Under the Hood: The Java Virtual Machine

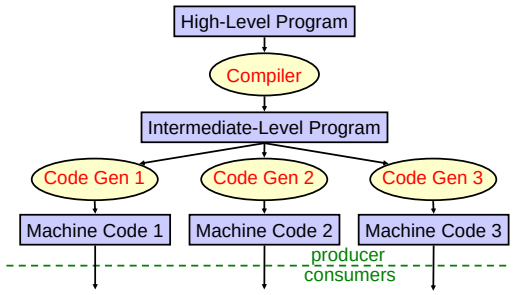
Compiling for Different Platforms

- Program written in some high-level language (C, Fortran, ML, ...)
- Compiled to intermediate form
- Optimized
- Code generated for various platforms (machine architecture + operating system)
- Consumers download code for their platform

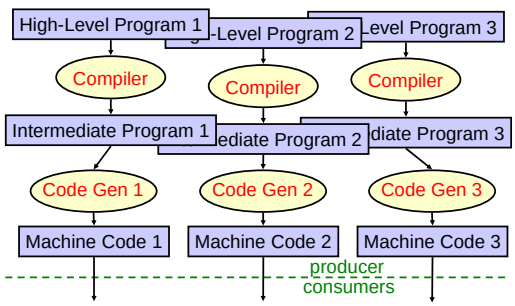
Problem: Too Many Platforms!

- Operating systems
  - DOS, Win 95, 98, NT, ME, 2K, XP, Longhorn, ...
  - Unix, Linux, FreeBSD, Aix, ...
  - VM/CMS, OS/2, Solaris, Mac OS X, ...
- Architectures
  - Pentium, PowerPC, Alpha, SPARC, MIPS, ...

Compiling for Different Platforms



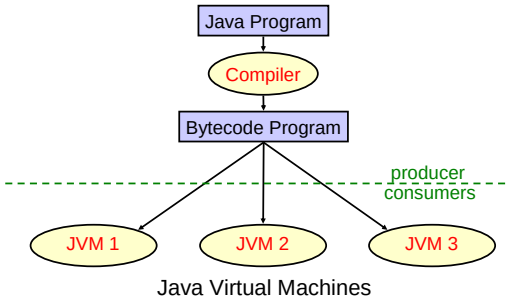
Compiling for Different Platforms



Dream: Platform Independence

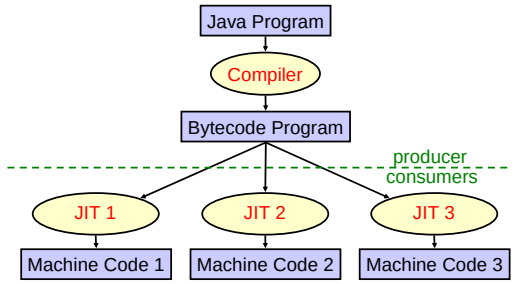
- Compiler produces *one* low-level program for all platforms
- Executed on a *virtual machine* (VM)
- A different VM implementation needed for each platform, but installed once and for all

Platform Independence with Java



Java Virtual Machines

Platform Independence with Java

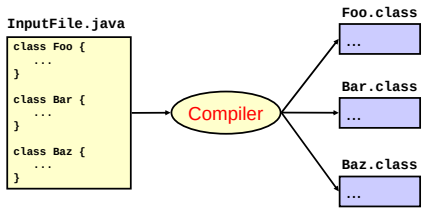


Just-in-Time Compilers

Java Bytecode

- Low-level compiled form of Java
- Platform-independent
- Compact
  - Suitable for mobile code, applets
- Easy to interpret
  - Java virtual machine (JVM) in your browser
  - Simple stack-based semantics
  - Support for objects

Class Files



Example

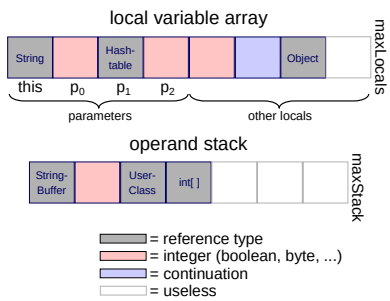
```
if (b) x = y + 1;
else x = z;

5:   iload_1    //load b
6:   ifeq 16    //if false, goto else
9:   iload_3    //load y
10:  iconst_1   //load 1
11:  iadd       //y+1
12:  istore_2   //save x
13:  goto 19    //skip else
16:  iload 4    //load z
18:  istore_2   //save x
19:  ...
```

then clause

else clause

Stack Frame of a Method



What is in a Class File?

Class File Format

magic number	4 bytes	`0xCAFEBABE`
major version	2 bytes	`0x0021`
minor version	2 bytes	`0x0000`

- magic number identifies the file as a Java class file
- version numbers inform the JVM whether it is able to execute the code in the file

Constant Pool

CP length	2 bytes
CP entry 1	(variable)
CP entry 2	(variable)
...	...

- constant pool consists of up to 65536 = 2^{16} entries
- entries can be of various types, thus of variable length

Constant Pool

1	Utf8 (unicode)	literal string (2 bytes length, characters)
3	Integer	Java int (4 bytes)
4	Float	Java float (4 bytes)
5	Long	Java long (8 bytes)
6	Double	Java double (8 bytes)
7	Class	class name
8	String	String constant -- index of a Utf8 entry
9	Fieldref	field reference -- name and type, class
10	Methodref	method reference -- name and type, class
11	InterfaceMethodref	interface method reference
12	NameAndType	Name and Type of a field or method

Constant Pool

- Many constant pool entries refer to other constant pool entries

  Fieldref
  - index to a Class
    - index to a Utf8 (name of class containing it)
  - index to a NameAndType
    - index to a Utf8 (name of field)
    - index to a Utf8 (type descriptor)

- Simple text (Utf8) names used to identify classes, fields, methods -- simplifies linking

Class File Format

access flags	2 bytes	"or" of several bits
this	2 bytes	index into CP
super	2 bytes	index into CP

- access flags tell whether class is public, private, protected, ...
- this, super are index into constant pool giving name of this class and parent class

Some Access Flags

public	0x0001
private	0x0002
protected	0x0004
static	0x0008
final	0x0010
native	0x0100
interface	0x0200
abstract	0x0400

Interfaces

count	2 bytes	length of table
InterfaceRef 1	2 bytes	index into CP
InterfaceRef 2	2 bytes	index into CP
...	...	...

- table of constant pool indices
- specifies interfaces this class implements

Fields

count	2 bytes	length of table
Field Table 1	variable	index into CP
Field Table 2	variable	index into CP
...	...	...

- table of field table entries, one for each field defined in the class

Field Table

access flags	2 bytes	e.g. public, static
name index	2 bytes	index of a Utf8
descriptor index	2 bytes	index of a Utf8
attributes count	2 bytes	number of attributes
attribute 1	variable	e.g. constant value
attribute 2	variable	...
...	...	...

Methods

count	2 bytes	length of table
Method Table 1	variable	index into CP
Method Table 2	variable	index into CP
...	...	...

- table of method table entries, one for each method defined in the class

Method Table

access flags	2 bytes	e.g. public, static
name index	2 bytes	index of a Utf8
descriptor index	2 bytes	index of a Utf8
attributes count	2 bytes	number of attributes
attribute 1	variable	e.g. constant value
attribute 2	variable	...
...	...	...

Descriptors

Primitive types

byte	B
char	C
double	D
float	F
int	I
long	J
short	S
boolean	Z
void	V

Class or Interface types

L followed by pathname followed by ;
e.g. `Ljava/lang/String;`

Array types

Number of **[** equal to dimension followed by component type, e.g.

```
[[I
[Ljava/lang/String;
```

Method Descriptors

Argument types in parens ( ) concatenated together, followed by return type; e.g.

```
public static void main(String[] args)
```

would have type

```
([Ljava/lang/String;)V
```

Attributes

Code	Bytecode of a method
ConstantValue	Used by final fields
Exceptions	Exceptions thrown by a method
InnerClasses	Inner classes of a class
LineNumberTable	Used for debugging
LocalVariableTable	Used for debugging
SourceFile	Source file name

Code Attribute

maxStack	2 bytes	max operand stack depth
maxLocals	2 bytes	number of local variables
codeLength	2 bytes	length of bytecode array
code	codeLength	the executable bytecode
excTableLength	2 bytes	number of exception handlers
exceptionTable	excTableLength	exception handler info
attributesCount	2 bytes	number of attributes
attributes	variable	e.g. LineNumberTable

Exception Table Entry

startPC	2 bytes	start of range handler is in effect
endPC	2 bytes	end of range handler is in effect
handlerPC	2 bytes	entry point of exception handler
catchType	2 bytes	type of exception handled

- An exception handler is just a designated block of code
- When an exception is thrown, table is searched in order for a handler that can handle the exception

Examples

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

Q) How many entries in the constant pool?

Examples

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

Q) How many entries in the constant pool?

A) 33

```
1)CONSTANT_Methodref[10](class_index = 6, name_and_type_index = 20)
2)CONSTANT_Fieldref[9](class_index = 21, name_and_type_index = 22)
3)CONSTANT_String[8](string_index = 23)
4)CONSTANT_Methodref[10](class_index = 24, name_and_type_index = 25)
5)CONSTANT_Class[7](name_index = 26)
6)CONSTANT_Class[7](name_index = 27)
7)CONSTANT_Utf8[1]("<init>")
8)CONSTANT_Utf8[1]("()V")
9)CONSTANT_Utf8[1]("Code")
10)CONSTANT_Utf8[1]("LineNumberTable")
11)CONSTANT_Utf8[1]("LocalVariableTable")
12)CONSTANT_Utf8[1]("this")
13)CONSTANT_Utf8[1]("LFoo;")
14)CONSTANT_Utf8[1]("main")
15)CONSTANT_Utf8[1]("([Ljava/lang/String;)V")
16)CONSTANT_Utf8[1]("args")
17)CONSTANT_Utf8[1]("[Ljava/lang/String;")
18)CONSTANT_Utf8[1]("SourceFile")
19)CONSTANT_Utf8[1]("Foo.java")
20)CONSTANT_NameAndType[12](name_index = 7, signature_index = 8)
21)CONSTANT_Class[7](name_index = 28)
22)CONSTANT_NameAndType[12](name_index = 29, signature_index = 30)
23)CONSTANT_Utf8[1]("Hello world")
24)CONSTANT_Class[7](name_index = 31)
25)CONSTANT_NameAndType[12](name_index = 32, signature_index = 33)
26)CONSTANT_Utf8[1]("Foo")
27)CONSTANT_Utf8[1]("java/lang/Object")
28)CONSTANT_Utf8[1]("java/lang/System")
29)CONSTANT_Utf8[1]("out")
30)CONSTANT_Utf8[1]("Ljava/io/PrintStream;")
31)CONSTANT_Utf8[1]("java/io/PrintStream")
32)CONSTANT_Utf8[1]("println")
33)CONSTANT_Utf8[1]("(Ljava/lang/String;)V")
```

Examples

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

Q) How many methods?

Examples

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

Q) How many methods?

A) 2

```
public static void main (String[] args)
      Code: maxStack=2 maxLocals=1 length=9
      exceptions=0
      attributes=2
        source lines=2
        local variables=1
          java/lang/String[] args startPC=0 length=9 index=0
------------------------------------------------------------
0:   getstatic java/lang/System.out
3:   ldc "Hello world"
5:   invokevirtual java/io/PrintStream.println(Ljava/lang/String;)V
8:   return

============================================================
void <init> ()
      Code: maxStack=1 maxLocals=1 length=5
      exceptions=0
      attributes=2
        source lines=1
        local variables=1
          Foo this startPC=0 length=5 index=0
------------------------------------------------------------
0:   aload_0
1:   invokespecial java/lang/Object.<init>()V
4:   return
```

Next Time

- Class loading and initialization
- Object initialization
- Method dispatch
- Exception handling
- Bytecode verification