Lecture Set 4: More About Methods and More About Operators

- Methods
  - Definitions
  - Invocations
- More arithmetic operators
- Operator Side effects
- Operator Precedence
- Short-circuiting

main method

```java
public static void main(String[] args) {
    // statements here
}
```

- All projects and examples have defined this method
- No explicit call needed
- Parts of the line
  - Name = main
  - Parameter List = String[]
  - Return type = void
  - Access = public -- more on this later
  - Modifier = static
Other public static methods

- A static method is associated with a class
  - not an individual instance (object)
- Must have all of the same parts as the main
  
  ```java
  public static returnType name(argList){
    body
  }
  ```
- For example – defining a method to print a number of stars
  ```java
  public static void printStars(int count){
    for (int curr = 0; curr < count; curr=count+1){
      System.out.print("*");
    }
  }
  ```
- For example – defining a method to print a number of stars
  ```java
  printStars(3);
  System.out.println();
  printStars(77);
  ```

MethodStars.java

method information:
parameters and arguments

- parameter list
  - type name for each item in the list
  - e.g. (MyGrid grid, char where)
- argument list
  - expression for each item in the list
  - e.g. (grid, ‘t’)

- Matched between the arguments and the parameters based on position in the list
Non-main static public methods: defining, invoking and commenting

- Defined based on a name and a list of parameters
  ```java
  public static void name(parameterlist){
      body
  }
  ```

- Invoked by stating its name and giving an argument for each element of the parameter list
  ```java
  name(argumentlist);
  ```

- Each method must have a well defined purpose
  - That information goes into a comment before the method definition
  - Each parameter's purpose should be explained
  - Return value's purpose should be explained

Expressions

- Java “expressions” that yield values
  - e.g.
    ```java
    x
    x + 1 - y
    x == y && z == 0
    foo.equals ("cat")
    ```

- Expressions have values of a specific type (int, boolean, etc.)

- Expressions can be assigned to variables, appear inside other expressions, etc.
Expressions and Side Effects

- Some expressions can also alter the values of variables
  e.g. `x = 1`
- `x = 1` is an expression?
  - Yes!
  - Value is result of evaluation right-hand side of `=`
  - It also alters the value of `x`
- Such alterations are called side effects

Are the Following Legal?

- `int x, y; x = y = 1;`  
  Yes. Result assigns 1 to `x` and to `y`
- `int x = 0, y = 1; boolean b = false;`  
  `if (b = (x <= y)) { x = y; }`  
  Yes. Result assigns true to `b` and 1 to `x`
Other Expressions with Side Effects

- Java includes abbreviations for common forms of assignment
- Example: increment operations (Basically equivalent to $x = x + 1$
  
  $++x$ “Pre-increment”
  
  Increments $x$, returns the new value of $x$
  
  (“increment $x$, then return it”)

  $x++$ “Post-increment”
  
  Increments $x$, returns the old value of $x$
  
  (“return $x$, then increment it”)

- Same or Different
  
  - $x == x++$  always true
  
  - $x == ++x$  never true

- Compare
  
  - $x++ * y++$
  
  - $++x * ++y$
  
  - $++x * y++$
  
  - $x++ * ++y$

Other Assignment Operators

- Example: decrement operations (Basically equivalent to $x = x - 1$
  
  $--x$ “Pre-decrement”
  
  Decrements $x$, returns the new value of $x$

  $x--$ “Post-decrement”
  
  Decrements $x$, returns the old value of $x$
  
  “return $x$, then decrement it”

- General modification by constant
  
  - General form: `<var> <op with>= <constant>`

  - Examples
    
    $x += 2$  equivalent to $x = x+2$

    $x -= 2$  equivalent to $x = x-2$

    $x *= 2$  equivalent to $x = x*2$

    $x /= 2$  equivalent to $x = x/2$
Precedence

- Explains how to evaluate expressions
  - What is value of \(1 - 2 + 3 \times 4\)?
  - Precedence rules answer this question
    - Higher-precedence operators evaluated first
    - Example from math: “Please, Excuse my Dear Aunt Sally” or PEMDAS
      Multiple and divide (higher precedence) before you add and subtract (lower precedence)
  - Java follows “Aunt Sally’s Rules” … but what about other operators?

Java Precedence Rules

- parentheses: `( )`
- unary ops: `+x -x ++x --x x++ x-- !x`
- multiply/divide: `* / %`
- add/subtract: `+ -`
- comparisons: `< > <= >=`
- equality: `== !=`
- logical and: `&&`
- logical or: `||`
- assignments: `= += *= /= %=` (these are right to left associative)

Higher precedence on top
Examples

- \( x \times y + -z \)
  Same as \((x\times y) + (-z)\)

- \((x <= y && y <= z || w > z)\)
  Same as \(((x <= y) && (y <= z)) || (w > z)\)

- What is value of \(1 - 2 + 3 \times 4\)?
  \[
  = 1 - 2 + 3 \times 4 \\
  = 1 - 2 + (3 \times 4) \\
  = (1 - 2) + 12 \\
  = -1 + 12 \\
  = 11
  \]

Should You Rely on Precedence?

- No!
- The only ones people can remember are
  - “Please Excuse My Dear Aunt Sally” (PEMDAS)
  - And maybe unary and increment/decrement operators

- Bad:
  \[
  \text{if } ((2 \times x++ < 5 \times z + 3) && \neg w \neq x / 2) \\
  \]

- Better:
  \[
  \text{if } ((2 \times x++ < 5 \times z + 3) && \neg w \neq x / 2) \\
  \]

- Best:
  \[
  \text{if } (((2 \times x++) < (5 \times z + 3) && \neg w \neq (x / 2))) \\
  \]
Short-circuiting Example

- As soon as Java knows an answer – it quits evaluating the expression.
- What does Java print?
  ```java
  int x = 0, y = 1;
  if ((y > 1) && (++x == 0)){
      --y;
  }
  System.out.println (x);
  => 0
  ```
  - Why?
    - y > 1 is false
    - The result of && will be false, regardless of second expression
    - Java therefore does not evaluate second expression of &&
  - This treatment of &&, || is called short-circuiting
    - Subexpressions evaluated from left to right
    - Evaluation stops when value of over-all expression is determined

Examples

- What does Java print?
  ```java
  int x = 0, y = 1;
  if ((y >= 1) && (++x == 0)) {
      --y;
  }
  System.out.println (x);
  => 1
  ```
- What does Java print?
  ```java
  int x = 0, y = 1;
  if ( ((y > 1) && (++x == 0))
      |
      ( (y == 1) && (x++ == 0) ) ) {
      --y;
  }
  System.out.println (x);
  System.out.println (y);
  => 1
  0
  ```
Examples (cont.)

- What does Java print?
  ```java
  int x = 0, y = 0;
  while (x++ <= 4){
      y += x;
  }
  System.out.println (y);
  => 15
  ```

Programming with Side-Effects

Generally:
- Side effects in conditions are hard to understand
- Good programming practice
  - Conditions should be side-effect-free
  - Side effects should be in “stand-alone statements”
- Major Goal: Strive to create the most readable and maintainable code.
Primitive Types and their Hierarchy

- double
- float
- long
- int
- short
- byte

int x = 7.2;
double y = 6;
- Changing to something else Further Up this list is acceptable
  - called "Widening Conversion"
- Changing to Something else Further Down this list is not acceptable
  - called "Narrowing Conversion"
- Explicit casting needed for when you want to go lower in the list

Type Casting - implicit

Which of the following are legal?

- int x = 3.5;
  - Illegal: 3.5 is not an int
- float x = 3;
  - Legal: 3 is an int, which is also a float
- long i = 3;
  - Legal: 3 is an int, which is also a long
- byte x = 155;
  - Illegal: 155 is too big to be a byte (> 127)
- double d = 3.14159F;
  - Legal: 3.14159F is a float, which is also a double
**Mixed Expressions with Explicit Type Casting**

- What is result of
  
  ```java
  float x = 3 / 4;
  ```
  
  - x assigned value `0.0F`
  - Why?
    - 3, 4 are ints
    - So integer / operation is used, yielding 0, before upcasting is performed
  
  - To get floating point result, use explicit casting
    
    ```java
    float x = (float) 3 / (float) 4;
    ```
    
    - Assigns x the value `0.75F`

- Can also do following
  
  ```java
  float x = (float) 3 / 4;
  ```
  
  - Why?
    - `(float) 3` returns a value type float (`3.0F`)
    - 4 is an int
    - In this case, Java compiler uses widening conversion on “lower” type (here, `int`) to obtain values in same type before computing operation

- Or:
  
  ```java
  float x = 3.0f / 4;
  ```