CMSC433, Spring 2002 Programming Language Technology and Paradigms Basic Java, continued

Alan Sussman
February 5, 2002

Administrivia

- Quiz returned today
- Revised version of project posted
  - clarify myCounter behavior
  - specify wire/socket format for timestamps
  - sample code for creating a scrolling text window
- Account issues?
  - change those passwords!

Last time

- Java basics
  - statements, expressions, operators
  - values – primitive types and references to objects
  - object operations – assignment, equals(), toString(), clone(), getClass(), instanceof, cast
  - arrays

Array declarations

- **int[] A** and **int A[]** have identical semantics
  - declares **A** to be a variable containing a reference to an array of ints
- **int[] A[], B;**
  - **A** is a ref to an array of refs to arrays of ints
  - **B** is a ref to an array of ints
- None of these allocate an array
- **A = new int [10];** allocates an array of 10 ints, and makes **A** be a reference to it

Array example

```
int[] array1 = {1, 3, 5};
int[][] a = new int[10][3];
// a.length == 10
// a[7].length == 3

a[5][2] = 42;
a[9] = array1;
// a[9][2] == 5

// use of array initializers
int[][] twoD = {{1, 2, 3}, {4, 5}, {6}};
Object [] args = {"one", "two", a};
main(new String [] {"a", "b", "c"});
```


String

- A class for representing *non-mutable* strings
- *string constants* converted to **String**
- + does string concatenation
- In some contexts objects automatically converted to **String** type
- Example:

```
public static void printArray(Object [] a) {
    for (int i = 0; i < a.length; i++)
        System.out.println("a[" + i + "] = " + a[i]);
}
```

Object/memory allocation

- Only way/time an object gets allocated is:
  - by executing **new**
    - one object per invocation of **new**
  - by having an array constant (e.g., {5, -5, 42})
  - by having a string constant (e.g.,"Hello world")
- Declaring a reference variable does *not* allocate an object
- Allocating an array does not automatically allocate the contents of the array

Allocation (cont.)

- Multi-dimensional array allocation
  - **int [][] a = new int [10][10];**
  - equivalent to, but faster than:
    int [][] a = new int [10][];
    for(int i=0; i<10; i++) a[i] = new int[10];
- No explicit deallocate required, nor allowed

Garbage collection

- Java uses garbage collection to find objects that cannot be referenced
  - i.e. do not have any pointers to them
- GC not a major performance bottleneck
  - fast garbage collectors have been implemented
  - on many commercial systems, GC runs on a single processor of a multiprocessor

Other Java notes

- Forward references resolved automatically
  - can refer to method/variable defined later
- Integer division by zero raises an exception
- Integer overflow drops extra bits
- Floating point errors create special values
  - NaN, POSITIVE_INFINITY, …
- Separate name spaces for methods, classes, variables, …
  - can produce confusing error messages

What's missing

- preprocessor (#include, #define, …)
- structs and unions
- enumerated types
- bit-fields
- variable-length argument lists
- multiple inheritance (of implementation)
- operator overloading
- templates/ parameterized types
  - GJ (generic Java)

Object-oriented programming in Java

Java Classes

- Each object is an instance of a class
  - an array is an object
- Each class is represented by a class object
  - of type **Class**
- Each class extends *one* superclass
  - **Object** if not specified
  - except class **Object**, which has no superclass

Classes (cont.)

- Each class has methods and fields/variables
  - variables hold primitive (built-in) values or object references
- Only use '.' to access object fields
  - e.g., **x.y(a.b)**
- Most methods invoked using C++ virtual method semantics
  - except static, private and final methods

Class modifiers

- **public** – class visible outside package
- **final** – no other class can extend this class
- **abstract** – no instances of this class can be created
  - only instances of extensions of the class

class Complex – a toy example

```
public class Complex {
 double r, i;
 Complex(double r, double i) {
  this.r = r;
  this.i = i;
 }
 Complex plus(Complex that) {
  return new Complex(
              r + that.r,
              i + that.i);
 }
 public String toString() {
  return "(" + r + "," + i + ")";
 }
```

```
public static void main(String[] args) {
  Complex a = new Complex(5.5,9.2);
  Complex b = new Complex(2.3,-5.1);
  Complex c;
  c = a.plus(b);
  System.out.println("a = " + a);
  System.out.println("b = " + b);
  System.out.println("c = " + c);
 }
}
```

prints:

a = (5.5,9.2)
b = (2.3,-5.1)
c = (7.8,4.1)

Class details

- Method names can be overloaded
  - method invoked is determined by both its name and the types of the parameters
  - resolved at compile-time, based on compile-time types
- Methods can also be overridden
  - define a method also defined by a superclass
  - arguments and result types must be identical
  - resolved at run-time, based on type of object method is invoked on

Classes and methods

- **this** refers to the object the method is invoked on
- **super** refers to the same object as **this**
  - but used to access methods/variables in superclass
- Methods
  - can be declared in both classes and interfaces
    - only implemented in classes
  - must have a return type
    - except constructors
    - **void** can be used only as a return type
  - references to objects or arrays can be returned