University of Arizona

CSc 620

Security Through Obscurity

Christian Collberg

January 23, 2002

Jasmin

Slide 2–1

```
.class public HelloWorld
.super java/lang/Object
.method public <init>()V
  aload_0
  invokenonvirtual java/lang/Object/<init>()V
  return
.end method
.method public static main([Ljava/lang/String;)V
  .limit stack 2   ; up to two items can be pushed
  ; push System.out onto the stack
  getstatic java/lang/System/out Ljava/io/PrintStream;
  ; push a string onto the stack
  ldc "Hello World!"
  ; call the PrintStream.println() method.
  invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
  return
.end method
```

Slide 2–2

Running Jasmin

The jasmin command runs Jasmin on a file. For example:

```
> jasmin myfile.j
```

assembles the file `"myfile.j"`. Jasmin looks at the .class directive contained in the file to decide where to place the output class file. So if myfile.j starts with:

```
> jasmin myfile.j
 .class mypackage/MyClass
```

then Jasmin will place the output class file `"MyClass.java"` in the subdirectory `"mypackage"` of the current directory. It will create the mypackage directory if it doesn't exist.

Slide 2–3

Directives

Directive statements are used to give Jasmin meta-level information. Directive statements consist of a directive name, and then zero or more parameters separated by spaces, then a newline. All directive names start with a "." character. The directives in Jasmin are:

```
.catch .class .end .field .implements .interface
.limit .line .method .source .super .throws .var
```

Some example directive statements are:

```
.limit stack 10
.method public myMethod()V
.class Foo\begin{itemize}
```

Slide 2–4

Instructions

An instruction statement consists of an instruction name, zero or more parameters separated by spaces, and a newline.

Jasmin uses the standard mnemonics for JVM opcodes as instruction names. For example, aload_1, bipush and iinc are all Jasmin instruction names.

Here are some examples of instruction statements:

```
ldc    "Hello World"
iinc   1 -1
bipush 10
```

Slide 2–5

Names and Types

- Class names in Jasmin should be written using the Java class file format conventions, so `java.lang.String` becomes `java/lang/String`.
- Type information is also written as they appear in class files (e.g. the descriptor I speficies an integer, `[Ljava/lang/Thread;` is an array of Threads, etc.).

Slide 2–6

Methods

- Method names are specified using a single token, e.g.

  ```
  java/io/PrintStream/println(Ljava/lang/String;)V
  ```

  is the method called `"println"` in the class `java.io.PrintStream`, which has the type descriptor `"(Ljava/lang/String;)V"`.
- In general, a method specification is formed of three parts: the characters before the last '/' form the class name. The characters between the last '/' and '(' are the method name. The rest of the string is the type descriptor for the method.

foo/baz/Myclass/	myMethod	(Ljava/lang/String;)V

Slide 2–7

Methods

- As another example, you would call the Java method:

  ```
  class mypackage.MyClass {
      int foo(Object a, int b[]) { ... }
  }
  ```

  using:

```
invokevirtual mypackage/MyClass/foo(Ljava/lang/Object;[I)I
```

Slide 2–8

Fields

- Field names are specified in Jasmin using two tokens, one giving the name and class of the field, the other giving its descriptor. For example:

```
getstatic mypackage/MyClass/my_font  Ljava/lang/Font;
```

gets the value of the field called `"my_font"` in the class `mypackage.MyClass`. The type of the field is `"Ljava/lang/Font;"` (i.e. a Font object).

Slide 2–9

File Structure

- Jasmin files start by giving information on the class being defined in the file - such as the name of the class, the name of the source file that the class originated from, the name of the superclass, etc.
- Typically, a Jasmin file starts with the three directives:

```
.source <source-file>
.class  <access-spec> <class-name>
.super  <class-name>
```

Slide 2–10

File Structure

- For example, the file defining MyClass might start with the directives:

```
.source MyClass.j
.class  public MyClass
.super  java/lang/Object
```

- `access-spec` is a list of zero or more of the following keywords:

```
public, final, super, interface, abstract
```

Slide 2–11

Field Definitions

- After the header information, the next section of the Jasmin file is a list of field definitions.
- A field is defined using the .field directive:

```
.field <access-spec> <field-name> <descriptor> [=<value>]
```

- Examples:

```
public int foo;
public static final float PI = 3.14;
```

compiles to

```
.field public foo
.field public static final PI F = 3.14
```

i

Slide 2–12

Method Definitions

- After listing the fields of the class, the rest of the Jasmin file lists methods defined by the class.
- A method is defined using the basic form:

```
.method <access-spec> <method-spec>
    <statements>
.end method
```

- Always add an explicit return at the end of the method.

```
.method foo()V
     return     ; must give a return statement
 .end method
```

Slide 2–13

Method Directives

- `.limit stack <integer>` Sets the maximum size of the operand stack required by the method.
- `.limit locals <integer>` Sets the number of local variables required by the method.
- `.var <var-number> is <name> <descriptor>` from ¡label1¿ to ¡label2¿ The `.var` directive is used to define the name, type descriptor and scope of a local variable number.

Slide 2–14

Method Directives – Example

```
.method foo()V
   .limit locals 1
    ; declare variable 0 as an "int Count;"
    ; whose scope is the code between Label1 and Label2

    .var 0 is Count I from Label1 to Label2
Label1:
    bipush 10
    istore_0
Label2:
    return
.end method
```

Slide 2–15

Exceptions

- `.throws <classname>` Indicates that this method can throw exceptions of the type indicated by `<classname>`. e.g.

```
.throws java/io/IOException
```

- `.catch <classname>`
  from ¡label1¿ to ¡label2¿ using ¡label3¿
  Appends an entry to the end of the exceptions table for the method. The entry indicates that when an exception which is an instance of `<classname>` or one of its subclasses is thrown while executing the code between `<label1>` and `<label2>`, then the runtime system should jump to `<label3>`. e.g.

```
.catch java/io/IOException from L1 to L2 using IO_Handler
```

Slide 2–16

JVM Instructions

- JVM instructions are placed between the `.method` and `.end method` directives.
- VM instructions can take zero or more parameters. Examples:

```
iinc 1 -3     ; decrement local variable 1 by 3
bipush 10     ; push the integer 10 onto the stack
pop           ; remove the top item from the stack.
```

Slide 2–17

Local variable instructions

```
ret <var-num>
aload <var-num>
astore <var-num>
dload <var-num>
dstore <var-num>
fload <var-num>
fstore <var-num>
iload <var-num>
istore <var-num>
lload <var-num>
lstore <var-num>
```

for example:

```
aload 1 ; push local variable 1 onto the stack
ret 2   ; return to the address held in local variable 2
```

Slide 2–18

The bipush, sipush and iinc instructions

```
bipush <int>
sipush <int>
```

for example:

```
bipush 100    ; push 100 onto the stack
```

The iinc instruction takes two integer parameters:

```
iinc  <var-num> <amount>
```

for example:

```
iinc 3 -10    ; subtract 10 from local variable 3
```

Slide 2–19

Branch instructions

```
goto  <label>
goto_w  <label>
if_acmpeq  <label>
if_acmpne  <label>
if_icmpeq  <label>
if_icmpge  <label>
if_icmpgt  <label>
if_icmple  <label>
if_icmplt  <label>
if_icmpne  <label>
ifeq  <label>
ifge  <label>
ifgt  <label>
ifle  <label>
iflt  <label>
ifne  <label>
```

Branch instructions. . .

```
ifnonnull  <label>
ifnull  <label>
jsr  <label>
jsr_w  <label>
```

For example:

```
Label1:
    goto Label1    ; jump to the code at Label1
                   ; (an infinite loop!)
```


Class and object operations

```
anewarray  <class>
checkcast  <class>
instanceof <class>
new        <class>
```

For example:

```
new java/lang/String   ; create a new String object
```


Method invokation

```
invokenonvirtual  <method-spec>
invokestatic      <method-spec>
invokevirtual     <method-spec>
```

For example:

```
; invokes java.io.PrintStream.println(String);
invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
```


Field manipulation instructions

```
getfield  <field-spec> <descriptor>
getstatic  <field-spec> <descriptor>
putfield  <field-spec> <descriptor>
putstatic  <field-spec> <descriptor>
```

For example:

```
; get java.lang.System.out, which is a PrintStream
getstatic java/lang/System/out Ljava/io/PrintStream;
```

Array instructions

```
newarray  <array-type>
multianewarray  <array-descriptor> <num-dimensions>
```

For example:

```
newarray int
newarray short
newarray float
multianewarray [[[I 2
```

Slide 2–24

The ldc and ldc_w instructions

```
ldc  <constant>
ldc_w  <constant>
```

<constant> is either an integer, a floating point number, or a quoted string. For example:

```
ldc 1.2             ; push a float
ldc 10              ; push an int
ldc "Hello World"   ; push a String
ldc_w 3.141592654   ; push PI as a double
```

Slide 2–25

Switch instructions

```
<lookupswitch> ::=
    lookupswitch
       <int1> : <label1>
       <int2> : <label2>
        ...
        default : <default-label>

<tableswitch> ::=
    tableswitch <low>
        <label1>
        <label2>
         ...
         default : <default-label>
```

Slide 2–26

lookupswitch example

```
; If the int on the stack is 3,
; jump to L1.
; If it is 5, jump to Label2.
; Otherwise jump to DefaultLabel.
lookupswitch
        3 : L1
        5 : Ll2
  default : Default
L1:
   ... got 3
L2:
   ... got 5
Default:
   ... got something else
```

Slide 2–27

tableswitch example

```
; If the int on the stack is 0,
; jump to Label1.
; If it is 1, jump to Label2.
; Otherwise jump to DefaultLabel.
tableswitch 0
    Label1
    Label2
  default : DefaultLabel
Label1:
  ... got 0
Label2:
  ... got 1
DefaultLabel:
  ... got something else
```

Slide 2–28

Example I - Count.j

```
.class public Count
.super java/lang/Object

.method public <init>()V
   aload_0
   invokenonvirtual java/lang/Object/<init>()V
   return
.end method

.method public static main([Ljava/lang/String;)V
   ; set limits used by this method
   .limit locals 4
   .limit stack 3
```

Slide 2–29

```
   ;    1 - the PrintStream object held in java.lang.System.out
   getstatic java/lang/System/out Ljava/io/PrintStream;
   astore_1

   ;    2 - the integer 10 - the counter used in the loop
   bipush 10
   istore_2

   ; now loop 10 times printing out a number
```

Slide 2–30

```
Loop:
  bipush 10     ; compute 10 - <local variable 2> ...
  iload_2
  isub
  invokestatic java/lang/String/valueOf(I)Ljava/lang/String;
  astore_3
  ; ... and print it
  aload_1    ; push the PrintStream object
  aload_3    ; push the string we just created - then ...
  invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V

  iinc 2 -1    ; decrement the counter and loop
  iload_2
  ifne Loop

  return
.end method
```

Slide 2–31

Example II - Arrays.j

```
.class public Arrays
.super java/lang/Object

.method public <init>()V
   ...
.end method

.method public static main([Ljava/lang/String;)V
  .limit locals 2
  .limit stack 4

  ;  String[] myarray = new String[2];
  iconst_2
  anewarray java/lang/String
  astore_1  ; stores this in local variable 1
```



```
  ;      myarray[0] = args[0];
  aload_1     ; push my array on the stack
  iconst_0
  aload_0     ; push the array argument to main() on the stack
  iconst_0
  aaload      ; get its zero'th entry
  aastore     ; and store it in my zero'th entry

  ; now print out myarray[0]
  getstatic java/lang/System/out Ljava/io/PrintStream;
  aload_1
  iconst_0
  aaload
  invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V

  return
.end method
```


Example III - Newarray.j

```
.class public NewArray
.method public static main([Ljava/lang/String;)V
   .limit stack 4
   .limit locals 2

   iconst_2
   newarray boolean        ; boolean b[] = new boolean[2]
   astore_1                ; stores it in local var 1

   aload_1                 ; b[0] = true;
   iconst_0
   iconst_1
   bastore

   return
.end method
```


Example IV - Switch.j

```
.method public static main([Ljava/lang/String;)V
   .limit stack 3
   iconst_1
   lookupswitch
        1 : Hello
        2 : Goodbye
     default : Foo
Hello:
Goodbye:
Foo:
   return
.end method
```

Slide 2–36

Example IV - Switch.j

```
.method public static main([Ljava/lang/String;)V
   .limit stack 3
   iconst_1
   tableswitch 0
        Hello
        Goodbye
     default : Foo
Hello:
Goodbye:
Foo:
   return
.end method
```

Slide 2–37

Example V - CheckCast.j

```
.class examples/Checkcast
.super java/lang/Object

....

.method public static main([Ljava/lang/String;)V
   .limit stack 2

   ; push System.out onto the stack
   getstatic java/lang/System/out Ljava/io/PrintStream;

   ; check that it is a PrintStream
   checkcast java/io/PrintStream

   return
.end method
```

Slide 2–38

Example VI - Abs.j

```
;   public static int abs(int x);
;     if (x < 0) x = -x
;     return x
.method public static abs(I)I
       .limit locals 1
       .limit stack  2

       iload_0    ; x
       dup        ; x x
       ifge  done ; x    is x < 0?
       ineg       ; -x   yes. x = -x
done:
       ireturn
.end method
```

Slide 2–39

Example VII - Dist.j

```
;    public static int dist(int x, int y)
;       returns abs(x - y)
.method public static dist(II)I
       .limit locals 2
       .limit stack 2

       iload_0   ; x
       iload_1   ; y x
       isub      ; y - x

       invokestatic java/lang/Math/abs(I)I
       ireturn
.end method
```

Readings and References

- The information in these slides has been shamelessly stolen from Jonathan Meyer's Jasmin pages, `http://mrl.nyu.edu/~meyer/jasmin/`. Additional examples are from `http://www.csam.montclair.edu/~bredlau/jasmin/JVM.html`.
- `http://bcel.sourceforge.net/JasminVisitor.java` is a program that uses the BCEL bytecode editor to generate Jasmin assembly code from a Java class file.