CMSC433, Spring 2002 Programming Language Technology and Paradigms Basic Java

Adam Porter

Jan 31, 2002

Administrivia

- Project 1 posted today, due Feb. 13, 6PM
- 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 refererence

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

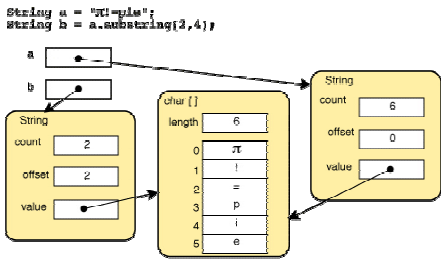
Values

- Object reference: null, or ref 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 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 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**