# More Java

CS151

Chris Pollett

Sept. 12, 2005.

# Outline

- Apache Ant
- Lexical Elements in Java
- Variables and Types in Java

# Apache Ant

- A tool for building Java projects.
- Descended from Unix make.
- make uses a file called a Makefile to describe how to build a project.
- Ant uses a file called build.xml, which uses the Ant XML project language.
- Ant can be extended using Java classes.
- Available from http://ant.apache.org/

# Example build.xml File

```
<project name="MyProject" default="dist" basedir=".">
    <description>simple example build file</description>
    <!-- set global properties for this build -->
    <property name="src" location="src"/>
    <property name="build" location="build"/>
    <property name="dist"  location="dist"/>
    <target name="init">
          <!-- Create the time stamp -->
          <tstamp/>
          <!-- Create the build directory structure used by compile -->
          <mkdir dir="${build}"/>
    </target>
    <target name="compile" depends="init" description="compile the source " >
          <!-- Compile the java code from ${src} into ${build} -->
          <javac srcdir="${src}" destdir="${build}"/>
    </target>
    <target name="dist" depends="compile" description="generate the distribution" >
          <!-- Create the distribution directory -->
          <mkdir dir="${dist}/lib"/>
          <!-- Put everything in ${build} into the MyProject-${DSTAMP}.jar file -->
          <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
    </target>
    <target name="clean" description="clean up" >
          <!-- Delete the ${build} and ${dist} directory trees -->
          <delete dir="${build}"/>
          <delete dir="${dist}"/>
    </target>
</project>
```

# Running Ant

- To get ant to perform the tasks listed for a given target type:

  ant target <ret>

- The ant command looks in the current directly for a build.xml file to find this target in.
- If no target is specified then ant tries to execute the default target given by the default attribute of the <project> tag.

# Lexical Elements

- We now begin a detailed discussion of Java starting with lexical elements.
- These are the basic building blocks of the programming language:
  - characters
  - identifiers
  - literals
  - operators
  - expressions

# Character Set

- Java is written in Unicode.
- Unicode is an international 16 bit character set that contains encodings of most language used in the world.
- The older character encoding, ASCII (aka ISO-8859-1), is the first 128 characters.
- Java can also be localized to accommodate different locales.
- The default locale is U.S. English. This version performs conversion between ASCII and Unicode on the fly.

# Identifiers

- These are used in Java to denote the names of classes, methods, variables, etc.
- A Java identifier can begin with a letter, followed by letters or digits.
- Letters can be letters in any Unicode alphabet. For instance,δ.
- Letters also include (_) and ($).
- So the following are identifiers:

_myVar  $a  AClassName b22

# Primitive Types and Literals

- Constant values of each primitive type are called *literals*.
- The following are the primitive types in Java:
  - Boolean type: boolean.
    - (true, false)
  - Integer types: byte, short, int, and long
    - respectively 1, 2, 4, 8 byte signed integers. No unsigned type in Java. Ex -12, 23, etc. To write in octal begin a 0; to write in hexadecimal begin with an 0x or 0X
  - Character type: char
    - Ex: A b E, \040 (Octal Ascii) \u5496 (hex Unicode), escape sequences: \n, \", etc.
    - Character literals are written between single quotes ('a'); whereas, string literals ("hello") are written between double quotes
  - Floating-points: float and double. 4 and 8 byte IEEE-754 floating point
    - 2.5  23.f  23d  48.4F  48.4D  1e-9d  1.45e10f

# Operators and Expressions

- According to operator precedence (from greatest to least) Java has the following operators:

  exp++, exp--, ++exp, --exp, +exp, -exp, ~exp, !exp,
  exp1*exp2, exp1/exp2, exp1%exp2, exp1+exp2,
  exp1-exp2, exp1 <<exp2, exp1 >>exp2, exp1 >>>exp2,
  exp1 < exp2, exp1>exp2, exp1 <= exp2, exp >=exp2,
  exp1 == exp2, exp1 != exp2, exp1 & exp2, exp1 ^ exp2,
  exp1| exp2, exp1 && exp2, exp1 || exp2,
  exp1?exp2:exp3, var=exp, var +=exp, var -=exp,
  var *=exp, var /=exp, var %=exp, var <<= exp,
  var >>=exp, var >>>=exp, var &=exp, var ^=exp,
  var |=exp.

- An expression is either a literal or a variable, or an operator applied to expressions or variables. Can use ().
- Expressions are evaluated according parenthesization, followed by operator precedence, and then from left to right.

# Some Comments about these Operators

- x/y and x%y throw an ArithmeticException when y is 0 and x and y are of integer type.
- For float types, no exceptions are generated.
  - IEEE 754-1985 has two magic numbers in addition to the usual floating point numbers: NaN (not a number) and infinity. For example 0.0/0.0 evaluates to NaN. 5/0 evaluates to Float.POSITIVE_INFINITY, etc.
- + can also be used to concatenate strings:

  “object”+ “-” + “oriented” evaluates to “object-oriented”

  “object”+ ‘-’ + “oriented” evaluates to “object-oriented”

# Assignment Operators

- These are the operators +=, -=, *=, etc.
- They have the form

  var op= exp
- They are equivalent to

  var = (var) op (exp)
- However, var will only be evaluated once in the original expression.
- So a[i++] += i and a[i++] = a[i++] + i will give different results:
- For example if i=2, first gives a[2] = a[2] +3; second gives a[2] = a[3] +4;

# Variables and Types

- A **type** denotes the set of all legal values of that type.
- A **variable** refers to a location in memory where a value can be stored.
- Each variable is associated with a type and this restricts what possible values it can have
- The type of a variable is specified in a **variable declaration**.

# Variable Declarations

- The basic format of a variable definition is

  Type VarName_1 [=InitialValue_1], VarName_2 [=InitialValue_2],…

- For example,

  int a, b=5, c;

- Java supports two kind of types: *primitive* types and *reference* types. We've already looked at primitive type.
- Declarations without an initial value are assigned a default initial value. These are Integer 0; Floating-point 0.0; char \u0000; boolean false; Reference null.

# Type Compatibility and Conversion

- Type T1 is **compatible** with type T2 if a value of type T1 can appear whenever a value of type T2 is expected and  vice versa.
- **Type conversion** is the conversion of values one type into values of another type. For example, might convert a byte to an int.
- Converting a numeric type to a type with a smaller range is called **narrowing**.  Ex: long a=5; byte b= (byte)a;
- Converting a numeric type of a smaller range to larger range is called **widening**.

# Reference Types

- A **reference type** is a class type, interface type, or an array type.
- A reference type says where a given object is stored in memory.
- Reference types differ from C/C++ pointers in that Java references:
  - cannot do pointer arithmetic
  - cannot cast to any type
  - cannot be assigned arbitrary values

# Garbage Collection

- The job of deallocation unreferenced object is done by the **garbage collector** in Java.
- In C/C++ this has to be done by the programmer.
- Garbage collection involves examining the heap memory space to figure out which objects are referenced.
- One technique for determining unreferenced objects is reference counting.
- Because this can happen at any time the disadvantage of garbage collection is it is hard to predict the run-time of your code.