Session 06: Arrays

Dr Tomasz Kurowski
t.j.kurowski@cranfield.ac.uk

Lecture outline

Introducing Arrays

Declaring Array Variables, Creating Arrays, and Initialising Arrays

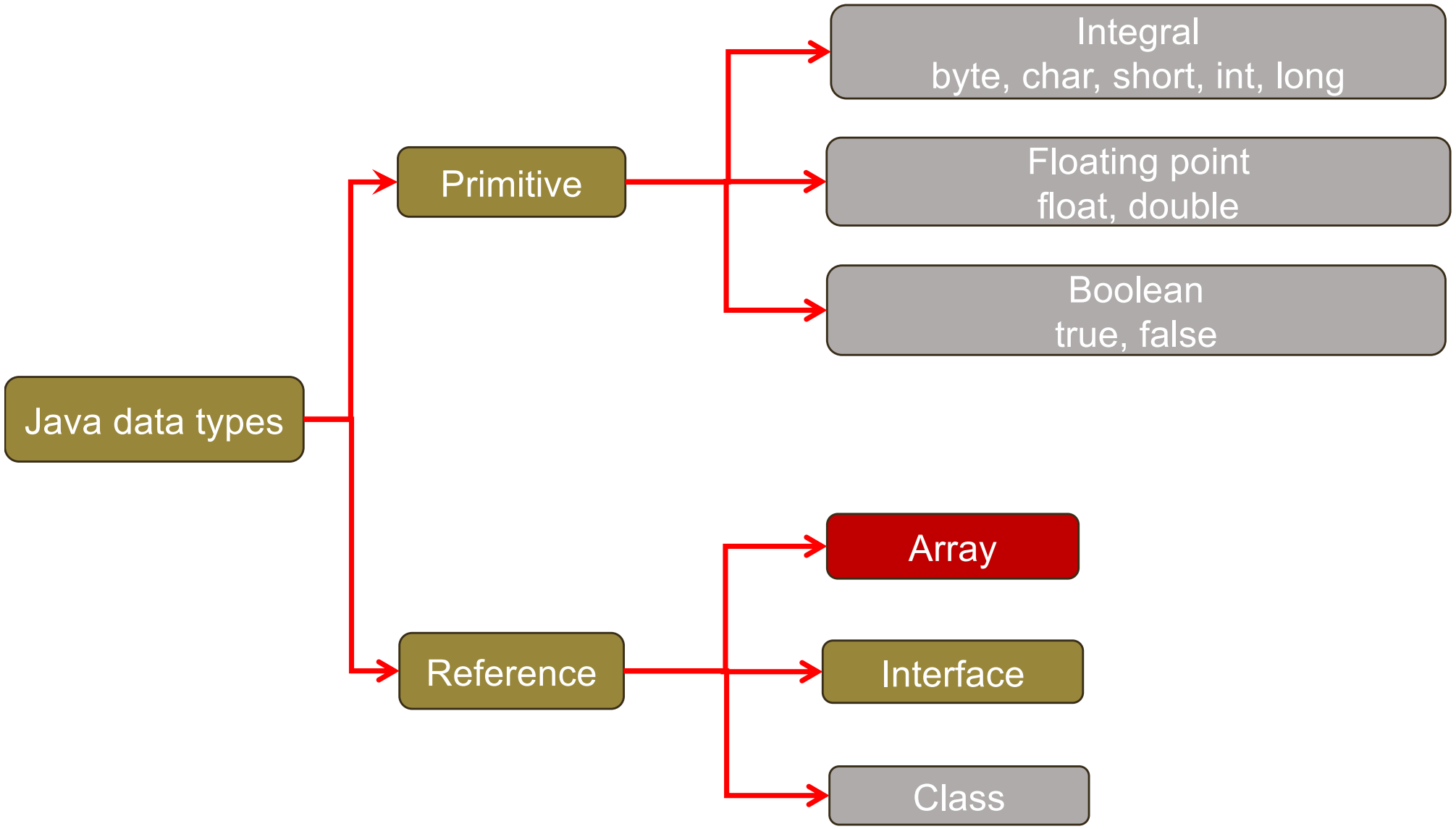
Passing Arrays to Methods

Copying Arrays

Multidimensional Arrays

Search and Sorting Methods

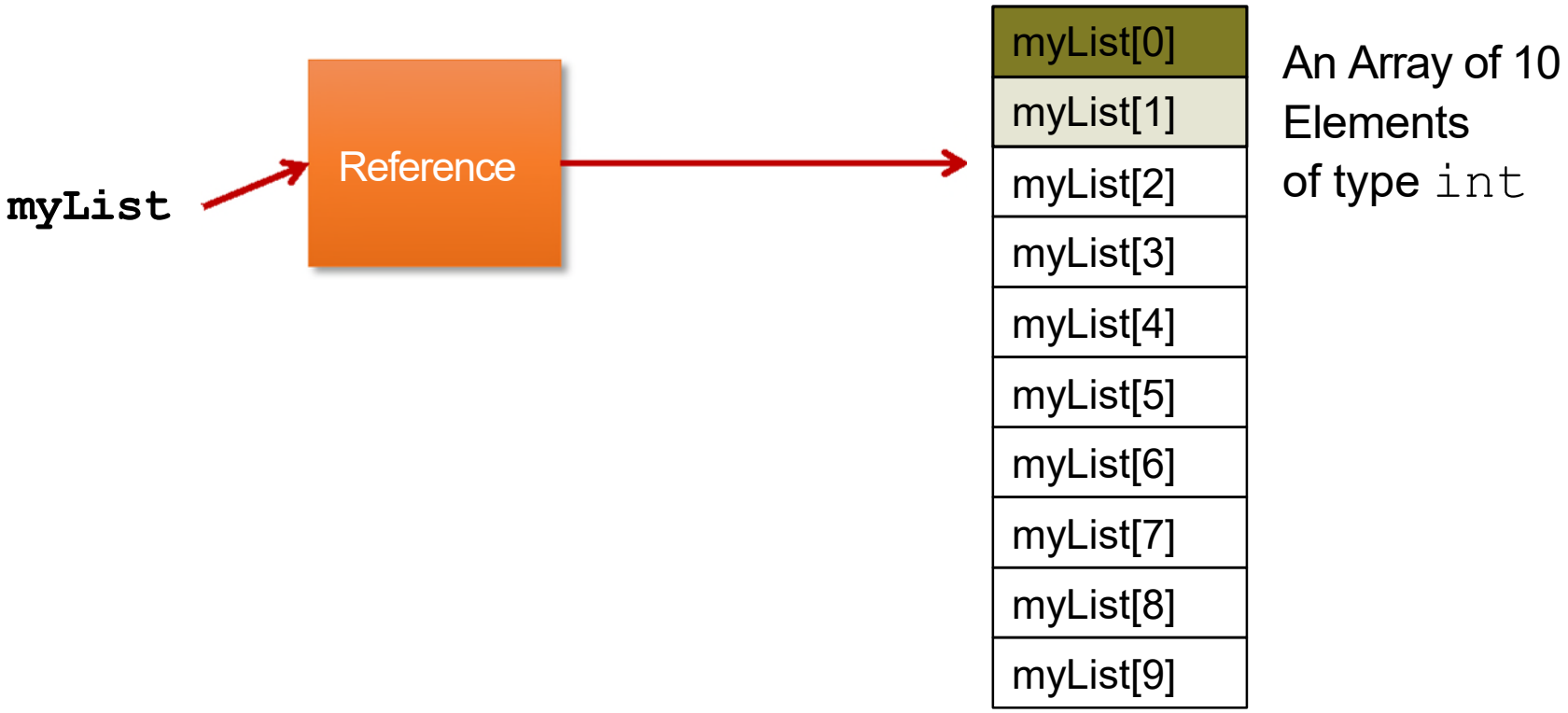
Java data types



Introducing Arrays

An array is a data structure that represents a collection of the same types of data.

```
int[] data = new int(datasize);
int[] myList = new int(10);
```



Declaring Array Variables

```
datatype[] arrayname;
```

Example: `double[] myList;`

These declarations do not create room in the data memory for the actual array elements at this stage.

It just creates a reference to a new data type (an array of doubles). Note that no array size is defined here.

Space for the elements is allocated in the next step …

Creating Arrays

```
arrayName = new datatype[arraySize];
```

Example:

```
myList = new double[10];
```

`myList[0]` references the first element in the array.

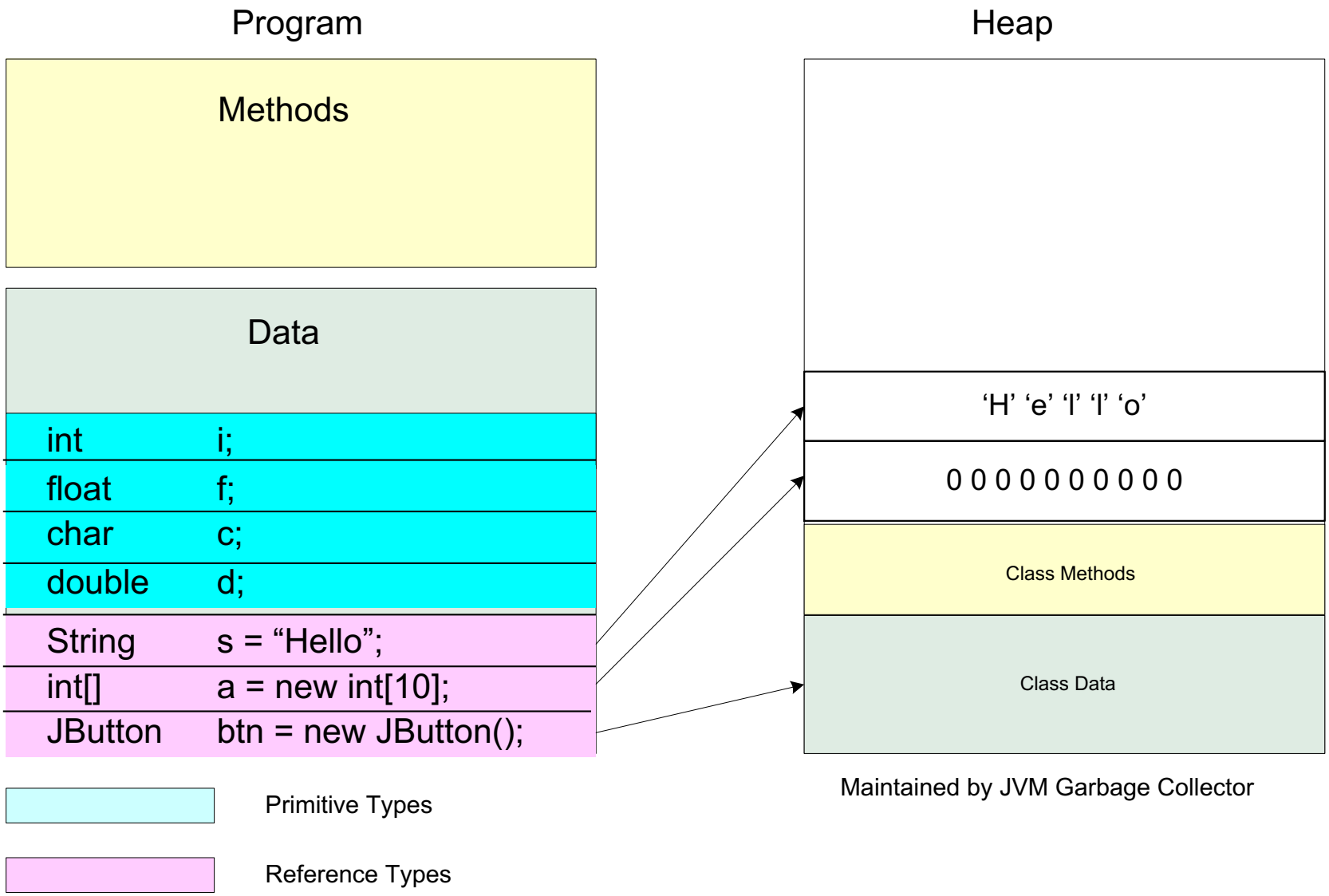
`myList[9]` references the last element in the array.

The first element is numbered as 0

This introduces the concept of dynamic storage : Physical Memory can be reused by other variables when the old one is no longer needed.

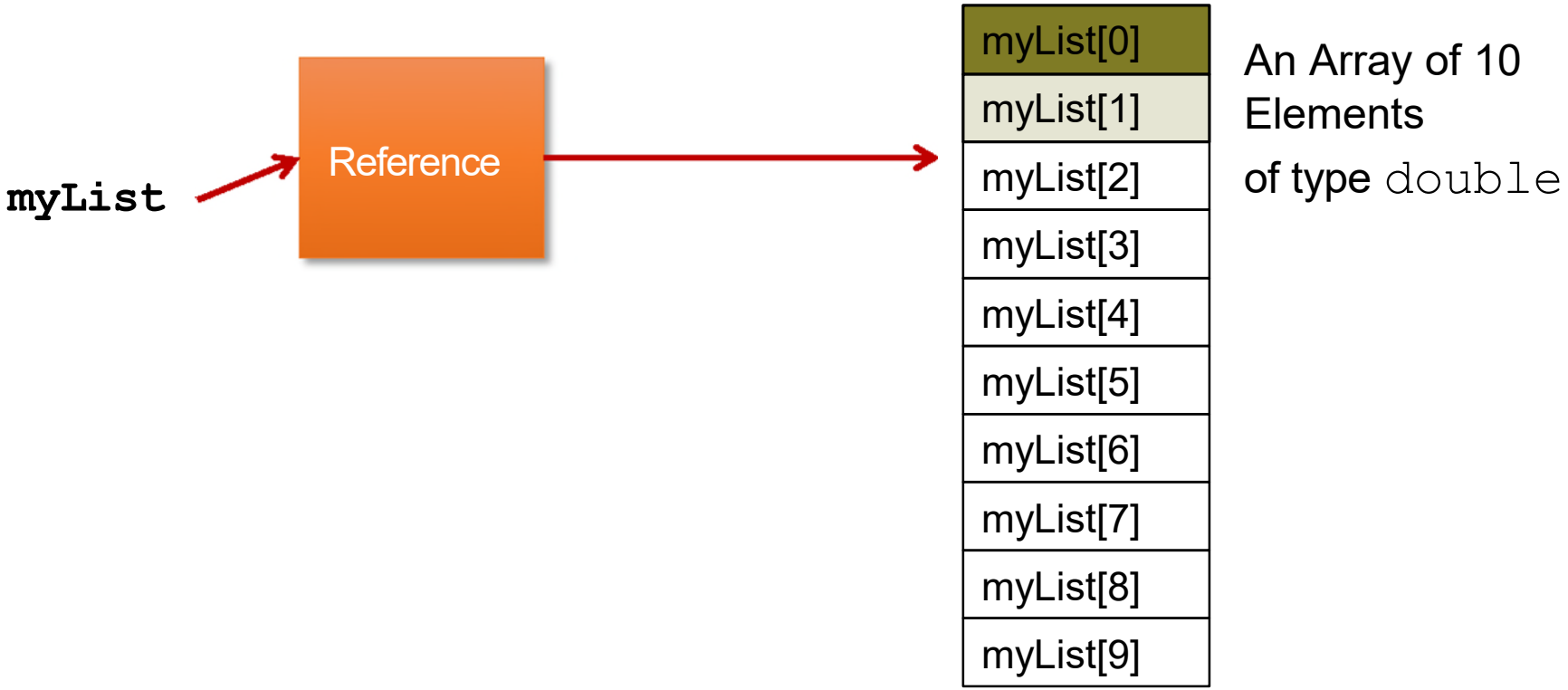
This memory is called the ‘heap’

Data and Heap Memory



Declaring and Creating in One Step

```
datatype[] arrayName = new datatype[arraySize];

double[] myList = new double[10];
```



The Length of Arrays

- Once an array is created, its size is fixed.
- It cannot be changed – space on the heap has been allocated.
- You can find its size using :

arrayVariable.length

For example,

`myList.length` returns 10

Initialising Arrays

- **Using a loop:**

```
for (int i = 0; i < myList.length; i++)
  myList[i] = i; //{0, 1, 2, 3, …}
```

- **Declaring, creating, initializing in one step:**

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

This shorthand syntax must be in one statement.

Declaring, creating, initializing Using the Shorthand Notation

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

This shorthand notation is equivalent to the following statements:

```
double[] myList = new double[4];

myList[0] = 1.9;

myList[1] = 2.9;

myList[2] = 3.4;

myList[3] = 3.5;
```

Testing Arrays

- Objective: The program receives 6 numbers from the keyboard, finds the largest number and counts the occurrence of the largest number entered from the keyboard.

- Suppose you entered 3, 5, 2, 5, 5, and 5, the largest number is 5 and its occurrence count is 4.

Test Array

Run

Assigning Grades

- Objective: read student scores (int) from the keyboard, get the best score, and then assign grades based on the following scheme:
  - Grade is A if score is >= best–10;
  - Grade is B if score is >= best–20;
  - Grade is C if score is >= best–30;
  - Grade is D if score is >= best–40;
  - Grade is F otherwise.

Assign Grade

Run

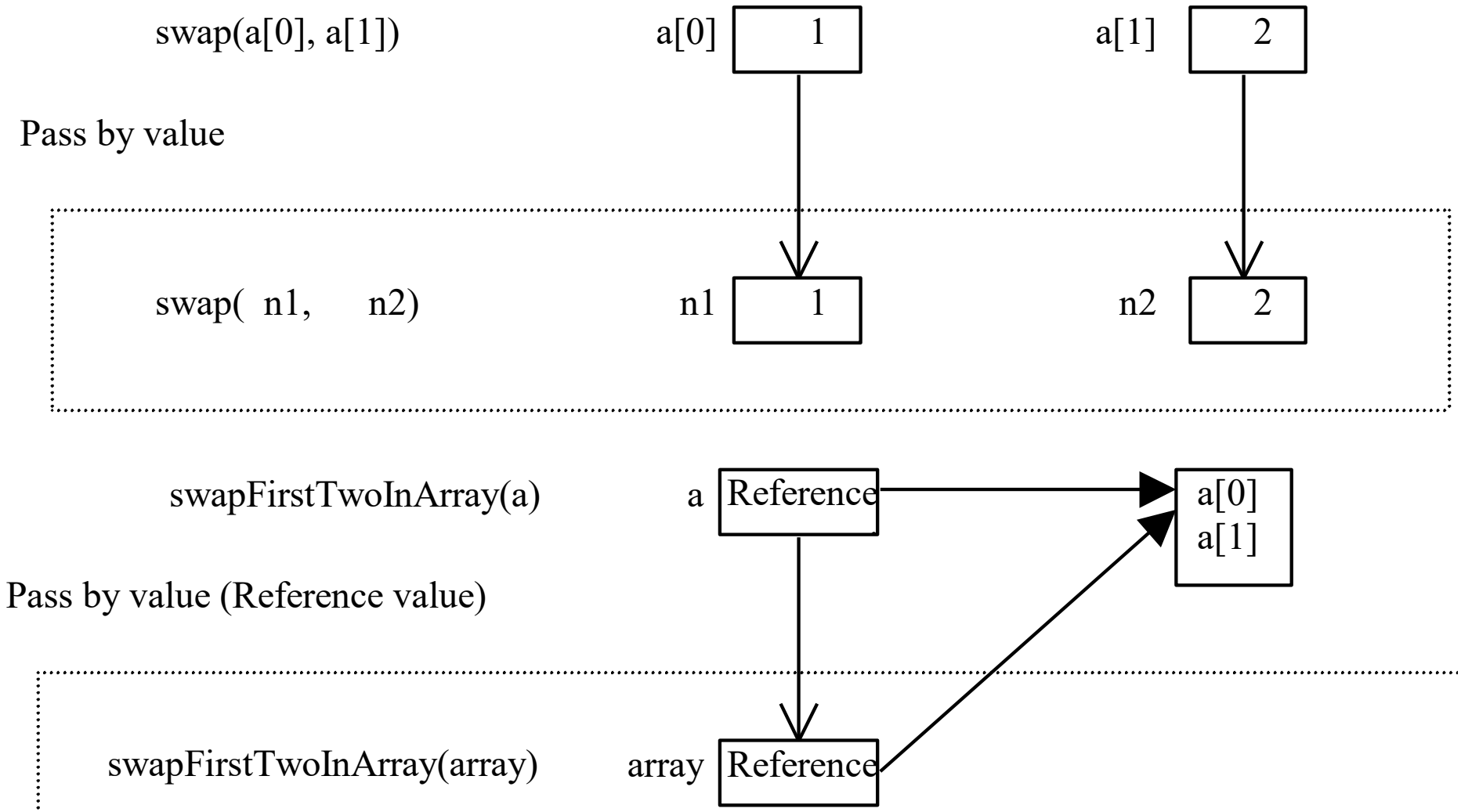
Passing Arrays to Methods

- Java uses *pass by value* to pass parameters to a method. There are important differences between passing a value of variables of primitive data types and passing arrays or other reference types.

- For a parameter of a primitive type value, the actual value is passed. Changing the value of the local parameter inside the method does not affect the value of the variable outside the method.

- For a parameter of an array type, the value of the parameter contains a reference to an array; this reference is passed to the method. Any changes to the array that occur inside the method body will affect the original array that was passed as the argument.

Example, cont.



Passing Arrays as Arguments

- Objective: Demonstrate differences of passing primitive data type variables and array variables.

TestPassArray

Run (Netbeans)

Computing Deviation Using Arrays

$$mean = \frac{\sum_{i=1}^{n} x_i}{n} \qquad deviation = \sqrt{\frac{\sum_{i=1}^{n} (x_i - mean)^2}{n-1}}$$

Deviation

Run (Netbeans)

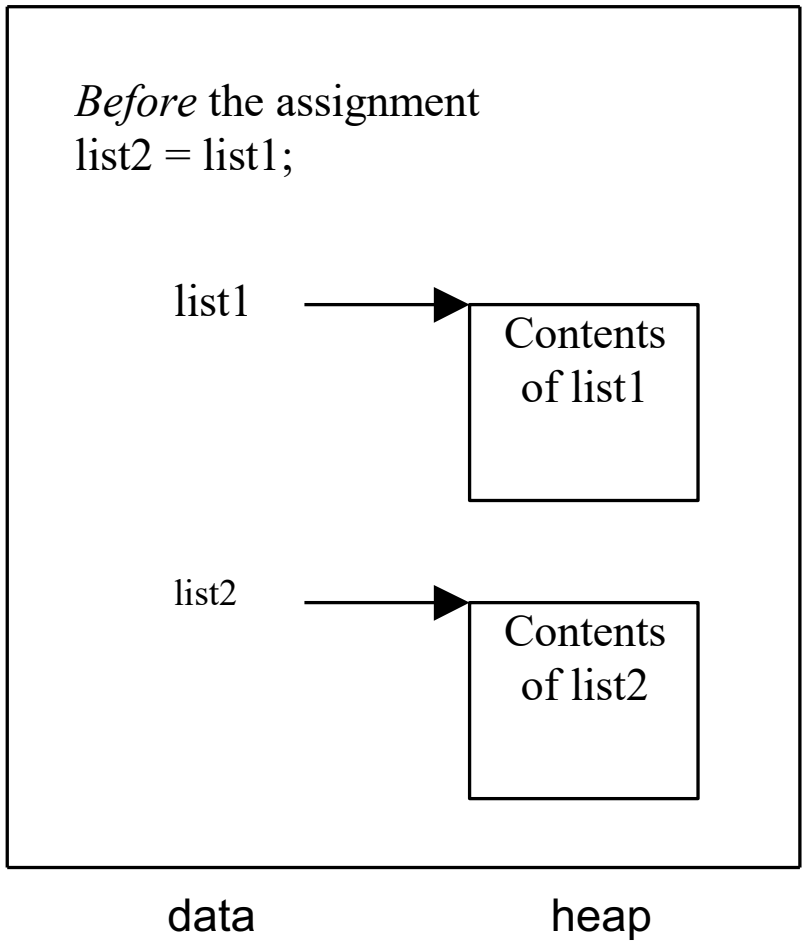
Copying Arrays

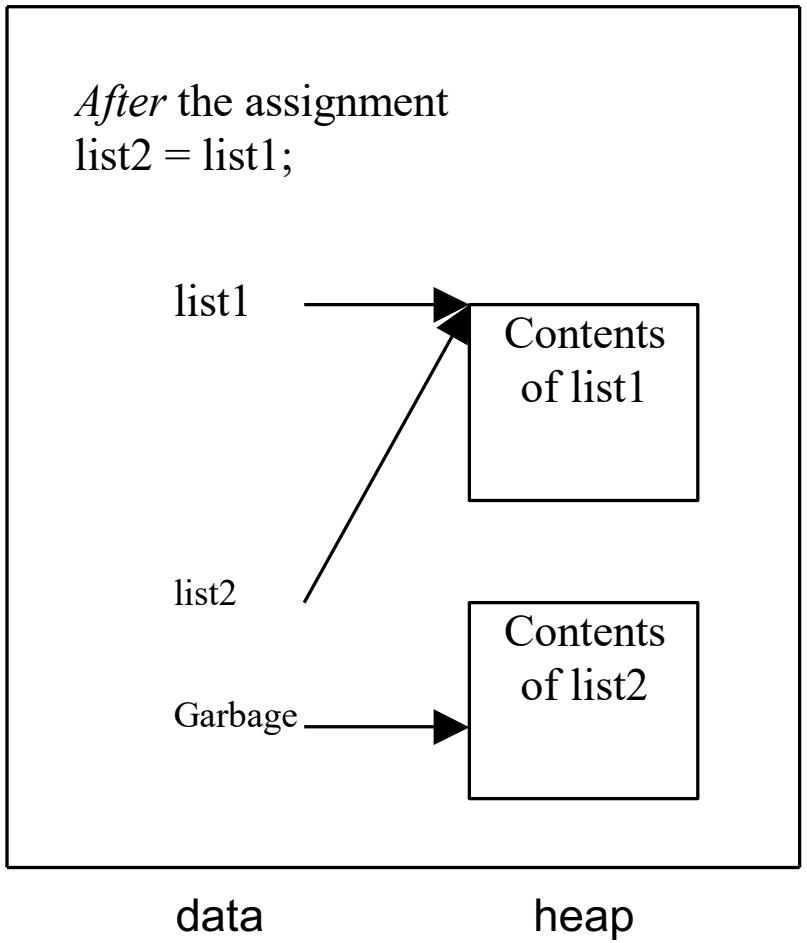
- In this program example, you will see that you cannot copy arrays using just a simple assignment.
- The program simply creates two arrays and attempts to copy one to the other, using an assignment statement.

TestCopyArray

Run (Netbeans)

Copying Arrays





Copying Arrays

Using a loop:

```
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray = new int[sourceArray.length];

for (int i = 0; i < sourceArrays.length; i++)
   targetArray[i] = sourceArray[i];
```

The `arraycopy` Utility

```
arraycopy(sourceArray, src_pos, targetArray,
 tar_pos, length);
```

Example:

```
System.arraycopy(sourceArray, 0, targetArray, 0,
 sourceArray.length);
```

Multidimensional Arrays

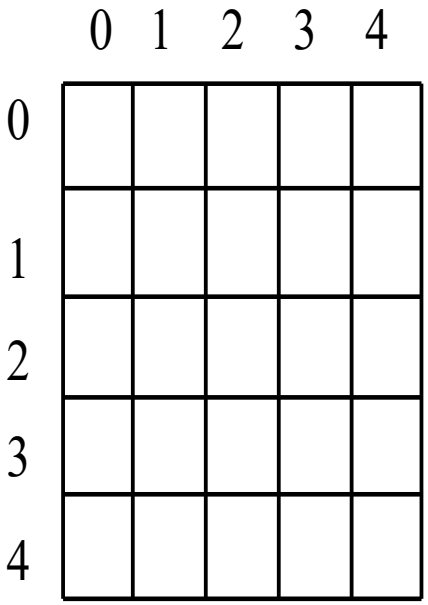
Declaring Variables of Multidimensional Arrays and Creating Multidimensional Arrays

```
int[][] matrix = new int[10][10];
```

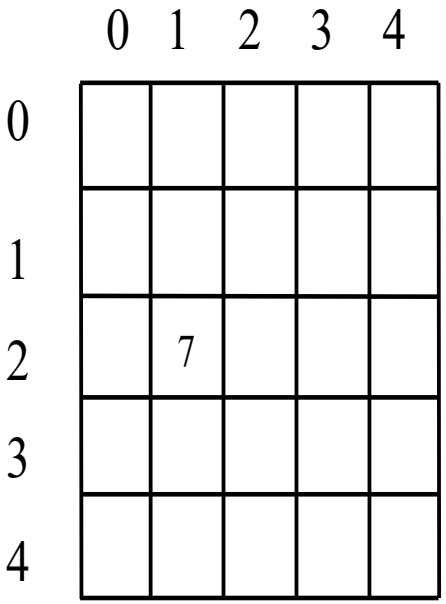
or

```
int matrix[][] = new int[10][10];
matrix[0][0] = 3;

for (int i=0; i<matrix.length; i++){
  for (int j=0; j<matrix[i].length; j++) {
    matrix[i][j] = (int)(Math.random()*1000);
  }
}

double[][] x;
```

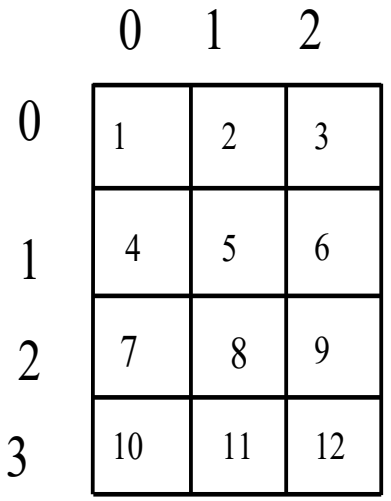
Multidimensional Array Illustration




```
matrix = new int[5][5];
```




```
matrix[2][1] = 7;
```




```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

Declaring, Creating, and Initializing Using Shorthand Notations

You can also use a shorthand notation to declare, create and initialize a two-dimensional array. For example,

```
int[ ][ ] array = { {1, 2, 3},  {4, 5, 6},  {7, 8, 9},
  {10, 11, 12} };
```

This is equivalent to the following statements:

```
int[ ][ ] array = new int[4][3];
array[0][0] = 1; array[0][1] = 2; array[0][2] = 3;
array[1][0] = 4; array[1][1] = 5; array[1][2] = 6;
array[2][0] = 7; array[2][1] = 8; array[2][2] = 9;
array[3][0] = 10; array[3][1] = 11; array[3][2] = 12;
```

Lengths of Multidimensional Arrays

```
int[] [] array = {
   {1, 2, 3},
   {4, 5, 6},
   {7, 8, 9},
   {10, 11, 12}
};

array.length
array[0].length
array[1].length
array[2].length
```

Ragged Arrays

Each row in a two-dimensional array is itself an array. So, the rows can have different lengths. Such an array is known as *a ragged array*. For example,

```
int[ ][ ] matrix = {
  {1, 2, 3, 4, 5},
  {2, 3, 4, 5},
  {3, 4, 5},
  {4, 5},
  {5}
};
```

Adding two matrices

- Objective: Use two-dimensional arrays to create two matrices, and then add and multiply the two matrices.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \\ b_{51} & b_{52} & b_{53} & b_{54} & b_{55} \end{pmatrix} = \begin{pmatrix} a_{11}+b_{11} & a_{12}+b_{12} & a_{13}+b_{13} & a_{14}+b_{14} & a_{15}+b_{15} \\ a_{21}+b_{21} & a_{22}+b_{22} & a_{23}+b_{23} & a_{24}+b_{24} & a_{25}+b_{25} \\ a_{31}+b_{31} & a_{32}+b_{32} & a_{33}+b_{33} & a_{34}+b_{34} & a_{35}+b_{35} \\ a_{41}+b_{41} & a_{42}+b_{42} & a_{43}+b_{43} & a_{44}+b_{44} & a_{45}+b_{45} \\ a_{51}+b_{51} & a_{52}+b_{52} & a_{53}+b_{53} & a_{54}+b_{54} & a_{55}+b_{55} \end{pmatrix}$$

TestMatrixOperation

Run (Netbeans)

Calculating Total Scores

- **Objective: write a program that calculates the total score for students in a class.**
- Suppose the scores are stored in a three-dimensional array named scores.
- The first index in scores refers to a student,
- The second refers to an exam, and
- The third refers to the part of the exam.

Calculating Total Scores

- Suppose there are 7 students, 5 exams, and each exam has two parts--the multiple-choice part and the programming part.

- So, scores[i][j][0] represents the score on the multiple-choice part for the i's student on the j's exam. Your program displays the total score for each student, .

TotalScore

Run in Netbeans