Chapter 7 Single-Dimensional Arrays

Objectives

To describe why arrays are necessary in programming

To declare array reference variables and create arrays

To initialize the values in an array

To access array elements using indexed variables.

To declare, create, and initialize an array using an array initializer

To program common array operations (displaying arrays, summing all elements, finding min and max elements, random shuffling, shifting elements)

To simplify programming using the for-each loops

To apply arrays in the LottoNumbers and DeckOfCards problems

To copy contents from one array to another

To develop and invoke methods with array arguments and return value

To define a method with variable-length argument list

To search elements using the linear or binary search algorithm.

To sort an array using the selection sort

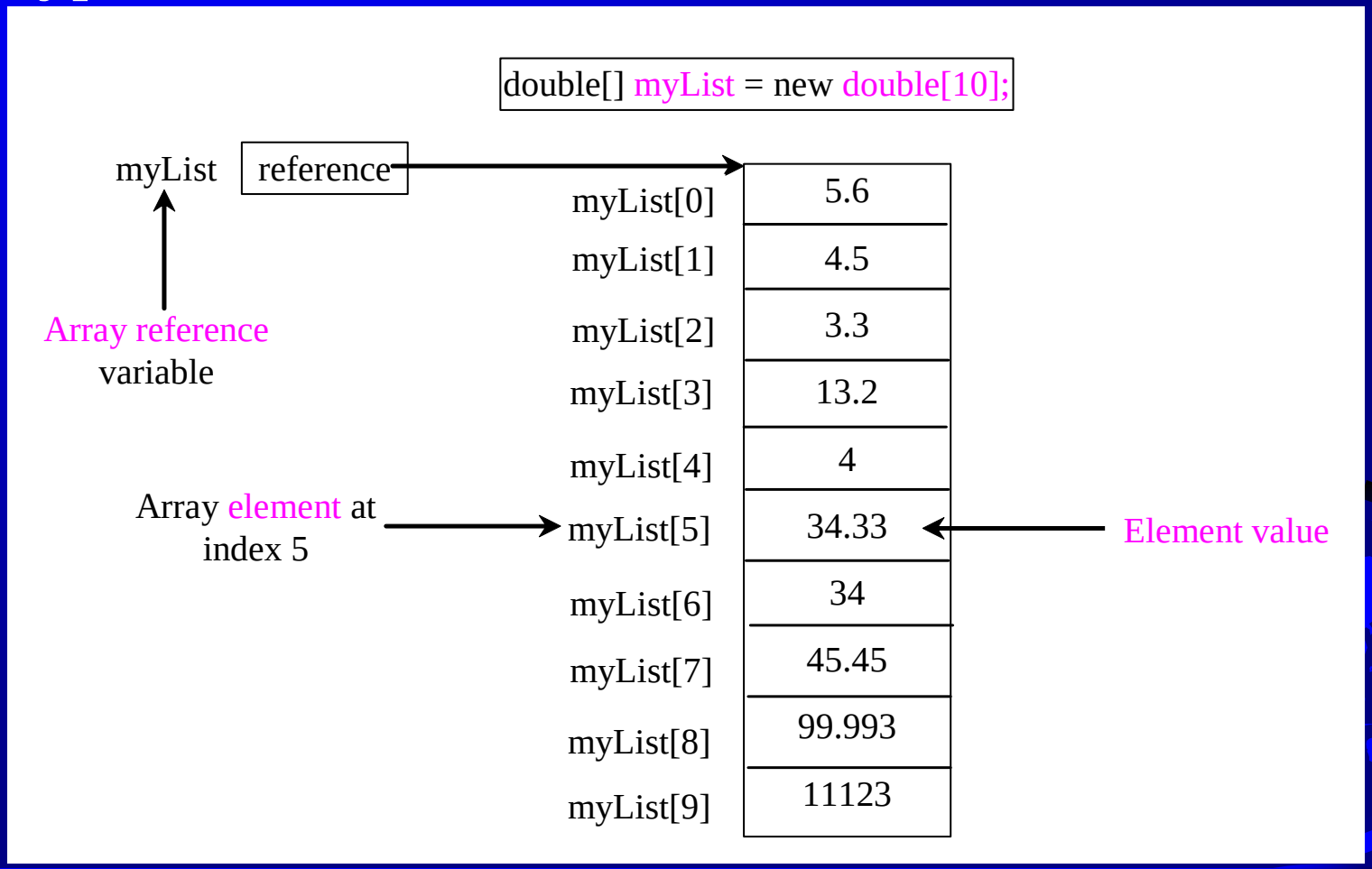
To sort an array using the insertion sort algorithm

To use the methods in the Arrays class

To learn how to pass arguments to the main method from the command line

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;
```

// C style is allowed, but **not preferred**

`datatype arrayRefVar[];`

– 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.

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 the element type:

- **0** for the numeric primitive data types
- **'\u0000'** for char types
- **false** for boolean types

Indexed Variables

Array elements are accessed through the index .

Array index starts from 0 to arrayRefVar.length-1.

- Example： ten double values； indices are from 0 to 9.

Element is represented using an *indexed variable*:

- arrayRefVar[index];
- can be used in the same way as a regular variable

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

Array Initializers

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

Declaring, creating, initializing in one step:

This shorthand syntax must be in one statement.

- Splitting it would cause a syntax **error**:

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

Fix it:

```
double[] myList = new double[4];
myList[0] = 1.9;
myList[1] = 2.9;
myList[2] = 3.4;
myList[3] = 3.5;
```

Processing Arrays

Often use a **for** loop—for two reasons:

- All elements are of the same type.
- array size is known

See the examples in the text.

1. (Initializing arrays with input values)
2. (Initializing arrays with random values)
3. (Printing arrays)
4. (Summing all elements)
5. (Finding the largest element)
6. (Finding the smallest index of the largest element)
7. (*Random shuffling*)
8. (*Shifting elements*)

Summing all elements

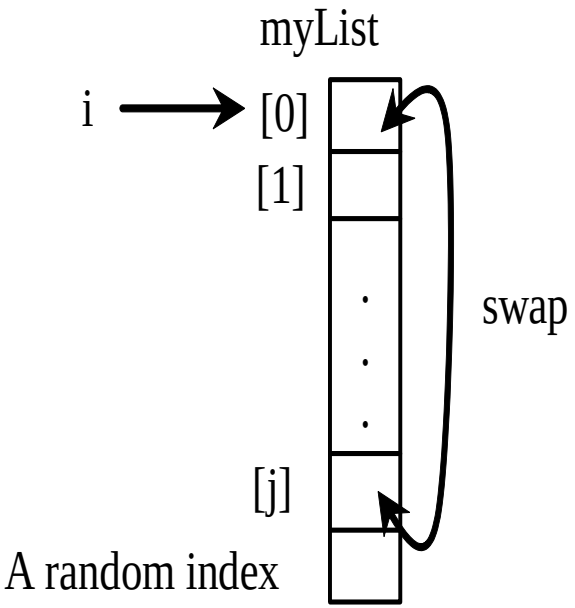
```
double total = 0;
for (int i = 0; i < myList.length; i++) {
  total += myList[i];
}
```

Finding the largest element

```
double max = myList[0];
for (int i = 1; i < myList.length; i++) {
  if (myList[i] > max) max = myList[i];
}
```

Random shuffling

```
for (int i = 0; i < myList.length; i++) {
  // Generate an index j randomly
  int j = (int)(Math.random()*myList.length);

  // Swap myList[i] with myList[j]
  double temp = myList[i];
  myList[i] = myList[j];
  myList[j] = temp;
}
```



Enhanced for Loop (for-each loop)

- Introduced from JDK 1.5
- To **traverse** the array **sequentially without using an index variable**.

For example, the following code displays all elements in the array myList:

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

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.

Problem: Deck of Cards

Picks four cards randomly from a deck of 52 cards. Each card has its **card suit and its rank.**

String[] suits = {**"Spades"**, **"Hearts"**, **"Diamonds"**, **"Clubs"**};

String[] ranks = {**"Ace"**, **"2"**, **"3"**, **"4"**, **"5"**, **"6"**, **"7"**, **"8"**, **"9"**, **"10"**, **"Jack"**, **"Queen"**, **"King"**};

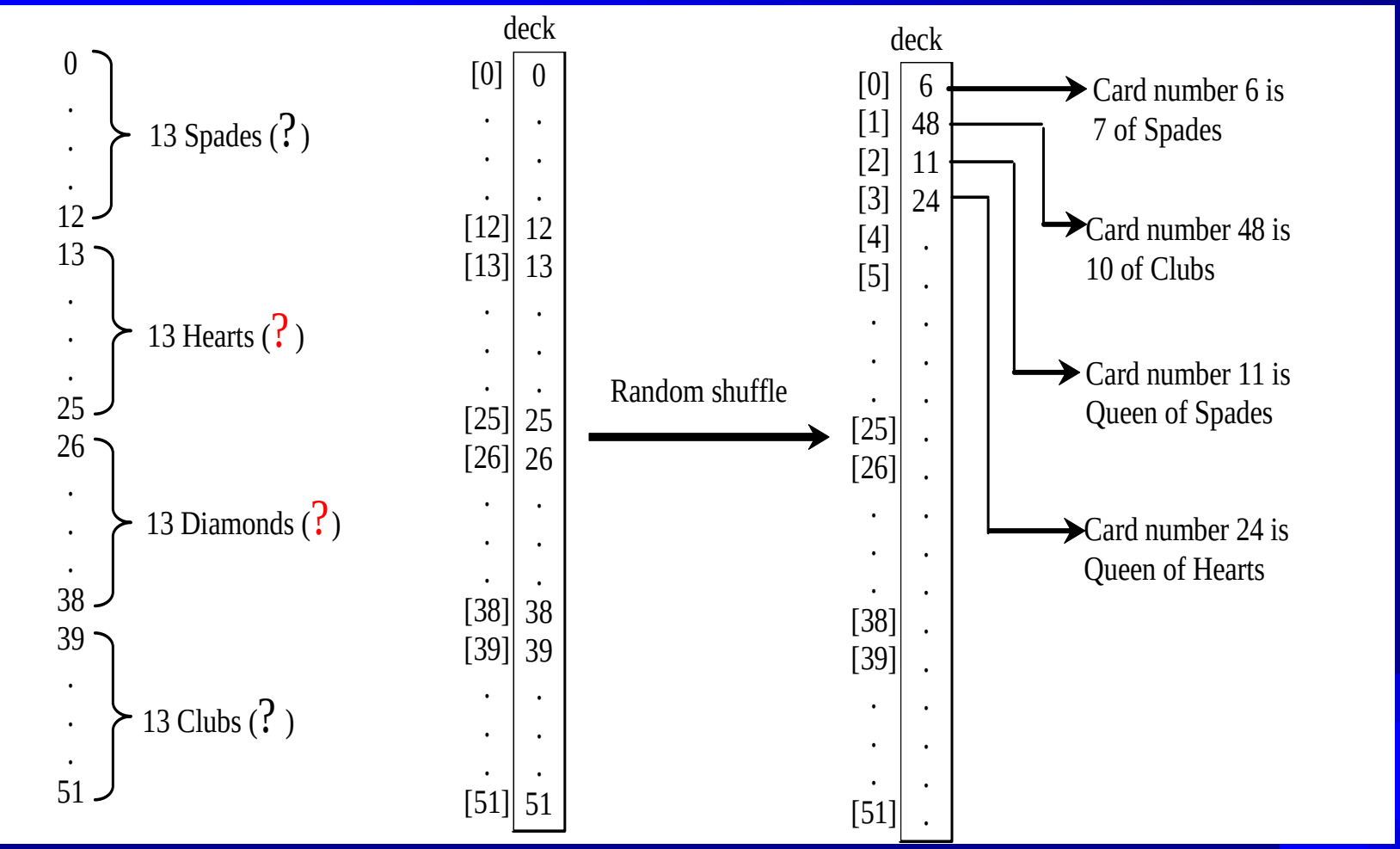
All the cards can be represented using an array named deck, filled with initial values 0 to 51, as follows (array index):

- Card suit: **cardNumber / 13**
- Card rank: **cardNumber % 13**

After shuffling the array **deck**, pick the first four cards from **deck**.

Card suit: **cardNumber / 13**

Card rank: **cardNumber % 13**




```
Card number 6: 7 of Spades
Card number 48: 10 of Clubs
Card number 11: Queen of Spades
Card number 24: Queen of Hearts
```

DeckOfCards.java

```java
public class DeckOfCards {
  public static void main(String[] args) {
    int[] deck = new int[52];
    String[] suits = {"Spades", "Hearts", "Diamonds", "Clubs"};
    String[] ranks = {"Ace", "2", "3", "4", "5", "6", "7", "8", "9",
      "10", "Jack", "Queen", "King"};

    // Initialize cards
    for (int i = 0; i < deck.length; i++)
      deck[i] = i;

    // Shuffle the cards
    for (int i = 0; i < deck.length; i++) {
      // Generate an index randomly
      int index = (int)(Math.random() * deck.length);
      int temp = deck[i];
      deck[i] = deck[index];
      deck[index] = temp;
    }

    // Display the first four cards
    for (int i = 0; i < 4; i++) {
      String suit = suits[deck[i] / 13];
      String rank = ranks[deck[i] % 13];
      System.out.println("Card number " + deck[i] + ": "
        + rank + " of " + suit);
    }
  }
}
```

create array **deck**

array of strings

array of strings

initialize deck

shuffle deck

suit of a card

rank of a card

Problem: Deck of Cards

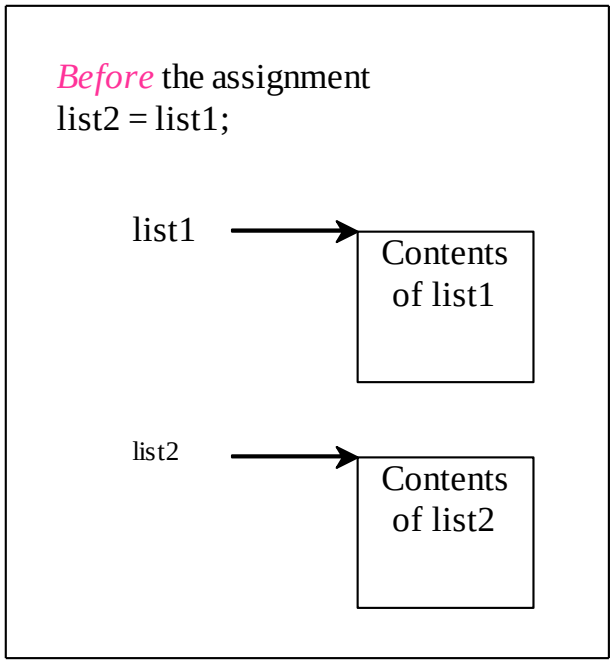
This problem builds a foundation for **future** more interesting and realistic **applications**:

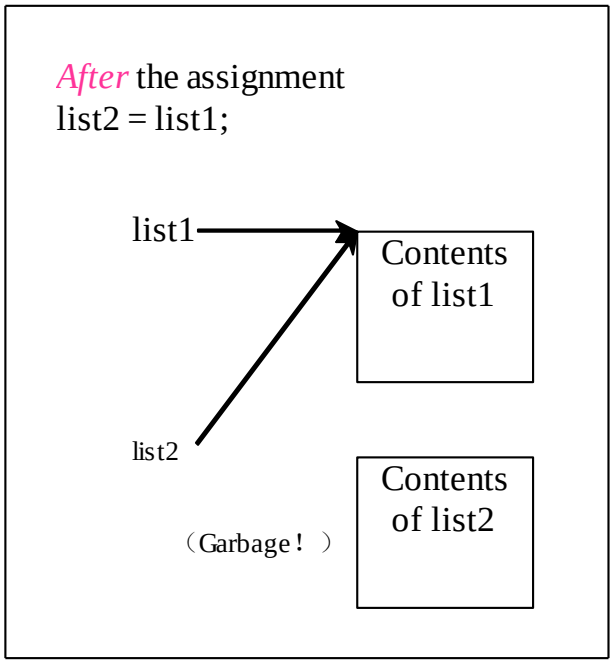
Copying Arrays

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;

not copy the contents of the array referenced by **list1** to **list2**
but merely copies the reference value from **list1** to **list2**



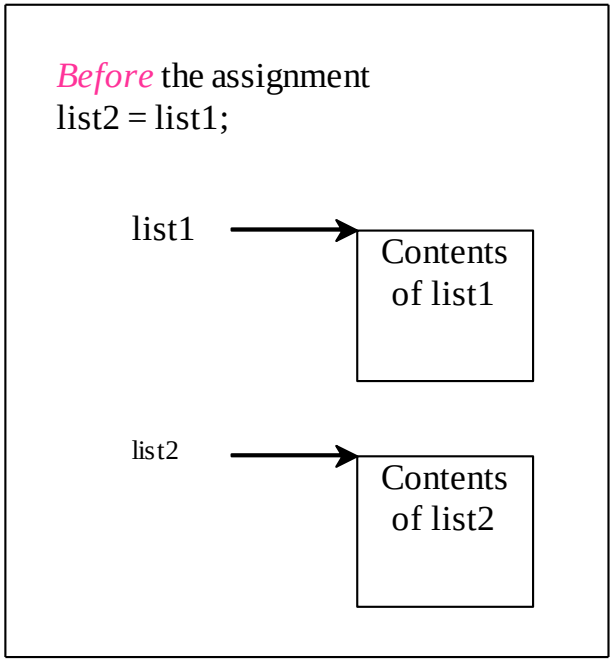


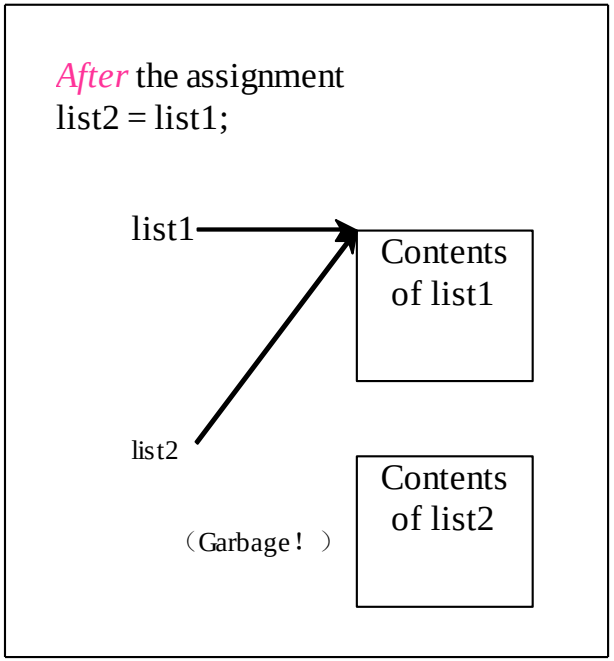
Garbage Collection :

List2 would be **garbage** because it is **no longer referenced.**

- Your program would compile and run correctly, but it would create an array **unnecessarily.**
- **Java automatically collects garbage** behind the scenes.

  Slow





Copying Arrays: three ways

1. **Using a loop** to copy elements one by one

```
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];
```

Copying Arrays: three ways

2. Use the static **arraycopy** method in the **System** class.

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

src_pos, tar_pos: starting positions in **two arrays**, respectively.
length: number of elements copied
targetArray: must have already been created

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

no explicit reference variable for the array

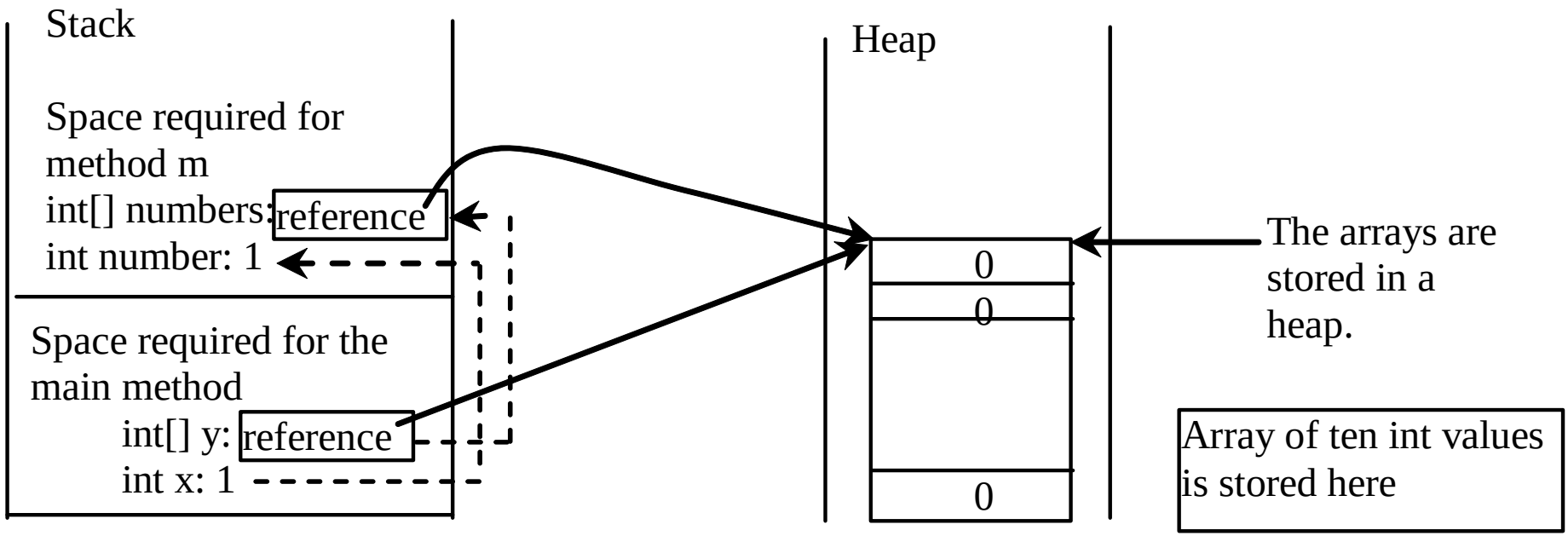
new elementType[]{value0, value1, ..., valuek};

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]
  }
}
```

```
x is 1
y[0] is 5555
```

Call Stack



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.

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： Passing Arrays as Arguments

Differences of passing primitive data type variables and array variables.

TestPassArray.java

```
public class TestPassArray {
  /** Main method */
  public static void main(String[] args) {
    int[] a = {1, 2};

    // Swap elements using the swap method
    System.out.println("Before invoking swap");
    System.out.println("array is {" + a[0] + ", " + a[1] + "}");
    swap(a[0], a[1]);
    System.out.println("After invoking swap");
    System.out.println("array is {" + a[0] + ", " + a[1] + "}");

    // Swap elements using the swapFirstTwoInArray method
    System.out.println("Before invoking swapFirstTwoInArray");
    System.out.println("array is {" + a[0] + ", " + a[1] + "}");
    swapFirstTwoInArray(a);
    System.out.println("After invoking swapFirstTwoInArray");
    System.out.println("array is {" + a[0] + ", " + a[1] + "}");
  }

  /** Swap two variables */
  public static void swap(int n1, int n2) {
    int temp = n1;
    n1 = n2;
    n2 = temp;
  }

  /** Swap the first two elements in the array */
  public static void swapFirstTwoInArray(int[] array) {
    int temp = array[0];
    array[0] = array[1];
    array[1] = temp;
  }
}
```

false swap

swap array elements

Example, cont.

```
Before invoking swap
array is {1, 2}
After invoking swap
array is {1, 2}
Before invoking swapFirstTwoInArray
array is {1, 2}
After invoking swapFirstTwoInArray
array is {2, 1}
```

animation

Returning an Array from a Method

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

```
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;
}
```

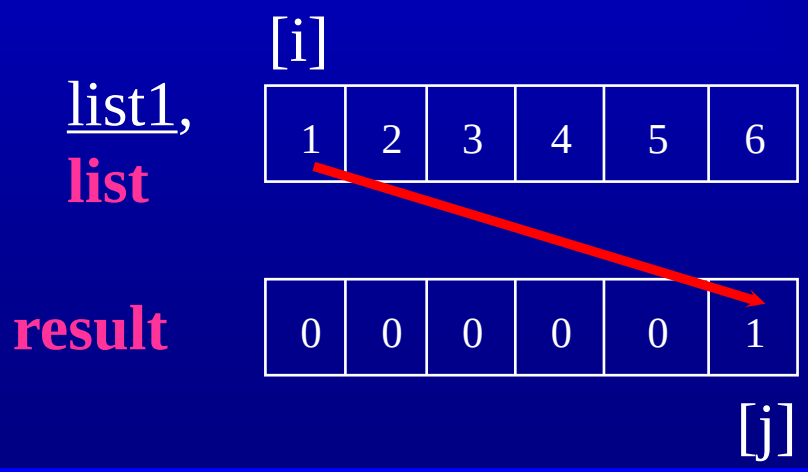
animation

Returning an Array from a Method

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

```
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;
}
```

i = 0 and j = 5
Assign list[0] to result[5]



animation

Trace the reverse Method, cont.

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

```
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;
}
```

After this, i becomes 1 and j becomes 4

list	1	2	3	4	5	6

result	0	0	0	0	0	1

animation

Trace the reverse Method, cont.

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

```
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;
}
```

i (=1) is less than 6

list	1	2	3	4	5	6
result	0	0	0	0	0	1

animation

Trace the reverse Method, cont.

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

```
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;
}
```

i = 1 and j = 4
Assign list[1] to result[4]

list	1	2	3	4	5	6
result	0	0	0	0	2	1

animation

Trace the reverse Method, cont.

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

```
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;
}
```

i = 5 and j = 0
Assign list[i] to result[j]

list	1	2	3	4	5	6
result	6	5	4	3	2	1

animation

Trace the reverse Method, cont.

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

```
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;
}
```

i (=6) < 6 is false. So exit the loop.

list	1	2	3	4	5	6

result	6	5	4	3	2	1

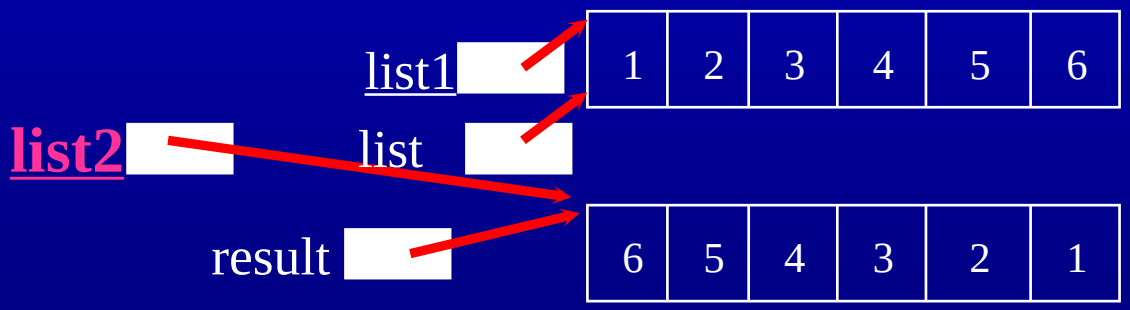
animation

Trace the reverse Method, cont.

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

```
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;
}
```

Return result



Problem: Counting Occurrence of Each Letter

Generate 100 lowercase letters randomly

Count the occurrence of each letter

```
The lowercase letters are:
e y l s r i b k j v j h a b z n w b t v
s c c k r d w a m p w v u n q a m p l o
a z g d e g f i n d x m z o u l o z j v
h w i w n t g x w c d o t x h y v z y z
q e a m f w p g u q t r e n n w f c r f

The occurrences of each letter are:
5 a 3 b 4 c 4 d 4 e 4 f 4 g 3 h 3 i 3 j
2 k 3 l 4 m 6 n 4 o 3 p 3 q 4 r 2 s 4 t
3 u 5 v 8 w 3 x 3 y 6 z
```

Problem: Counting Occurrence of Each Letter

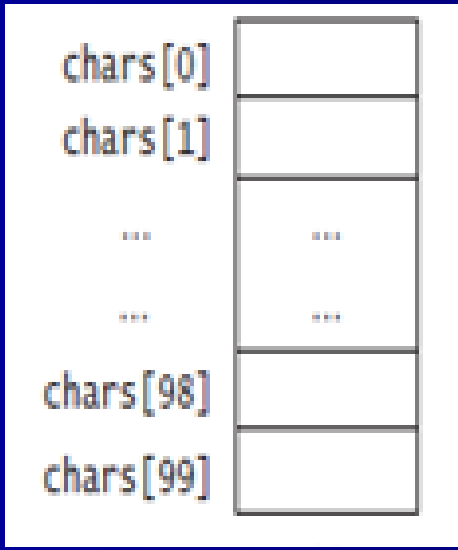
Generate 100 lowercase letters randomly

- Obtain a random letter :

  （Listing 6.10 implemented）

  RandomCharacter .getRandomLowerCaseLetter()

- Assign 100 letters to a **chars** array




```
char[] chars = new char[100];

// Create lowercase letters randomly and assign
// them to the array
for (int i = 0; i < chars.length; i++)
  chars[i] = RandomCharacter.getRandomLowerCaseLetter();
```

CountLettersInArray.java

```java
public class CountLettersInArray {
  /** Main method */
  public static void main(String[] args) {
    // Declare and create an array
    char[] chars = createArray();

    // Display the array
    System.out.println("The lowercase letters are:");
    displayArray(chars);

    // Count the occurrences of each letter
    int[] counts = countLetters(chars);

    // Display counts
    System.out.println();
    System.out.println("The occurrences of each letter are:");
    displayCounts(counts);
  }

  /** Create an array of characters */
  public static char[] createArray() {
    // Declare an array of characters and create it
    char[] chars = new char[100];

    // Create lowercase letters randomly and assign
    // them to the array
    for (int i = 0; i < chars.length; i++)
      chars[i] = RandomCharacter.getRandomLowerCaseLetter();

    // Return the array
    return chars;
  }

  /** Display the array of characters */
  public static void displayArray(char[] chars) {
    // Display the characters in the array 20 on each line
    for (int i = 0; i < chars.length; i++) {
      if ((i + 1) % 20 == 0)
        System.out.println(chars[i]);
```

create array

pass array

return array

pass array

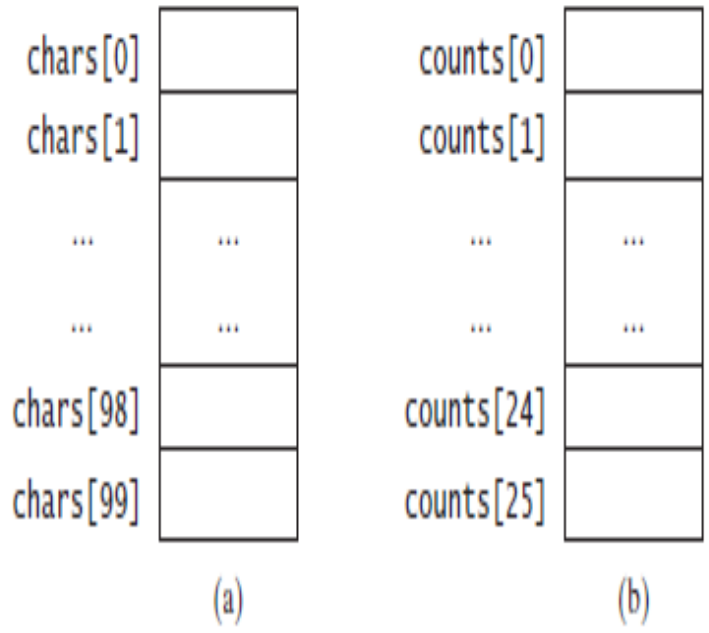
```java
/** Create an array of characters */
public static char[] createArray() {
  // Declare an array of characters and create it
  char[] chars = new char[100];

  // Create lowercase letters randomly and assign
  // them to the array
  for (int i = 0; i < chars.length; i++)
    chars[i] = RandomCharacter.getRandomLowerCaseLetter();

  // Return the array
  return chars;
}

/** Display the array of characters */
public static void displayArray(char[] chars) {
  // Display the characters in the array 20 on each line
  for (int i = 0; i < chars.length; i++) {
    if ((i + 1) % 20 == 0)
      System.out.println(chars[i]);
```

Problem: Counting Occurrence of Each Letter

Count the occurrence of each letter in the chars array by using another **counts** array

- process each letter in the array and increases its count by one.
- How?

```
for (int i = 0; i < chars.length; i++)
  if (chars[i] == 'a')
    counts[0]++;
  else if (chars[i] == 'b')
    counts[1]++;
  ...
```

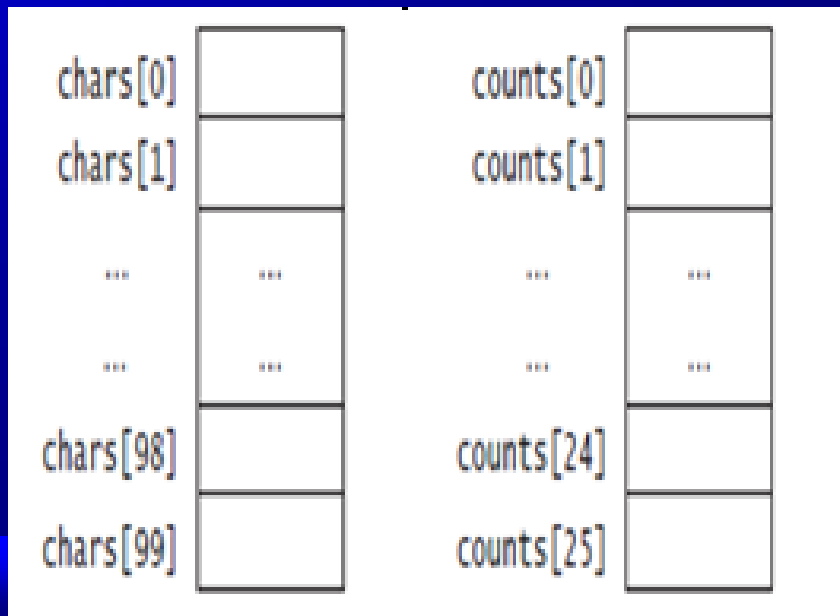


The chars array stores 100 characters, and the counts array stores 26 counts, each of which counts the occurrences of a letter.

```
for (int i = 0; i < chars.length; i++)
  counts[chars[i] - 'a']++;
```

If the letter (`chars[i]`) is `'a'`, the corresponding count is `counts['a' - 'a']` (i.e., `counts[0]`). If the letter is `'b'`, the corresponding count is `counts['b' - 'a']` (i.e., `counts[1]`), since the Unicode of `'b'` is one more than that of `'a'`. If the letter is `'z'`, the corresponding count is `counts['z' - 'a']` (i.e., `counts[25]`), since the Unicode of `'z'` is 25 more than that of `'a'`.

```
for (int i = 0; i < chars.length; i++)
  if (chars[i] == 'a')
    counts[0]++;
  else if (chars[i] == 'b')
    counts[1]++;
  ...
```



Variable-Length Argument Lists

To pass a **variable number of arguments** of the same type to a method.

printMax(3, 5, 7);

printMax(34, 55, 43, 78, 9, 45);

```
public static void main(String[] args) {
  printMax(34, 3, 3, 2, 56.5);

}

public static void printMax(double... numbers) {
  if (numbers.length == 0) {
    System.out.println("No argument passed");
    return;
  }
```

Variable-Length Argument Lists

To pass a variable number of arguments of the same type to a method.

In the method declaration, the parameters:

typeName... parameterName

the type followed by an ellipsis

only one variable-length parameter in a method

must be the last parameter.

Java treats a variable-length parameter as an array.

When invoking a method, argument can be

- an argument list
- an array

VarArgsDemo.java

```
1 public class VarArgsDemo {
2   public static void main(String[] args) {
3     printMax(34, 3, 3, 2, 56.5);
4     printMax(new double[]{1, 2, 3});
5   }
6
7   public static void printMax(double... numbers) {
8     if (numbers.length == 0) {
9       System.out.println("No argument passed");
10      return;
11    }
12
13    double result = numbers[0];
14
15    for (int i = 1; i < numbers.length; i++)
16      if (numbers[i] > result)
17        result = numbers[i];
18
19    System.out.println("The max value is " + result);
20  }
21 }
```

Searching Arrays

looking for a specific element in an array

–Searching is a common task in computer programming.

There are many algorithms and data structures devoted to searching.

–two commonly used approaches :

linear search and *binary search*.

Linear Search

Compares the key ***sequentially*** with each element in the array

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, returns the index of the element in the array

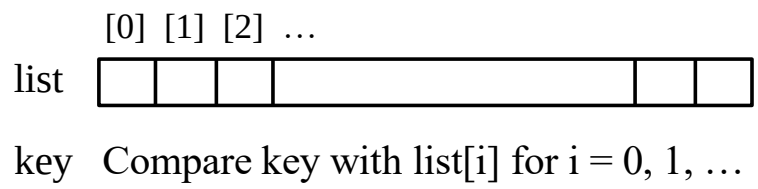
–If no match is found, the search returns -1.

```
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
```

Linear Search

Compares the key ***sequentially*** with each element in the array

```
public class LinearSearch {
  /** 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;
  }
}
```



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 ***java.util.Arrays.binarySearch()*** method

in the ***java.util.Arrays* class**.

array must be **pre-sorted** in **increasing/ascending order**

– found: return the index that matches the key

```
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**

The ***java.util.Arrays.binarySearch()*** method

array must be **pre-sorted** in **increasing/ascending order**

– found: return the index that matches the key

–Not found: **return -insertion point - 1**

- insertion point indicates where the key would be inserted.

```
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**,
so return is **-3-1**)

* Java provides several overloaded methods for searching a key in an array of **int, char, short, long, float, double**

The *java.util.Arrays.sort()* Method

Java provides several overloaded sort methods for sorting an array of **int, char, short, long, and float and double**

```
double[] numbers = {6.0, 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);
```

Command-Line Arguments

Main Method Is Just a Regular Method

How to pass arguments to main?

For example, the main method in class B is invoked by a method in A, as shown below:

```
public class A {
  public static void main(String[] args) {
    String[] strings = {"New York",
      "Boston", "Atlanta"};
    B.main(strings);
  }
}
```

```
class B {
  public static void main(String[] args) {
    for (int i = 0; i < args.length; i++)
      System.out.println(args[i]);
  }
}
```

Command-Line Parameters

```
class TestMain {
  public static void main(String[] args) {
   ...
   }
}
```

```
java TestMain arg0 arg1 arg2 ... Argn
```

For example:

java TestMain "First num" alpha 53

In the main method, get the arguments from

`args[0], args[1], ..., args[n]`**, which corresponds to** `arg0, arg1, ..., argn` **in the command line.**

Problem: Calculator

The program takes three arguments (operand1 operator operand2) from the command line and displays the expression and the result of the arithmetic operation.

Add →

Subtract →

Multiply →

Divide →

Command Prompt

```
c:\book>java Calculator
Usage: java Calculator operand1 operator operand2

c:\book>java Calculator 63 + 40
63 + 40 = 103

c:\book>java Calculator 63 - 40
63 - 40 = 23

c:\book>java Calculator 63 "*" 40
63 * 40 = 2520

c:\book>java Calculator 63 / 40
63 / 40 = 1

c:\book>
```

On a command line:

- `*` refers to all the files in the current directory
- `"*"` refers to multiplication operator

args[] : strings passed to the main program

- operands : **args[0] , args[2]**
- Operator: **args[1].**

args.length : number of strings passed

- Used to determine whether three arguments have been provided in the command line.
- If not, terminate the program using **System.exit(0).**

Calculator.java

```
1  public class Calculator {
2    /** Main method */
3    public static void main(String[] args) {
4      // Check number of strings passed
5      if (args.length != 3) {
6        System.out.println(
7          "Usage: java Calculator operand1 operator operand2");
8        System.exit(0);
9      }
10
11     // The result of the operation
12     int result = 0;
13
14     // Determine the operator
15     switch (args[1].charAt(0)) {
16       case '+': result = Integer.parseInt(args[0]) +
17                          Integer.parseInt(args[2]);
18                 break;
19       case '-': result = Integer.parseInt(args[0]) -
20                          Integer.parseInt(args[2]);
21                 break;
22       case '*': result = Integer.parseInt(args[0]) *
23                          Integer.parseInt(args[2]);
24                 break;
25       case '/': result = Integer.parseInt(args[0]) /
26                          Integer.parseInt(args[2]);
27     }
28
29     // Display result
30     System.out.println(args[0] + ' ' + args[1] + ' ' + args[2]
31       + " = " + result);
32   }
33 }
```

`Integer.parseInt(args[0])` (line 16) converts a digital string into an integer. The string must consist of digits. If not, the program will terminate abnormally.