Programming 2

THE 6 PROGRAMMING INSTRUCTIONS IN JAVA

Review of the 6 programming instructions

- **Key Concept:** all programs can be broken down to a combination of one of the six instructions
- **Assignment Statements** – can create variables to represent information
- **Input** – can receive information from the user or listen for events
- **Output** – can display information on a screen, write information to a file, etc.
- **Math** – can process information (in particular perform math on numbers)
- **Conditional Execution** – can perform operations under particular circumstances, depending on the value of variables
- **Repetition** – perform a set of instructions multiple times (either a set number of times or until a condition is met)

Assignment Statements

- In Java, before you set a variable to a particular value, you need to **declare the variable**
  - You need to choose the appropriate data type the variable will be
  - You need to notify Java what data type it will be
- Formula: *DataType + VariableName;*
  - int age;
  - String name;
  - double weight = 175.50;
  - Note: you can declare multiple variables in one line
    - int age, weight, zipcode;
- In Java, Data types fall under two main categories:
  - **primitive data types**
  - **abstract data types**

Primitive Data Types

- **Primitive data types** are the basic building blocks in Java programming.
  - **byte** – integer with a range from -128 to 127 (8-bit)
  - **short** – integer with a range between -32,768 to 32,767 (16 bit)
  - **int** – integer with a range between -2,147,483,648 and 2,147,483,647 (32 bit)
  - **long** – an integer with a range between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807 (64-bit)
  - **float** – single precision floating point number. Only use this when precise doubles are not crucial.
    - These numbers are stored in 32-bit memory locations
    - Do not use with precise values, such as currency or scientific calculations
  - **double** – double precision floating point number. They offer twice the precision as floats, and for most of our work, they are sufficient
    - These numbers are stored in 64-bit memory locations
    - Do not use with precise values, such as currency or scientific calculations
  - **boolean** – these have only 1 of two values: true or false
  - **char** - The char data type is a single 16-bit Unicode character.
    - It has a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (the question mark ‘?’)
    - That gives you a possible 65,535 different characters

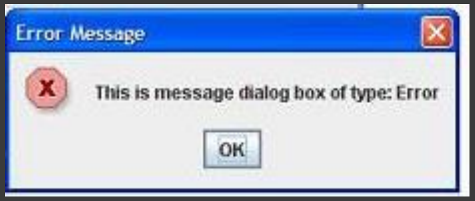
Abstract Data Types

- There is a huge range of “other” data types that fall under the category of **abstract data types**
  - abstract data types are more complex data types
  - they are often used to represent real-life objects
- Some abstract data types are built into Java:
  - strings
  - arrays, etc.
- Some need to be imported:
  - Date objects
  - Labels
  - Buttons
  - TextFields
- Some are objects that you can imagine
  - **Card** – if you wanted to code a card game
  - **Ship** – if you wanted to code a video game
  - **Student** – if you wanted a program for a school application

Output in Java

- With a Console Application
  - System.out.print(“Hello.”);
    - This will position the cursor at the end of the line (great for input)
  - System.out.println(“Hello.”);
    - This will output “Hello.” and then position the cursor on the next line.
  - Note: with both output commands, you can place strings, variables, & expressions
    - e.g. System.out.println(“The answer is” + **(10 * 3) + (22 / 3)**);
- Note: there are many ways to produce output, but this is just a start

Output: Swing Pop-up Message



This is message dialog box of type: Error

- Make sure you import Swing
  - `import javax.swing.*;`
- Show a message dialogue box
  - `JOptionPane.showMessageDialog(null, “We have a problem Houston", "alert", JOptionPane.ERROR_MESSAGE);`
  - About the arguments in the parentheses:
    1. **null** – means don’t place it into another window or frame
    2. **“We have a problem Houston”** – is the message you want to display
    3. **“alert”** – is the title for the dialog box
    4. **JOptionPane.ERROR_MESSAGE** – displays a generic error message icon. Other icons are
       - INFORMATION_MESSAGE
       - WARNING_MESSAGE
       - PLAIN_MESSAGE

Input for a Console Application

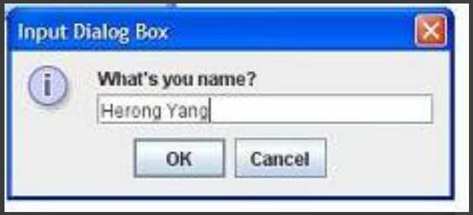
- First, import the Scanner class
  - `import java.util.Scanner;`
- Next, create a Scanner object where you declare your variables
  - `Scanner reader = new Scanner(System.in);`
- Create a variable to capture the input
  - `double fahrenheit;`
- using System.out.print(), produce a prompt
  - `System.out.print(“Enter degrees: “);`
- Get your input
  - `fahrenheit = reader.nextDouble();`
- What if we want to get another data type other than a double?
  - `int pounds = reader.nextInt();`
  - `String name = reader.nextLine();`

Input using Pop-ups (Swing)

To get input using a pop-up window, use the showInputDialog() method

- Import swing
  - `import javax.swing;`
- Assign a variable to a showInputDialog method

  -
    ```
    String answer =
    JOptionPane.showInputDialog(null, “What’s your
    name?”, “Input Dialog Box”,
    JOptionPane.INFORMATION_MESSAGE);
    ```



What's you name?

Herong Yang

Strings in Java

- **Strings** – are another form of abstract data type
  - they are considered objects
  - you can concatenate strings (just like with Python)
    - `String output = firstName + ", " + lastName;`
  - you can call String methods
    - `output.length()` – returns the number of characters in the string
  - You can print index position of strings
    - `output.charAt(i);`
  - You can run a string through a for loop
    - We'll cover that soon
- We'll deal with more abstract data types later

Math

- Math Operators: these are pretty much the same as in Python with a few exceptions:

Operation	Operator Symbol	Example	Result
Multiplication	*	15 * 3	45
Division	/	20.0 / 3.0	6.6666667 for a float 6.666666666666667 for a double
Integer Division	/	20/3	6 the result is truncated (the remainder is dropped)
Modular Division	%	20%3	2 (the integer remainder only)
Addition	+	15 + 5	20
Subtraction	-	15 - 5	10

More On Math

- **Type Casting**:
  - no, this is not when Gary Busey gets the part of the crazy maniac or Michael Cera plays an awkward teenager
  - It’s when you change the data type of a number,
    - (int) 20.6; would produce 20
    - (double) 5; would produce 5.000000
- **The Math Class**:

Method	Description	Example Code	Result
abs()	Absolute value	Math.abs(-75)	75
max()	Higher of two numbers	Math.max(99, 41)	99
min()	Lower of the two numbers	Math.min(99, 41)	41
pow()	Exponentiation	Math.pow(2, 5)	32
random()	Random number generator	Math.random()	Produces a random number between 0.0 and 1.0
round()	Round to an integer	Math.round(27.59)	28
sqrt()	Square root	Math.sqrt(144)	12

Conditional Execution

- There are **4 types** of conditional execution in Java
  - the if structure
  - the if else structure
  - the if else if else structure
  - the case/switch structure
    - we’ll cover this when we cover the while loop
- **Conditions**: remember, with conditional execution, a condition is a statement that either evaluates to true or false:
  - ex. (age > 20)
  - either the age is greater than twenty (true)
  - or it is not greater than twenty (false)

A Note on Operators

- == is equal to
- != is not equal to
- < less than
- > greater than
- <= less than or equal to
- >= greater than or equal to
- && AND (logical operator)
- || OR (logical operator)
- ! NOT (logical operator)
- Practice: does the expression evaluate to true or false?
  - 4 <= 6
  - 3 < 5
  - 1 == 6
  - 9 > 7

- 7 <= 6
- (7 > 3) || (1 < 0)
- 4 != 4
- (7 > 3) && (1 < 0)
- 8 >= 10
- (3 > 7) || (1 < 0)
- 5 < 3
- 7 != 4
- 7 > 0
- 2 == 2
- 8 >= 8
- !(a == a)
- (7 > 3) && (1 > 0)
- !(5 == 4)

The if structure:

- in Java you can do the if structure two ways…

  -
    ```
    if (age > 40) System.out.println(“You’re getting old!”);
    ```

  -
    ```
    if (age > 55) {
        System.out.println(“You’re really getting old!”);
        oldMan = true;
    }
    ```
  - Note: if the code that executes under the condition is extra long or requires more than 1 statement, you must surround your statements with curly braces

if-else and if-else if-else

```
void applyBrakes(){
if (isMoving) {
   currentSpeed--;
}
else {
   System.err.println("The
   bicycle has already
   stopped!");
   }
}
```

if-else

```
if (testscore >= 90) {
   grade = 'A';
}
else if (testscore >= 80)
   { grade = 'B';
}
else if (testscore >= 70)
   { grade = 'C';
}
else if (testscore >= 60)
   { grade = 'D';
}
else {
   grade = 'F';
}
```

if-else if-else

The Case-Switch Structure

```
String comment// The generated insult.
int which = (int)(Math.random() * 3);

switch (which) {
    case 0: comment = "You look so much
    better than usual."; break;
    case 1: comment = "Your work is up
    to its usual standards."; break;
    case 2: comment = "You're quite
    competent for so little
    experience."; break;
    default: comment = "Oops --
    something is wrong with this
    code.";
}
```

Sample Case-Switch Code

- Switch structures only work with integers
  - which is a random number (0, 1, or 2)
- Use the switch keyword
  - In the parentheses, include an integer (variable or literal)
- Each case is like an if statement
  - End each case with a break, so it doesn't execute more than 1 line of code
  - You can wrap multiple lines of code inside each case (use braces to surround the block)
- default is what gets run if none of the none of the cases apply

Notes