Compiler Construction

Code Generation - Tutorial

Christopher Liebmann, Sebastian Puck
Practical Instructor: Alexander Perko

cc@ist.tugraz.at

Institute of Software Technology
Graz University of Technology
Austria

Summer Term 2024

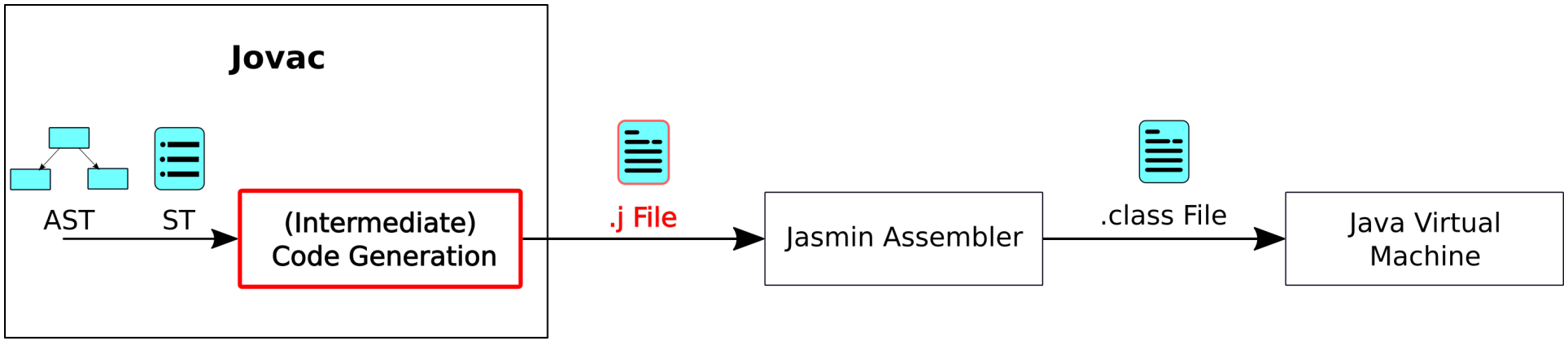
Version: May 8, 2024

Outline

- Task 3 - Overview
- Jasmin / JVM “Crash Course”
- Implementation Hints

Task 3 - Overview

Code Generation Pipeline



Output Requirements

- For **each class** in a .jova file ⇒ a **single .j (Jasmin) file**
- Naming: <`className`>`.j`, where <`className`> = name of class
- E.g.: `Main.j`, `MyClass.j`, ...
- Create directory (structure) specified by parameter `out_path`
  ⇒ all .j files of the program should be placed there

Output Requirements - Example

- Method `task3(...)` called with:
  `file_path` = `input/codegen/mytests/simple.jova`
  `out_path` = `out/mytests/simple`

```
1 Simple {
2   ...
3 }
4 Main {
5   ...
6 }
```

- Create path: `out/mytests/simple/`
- Generate .j files in this directory:
  ⇒ `out/mytests/simple/Simple.j`
  ⇒ `out/mytests/simple/Main.j`

Jasmin / JVM

What is Jasmin?

- Assembler for JVM
- Input: `<className>.j` file(s)
  - Contains description of Jova class
  - Simple assembler-like syntax using JVM instruction set
- Generates binary Java class files
- Output: `<className>.class` file(s)
- Performs rudimentary/ASM-specific error checks

Execute Compiled Jova Program

- Provided: Executable `libs/jasmin.jar`
- Run Jasmin on generated .j file(s):
  `java -jar jasmin.jar <className>.j`
  ⇒ E.g.: `java -jar jasmin.jar Main.j`
- Execute generated `.class` file(s) on JVM:
  `java -cp <classPath> <className>`
  ⇒ E.g.: `java -cp . Main`
- Note: Specified class must have entry point (i.e. `main` method)

Jasmin syntax

- One class per .j file
- One statement per line
- Class described with options, directives, fields and methods of the class
- Methods functionality described with JVM instructions

Example: DoNix.j

```
1  .source noSource
2  .class public DoNix
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 0
7  .limit locals 1
8    ;nothing to do here
9    return
10 .end method
```

File structure: Class options

- **`.source`**: Source of assembly (optional)
  ⇒ E.g.: `.source Simple.jova`
- **`.class`**: Resulting java class description
  ⇒ E.g.: `.class public Simple`
- **`.super`**: Superclass of resulting java class
  ⇒ E.g.: `.super java/lang/Object`
  ⇒ For inheritance, e.g.: `.super MySuperClass`
- **`.field`**: Specifies field of class
  ⇒ E.g.: `.field public my_field I`

File structure: Methods

- `.method <method signature>`
  ⇒ E.g.: `.method public myMethod(I)V`
- `.limit stack n`: Limits method stack to size n
  ⇒ n = the minimal needed size of the stack to perform all operations of the method
- `.limit locals n`: Limits method local array to size n
  ⇒ n = *#parameters* + *#local_vars* + *#temp_var* + *this*
- Body comprises of JVM instructions
- Last instruction: `return`
  ⇒ requires matching type on top of stack for non-void returns (e.g. `ireturn`)
- `.end method`: marks end of method description

Default Constructor

- Needs to be defined for every (non-static) class
- You can use the following definition as a default constructor for a class:

```
1 .method public <init>()V
2   aload_0
3   invokespecial java/lang/Object/<init>()V
4   return
5 .end method
```

Default Constructor - Inheritance

- For a subclass which inherits from a superclass `B`:

```
1 .method public <init>()V
2   aload_0
3   invokespecial B/<init>()V
4   return
5 .end method
```

Method - Data Management

- Two data structures per method (JVM stack frame):
  ⇒ **Operand Stack**
  ⇒ **Local Variable Array**
- JVM instructions manipulate both
- Built-in data types:
  - **Primitive types** (byte, short, **int**, long, char, float, double)
  - **Reference types** (**class types**, array types, interface types)
  - Limited support for boolean

Local Variable Array

- Each method has a local variable array (0-based indexing)
- Used for local/temp. variables
- Can store arbitrary types
  $\Rightarrow$ Items need to be initialized before read access
- Contains method parameters (stored in lowest indices)
- `this` at index 0 (non-static methods)
- Set size with: `.limit locals`

Operand Stack

- Each method has own operand stack (LIFO)
- Stores values for operations
- Can hold values of arbitrary type
- Stack operations:
  - Push values onto stack
  - Pop/Fetch values from stack
  - Instructions which require one or multiple values on stack (order does matter!)

Data Managment: Types

- Primitive types:
  - Integer - indicated by letter `I`
  - Void - indicated by letter `V`
  - Boolean - indicated by letter `Z`
- Class types follow the format: `Lpackage/Classname;`
  $\Rightarrow$ E.g.: String class: `Ljava/lang/String;`
- Array indicated by a leading `[`
  $\Rightarrow$ E.g.: String array: `[Ljava/lang/String;`

Instructions

- Can involve stack and local array
- Most instructions operate on expected types
- Instruction template: <T><op>
  ⇒ <T> = type letter
  ⇒ <op> = operation
- E.g.: `iadd`, `fadd`
- Instructions which do not operate on specific type:
  ⇒ E.g.: `swap`, `dup`

Instructions: Stack/Locals Interaction

- `iload n` - pushes int from local array index n onto stack
- `istore n` - pops int from stack and stores it at local array index n
- `aload n` - pushes object-ref from local array index n onto stack
- `astore n` - pops objct-ref from stack and stores it at local array index n

Instructions: Constants

- `bipush n` - pushes byte sized interger onto stack
  ⇒ E.g.: `bipush 10`
- `sipush n` - pushes short onto stack
  ⇒ E.g.: `sipush 200`
- `iconst_<n>` - pushes int const n onto stack, for n in range -1 - 5
- `ldc` - can be used to load int or ref. to string constant onto stack
  ⇒ E.g.: `ldc "Hello World"`

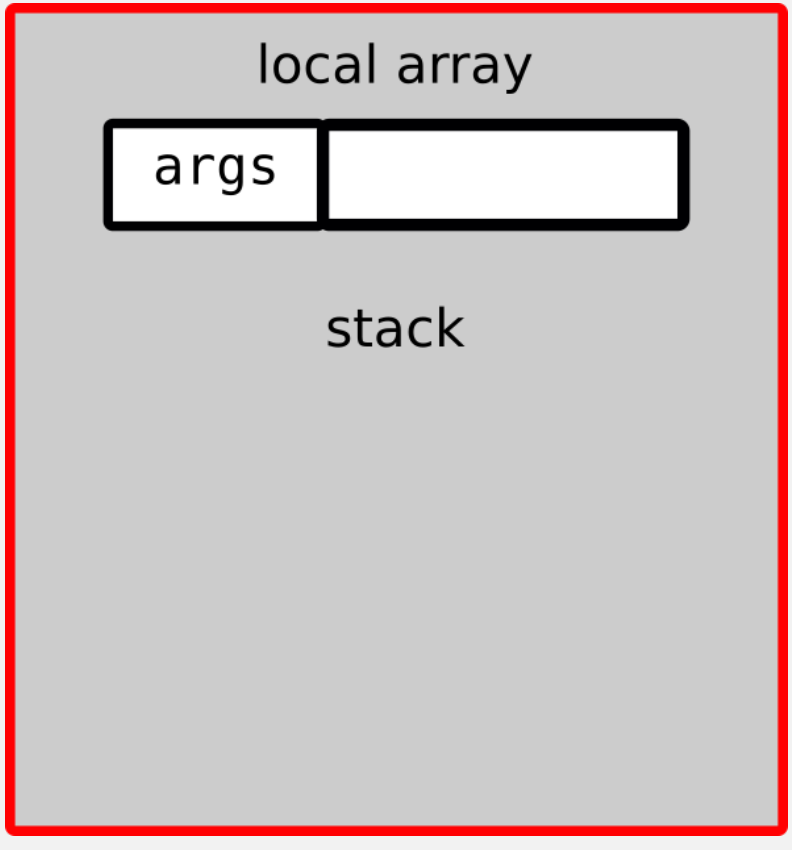
Instructions: Arithmetic Operators

- `iadd` - add two integers
- `isub` - subtract two integers
- `imul` - multiply two integers
- `idiv` - divide two integers
- `irem` - modulo division of two integers
- `ineg` - toggles sign of int on top of stack

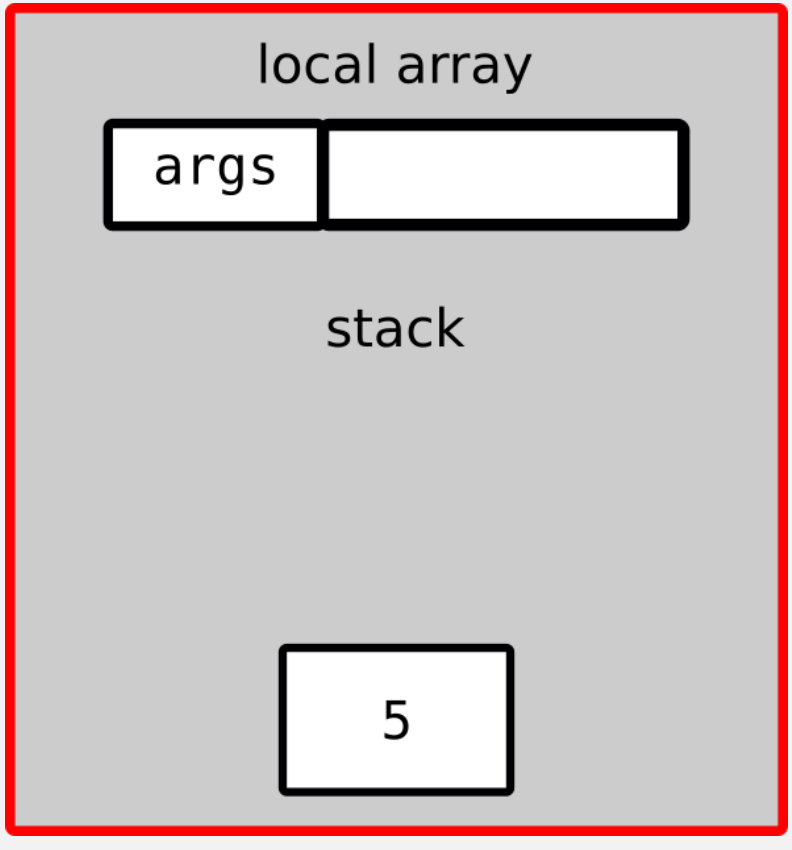
Code Example - BasicInstructions.j

```
1  .source noSource
2  .class public BasicInstructions
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 3
7  .limit locals 2
8    bipush 5           ;push integer 5 onto stack
9    istore 0           ;pop integer 5 and store in index 0
10                      ;(overwriting args!)
11   ldc "Hello i11"    ;push string Hello i11 onto stack
12   astore 1           ;store string in index 1
13
14   iload 0            ;load 5
15   dup                ;duplicate top of stack
16   bipush 2           ;push integer 2 onto stack
17   isub               ;pop 5 and 2 and store result 3 onto stack
18   iadd               ;pop 5 and 3 and store result 8 onto stack
19   istore 0           ;store result 8 in index 0
20
21   return
22 .end method
```

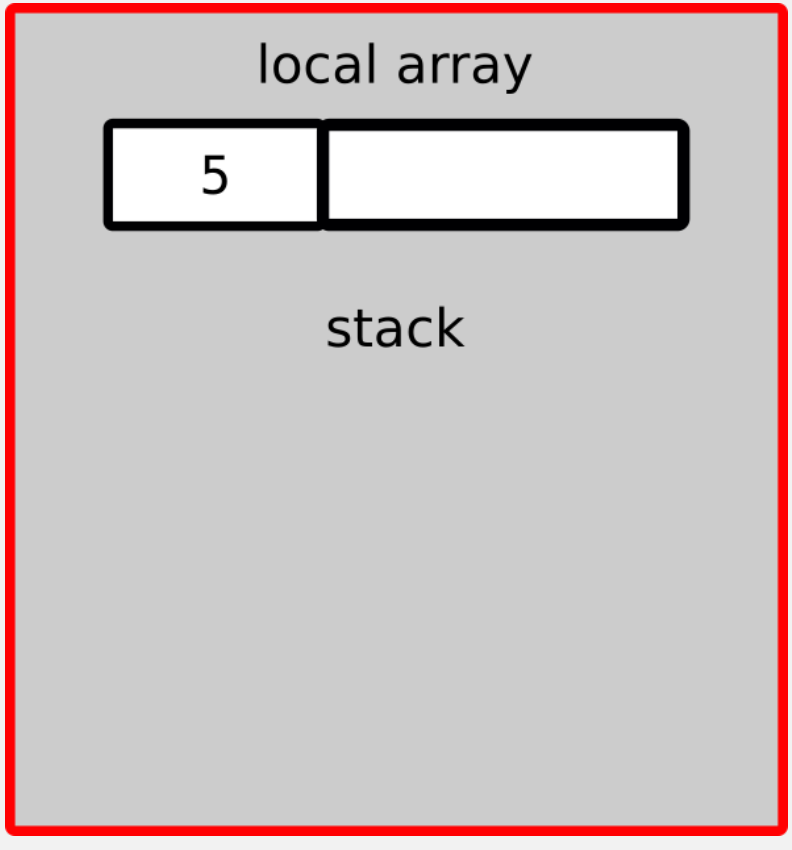
```
1  .source noSource
2  .class public BasicInstructions
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 3
7  .limit locals 2
8    bipush 5
9    istore 0
10
11   ldc "Hello i11"
12   astore 1
13
14   iload 0
15   dup
16   bipush 2
17   isub
18   iadd
19   istore 0
20
21   return
22 .end method
23
```



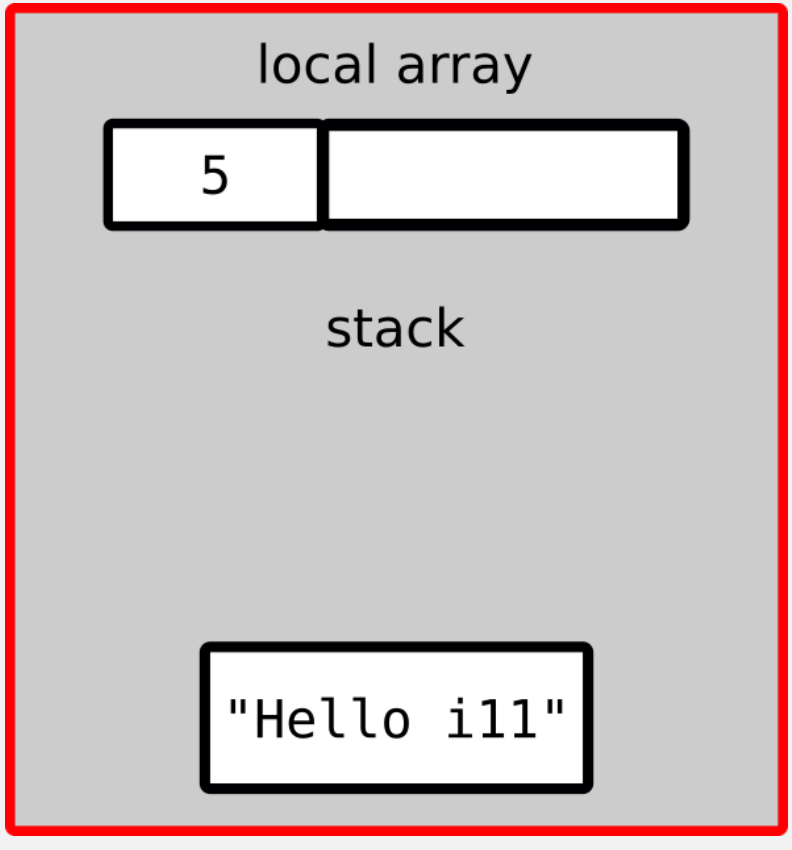
```
1  .source noSource
2  .class public BasicInstructions
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 3
7  .limit locals 2
8    bipush 5
9    istore 0
10
11   ldc "Hello i11"
12   astore 1
13
14   iload 0
15   dup
16   bipush 2
17   isub
18   iadd
19   istore 0
20
21   return
22 .end method
```



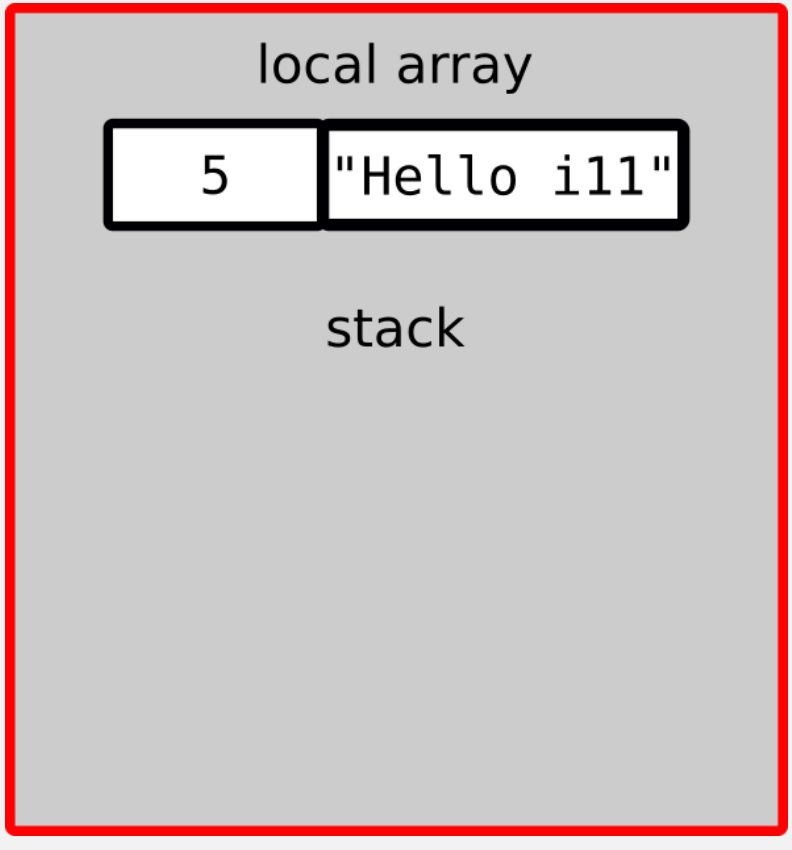
```
1  .source noSource
2  .class public BasicInstructions
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 3
7  .limit locals 2
8    bipush 5
9    istore 0
10
11   ldc "Hello i11"
12   astore 1
13
14   iload 0
15   dup
16   bipush 2
17   isub
18   iadd
19   istore 0
20
21   return
22 .end method
```



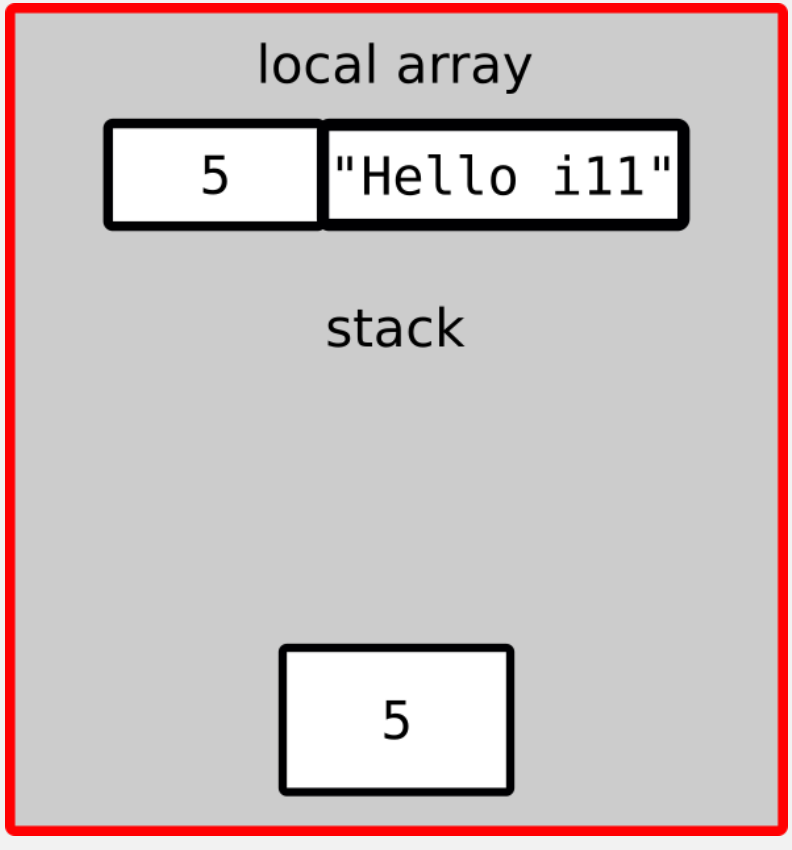
```
1  .source noSource
2  .class public BasicInstructions
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 3
7  .limit locals 2
8    bipush 5
9    istore 0
10
11   ldc "Hello i11"
12   astore 1
13
14   iload 0
15   dup
16   bipush 2
17   isub
18   iadd
19   istore 0
20
21   return
22 .end method
```



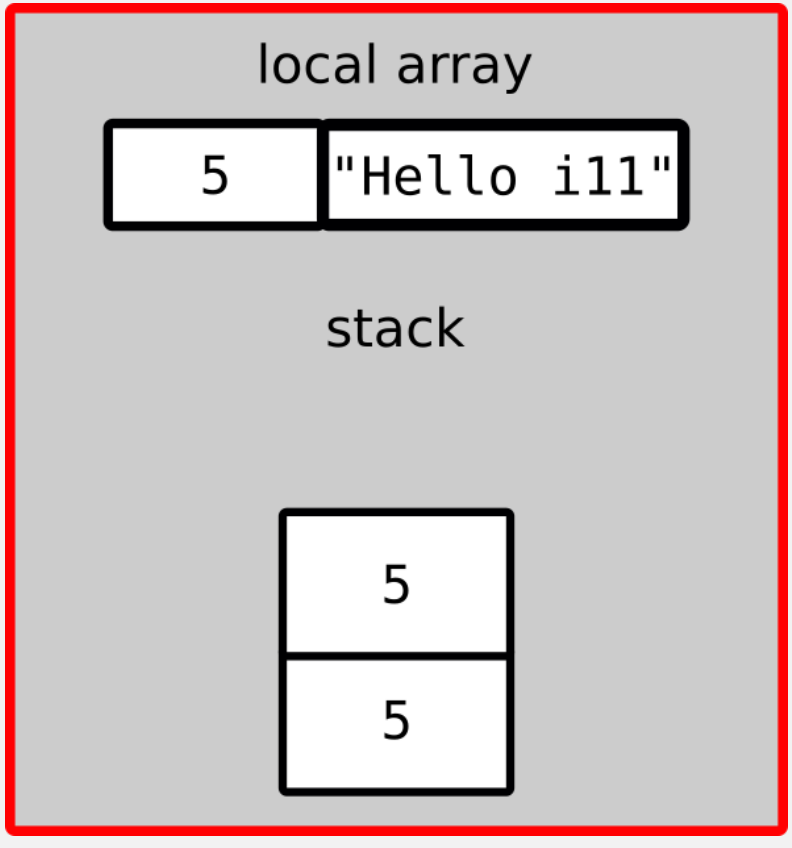
```
1  .source noSource
2  .class public BasicInstructions
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 3
7  .limit locals 2
8    bipush 5
9    istore 0
10
11   ldc "Hello i11"
12   astore 1
13
14   iload 0
15   dup
16   bipush 2
17   isub
18   iadd
19   istore 0
20
21   return
22 .end method
```



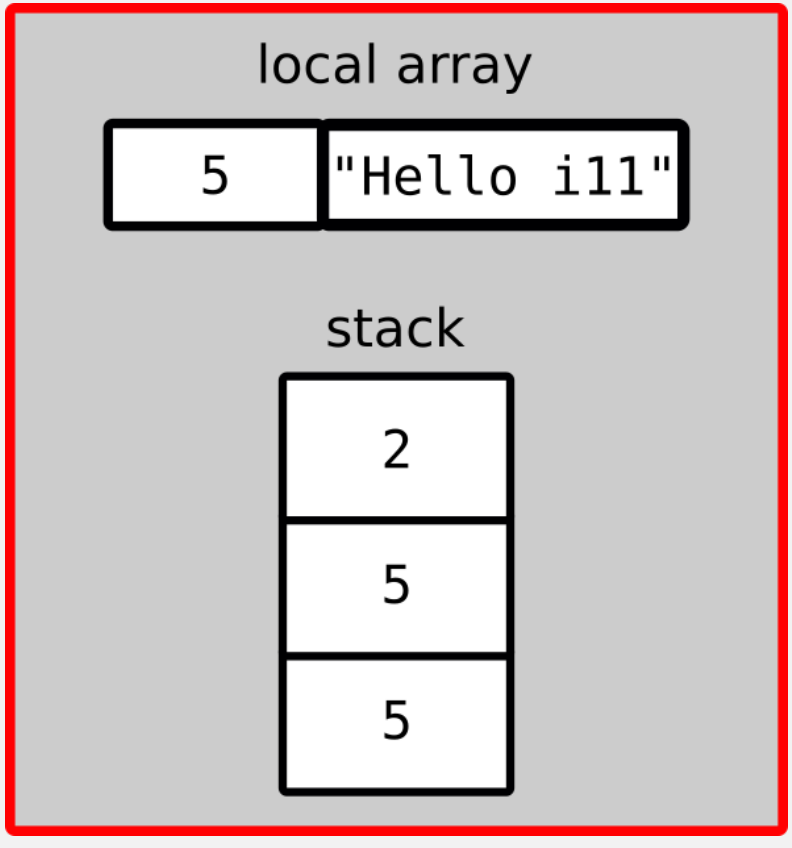
```
1  .source noSource
2  .class public BasicInstructions
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 3
7  .limit locals 2
8    bipush 5
9    istore 0
10
11   ldc "Hello i11"
12   astore 1
13
14   iload 0
15   dup
16   bipush 2
17   isub
18   iadd
19   istore 0
20
21   return
22 .end method
```



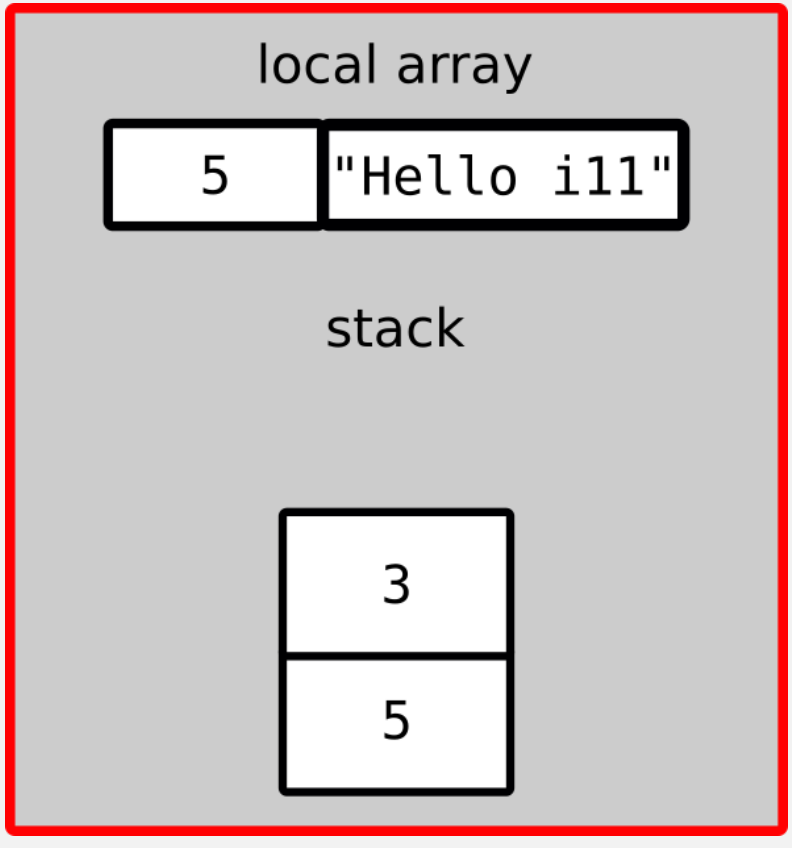
```
1  .source noSource
2  .class public BasicInstructions
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 3
7  .limit locals 2
8    bipush 5
9    istore 0
10
11   ldc "Hello i11"
12   astore 1
13
14   iload 0
15   dup
16   bipush 2
17   isub
18   iadd
19   istore 0
20
21   return
22 .end method
```



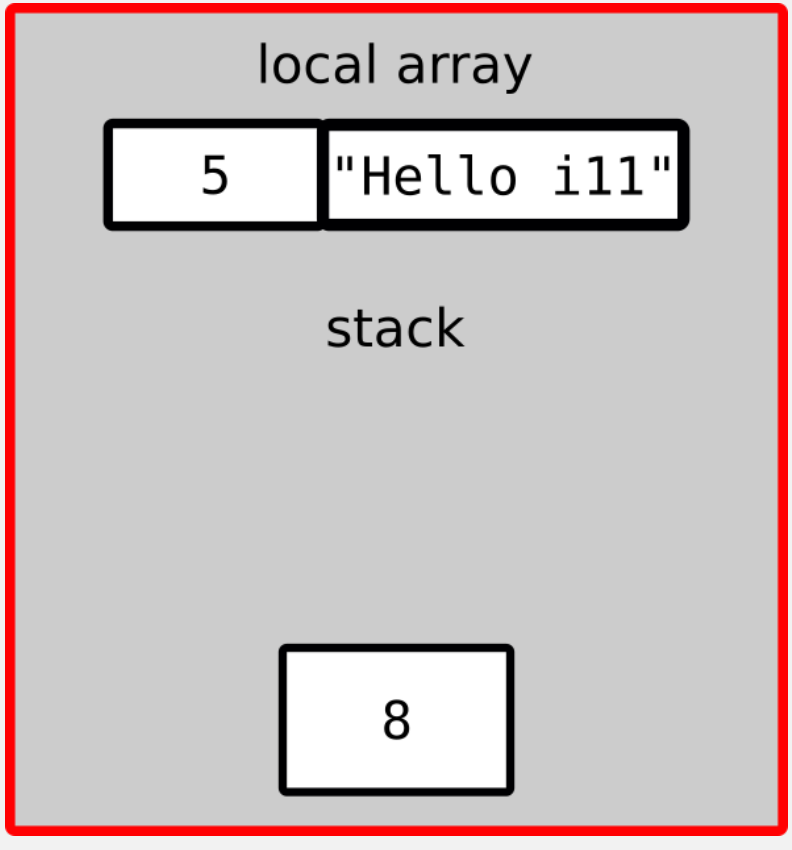
```
1  .source noSource
2  .class public BasicInstructions
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 3
7  .limit locals 2
8    bipush 5
9    istore 0
10
11   ldc "Hello i11"
12   astore 1
13
14   iload 0
15   dup
16   bipush 2
17   isub
18   iadd
19   istore 0
20
21   return
22 .end method
```



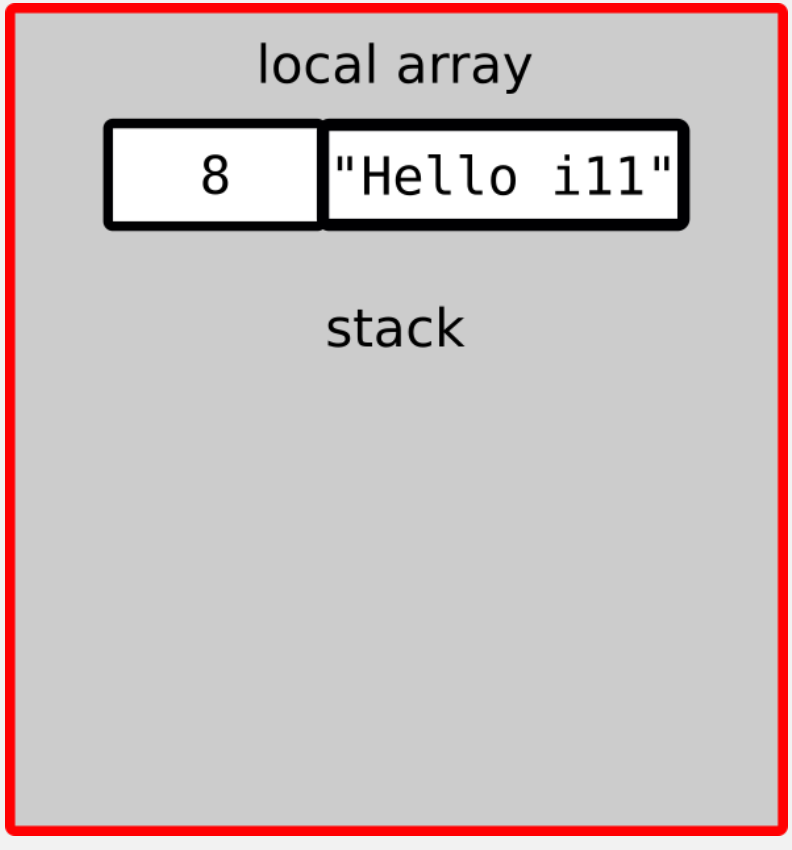
```
1  .source noSource
2  .class public BasicInstructions
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 3
7  .limit locals 2
8    bipush 5
9    istore 0
10
11   ldc "Hello i11"
12   astore 1
13
14   iload 0
15   dup
16   bipush 2
17   isub
18   iadd
19   istore 0
20
21   return
22 .end method
```



```
1  .source noSource
2  .class public BasicInstructions
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 3
7  .limit locals 2
8    bipush 5
9    istore 0
10
11   ldc "Hello i11"
12   astore 1
13
14   iload 0
15   dup
16   bipush 2
17   isub
18   iadd
19   istore 0
20
21   return
22 .end method
```



```
1  .source noSource
2  .class public BasicInstructions
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 3
7  .limit locals 2
8    bipush 5
9    istore 0
10
11   ldc "Hello i11"
12   astore 1
13
14   iload 0
15   dup
16   bipush 2
17   isub
18   iadd
19   istore 0
20
21   return
22 .end method
```



Instructions: Labels and Jumps

- `<labelname>:` - sets a label in the code
- `goto <labelname>` - continues execution of current method at position of label `<labelname>`
- `if_icmpXX <labelname>` - pops 2 ints off the stack, compares them, and jumps to `<labelname>` if comparison evaluates to true

Instructions: Conditional Jumps

With val1 and val2 popped from the operand stack:

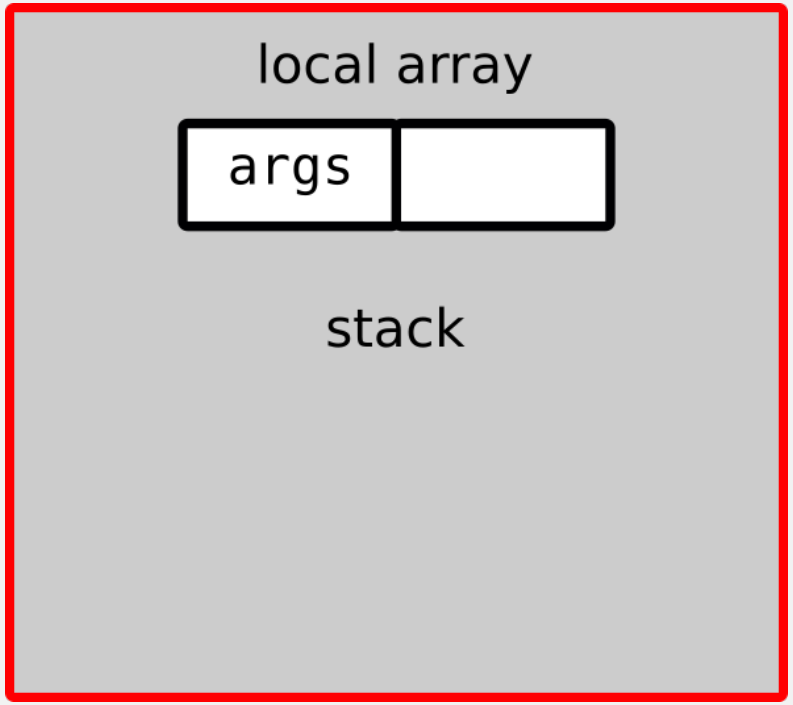
- `if_icmplt` - if val1 $<$ val2, jump to label
- `if_icmpgt` - if val1 $>$ val2, jump to label
- `if_icmpeq` - if val1 $==$ val2, jump to label
- `if_icmpne` - if val1 $!=$ val2, jump to label
- `ifeq` - if TOS is 0, jump to label
- `ifne` - if TOS is not 0, jump to label

$\Rightarrow$ Can be used to assemble relational and logical operators

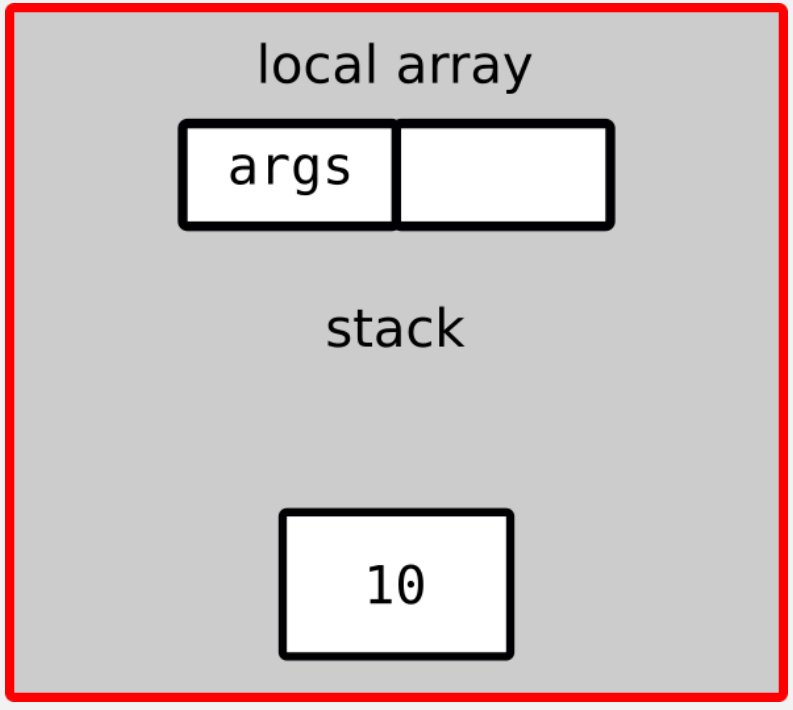
Code Example - LabelsAndJumps.j

```
1  .source noSource
2  .class public LabelsAndJumps
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 2
7  .limit locals 2
8      bipush 10
9      istore 0              ;store 10 in index 0
10     goto L_skip_redef
11     bipush 20
12     istore 0              ;store 20 to index 0 - skipped
13   L_skip_redef:
14     iload 0               ;push value of index 0: 10
15     bipush 15             ;push 15
16     if_icmplt L_is_lesser
17     ldc "greater"         ;push string to stack - skipped
18     goto L_end
19   L_is_lesser:
20     ldc "lesser"          ;push string to stack
21   L_end:                  ;result: string "lesser" on top of stack
22     return
23 .end method
```

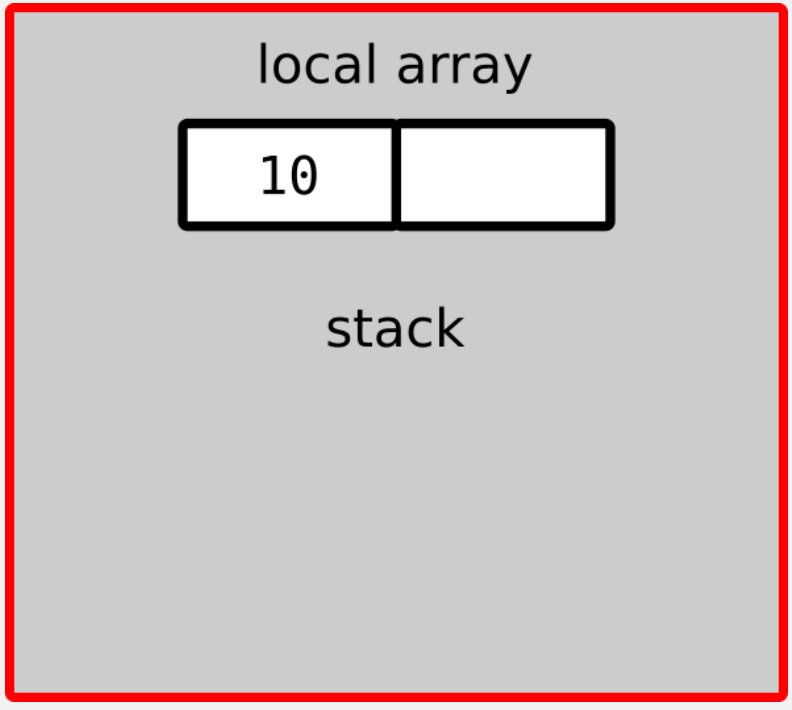
```
1  .source noSource
2  .class public LabelsAndJumps
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 2
7  .limit locals 2
8      bipush 10
9      istore 0
10     goto L_skip_redef
11     bipush 20
12     istore 0
13   L_skip_redef:
14     iload 0
15     bipush 15
16     if_icmplt L_is_lesser
17     ldc "greater"
18     goto L_end
19   L_is_lesser:
20     ldc "lesser"
21   L_end:
22     return
23 .end method
```



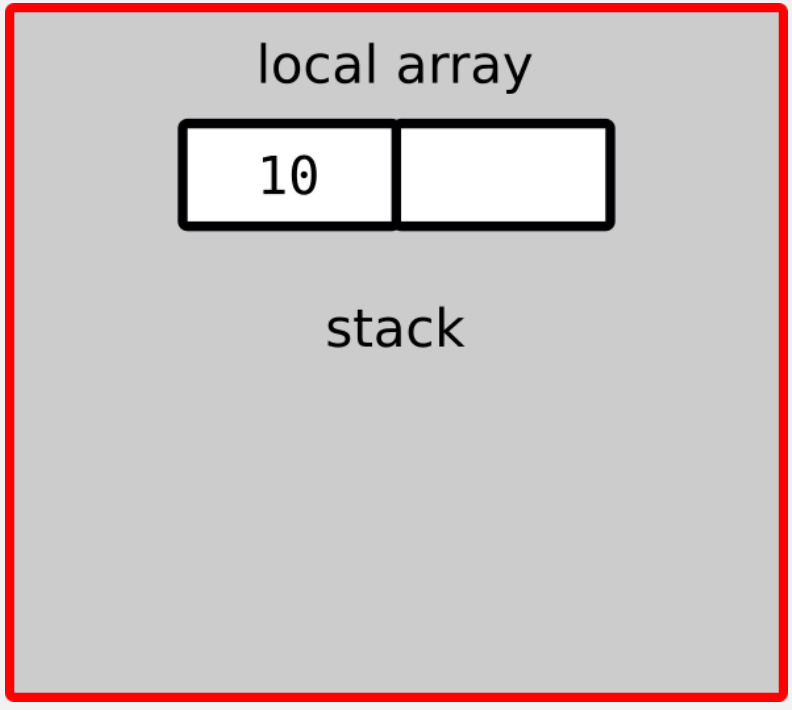
```
1  .source noSource
2  .class public LabelsAndJumps
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 2
7  .limit locals 2
8      bipush 10
9      istore 0
10     goto L_skip_redef
11     bipush 20
12     istore 0
13   L_skip_redef:
14     iload 0
15     bipush 15
16     if_icmplt L_is_lesser
17     ldc "greater"
18     goto L_end
19   L_is_lesser:
20     ldc "lesser"
21   L_end:
22     return
23 .end method
```



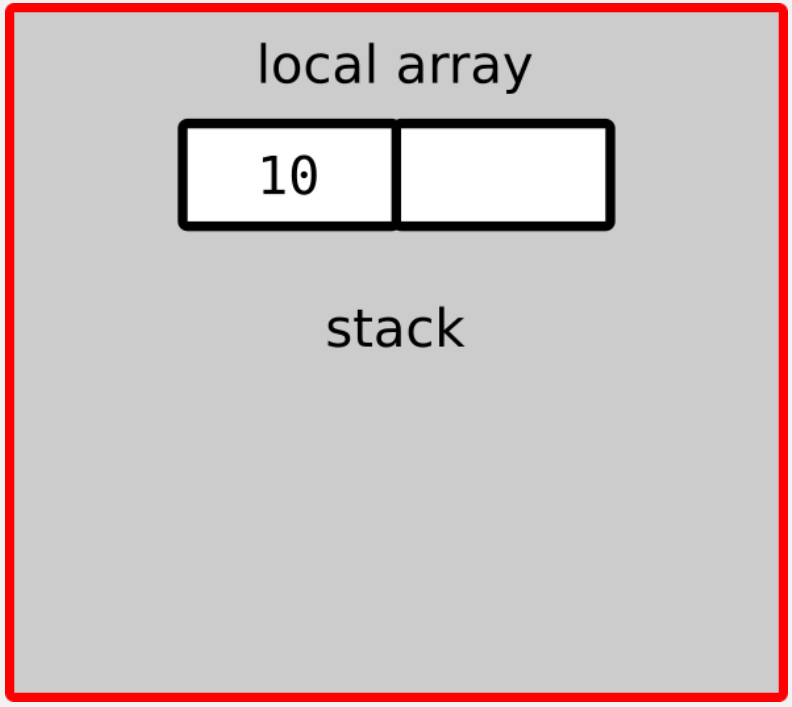
```
1  .source noSource
2  .class public LabelsAndJumps
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 2
7  .limit locals 2
8      bipush 10
9      istore 0
10     goto L_skip_redef
11     bipush 20
12     istore 0
13   L_skip_redef:
14     iload 0
15     bipush 15
16     if_icmplt L_is_lesser
17     ldc "greater"
18     goto L_end
19   L_is_lesser:
20     ldc "lesser"
21   L_end:
22     return
23 .end method
```



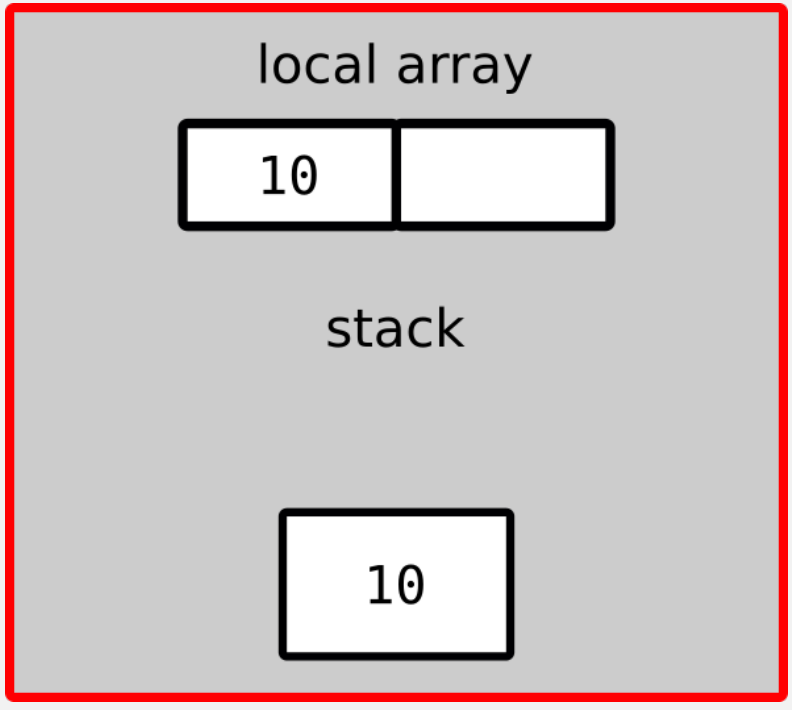
```
1  .source noSource
2  .class public LabelsAndJumps
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 2
7  .limit locals 2
8      bipush 10
9      istore 0
10     goto L_skip_redef
11     bipush 20
12     istore 0
13   L_skip_redef:
14     iload 0
15     bipush 15
16     if_icmplt L_is_lesser
17     ldc "greater"
18     goto L_end
19   L_is_lesser:
20     ldc "lesser"
21   L_end:
22     return
23 .end method
```



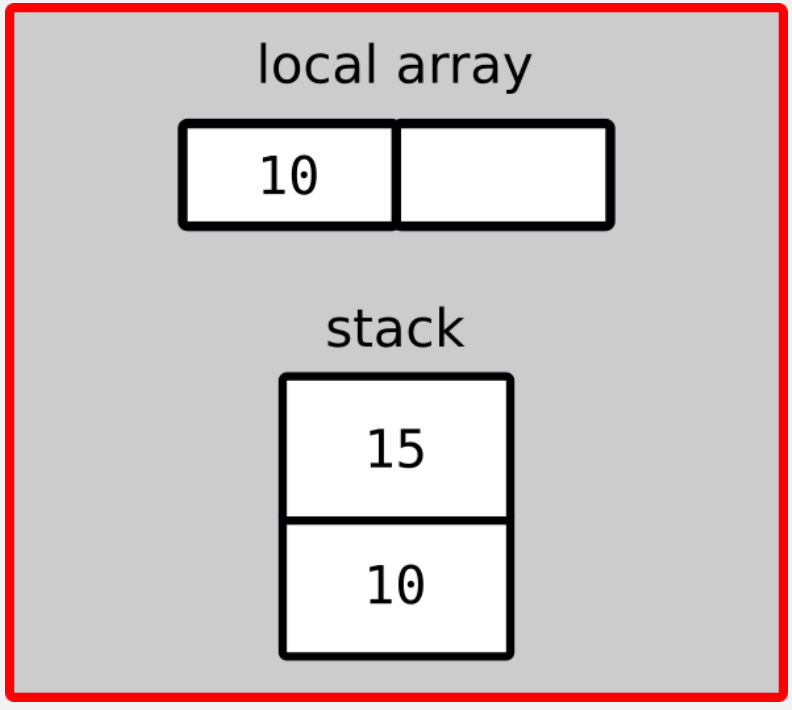
```
1  .source noSource
2  .class public LabelsAndJumps
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 2
7  .limit locals 2
8      bipush 10
9      istore 0
10     goto L_skip_redef
11     bipush 20
12     istore 0
13   L_skip_redef:
14     iload 0
15     bipush 15
16     if_icmplt L_is_lesser
17     ldc "greater"
18     goto L_end
19   L_is_lesser:
20     ldc "lesser"
21   L_end:
22     return
23 .end method
```



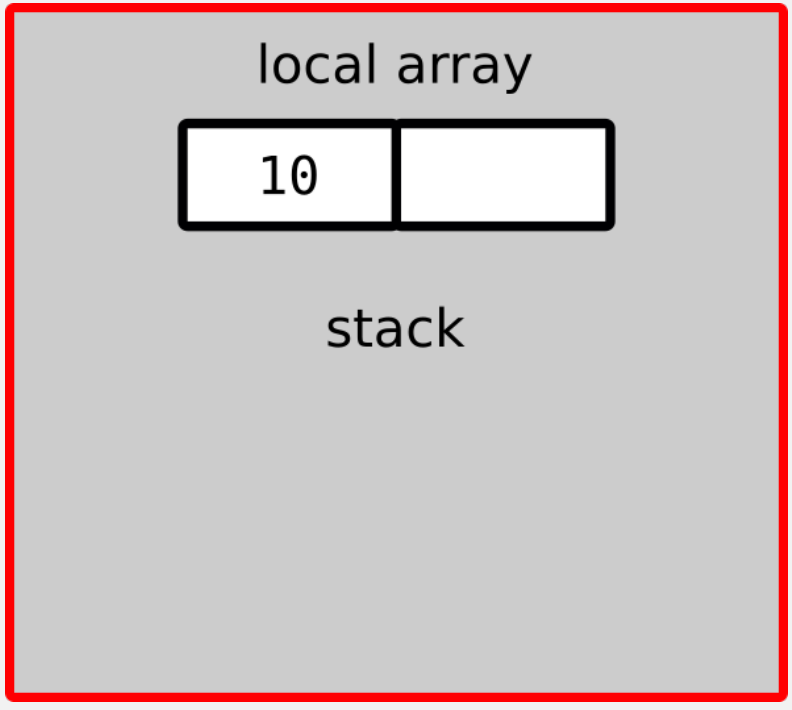
```
1  .source noSource
2  .class public LabelsAndJumps
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 2
7  .limit locals 2
8      bipush 10
9      istore 0
10     goto L_skip_redef
11     bipush 20
12     istore 0
13   L_skip_redef:
14     iload 0
15     bipush 15
16     if_icmplt L_is_lesser
17     ldc "greater"
18     goto L_end
19   L_is_lesser:
20     ldc "lesser"
21   L_end:
22     return
23 .end method
```



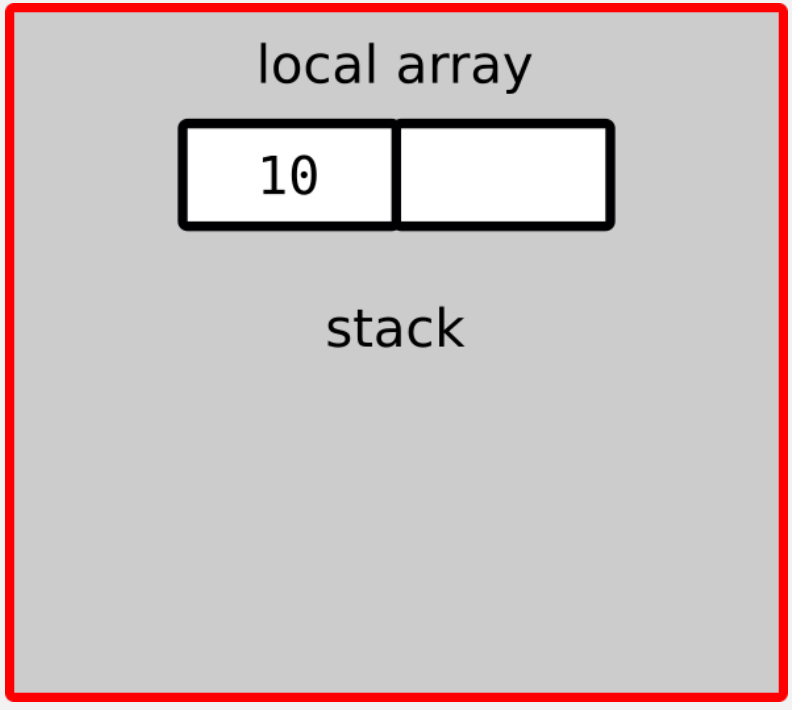
```
1  .source noSource
2  .class public LabelsAndJumps
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 2
7  .limit locals 2
8      bipush 10
9      istore 0
10     goto L_skip_redef
11     bipush 20
12     istore 0
13   L_skip_redef:
14     iload 0
15     bipush 15
16     if_icmplt L_is_lesser
17     ldc "greater"
18     goto L_end
19   L_is_lesser:
20     ldc "lesser"
21   L_end:
22     return
23 .end method
```



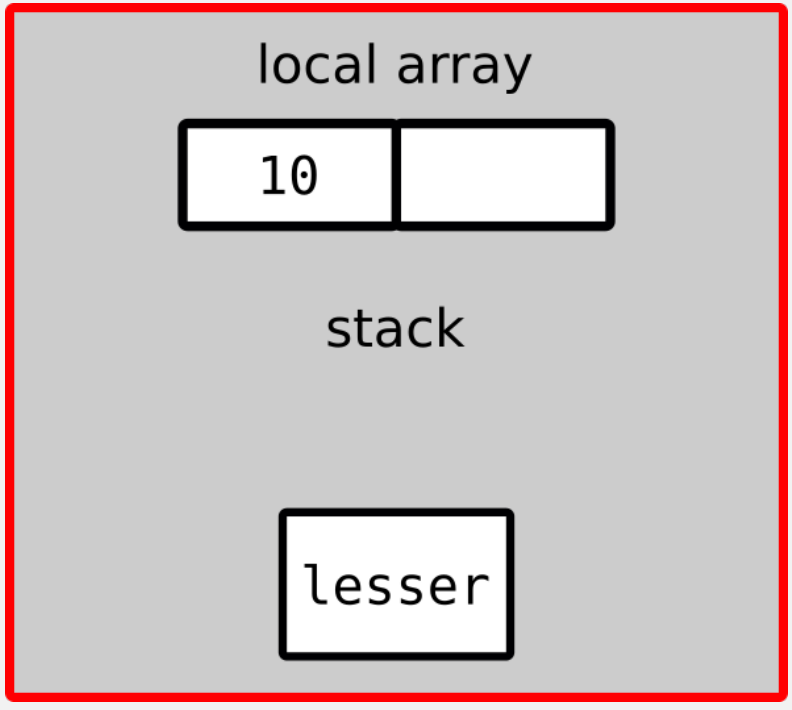
```
1  .source noSource
2  .class public LabelsAndJumps
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 2
7  .limit locals 2
8      bipush 10
9      istore 0
10     goto L_skip_redef
11     bipush 20
12     istore 0
13   L_skip_redef:
14     iload 0
15     bipush 15
16     if_icmplt L_is_lesser
17     ldc "greater"
18     goto L_end
19   L_is_lesser:
20     ldc "lesser"
21   L_end:
22     return
23 .end method
```



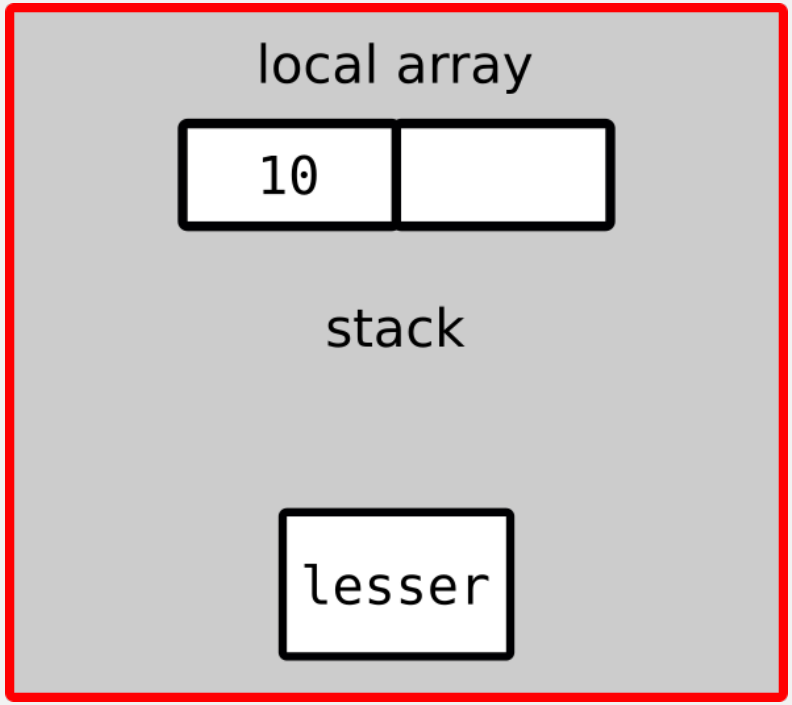
```
1  .source noSource
2  .class public LabelsAndJumps
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 2
7  .limit locals 2
8      bipush 10
9      istore 0
10     goto L_skip_redef
11     bipush 20
12     istore 0
13   L_skip_redef:
14     iload 0
15     bipush 15
16     if_icmplt L_is_lesser
17     ldc "greater"
18     goto L_end
19   L_is_lesser:
20     ldc "lesser"
21   L_end:
22     return
23 .end method
```



```
1  .source noSource
2  .class public LabelsAndJumps
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 2
7  .limit locals 2
8      bipush 10
9      istore 0
10     goto L_skip_redef
11     bipush 20
12     istore 0
13   L_skip_redef:
14     iload 0
15     bipush 15
16     if_icmplt L_is_lesser
17     ldc "greater"
18     goto L_end
19   L_is_lesser:
20     ldc "lesser"
21   L_end:
22     return
23 .end method
```



```
1  .source noSource
2  .class public LabelsAndJumps
3  .super java/lang/Object
4
5  .method public static main([Ljava/lang/String;)V
6  .limit stack 2
7  .limit locals 2
8      bipush 10
9      istore 0
10     goto L_skip_redef
11     bipush 20
12     istore 0
13   L_skip_redef:
14     iload 0
15     bipush 15
16     if_icmplt L_is_lesser
17     ldc "greater"
18     goto L_end
19   L_is_lesser:
20     ldc "lesser"
21   L_end:
22     return
23 .end method
```



Instructions: Field Access

- Use put-/getfield to access field of an object
- Expects correct object-ref on stack
  ⇒ `putfield <class ID>/<field ID> <type>`
  ⇒ `getfield <class ID>/<field ID> <type>`
- Example:
  - Jova: Class `MyClass` with field `int my_field`
  - Jasmin signature: `MyClass/my_field I`
  - Jasmin instruction: `putfield MyClass/my_field I`

Instructions: Non-Static Method Calls

- Call method of object-ref on stack
- Requires arguments and object-ref on stack (in correct order!)
- Usage: `invokevirtual <class ID>/<method signature>`
- Example:
  - Jova: Class `MyClass` with method `int myMethod(int a)`
  - Jasmin signature: `MyClass/myMethod(I)I`
  - Jasmin instruction: `invokevirtual MyClass/myMethod(I)I`

Instructions: Print

- Based on virtual invokation
- Push PrintStream object-ref onto stack
  ⇒ `getstatic java/lang/System/out Ljava/io/PrintStream;`
- Push arguments onto stack (`iload`, `aload`, `ldc`, etc.)
- Invoke matching `PrintStream` method
  ⇒ `invokevirtual java/io/PrintStream/print(I)V`
  ⇒ `invokevirtual java/io/PrintStream/print(Ljava/lang/String;)V`
  ⇒ `invokevirtual java/io/PrintStream/print(Z)V`
- Read methods can be translated in a similar manner

Hints

General Hints

- Use the online documentation to find additional instructions/explanations
- Look at provided examples
- Try and write/translate some simple Jasmin code yourselves and "play around" with Jasmin
- When in doubt: Model a problem in Java $\rightarrow$ compile it and use `javap -c <Class>` to decompile the .class files

Implementation Hints

- One possibility: Use custom visitor/listener to generate instructions from the parse tree in combination with the symbol table
- Or create internal intermediate representation of Jova code which then gets translated to Jasmin code
- Use `aload_0` to get `this`
- Number your labels
- Map Jova variables to local variable array indices in the symbol table
- Instruction `swap`/`dup` may be useful in some cases

References/Resources

- Jasmin User Guide:
  `http://jasmin.sourceforge.net/guide.html`
- JVM structure:
  `https://docs.oracle.com/javase/specs/jvms/se21/html/jvms-2.html`
- JVM instruction set:
  `https://docs.oracle.com/javase/specs/jvms/se21/html/jvms-6.html`
- Aho, Alfred V., Ravi Sethi, and Jeffrey D. Ullman. Compilers, Principles, Techniques. Boston: Addison wesley, 1986.

Questions?

Happy Hacking!