<< Lecture 3 >>

Today's Plan

- Introduction to Netbeans
- Continue on Java's basics, including:
  - basic types (aka primitive types)
  - arrays
  - control flow
  - **classes** (likely Monday)

Let us start by reviewing last lecture's last code.

```
class Demo {

  // the following method computes the
  // area of a triangle
  static int getArea(int x,int y) {
     return x*y/2;
  }

  public static void main(String[] x) {
    int base = 10, area, height = 15 ;
    area = getArea(base, height);
    System.out.println("Area " + area);
  }

}
```

- We learned that our code always must belong in a class
- The **public static void main** method
- We created a method (**getArea**) that is **static** but **not public**, and **return**s an **int** value
- We learned about basic types (e.g., int, char, byte, boolean, etc.)
- Variables are defined with an **unchanging type**, and they must be assigned values of that type only

Using Netbeans

- I compiled and ran the previous code using the **javac** and **java** tools provided by the **jdk**, and I used notepad to write the code
- Using an IDE avoids the need to repeat this tedious process, saving us considerable time
- I demonstrate how to create a standard Java project:
  - I choose **New project**, and a new dialog appears, one asking for the type of project
  - I choose **Java** from the left list and **Java Application** from the right list, and I accept
  - The new dialog shows **the location of the project** (important!) and its suggested name
  - I choose **Demo** as the name of the project, and finally create the project
- Observe that Netbeans just created source code ready to work on!
- To compile and execute the code, you can press the play button (or by pressing key F6)
- To demonstrate the above feature, I copy the code of the Demo example class from the slides
- All should work as planned
- The standard output of the program (the text that goes into the console) is shown in the box below the code
- The left insets show the project files as well as the other projects
- You can also delete a project, and you can choose to erase all of its files in the process

Operations with basic types

- Operations with numbers:
  addition (+), subtraction (–), multiplication (*), division (/), modulo or remainder (%).

```
static int getArea(int x,int y) {
   return x * y / 2;
}
```

- Note that the above division returns **int**, because both operands (numerator and denominator) are of type **int**
- In mathematical operations, the returned type is the type of highest precision among the operands

```
System.out.println( 15-0.5 );   // 14.5
System.out.println( 15*2 );     // 30
System.out.println( 15*2.0 );   // 30.0
System.out.println( 15/2 );     // 7
System.out.println( 15/2.0 );   // 7.5
```

- Comparing primitive values (generate **boolean**):
  value equality (==), strict inequalities (<, >), non strict inequalities (<=, >=)

```
System.out.println( true==true ); // true
System.out.println( false==false);// true
System.out.println( -10==10 );    // false
System.out.println( -10<=10 );    // true
System.out.println( -10>=10 );    // false
```

- Note: all forms of integers are compared in the mathematical sense (including **char**s!)

```
System.out.println( 1 == 1.0 );   // true
System.out.println( 'A' < 60 );   // false
System.out.println( 15 > 14L );   // true
```

- 'A' is a character (single quotations), and has unicode value of 65; **char**s are, indeed, integers
- Operations with Booleans:
  and (&&), or (||), not (!)
- Bit-wise operations:
  and (&), or (|), xor (^), complement (~),
  bit shifts (<<, >>, <<<, >>>)
- For logic, prefer standard Boolean operations; they are more readable than bit operations
- **Note that Strings are excluded!**
  - Strings are **not** primitive/basic types

Operator precedence

(taken from https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html)

Operators	Precedence
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>

relational	`< > <= >= instanceof`
equality	`== !=`
bitwise AND	`&`
bitwise exclusive OR	`^`
bitwise inclusive OR	`|`
logical AND	`&&`
logical OR	`||`
ternary	`? :`
assignment	`= += -= *= /= %= &= ^= |= <<= >>= >>>=`

Arrays

- Arrays are our most basic data structure; they consist of a *sequence** of values arranged contiguously in memory (ideally)
- Arrays are blocks of values of the same type
- The following is an example definition of an array of chars in Java

```
char[] vowels = new char[5];
vowels[0] = 'a'; vowels[1] = 'e';
vowels[2] = 'i'; vowels[3] = 'o';
vowels[4] = 'u';
```

- The previous code defines variable **vowels** to be an array of chars (type **char[]**)
- Then, it assigns **vowels** a **new** empty array of **length 5** (i.e., it consists of 5 elements)
- Element **0** of the array (**vowels[0]**) is set to value '**a**', element 1 is set to '**b**', etc.
- The last element is **vowels[4]**; in an array of **length n**, indices range from **0 to n-1**, in this case, from 0 to 5-1
- To get the i-th element, we just do vowels[i]

```
System.out.println( vowels[0] );
System.out.println( vowels[1] );
System.out.println( vowels[2] );
System.out.println( vowels[3] );
System.out.println( vowels[4] );
```

- The above statements show that the elements in the array are kept in their designated order

- *An array can be thought of a collection of variables of the same type; for example, vowels being an array of length 5, we could think we have vowels_0, vowels_1, …, vowels_4, except that we do not need to type all the names*
- We can recover the length using **.length**:

```
System.out.println( vowels.length );
```

- But while the elements of the array can be changed individually (e.g., doing **vowels[0]='A'**), the size is fixed
- To make vowels an array of length, say, 10, we need to create a totally new array and refill it…
- To define an array with values quickly, do:

```
char[] vowels = {'a','e','i','o','u'};
```

- I should make clear early on that, as already mentioned, the equality (==) operation only works on primitive types
- The array is not a primitive type, so the == operation tests whether we have the same *instance* of the array (think of it as a location in memory)

```
int[] U = {1,2,3}, V = {1,2,3}, W = U;
System.out.println( U==V );    // false
System.out.println( U==W );    // true
System.out.println( V==W );    // false
```

Control flow

- The **if statement** is the most basic control flow structure
- Usage: **if (**boolean**)** {true case}
- Or: **if (**boolean) {true case} **else** {false case}
- **The parentheses are mandatory**
- However, the brackets are not necessary when the block consists of exactly one instruction
- If statements can consume several lines
- ….

<< Lecture 4 will resume from here >>