The UNIVERSITY of VERMONT

Benjamin Fenster
CS-100 (Damon)
19 January 2003

Notes – Introduction to Java

I. Introduction
   a. Everything is a class in Java. NO global variables or standalone functions.
   b. No .h files. "Header" and code are all in the same .java file.
   c. Generally, every class will have its own file. The file has the same name as the class.
   d. Even the directory has to be named correctly.
   e. Because of that, the environment can find the class from the name alone. (No #include is necessary)
   f. Inheritence
      i. "extends" at most once per class, no single inheritance
      ii. Everything inherits from something. "Object" is the only top-level class. If "extends" isn't used, Object is the default variable.

II. Objects
   a. Fields
      i. `visibility type name = value;`
      ii. Assume Private visibility normally, but it's not the default!
      iii. Can be static
         1. Shared among all instances.
         2. Can access fields through an instance or through ClassName.field
         3. Use for constants.
         4. "final" is like C++ "const" – the value can never change after it's initialized. Thus, it *must* be given an initial value.
         5. `final static public int mumble = 3;`
   b. Methods
      i. No 'virtual'! All methods are virtual. If you don't want it overridable, say "final."
      ii. Invoked through the object. MyObj.incr(3);
      iii. "this" exists.
      iv. Can have static methods
         1. Use for global functions.
         2. Invoked with class name: myClass.staticMethod();
         3. NOT virtual (the exception to the rule).
      v. Special static method called main()
         1. Must have an array of strings as its arguments (usually called 'args')
         2. Returns void
         3. Should be public
         4. Can have one per class, so can start running the program from anywhere.
         5. Can use main() as a test driver for the class if another isn't needed.
         6. Can have alternate versions of the program – run one class for one thing or run the other for something else.
   c. Types
      i. Classes
      ii. Primitives
         1. Similar to those in C++.
         2. byte, short, int, long
         3. boolean, char, float, double
         4. void
         5. The only non-objects in Java.
         6. Can "wrap" the primitives into objects though: Byte wraps byte. Integer wraps int. Etc.
      iii. Comparisons are boolean now, so can't say `int i = (j == 3)` anymore
      iv. char is 16-bit Unicode, so not limited to ASCII.

v. No “unsigned” variables anymore.
vi. Arrays
1. Type followed by brackets: int[]
2. Never give bounds in the declaration – just ask for the array.
3. Arrays are objects and thus can be assigned to fields of type Object.
4. Length is built into the object: a.length
5. Java checks bounds. No boundary errors anymore!
6. Insertions are slow! One of the things Sun didn’t get to fix before the user base grew.
7. Allocated using “new”: `int arr = new int[10];`
8. Can use set braces like in C++, but can use them even in open code.
vii. No pointers or references
1. All objects are passed by reference.
2. Fields store references – objects are never actually embedded.
viii. There is a ‘null’.
ix. No enums. Use sets of constants instead.
x. No bit packing (short of doing bitwise manipulation manually).

d. Object class
i. Remember that everything is an instance of Object.
Can have fields or arrays that hold anything!
ii. Built-in, overridable methods
1. toString() returns a string version of the object.
2. equals() returns true if the current object equals one passed in.
3. Others will be described later.

e. Usage
i. Must initialize new fields either with an explicit initializer or in the constructor.
ii. When instantiating a sub-class be sure to call the constructor for the super-class. Just say ‘super’ super(args). Don’t use the actual class name.
iii. No ‘delete’ operator because Java does garbage collection
iv. Can force an object to be deleted by removing all references (set to null)
v. Can write a ‘finalizer’ method to run at destruction, but it’s very rare to do so.
vi. Garbage collection is done only when needed (memory is low) and it’s hidden as much as possible so the user shouldn’t notice.

III. Statements
a. Java ‘s philosophy is to make bad code hard to write.
b. No gotos and no labels.
c. Loops can be named though, and ‘break’ can be used with a name. Can break out of the outer loop from an inner loop!
d. “synchronized” is used in multi-threading. More later.
e. Variable declarations are about the same as C++.
f. No implicit constructor calls. Use “new”.
g. Every variable *must* be initialized before being used. All control paths must receive initialized variables.
h. No forward declarations. Java reads the entire class to determine what’s legal before proceeding.
i. Operators
i. Missing -> * & sizeof and ,
ii. Gain an ‘instanceof’ keyword: `obj instanceof class`
iii. Gain a >>> operator to shift write logically – no sign extension.
iv. Gain a & and | operator that don’t short circuit – force all comparisons to always happen.
v. The == and != operators now determine whether two objects are *the same object*, not whether they are “equal.” Use .equals() for that.

IV. Comments and JavaDoc
a. /* */ and // just like C++

b. Javadoc Comments start with /**
c. Use Javadoc to build HTML documentation that documents your code automatically.
d. `@param name description` (don't include the type – pulled from code)
e. `@return description` (does it return null? Etc)
f. `@author name` (use for the whole class)
g. Others will be introduced as appropriate.

V. Packages
a. Arranged hierarchically.
b. Corresponds to the directory structure.
c. Package uvm.cs100 mirrors {root}/uvm/cs100/{code}

VI. Standard Libraries
a. Most start with java. or javax.
b. java.lang for String, Integer, System
c. java.util for Collections, Date
d. java.io for files and printing
e. java.math for… well… math
f. java.text for text manipulation
g. java.awt and javax.swing are the original (and new) user input/output classes.
h. Can use the entire pathname to access classes from different packages (like java.util.Collection).
i. Can also import by saying `import java.util.Collection;` and then just using the class name.
j. java.lang and the current package are imported by default.
k. Can import an entire package too: `import java.util.*;` but does not import sub packages.

VII. Useful Classes
a. System: in, out, err, exit()
b. Iterator
    i. Returned by many methods.
    ii. HasNext() method returns true if there's anything left in the list.
    iii. Next() returns Object object that can be cast to the appropriate type.
    iv. while(iter.hasNext()) Students = (Student)iter.next();