CMSC433, Spring 2001 Programming Language Technology and Paradigms Basic Java

Alan Sussman
September 17, 2001

Administrivia

- C++ project due Thursday, 6PM
- We have a TA - Yu-Lin (Tristan) Wen
  - email and office hours posted
- Java project posted soon
  - a simple Web server
  - due Wednesday, Oct. 3
- Java readings from *Thinking in Java*
  - we'll do parts of many chapters, but definitely read Chapter 1 for an overview
  - I'll add some suggestions on Web page, but use it as a reference

Last time

- Casts
  - static, dynamic, const, reinterpret
- Chaining
  - for calls to void and copy constructors, operator=
- Namespaces
  - to help alleviate name clash problems
- Exceptions
  - are like objects, and can be thrown/caught

What Makes Java Different

- Java fully specified
  - e.g., constants, size of types, array bounds checking, no dangling pointers
  - language specification intended to completely specify the behavior of *all* programs
    - not just correct ones
    - all runtime errors must be caught
  - KISS principle applied
    - many useful, but not essential, features from C++ not included (operator overloading, templates, multiple inheritance, standalone functions, …)

Java semantics

- Machine architecture insensitive
  - size of machine word
  - floating point format (must use IEEE 754)
  - big/little-endian
- Compiled to machine independent byte code
- Many C++ programs break when moved to machine with different word size or endianness

Java security

- Can strictly limit access to code
  - *untrusted* code can't access files and are limited in network connectivity by default
- Compiled code (byte codes) can be verified for correctness (by another program)
- But security bugs are still possible
  - *e.g.*, denial of service, insecure mode code
  - but *all* C++ programs run in an insecure mode

Java features

- Strong type system
- Multi-threading and synchronization
- Garbage collection
- Exceptions
- class **Class**
  - classes are objects too
- class **Object**
  - the class all classes are derived from

Java libraries

- Utilities
  - collection classes, Zip files, internationalization
- GUIs, graphics and media
- Networking
  - sockets, URLs, RMI, CORBA
- Threads
- Databases
- Cryptography/security

Java Basics

- Mostly C++ syntax
- Statements
  - empty, expressions, blocks {}
  - control flow (if, switch, while, for, …)
  - throw and try/catch/finally
  - synchronized
  - no goto

Expressions

- Mostly C/C++ syntax
- Standard math operators: +, -, *, /, %
- Bit operations: &, |, ^, ~, <<, >>, >>>
- Update ops: =, +=, -=, *=, /=, …
- Relational ops: <, <=, ==, >=, >, !=
- Boolean ops: &&, ||, !
- Conditional expression: b ? e1 : e2

Class operations

- Select method/variable/class/subpackage: .
- Class operators: new, instance, (Class)
- No pointer operations: *, &, ->

Hello World example

```
public class HelloWorld {
   public static void main(String [] args) {
      if (args.length == 1)
         System.out.println("Hello " + args[0]);
      else
         System.out.println("Hello world");
   }
}
```

Naming conventions

- Classes/Interfaces start with a capital letter
- packages/methods/variables start with a lowercase letter
- for names with multiple words, CapitalizeFirstLetterOfEachWord
- don't use underscores
- CONSTANTS all in uppercase

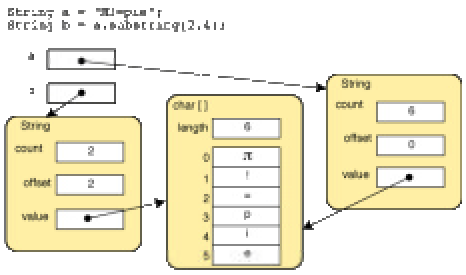
Values

- Object reference: null, or reference to object
- boolean – a built-in type
- char – Unicode; 16 bits
- byte/short/int/long – 8/16/32/64 bits, signed
- float/double – 32/64 bits IEEE 754

Objects and references

- all objects allocated on the heap
- no object can contain another object
- all class variables/fields are references to an object
- Reference is like a C++ pointer, except
  - can only point to start of heap-allocated object
  - no pointer arithmetic allowed
  - use . instead of -> to access fields/methods

String example



Object operations

- = assignment
  - for object references: copies reference, not object
- == equality test
  - for object references: true if both are references to same object
- foo.equals(bar)
  - intended to compare contents of objects
  - by default same as ==, but can/should be overridden
- foo.toString()
  - returns String representation of the object, can/should be overridden

More Object operations

- **foo.clone()**
  - returns shallow copy of **foo** (not supported on all Objects)
- **foo.getClass()**
  - returns class of **foo** (result is of type **Class**)
- **foo instanceof Bar**
  - true if object referenced by **foo** is a subtype of class **Bar**

More Object operations

- **(Bar) foo**
  - run-time exception if type of the object referenced by **foo** is not a subclass of **Bar**
  - compile-time error if **Bar** is not a subtype of **foo** (i.e. it always throws an exception)
  - doesn't transform anything, just allows treating the result as if it were of type **Bar**

Arrays

- a special kind of object (with lots of syntax)
- can declare arrays of any type
- have one instance variable: length
- also have contents indexed with a subscript from 0 … length-1
- can be initialized using $\{val_0, val_1, \ldots, val_n\}$ notation
  - inefficient for large arrays

Array declarations

- Little surprising for C/C++ programmer
- **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"});
```


CMSC433, Spring 2001 Programming Language Technology and Paradigms Basic Java, continued

Alan Sussman
September 20, 2001

Administrivia

- C++ project due today, 6PM
  - turn in something on time, even if you keep working until 6AM
  - your grade will be the *higher* of the on-time submission and 80% of the late submission
- Java project posted later today
  - web server will be up and running for testing soon
  - we'll talk about it more on Tuesday

Last time

- Java basics
  - statements, expressions
  - naming conventions
  - values are either object references or primitive types
- Objects
  - all dynamically allocated
  - only accessed through, and can contain, object references
  - many special object operations available
    - =, ==, equals(), toString(), clone(), getClass(), instanceOf
- Arrays
  - objects with a *length* instance variable
  - array name is a reference to the array object

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 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

Instance variable / method modifiers

- Visibility/access
  - **public** – visible everywhere
  - **protected** – visible within same package or in subclass
  - **package** (default) – visible within same package
  - **private** – visible only within this class
- **static** – a class method or variable

Instance variable modifiers

- **transient** – not stored when object serialized
- **volatile** – don't assume the variable hasn't changed since the last time it was accessed
  - might be modified by another thread that doesn't have a lock on the object
- **final** – can't be changed; must be initialized in declaration or in constructor

Method modifiers

- **abstract** – no implementation provided
  - class must be abstract
- **final** – this method cannot be overridden
  - useful for security
  - allows compiler to inline method
- **native** – implemented in another language
- **synchronized**
  - locks object before method is executed
  - lock released after method finishes

Method arguments

- Only pass-by-value
  - but object parameters are references to heap objects that can be changed
- Only arguments used to distinguish methods
  - not return types
- Syntax same as C/C++