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

```java
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**

```
String s = new String("abc");
```

**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 declared type 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, …, 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