CS614: Advanced Compilers

Java Bytecode and Class Files

Manas Thakur
CSE, IIT Bombay

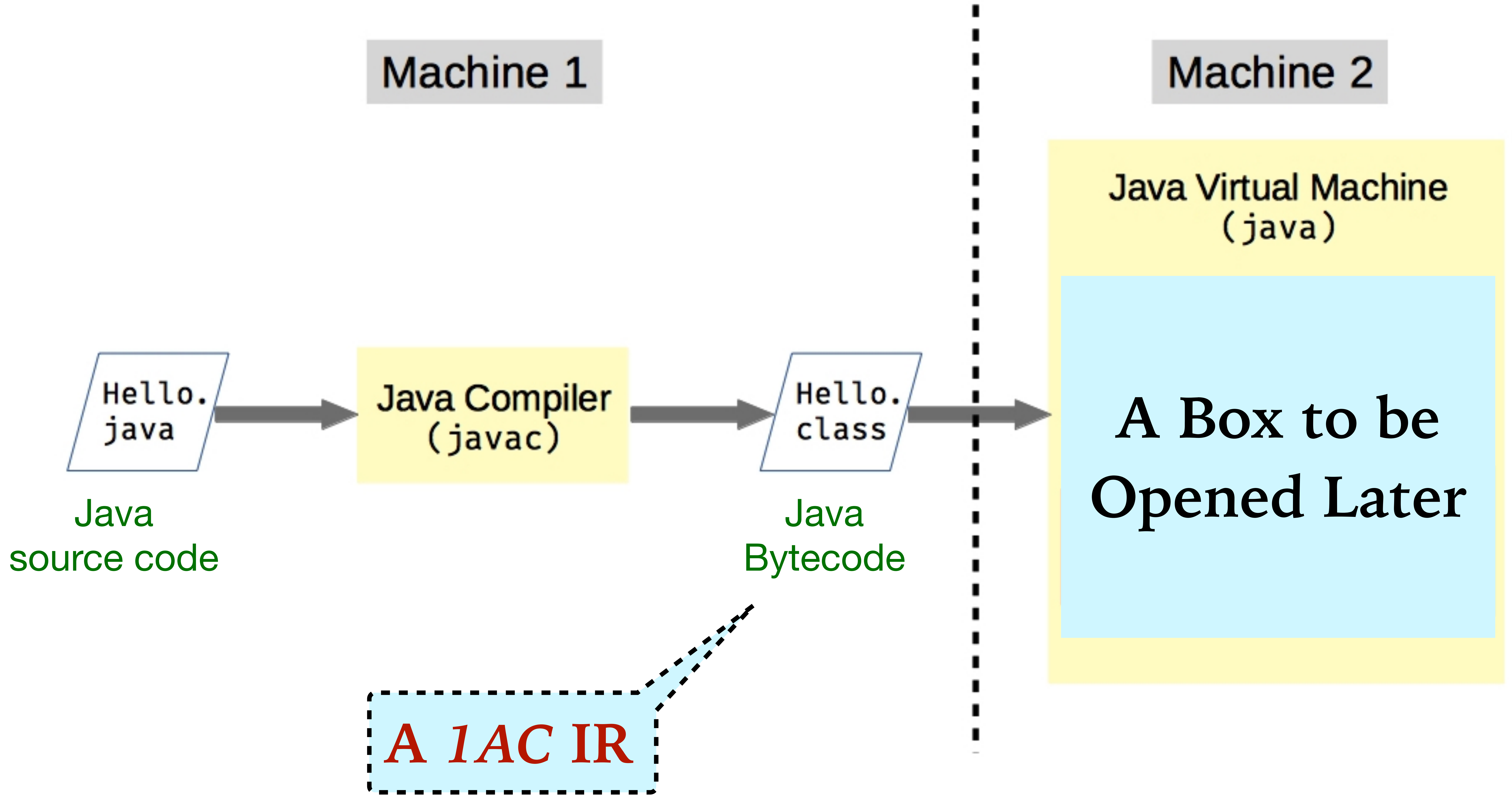
Spring 2025

Things we learnt in the last class

- Lowering OO languages
  - Field resolution (static)
  - Method resolution (using vtables)
  - Effects of inheritance
- How semantics are tied to compilation

Fields are bound statically, methods dynamically!

Java Program Translation



A Bit of Bytecode

- Generated by the static Java compiler
- Mid-level IR
- Machine independent
- Follows a stack model
- **Format:**
    - <opcode> <operands>
- Opcode is one Byte (8 bits)
    - 256 (2^8) in number

Bytecode examples

```
int a = 10;
int b = 20;
int c = a + b;
```

```
 0: bipush        10
 2: istore_1
 3: bipush        20
 5: istore_2
 6: iload_1
 7: iload_2
 8: iadd
 9: istore_3
10: return
```

Bytecode Indices (BCIs)

Bytecode examples

```
A obj = new A(10);
int objA = obj.getA();
System.out.println(objA);
```

```
 0: new           #2                  // class A
 3: dup
 4: bipush        10
 6: invokespecial #3                  // Method A."<init>":(I)V
 9: astore_1
10: aload_1
11: invokevirtual #4                  // Method A.getA:()I
14: istore_2
15: getstatic     #5                  // Field java/lang/System.out:Ljava/io/PrintStream;
18: iload_2
19: invokevirtual #6                  // Method java/io/PrintStream.println:(I)V
22: return
```

Constant-pool indices

Method invocations

Type descriptors

Type expressions in Bytecode

Java Bytecode	Type	Description
B	byte	signed byte
C	char	Unicode character
D	double	double-precision floating-point value
F	float	single-precision floating-point value
I	int	integer
J	long	long integer
L<classname>	reference	an instance of class <classname>
S	short	signed short
Z	boolean	true or false
[	reference	one array dimension

Kinds of invokes

- `invokevirtual:` Normal virtual calls
- `invokestatic:` Calls to static methods
- `invokespecial:` Constructor calls
- `invokeinterface:` Calls to interface methods
- `invokedynamic:` Find out what does it do

Which would be more expensive?

Why to have both?

Homework
Write programs to generate each kind of Bytecode invoke.

Class File Structure

```
ClassFile {
    u4             magic;
    u2             minor_version;
    u2             major_version;
    u2             constant_pool_count;
    cp_info        constant_pool[constant_pool_count-1];
    u2             access_flags;
    u2             this_class;
    u2             super_class;
    u2             interfaces_count;
    u2             interfaces[interfaces_count];
    u2             fields_count;
    field_info     fields[fields_count];
    u2             methods_count;
    method_info    methods[methods_count];
    u2             attributes_count;
    attribute_info attributes[attributes_count];
}
```

Find out their values for your installation!

Class File Disassembler (`javap`)

- Visualize Bytecode:
  - `javap -c Klass`
- Private methods:
  - `javap -p -c Klass`
- Verbose information (including constant pool):
  - `javap -v -c Klass`
- Understanding interesting constructs/features by demystifying Java class files:
  - Debugging (line-number tables)
  - Bytecode verification (stack maps) — details later

Java Object Layout Tool (`jol`)

- Visualize Object Structure:
  - `java -jar jol-cli-latest.jar internals Klass`
- With class files in current directory:
  - `java -jar jol-cli-latest.jar internals -cp . Klass`
- Compressed oops to address larger memories
- Understanding field ordering:
  - Header and Payload
  - Hiding fields in alignment gaps
  - Internal and external padding

Assignment 1 (10 marks; due February 1st)

- For each field reference and (virtual) method call, list the (set of) binding(s) visible at compile time.
  - Traverse class, field, method and variable declarations, and store information in a "symbol table"
  - Iterate over statements; print required information
  - Might require two visits over the AST
  - Grammar file and submission guidelines, along with a few public testcases, to be uploaded on **Moodle** by tomorrow
  - *Partially evaluate* your submission using **CompL Evaluator** before making submission

Next Class

The World of Optimizations!