Motivating Arrays

```
final String
      karen = "Karen Smith",
      john = "John Duncan",
      paul = "Paul Jacobs",
      suzanne = "Suzanne Enders",
      peter = "Peter Phillips";  // 10 more to come ...

    System.out.println(karen);
    System.out.println(john);
...
```

Per member repeating tasks

- Generate Comma separated list:

```
Karen Smith, John Duncan, Paul Jacobs, Suzanne Enders, Peter Phillips
```

- Generate HTML list emphasizing family names:

```
<ul>
  <li>Karen <emph>Smith</emph></li>
  <li>John <emph>Duncan</emph></li>
  <li>Paul <emph>Jacobs</emph></li>
  <li>Suzanne <emph>Enders</emph></li>
  <li>Peter <emph>Phillips</emph></li>
</ul>
```

Example: `int` array of primes

```
final int[] primes ❶ = new int[5]; ❷

primes[0] = 2; ❸
primes[1] = 3;
primes[2] = 5;
primes[3] = 7;
primes[4] = 11;
```

Related exercises

Exercise 137: Assignment to final variable?

Loop prime values

```
for (int i = 0; i < 5; i++) {
  System.out.println("At index " + i + ": value == " + primes[i]);
}
```

Result:

```
At index 0: value == 2
At index 1: value == 3
At index 2: value == 5
At index 3: value == 7
At index 4: value == 11
```

Mind the limit!

```
for (int i = 0; i < 6; i++) {
  System.out.println("At index " + i + ": value == " + primes[i]);
}
```

Result:

```
At index 0: value == 2
At index 1: value == 3
At index 2: value == 5
At index 3: value == 7
At index 4: value == 11
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
  at qq.arrayex.Motivate.main(Motivate.java:27)
```

Safer: Using *length*

```
System.out.println("primes.length == " + primes.length);
for (int i = 0; i < primes.length; i++) {
  System.out.println("At index " + i + ": value == " + primes[i]);
}
```

Result:

```
primes.length == 5
At index 0: value == 2
At index 1: value == 3
At index 2: value == 5
At index 3: value == 7
At index 4: value == 11
```

Even better: “for-each” style loop

```
for (final int p: primes) {
  System.out.println("value == " + p);
}
```

Result:

```
value == 2
value == 3
value == 5
value == 7
value == 11
```

Mind the limit, part two

```
final int[] primes = new int[5]; // Last index is 4 rather than 5!

primes[0] = 2;
primes[1] = 3;
primes[2] = 5;
primes[3] = 7;
primes[4] = 11;
primes[5] = 13; // Excessing array limit
```

Result:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
  at qq.arrayex.Motivate.main(Motivate.java:25)
```

Primitive data one step initialization

Combining array allocation and value assignment:

```
final int[]
  primes = {2, 3, 5, 7, 11};
```

```
final int[] primes = new int[5];
primes[0] = 2;
primes[1] = 3;
primes[2] = 5;
primes[3] = 7;
primes[4] = 11;
```

Reference data one step initialization

Combining array allocation and value assignment:

```
public class Rectangle {
    private int width, height;
    private boolean hasSolidBorder;
    public Rectangle(int width, int height,
                     boolean hasSolidBorder) {
        this.width = width;
        this.height = height;
        this.hasSolidBorder = hasSolidBorder;
    }
}
```

```
final Rectangle[] rectList = new Rectangle[] {
  new Rectangle(2, 5, true),
  new Rectangle(4, 1, false)
};
```

Related exercises

Exercise 138: Converting string arrays to HTML.
Exercise 139: Route navigation
Exercise 140: Examinations and mark frequencies
Exercise 141: Pangram checker

Array

- Series of objects having identical type.
- Array consists of array elements.
- Element access by index value.
- Holding either primitive types or object references (Class instance or array).
- Contiguous storage in memory.
- Arbitrary array dimensions by virtue of nesting: One-dimensional, two-dimensional etc.

Two syntax variants

1. `type[] arrayName; // preferred`
2. `type arrayName[];`

Array instances are special!

```
...println("        String: " + "".getClass().getTypeName());
...println("         int[]: " + new int[]{}.getClass().getTypeName());
...println("      double[]: " + new double[]{}.getClass().getTypeName());
...println("     boolean[]: " + new boolean[]{}.getClass().getTypeName());
...println("StringBuffer[]: " + new StringBuffer[]{}.getClass().getTypeName());
```

```
        String: java.lang.String
         int[]: int[]
      double[]: double[]
     boolean[]: boolean[]
      String[]: java.lang.String[]
StringBuffer[]: java.lang.StringBuffer[]
```

Array creation details

```
final String[] shapes
  = new String[]{
   new String("Triangle"),
   new String("Circle")
  };
```

Stack **Heap**

Create String instance »Triangle«

Array creation details

```
final String[] shapes
  = new String[]{
   new String("Triangle"),
   new String("Circle")
  };
```

Stack

Heap

Triangle

Create String instance »Circle«

Array creation details

```
final String[] shapes
  = new String[]{
   new String("Triangle"),
   new String("Circle")
  };
```

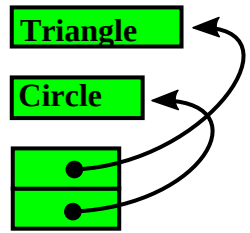
Create array containing two String references.

Array creation details

```
final String[] shapes
  = new String[]{
   new String("Triangle"),
   new String("Circle")
  };
```

Stack

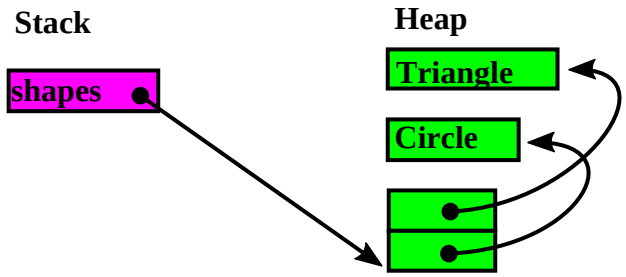
Heap



Create local variable shapes assigning reference to array.

Array creation details

```
final String[] shapes
  = new String[]{
   new String("Triangle"),
   new String("Circle")
  };
```



Array parameter passing

```
public static void main(String[] args) {
  final int [] lectures = new int[3]; // Three lectures
  fill(lectures, 25); // Default lecture having 25 participants
  System.out.println("Second lecture has got " + lectures[1] +
          " participants");
}
/**
  * Initialize array with default value.
  *
  * @param values Array to be initialized.
  * @param common New value for all array elements.
  */
static void fill(final int[] values, final int common) {
  for (int i = 0; i < values.length; i++) {
    values[i] = common;
  }
}
```

```
Second lecture has got 25 participants
```

Parameter passing details

```
...main(...) {
  int [] lectures =
    new int[3];
  fill(lectures, 25);
  ...println("Second ..."
   + lectures[1] +
    " participants");}

..fill(final int[] values,
  final int common) {
  for (int i = 0;
   i < values.length; i++) {
    values[i] = common;
  }}
```

Stack

Heap

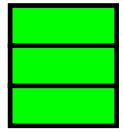
Create int array of size 3.

Parameter passing details

```
...main(...) {
  int [] lectures =
    new int[3];
  fill(lectures, 25);
  ...println("Second ..."
   + lectures[1] +
    " participants");}

..fill(final int[] values,
  final int common) {
  for (int i = 0;
   i < values.length; i++) {
    values[i] = common;
  }}
```

Stack

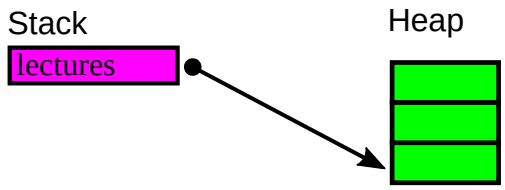
Heap



Assign array reference to variable lectures.

Parameter passing details

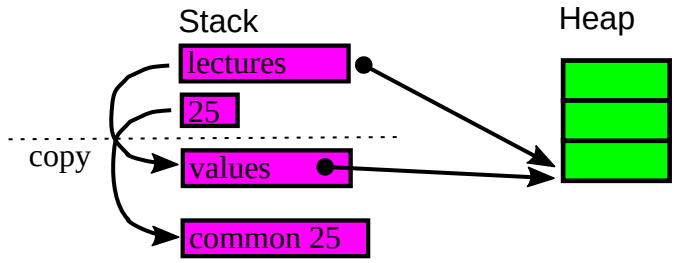
```
...main(...) {
  int [] lectures =
    new int[3];
  fill(lectures, 25);
  ...println("Second ..."
   + lectures[1] +
    " participants");}

..fill(final int[] values,
  final int common) {
  for (int i = 0;
   i < values.length; i++) {
    values[i] = common;
  }}
```



»Call By Value« of fill(...) method passing both array reference lectures and primitive int 25.

Parameter passing details

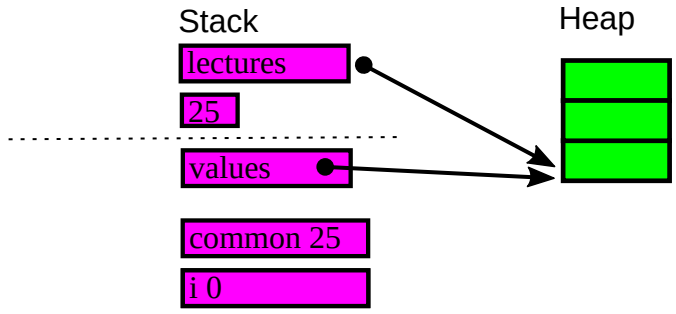
```
...main(...) {
  int [] lectures =
    new int[3];
  fill(lectures, 25);
  ...println("Second ..."
   + lectures[1] +
    " participants");}

..fill(final int[] values,
  final int common) {
  for (int i = 0;
   i < values.length; i++) {
    values[i] = common;
  }}
```



Starting for loop. New local variable i on stack.

Parameter passing details

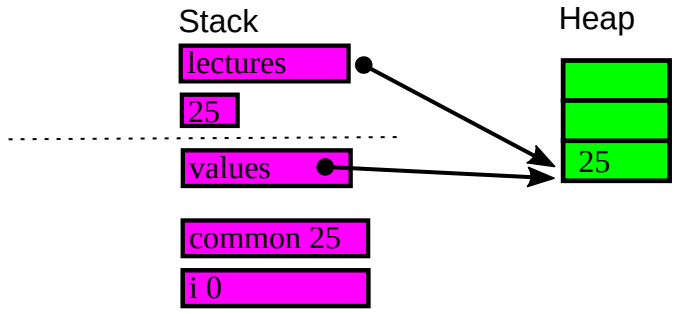
```
...main(...) {
  int [] lectures =
    new int[3];
  fill(lectures, 25);
  ...println("Second ..."
   + lectures[1] +
    " participants");}

..fill(final int[] values,
  final int common) {
  for (int i = 0;
   i < values.length; i++) {
    values[i] = common;
  }}
```



Init first array element to common value.

Parameter passing details

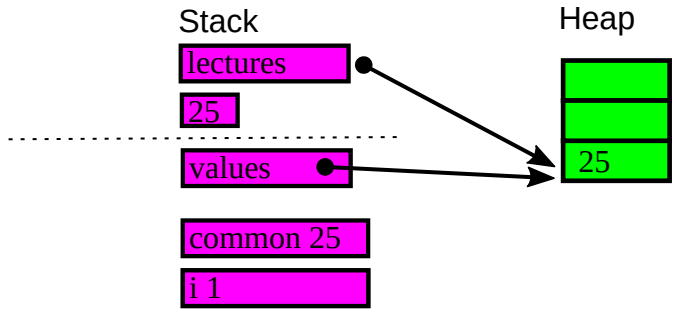
```
...main(...) {
  int [] lectures =
    new int[3];
  fill(lectures, 25);
  ...println("Second ..."
   + lectures[1] +
    " participants");}

..fill(final int[] values,
  final int common) {
  for (int i = 0;
   i < values.length; i++) {
    values[i] = common;
  }}
```



Loop variable increment.

Parameter passing details

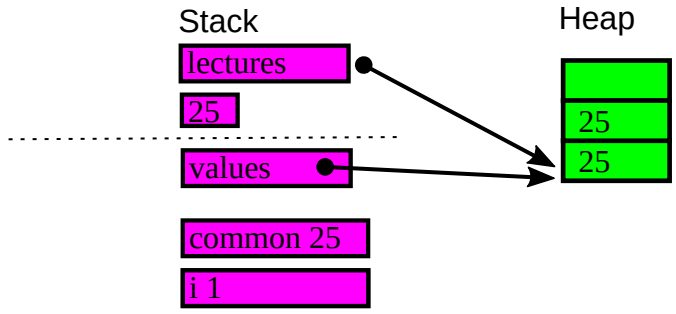
```
...main(...) {
  int [] lectures =
    new int[3];
  fill(lectures, 25);
  ...println("Second ..."
   + lectures[1] +
    " participants");}

..fill(final int[] values,
  final int common) {
  for (int i = 0;
   i < values.length; i++) {
    values[i] = common;
  }}
```



Init second array element to common value.

Parameter passing details

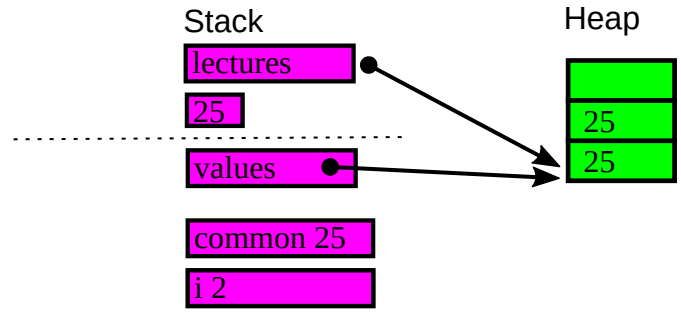
```
...main(...) {
  int [] lectures =
    new int[3];
  fill(lectures, 25);
  ...println("Second ..."
   + lectures[1] +
    " participants");}

..fill(final int[] values,
  final int common) {
  for (int i = 0;
   i < values.length; i++) {
    values[i] = common;
  }}
```



Loop variable increment.

Parameter passing details

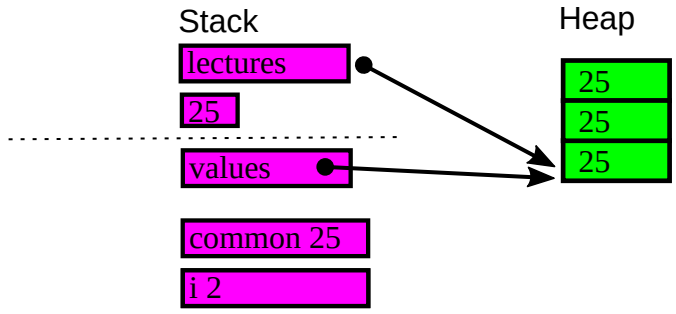
```
...main(...) {
  int [] lectures =
    new int[3];
  fill(lectures, 25);
  ...println("Second ..."
   + lectures[1] +
    " participants");}

..fill(final int[] values,
  final int common) {
  for (int i = 0;
   i < values.length; i++) {
    values[i] = common;
  }}
```



Init third array element to common value.

Parameter passing details

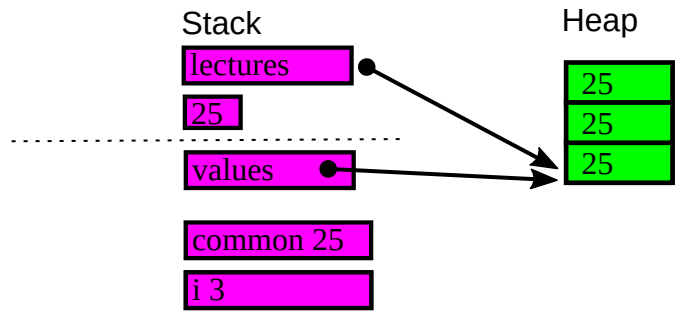
```
...main(...) {
  int [] lectures =
    new int[3];
  fill(lectures, 25);
  ...println("Second ..."
   + lectures[1] +
    " participants");}

..fill(final int[] values,
  final int common) {
  for (int i = 0;
   i < values.length; i++) {
    values[i] = common;
  }}
```



Loop variable increment.

Parameter passing details

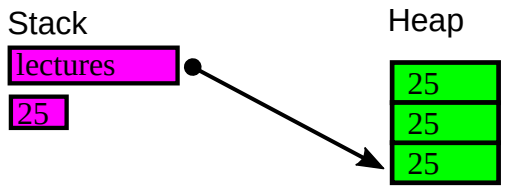
```
...main(...) {
  int [] lectures =
    new int[3];
  fill(lectures, 25);
  ...println("Second ..."
   + lectures[1] +
    " participants");}

..fill(final int[] values,
  final int common) {
  for (int i = 0;
   i < values.length; i++) {
    values[i] = common;
  }}
```



Loop termination, return to main(...)

Parameter passing details

```
...main(...) {
  int [] lectures =
    new int[3];
  fill(lectures, 25);
  ...println("Second ..."
   + lectures[1] +
    " participants");}

..fill(final int[] values,
  final int common) {
  for (int i = 0;
   i < values.length; i++) {
    values[i] = common;
  }}
```



Print second array element.

Value and reference types

```
// Value type
final boolean values[] = new boolean[]{true, true, false, true};

// Reference type
final String shapes[] = new String[]{"Triangle", "Circle"};
```

Same result:

```
final boolean values[] = {true, true, false, true};

final String shapes[] = {"Triangle", "Circle"};
```

Related exercises

Exercise 142: Reconsidering System.out.format().

Arrays.toString(...) and Arrays.sort(...)

```
final String[] names = {"Eve", "Aaron", "Paul", "Mandy"};

System.out.println("     toString: " + Arrays.toString(names));

Arrays.sort(names);

System.out.println("sort|toString: " + Arrays.toString(names));
```

Result:

```
     toString: [Eve, Aaron, Paul, Mandy]
sort|toString: [Aaron, Eve, Mandy, Paul]
```

Arrays.binarySearch(...)

```
final String[] names = {"Aaron", "Eve", "Mandy",  "Paul"};

// Precondition: Array must be ordered!
...println("sort|find(Mand): " + Arrays.binarySearch(names, "Mand"));
...println("sort|find(Mandy): " + Arrays.binarySearch(names, "Mandy"));
...println("sort|find(Mandyer): " + Arrays.binarySearch(names, "Mandyer"));
```

Result:

```
sort|find(Mand): -3
sort|find(Mandy): 2
sort|find(Mandyer): -4
```

Related exercises

Exercise 143: Understanding search results

Arrays.fill(...)

```
final String[] names =
  {"Eve", "Aaron", "Paul", "Mandy"};

System.out.println("toString: " +
   Arrays.toString(names));

Arrays.fill(names, "N.N");

System.out.println("toString: " +
    Arrays.toString(names));
```

```
toString: [Eve, Aaron, Paul, Mandy]
toString: [N.N, N.N, N.N, N.N]
```

Arrays.copyOfRange(...)

```
final String[] names = {"Eve", "Aaron", "Paul", "Mandy"};

final String[] lastTwoNames = Arrays.copyOfRange(names, 2, 6);

System.out.println("toString: " + Arrays.toString(lastTwoNames));
```

Result:

```
toString: [Paul, Mandy, null, null]
```

Arrays.equals(...)

```
final String[]
  l1 = {new String("Eve"), new String("Aaron"),
        new String("Paul"), new String("Mandy")},

  l2 = {new String("Eve"), new String("Aaron"),
        new String("Paul"), new String("Mandy")},

  l3 = {new String("Eve"), new String("Aaron"),
        new String("Paul"), new String("Mobile")};

System.out.println("l1.equals(l2):" + Arrays.equals(l1, l2));
System.out.println("l1.equals(l3):" + Arrays.equals(l1, l3));
```

Result:

```
l1.equals(l2):true
l1.equals(l3):false
```

Lack of extendability

```
final String[] member = {"Eve", "John", "Peter", "Jill"};

final String newCourseMember = "Ernest";

member.length = 5; // Error: Size unchangeable

member[4] =  newCourseMember;
```

Extending an array

```
public static void main(String[] args) {
 ❶ String[] member = {"Eve", "John", "Peter", "Jill"};
  final String newMember = "Ernest";
  member ❷= append(member, newMember);
}
static String[] append (final String[] values, final String newValue) {
  final String[] copy = ❸ new String[values.length + 1];
  for (int i = 0; i < values.length; i++) { ❹
    copy[i] = values[i]; ❺
  }
  copy[copy.length - 1] = newValue; ❻
  return copy;
}
```

Extension result

```
final String[] member = {"Eve", "John", "Peter", "Jill"};
System.out.println("Original array: " + Arrays.toString(member));
final String newMember = "Ernest";
member = append(member, newMember);
System.out.println("Extended array: " + Arrays.toString(member));

Original array: [Eve, John, Peter, Jill]
Extended array: [Eve, John, Peter, Jill, Ernest]
```

Using `Arrays.copyOf()`

```
public static void main(String[] args) {
    final int [] start = {1,7,-4},
    added = append(start, 77);
    System.out.println("added: " + Arrays.toString(added));
  }
static public int[] append(final int[] values, final int newValue) {
  final int[] result = Arrays.copyOf(values, values.length + 1);
  result[values.length] = newValue;
  return result;}
```

Result:

```
added: [1, 7, -4, 77]
```

Related exercises

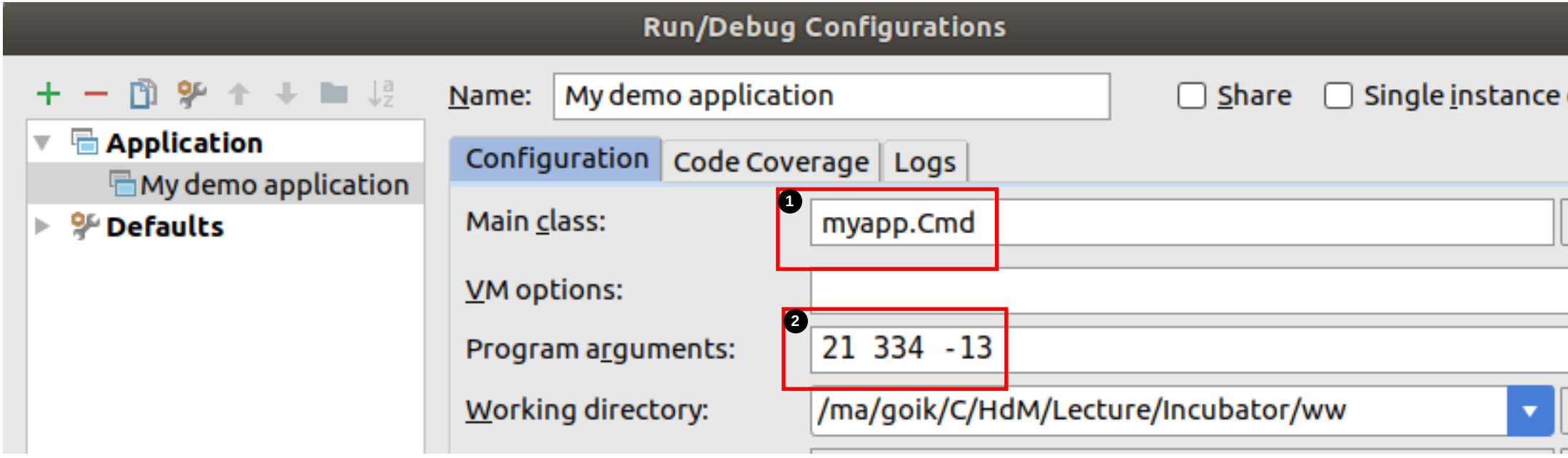
Exercise 144: Implementing append directly
Exercise 145: Purge duplicates
Exercise 146: A container of fixed capacity holding integer values
Exercise 147: Allow for variable capacity holding integer values

public static void main(String[] args)

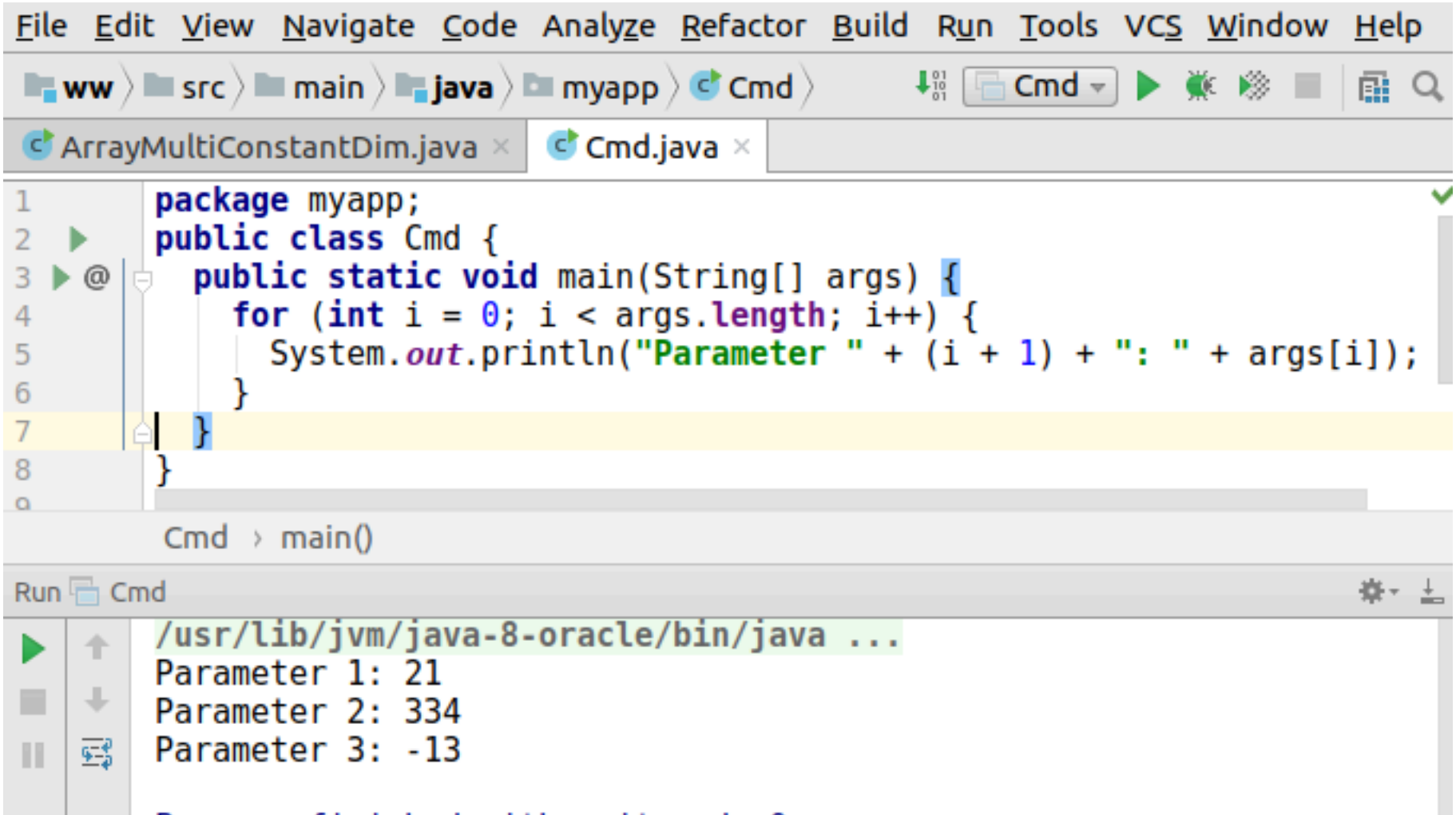
```
package myapp;
public class Cmd {
  public static void main(String[] args) {
    for (int i = 0; i < args.length; i++) {
      System.out.println("Parameter " + (i + 1) + ": " + args[i]);
    }
  }
}
```

```
java myapp.Cmd 21 334 -13
Parameter 1: 21
Parameter 2: 334
Parameter 3: -13
```

Intellij IDEA run configuration



Intellij IDEA run configuration



Creating executable jar

```
<artifactId>maven-shade-plugin</artifactId>
...
<transformer ...>
  <manifestEntries>
    <Main-Class>myapp.Cmd</Main-Class>
  </manifestEntries>...

mvn package ...
java -jar target/ww-1.0-SNAPSHOT.jar 21 334 -13
Parameter 1: 21
Parameter 2: 334
Parameter 3: -13
```

```
unzip ww-1.0-SNAPSHOT.jar
cat  tmp/META-INF/MANIFEST.MF
...
Created-By: Apache Maven 3.5.0
Build-Jdk: 1.8.0_151
Main-Class: myapp.Cmd
```

Related exercises

Exercise 148: Reading console input
Exercise 149: Prettifying output representation

Two-dimensional arrays

```
final int[][] matrix = new int[2][3];

for (int row = 0; row < 2; row++) {
  for (int col = 0; col < 3; col++) {
    matrix[row][col] = col + row;
  }
}
for (int row = 0; row < 2; row++) {
  System.out.println(Arrays.toString(matrix[row]));
}
```

Behind the scenes

```
final int[][] matrix = new int[2][]; // Array containing two int arrays
matrix[0] = new int[3];              // first int array
matrix[1] = new int[3];              // second int array
```

Memory allocation

Stack Heap

```
int[][] matrix = new int[2][];

matrix[0] = new int[3];
matrix[1] = new int[3];

for (int row = 0; row < 2; row++){
  for (int col = 0; col < 3; col++){
    matrix[row][col] = col + row;
  }
}
```

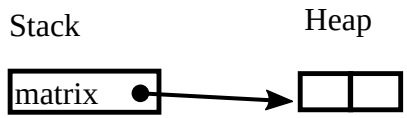
Memory allocation

Stack

Heap

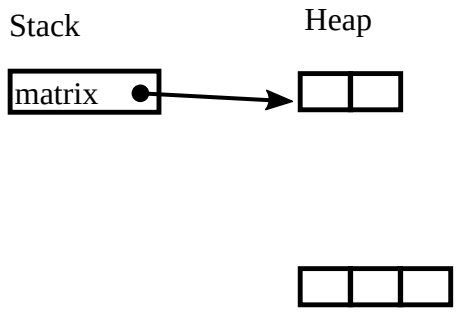
```
int[][] matrix = new int[2][];

matrix[0] = new int[3];
matrix[1] = new int[3];

for (int row = 0; row < 2; row++){
  for (int col = 0; col < 3; col++){
    matrix[row][col] = col + row;
  }
}
```

Memory allocation




```
int[][] matrix = new int[2][];

matrix[0] = new int[3];
matrix[1] = new int[3];

for (int row = 0; row < 2; row++){
  for (int col = 0; col < 3; col++){
    matrix[row][col] = col + row;
  }
}
```

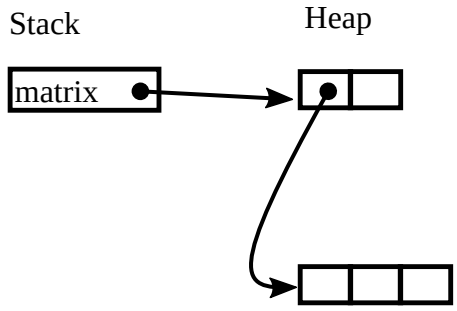
Memory allocation




```
int[][] matrix = new int[2][];

matrix[0] = new int[3];
matrix[1] = new int[3];

for (int row = 0; row < 2; row++){
  for (int col = 0; col < 3; col++){
    matrix[row][col] = col + row;
  }
}
```

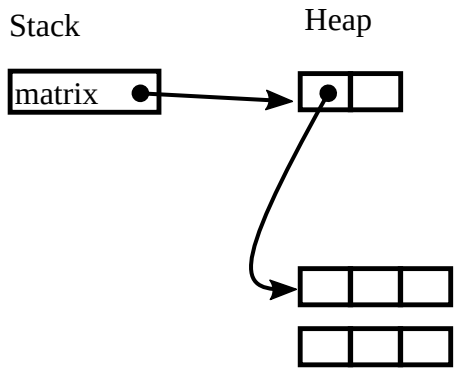
Memory allocation

```
int[][] matrix = new int[2][];

matrix[0] = new int[3];
matrix[1] = new int[3];

for (int row = 0; row < 2; row++){
  for (int col = 0; col < 3; col++){
    matrix[row][col] = col + row;
  }
}
```



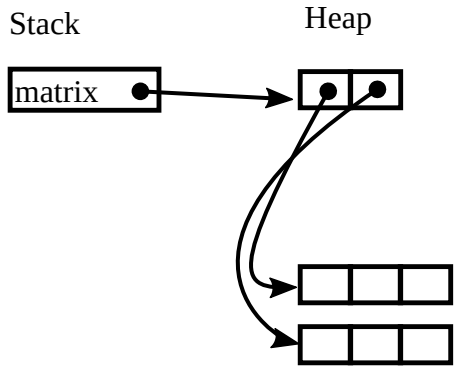
Memory allocation

```
int[][] matrix = new int[2][];

matrix[0] = new int[3];
matrix[1] = new int[3];

for (int row = 0; row < 2; row++){
  for (int col = 0; col < 3; col++){
    matrix[row][col] = col + row;
  }
}
```



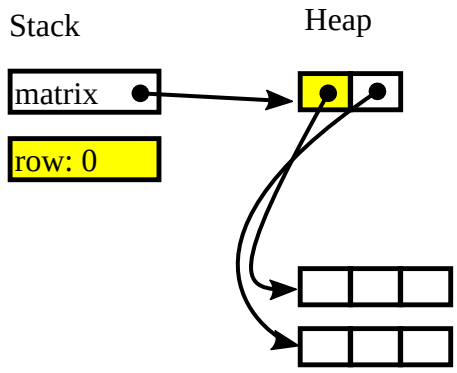
Memory allocation

```
int[][] matrix = new int[2][];

matrix[0] = new int[3];
matrix[1] = new int[3];

for (int row = 0; row < 2; row++){
  for (int col = 0; col < 3; col++){
    matrix[row][col] = col + row;
  }
}
```



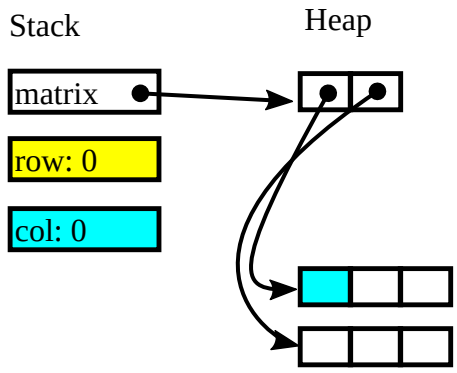
Memory allocation

```
int[][] matrix = new int[2][];

matrix[0] = new int[3];
matrix[1] = new int[3];

for (int row = 0; row < 2; row++){
  for (int col = 0; col < 3; col++){
    matrix[row][col] = col + row;
  }
}
```



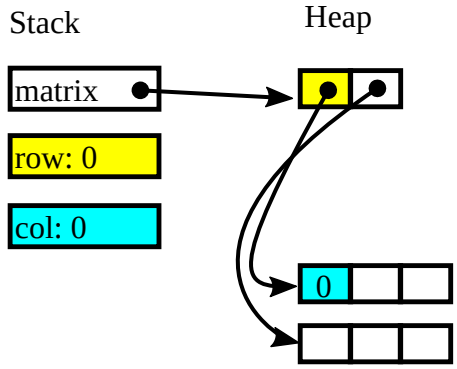
Memory allocation

```
int[][] matrix = new int[2][];

matrix[0] = new int[3];
matrix[1] = new int[3];

for (int row = 0; row < 2; row++){
  for (int col = 0; col < 3; col++){
    matrix[row][col] = col + row;
  }
}
```



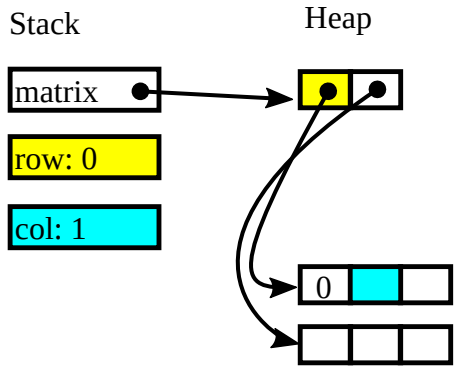
Memory allocation

```
int[][] matrix = new int[2][];

matrix[0] = new int[3];
matrix[1] = new int[3];

for (int row = 0; row < 2; row++){
  for (int col = 0; col < 3; col++){
    matrix[row][col] = col + row;
  }
}
```



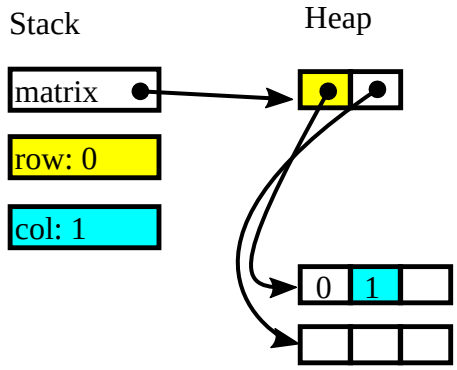
Memory allocation

```
int[][] matrix = new int[2][];

matrix[0] = new int[3];
matrix[1] = new int[3];

for (int row = 0; row < 2; row++){
  for (int col = 0; col < 3; col++){
    matrix[row][col] = col + row;
  }
}
```



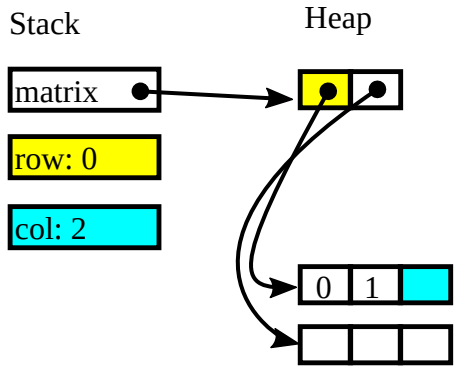
Memory allocation

```
int[][] matrix = new int[2][];

matrix[0] = new int[3];
matrix[1] = new int[3];

for (int row = 0; row < 2; row++){
  for (int col = 0; col < 3; col++){
    matrix[row][col] = col + row;
  }
}
```



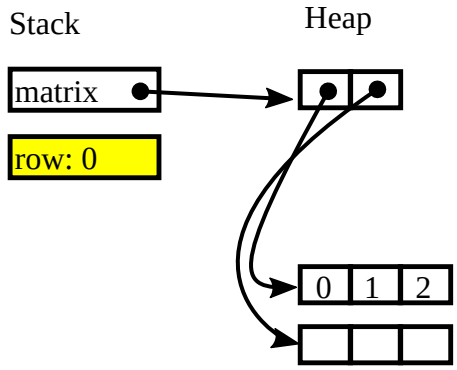
Memory allocation

```
int[][] matrix = new int[2][];

matrix[0] = new int[3];
matrix[1] = new int[3];

for (int row = 0; row < 2; row++){
  for (int col = 0; col < 3; col++){
    matrix[row][col] = col + row;
  }
}
```



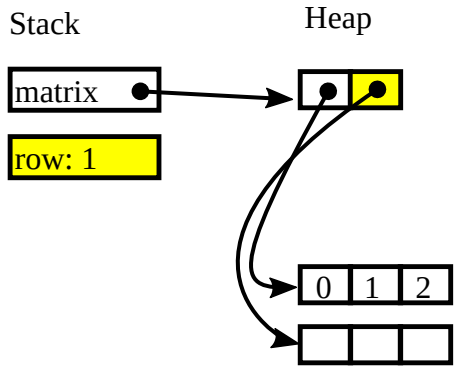
Memory allocation

```
int[][] matrix = new int[2][];

matrix[0] = new int[3];
matrix[1] = new int[3];

for (int row = 0; row < 2; row++){
  for (int col = 0; col < 3; col++){
    matrix[row][col] = col + row;
  }
}
```



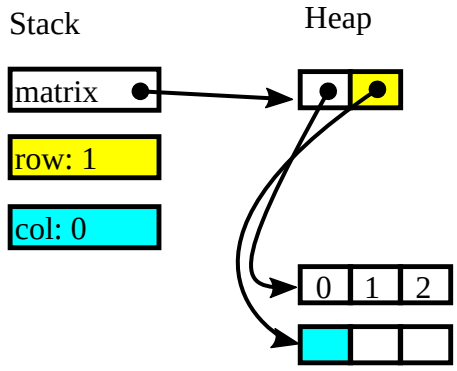
Memory allocation

```
int[][] matrix = new int[2][];

matrix[0] = new int[3];
matrix[1] = new int[3];

for (int row = 0; row < 2; row++){
  for (int col = 0; col < 3; col++){
    matrix[row][col] = col + row;
  }
}
```



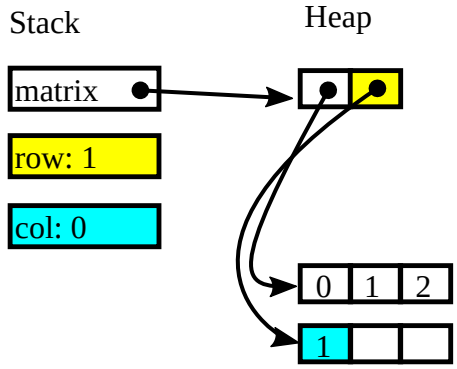
Memory allocation

```
int[][] matrix = new int[2][];

matrix[0] = new int[3];
matrix[1] = new int[3];

for (int row = 0; row < 2; row++){
  for (int col = 0; col < 3; col++){
    matrix[row][col] = col + row;
  }
}
```



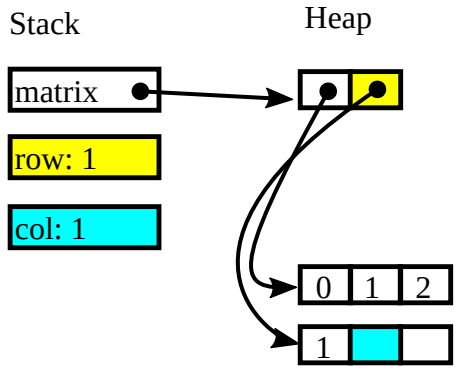
Memory allocation

```
int[][] matrix = new int[2][];

matrix[0] = new int[3];
matrix[1] = new int[3];

for (int row = 0; row < 2; row++){
  for (int col = 0; col < 3; col++){
    matrix[row][col] = col + row;
  }
}
```



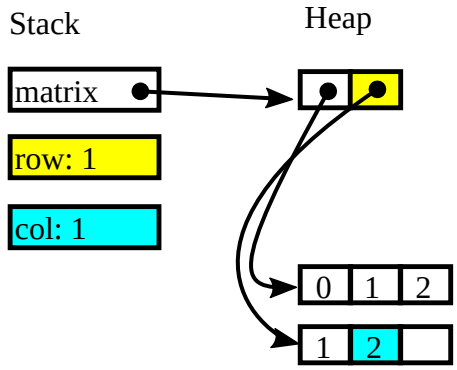
Memory allocation

```
int[][] matrix = new int[2][];

matrix[0] = new int[3];
matrix[1] = new int[3];

for (int row = 0; row < 2; row++){
  for (int col = 0; col < 3; col++){
    matrix[row][col] = col + row;
  }
}
```



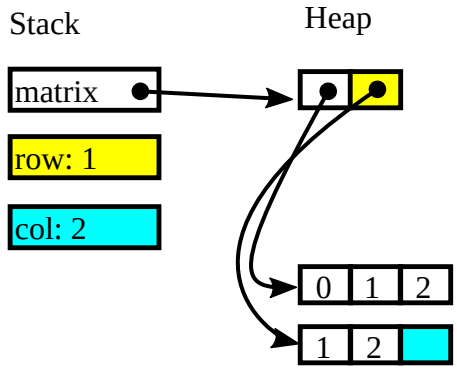
Memory allocation

```
int[][] matrix = new int[2][];

matrix[0] = new int[3];
matrix[1] = new int[3];

for (int row = 0; row < 2; row++){
  for (int col = 0; col < 3; col++){
    matrix[row][col] = col + row;
  }
}
```



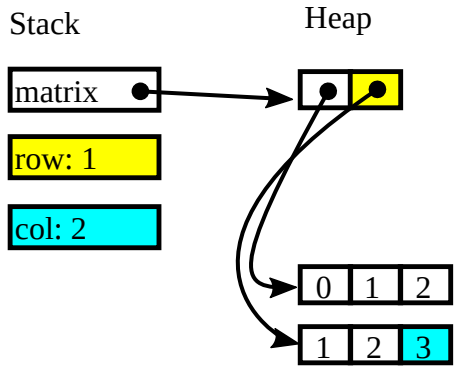
Memory allocation

```
int[][] matrix = new int[2][];

matrix[0] = new int[3];
matrix[1] = new int[3];

for (int row = 0; row < 2; row++){
  for (int col = 0; col < 3; col++){
    matrix[row][col] = col + row;
  }
}
```



Memory allocation

```
int[][] matrix = new int[2][];

matrix[0] = new int[3];
matrix[1] = new int[3];

for (int row = 0; row < 2; row++){
  for (int col = 0; col < 3; col++){
    matrix[row][col] = col + row;
  }
}
```



Related exercises

Exercise 150: 2-dimensional arrays and `.length`

Static array initialization

```
final int[][] matrix = new int[][] {
  {0, 1, 2},
  {1, 2, 3}
};
```

Static array initialization, variable lengths

```
final String[][] groups = new String[][] {
  {"Jill", "Tom"},
  {"Jane", "Smith", "Joe"},
  {"Jeff"}
};

for (int row = 0; row < groups.length; row++) {
  System.out.println(Arrays.toString(groups[row]));
}
```

```
[Jill, Tom]
[Jane, Smith, Joe]
[Jeff]
```

Related exercises

Exercise 151: External array and string exercises
Exercise 152: Tic-tac-toe using a two-dimensional array
Exercise 153: Changing the game's internal representation
Exercise 154: Tic-tac-toe, Computer vs. human
Exercise 155: Adding support to retrieve statistical data.
Exercise 156: Testing an implementation
Exercise 157: Improving prime number calculation performance
Exercise 158: Calculating the median
Exercise 159: A simple character based plotting application