Java Overview
Homework 6

• Will post tonight
• Due next Tuesday
• Practice with some basic Java concepts – arithmetic, types, etc.
Today: Java overview

- Read in data (for now, numbers)
  - We will use a **Scanner** for this

- Save numbers in variables
  - We will use **declarations** to create variables

- Do arithmetic on numbers to get new numbers
  - We will use **assignment** statements

- Test whether or not to do something
  - We will use **if** statements

- Do something repeatedly
  - We will use **while*/**for** loops

- Print output
  - We will use **System.out.print** and **System.out.println**

- Use **methods**
  - A “method” is a function that belongs to a class
Variables

• Every variable has a **name**
  – Variables start with a lowercase letter
  – Multiword variables are written using “camelCase”
  – “camelCase” is the standard Java naming style (more later)
• Every variable has a **type** of value that it can hold
  – The type of a variable **cannot be changed**
  – Java is statically typed
  – In Python, you can change variable type on the fly
Some Java data types

• In Java, the most important **primitive** (simple) types are:
  – **int** variables hold integer values
  – **double** variables hold floating-point numbers (numbers containing a decimal point)
  – **boolean** variables hold a **true** or **false** value

• Other primitive types are
  – **char** variables hold single characters
  – **float** variables hold less accurate floating-point numbers
  – **byte, short** and **long** hold integers with fewer or more digits

• Another important type is the **String**
  – A **String** is an **Object**, not a primitive type
  – A **String** is composed of zero or more **chars**
Declarations, statements, comments

- A **declaration** gives **type** information to the computer
  - You must declare:
  - The type of value (*int*, *String*, etc.) each variable can hold
  - The type of every parameter to a method
  - The type returned by every method

- A **statement** tells the computer to do something
  - Statements should really be called “commands”
  - Statements may only occur within methods
  - Variable declarations and statements can be combined in one

- **Comments** are ignored by the compiler
  - As in Python, there are different comments for people who use your methods, and those who read your code
Declaring variables

• In Python, a variable may hold a value of any type
• Every variable that you use in a program must be declared (in a declaration)
  – The declaration specifies the type of the variable
  – The declaration may give the variable an initial value
• Examples:
  – int age;
  – int count = 0;
  – double distance = 37.95;
  – boolean isReadOnly = true;
  – String greeting = "Welcome to CIT 590";
  – String outputLine;
Assignment statements

• Values can be assigned to variables by assignment statements
  – The syntax is: `variable = expression;`
  – The expression must be of the same type as the variable or a type that can be converted without loss of precision
  – The expression may be a simple value or it may involve computation

• Examples:
  – `name = "Rado";`
  – `count = count + 1;`
  – `area = (4.0 / 3.0) * 3.1416 * radius * radius;`
  – `isReadOnly = false;`

• When a variable is assigned a value, the old value is discarded and totally forgotten
Multiple values

- An array lets you associate one name with a fixed (but possibly large) number of values.
- Arrays are like Python’s lists, but much less flexible:
  - You can’t easily extend their size (Python takes care of this for you under the hood).
  - All values must have the same type.
- The values are distinguished by a numerical index between 0 and array size minus 1.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>myArray</td>
<td>12</td>
<td>43</td>
<td>6</td>
<td>83</td>
<td>14</td>
<td>-57</td>
<td>109</td>
<td>12</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>
Declaration and definition

- To **declare** an array is to tell its type, but *not* its size
  - Example: `int[] scores;`
- To **define** an array is to give its size
  - Example: `scores = new int[40];`
- Declaration and definition can be combined
  - Example: `int[] scores = new int[40];`
- The initial content of an array is zero (for numbers), **false** (for booleans), or **null** (for objects)
Using array elements

- $x = \text{myArray}[1]$;  // sets $x$ to 43
- $\text{myArray}[4] = 99$;  // replaces 14 with 99
- $m = 5$
- $y = \text{myArray}[m]$;  // sets $y$ to -57
- $z = \text{myArray}[\text{myArray}[9]]$;  // sets $z$ to 109

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>myArray</td>
<td>12</td>
<td>43</td>
<td>6</td>
<td>83</td>
<td>14</td>
<td>-57</td>
<td>109</td>
<td>12</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>
Arrays of arrays

- The elements of an array can themselves be arrays
- Once again, there is a special syntax
- Declaration: `int[ ][ ] table;
- Definition: `table = new int[10][15];`
- Combined: `int[ ][ ] table = new int[10][15];`
- The first index (10) is usually called the row index; the second index (15) is the column index
- An array like this is called a **two-dimensional array**
Printing

• There are two methods you can use for printing:
  – `System.out.println(something);`
    • This prints something and ends the line
  – `System.out.print(something);`
    • This prints something and *doesn’t* end the line (so the next thing you print will go on the same line)

• These methods will print any **one** thing, but only one at a time

• You can concatenate Strings with the `+` operator

• Anything concatenated with a String is automatically converted to a String
  – Example:
    `System.out.println("There are " + appleCount + " apples and " + orangeCount + " oranges.");`
  – Note the difference with Python (where you have to call `str`)
Reading in numbers

• First, import the Scanner class:
  ```java
  import java.util.Scanner;
  ```

• Create a scanner and assign it to a variable:
  ```java
  Scanner scanner = new Scanner(System.in);
  ```
  – The name of our scanner is `scanner`
  – `new Scanner(...)` says to make a new one
  – `System.in` says the scanner is to take input from the keyboard

• Finally, read in the number:
  ```java
  int myNumber = scanner.nextInt();
  ```

• Eclipse example: GuessTheNumberLoop.java
Arithmetic expressions

- Arithmetic expressions may contain:
  - + to indicate addition
  - - to indicate subtraction
  - * to indicate multiplication
  - / to indicate division
  - % to indicate remainder of a division (integers only)
  - Sorry, no exponentiation
  - Exponentiation is done with the method `Math.pow(base, exponent)`

- Use parentheses ( ) to indicate the order in which to do things

- An operation involving two `ints` results in an `int`
  - When dividing one `int` by another, the fractional part of the result is thrown away: `14 / 5` gives `2`

- Any operation involving a `double` results in a `double`: `14.0 / 5` gives approximately `2.8`
  - More on types later
Boolean expressions

• Arithmetic **comparisons** result in a **boolean** value

• There are six comparison operators:
  - `<` (less than), `<=` (less than or equals), `>` (greater than), `>=` (greater than or equals), `==` (equals), `!=` (not equals)

• There are three **boolean operators**:
  - `&&` “and”--true only if both operands are true
  - `||` “or”--true if either operand is true
  - `!` “not”--reverses the truth value of its one operand

• Example:
  - `(x > 0) && !(x > 99)`
  - “x is greater than zero and is not greater than 99”
String concatenation

- You can **concatenate** (join together) Strings with the `+` operator
  - Example: `fullName = firstName + "  " + lastName;`
- In fact, you can concatenate any value with a String and that value will automatically be turned into a String
  - Example:
    ```java
    System.out.println("There are " + count + " apples.");
    ```
- Be careful, because `+` also still means addition
  - `int x = 3;`
  - `int y = 5;`
  - `System.out.println(x + y + " != " + x + y);`
  - The above prints `8 != 35` as a string
  - “Addition” is done left to right
  - “Conversion” to String is also done left to right
  - Use parentheses if you want to change the order
if statements

• An if statement lets you choose whether or not to execute one statement, based on a boolean condition
  – Syntax:
    
    if (boolean_condition)
    statement;
  
• Example:
  
    if (x < 100)  // adds 1 to x, but only if x is less than 100
                 x = x + 1;

• Python programmers: The parentheses are required
  – Don’t put a semicolon after the parentheses
  – This is the semicolon of death
  – Your if statement will control only the semicolon, i.e., nothing
if-else statements

• An if statement may have an optional **else part**, to be executed if the boolean condition is false
  – Syntax:
    ```java
    if (boolean_condition) {
        statement;
    } else {
        statement;
    }
    ```

• No need for braces if only one statement in each
  – More on this on next slide

• Java’s equivalent to Python’s **elif** is **else if**

• Eclipse example: GuessTheNumberLoop.java
Compound statements

- Multiple statements can be grouped into a single statement by surrounding them with braces, `{ }`
- Example:
  ```java
  if (score > 100) {
      score = 100;
      System.out.println("score has been adjusted");
  }
  ```
- Unlike other statements, there is no semicolon after a compound statement
- Braces can also be used around a single statement, or no statements at all (to form an “empty” statement)
while loops

- A **while loop** will execute the enclosed statement as long as a boolean condition remains **true**
  
  - Syntax: `while (boolean_condition) {
    statement;
  }
  
  - Example:
    ```
    n = 1;
    while (n < 4) {
      System.out.println(n + " squared is " + (n * n));
      n = n + 1;
    }
    ```

- **Danger**: If the condition never becomes false, the loop never exits, and the program never stops
The increment/decrement operators

- `++` adds 1 to a variable
  - If before a variable (**preincrement**), it means to add one to the variable, then use the result
  - If put after a variable (**postincrement**), it means to use the current value of the variable, then add one to the variable
  - When used as a statement, preincrement and postincrement have identical results

- `--` works in the same way, except it subtracts 1

- Don’t do weird things with these
  - Try to use them in single statements

- Eclipse example: Increment.java
The for loop

• The **for** loop is slightly complicated, but *very* handy

• Syntax:

```java
for (initialize ; test ; increment) {
    // loop statements
}
```

  – Notice that there is no semicolon after the **increment**

• Execution:

  – The **initialize** part is done first and only once
  – The **test** is performed – as long as it is **true**, we enter the loop
  – The **statements** are executed
  – The **increment** is executed

• Eclipse example: PrintMany.java
Parts of the for loop

• Initialize: In this part you define the loop variable with an assignment statement, or with a declaration and initialization
  – Examples:
    – i = 0
    – int i = 0 //this will create a new local variable i inside the loop

• Test: A boolean condition
  – Just like in the other control statements we have used

• Increment: An assignment to the loop variable, or an application of ++ or -- to the loop variable
  – This may be the only good use of ++ and --!
When do you use each loop?

• Java loops are intuitively similar to Python loops
  – Somewhat harder to iterate through collections

• Use the **for** loop if you know ahead of time how many times you want to go through the loop
  – Example: Stepping through an array

• Use the **while** loop in almost all other cases
  – Example: Compute the next step in an approximation until you get close enough
The break/continue statements

- Exactly the same as in Python
- **Break** will immediately exit the innermost loop
- **Continue** will return you to the beginning of the innermost loop
The assert statement

- The purpose of the `assert` statement is to document something you believe to be true
- There are two forms of the `assert` statement:
  1. `assert booleanExpression;`
     - It does nothing if the boolean expression evaluates to `true`
     - If the boolean expression evaluates to `false`, this statement throws an `AssertionError`
  2. `assert booleanExpression: expression;`
     - This form acts just like the first form
     - In addition, if the boolean expression evaluates to `false`, the second expression is used as a detail message for the `AssertionError`
     - The second expression may be of any type except `void`
- `Assert` statements are disabled by default in Java
  - This is for efficiency reasons
Comments

• Python: Single-line comments start with `#`
• Java: Single-line comments start with `//`

• Java: Multi-line comment start with `/*` and end with `*/`

• Python: Documentation comments are enclosed in triple quotes, and are put right after the `def` line
• Java: Documentation comments start with `/**` and end with `*/`, and are put just *before* the definition of a variable, method, or class
  – Documentation comments are more heavily used in Java, and there are much better tools for working with them