Variables and Types
Learning Objectives

- To know what a variable is
- To be able to declare variables
- To be familiar with primitive types
- To be able to perform operations on primitive type variables
- To be able to solve problems using primitive type variables
Overview

● One role of a computer program is to model and manipulate real or imaginary world entities

● In this module, we will learn how to represent the properties (or attributes) of the entities that our program will manipulate

● Example:

  ○ Entity: student

  ○ Properties: name, age, height, etc.
Definitions

- A **variable** helps us capture and store details about the problem or entity we are solving or modeling
- A **variable** is a **name** that is associated with a **value** (**data**)  
- The **value** is stored in a **memory** location in the computer
- The **value** can **change**
- A **variable** (**value**) must have a **type**
- A **type** defines the possible values values and the operations that can be performed on those values
Modeling with variables

- We are building a program to keep track of the **CIS 110 students**; we need to record information about them.
- What information should we store?

<table>
<thead>
<tr>
<th>Information / variable</th>
<th>Examples</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Malcom, Maya, Toni,…</td>
<td>Text</td>
</tr>
<tr>
<td>Age</td>
<td>13, 15, …</td>
<td>Number</td>
</tr>
<tr>
<td>Is a CIS major?</td>
<td>True, False</td>
<td>Text</td>
</tr>
<tr>
<td>Height</td>
<td>5.7, 6.0, 4.2, …</td>
<td>Number</td>
</tr>
</tbody>
</table>

- Can you think of any additional information that we could store?
Data types

- Two types of data types
  - Primitive types
  - Object types
Primitive types

- **int**: stores whole numbers (positive or negative) like 3, –5, 19000
- **double**: stores decimal numbers (positive or negative) like 3.5, –5.1, 19000.1
- **boolean**: stores Boolean values, either **true** or **false**

There are others we will introduce later
Object types

- They hold a reference to an object
- **String** is an object type and is associated with **text** (sequence of characters) values

More on these in a future lecture
Modeling with variables and Java types

- We are building a program to keep track of the **CIS 110 students**; we need to record information about them
- We update our table to use Java types

<table>
<thead>
<tr>
<th>Information / variable</th>
<th>Examples</th>
<th>Java Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Malcom, Maya, Toni,…</td>
<td>String</td>
</tr>
<tr>
<td>Age</td>
<td>17, 15, …</td>
<td>int</td>
</tr>
<tr>
<td>Is a CIS major?</td>
<td>True, False</td>
<td>boolean</td>
</tr>
<tr>
<td>Height</td>
<td>5.7, 6.0, 4.2, …</td>
<td>double</td>
</tr>
</tbody>
</table>

- What is the Java type of the information you added?
Variable declaration

- Creates a variable
- Associates a variable to a type
- The type determines how many bits (number of 0s and 1s) the computer will use to store the value associated with the variable

Examples

```java
// declaring the variable name
String name;

// declaring the variable age
int age;
```
Variable initialization

- Assigns a value to a variable: using the = sign
- The value and the type of the variable must be compatible
- Examples

```java
// declaring and initializing the variable name
String name = "Malcom";

// declaring the variable age
int age;
age = 14;

boolean is_taking_CIS110 = true;
```

Always surround a String value with "
One line: declaration + initialization
Two steps: declaration then initialization
Printing a variable

- Put the variable name without the quotes in the print command

```java
String name = "Maya";
System.out.print(name);
Prints Maya
```

- Use the + operator to append the value of a variable to a text in the print command

```java
System.out.print("Name of the student: " + name);
Prints Name of the student: Maya
```
Operations on variables

- Assignment statement initializes or changes the value of a variable previously declared
- Operators can be applied to values to perform computation
  - Such as Mathematical operators applied to `int` and `double` values
# Operations on boolean

<table>
<thead>
<tr>
<th>Type of operand 1</th>
<th>Operator</th>
<th>Type of operand 2</th>
<th>Type of result</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>&amp;&amp;</td>
<td>boolean</td>
<td>boolean</td>
<td>true &amp;&amp; false == false</td>
</tr>
<tr>
<td>boolean</td>
<td></td>
<td></td>
<td></td>
<td>boolean</td>
</tr>
<tr>
<td>boolean</td>
<td>!</td>
<td>N/A</td>
<td>boolean</td>
<td>!true == false</td>
</tr>
</tbody>
</table>
## Operations on `int`

<table>
<thead>
<tr>
<th>Type of operand 1</th>
<th>Operator</th>
<th>Type of operand 2</th>
<th>Type of result</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int</code></td>
<td>+</td>
<td><code>int</code></td>
<td><code>int</code></td>
<td>3 + 5 == 8</td>
</tr>
<tr>
<td><code>int</code></td>
<td>-</td>
<td><code>int</code></td>
<td><code>int</code></td>
<td>4 - 6 == -2</td>
</tr>
<tr>
<td><code>int</code></td>
<td>*</td>
<td><code>int</code></td>
<td><code>int</code></td>
<td>2 * 3 == 6</td>
</tr>
<tr>
<td><code>int</code></td>
<td>/</td>
<td><code>int</code></td>
<td><code>int</code></td>
<td>3 / 2 == 1</td>
</tr>
</tbody>
</table>
Operations on `double`

<table>
<thead>
<tr>
<th>Type of operand 1</th>
<th>Operator</th>
<th>Type of operand 2</th>
<th>Type of result</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>+</td>
<td>double</td>
<td>double</td>
<td>3.5 + 5.5 == 9.0</td>
</tr>
<tr>
<td>double</td>
<td>-</td>
<td>double</td>
<td>double</td>
<td>4.0 - 6.0 == -2.0</td>
</tr>
<tr>
<td>double</td>
<td>*</td>
<td>double</td>
<td>double</td>
<td>2.5 * 1.0 == 2.5</td>
</tr>
<tr>
<td>double</td>
<td>/</td>
<td>double</td>
<td>double</td>
<td>3.0 / 2.0 == 1.5</td>
</tr>
</tbody>
</table>
Operations on *double* and *int*

- When one of the operand is of type *double*, the results is of type *double*

<table>
<thead>
<tr>
<th>Type of operand 1</th>
<th>Operator</th>
<th>Type of operand 2</th>
<th>Type of result</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>+</td>
<td>int</td>
<td>double</td>
<td>3.5 + 5 == 8.5</td>
</tr>
<tr>
<td>int</td>
<td>-</td>
<td>double</td>
<td>double</td>
<td>4 - 6.0 == -2.0</td>
</tr>
<tr>
<td>double</td>
<td>*</td>
<td>int</td>
<td>double</td>
<td>2.5 * 1 == 2.5</td>
</tr>
<tr>
<td>double</td>
<td>/</td>
<td>int</td>
<td>double</td>
<td>3.0 / 2 == 1.5</td>
</tr>
</tbody>
</table>
Operator Type Errors

- Sometimes mixing variable types and values will result in compiler errors:

```java
// Wrong value for the specified variable type
int pi = 3.14159;
double x = true;

// Using operators with incompatible/mismatching types
int y = 1 + false;
boolean z = 110 && 120;
```
Comparison: Equality

● The `==` operator is used to check for equality.
● The result is a `boolean` value (`true` or `false`)

```java
int x = 4;
int y = 5;
x == y;  // evaluates to `false`
```

● The result of the comparison can be stored in a `boolean` value

```java
int x = 4;
int y = 5;
boolean v = (x == y);
System.out.println(v);  // prints `false`
```
Comparison: Inequality

- The `!=` operator is used to check for inequality (not equals).
- The result is a `boolean` value (`true` or `false`)

```java
int x = 4;
int y = 5;
x != y; // evaluates to true
```

- The result of the comparison can be stored in a `boolean` value

```java
int x = 4;
int y = 5;
boolean v = (x != y);
System.out.print(v); // prints true
```
Comparison: Others

- For types like `int` and `double`, we can also perform other comparisons.
- The result is a `boolean` value (`true` or `false`);

<table>
<thead>
<tr>
<th>Operator Name</th>
<th>Syntax</th>
<th>Example</th>
<th>Example Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less than</td>
<td><code>&lt;</code></td>
<td><code>5 &lt; 6</code></td>
<td><code>true</code></td>
</tr>
<tr>
<td>Less than or equal to</td>
<td><code>&lt;=</code></td>
<td><code>5 &lt;= 5</code></td>
<td><code>true</code></td>
</tr>
<tr>
<td>Greater than</td>
<td><code>&gt;</code></td>
<td><code>2 &gt; 3</code></td>
<td><code>false</code></td>
</tr>
<tr>
<td>Greater than or equal to</td>
<td><code>&gt;=</code></td>
<td><code>5 &gt;= 1</code></td>
<td><code>true</code></td>
</tr>
</tbody>
</table>
The modulo (%) operator

- The mod operator \((x \ % \ y)\) returns the remainder after you divide \(x\) (first number) by \(y\) (second number)

\[
5 \ % \ 2 \rightarrow 1
\]

\[
4 \ % \ 2 \rightarrow 0
\]
## Compound Assignment Operators

- Shortcuts that do a math operation and assignment in one step

<table>
<thead>
<tr>
<th>+ shortcuts</th>
<th>- shortcuts</th>
<th>* shortcut</th>
<th>/ shortcut</th>
<th>% shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = x + 1;</td>
<td>x = x - 1;</td>
<td>x = x * 2;</td>
<td>x = x / 2;</td>
<td>x = x % 2;</td>
</tr>
<tr>
<td>x += 1;</td>
<td>x -= 1;</td>
<td>x *= 2;</td>
<td>x /= 2;</td>
<td>x %= 2;</td>
</tr>
<tr>
<td>x++</td>
<td>x--</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Second row only works for operating on the same value being assigned into.
- Last row only works for adding/subtracting 1.
Operator Chaining & Priority

- You can chain multiple operators together in one line:
  ```java
  int sum = 110 + 120 + 160 + 121 + 240;
  ```

- Sometimes the order of operations is unclear. Example:
  ```java
  boolean x = 110 + 120 * 2 == 2;
  ```

- To avoid confusion, use parenthesis to specify the order of operations:
  ```java
  boolean y = (110 + (120 * 2)) == 2;
  ```

Does the same thing as the above without parenthesis. Parenthesis are still recommended for general use.
Casting

- Type casting converts a variable from one type to another
- `(int)` is used to cast a value to `int`
  
  \[(\text{int}) \ 3.5 \rightarrow 3\]

- `(double)` is used to cast a value to `double`
  
  \[(\text{double}) \ 3/2 \rightarrow 1.5\]

- Casting applies only to the closest operand
  
  \[(\text{double}) \ 3/2 \text{ is the same as } ((\text{double}) \ 3)/2 \text{ or } 3.0/2\]
Live Coding DEMO

LeapYear.java

Program that will determine if a year is a leap year.