Today’s Topics

• Questions / Comments?
• Variables & Types
• Arithmetic Operators
• Equality & Relational Operators
• if / else
Variables and Type

• A program often needs to be able to store and change data as it is running. For example, a program might need to store a user's name, or the user's age, or the air temperature, or whatever.

• Variables allow the programmer to give a name to data that can be stored and changed.
Variables and Type

• A variable is a location in memory (RAM) with the following properties.

• name --- given by the programmer --- try to be descriptive of the data the variable will hold

• type --- decided by the programmer from all available types, depending on what kind of values the programmer wants to store

• size --- amount of memory used, this is determined by the type

• value --- self-explanatory
  – value can be stored / changed / read by a program.
Primitive Types for Variables

• **byte, short, int, long** --- these are integer types --- they hold *whole* numbers like –31, 4256, 56, 2632, 755, -901 (they hold different ranges)

• **char** --- this is a type that holds *one* character like a letter, digit, *, &, or an escape sequence …

• **boolean** --- holds one of *true* or *false*

• **float, double** --- numeric types that hold numbers with decimal values like -5.5, 98.6, 1002.99995 etc. (they hold different ranges and to different precisions)

• We can look up the ranges/precision of the numeric types.
Variables and Type

- To be able to specify to our program that we want to use a variable, **before we can use it we must declare the variable.** That is, we state the desired type as well as a descriptive name that we make up (must start with a letter and may contain letters, digits, and _).  

- Example: If we wanted a place to store a person's age (years only e.g. 18, 21, 55 etc.), we might create a variable like this:

```plaintext
int age; // declares the variable age of type int
```

- age is a name we made up, what does int do?
• **int age;**  // declares the variable age of type int

• age is a name we made up, what does int do?

• We chose int because we knew we wanted to only store whole numbers and the int type stores whole numbers. We could have chosen byte, short or long which also hold whole numbers but we typically use int for whole numbers unless we have a good reason to use one of those other three.
Variables and Type

```java
int area = 50;  // This line of code not only 
// declares a variable but also initializes its value.
```

- **name** of variable is **area**
- **type** of variable is **int**
- **size** of variable is 32 bits (because an int has a size of 32 bits.)
- **value** of variable is currently **50**

note the same thing can be done with 2 lines of code:

```java
int area;
area = 50;  // this is an assignment statement (it assigns 
// the value 50 to the variable area
```
Program Flow

- Java programs start executing in the main method.
- They execute sequentially, in order (unless something changes the program flow --- more on that later today and throughout the semester) until the end of the main method when the program ends.
public class VariableExample
{
    public static void main(String args[])
    {
        int result; // declares variable with name result and type int
        // allocates enough memory for an int to be stored
        int some_number; // declares variable with name some_number and type
        // int and allocates enough memory for an int to be stored
        result = 35; // an assignment statement that stores the value 35 in the
        // variable result.
        some_number = 24; // an assignment statement that stores the value 24
        // in the variable some_number.
        result = some_number + 5;
        // uses the + operator to add the value in some_number to 5
        // and stores (assigns) the sum to the variable result, thereby
        // changing result’s value.
    } // end main method body
} // end class VariableExample body
<table>
<thead>
<tr>
<th>Memory after line</th>
<th>result</th>
<th>some_number</th>
</tr>
</thead>
<tbody>
<tr>
<td>line 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>line 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>line 3</td>
<td>35</td>
<td>some_number</td>
</tr>
<tr>
<td>line 4</td>
<td>35</td>
<td>24</td>
</tr>
<tr>
<td>line 5</td>
<td>29</td>
<td>24</td>
</tr>
</tbody>
</table>
Types for Variables

- Variables can be of the primitive types (int, char, double, etc.) as well as predefined classes provided by Java or a class that a programmer creates.
- For example a variable can be of type `String` which is a class provided by Java.
- We’ll see more of variables and types shortly.
Operators

- + when used with numeric types (e.g. int, float etc.) acts as an add, however, + when used with Strings is a concatenation operator.
  - example: `text_string = “Hello ” + first_name;`
- = is the assignment operator.
- * is multiplication, / is division, % is modulus and - is subtraction.
- () left and right parentheses are considered operators as well.
The modulus operator (Integer division & remainder)

- `%` is the modulus operator
  
  ```
  int val1=16, val2 = 5, remain;
  remain = val1 % val2;
  ```

- The modulus operator returns the remainder after `val1` is divided by `val2`.

- Suppose `val1 = 16` and `val2 = 5`, `remain` would have the value 1 because 16/5 is 3 with a remainder of 1.
Arithmetic Operator precedence levels

• ( ) parentheses are evaluated first
• if parens are nested then the innermost pair is evaluated first.
• *, /, % multiplication, division and modulus are next --- if several of these, they are evaluated left to right.
• +, - addition and subtraction are last --- if several of these, they are evaluated left to right
• It’s useful to memorize this precedence list
some_result = ( 5 + 4 - 1) * 6 + ( 2 - 10 );

• the order in which operations are performed could change the computation's answer.

• *(wrong way)* suppose we did 2-10 = -8 and then 6-8 = -2 then 5+4-1 = 8, 8*(-2) = -16

• *(correct way)* 5+4-1=8, 2-10=-8, 8*6 = 48, 48-8 = 40
Now that we know that operators which are at the same precedence level get evaluated from left to right, we can figure out how and why the following line of code works as it does:

System.out.println("10 plus 5 is " + 10 + 5);

will print
10 plus 5 is 105

the first + acts like concatenation and joins the string and the 10 which gets converted to a String automatically. Then the second + acts like concatenation of the String to 5 which is converted to a String automatically.
Equality and Relational operators

`==` is equal to operator
`!=` is not equal to operator
`<` is less than operator
`>` is greater than operator
`<=` is less than or equal to operator
`=<` is greater than or equal to operator
if / else structure

- So far all the code we’ve seen was executed sequentially --- every line of code in order from the first to the last within the main method.

```java
if (condition)
{
    statement_to_do_if_condition_is_true;
    another_statement_to_do_if_condition_is_true;
}
else
{
    statement_to_do_if_condition_is_false;
    another_statement_to_do_if_condition_is_false;
    yet_another_statement_to_do_if_condition_is_false;
}
```

Michael Eckmann  -  Skidmore
College  -  CS 106 -  Fall 2015
if / else structure

• condition is something that will evaluate to true or false (usually using relational or equality operators.)
• if you only want to do one statement (in the if or else portion) curly braces are not required
• if you want to do more than one statement the curly braces are necessary
• also, the else portion of the if / else structure is optional
valid if / else example

if (radius > 0)
{
    area = 3.14159 * radius * radius;
    circumference = 2 * 3.14159 * radius;
}
else
{
    System.out.println("Radius is invalid");
}
a few more valid if / else examples

if (numerical_grade >= 90)
    System.out.println("your grade is an A!");

/*---------------------------------------------------------------*/

if (account_balance < 200)
    System.out.println("Acct below $200 minimum");
else
    System.out.println("Do not charge a service fee");
Exercise

• Write an application given two integers, displays the larger number followed by the words “is larger”. If the numbers are equal, print the message “These numbers are equal.”
compound statements

• A single statement is a line of code ending in a semicolon.
• A compound statement is a set of statements within a pair of curly braces.
• A compound statement may be used anywhere in a program that a single statement can be used.

• example of a single statement

  \[
  \text{amount} = 45;
  \]

• example of a compound statement

  \[
  \{
  \text{area} = 3.14159 \times \text{radius} \times \text{radius};
  \text{circumference} = 2 \times 3.14159 \times \text{radius};
  \}
  \]
if / else structure revisited

• The if part of the if / else structure can contain a single statement or a compound statement. Same for the else portion of the if / else.

```java
if (degrees_fahr <= 45)
    System.out.println("It is pretty cold.");
else
    System.out.println("It is not too cold.");
    System.out.println("You don't need a jacket.");
```

• What would print out if `degrees_fahr = 30` before the above if / else?
if / else and compound statements

• The code on the preceding slide would print “You don’t need a jacket.” regardless of the value of degrees_fahr.

• The following code, using a compound statement, only prints that line if degrees_fahr > 45.

```java
if (degrees_fahr <= 45)
    System.out.println("It is pretty cold.");
else
{
    System.out.println("It is not too cold.");
    System.out.println("You don't need a jacket.");
}
```
another subtlety of the if / else

• Java always associates an else with the preceding if unless curly braces are used to force it not to do this.
• example:
  if (i == 0)
    if (j == 0)
      System.out.println("both i and j are 0");
  else
    System.out.println("i is not 0");

• The above code will not execute like the programmer thought, regardless of the indentation.
• The else will be paired with the if (j==0)  NOT  the if (i == 0).
another subtlety of the if / else

• An else can be forced by the programmer to pair up with a different (not the immediately preceding) if by using curly braces.

```java
if (i == 0)
{
    if (j == 0)
        System.out.println("both i and j are 0");
}
else
    System.out.println("i is not 0");
```

• The above code will now execute like the programmer intended.
import statement

• Normally we need to tell the compiler where to look for the class or method we are using in the Java API. This is done by the use of the import statement at the beginning of your source code file.

• To use a class called JOptionPane to create dialog boxes, we need to add the following line to the beginning of our code:
  – import javax.swing.JOptionPane;
import statement

• The import statement is of the form
  – `import package.class;`

• `javax.swing` is a package and `JOptionPane` is the class we want to use in it.

• `import javax.swing.*` would allow us to use any class in the `javax.swing` package.
import statement

• We didn’t need to import anything to use System.out in the HelloWorld program because System is a class that is part of the java.lang package and java.lang gets implicitly imported in all Java programs.

• Java API Specification is on the web and is where you’ll find all the classes available to you in the Java API. There is a link to it on our course web page.
javax.swing.JOptionPane

• Contains methods to show dialog boxes.
• JOptionPane.showMessageDialog displays messages in a dialog box.
• JOptionPane.showInputDialog allows for user input in a dialog box.
• Nicer to use than reading from / printing to the console screen.

Michael Eckmann - Skidmore College - CS 106 - Fall 2015
JoptionPane class for I/O

• showMessageDialog takes two parameters, the first of which should always be null for now.
• The second is a String that should be outputted.

• showInputDialog takes one parameter which is a String that displays --- it should tell the user what to enter. The user's input is returned as a String.
JOptionPane class for I/O

• Example usage:

```java
import javax.swing.JOptionPane; // need this line above your class

String input_string; // variable to store the user's input

JOptionPane.showMessageDialog(null, "Hey");
input_string = JOptionPane.showInputDialog("Enter something.");
```
javax.swing.JOptionPane

• Let’s write Hello World using a message dialog box instead of System.out.println.
java.util.Scanner

• Scanner input_scan = new Scanner(System.in);
• methods that you can call on your scanner object include:
  – nextInt() --- reads an int from keyboard
  – nextDouble() --- reads a double from keyboard
  – nextLine() --- reads a line (as a String) from keyboard
  – next() --- reads a “word” from the keyboard --- which is a string of nonwhitespace chars delimited by whitespace. whitespace is \n, blank space character, \t, \r
java.util.Scanner

- `nextLine()` --- reads a line (as a String) from keyboard
  - this method “consumes” the \n but does not make it part of the String that is returned.
    
    ```java
    String s1, s2;
    Scanner my_scan = new Scanner(System.in);
    s1 = my_scan.nextLine();
    s2 = my_scan.nextLine();
    ```
  - if input is:
    CS106
    Introduction to Computer Science I
  - there is a \n after CS106 in the input (e.g. user hit enter key), but s1 will be “CS106” and s2 will be “Introduction to Computer Science I” neither will have \n as a character in its String.
java.util.Scanner

• There's an unfortunate problem with Scanner when using one to get both numeric and String input.

• If you need to get both numeric and String input from user input from the keyboard I recommend creating two Scanners, one that only gets numeric input (nextInt, nextDouble) and one that only gets the String input (nextLine, next).