**Ahead-of-time compiler**

Compile time:
- C source code → C compiler → x86 assembly code → x86 assembler → x86 machine code

Run time:
- x86 machine code → x86 computer → Output

**Typical Compiler**

Source Program → Lexical Analyzer → Syntax Analyzer → Semantic Analyzer → Intermediate Code Generator → Code Optimizer → Code Generator → Target Program

**Interpreter**

Source Program → Interpreter = virtual machine → Output

**Compilers... that target interpreters**

Java source code → Java Compiler → Java bytecode

Java bytecode → Java Virtual Machine → Output
Interpreters... that use compilers.

Source Program -> Compiler -> Target Program -> Virtual Machine

Data -> Output

JIT Compilers and Optimization

Java source code -> javac -> bytecode

Just-in-time compiler

x86 machine code

Output

Virtual Machine Model

High-Level Language Program

Bytecode compiler

Compile time

Virtual Machine Language

Ahead-of-time compiler

Run time

Virtual machine (interpreter)

Native Machine Language

On translation, layout, and implementation
We show natural, common, or conventional translations.

Java: No guarantee of this implementation/layout.
Language is (mostly clean) abstraction.

C: Much of implementation/layout guaranteed.
Language exposes many machine details.
Data in Java

Integers, floats, doubles, pointers – same as C
Null is typically represented as 0
Characters and strings
Arrays
Objects
  pointers? called ‘references’ – much more constrained

Data in Java

Arrays
  Every element initialized to 0 or null
  Length specified in immutable field at start of array (int – 4 bytes)
  array.length returns value of this field
  Since it has this info, what can it do?

  int array[5]:

  C
  22 22 22 22 22
  0 4 20 24
  Java
  5 00 00 00 00 00

Data in Java

Characters and strings
  Two-byte Unicode instead of ASCII
  Represents most of the world’s alphabets
  String not bounded by a ‘\0’ (null character)
  Bounded by hidden length field at beginning of string

  the string ‘CS 240’:

  C: ASCII
  43 53 20 32 30 30 \0
  0 1 4 7
  Java: Unicode
  6 00 00 43 00 53 00 20 00 32 00 34 00 30

Data in Java

Bounds-checking sounds slow, but:
  1. Length is likely in cache
  2. Compiler may store length in register for loops.
  3. Compiler may prove that some checks are redundant.
Data structures (objects) in Java

Objects are always stored by reference, never stored inline.
Include complex data types (arrays, other objects, etc.) using references

C

```
struct rec {
    int i;
    int a[3];
    struct rec *p;
};
```

Java

```
class Rec {
    int i;
    int[] a = new int[3];
    Rec p;
};
```

Pointers/References

Points in C can point to any memory address
References in Java can only point to [the starts of] objects
And can only be dereferenced to access a field or element of that object

C

```
struct rec { 
    int i; 
    int a[3];
    struct rec *p;
};
struct rec* r = malloc(...);
some_fn(&r->a[1]); //ptr
```

Java

```
class Rec {
    int i;
    int[] a = new int[3];
    Rec p;
};
Rec r = new Rec();
some_fn(r.a[1]); // ref, index
```

Casting in C

We can cast any pointer into any other pointer; just look at the same bits differently

C

```
struct BlockInfo {
    int sizeAndTags;
    struct BlockInfo* next;
    struct BlockInfo* prev;
};
typedef struct BlockInfo BlockInfo;
...
BlockInfo *b;
BlockInfo *newBlock;
newBlock = (BlockInfo*) ( (char*) b + x );
```

Java

```
struct BlockInfo {
    int sizeAndTags;
    struct BlockInfo* next;
    BlockInfo* prev;
};
typedef BlockInfo BlockInfo;
...
BlockInfo *b;
BlockInfo *newBlock;
newBlock = (BlockInfo) ( (char*) b + x );
```
Type-safe casting in Java

Can only cast compatible object references

```
// Vehicle is a super class of Boat and Car, which are siblings
Vehicle v = new Vehicle();
Car c1 = new Car();
Boat b1 = new Boat(); // ok, everything needed for Vehicle
Vehicle v2 = v1; // ok, v1 is already a Vehicle
Car c2 = new Boat(); // incompatible type - Boat and Car are siblings
Car c3 = new Vehicle(); // wrong direction; elements in Car not in Vehicle
Boat b2 = (Boat) v; // runtime error; Vehicle does not contain propellers
Car c4 = (Car) v2; // ok, v2 started out as Car
Car c5 = (Car) b1; // incompatible types, b1 is Boat
```

How is this implemented / enforced?

Java objects

```
class Point {
    int x;
    int y;
    Point() {
        x = 0;
        y = 0;
    }
    boolean samePlace(Point p) {
        return (x == p.x) && (y == p.y);
    }
    String toString() {
        return "(" + x + " + y + ");
    }
}
```

Implementing dynamic dispatch

```
Point p = new Point();
p->header = ...;
p->vtable = &Point_vtable;
Point_constructor(p);
return p.samePlace(q); return p.vtable[0](this=p, q);
```
Subclassing

How do we access superclass pieces?
- fields
- inherited methods

Where do we put extensions?
- new field
- new method
- overriding method

class ColorPoint extends Point{
    String color;
    boolean getColor() {
        return color;
    }
    String toString() {
        return super.toString() + "[" + color + "]";
    }
}

dynamic (method) dispatch

Java:
Point p = ???;
return p.toString();
return p.vtable[1](p);

What happens (pseudo code):

color:Point.samePlace()
Point vtable
Code: Point.toString()

ColorPoint object

Point vtable

x
y
color