Basic Java

```
// Hello.java-the traditional first application
public class Hello
   {
   public static void main (String [] args)
      {
      System.out.println ("Hello JAVA!");
      }
   }
```

Similar to C/C++

probably accounts for it's popularity

```
// Echo.java-the traditional second application
public class Echo
   {
   public static void main (String [] args)
      {
      for (int i = 0; i < args.length; i++)
         System.out.println (args [i]);
      }
   }
```

Basic Java

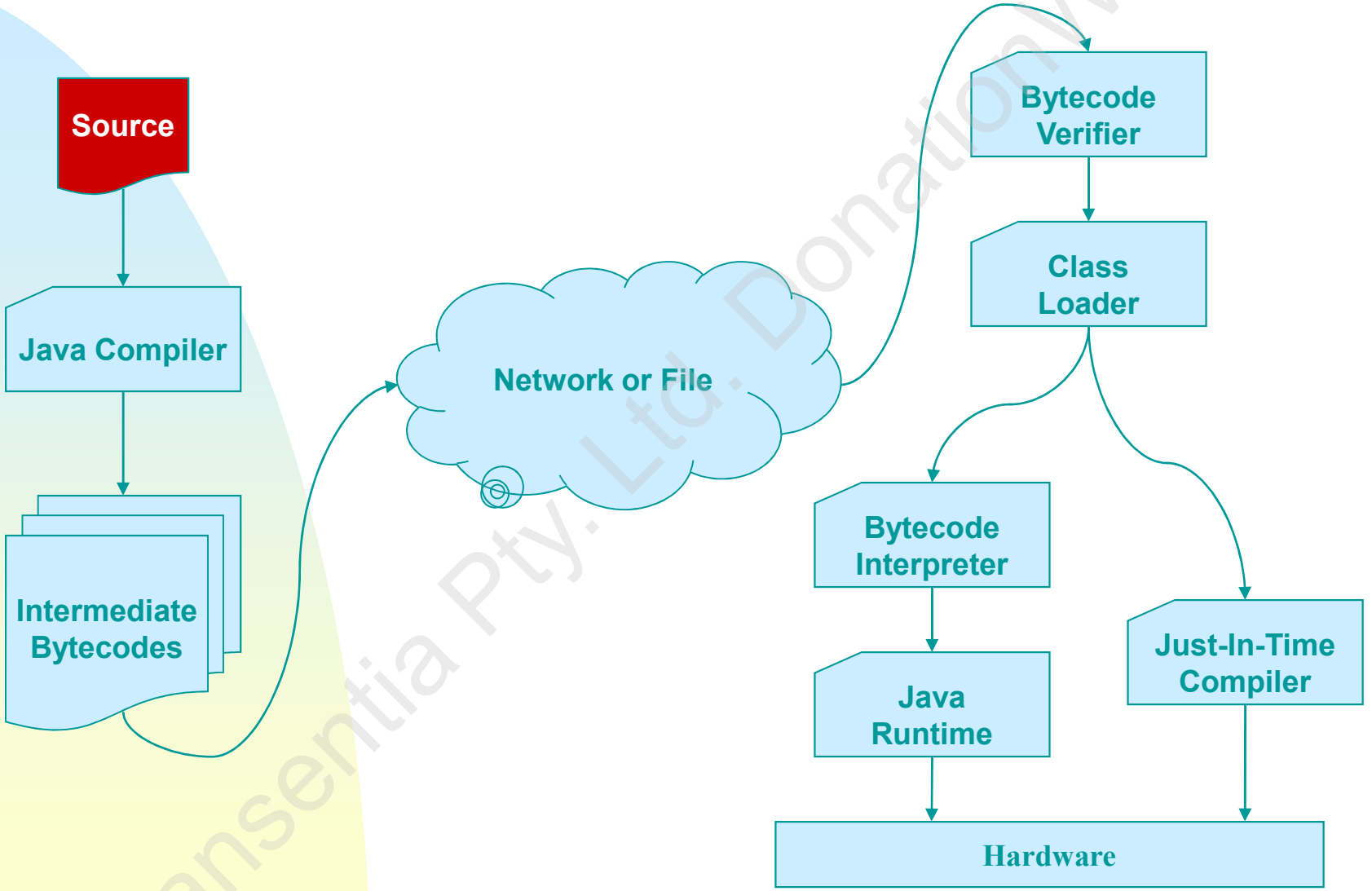

```
// Echo11.java-the traditional second application
// modified for Java 1.1
public class Echo11
  {
  private static PrintWriter stdout = new PrintWriter (System.out);

  public static void main (final String [] args)
    {
    for (int i = 0; i < args.length; i++)
      stdout.println (args [i]);
    stdout.flush ();
    }
  }
```

Basic Java

"…while it is the ++ operator that gives the C++ language its name, it also led to the first joke made by anti-C++ programmers who have long complained about the bug-ridden code that is too often produced by sloppy C++ coding. This joke points out that even the name of the language contains a bug: 'After all, it should really be called ++C, since we only want to use a language after it has been improved.'"

Basic Java



Basic Java

- Tokens
  - whitespace; comments: //, /*…*/, /**…*/; miscelleny: [, {, ++, etc.
  - Unicode
    - very low-level facility
      - dealt with before compiler proper sees input
      - cf. C++ trigrams
    - 16 bit encoding
    - all chars and Strings are Unicode
      - “\u005c\u0022” == “\””
      - ‘\u2620’ == 

Basic Java

- Idea is for all source to be Unicode too:

```
public class HelloUni
  {
  public static void main (String [] args)
    {
    System.out.println ("Hello, World");
    }
  }
```

```
public class HelloUni
  {
  \u0070\u0075\u0062\u006c\u0069\u0063
\u0073\u0074\u0061\u0074\u0069\u0063
\u0076\u006f\u0069\u0064 \u006d\u0061\u0069\u006e
\u0028\u0053\u0074\u0072\u0069\u006e\u0067 \u005b\u005d
\u0061\u0072\u0067\u0073\u0029
    \u007b

\u0053\u0079\u0073\u0074\u0065\u006d\u002e\u006f\u0075\u007
4\u002e\u0070\u0072\u0069\u006e\u0074\u006c\u006e
\u0028\u0022\u0048\u0065\u006c\u006c\u006f\u002c
\u0057\u006f\u0072\u006c\u0064\u0022\u0029\u003b
    \u007d
  }
```

Basic Java

- No preprocessor
  - hooray!
  - no #define
    - public static final int CONST = 99;
    - no macros
  - no #include
    - import
    - packages
  - no conditional compilation
    - constant folding instead: if (false) elided

Basic Java

- Reserved words
  - 59. all lower case: if; class; interface; try, etc.
  - const, goto: there but do nothing
    - *"…may allow a Java compiler to produce better error messages if these C++ keywords incorrectly appear in Java programs."*
- Identifiers
  - whîteThé; X != x; a_double; the$thing
  - fully qualified: java.lang.String.toString ()
- Literals
  - "Hello World"; 0xCAFEBABE; 3.14D; 99L; 0777; "this" + "that" == "thisthat"; '\u2297' == '⊗'

Basic Java

- Basic flow of control
  - *mostly* as in C/C++
    - ☞ no goto
      - labelled statements

```
TEST:
if (check (i))
  {
  for (int j = 0; j < 10; j ++)
    {
    if (j > i) break;
    if (a [i][j] == null)
      break TEST;
    // do something ...
    }
  }
// break TEST goes to HERE!
```

```
if (condition0)
  {
  statement(s);
  }

else if (condition1)
  {
  statement(s);
  }

else
  {
  statement(s);
  }
```

```
for (initial; condition; progress)
  {
  statement(s);
  }
```

```
for ( ; ; )
  {
  statement(s);
  if (condition)
    break;
  statement(s);
  }
```

```
for ( ; ; )
  ;
```

```
while (condition)
  {
  statement(s);
  }
```

```
do
  {
  statement(s);
  } while (condition);
```

```
switch (expression)
  {
  case v0:
    statement(s);
    break;

  case v1:
  case v2:
    statement(s);
    break;

  default:
    statement(s);
    break;
  }
```

Basic Java

- Types
  - 4 categories
    - primitive types
    - class types
    - array types
    - interfaces
- Data values
  - 2 categories
    - primitive values
    - references
      - type-bound pointers

Basic Java

- Primitive types
  - 3 categories
    - arithmetic
      - integral: byte (8); short (16); int (32); long (64)
      - floating point: float (32); double (64)
      - promotion: byte⇨short⇨int⇨long⇨float⇨double
      - all types are signed
    - boolean
      - true/false
      - no automatic conversions: if (some_value) not allowed
    - character
      - Unicode (16 bits), ASCII is a standard subrange
      - always unsigned: (int) char may produce a negative number

Basic Java

- Operators
  - standard C set
  - additions
    - >>> and >>>=
      - all integral types are signed, so >> uses sign extension
      - 11101000 (-24) >> 2 gives 11111010 (-6)
      - 11101000 >>> 2 gives 00111010 (58)
    - & and | (vs. && and ||)
    - string concatenation: +
    - instanceof
  - subtractions
    - , operator restricted use; no sizeof operator

Basic Java

Classification	*Operator*
Arithmetic	+ - / * % 'unary -'
Relational	< <= >= > == !=
Logical	&& || ! & |
Bitwise	~ ^ << >> >>> | &
Miscellaneous	?: (type) instanceof new
Assignment	= += -= *= /= %= >>= >>>= <<= &= ^= |=
Autoinc(dec)rement	++ --

```
short countBitsInAnInt ()
  {
  int u = ~0;  // set u to all 1s
  short n = 0;
  do
    {
    n ++;
    } while ((u >>>= 1) != 0);
  return (n);
  }
```

```
{
x = 99;
y = 1;
if ((x += y) == 100)
  something gets done ...;
```

```
a = b = c = 0;
```

```
{
int x = 99,
    y = 0;
if ((y != 0) && ((x / y) > 10))
  nothing gets done here...;
```

```
public class Auto
  {
  public static void main (String [] args)
    {
    int v;

    v = 0;
    System.out.println ("v ++: " +  v ++);

    v = 0;
    System.out.println ("++ v: " +  ++ v);
    }
  }
```

Basic Java

- Casting
  - type conversions
    - long aLong = (long) anInt;
      - as in C/C++
  - assertions
    - not the same as in C/C++
    - Circle c = (Circle) hashtable.get (“key”);

assertion: the ‘real’ (dynamic) type of this object is Circle (“downcasting”)

defined (static) return type is Object

    - if the assertion turns out to be false a ClassCastException is thrown

Basic Java

- Miscelleny
  - class-level declarations can be in any order
  - can't redefine variables within a method
  - void differences
    - no cast -> void
    - no void in parameter lists
    - no pointers, so no void *
  - also missing:
    - bitfields
    - typedefs
    - varargs
    - enums

```
public void method ()
  {
  int i = 0;

  if ( … )
    {
    int i; // Variable 'i' is already defined in this method.
    …
    }
  }
```

Basic Java

- Arrays
  - conglomerations of data
    - all of identical type
  - usually must be “new’ed”
    - different to most other languages
    - possibly more flexible
    - can be created statically if initial contents known

```
int [ ] knownThings = {7, 3, 8, 13};
```

  - possess a ‘length’ attribute

```
int [] array = new int [10];
...
public void fillArrray ()
  {
  for (int x = 0; x < array.length; x ++) array [x] = x;
  }
```

  - anonymous arrays

```
long first5Total = sum (new int [] {1, 2, 3, 4, 5});
stdout.println (new char [] {‘h’, ‘i’});
```

Basic Java

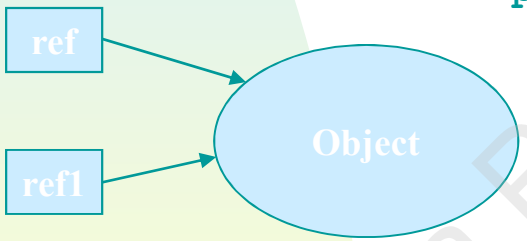
- Class types
  - a class is a recipe; an object is an instance of that recipe (created by the ‘new’ operator)
    - we specify classes but deal with (references to) objects
      - c.f.. recipes and cakes
    - the JVM loads classes dynamically “as needed” (i.e. when the program needs to make an object)
      - may also fetch a class from across a network before loading it
- Interfaces
  - a mechanism for specifying the way in which an object should be used (a ‘protocol’ or ‘contract’)

Basic Java

- Primitive values
  - indivisible—ints, chars, etc.
  - associated location has a *fixed* type
  - unshared
- References
  - Java's version of pointers "behind the scenes"
    - always bound to a given type (hierarchy)
  - a given location may store values of many different (perhaps related) types over time
  - composite data
  - addresses—arrays (incl. Strings); objects

Basic Java

- NO POINTERS
  - A Good Thing: “goto of data structures”
  - objects handled by *reference*
  - Obj == Obj1 doesn’t work as expected:
    - Obj.equals (Obj1)




```
public static void main (String [] args)
   {
   if (new Integer (3) == new Integer (3))
     System.out.println ("Same references!");
   if (new Integer (3).equals (new Integer (3)))
     System.out.println ("Same values!");
   }
```

  - *all* objects must be new’ed (*including* arrays)

Basic Java

- Garbage collection
  - programmers don't dispose of allocated memory by hand—the runtime system does this
    - generally A Good Thing—bye bye memory leaks
- Exceptions
  - an 'out-of-band' signalling mechanism
  - similar to C++ but better
    - the 'finally' section is Another Good Thing
- Threads
  - *"multiple concurrent loci of execution"*
    - i.e. allowing more than one thing to be done at a time…
  - synchronized keyword
    - resource control for competing threads