JAVA PROGRAMMING

Week 2: More Data Types and Operators

Lecturer:

- Hồ Tuấn Thanh, M.Sc.

Plan

1. Arrays
2. Strings
3. Operators

Plan

1. Arrays
2. Strings
3. Operators

Array

- An array is a collection of variables of the same type, referred to by a common name.
- In Java, arrays can have one or more dimensions, although the one-dimensional array is the most common.
- Arrays are used for a variety of purposes because they offer a convenient means of grouping together related variables.

One-Dimensional Arrays

- General form:

```
type arrayname[] = new type[size];
```

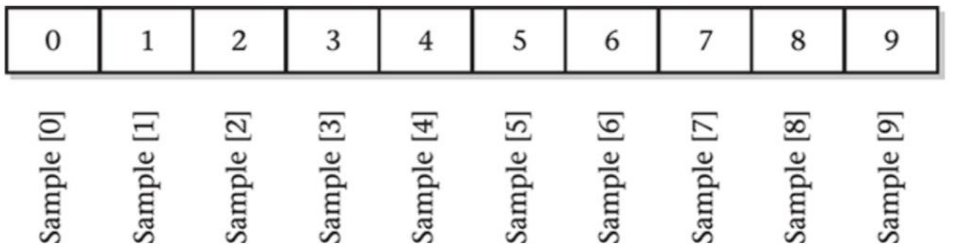
- type declares the element type of the array. The element type determines the data type of each element contained in the array.
- The number of elements that the array will hold is determined by size.
- The creation of an array is a two-step process
  - declare an array reference variable.
  - allocate memory for the array, assigning a reference to that memory to the array variable.

  → arrays in Java are dynamically allocated using the new operator.

Example


```
1.  // Demonstrate a one-dimensional array.
2.  public class ArrayDemo {
3.      public static void main(String[] args) {
4.          int sample[] = new int [10];
5.          int i;
6.          for(i = 0; i < 10 ; i++) sample[i] = i;
7.          for(i = 0; i < 10; i++) {
8.              System.out.println("sample[" + i +
9.                                 "] = " + sample[i]);
10.         }
11.     }
12. }
```

```
sample[0] = 0
sample[1] = 1
sample[2] = 2
sample[3] = 3
sample[4] = 4
sample[5] = 5
sample[6] = 6
sample[7] = 7
sample[8] = 8
sample[9] = 9
```



```
1.  public class MinMax {
2.      public static void main(String[] args) {
3.          int n = 10;
4.          int nums[] = new int[n];
5.          int min, max;
6.          nums[0] = 99; nums[1] = -10; nums[2] = 100123;
7.          nums[3] = 18; nums[4] = -978; nums[5] = 5623;
8.          nums[6] = 463; nums[7] = -9; nums[8] = 287;
9.          nums[9] = 49; min = max = nums[0];
10.         for(int i = 1; i < n; i++) {
11.             if(nums[i] < min) min = nums[i];
12.             if(nums[i] > max) max = nums[i];
13.         }
14.         System.out.println("Min and max: " + min
15.                                                 + " " + max);
16.     }
17. }
```

type arrayname[] = { val1, val2, val3, ... , valN };

```
1.  public class MinMax2 {
2.      public static void main(String[] args) {
3.          int n = 10;
4.          int nums[] = {99, -10, 100123, 18, -978,
5.          5623, 463, -9, 287, 49};
6.          int min, max;
7.          min = max = nums[0];
8.          for(int i = 1; i < nums.length; i++) {
9.              if(nums[i] < min) min = nums[i];
10.             if(nums[i] > max) max = nums[i];
11.         }
12.         System.out.println("Min and max: " + min
13.                                 + " " + max);
14.     }
15. }
```

```
1.  public class ArrayErr {
2.      public static void main(String[] args) {
3.          int sample[] = new int[10];
4.          int i;
5.          //generate an array overrun
6.          for(i = 0; i < 100; i++) {
7.              sample[i] = i;
8.          }
9.      }
10. }
```

→ an ArrayIndexOutOfBoundsException is generated and the program is terminated.

MULTIDIMENSIONAL ARRAYS

10

Two-Dimensional Arrays

- Multidimensional array is an array of arrays.
- Two-dimensional array is the simplest form of the multidimensional array.
  - A list of one-dimensional arrays.
- Example:

```
int table[][] = new int[10][20];
```

Example


```
1.  int table[][] = new int[3][4];
2.  for(int i = 0; i < 3; ++i) {
3.      for(int j = 0; j < 4; ++j) {
4.          table[i][j] = (i*4) + j + 1;
5.          System.out.print(table[i][j] + "  ");
6.      }
7.      System.out.println();
8.  }
```

Irregular Arrays

- When allocating memory for a multidimensional array:
  - Need to specify only the memory for the first (leftmost) dimension.
- It is possible to allocate the remaining dimensions separately.
  - Example: when you allocate dimensions separately, you do not need to allocate the same number of elements for each index.
- Since multidimensional arrays are implemented as arrays of arrays, the length of each array is under your control.

```
1.  ...
2.  int riders[][] = new int[7][];
3.  // The second dimensions are 10 elements long
4.  riders[0] = new int[10]; riders[1] = new int[10];
5.  riders[2] = new int[10]; riders[3] = new int[10];
6.  riders[4] = new int[10];
7.  // The second dimensions are 2 elements long
8.  riders[5] = new int[2]; riders[6] = new int[5];
9.  int i, j;
10. //fabricate some fake data
11. for(i = 0; i < 5; i++)
12.             for(j = 0; j < 10; j++) riders[i][j] = i + j + 10;
13. for(i = 5; i < 7; i++)
14.             for(j = 0; j < 2; j++) riders[i][j] = i + j + 10;
15. ...
```

```
1.  ...
2.  System.out.println("Riders per trip during the week:");
3.  for(i = 0; i < 5; i++) {
4.          for(j = 0; j < 10; j++)
5.                  System.out.print(riders[i][j] + " ");
6.          System.out.println();
7.  }
8.  System.out.println();
9.  System.out.println("Riders per trip on the weekend:");
10. for(i = 5; i < 7; i++) {
11.         for(j = 0; j < 2; j++)
12.                 System.out.print(riders[i][j] + " ");
13.         System.out.println();
14. }
```

Arrays of three or more dimensions

- Java allows arrays with more than two dimensions.
- General form:

```
type name[ ][ ]...[ ] = new type[size1][size2]...[sizeN];
```

- Example:

```
int multidim[][][] = new int[4][10][3];
```

Initializing multi-dimensional arrays


```
1.  typespecifier array_name[] [] = {
2.              { val, val, val, ..., val },
3.              { val, val, val, ..., val },
4.              .
5.              .
6.              .
7.              { val, val, val, ..., val }
8.  };
```

Example

```
1.  // Initialize a two-dimensional array.
2.  public class Squares {
3.      public static void main(String[] args) {
4.          int sqrs[][] = {
5.              {1, 1},{2, 4},{3, 9},{4, 16},
6.              {5, 25},{6, 36},{7, 49},{8, 64},
7.              {9, 81},{10, 100}
8.          };
9.          for(int i = 0; i < 10; i++) {
10.             for(int j = 0; j < 2; j++)
11.                 System.out.print(sqrs[i][j] + " ");
12.                 System.out.println();
13.         }
14.     }
15. }
```

```
4 16
5 25
6 36
7 49
8 64
9 81
10 100
```

Alternative array declaration

type[] varname;

- Example:

```
int counter[] = new int[3];
int[] counter = new int[3];
char table[][] = new char[3][4];
char[][] table = new char[3][4];
int [] nums, nums2, nums3;// create three arrays
int nums[], nums2[], nums3[]; // also, create three arrays
int[] someMethod() {..}
```

Assigning array references

- When you assign one array reference variable to another, you are simply changing what object that variable refers to.
- You are not causing a copy of the array to be made, nor are you causing the contents of one array to be copied to the other.
- Example: AssignARef.java

```
Here is nums1: 0 1 2 3 4 5 6 7 8 9
Here is nums2: 0 -1 -2 -3 -4 -5 -6 -7 -8 -9
Here is nums2 after assignment: 0 1 2 3 4 5 6 7 8 9
Here is nums1 after change through nums2: 0 1 2 99 4 5 6 7 8 9
```

Using length

- Recall: In Java, arrays are implemented as objects.

→Benefit: each array has associated with it a length instance variable that contains the number of elements that the array can hold.

- Example: LengthDemo.java (See next slide)

```
1.  int list[] = new int[10];
2.  int nums[] = {1, 2, 3};
3.  int table[][] = { {1, 2, 3},{4, 5}, {6, 7, 8, 9} };
4.  System.out.println("Length of list is " + list.length);
5.  System.out.println("Length of nums is " + nums.length);
6.  System.out.println("Length of table is " + table.length);
7.  System.out.println("Length of table[0] is " + table[0].length);
8.  System.out.println("Length of table[1] is " + table[1].length);
9.  System.out.println("Length of table[2] is " + table[2].length);
10. System.out.println();
11. // Use length to initialize list
12. for(int i = 0; i < list.length; i++)
13.             list[i] = i * i;
14. System.out.print("Here is list: ");
15. for(int i = 0; i < list.length; i++)
16.             System.out.print(list[i] + " ");
17. System.out.println();
```

```
Length of list is 10
Length of nums is 3
Length of table is 3
Length of table[0] is 3
Length of table[1] is 2
Length of table[2] is 4

Here is list: 0 1 4 9 16 25 36 49 64 81
```

The for-each style for loop

for(type itr-var : collection) statement-block

- **type** specifies the type, and
- **itr-var** specifies the name of an iteration variable that will receive the elements from a collection, one at a time, from beginning to end.
- The collection being cycled through is specified by **collection**.
- With each iteration of the loop, the next element in the collection is retrieved and stored in **itr-var**.
- The loop repeats until all elements in the collection have been obtained.

→ When iterating over an array of size N, the enhanced for obtains the elements in the array in index order, from 0 to N–1.

Example


```
1.  // Use a for-each style for loop
2.  public class ForEach {
3.      public static void main(String[] args) {
4.          int nums[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
5.          int sum = 0;
6.          // Use for-each style for to display and sum the values.
7.          for(int x : nums) {
8.              System.out.println("Value is: " + x);
9.              sum += x;
10.         }
11.         System.out.println("Summation: " + sum);
12.     }
13. }
```

```
1.  public class NoChange {
2.      public static void main(String[] args) {
3.          int nums[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
4.          // Use for-each style for to display and sum the values.
5.          for(int x : nums) {
6.              System.out.print(x + " ");
7.              x = x * 10; // no effect on nums
8.          }
9.          System.out.println();
10.         for(int x : nums) {
11.             System.out.print(x + " ");
12.         }
13.         System.out.println();
14.     }
15. }
16.
```

```
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
```

Iterating over multi-dimensional arrays

- Enhanced for also works on multidimensional arrays.
- Remember: Multi-dimensional arrays consist of arrays of arrays.
- In general, when using the foreach for to iterate over an array of N dimensions, the objects obtained will be arrays of N–1 dimensions.

Example


```
1.  int sum = 0;
2.  int nums[][] = new int[3][5];
3.  // give nums some values
4.  for(int i = 0; i < 3; i++)
5.      for(int j = 0; j < 5; j++)
6.          nums[i][j] = (i+1)*(j+1);
7.  // Use for-each for loop to display and sum the values
8.  for(int x[] : nums) {// Notice how x is declared
9.      for(int y: x) {
10.         System.out.println("Value is: " + y);
11.         sum += y;
12.     }
13. }
14. System.out.println("Summation: " + sum);
```

Applying the Enhanced for


```
1.  int nums[] = {6, 8, 3, 7, 5, 6, 1, 4};
2.  int val = 5;
3.  boolean found = false;
4.  // Use for-each style for to search nums for val.
5.  for(int x : nums) {
6.      if (x == val) {
7.          found = true;
8.          break;
9.      }
10. }
11. if(found)
12.     System.out.println("Value found!");
```

Plan

1. Arrays
2. Strings
3. Operators

Constructing Strings

- Using new and calling the String constructor.
- Constructing a String from another String

Example


```
1.  // Introduce String.
2.  public class StringDemo {
3.      public static void main(String[] args) {
4.          System.out.println("In Java, strings are objects.");
5.          String str1 = new String("Java strings are objects");
6.          String str2 = "They are constructed in various ways.";
7.          String str3 = new String(str2);
8.          System.out.println(str1);
9.          System.out.println(str2);
10.         System.out.println(str3);
11.     }
12. }
```

```
In Java, strings are objects.
Java strings are objects
They are constructed in various ways.
They are constructed in various ways.
```

Operating on Strings

- Operations

boolean equals(*str*)	Returns true if the invoking string contains the same character sequence as *str*.
int length( )	Obtains the length of a string.
char charAt(*index*)	Obtains the character at the index specified by *index*.
int compareTo(*str*)	Returns less than zero if the invoking string is less than *str*, greater than zero if the invoking string is greater than *str*, and zero if the strings are equal.
int indexOf(*str*)	Searches the invoking string for the substring specified by *str*. Returns the index of the first match or –1 on failure.
int lastIndexOf(*str*)	Searches the invoking string for the substring specified by *str*. Returns the index of the last match or –1 on failure.

- String substring(int startIndex, int endIndex)

Arrays of Strings


```
1.  String strs[] = {"This", "is", "a", "test."};
2.  System.out.println("Original array: ");
3.  for(String s: strs)
4.          System.out.print(s + " ");
5.  System.out.println("\n");
6.  // change a string
7.  strs[1] = "was";
8.  strs[3] = "test, too!";
9.  System.out.println("Modified array: ");
10. for(String s: strs)
11.         System.out.print(s + " ");
12. System.out.println("\n");
```

```
Original array:
This is a test.

Modified array:
This was a test, too!
```

Strings are immutable

- The contents of a String object are immutable.
- This restriction allows Java to implement strings more efficiently.
  - When you need a string that is a variation on one that already exists, simply create a new string that contains the desired changes.
  - Unused String objects are automatically garbage collected.
- String reference variables may change the object to which they refer.
  - The contents of a specific String object cannot be changed after it is created.
- Java also supplies StringBuilder and StringBuffer that support strings that can be changed.

Using a string to control a switch statement


```
1.  String command = "cancel";
2.  switch(command) {
3.      case "connect":
4.          System.out.println("Connecting");
5.          break;
6.      case "cancel":
7.          System.out.println("Canceling");
8.          break;
9.      case "disconnect":
10.         System.out.println("Disconnecting");
11.         break;
12.     default:
13.         System.out.println("Command Error!");
            break;
}
```

Using command-line arguments


```
1.  // Display all command-line information
2.  public class CLDemo {
3.
4.      public static void main(String[] args) {
5.          System.out.println("There are " + args.length +
6.                                          " command-line arguments.");
7.          System.out.println("They are: ");
8.          for(int i = 0; i < args.length; i++)
9.              System.out.println("arg[" + i + "]:" + args[i]);
10.     }
11. }
```

Plan

1. Arrays
2. Strings
3. Operators

The bitwise operators

Operator	Result
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Unsigned shift right
<<	Shift left
~	One's complement (unary NOT)

p	q	p & q	p | q	p ^ q	~p
0	0	0	0	0	1
1	0	0	1	1	0
0	1	0	1	1	1
1	1	1	1	0	0

Example


```
1.  // Uppercase letters.
2.  public class UpCase {
3.      public static void main(String[] args) {
4.          char ch;
5.          for(int i = 0; i < 10; i++) {
6.              ch = (char)('a' + i);
7.              System.out.print(ch);
8.              // This statement turns off the 6th bit.
9.              ch = (char)((int)ch & 65503);
10.             // ch is now uppercase
11.             System.out.print(ch + " ");
12.         }
13.     }
14. }
```

```
1.  // Display the bits within a byte.
2.  public class ShowBits {
3.      public static void main(String[] args) {
4.          int t;
5.          byte val;
6.          val = 123;
7.          for(t = 128; t > 0; t = t/2) {
8.              if((val & t) != 0) System.out.print("1 ");
9.              else System.out.print("0 ");
10.         }
11.     }
12. }
```

The Shift Operators

<<	Left shift
>>	Right shift
>>>	Unsigned right shift

value << num-bits

value >> num-bits

value >>> num-bits

- value is the value being shifted by the number of bit positions specified by num-bits.

Bitwise Shorthand Assignments

- x = x ^ 127;
- x ^= 127;

The ? operator


```
if (condition)
        var = expression1;
else
        var = expression2;
```

is equivalent to

```
var = condition ? expression1 : expression2;
```

- Example:

```
// get absolute value of val
absval = val < 0 ? -val : val;
```

```
if(val < 0) absval = - val;
else absval = val;
```

```
1.  // Prevent a division by zero using the ?.
2.  public class NoZeroDiv {
3.      public static void main(String[] args) {
4.          int result;
5.          for(int i = -5; i < 6; i++) {
6.              result = (i != 0) ? (100/i) : 0;
7.              if(i != 0)
8.                  System.out.println("100 / " + i
9.                                          + " is " + result);
10.         }
11.     }
12. }
```

```
1.  public class NoZeroDiv2 {
2.      public static void main(String[] args) {
3.          for(int i = -5; i < 6; i++) {
4.              if((i != 0) ? true : false)
5.                  System.out.println("100 / " + i +
6.                                          " is " +  100/i);
7.          }
8.      }
9.  }
```

QUESTION ?