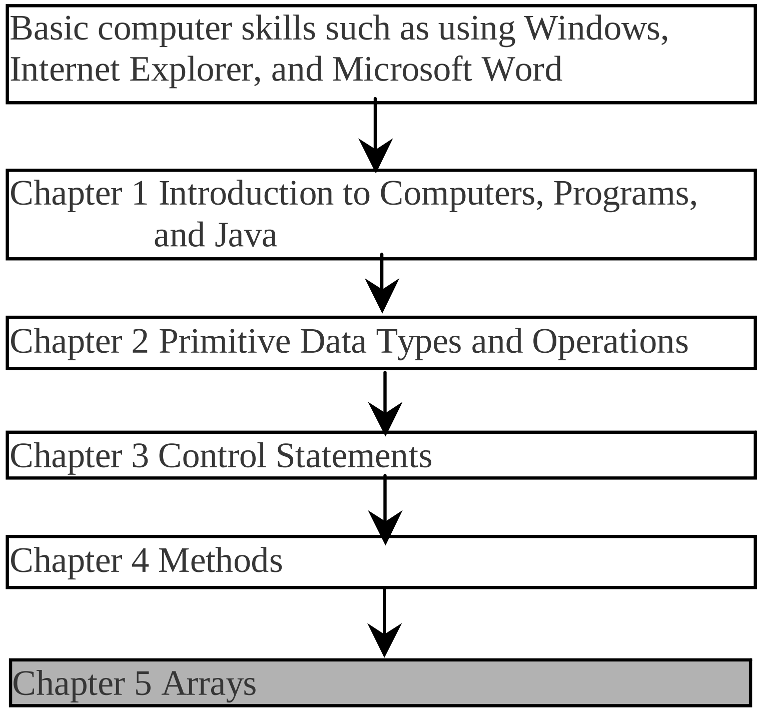
Chapter 5 Arrays

Prerequisites for Part I

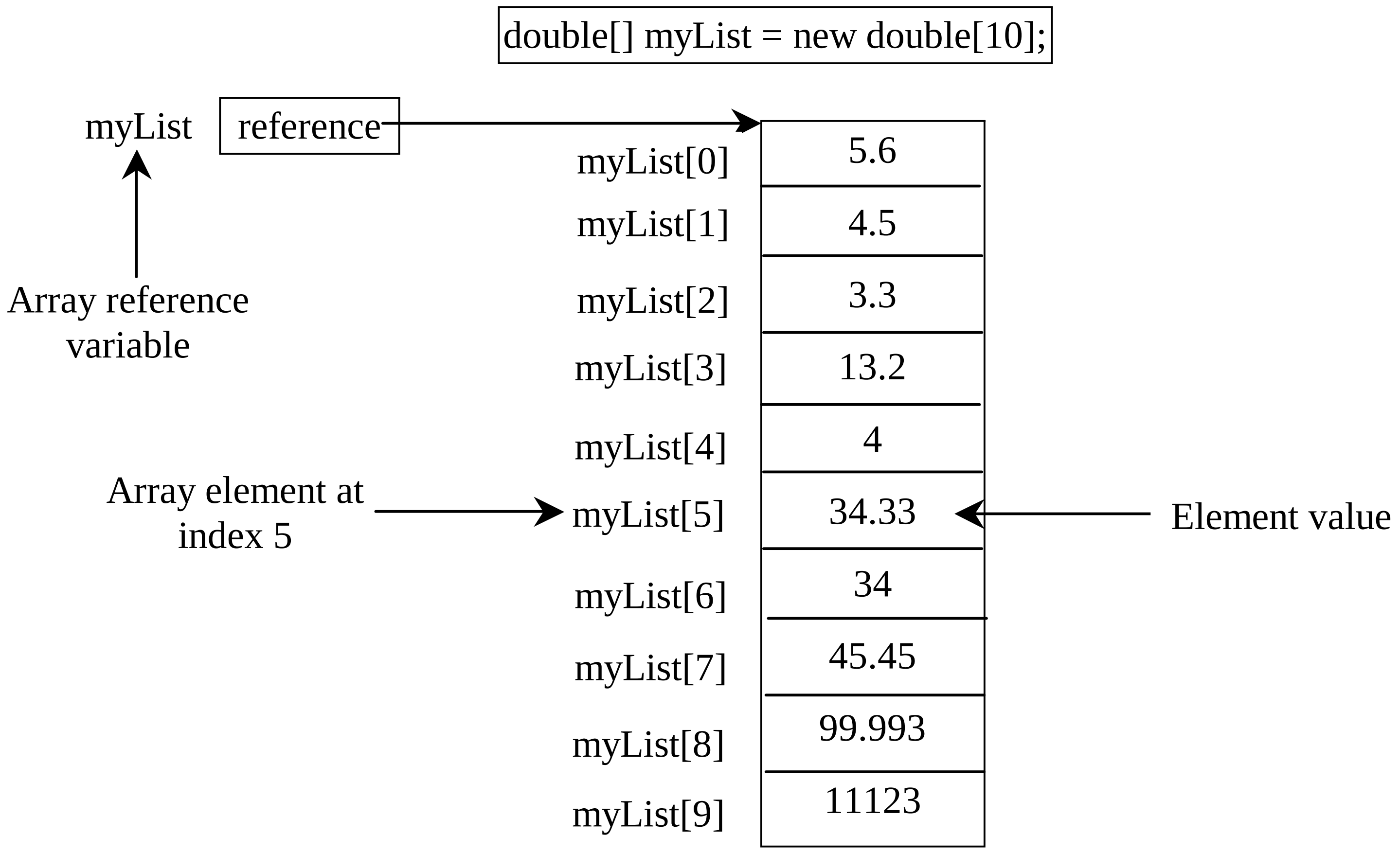

Basic computer skills such as using Windows, Internet Explorer, and Microsoft Word
Chapter 1 Introduction to Computers, Programs, and Java
Chapter 2 Primitive Data Types and Operations
Chapter 3 Control Statements
Chapter 4 Methods
Chapter 5 Arrays

Objectives

- To describe why an array is necessary in programming (§5.1).
- To learn the steps involved in using arrays: declaring array reference variables and creating arrays (§5.2).
- To initialize the values in an array (§5.2).
- To simplify programming using JDK 1.5 enhanced for loop (§5.2).
- To copy contents from one array to another (§5.3).
- To develop and invoke methods with array arguments and ruturn type (§5.4-5.5).
- To sort an array using the selection sort algorithm (§5.6).
- To search elements using the linear or binary search algorithm (§5.7).
- To declare and create multidimensional arrays (§5.8).
- To declare and create multidimensional arrays (§5.9 Optional).

Introducing Arrays

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



Declaring Array Variables

- ☞ `datatype[] arrayRefVar;`

  Example:

  ```
  double[] myList;
  ```

- ☞ `datatype arrayRefVar[];` <u>// This style is correct, but not preferred</u>

  Example:

  ```
  double myList[];
  ```

Creating Arrays

```
arrayRefVar = 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.

Declaring and Creating in One Step

- ☞ `datatype[] arrayRefVar = new datatype[arraySize];`

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

- ☞ `datatype arrayRefVar[] = new datatype[arraySize];`

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

The Length of an Array

Once an array is created, its size is fixed. It cannot be changed. You can find its size using

```
arrayRefVar.length
```

For example,

```
myList.length returns 10
```

Default Values

When an array is created, its elements are assigned the default value of

0 for the numeric primitive data types,

'\u0000' for char types, and

false for boolean types.

Indexed Variables

The array elements are accessed through the index. The array indices are *0-based*, i.e., it starts from 0 to arrayRefVar.length-1. In the example in Figure 5.1, myList holds ten double values and the indices are from 0 to 9.

Each element in the array is represented using the following syntax, known as an *indexed variable*:

```
    arrayRefVar[index];
```

Using Indexed Variables

After an array is created, an indexed variable can be used in the same way as a regular variable. For example, the following code adds the value in myList[0] and myList[1] to myList[2].

```
myList[2] = myList[0] + myList[1];
```

Array Initializers

- ☞ 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;
```

CAUTION

Using the shorthand notation, you have to declare, create, and initialize the array all in one statement. Splitting it would cause a syntax error. For example, the following is wrong:

```
double[] myList;

myList = {1.9, 2.9, 3.4, 3.5};
```

Processing Arrays

See the examples in the text.

1. (Initializing arrays)
2. (Printing arrays)
3. (Summing all elements)
4. (Finding the largest element)
5. (Finding the smallest index of the largest element)

JDK 1.5
Feature

Enhanced for Loop

JDK 1.5 introduced a new for loop that enables you to traverse the complete array sequentially without using an index variable. For example, the following code displays all elements in the array myList:

```
for (double value: myList)
  System.out.println(value);
```

In general, the syntax is

```
for (elementType value: arrayRefVar) {
  // Process the value
}
```

You still have to use an index variable if you wish to traverse the array in a different order or change the elements in the array.

Example 5.1
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.

TestArray

Example 5.2
Assigning Grades

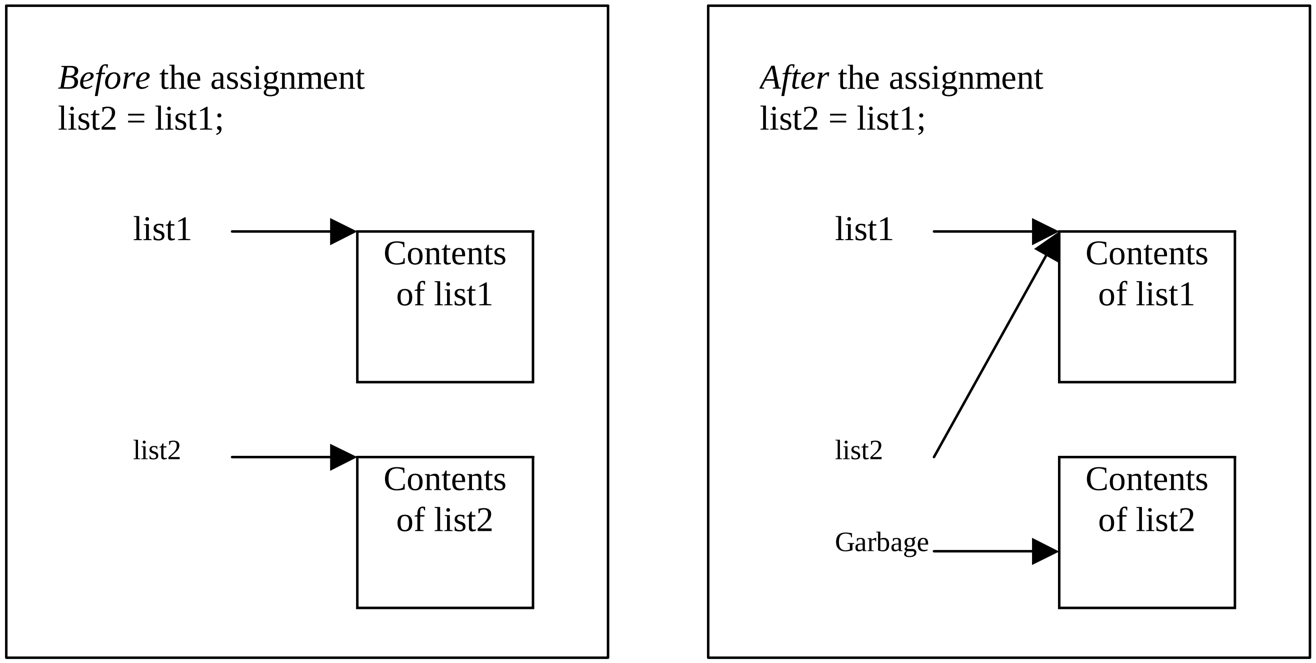
- Objective: read student scores (int), get the best score, and then assign grades based on the following scheme:
  - Grade is A if score is >= best–10;
  - Grade is B if score is >= best–20;
  - Grade is C if score is >= best–30;
  - Grade is D if score is >= best–40;
  - Grade is F otherwise.

AssignGrade

Copying Arrays

Often, in a program, you need to duplicate an array or a part of an array. In such cases you could attempt to use the assignment statement (=), as follows:

list2 = list1;



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);
```

Passing Arrays to Methods

```
public static void printArray(int[] array) {
  for (int i = 0; i < array.length; i++) {
    System.out.print(array[i] + " ");
  }
}
```

```
        Invoke the method

        int[] list = {3, 1, 2, 6, 4, 2};
        printArray(list);
```

```
                Invoke the method
                printArray(new int[]{3, 1, 2, 6, 4, 2});
```

Anonymous array

Anonymous Array

The statement

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

creates an array using the following syntax:

```
new dataType[]{literal0, literal1, ..., literalk};
```

There is no explicit reference variable for the array. Such array is called an *anonymous array*.

Pass By Value

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.

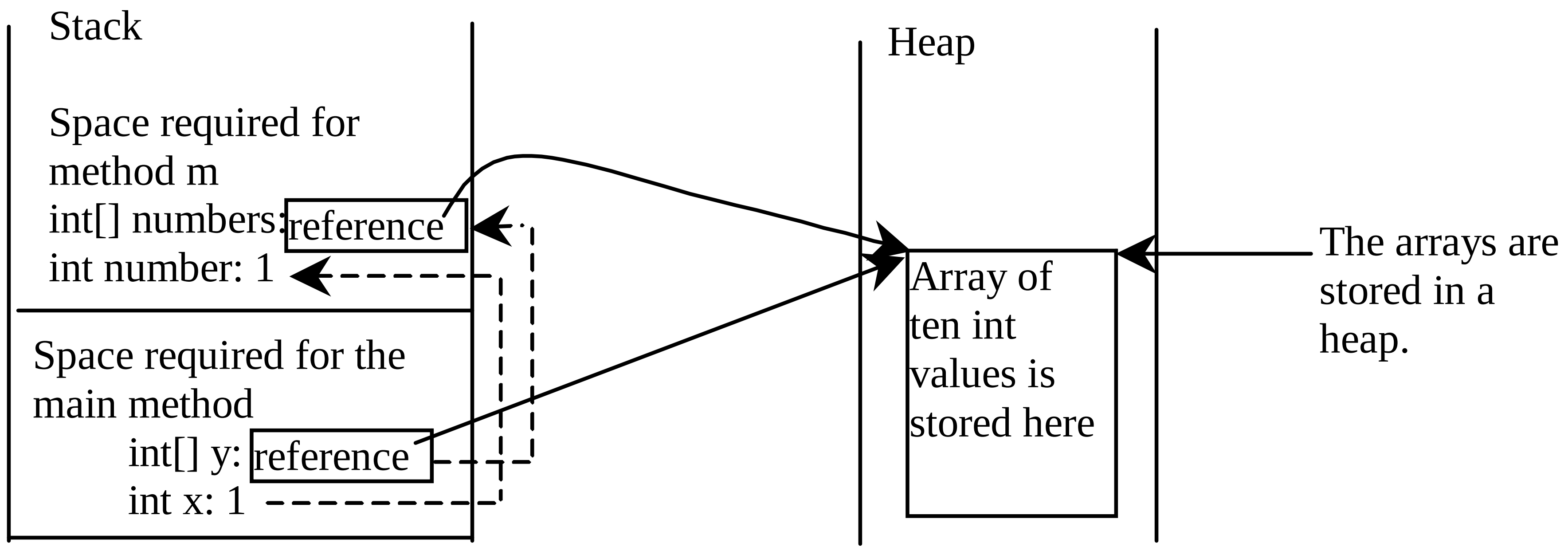
☞ For a parameter of a primitive type value, the actual value is passed. Changing the value of the local parameter inside the method does not affect the value of the variable outside the method.

☞ 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.

Simple Example

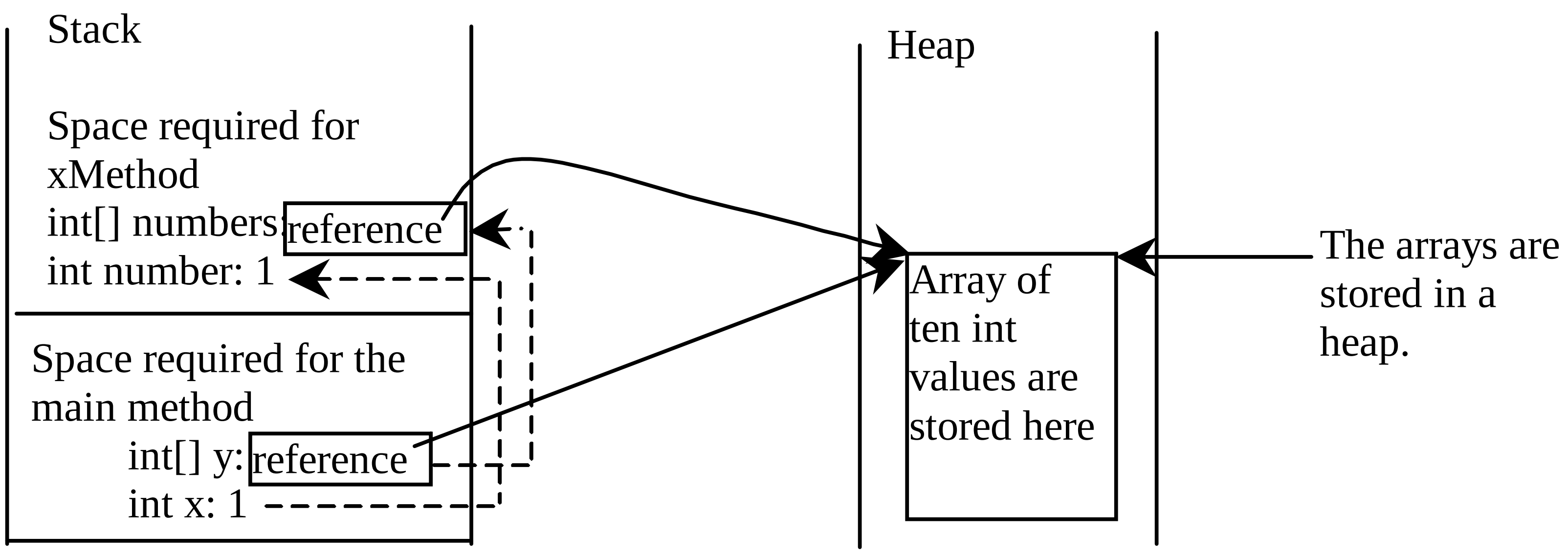
```
public class Test {
  public static void main(String[] args) {
    int x = 1; // x represents an int value
    int[] y = new int[10]; // y represents an array of int values

    m(x, y); // Invoke m with arguments x and y

    System.out.println("x is " + x);
    System.out.println("y[0] is " + y[0]);
  }

  public static void m(int number, int[] numbers) {
    number = 1001; // Assign a new value to number
    numbers[0] = 5555; // Assign a new value to numbers[0]
  }
}
```

Call Stack


Stack
Space required for method m
int[] numbers:
reference
int number: 1
Space required for the main method
int[] y:
reference
int x: 1
Heap
Array of ten int values is stored here
The arrays are stored in a heap.

When invoking m(x, y), the values of x and y are passed to number and numbers. Since y contains the reference value to the array, numbers now contains the same reference value to the same array.

Heap


Stack
Space required for xMethod
int[] numbers:
reference
int number: 1
Space required for the main method
int[] y:
reference
int x: 1
Heap
Array of ten int values are stored here
The arrays are stored in a heap.

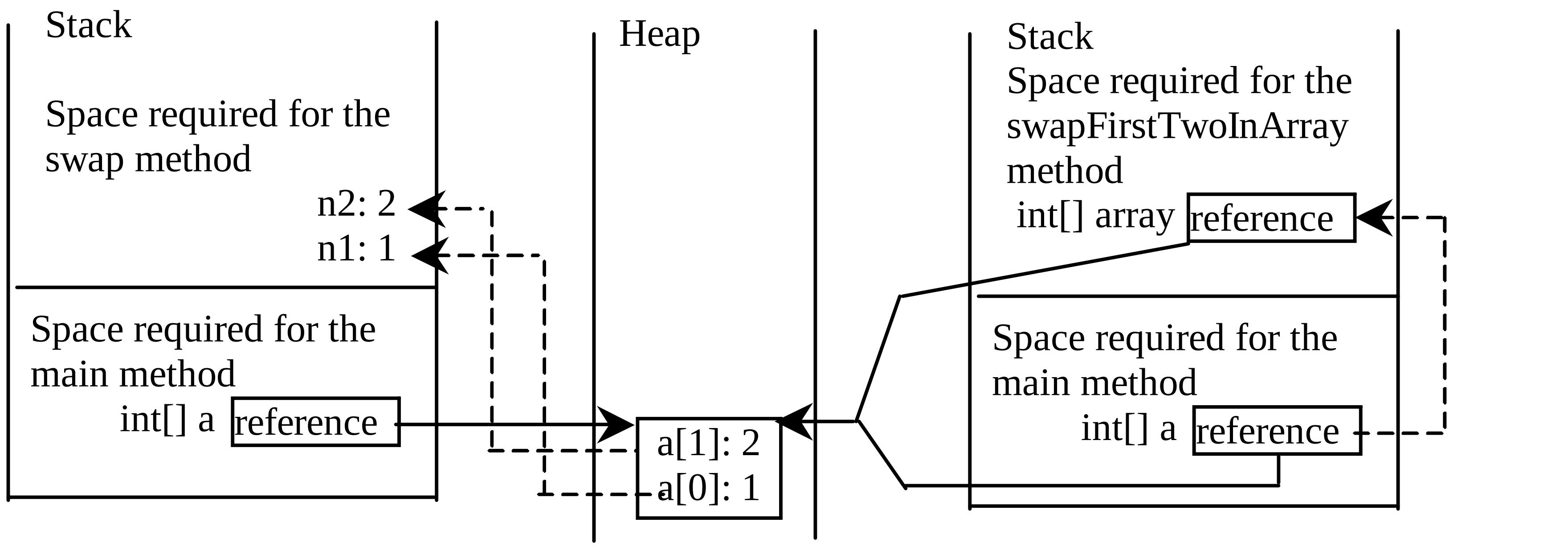
The JVM stores the array in an area of memory, called *heap*, which is used for dynamic memory allocation where blocks of memory are allocated and freed in an arbitrary order.

Example 5.3
Passing Arrays as Arguments

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

TestPassArray

Example 5.3, cont.


Stack
Space required for the swap method
n2: 2
n1: 1
Space required for the main method
int[] a
reference
Heap
a[1]: 2
a[0]: 1
Stack
Space required for the swapFirstTwoInArray method
int[] array
reference
Space required for the main method
int[] a
reference

Invoke swap(int n1, int n2). The primitive type values in a[0] and a[1] are passed to the swap method.

The arrays are stored in a heap.

Invoke swapFirstTwoInArray(int[] array). The reference value in a is passed to the swapFirstTwoInArray method.

Returning an Array from a Method

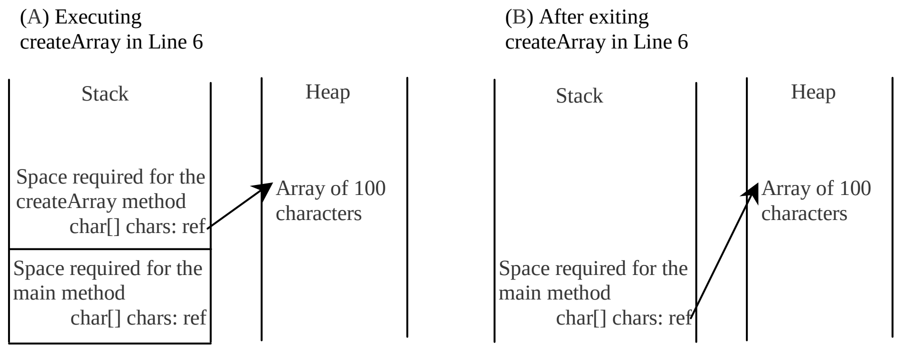
```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1;
       i < list.length; i++, j--) {
    result[j] = list[i];
  }

  return result;
}
```

```
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

Example 5.4
Counting Occurrence of Each Letter

A method may return an array, as shown in the following example.

- Generate 100 lowercase letters randomly and assign to an array of characters.
- Count the occurrence of each letter in the array.

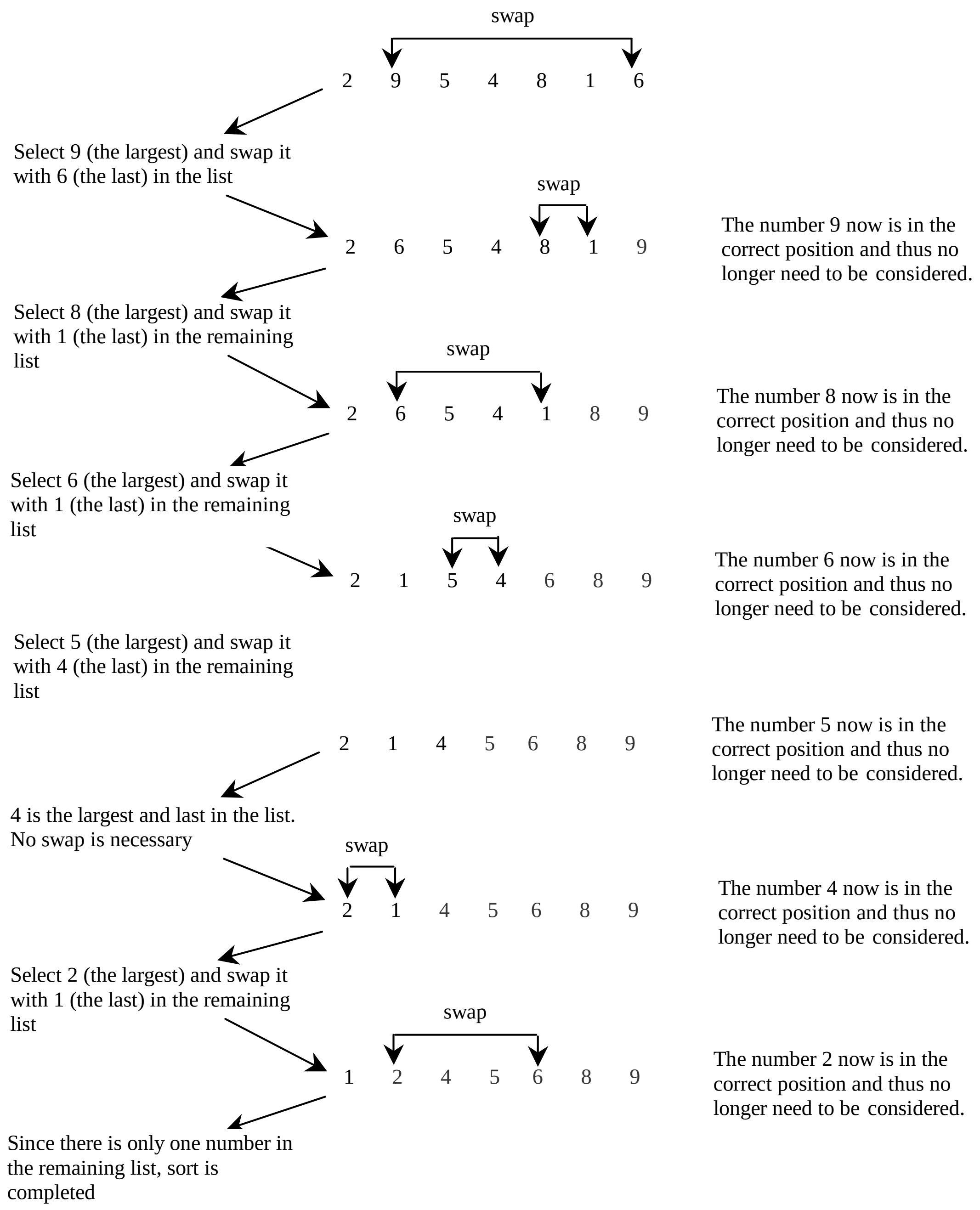


(A) Executing createArray in Line 6
Stack
Heap
Space required for the createArray method
char[] chars: ref
Array of 100 characters
Space required for the main method
char[] chars: ref
(B) After exiting createArray in Line 6
Stack
Heap
Array of 100 characters
Space required for the main method
char[] chars: ref

CountLettersInArray

Selection Sort

Selection sort finds the largest number in the list and places it last. It then finds the largest number remaining and places it next to last, and so on until the list contains only a single number. Figure 5.17 shows how to sort the list {2, 9, 5, 4, 8, 1, 6} using selection sort.


swap
2 9 5 4 8 1 6
Select 9 (the largest) and swap it with 6 (the last) in the list
swap
2 6 5 4 8 1 9
The number 9 now is in the correct position and thus no longer need to be considered.
Select 8 (the largest) and swap it with 1 (the last) in the remaining list
swap
2 6 5 4 1 8 9
The number 8 now is in the correct position and thus no longer need to be considered.
Select 6 (the largest) and swap it with 1 (the last) in the remaining list
swap
2 1 5 4 6 8 9
The number 6 now is in the correct position and thus no longer need to be considered.
Select 5 (the largest) and swap it with 4 (the last) in the remaining list
2 1 4 5 6 8 9
The number 5 now is in the correct position and thus no longer need to be considered.
4 is the largest and last in the list. No swap is necessary
swap
2 1 4 5 6 8 9
The number 4 now is in the correct position and thus no longer need to be considered.
Select 2 (the largest) and swap it with 1 (the last) in the remaining list
swap
1 2 4 5 6 8 9
The number 2 now is in the correct position and thus no longer need to be considered.
Since there is only one number in the remaining list, sort is completed

From Idea to Solution

```
for (int i = list.length - 1; i >= 1; i--) {
  select the largest element in list[0..i];
  swap the largest with list[i], if necessary;
  // list[i] is in place. The next iteration apply on list[0..i-1]
}
```

From Idea to Solution

```
for (int i = list.length - 1; i >= 1; i--) {
  select the largest element in list[0..i];
  swap the largest with list[i], if necessary;
  // list[i] is in place. The next iteration apply on list[0..i-1]
}
```

```
    // Find the maximum in the list[0..i]
    double currentMax = list[0];
    ~~int currentMaxIndex = 0;~~

    for (int j = 1; j <= i; j++) {
      if (currentMax < list[j]) {
        currentMax = list[j];
        ~~currentMaxIndex = j;~~
      }
    }
```

From Idea to Solution

```
for (int i = list.length - 1; i >= 1; i--) {
  select the largest element in list[0..i];
  swap the largest with list[i], if necessary;
  // list[i] is in place. The next iteration apply on list[0..i-1]
}
```

```
// Swap list[i] with list[currentMaxIndex] if necessary;
if (currentMaxIndex != i) {
  list[currentMaxIndex] = list[i];
  list[i] = currentMax;
}
```

Wrap it in a Method

```
/** The method for sorting the numbers */
public static void selectionSort(double[] list) {
  for (int i = list.length - 1; i >= 1; i--) {
    // Find the maximum in the list[0..i]
    double currentMax = list[0];
    int currentMaxIndex = 0;

    for (int j = 1; j <= i; j++) {
      if (currentMax < list[j]) {
        currentMax = list[j];
        currentMaxIndex = j;
      }
    }

    // Swap list[i] with list[currentMaxIndex] if necessary;
    if (currentMaxIndex != i) {
      list[currentMaxIndex] = list[i];
      list[i] = currentMax;
    }
  }
}
```

Invoke it

selectionSort(yourList)

The Arrays.sort Method

Since sorting is frequently used in programming, Java provides several overloaded sort methods for sorting an array of int, double, char, short, long, and float in the java.util.Arrays class. For example, the following code sorts an array of numbers and an array of characters.

```
double[] numbers = {5.0, 4.4, 1.9, 2.9, 3.4, 3.5};
java.util.Arrays.sort(numbers);

char[] chars = {'a', 'A', '4', 'F', 'D', 'P'};
java.util.Arrays.sort(chars);
```

Optional

Exercise 5.14 Bubble Sort

int[] myList = {2, 9, 5, 4, 8, 1, 6}; // Unsorted

The bubble-sort algorithm makes several iterations through the array. On each iteration, successive neighboring pairs are compared. If a pair is in decreasing order, its values are swapped; otherwise, the values remain unchanged. The technique is called a *bubble sort* or *sinking sort* because the smaller values gradually "bubble" their way to the top and the larger values sink to the bottom.

Iteration 1:	2, 5, 4, 8, 1, 6, 9
Iteration 2:	2, 4, 5, 1, 6, 8, 9
Iteration 3:	2, 4, 1, 5, 6, 8, 9
Iteration 4:	2, 1, 4, 5, 6, 8, 9
Iteration 5:	1, 2, 4, 5, 6, 8, 9
Iteration 6:	1, 2, 4, 5, 6, 8, 9

Optional

Exercise 5.15 Insertion Sort

int[] myList = {2, 9, 5, 4, 8, 1, 6}; // Unsorted

The insertion sort algorithm sorts a list of values by repeatedly inserting an unsorted element into a sorted sublist until the whole list is sorted.

Iteration 1:	*2*, *9*, 5, 4, 8, 1, 6
Iteration 2:	*2*, *5*, *9*, 4, 8, 1, 6
Iteration 3:	*2*, *4*, *5*, *9*, 8, 1, 6
Iteration 4:	*2*, *4*, *5*, *8*, *9*, 1, 6
Iteration 5:	*1*, *2*, *4*, *5*, *8*, *9*, 6
Iteration 6:	*1*, *2*, *4*, *5*, *6*, *8*, *9*

Searching Arrays

Searching is the process of looking for a specific element in an array; for example, discovering whether a certain score is included in a list of scores. Searching, like sorting, is a common task in computer programming. There are many algorithms and data structures devoted to searching. In this section, two commonly used approaches are discussed, *linear search* and *binary search*.

Linear Search

The linear search approach compares the key element, key, *sequentially* with each element in the array list. The method continues to do so until the key matches an element in the list or the list is exhausted without a match being found. If a match is made, the linear search returns the index of the element in the array that matches the key. If no match is found, the search returns -1.

From Idea to Solution

```
/** The method for finding a key in the list */
public static int linearSearch(int[] list, int key) {
  for (int i = 0; i < list.length; i++)
    if (key == list[i])
      return i;
  return -1;
}
```

Trace the method

```
int[] list = {1, 4, 4, 2, 5, -3, 6, 2};
int i = linearSearch(list, 4);  // returns 1
int j = linearSearch(list, -4); // returns -1
int k = linearSearch(list, -3); // returns 5
```

Binary Search

For binary search to work, the elements in the array must already be ordered. Without loss of generality, assume that the array is in ascending order.

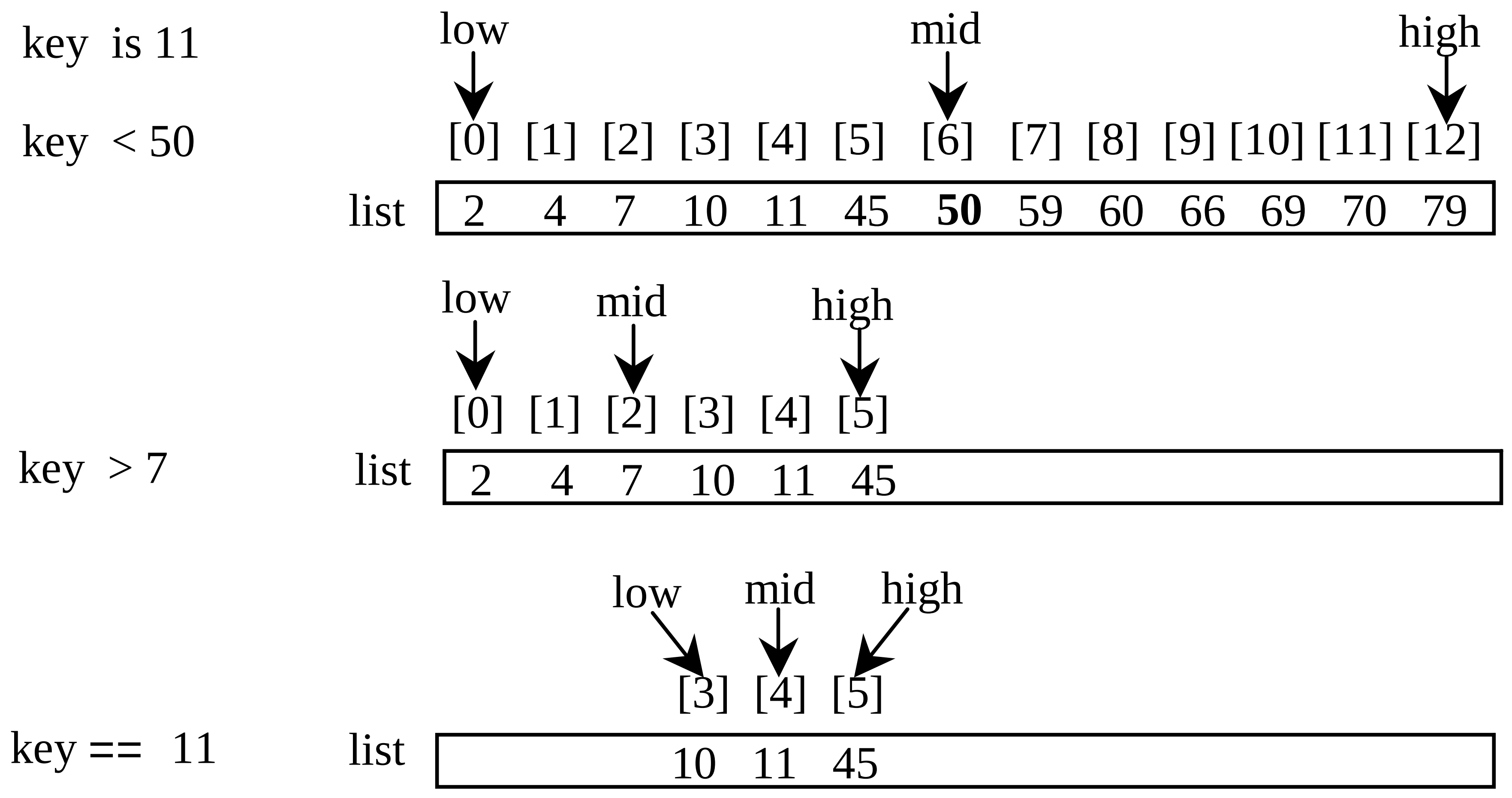
e.g., 2 4 7 10 11 45 50 59 60 66 69 70 79

The binary search first compares the key with the element in the middle of the array. Consider the following three cases:

Binary Search, cont.

- If the key is less than the middle element, you only need to search the key in the first half of the array.
- If the key is equal to the middle element, the search ends with a match.
- If the key is greater than the middle element, you only need to search the key in the second half of the array.

Binary Search, cont.



Binary Search, cont.

The binarySearch method returns the index of the search key if it is contained in the list. Otherwise, it returns –insertion point - 1. The insertion point is the point at which the key would be inserted into the list.

From Idea to Soluton

```
/** Use binary search to find the key in the list */
public static int binarySearch(int[] list, int key) {
  int low = 0;
  int high = list.length - 1;

  while (high >= low) {
    int mid = (low + high) / 2;
    if (key < list[mid])
      high = mid - 1;
    else if (key == list[mid])
      return mid;
    else
      low = mid + 1;
  }

  return -1 - low;
}
```

The Arrays.binarySearch Method

Since binary search is frequently used in programming, Java provides several overloaded binarySearch methods for searching a key in an array of int, double, char, short, long, and float in the java.util.Arrays class. For example, the following code searches the keys in an array of numbers and an array of characters.

```
int[] list = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79};
System.out.println("Index is " +
  java.util.Arrays.binarySearch(list, 11));
```

Return is 4

```
char[] chars = {'a', 'c', 'g', 'x', 'y', 'z'};
System.out.println("Index is " +
  java.util.Arrays.binarySearch(chars, 't'));
```

Return is –4 (insertion point is 3)

For the binarySearch method to work, the array must be pre-sorted in increasing order.

Optional

Recursive Implementation

```
/** Use binary search to find the key in the list */
public static int recursiveBinarySearch(int[] list, int key) {
  int low = 0;
  int high = list.length - 1;
  return recursiveBinarySearch(list, key, low, high);
}

/** Use binary search to find the key in the list between
    list[low] list[high] */
public static int recursiveBinarySearch(int[] list, int key,
  int low, int high) {
  if (low > high)  // The list has been exhausted without a match
    return -low - 1;

  int mid = (low + high) / 2;
  if (key < list[mid])
    return recursiveBinarySearch(list, key, low, mid - 1);
  else if (key == list[mid])
    return mid;
  else
    return recursiveBinarySearch(list, key, mid + 1, high);
}
```

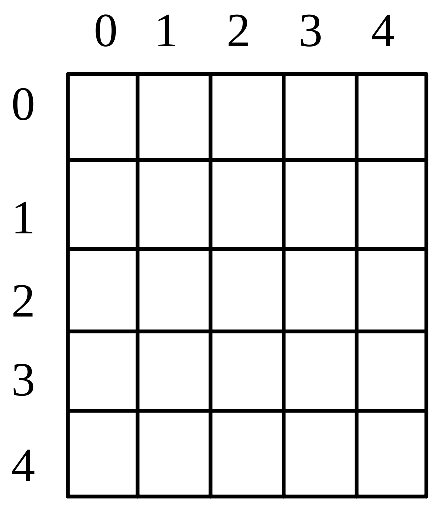
Two-dimensional Arrays

```
// Declare array ref var
dataType[][] refVar;

// Create array and assign its reference to variable
refVar = new dataType[10][10];

// Combine declaration and creation in one statement
dataType[][] refVar = new dataType[10][10];

// Alternative syntax
dataType refVar[][] = new dataType[10][10];
```

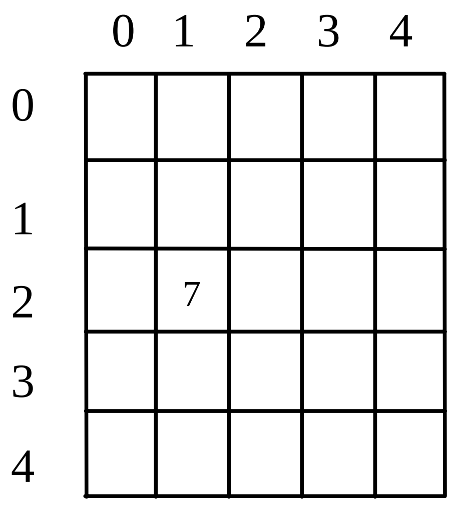
Declaring Variables of Two-dimensional Arrays and Creating Two-dimensional Arrays

```
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;
```

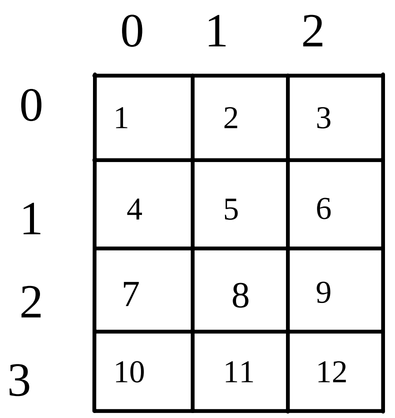
Two-dimensional Array Illustration



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



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



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

matrix.length? 5

matrix[0].length? 5

array.length? 4

array[0].length? 3

Declaring, Creating, and Initializing Using Shorthand Notations

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

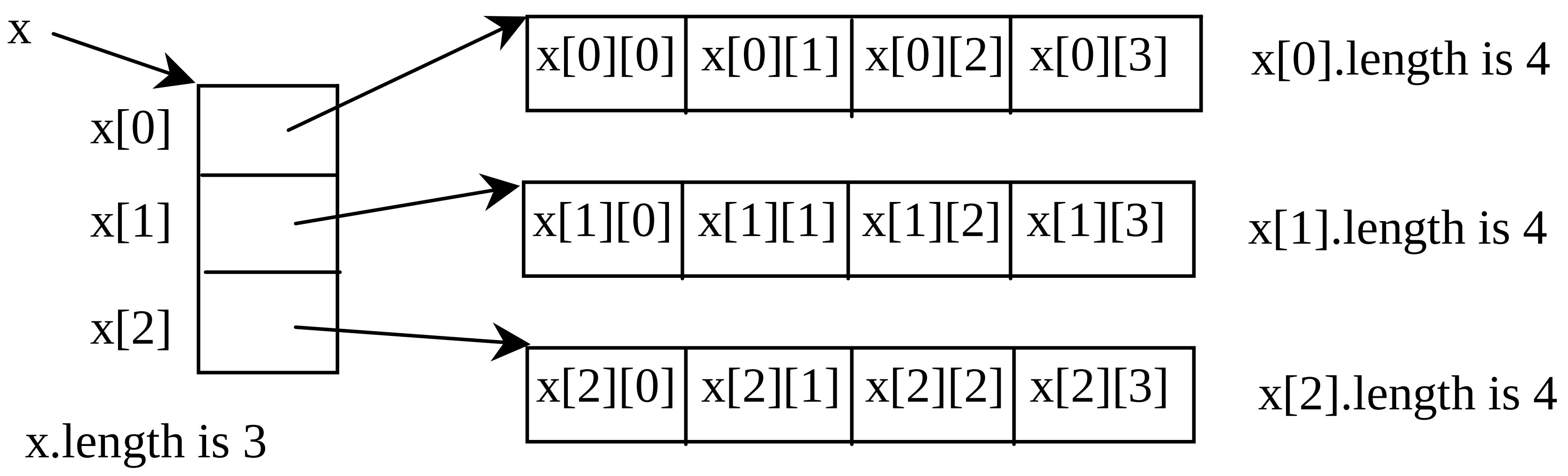
```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

Same as

```
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 Two-dimensional Arrays

int[][] x = new int[3][4];



Lengths of Two-dimensional Arrays, cont.

```
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
array[3].length
```

array[4].length   ArrayIndexOutOfBoundsException

Lengths of Two-dimensional 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
array[3].length
```

array[4].length   ArrayIndexOutOfBoundsException

Ragged Arrays

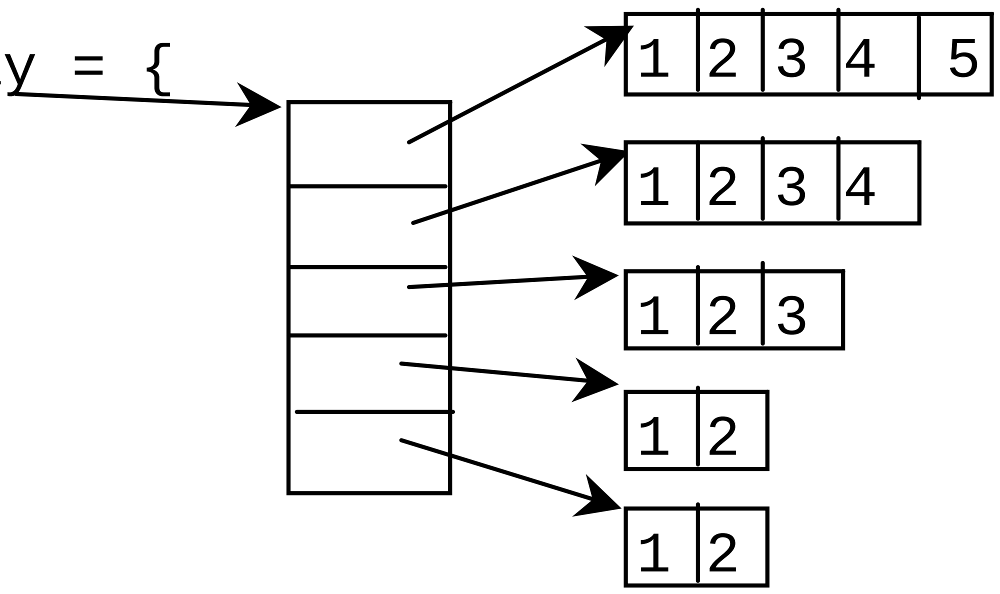
Each row in a two-dimensional array is itself an array. So, the rows can have different lengths. Such an array is known as *a ragged array*. For example,

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

```
matrix.length is 5
matrix[0].length is 5
matrix[1].length is 4
matrix[2].length is 3
matrix[3].length is 2
matrix[4].length is 1
```

Ragged Arrays, cont.

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



Example 5.5
Grading Multiple-Choice Test

Students' Answers to the Questions:

	0	1	2	3	4	5	6	7	8	9
Student 0	A	B	A	C	C	D	E	E	A	D
Student 1	D	B	A	B	C	A	E	E	A	D
Student 2	E	D	D	A	C	B	E	E	A	D
Student 3	C	B	A	E	D	C	E	E	A	D
Student 4	A	B	D	C	C	D	E	E	A	D
Student 5	B	B	E	C	C	D	E	E	A	D
Student 6	B	B	A	C	C	D	E	E	A	D
Student 7	E	B	E	C	C	D	E	E	A	D

- Objective: write a program that grades multiple-choice test.

Key to the Questions:

	0	1	2	3	4	5	6	7	8	9
Key	D	B	D	C	C	D	A	E	A	D

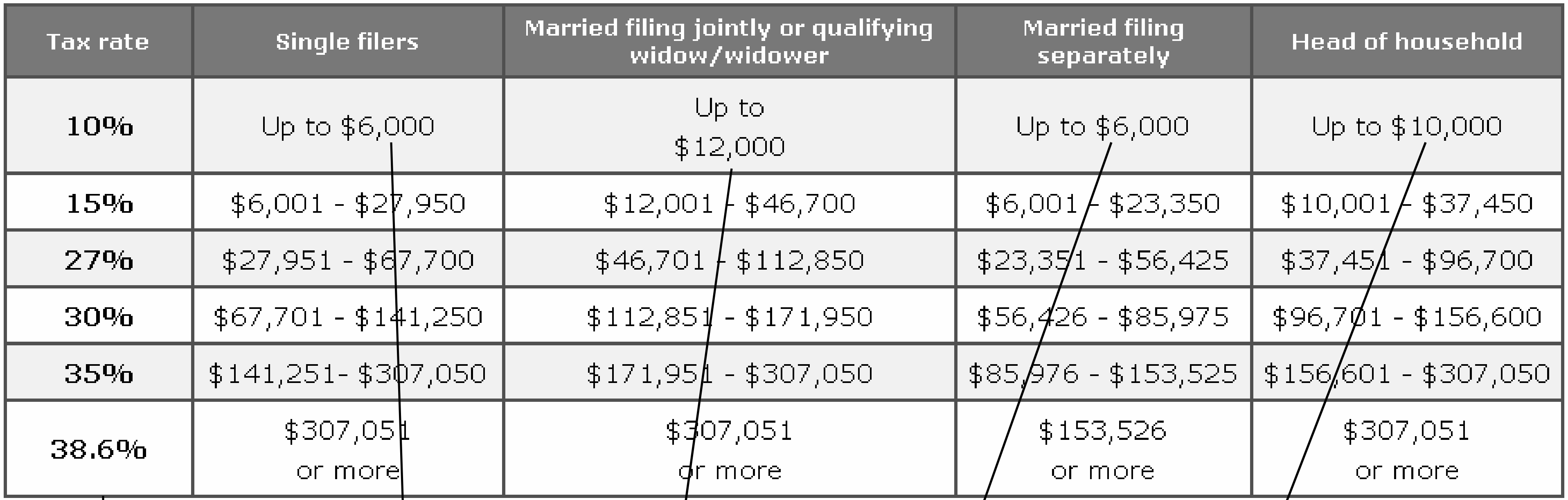
GradeExam

Example 5.6
Computing Taxes Using Arrays

Example 4.4, “Computing Taxes with Methods,” simplified Example 3.1, “Computing Taxes.” Example 4.4 can be further improved using arrays. Rewrite Example 3.1 using arrays to store tax rates and brackets.

ComputeTax

Refine the table



Tax rate	Single filers	Married filing jointly or qualifying widow/widower	Married filing separately	Head of household
10%	Up to $6,000	Up to $12,000	Up to $6,000	Up to $10,000
15%	$6,001 - $27,950	$12,001 - $46,700	$6,001 - $23,350	$10,001 - $37,450
27%	$27,951 - $67,700	$46,701 - $112,850	$23,351 - $56,425	$37,451 - $96,700
30%	$67,701 - $141,250	$112,851 - $171,950	$56,426 - $85,975	$96,701 - $156,600
35%	$141,251- $307,050	$171,951 - $307,050	$85,976 - $153,525	$156,601 - $307,050
38.6%	$307,051 or more	$307,051 or more	$153,526 or more	$307,051 or more

10%
15%
27%
30%
35%
38.6%

6000	12000	6000	10000
27950	46700	23350	37450
67700	112850	56425	96745
141250	171950	85975	156600
307050	307050	153525	307050

Reorganize the table

6000	12000	6000	10000
27950	46700	23350	37450
67700	112850	56425	96745
141250	171950	85975	156600
307050	307050	153525	307050

Rotate

6000	27950	67700	141250	307050	Single filer
12000	46700	112850	171950	307050	Married jointly
6000	23350	56425	85975	153525	Married separately
10000	37450	96745	156600	307050	Head of household

Declare Two Arrays

6000	27950	67700	141250	307050	Single filer
12000	46700	112850	171950	307050	Married jointly
6000	23350	56425	85975	153525	Married separately
10000	37450	96745	156600	307050	Head of household

```
int[][] brackets = {
  {6000, 27950, 67700, 141250, 307050}, // Single filer
  {12000, 46700, 112850, 171950, 307050}, // Married jointly
  {6000, 23350, 56425, 85975, 153525}, // Married separately
  {10000, 37450, 96700, 156600, 307050} // Head of household
};
```

10%
15%
27%
30%
35%
38.6%

```
double[] rates = {0.10, 0.15, 0.27, 0.30, 0.35, 0.386};
```

Multidimensional Arrays

Occasionally, you will need to represent n-dimensional data structures. In Java, you can create n-dimensional arrays for any integer n.

The way to declare two-dimensional array variables and create two-dimensional arrays can be generalized to declare n-dimensional array variables and create n-dimensional arrays for n >= 3. For example, the following syntax declares a three-dimensional array variable scores, creates an array, and assigns its reference to scores.

```
double[][][] scores = new double[10][5][2];
```

Example 5.7 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. 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