concept in CS versus
  math.
  - Fetch-decode-execute
  - Dynamic. Something happens.
    x = x + 1;

Read = as "gets" (short for "gets assigned the
value").

---

Assignment Statements
- Here's an occasional point of confusion:
  - a = 7;       // what's in a?
  - b = a;       // what's in b?
  - System.out.println("a is " + a + "b is " +b);

// what's in a now???

Variable values on left of = are clobbered
Variable values on right of = are unchanged
  - copy of value assigned to a also assigned to b
  - but that doesn't change value assigned to a
Assignment Statements

- Here’s an occasional point of confusion:
  
  ```java
  a = 7;         // what's in a?
  b = a;         // what's in b?
  // what's in a now???
  System.out.println("a is " + a + "b is " +b);
  a = 8;
  System.out.println("a is " + a + "b is " +b);
  ```

- Memory locations a and b are distinct
  - value of a at that point assigned to b
  - changing a later does not affect previous copy

Tracing Assignment Statements

- Use your finger to follow program execution.
- Make a little table of variables to track values

```java
a = 7;         // what's in a?
b = a;         // what's in b?
// what's in a now???
System.out.println("a is " + a + "b is " +b);
a = 8;
System.out.println("a is " + a + "b is " +b);
```

- Variables:
  - a:
  - b:

```
Tracing Assignment Statements

- Use your finger to follow program execution.
- Make a little table of variables to track values

```
```
```
```
```
Tracing Assignment Statements

- Use your finger to follow program execution.
- Make a little table of variables to track values

\[
\begin{align*}
\text{a} &= 7; & \text{// what's in a?} \\
\text{b} &= \text{a}; & \text{// what's in b?} \\
\text{System.out.println(} \text{"a is " + a + \" b is \" +b);} \\
\text{a} &= 8; \\
\text{System.out.println(} \text{"a is " + a + \" b is \" +b);}
\end{align*}
\]

Variables:
- a: 8
- b: 7

Questions?

Recap: Variable Declaration

- variable declaration is instruction to compiler
  - reserve block of main memory large enough to store data type specified in declaration
- variable name is specified by identifier
- syntax:
  - \texttt{typeName variableName;}
  - int a
Data Types

- For every variable, we have to declare a data type
- Java language provides eight primitive data types
  - i.e. simple, fundamental
- For more complicated things, can use data types
  - created by others provided to us through the Java libraries
  - that we invent
  - More soon - today, let’s stay with the primitives

Primitive Data Types: Non-numeric

- Character Type
  - named char
  - Java uses the Unicode character set so each char occupies 2 bytes of memory.
- Boolean Type
  - named boolean
  - Variables of type boolean have only two valid values
    - true and false
  - Often represents whether particular condition is true
  - Ignore these for now. We’ll see them again later.

Primitive Data Types: Numbers

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>1 byte</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>2 bytes</td>
<td>-32,768</td>
<td>32,767</td>
</tr>
<tr>
<td>int</td>
<td>4 bytes</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>float</td>
<td>4 bytes</td>
<td>approx -3.4E38 (7 sig. digits)</td>
<td>approx 3.4E38 (7 sig. digits)</td>
</tr>
<tr>
<td>double</td>
<td>8 bytes</td>
<td>approx -1.7E308 (15 sig. digits)</td>
<td>approx 1.7E308 (15 sig. digits)</td>
</tr>
</tbody>
</table>

- Six primitives for numbers
  - fixed size, so finite capacity
  - integer vs. floating point

Data Type Sizes

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>4 bytes</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>double</td>
<td>8 bytes</td>
<td>approx -1.7E308 (15 sig. digits)</td>
<td>approx 1.7E308 (15 sig. digits)</td>
</tr>
</tbody>
</table>

- fixed size, so finite capacity

Floating Point Numbers

- significant digits
  - 42
  - 4.2
  - 420000000
  - .000042

Floating Point Numbers

- significant digits
  - 42 = $4.2 \times 10 = 4.2 \times 10^1$
  - 4.2 = $4.2 \times 1 = 4.2 \times 10^0$
  - 42000000 = $4.2 \times 10000000 = 4.2 \times 10^7$
  - .000042 = $4.2 \times .00001 = 4.2 \times 10^{-5}$
Floating Point Numbers
- significant digits
  - $42 = 4.2 \times 10^1$
  - $4.2 = 4.2 \times 10^0$
  - $42000000 = 4.2 \times 10^7$
  - $0.00042 = 4.2 \times 10^{-5}$
- only need to remember
  - nonzero digits
  - where to put the decimal point
    - floats around when multiply/divide by 10
      - (all in binary inside computer)
    - enormous range, but limited precision

Data Types: Int and Double
- int
  - integer
  - 4 bytes, about -2 billion to 2 billion
- double
  - real number
  - (double-precision floating point)
  - 8 bytes, 15 sig figs, humongous range
  - (Number systems briefly explained in Appendix K)

Primitive Data Types: Numbers

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>1 byte</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>2 bytes</td>
<td>-32,768</td>
<td>32,767</td>
</tr>
<tr>
<td>int</td>
<td>4 bytes</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>float</td>
<td>4 bytes</td>
<td>approx -3.4E38 (7 sig.digits)</td>
<td>approx 3.4E38 (7 sig.digits)</td>
</tr>
<tr>
<td>double</td>
<td>8 bytes</td>
<td>approx -1.7E308 (15 sig. digits)</td>
<td>approx 1.7E308 (15 sig. digits)</td>
</tr>
</tbody>
</table>

- Primary primitives are **int** and **double**
  - Just worry about those for now
  - Don’t need to memorize exact limits, but know roughly what the limits are.

Variable Declaration Examples
- person’s age in years
- height of mountain to nearest meter
- length of bacterium in centimeters
- number of pets at home

Recap: Assignment Statements
- Assignment statement assigns value to variable
- Assignment statement is
  - identifier
  - followed by assignment operator (=)
  - followed by expression
  - followed by semicolon (;)

```
  b = 3;
c = 8;
a = b + c;
weekly_pay = pay_rate * hours_worked;
```
- Note that = is not a test for equality!

Recap: Assignment Statements
- Assignment statement assigns value to variable
- Assignment statement is
  - identifier
  - followed by assignment operator (=)
  - followed by expression
  - followed by semicolon (;)

```
  b = 3;
c = 8;
a = b + c;
weekly_pay = pay_rate * hours_worked;
```
- Note that = is not a test for equality!
Expressions

- expression is combination of
  - one or more operators and operands
  - operator examples: +, *, /, ...
  - operand examples: numbers, variables, ...
  - usually performs a calculation
    - don’t have to be arithmetic but often are
  - examples
    
    3
    7 + 2
    7 + 2 * 5
    (7 + 2) * 5

Arithmetic Operators

- +  addition
  - Works on int, double, byte, short, long, ...
- -  subtraction
  - Works on all numeric types, too
- *  multiplication
  - Didn’t have times sign on keyboard
  - Works on all numeric types, too

More Arithmetic Operators

- /  division
  - Integer division on integer types!
  - Example: 13 / 5 results in 2
  - Just like before you learned fractions
  - Normal division on double and float
- %  remainder (aka "mod")
  - Only works on integer types
  - Example: 13 % 5 results in 3

(Aside: Operator Overloading)

- Hmm… the same symbol / can do different things for ints and doubles:
  - 13/5 results in 2 (the type is int)
  - 13.0/5.0 results in 2.6 (the type is double)
- Similar for other operators, e.g., +
  - 13+5 is 18 (18 is an int)
  - 13.0+5.0 is 18.0 (18.0 is a double)
  - "13"+"5" is "135" ("135" is a String)!

Operator Precedence

- What does this expression evaluate to?
  
  7 + 2 * 5
Operator Precedence

- What does this expression evaluate to?
  $7 + 2 \times 5$
- Multiplication has higher operator precedence than addition (just like in algebra)

<table>
<thead>
<tr>
<th>precedence</th>
<th>operator</th>
<th>operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 higher</td>
<td>+ -</td>
<td>unary plus and minus</td>
</tr>
<tr>
<td>2</td>
<td>* / %</td>
<td>multiply, divide, remainder</td>
</tr>
<tr>
<td>3 lower</td>
<td>+ -</td>
<td>add, subtract</td>
</tr>
</tbody>
</table>

Operator Precedence

- What does this expression evaluate to?
  $7 + 2 \times 5$
- Multiplication has higher operator precedence than addition (just like in algebra)

<table>
<thead>
<tr>
<th>precedence</th>
<th>operator</th>
<th>operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 higher</td>
<td>+ -</td>
<td>unary plus and minus</td>
</tr>
<tr>
<td>2</td>
<td>* / %</td>
<td>multiply, divide, remainder</td>
</tr>
<tr>
<td>3 lower</td>
<td>+ -</td>
<td>add, subtract</td>
</tr>
</tbody>
</table>

Use parentheses to change precedence order or just clarify intent

$(7 + 2) \times 5$  $7 + (2 \times 5)$

Associativity

- What about this?
  $7 - 5 - 3$
- The result is -1
  - $(7 - 5) - 3$, not $7 - (5 - 3)$
  - Arithmetic operators of same precedence are left associative
  - Matters for some operators; doesn’t for others
  - Use parentheses to be clear!
  - (Operators and precedence in Appendix E)

Shorthand: Variable Declaration and Assignment

- **Variable declaration** reserves a chunk of memory and gives it a name.
- **Assignment** sticks a value into the variable.
- Can do both in one statement:
  - `typeName variableName;`
  - `variableName = value;`
  - `typeName variableName = value;`
    - can declare and assign in one step

Questions?