Topics for today

- Introduce Java programming language
  - Assignment and type casting
  - Expressions
  - Operator precedence
- Code example: Hamilton Lyrics
- If we have time: Two new kinds of operators:
  - RELATIONAL operators (> , <, ==, !=, >=, <=)
  - LOGICAL operators (&&, ||, !)
- A new data type: boolean
Review: Assignment

- Assignment uses =, but it is not an algebraic equation

= means, "store the value at right in the variable at left"

- The right side expression is evaluated first, and then its result is stored in the variable at left.

- What happens here?

int x = 3;
x = x + 2; // ???

(A) x
    x+2

(B) x
    2

(C) x
    3

(D) Other or causes an error
Assignment

- Assignment uses = symbol that you’ve seen in math class, but it is not an algebraic equation in the sense that you’ve seen in math class

  = means, "store the value at right in the variable at left"

- The right side expression is evaluated first, and then its result is stored in the variable at left.

- What happens here?

```c
int x = 3;
x = x + 2;    // ???
```
Compiler errors with variables

- A variable can't be used until it is assigned a value.

```java
int x;
println(x);  // Error: x has no value
```

- You may not declare the same variable twice.

```java
int x;
int x;  // ERROR: x already exists
int y = 3;
int y = 5;  // Error: y already exists
```

- How can this code be fixed?
Operators and Expressions

Type int
Expressions, operators

- **expression**: A value or operation that computes a value.
  
  **Examples**: 
  
  \[
  1 + 4 * 5 \\
  (7 + 2) * 6 / 3
  \]

  - The simplest expression is a *literal value*, like 42.
  - A complex expression can use operators and parentheses.

- **operator**: Combines multiple values or expressions.
  
  - \(+\) - \(-\) * / %
  
  - *add*, *subtract/negate*, *multiply*, *divide*, *modulus* (remainder)

- As a program runs, its expressions are *evaluated*.
  
  1 + 1 evaluates to 2  . (How would we print the output 1 + 1?)
Integer division and remainder

- When we divide integers, the quotient an integer
  
  \[
  14 \div 4 = 3, \text{ not } 3.5. \quad (\text{Java ALWAYS rounds down.})
  \]

\[
\begin{array}{ccc}
3 & 4 & 52 \\
14 & 45 & 1425 \\
12 & 40 & 135 \\
2 & 5 & 75 \\
2 & 54 & 21
\end{array}
\]

- To access the remainder portion, use % operator:
  
  \[
  14 \% 4 \quad \text{is} \quad 2
  \]
  
  \[
  45 \% 10 \quad \text{is} \quad 5
  \]
  
  \[
  1425 \div 27 \quad \text{is} \quad \ldots \quad ? \quad (\text{your turn})
  \]
Operators and Expressions

Type double
Real numbers: double

- Examples: 6.022, -42.0, 2.143e17
  - Placing .0 or . after an integer makes it a double.

- The operators + - * / % () all still work with double.
  - / produces an exact* answer: 15.0 / 2.0 is 7.5
  - Precedence is the same: () before * / % before + -

* to a certain number of decimal places/significant digits
Quirks of real numbers

- Some double values print poorly (too many digits).
  
  ```
  double result = 1.0 / 3.0;
  println(result); // 0.3333333333333333
  ```

- The computer represents double in an imprecise way
  
  - Because only a finite amount of space in the “box”
  ```
  println(0.1 + 0.2);
  ```

  - Instead of 0.3, the output is 0.30000000000000004
Type casting

- **type cast**: A conversion from one type to another.
  - To promote an int into a double to get exact division from `/`
  - To truncate a double from a real number to an integer

- Syntax:

  \[(\text{type})\ \text{expression}\]

Examples:
```
double result = (double) 19 / 5; // 3.8
int result2 = (int) result; // 3
int x = (int) (10.0 * 10.0 * 10.0); // 1000
```
More about type casting

- Type casting has high precedence and only casts the term immediately next to it.

  ```
  double x = (double) 1 + 1 / 2;       // 1.0
  double y = 1 + (double) 1 / 2;       // 1.5
  ```

- You can use parentheses to force evaluation order.

  ```
  double average = (double) (a + b + c) / 3;
  ```

- A conversion to double can be achieved in other ways.
  - Common trick: Multiplying by 1.0

    ```
    double average = 1.0 * (a + b + c) / 3;
    ```
Example: Hamilton Lyrics Generator

Good style
Relational and Logical Operators

The operators we learned previously were primarily *arithmetical* operators.
Relational operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>equals</td>
<td>1 + 1 == 2</td>
<td>true</td>
</tr>
<tr>
<td>!=</td>
<td>does not equal</td>
<td>3.2 != 2.5</td>
<td>true</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>10 &lt; 5</td>
<td>false</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>10 &gt; 5</td>
<td>true</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
<td>126 &lt;= 100</td>
<td>false</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
<td>5.0 &gt;= 5.0</td>
<td>true</td>
</tr>
</tbody>
</table>

- if statements and loops use logical tests.

  ```
  for (int i = 0; i < 10; i++) { ... 
  if (age >= 40) { ... 
  ```

- These are boolean expressions. Boolean is a logical data type.
New data type: boolean

- Only two possible values: true and false
  - Actually write “true” and “false” (without the quotes) in your code:

```java
boolean isRainy = false;
if (isRainy) {
    println("Bring umbrella");
}
```

- Named after George Boole, a pioneer in the area of formal logic
- I’m assuming not actually an angry person, just had resting angry face
Misuse of if

What's wrong with the following code?

```java
int percent = readInt("What percentage did you earn? ");
if (percent >= 90) {
    println("You got an A!");
}
if (percent >= 80) {
    println("You got a B!");
}
if (percent >= 70) {
    println("You got a C!");
}
if (percent >= 60) {
    println("You got a D!");
}
if (percent < 60) {
    println("You got an NP!");
}
...
Solution: if/else-if

Ends with else: exactly 1 path taken. Ends with if: 0-1 paths taken.

```java
if (test) {
    statements;
} else if (test) {
    statements;
} else if (test) {
    statements;
}
```

Example:

```java
if (place == 1) {
    println("Gold medal!!");
} else if (place == 2) {
    println("Silver medal!");
} else if (place == 3) {
    println("Bronze medal.");
}
```
Unnecessary if

- The following code is unnecessarily verbose and redundant:

```java
if (x < 0) {
    println("x is negative");
} else if (x >= 0) {
    println("x is non-negative");
}
```

- The second test is unnecessary and can be removed:

```java
if (x < 0) {
    println("x is negative");
} else {
    println("x is non-negative");
}
```
Logical operators

Tests can be combined using logical operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>and</td>
<td>(2 == 3) &amp;&amp; (-1 &lt; 5)</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>or</td>
</tr>
<tr>
<td>!</td>
<td>not</td>
<td>!(2 == 3)</td>
<td>true</td>
</tr>
</tbody>
</table>

"Truth tables" for each, used with logical tests $p$ and $q$:

| $p$  | $q$  | $p$ && $q$ | $p$ || $q$ |
|------|------|-----------|-----------|
| true | true | true      | true      |
| true | false| false     | true      |
| false| true | false     | true      |
| false| false| false     | false     |

<table>
<thead>
<tr>
<th>$p$</th>
<th>!p</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>
“Or” / || in code

- “Or” in code means “one or the other or both”
- In English, sometimes we use “or” to include the both case, and sometimes we don’t mean it to include the both case:
  - Inclusive “or” includes the both case: “You will be in great shape if you go running or biking daily”—of course you will also be in great shape if you do both!
  - Exclusive “or” excludes the both case: “Your lunch special comes with soup or salad.”—usually restaurants mean you have to choose only one or other.

| p      | q      | p && q | p || q |
|--------|--------|--------|--------|
| true   | true   | true   | true   |
| true   | false  | false  | true   |
| false  | true   | false  | true   |
| false  | false  | false  | false  |