**CMPE212 – Reminders**

- Assignment 1 is posted.
- Lab starts tomorrow (Thursday 8:30am) in JEFF155. We also have JEFF157, if we need it. Meet your TAs! Handle questions on:
  - JDK, Eclipse installation?
  - Exercises 1 and 3?
  - Assn 1.
- Don’t forget to do “Quiz 0”.
- Your onQ group name is the first name of your grader.

**Today**

- Continue Basic Java Syntax:
  - Constants
  - Casting
  - Building Expressions. What goes into an expression:
    - Variables
    - Literal Values
    - Keywords
    - Operators
    - Method Invocations
    - Punctuation

---

**Constant Attribute Declaration**

- Syntax:

  ```java
  [private|public] [static] final type ATTRIBUTE_NAME = literal_value;
  ```

- The Java keyword, `final` can be used to make sure a variable value is no longer ‘variable’.
- Usually these are declared `public static`.
- The value must be assigned – this part is no longer optional.

**Constants, Cont.**

- Java will not allow your program to change a constant’s value once it has been declared.
- For example:

  ```java
  final int NUM_DAYS_IN_YEAR = 365;
  final double MM_PER_INCH = 25.4;
  ```

- Note that constant names are all in upper-case, by convention.
- You can also declare constants inside a method.

---

**Type Casting**

- When a value of one type is stored into a variable of another type.
- Casting of primitive types in one direction is automatic, you do not have to deliberately or “explicitly” cast:

  ```java
  byte > short > int > long > float > double
  ```

**Type Casting - Cont.**

- For example in the statement:

  ```java
  double myVar = 3;
  ```

  the number 3 is automatically cast to a double (3.0) before it is stored in the variable `myVar`.
- However, if you tried the following:

  ```java
  int anotherVar = 345.892;
  ```

  the compiler would protest loudly because a `double` cannot be stored in an `int` variable without loss of precision. Wrong direction!
Casting Operator

- If you really want to cast in the other direction, then you must make an explicit cast. For example:

```java
int anotherVar = (int)345.892;
```

is legal. The "(int)" part of the statement casts the `double` to an `int`. The variable `anotherVar` would hold the value 345.

- Note how numbers are truncated, not rounded!

Aside – Casting Objects

- We will find (later) that you can cast objects using the casting operator too.
- Objects must have an inheritance relationship in order for the cast to succeed.
- For example:

```java
Integer aVal = new Integer(45);
Number aNum = (Number)aVal;
```

- The `Integer` class extends the `Number` class.
- More about this later...

What Makes an Expression?

- What are all the components available to a programmer to use to put a line of code together?
  - Variables
  - Literal Values
  - Keywords
  - Operators
  - Method Invocations
  - Punctuation

- We are working our way through this list!
- Look at Operators next:

Arithmetic Operators

- The standard binary arithmetic operators in Java are:
  - Addition (+)
  - Subtraction (-)
  - Multiplication (*)
  - Division (/)
  - Modulus or Remainder (%) (ie. 12 % 5 yields 2)

- All of these operations apply to all numeric primitive data types.
- All require values on both sides of the operator (why they are called "binary operators").
- Java does not have ** or ^ (exponentiation). Use Math.pow(x, y) to get x^y.

Integer Arithmetic

- Arithmetic operations between integers produce integer results by truncating the answer; fractional parts are discarded:
  - Examples:
    - 3 / 4 stores as 0
    - 4 / 4 stores as 1
    - 5 / 4 stores as 1

- If you have an integer on both sides of an arithmetic operator, the result will be an integer.
- What happens if there is a double on one side?

Mixed Type Arithmetic Expressions

- Suppose you have a "mixed type" expression involving an arithmetic operator.
- To evaluate the expression, Java will cast one side to match the other.
- For example if one side is an `int` and the other side is a `double`, the `int` will be automatically cast to a `double` before the operation takes place.
- For example:
  - 9 / 2 stores as 4
  - 9 / 2.0 stores as 4.5
  - 4 * 12 stores as 48
  - 4.0 * 12 stores as 48.0
Strings and the “+” Operator

• Not only can “+” operate on numeric values, but it can also handle String's on either or both sides.
• If one side is not a String, it will be changed to one, and then it will be concatenated to the String on the other side:
  4 + "you" stores as "4you"
  "apples" + "oranges" + 9 + 9 stores as "applesoranges999"
  3 + 7 + "little piggies" stores as "10little piggies"
• Expressions are evaluated from left to right, unless precedence rules apply.

Unary Arithmetic Operators

• Unary operators include “+” and “-”, where -aNum negates the value produced by aNum, for example.
• They also include the increment (++) and decrement (--) operators which increase or decrease an integer value by 1.
• Preincrement and predecrement operators appear before a variable. They increment or decrement the value of the variable before it is used in the expression.
• Example:
  int i = 4, j = 2, k;
  k = ++i - j;         // i = 5, j = 2, k = 3

Assignment Operators

= set equal to
*= multiply and set equal to
 /= divide and set equal to
- = subtract and set equal to
+= add and set equal to
%= modulo and set equal to

Logical Binary Operators

• Return either true or false.
  == equals to
  != not equals to
  > greater than
  < less than
  >= greater than or equal to
  <= less than or equal to
  & & logical “And”
  | | logical “Or”
Aside – "|" or "||"?

- What’s the difference?
- A single | or & always evaluates both sides of the expression, whether it is necessary or not.
- && stops if the left side is false, || stops if the left side is true.
- Called “short circuit evaluation”.
- The advantage is in the decrease in program run time.

Logical Operators - Cont.

- The one unary logical operator is “!”.
- Called the “Not” operator.
- It reverses the logical value of a boolean.
- For example:

  !(5 > 3) evaluates to false

Precedence Rules

- Operator precedence rules determine which operations take place in what order:
  - Unary operations and casting are done first.
  - Then *, /, %
  - Then +, -
  - Then <, >, <>, >=
  - Then &&
  - Then |, ||
  - Then =, *=, +=, -=, /=
- Use ‘[]’ to control order of operations, as the expression inside ‘[]’ will be evaluated before stuff outside of ‘[]’.
- Why are assignment operators always last?

Expressions

- Expressions are combinations of variables, literal values, operators, keywords, method calls, etc.
- For example:

  ```java
  int aNum = 4 + 3 * 7; // aNum is 25
  int aNum = (4 + 3) * 7; // aNum is 49
  (4 > 7) || (10 > -1) // yields true
  (5.5 >= 5) && (4 != 1.0) // yields true
  double circ = 3.14 * 2 * r;
  ```

What’s Left on Expressions?

- What are all the components available to a programmer to use to put a line of code together?
  - Variables
  - Literal Values
  - Keywords +
  - Operators
  - Method Invocations
  - Punctuation

Still need to do these ones.

50 Java Keywords

- abstract
- double
- int
- super
- assert
- else
- interface
- switch
- boolean
- enum
- long
- synchronized
- break
- extends
- native
- this
- byte
- for
- new
- throw
- case
- final
- package
- throws
- catch
- finally
- private
- transient
- char
- float
- protected
- try
- class
- goto
- public
- void
- const
- if
- return
- volatile
- continue
- implements
- short
- while
- default
- import
- static
- do
- instanceof
- strictfp

the ones we will use:

- Prof. Alan McLeod
The “Other” Keywords (from Oracle)

- `const` and `goto` are “not used by current versions of the Java programming language.”
- `native` is used in method declarations to specify that the method is not implemented in the same Java source file, but rather in another language.
- `synchronized` “when applied to a method or code block, guarantees that at most one thread at a time executes that code.”
- `transient` “indicates that a field is not part of the serialized form of an object.”
- `volatile` “specifies that the variable is modified asynchronously by concurrently running threads.”
Method Invocations

- Three aspects to consider:
  1. Naming the method.
  2. Providing arguments(s) or not.
  3. Doing something with the return value or not.

Method Invocations – 1. Naming the Method

- If the method is in the same class, then just use the name of the method.
  - (The compiler assumes the existence of the object reference to the current object as this. Or it assumes the name of the current object, when static)

- If the method is not in the same class you must first identify the object owning the method and obtain the method using the dot operator.
- If the method is declared as static, then you can invoke the method without instantiating the class that contains the method.

Method Invocations – 2. Providing Arguments for the Parameters

- If the method has been declared to accept arguments, i.e., it has a non-empty parameter list, then you must supply a matching list of arguments.
- If the method does not have any parameters, then you must still use empty brackets, (), when you invoke the method.

Method Invocations – 3. Using a Return Value

- A non-void method will return something.
- You can use that “something” in an expression, or just store it in a variable – your choice.
- The method has declared the type of that “something”.

- If the method was declared as void, you will not get a return value and you can only invoke the method by itself, not as part of an expression or an assignment statement.

Method Invocations - Examples

- See the MethodInvocations.java program.