CS 153: Concepts of Compiler Design

November 14 Class Meeting

Department of Computer Science
San Jose State University

Fall 2017
Instructor: Ron Mak
www.cs.sjsu.edu/~mak

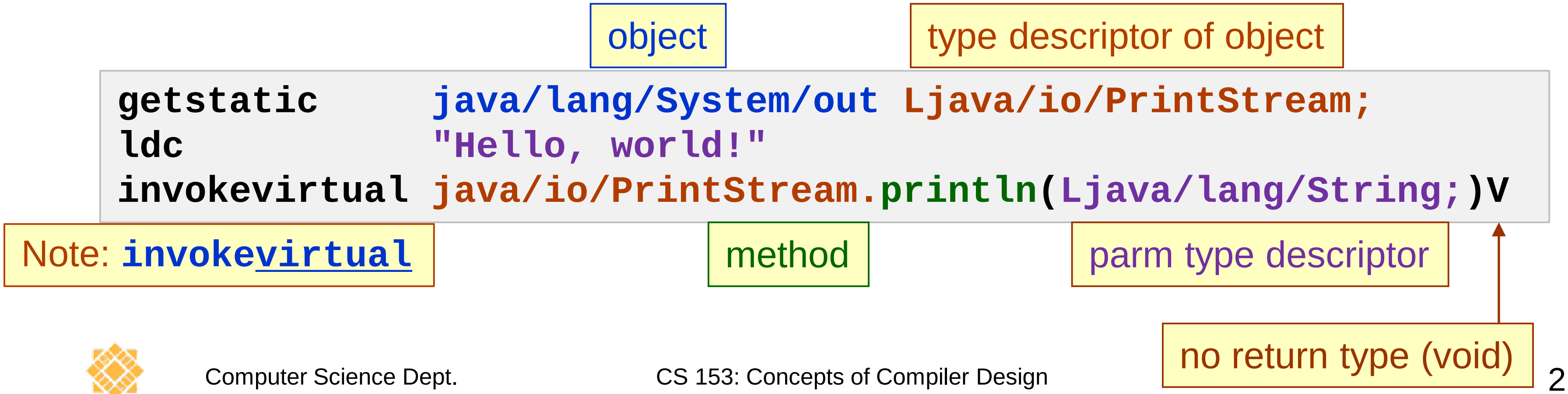
Code to Call **`System.out.println()`**

- What does the method call

  ```
  System.out.println("Hello, world!")
  ```

  require on the operand stack?

  - A reference to the <u>object</u> **`java.lang.System.out`** with <u>datatype</u> **`java.io.PrintStream`**
  - A reference to the <u>object</u> **`"Hello, world!"`** with <u>datatype</u> **`java.lang.String`**



object

type descriptor of object

```
getstatic     java/lang/System/out Ljava/io/PrintStream;
ldc           "Hello, world!"
invokevirtual java/io/PrintStream.println(Ljava/lang/String;)V
```

Note: **`invoke<u>virtual</u>`**

method

parm type descriptor

no return type (void)

`System.out.println()`, *cont'd*

- Compile the Pascal call

```
writeln('Sum = ', sum)
```

as if it were the Java

```
System.out.println(
    new StringBuilder(" Sum = ")
            .append(sum)
            .toString()
);
```

Remember to use **`javap`**!

Each call to **`invokevirtual`** requires an object reference and then any required actual parameter values on the operand stack.

```
getstatic java/lang/System/out Ljava/io/PrintStream;
new       java/lang/StringBuilder
dup
ldc "Sum = "
invokenonvirtual java/lang/StringBuilder/<init>(Ljava/lang/String;)V
getstatic     Adder/sum I
invokevirtual java/lang/StringBuilder/append(I)Ljava/lang/StringBuilder;
invokevirtual java/lang/StringBuilder/toString()Ljava/lang/String;
invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
```

Why do we need this **dup** instruction?

String.format()

- A more elegant way to compile a call to Pascal's standard **writeln()** procedure is to use Java's **String.format()** method.

- Compile Pascal

```
writeln('The square root of ', n:4,
        ' is ', root:8:4);
```

as if it were the Java

```
System.out.print(
    String.format(
        "The square root of %4d is %8.4f\n",
        n, root)
);
```

`String.format()`, *cont'd*

- The Java `String.format()` method has a variable-length parameter list.
- The first parameter is the format string.
- Similar to C's format strings for `printf()`.

- The code generator must construct the format string.
  - Pascal:

```
('The square root of ', n:4, ' is ', root:8:4)
```

  - Equivalent Java:

```
("The square root of %4d is %8.4f\n", n, root)
```

`String.format()`, *cont'd*

- The remaining parameters are the values to be formatted, one for each format specification in the format string.

- Jasmin passes these remaining parameters as a one-dimensional array of objects.

- Therefore, we must emit code to create and initialize the array and leave its reference on the operand stack.

String.format(), *cont'd*

```
s = String.format(
        "The square root of %4d is %8.4f\n",
        n, root);
```

- Instruction **aastore** operands on the stack:
  - Array reference
  - Index value
  - Element value (object reference)

```
ldc             "The square root of %4d is %8.4f\n"
iconst_2
anewarray       java/lang/Object
dup
iconst_0
getstatic       FormatTest/n I
invokestatic    java/lang/Integer.valueOf(I)Ljava/lang/Integer;
aastore
dup
iconst_1
getstatic       FormatTest/root F
invokestatic    java/lang/Float.valueOf(F)Ljava/lang/Float;
aastore
invokestatic    java/lang/String.format(Ljava/lang/String;[Ljava/lang/Object;)
                                                    Ljava/lang/String;
putstatic       FormatTest/s Ljava/lang/String;
```

Create an **array of size 2** and leave the array reference on the operand stack.

Store **element 0:** The value of **n**.

Why the **dup** instructions?

Store **element 1:** The value of **root**.

String.format(), *cont'd*

```
System.out.print(
    String.format("The square root of %4d is 8.4f\n",
                  n, root);
);
```

```
getstatic    java/lang/System/out Ljava/io/PrintStream;
ldc          "The square root of %4d is %8.4f\n"
iconst_2
anewarray    java/lang/Object
dup
iconst_0
getstatic     FormatTest/n I
invokestatic  java/lang/Integer.valueOf(I)Ljava/lang/Integer;
aastore
dup
iconst_1
getstatic     FormatTest/root F
invokestatic  java/lang/Float.valueOf(F)Ljava/lang/Float;
aastore
invokestatic  java/lang/String.format(Ljava/lang/String;
                                     [Ljava/lang/Object;)Ljava/lang/String;
invokevirtual java/io/PrintStream.print(Ljava/lang/String;)V
```

Easier: Use the newer `System.out.printf()`.

Code Generation for Arrays and Subscripts

- Code to allocate memory for an array variable.
- Code to allocate memory for each non-scalar array element.
- Code for a subscripted variable in an expression.
- Code for a subscripted variable that is an assignment target.

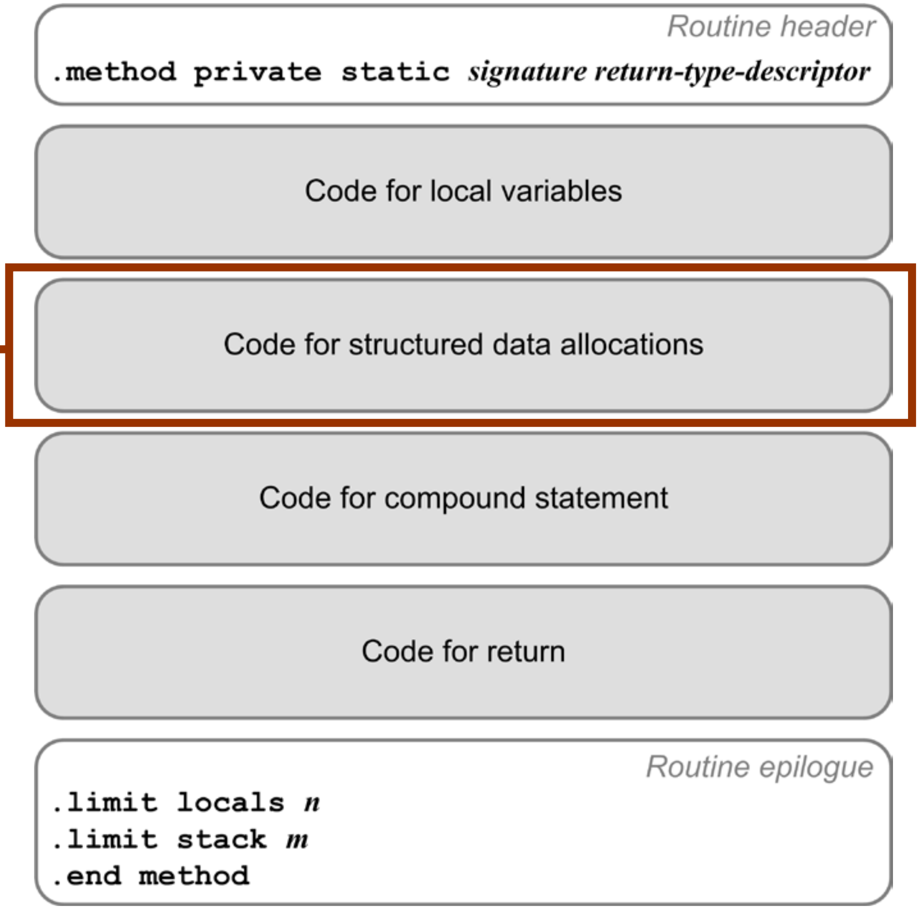
Arrays and Subscripts, *cont'd*

- Allocate memory for single-dimensional arrays:
  - Instruction **`newarray`** for scalar elements.
  - Instruction **`anewarray`** for non-scalar elements.
- Allocate memory for multidimensional arrays:
  - Instruction **`multianewarray`**.

Allocating Memory for Arrays

- □ Recall the code template for a Jasmin method.

Code to allocate **arrays** here!



Routine header

```
.method private static signature return-type-descriptor
```

Code for local variables

Code for structured data allocations

Code for compound statement

Code for return

Routine epilogue

```
.limit locals n
.limit stack m
.end method
```

- □ Pascal <u>automatically allocates</u> memory for arrays declared in the main program or locally in a procedure or function.
  - The memory allocation occurs whenever the routine is called.
  - This is separate from <u>dynamically allocated</u> data using pointers and **new**.

Therefore, our generated Jasmin code must implement this automatic runtime behavior.

Example: Allocate Memory for Scalar Arrays

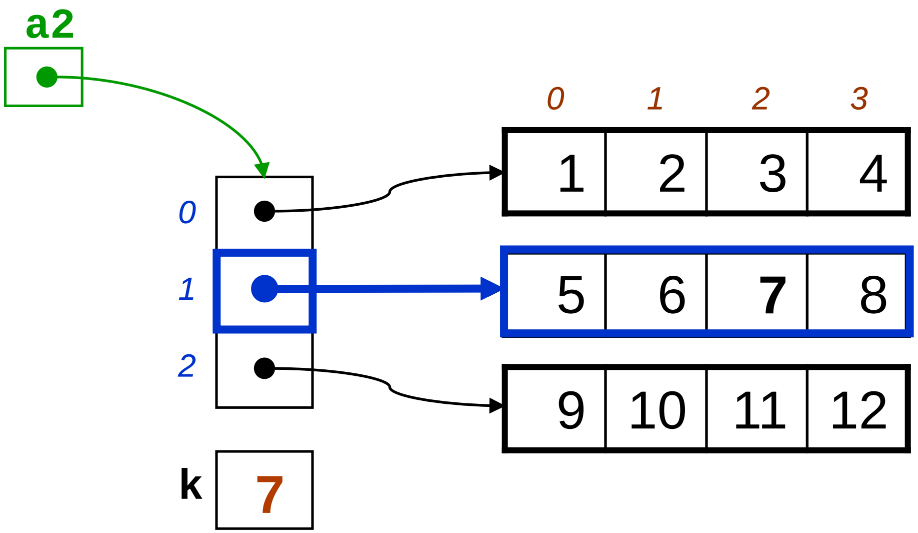
```
PROGRAM ArrayTest;

TYPE
    vector = ARRAY[0..9] OF integer;
    matrix = ARRAY[0..4, 0..4] OF integer;
    cube   = ARRAY[0..1, 0..2, 0..3] OF integer;

VAR
    i, j, k, n : integer;
    a1         : vector;
    a2         : matrix;
    a3         : cube;

BEGIN
    ...
END.
```

```
bipush   10
newarray int
putstatic         arraytest/a1 [I

iconst_5
iconst_5
multianewarray    [[I 2
putstatic         arraytest/a2 [[I

iconst_2
iconst_3
iconst_4
multianewarray    [[[I 3
putstatic         arraytest/a3 [[[I
```

Access an Array Element of a 2-D Array

```
PROGRAM ArrayTest;

TYPE
  matrix = ARRAY[0..2, 0..3]
             OF integer;

VAR
  i, j, k : integer;
  a2      : matrix;

BEGIN
  ...
  i := 1;
  j := 2;
  k := a2[i, j];
  ...
END.
```

1	2	3	4
5	6	7	8
9	10	11	12



```
getstatic   arraytest/a2 [[I
getstatic   arraytest/i I
aaload
getstatic   arraytest/j I
iaload
putstatic   arraytest/k I
```

Subscripted Variables in Expressions

```
PROGRAM ArrayTest;

TYPE
    vector = ARRAY[0..9] OF integer;
    matrix = ARRAY[0..4, 0..4] OF integer;
    cube   = ARRAY[0..1, 0..2, 0..3] OF integer;

VAR
    i, j, k, n : integer;
    a1         : vector;
    a2         : matrix;
    a3         : cube;

BEGIN
    ...
    j := a1[i];
    k := a2[i, j];
    n := a3[i, j, k];
    ...
END.
```

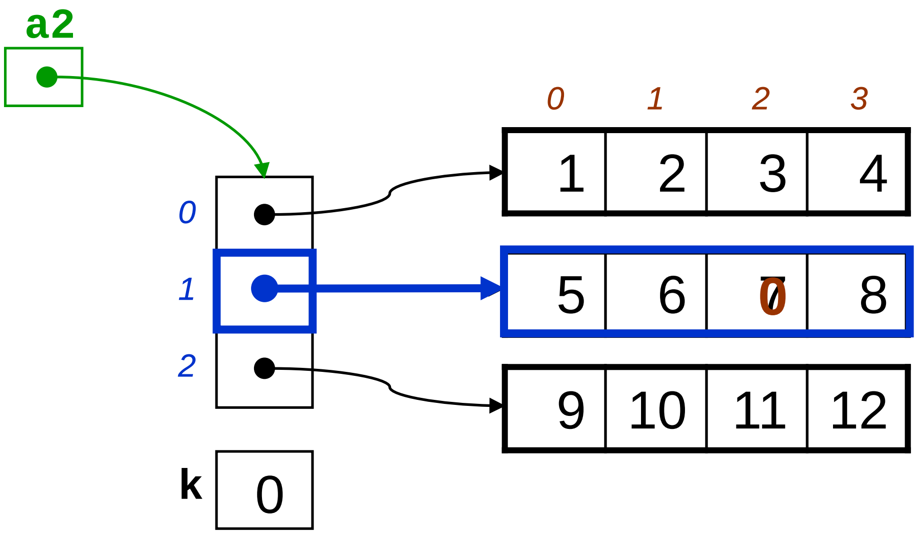
```
getstatic  arraytest/a1 [I
getstatic  arraytest/i I
iaload
putstatic  arraytest/j I

getstatic  arraytest/a2 [[I
getstatic  arraytest/i I
aaload
getstatic  arraytest/j I
iaload
putstatic  arraytest/k I

getstatic  arraytest/a3 [[[I
getstatic  arraytest/i I
aaload
getstatic  arraytest/j I
aaload
getstatic  arraytest/k I
iaload
putstatic  arraytest/n I
```

- **`iaload`** : push a scalar value from an array element value
- **`aaload`**: push an array element address

Set an Array Element of a 2-D Array

```
PROGRAM ArrayTest;

TYPE
  matrix = ARRAY[0..2, 0..3]
                OF integer;

VAR
  i, j, k : integer;
  a2      : matrix;

BEGIN
  ...
  i := 1;
  j := 2;
  k := 0;
  a2[i, j] := k;
  ...
END.
```

1	2	3	4
5	6	7	8
9	10	11	12



```
getstatic  arraytest/a2 [[I
getstatic  arraytest/i I
aaload
getstatic  arraytest/j I
getstatic  arraytest/k I
iastore
```

More Subscripted Variables

```
PROGRAM ArrayTest;

TYPE
    vector = ARRAY[0..9] OF integer;
    matrix = ARRAY[0..4, 0..4] OF integer;
    cube   = ARRAY[0..1, 0..2, 0..3] OF integer;

VAR
    i, j, k, n : integer;
    a1         : vector;
    a2         : matrix;
    a3         : cube;

BEGIN
    ...
    a3[i][a1[j]][k] := a2[i][j] - a3[k, 2*n][k+1];
    ...
END.
```

Instruction **aaload** pushes the address of one dimension of an array. Instruction **iaload** pushes the integer value of an array element.

```
getstatic arraytest/a3 [[[I
getstatic arraytest/i I
aaload
getstatic arraytest/a1 [I
getstatic arraytest/j I
iaload
aaload
getstatic arraytest/k I

getstatic arraytest/a2 [[I
getstatic arraytest/i I
aaload
getstatic arraytest/j I
iaload

getstatic arraytest/a3 [[[I
getstatic arraytest/k I
aaload
iconst_2
getstatic arraytest/n I
imul
aaload
getstatic arraytest/k I
iconst_1
iadd
iaload

isub
iastore
```

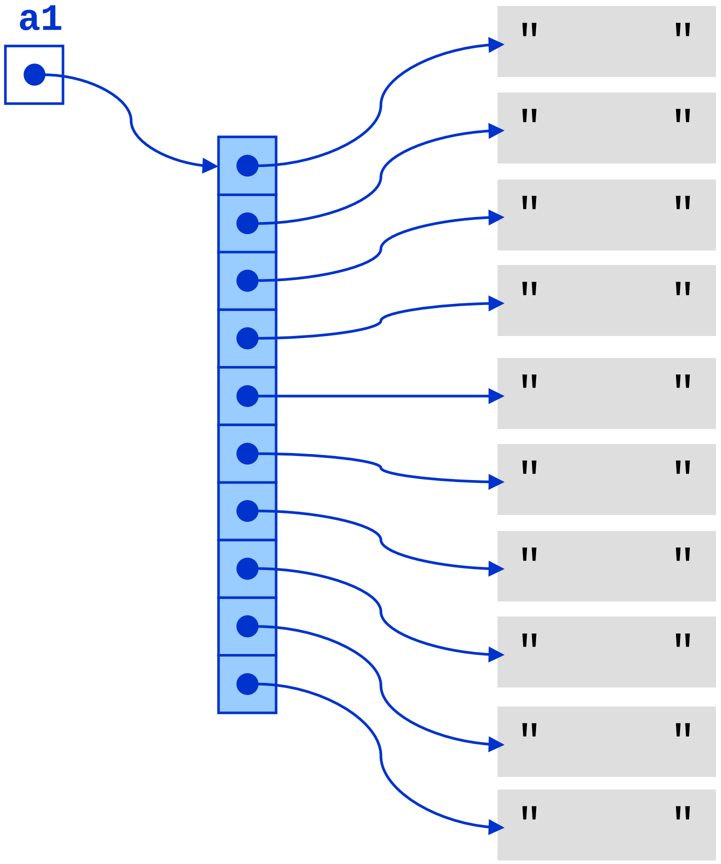
What's on the stack after this instruction?

Allocate Memory for Non-Scalar Arrays

- For a non-scalar array, we must generate code to :
  - Allocate memory for the array itself.
  - Similar to a scalar array, except that each element will contain a reference to its data.
  - Allocate memory for the data of each array element and initialize each element.

Allocate Memory for a 1-D String Array

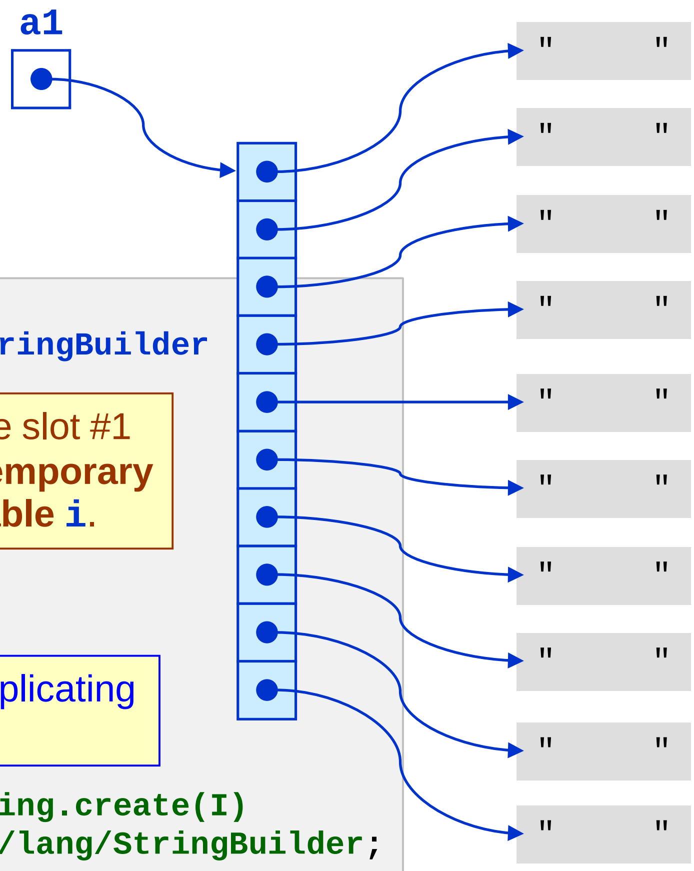
```
PROGRAM AllocArrayTest2;

TYPE
    string = ARRAY[1..5] OF char;
    vector = ARRAY[0..9] OF string;

VAR
    a1 : vector;

BEGIN
END.
```



- Each array element should contain a <u>reference</u> to a string object.

Memory for a 1-D String Array, *cont’d*

```
PROGRAM AllocArrayTest2;

TYPE
    string = ARRAY[1..5] OF char;
    vector = ARRAY[0..9] OF string;

VAR
    a1 : vector;

BEGIN
END.
```



```
       bipush      10
       anewarray   java/lang/StringBuilder

       iconst_0
       istore_1
L001:
       iload_1
       bipush      10
       if_icmpge   L002

       dup
       iload_1
       iconst_5
       invokestatic   PaddedString.create(I)
                          Ljava/lang/StringBuilder;
       aastore

       iinc   1 1
       goto   L001
L002:
       putstatic      allocarraytest2/a1
                         [Ljava/lang/StringBuilder;
```

Allocate slot #1 as the **temporary variable i**.

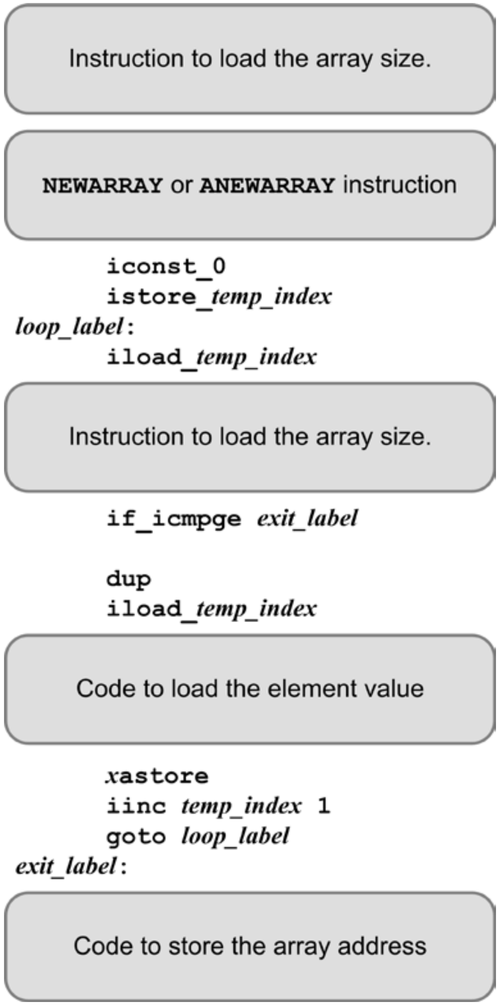
What are we duplicating and why?

Like the Java code:

```
for (int i = 0; i < 10; ++i)
{
  a1[i] =
    PaddedString.create(5);
}
```

`PaddedString` is a class in the Pascal Runtime Library.

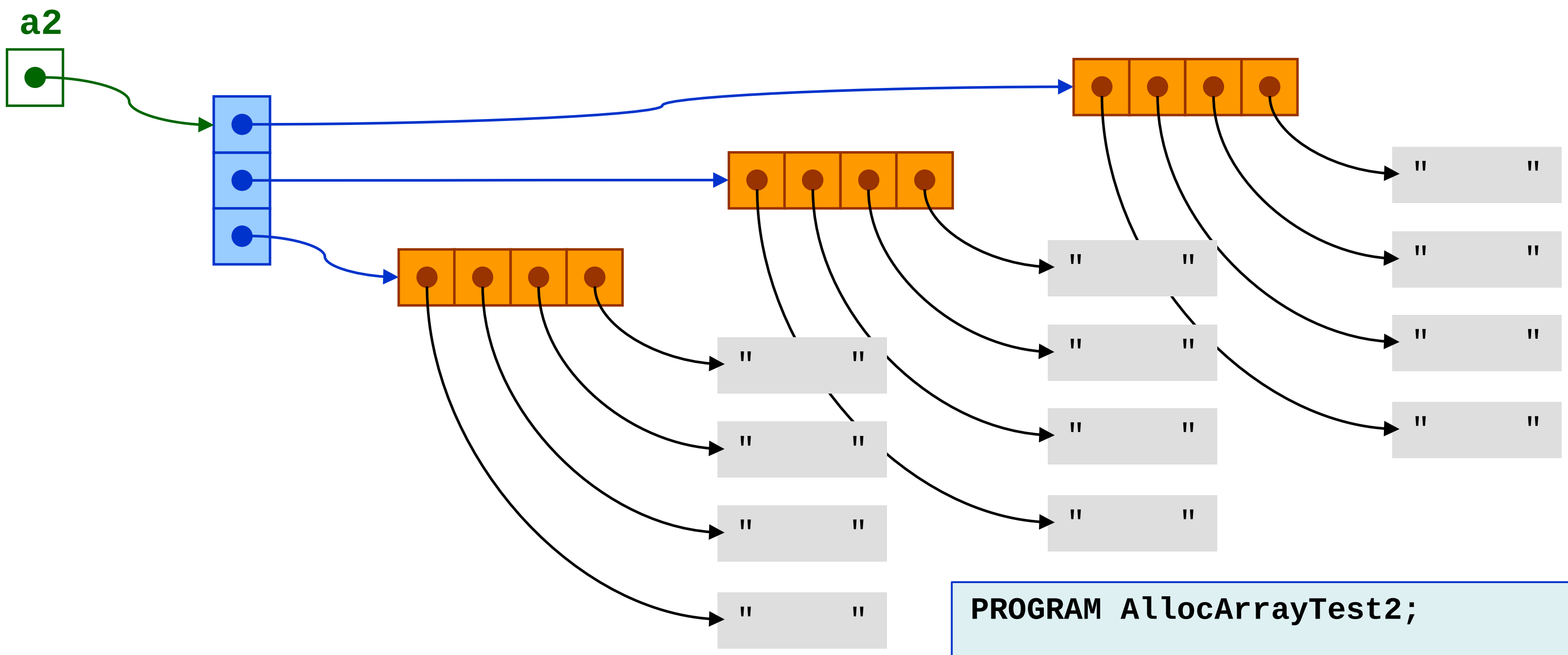
Code Template: 1-D Non-Scalar Array



```
        Instruction to load the array size.

        NEWARRAY or ANEWARRAY instruction

        iconst_0
        istore_temp_index
loop_label:
        iload_temp_index

        Instruction to load the array size.

        if_icmpge exit_label

        dup
        iload_temp_index

        Code to load the element value

        xastore
        iinc temp_index 1
        goto loop_label
exit_label:

        Code to store the array address
```

```
         bipush     10

         anewarray  java/lang/StringBuilder

         iconst_0
         istore_1
L001:
         iload_1

         bipush     10

         if_icmpge  L002
         dup
         iload_1

         iconst_5
         invokestatic  PaddedString.create(I)Ljava/lang/StringBuilder;

         aastore
         iinc   1 1
         goto   L001
L002:

         putstatic     allocarraytest2/a1 [Ljava/lang/StringBuilder;
```

Allocate Memory for a 2-D String Array



```
PROGRAM AllocArrayTest2;

TYPE
  string = ARRAY[1..5] OF char;
  matrix = ARRAY[0..2, 0..3] OF string;

VAR
  a2 : matrix;

BEGIN
END.
```

Memory for a 2-D String Array, *cont’d*

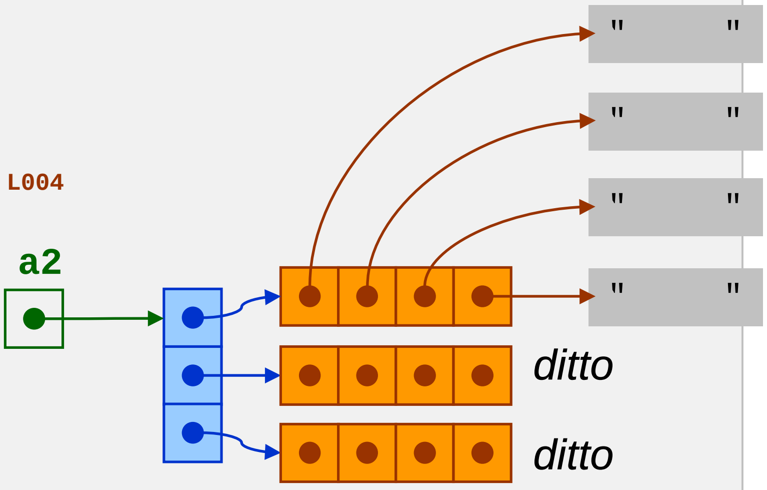
```
PROGRAM AllocArrayTest2;

TYPE
  string = ARRAY[1..5] OF char;
  matrix = ARRAY[0..2, 0..3] OF string;

VAR
  a2 : matrix;

BEGIN
END.
```

Allocate slots #1 and #2 as the **temporary variables** **i** and **j**.

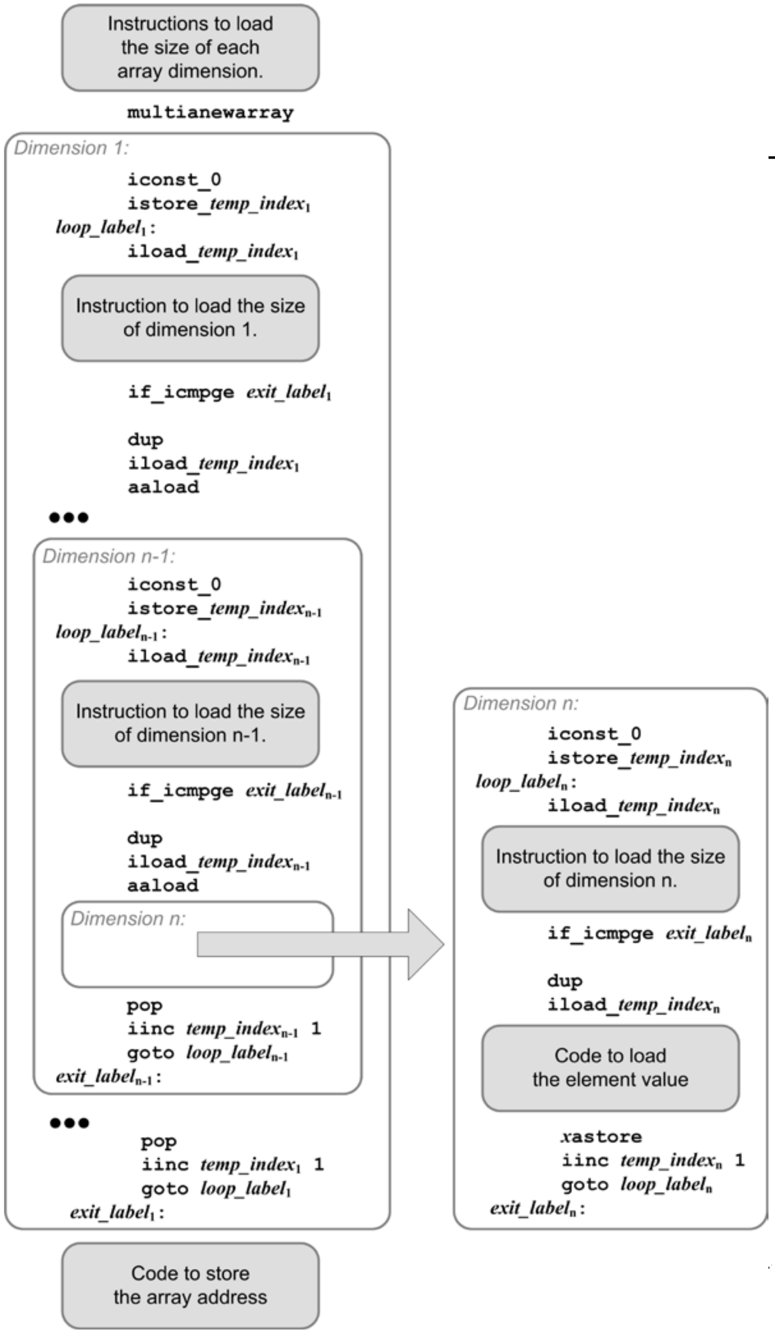
Like the Java code:

```
for (int i = 0; i < 3; ++i)
{
    for (int j = 0; j < 4; ++j)
    {
        a2[i][j] =
            PaddedString.create(5);
    }
}
```

```
       iconst_3
       iconst_4
       multianewarray  [[Ljava/lang/StringBuilder; 2

       iconst_0
       istore_1
L003:
       iload_1
       iconst_3
       if_icmpge  L004

       dup
       iload_1
       aaload

       iconst_0
       istore_2
L005:
       iload_2
       iconst_4
       if_icmpge  L006

       dup
       iload_2
       iconst_5
       invokestatic  PaddedString.create(I)Ljava/lang/StringBuilder;
       aastore

       iinc    2 1
       goto    L005
L006:
       pop
       iinc    1 1
       goto    L003
L004:
       putstatic  allocarraytest2/a2 [[Ljava/lang/StringBuilder;
```



Code Template: *n*-D Non-Scalar Array



Instructions to load the size of each array dimension.

multianewarray

Dimension 1:

iconst_0
istore_*temp_index*$_1$
loop_label$_1$:
iload_*temp_index*$_1$

Instruction to load the size of dimension 1.

if_icmpge *exit_label*$_1$

dup
iload_*temp_index*$_1$
aaload

•••

Dimension n-1:

iconst_0
istore_*temp_index*$_{n-1}$
loop_label$_{n-1}$:
iload_*temp_index*$_{n-1}$

Instruction to load the size of dimension n-1.

if_icmpge *exit_label*$_{n-1}$

dup
iload_*temp_index*$_{n-1}$
aaload

Dimension n:

pop
iinc *temp_index*$_{n-1}$ 1
goto *loop_label*$_{n-1}$
exit_label$_{n-1}$:

•••

pop
iinc *temp_index*$_1$ 1
goto *loop_label*$_1$
exit_label$_1$:

Code to store the array address

Dimension n:

iconst_0
istore_*temp_index*$_n$
loop_label$_n$:
iload_*temp_index*$_n$

Instruction to load the size of dimension n.

if_icmpge *exit_label*$_n$

dup
iload_*temp_index*$_n$

Code to load the element value

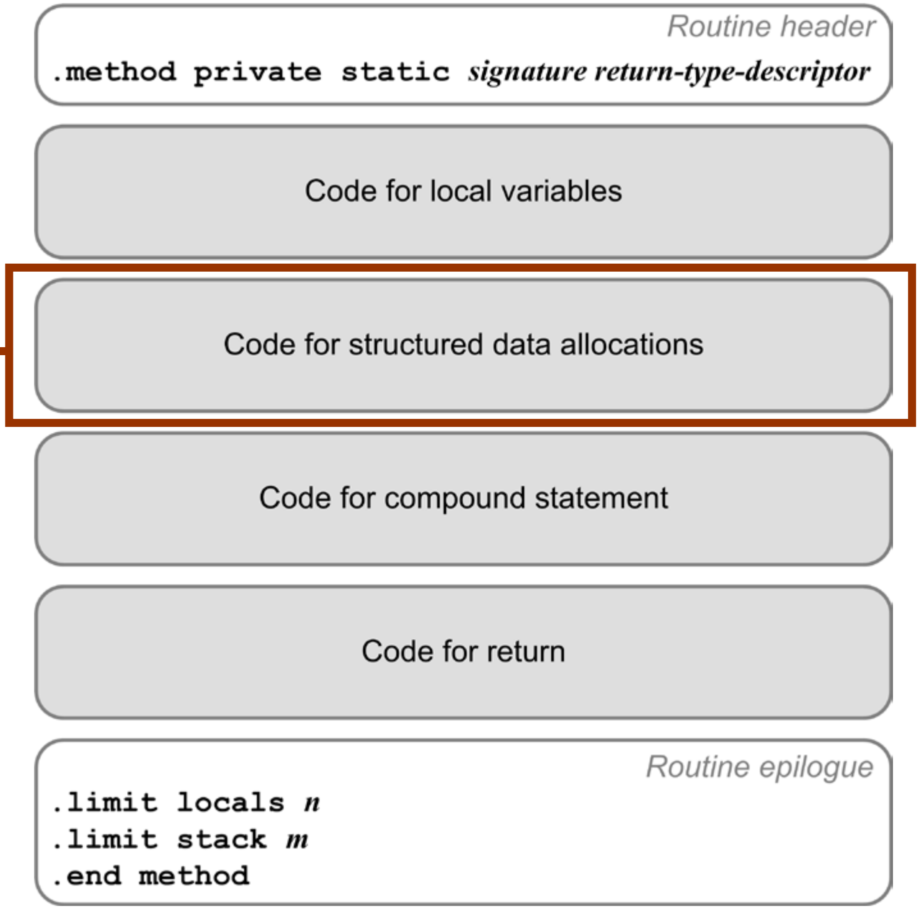
xastore
iinc *temp_index*$_n$ 1
goto *loop_label*$_n$
exit_label$_n$:

```
      iconst_5
      iconst_4
      multianewarray  [[Ljava/lang/StringBuilder; 2

      iconst_0
      istore_1
L003:
      iload_1
      iconst_3
      if_icmpge  L004

      dup
      iload_1
      aaload

      iconst_0
      istore_2
L005:
      iload_2
      iconst_4
      if_icmpge  L006

      dup
      iload_2
      iconst_5
      invokestatic  PaddedString.create(I)Ljava/lang/StringBuilder;
      aastore

      iinc    2 1
      goto    L005
L006:
      pop
      iinc    1 1
      goto    L003
L004:
      putstatic  allocarraytest2/a2 [[Ljava/lang/StringBuilder;
```

Records and Fields

- Recall the code template for a Jasmin method.

Code to allocate **records** here!



Routine header

```
.method private static signature return-type-descriptor
```

Code for local variables

Code for structured data allocations

Code for compound statement

Code for return

Routine epilogue

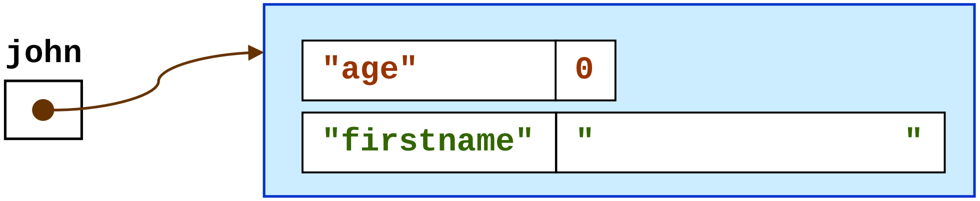
```
.limit locals n
.limit stack m
.end method
```

- Implement the value of <u>each</u> Pascal record variable as a `java.util.HashMap` object.
  - **Keys:** Field names (as strings)
  - **Values:** Field values (as objects)

Pascal Records in the JVM

- Each record value is a separate hash table.
  - **Keys:** field names
  - **Values:** field values

- Allocate and initialize each value.



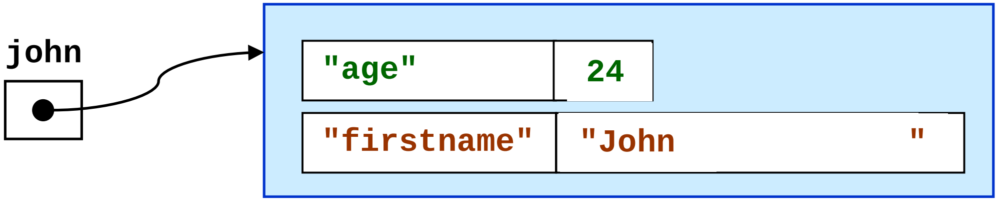
```
PROGRAM RecordTest2;

TYPE
  String16 =
    ARRAY [1..16] OF char;

  PersonRec =
    RECORD
      firstName : String16;
      age       : integer;
    END;

VAR
  john : PersonRec;

BEGIN
  ...
END.
```

```
new  java/util/HashMap
dup
invokenonvirtual  java/util/HashMap/<init>()V
dup
ldc     "age"
iconst_0
invokestatic   java/lang/Integer.valueOf(I)Ljava/lang/Integer;
invokevirtual  java/util/HashMap.put(Ljava/lang/Object;
                         Ljava/lang/Object;)Ljava/lang/Object;
pop
dup
ldc      "firstname"
bipush  16
invokestatic   PaddedString.create(I)Ljava/lang/StringBuilder;
invokevirtual  java/util/HashMap.put(Ljava/lang/Object;
                         Ljava/lang/Object;)Ljava/lang/Object;
pop
putstatic  recordtest2/john Ljava/util/HashMap;
```

Convert the **int** value 0 to an **Integer** object.

Why **pop**?

Set the Values of Record Fields

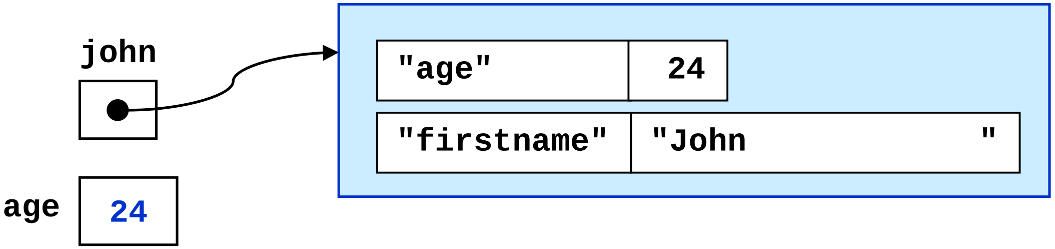


```
PROGRAM RecordTest2;

TYPE
  String16 = ARRAY [1..16]
               OF char;

  PersonRec = RECORD
                firstName
                  : String16;
                age
                  : integer;
              END;

VAR
  john : PersonRec;
  age  : integer;

BEGIN
  john.age := 24;
  john.firstName := 'John';
  age := john.age;
END.
```

```
getstatic       recordtest2/john Ljava/util/HashMap;
ldc             "age"
bipush          24
invokestatic    java/lang/Integer.valueOf(I)Ljava/lang/Integer;
invokevirtual   java/util/HashMap.put(Ljava/lang/Object;
                                      Ljava/lang/Object;)Ljava/lang/Object;
pop

getstatic       recordtest2/john Ljava/util/HashMap;
ldc             "firstname"
invokevirtual   java/util/HashMap.get(Ljava/lang/Object;)Ljava/lang/Object;
checkcast       java/lang/StringBuilder
dup
iconst_0
invokevirtual   java/lang/StringBuilder.setLength(I)V
ldc             "John"
invokevirtual   java/lang/StringBuilder.append(
                             Ljava/lang/String;)Ljava/lang/StringBuilder;
bipush          16
iconst_4
invokestatic    PaddedString.blanks(II)Ljava/lang/StringBuilder;
invokevirtual   java/lang/StringBuilder.append(
                        Ljava/lang/CharSequence;)Ljava/lang/StringBuilder;
pop
```

Access Values of Record Fields

```
PROGRAM RecordTest2;

TYPE
  String16 = ARRAY [1..16]
               OF char;

  PersonRec = RECORD
                firstName
                  : String16;
                age
                  : integer;
              END;

VAR
  john : PersonRec;
  age  : integer;

BEGIN
  john.age := 24;
  john.firstName := 'John';
  age := john.age;
END.
```



```
getstatic      recordtest2/john Ljava/util/HashMap;
ldc            "age"
invokevirtual  java/util/HashMap.get(Ljava/lang/Object;)
                                       Ljava/lang/Object;
checkcast      java/lang/Integer
invokevirtual  java/lang/Integer.intValue()I
putstatic      recordtest2/age I
```