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.

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}

    .class public HelloWorld
    .super java/lang/Object
    .method public <init>()V
      aload_0
      invokevirtual 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


Jasmin instructions are specified using a single token, e.g.,

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

In general, a method specification is formed of three parts:

- `foo/haz/Myclass/mymethod(Java/lang/String);` V

The rest of the string is the type descriptor for the method.

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

`class mypackage.MyClass {`  

As another example, you would call the Java method:

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

`invokevirtual mypackage/MyClass/foo(java/lang/Object;[I)`
File Structure

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

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

- Examples:
  
  ```java
  public int foo;
  public static final float PI = 3.14;
  ```

  compiles to

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

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:

  ```java
  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).

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:

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

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`

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`

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.
JVM Instructions

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

Examples:

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

The bipush, sipush and iinc instructions

```
bipush 100  ; push 100 onto the stack
sipush <int>  ; sipush <int>
```

The iinc instruction takes two integer parameters:

```
iinc 1 -3  ; decrement local variable 1 by 3
iinc 3 -10  ; subtract 10 from local variable 3
```

Local variable instructions

```
ret <var-num>
aload <var-num>
astore <var-num>
ldc <var-num>
ldc2_w <var-num>
ldc2_a <var-num>
```

Branch instructions

```
goto <label>
goto_w <label>
if_acmpeq <label>
if_acmpne <label>
if_icmpeq <label>
if_icmpne <label>
if_icmpgt <label>
if_icmplt <label>
if_icmpge <label>
if_icmple <label>
ifeq <label>
ifge <label>
ifgt <label>
ifle <label>
iflt <label>
ifne <label>
ret 2  ; return to the address held in local variable 2
```
Method invocation

```
invokevirtual <method-spec>
invokestatic <method-spec>
invokevirtual java/io/PrintStream/println(Ljava/lang/String;)
```

For example:
```
; invokes java.io.PrintStream.println(String);
invokevirtual java/io/PrintStream/println(Ljava/lang/String;)
```

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!)
```

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;
```

Class and object operations

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

For example:
```
new java/lang/String ; create a new String object
```
Array instructions

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

For example:
newarray int
ewarray short
ewarray float
multianarray [[[[[I 2

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

Switch instructions

lookupswitch example

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

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

; 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 : L2
default : Default
L1:
... got 3
L2:
... got 5
Default:
... got something else
: 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

---

Example I - Count.j

.class public Count
.super java/lang/Object

.method public <init>()V
  aload_0
  invokespecial 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
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
    astore
    return
.end method
```

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
    ...}
```

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
```

```
    ; 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
    ...}
```
Example VI - Abs.j

```java
public static int abs(int x);
    if (x < 0) x = -x
    return x
.method public static abs(D)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
```

Example IV - Switch.j

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

Example VII - Dist.j

```java
public static int dist(int x, int y)
    returns abs(x - y)
.method public static dist(D)I
    .limit locals 2
    .limit stack 2
    iload_0 ; x
    iload_1 ; y x
    isub ; y - x
    invokevirtual java/lang/Math/abs(I)I
    ireturn
.end method
```

Example V - CheckCast.j

```java
.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/putLjava/io/PrintStream;
    ; check that it is a PrintStream
    checkcast java/io/PrintStream
    return
.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.cs.montclair.edu/~bredlav/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.