Still More Java

CS151

Chris Pollett

Sept. 14, 2005.

Outline

- More on reference types
- Arrays
- Statements
- Class Declarations

More on Reference Types

- Creating a variable of primitive type creates a storage location for its values.
- Declaring a variable of reference type does not create a storage location for an object or an array. Only a storage location for a reference is created.
- To create an actual object the new operator (initializer) must be used:

```
Point p;
p = new Point();
int[] ia;
ia = new int[3];
```

- An expression like r1==r2 for two reference types checks equality of references not the equality of the states of the objects. r1.equals(r2) tries to check for equality of states if it is implemented correctly.

Arrays

- There are two ways to create and initialize one dimensional arrays:
  1. With the new operator: new Type[n]
     For example: int i[] = new int[20];
  2. With the one dimension array initializer: {v1,…, vn}
     For example, char c[] = {‘a’, ‘b’, ‘c’};
- Both of these can be generalized to multidimensional arrays. So the following declarations are legal:
  int m[][] = {{1,2,3}, {4,5,6}};
  Point a[][][] =new Point[1000][50][345];
- To access an element of array we could do things like
  c[1] (value is ‘b’ above), m[1][2] (value is 6 above)
- arr.length return the length of the array arr for a 1-D array
- Accessing an element beyond the end of an array yields an IndexOutOfBoundsException

Statements

- There are two major categories of Java Statements: *simple statements* and *compound statements*.
- Simple statements include: expression statements, local variable declarations, break statements, continue statements, and return statements.
- Compound statements include statement blocks, selection statements, loop statements, and try-catch statements. try-catch will be describe on Monday.

Expressions, Blocks, local declarations

- Assignment expressions, increment/decrement expressions, object creation, and method invocation can be made into statements by adding a ';'.

  For example: x=5; m<<=k; i++; j--;

  p = new Point(); p.move(1,2); i= new int[3];

- A statement block consists of a sequence of statements or local variable declarations within braces:

```
{
    statement1;
    statement2;
 …
}
```

For example,

```
{ int i;/* start i's scope*/ i=10; int j=10;
…}//end i's scope
```

- Local variable declarations terminated by a ';' are also statements. Such declarations have a *scope* which is the extent of the code the variable is visible in. This scope begins at the declaration and ends at the end of the enclosing block.

Return and selection

- The *return statement* terminates the execution of a method and returns control to its caller. Its syntax is:

  return [Expression];

  The type of Expression must match the return type of the method that contains the return statement. Expression are allowed only if the return type of the method is not void.

- Java has two types of selection statements: if and switch:

  if(*Condition*) *Statement*

  if(*Condition*) *Statement1* else *Statement2*

  switch(*Expression*) {

  case *CaseLabel1*: //labels must be constant integer expressions

  Statement1 …

  case $CaseLabel_n$: //might use break statement to get passed later cases

  $Statement_n$

  default: //done if no labels apply.

  $Statement_{n+1}$}

Looping

- There are four kinds of loop statements in Java 5: while loops, do-while loops, for loops, and for-each loops
- These have the format:

  while(*Condition*) *Statement*

  do *Statement* while(*Condition*);

  for(*InitExpr*; *Condition*; *IncrExpr*) *Statement*

  for(*Type var* : *collObj*) *Statement*
- As for-each is new to Java 5, here is an example:

```
String[] moreNames = { "d", "e", "f" };
for (String name: moreNames)
  System.out.println(name.charAt(0));
```

Statement Labels, break and continue

- Each statement can have an optional label, which is an identifier:

  [Statement:] Statement

- Labels can be used in conjunction with break and continue statements to effect flow of control:

```
//my infinite loop
i=0;
outer: while (i < 5)
{ inner: for(int j=0; j <4; j++)
  { if(j == 2) continue outer;
     if(i == 1 && j==3) break; //could have written break inner;
        System.out.println(“”+(i+j));
  }
  i++;
}
```

Class Declarations

- Java Classes are the basic compilation units for Java. That is, they are units that can be compiled individually.
- A *class declaration* defines a class as well as a reference type.
- The basic syntax of a class declaration is:

  [*ClassModifiers*] class *ClassName* [extends *SuperClass*] [implements *Interface1*, *Interface2*,..]

  {

  ClassMemberDeclarations

  }
- ClassModifiers have the following effects: **<no modifier>** - class is accessible to all classes within the package, **public** - accessible by any class, **abstract** - class has an abstract method, **final** - class may not be extended.
- Each file may contain at most one public class and must end in the extension .java.

More on Class Declarations

- The **extends** clause specifies the superclass of a given class.
- The **implements** clause lists all the interfaces implements by the class.
- ClassMemberDeclarations consists of a list of member declarations. These declarations may be field, method, or nested class declarations.
- The order of these declaration does not matter to the compiler but good order ensures readability of your code.

Still More on Class Declarations

- A method declaration has the syntax:

  [*MethodModifiers*] *ReturnType*
  MethodName([*ParameterList*]){*Statement*}

- A field declaration has the syntax:

  [*FieldModifiers*] *Type FieldName1* [=*Initializer1*], *FieldName2*
  [=*Initializer2*],…;

- Modifiers that can be applied to methods, fields, and inner classes are: **<no modifier>** -package accessible, **public** - any class accessible, **protected** -package and subclass, **private** - only itself, **static** -shared by all instances of the class, and **final** - can't be overridden.
- Modifiers which apply only to methods are: **abstract** - means deferring implementation to a subclass, **synchronized** - is atomic in a multithread environment, **native** - is a method from some other language.
- Modifiers only used for fields are: **volatile** - the field might be changed by nonsynchronized methods, and **transient -** the field is not a persistent state of the instances**.**

Method Declarations

- The return type of a method is required. If the method doesn't return a value its return type should be void.
- Parameters in the parameter list have the form:

  [final] *Type ParameterName*
- Final means the parameter cannot be assigned a value in the method.
- The last element of the parameter list may be of the form "Type …" For example, void myMethod(String …args){/**/}. This is called using varargs. It means you can call myMethod with either a String array or with a sequence of String arguments. That is,

  String myArray[]; /*some stuff*/ myMethod(myArray);

  String str1, str2, str3; /* some stuff*/ myMethod(str1, str2, str3);

Style Conventions

- When writing Java, class and interface names should begin with uppercase letters. For example, Point.
- Fields and method names should begin with lower case letters. For example, point.
- Remaining words should be camel cased:

  MySuperDuperClass myNiftyField
  someObject