Object Oriented and Java Programming

Course 3

Qiong Liu

qiong.liu@cyu.fr
CY Tech, CY Cergy Paris University

October 2024

Contents

Outline

Arrays

Definition:

- A collection of elements of the same type, stored in continuous memory locations.
- Arrays are indexed, with the first element starting at index 0.

Characteristics:

- Fixed size: Once an array is created, its size cannot be changed.
- Homogeneous elements: All elements in the array must be of the same type.

Arrays: Failed Examples

- `int[] mixedArray = {1, 2, "Three", 4};`
  Error: Incompatible types. The array must contain only integers, but a string was found.
- `String[] fruits = {"Apple", 123, "Mango"};`
  Error: Incompatible types. The array must contain only strings, but an integer was found.
- `int[] negativeSize = new int[-5];`
  Error: NegativeArraySizeException. Array size must be a positive integer.

Arrays: Successful Examples

- `int[] numbers = {10, 20, 30, 40, 50};`
  Outcome: Successfully creates an array of integers.
- `String[] fruits = {"Apple", "Banana", "Mango"};`
  Outcome: Successfully creates an array of strings.
- `int[][] matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};`
  Outcome: Successfully creates a 2D integer array (matrix).

Array Initialization : Static

Static Initialization:

- Array elements are initialized at the time of declaration.

Example:

```
int[] numbers = {1, 2, 3, 4, 5};
String[] names = {"Alice", "Bob", "Charlie"};
```

Outcome:

- Creates arrays with pre-defined values.

Array Initialization: Dynamic

Dynamic Initialization:

- The size of the array is declared first, and elements are assigned later.

Example:

```
int[] numbers = new int[5];
numbers[0] = 1;
numbers[1] = 2;
numbers[2] = 3;
numbers[3] = 4;
numbers[4] = 5;
```

Outcome:

- The array size is set to 5, and each element is assigned individually.

Default Initialization:

- If no value is assigned, Java assigns default values.
- Numeric types default to 0, object references to `null`.

2D Arrays

Definition:

- A 2D array is essentially an array of arrays, where each element is itself an array.
- It is commonly used to represent matrices or grids.
- Each element in a 2D array is accessed using two indices: row and column.

Syntax:

```
int[][] matrix = {
    {1, 2, 3}, {4, 5, 6}, {7, 8, 9}
};  // Declaring and initializing a 3x3 2D array
System.out.println(matrix[0][0]);  // Access the first
    element (1)
matrix[1][2] = 10;  // Modify the element at row 2,
    column 3
```

2D Arrays

Usage:

- 2D arrays are often used in applications such as game boards, image processing, and tabular data.
- Accessing elements is done using nested loops, for example:

```
for (int i = 0; i < matrix.length; i++) {
    for (int j = 0; j < matrix[i].length; j++) {
        System.out.print(matrix[i][j] + " ");
    }
    System.out.println();  // Print each row on a
        new line
}
```

Array Access

Accessing Array Elements:

- Accessing elements in an array using valid indices.

Example 1: Integer Array

```
int[] numbers = {10, 20, 30, 40, 50};
System.out.println(numbers[0]); // Output: 10
System.out.println(numbers[3]); // Output: 40
```

Example 2: String Array

```
String[] fruits = {"Apple", "Banana", "Mango"};
System.out.println(fruits[2]);  // Output: Mango
```

Outcome:

- Successfully retrieves elements from arrays using valid indices.

Array Access and Modification

Avoiding Common Access Errors:

- Always check the length of the array using `arrayName.length` before accessing elements.
- Use loops to iterate through arrays safely.

Example of Safe Access Using a Loop:

```
int[] numbers = {10, 20, 30, 40, 50};
for (int i = 0; i < numbers.length; i++) {
    System.out.println(numbers[i]);  // Prints all
        elements safely
}
```

Array Modification Syntax

- `arrayName[index] = newValue;`

Example:

```
int[] numbers = {1, 2, 3, 4, 5};
numbers[2] = 10;  // Modify the element at index 2
```

Practices for Array Modification

Modifying Arrays:

- Ensure that you're modifying valid indices (between 0 and `array.length - 1`).
- Use loops to ensure safe modification across arrays.

Example:

```
int[] numbers = {1, 2, 3, 4, 5};
for (int i = 0; i < numbers.length; i++) {
    numbers[i] = numbers[i] * 2;  // Modify each
        element by doubling its value
}
System.out.println(Arrays.toString(numbers));
// Output: [2, 4, 6, 8, 10]
```

Java Arrays: Exercise

Exercise 1: Month Days Array

Create an array that stores the number of days for each of the 12 months (assuming February has 28 days).

a) Initialize the array.

b) Use a loop to print the number of days in each month, along with the month's number.

Exercise 2: Two-Dimensional Array Manipulation

Given a 2D array of integers:

```
int[][] matrix = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}};
```

a) Write a loop to calculate and print the sum of all the elements in the array.

b) Swap the first row with the last row and print the modified array.

Java Arrays: Exercise

Exercise 3: Array Maximum, Minimum, and Average

- Create an integer array of size 10, and initialize it with random values between 1 and 100.
- Write a method that finds and returns the maximum value in the array.
- Write a method that finds and returns the minimum value in the array.
- Write a method to calculate and return the average of all the elements in the array.
- Print the original array, followed by the maximum value, minimum value, and the average of the array elements.

Example Output:

```
Array: [23, 5, 67, 34, 89, 12, 55, 78, 90, 11]
Maximum value: 90
Minimum value: 5
Average value: 46.4
```

Basic Operations for Arrays

- Ergodic
- Fill
- Range
- Binary-search

Array Library and Main Functions in Java

`java.util.Arrays`

- `Arrays.sort(array)`: Sorts the array in ascending order.
- `Arrays.toString(array)`: Returns a string representation of the array.
- `Arrays.fill(array, value)`: Fills the array with the specified value.
- `Arrays.equals(array1, array2)`: Compares two arrays for equality (element-wise).
- `Arrays.copyOf(array, newLength)`: Copies the original array into a new array of specified length.
- `Arrays.binarySearch(array, key)`: Performs a binary search for a specified value in a sorted array.

Ergodic

Definition:

- Ergodic, refers to traversing or iterating through all the elements of an array.

Traversing a 1D Array:

- You can traverse an array using loops, typically with a `for` loop.

Syntax:

```
// Using a for loop to traverse an array
int[] numbers = {1, 2, 3, 4, 5};
for (int i = 0; i < 5; i++) {
    System.out.println(numbers[i]);  // Accessing each
        element by index
}
```

Fill Arrays

Definition:

- Filling an array refers to assigning values to all or specific elements of an array, either manually or using built-in methods.

Manual Fill:

```
int[] numbers = new int[5];  // Create an empty array
    with 5 elements
for (int i = 0; i < numbers.length; i++) {
    numbers[i] = i + 1;  // Fill array with values: 1,
        2, 3, 4, 5
}
```

Fill Arrays

Using `Arrays.fill()`:

- Java provides the `Arrays.fill()` method to quickly fill an array with the same value.

```
import java.util.Arrays;
int[] numbers = new int[5];
Arrays.fill(numbers, 42);  // Fill array with value 42
Array.fill(numbers,2,3,43)// Fill array with 43 of
    position 2,3
```

Key Points:

- `Arrays.fill()` can be used to fill all or part of an array.

Query Arrays with Binary Search

Binary Search:

- `Arrays.binarySearch(arr, key)`: Returns the index of `key` in array `arr`, or a negative value if not found.
- `Arrays.binarySearch(Object[] a, int fromIndex, int toIndex, Object key)`: Searches a subarray (from `fromIndex` to `toIndex - 1`) for `key`.

Examples:

- Basic Binary Search:

```
1        int[] arr = {10, 20, 30, 40, 50};
2        int idx = Arrays.binarySearch(arr, 40);
             // idx = 3
```

- Subarray Binary Search:

```
1        Integer[] arr = {10, 20, 30, 40, 50};
2        int idx = Arrays.binarySearch(arr, 1, 4,
             40);  // idx = 3
```

Array Sort Algorithm

- Bubble Sort
- Selection Sort
- Reverse Sort

Bubble Sort

Steps:

- Compare adjacent elements.
- Swap them if they are in the wrong order.
- Repeat the process for all elements.
- Continue the process until no swaps are needed.

Example:

```
// Initial array: {5, 1, 4, 2, 8}
Step 1: {1, 5, 4, 2, 8} // 5 > 1, so swap
Step 2: {1, 4, 5, 2, 8} // 5 > 4, so swap
Step 3: {1, 4, 2, 5, 8} // 5 > 2, so swap
Step 4: {1, 4, 2, 5, 8} // 5 < 8, no swap

// Continue to next iteration
Step 1: {1, 4, 2, 5, 8}
Step 2: {1, 2, 4, 5, 8} // 4 > 2, so swap
Step 3: {1, 2, 4, 5, 8} // No further swaps needed
```

Bubble Sort

Java Code:

```
public class BubbleSort {
    public static void bubbleSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n-1; i++) {
            for (int j = 0; j < n-i-1; j++) {
                if (arr[j] > arr[j+1]) {
                    // Swap arr[j] and arr[j+1]
                    int temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
            }
        }
    }
}
```

Time Complexity: $O(n^2)$ in the worst case.

Selection Sort

Steps:

- In-place comparison-based sorting algorithm.
- Find the minimum (or maximum) element from the unsorted part of the array.
- Swap it with the first unsorted element.
- Move the boundary between sorted and unsorted parts of the array.
- Repeat the process until the entire array is sorted.

Example:

```
// Initial array: {29, 10, 14, 37, 13}
Step 1: {10, 29, 14, 37, 13}
Step 2: {10, 13, 14, 37, 29}
Step 3: {10, 13, 14, 37, 29}
Step 4: {10, 13, 14, 29, 37}
// Array is sorted.
```

Time Complexity: $O(n^2)$ in all cases.

Selection Sort Code Structure - Part 1

```
// Selection sort algorithm
public static void selectionSort(int[] arr) {
    int n = arr.length;
    // Traverse through all elements
    for (int i = 0; i < n-1; i++) {
        // Find the minimum element in unsorted part
        int minIdx = i;
        for (int j = i+1; j < n; j++) {
            if (arr[j] < arr[minIdx]) {
                minIdx = j; // Update min index
            }
        }
        // Swap the found minimum with the first
           element
        int temp = arr[minIdx];
        arr[minIdx] = arr[i];
        arr[i] = temp;
    }
```

Selection Sort Code Structure - Part 2

Main Method:

```
    // Main method to run the algorithm
    public static void main(String[] args) {
        int[] arr = {29, 10, 14, 37, 13};
        selectionSort(arr);

        // Print sorted array
        for (int i : arr) {
            System.out.print(i + " ");
        }
    }
}
```

Output:

- Input: {29, 10, 14, 37, 13}
- Output: 10 13 14 29 37

Reverse Sort- Part 1

Reverse Sort is a sorting method that arranges elements in descending order.

Steps:

- Use a sorting algorithm to sort the array in ascending order.
- Reverse the array by swapping elements manually or using built-in methods.

Java Code:

```
public static void reverseArray(int[] arr) {
    int n = arr.length;
    for (int i = 0; i < n / 2; i++) {
        int temp = arr[i];
        arr[i] = arr[n - 1 - i];
        arr[n - 1 - i] = temp;
    }
}
```

Reverse Sort- Part 2

Note: For sorting arrays in descending order directly, we can use `Arrays.sort()` with `Collections.reverseOrder()` if the array is an `Integer[]` array:

```
Arrays.sort(arr, Collections.reverseOrder());
```

Exercises

Exercise 4: Ascending Order with Bubble Sort

- 10 students participated in an English competition with the following scores: $\{10, 13, 17, 19, 9, 12, 15, 18, 11, 14\}$.
- Write a program using **Bubble Sort** to sort the scores in ascending order.

Exercise 5: Descending Order with Reverse Sort

- After sorting the scores in ascending order, write another program to reverse the sorted array and display the scores in descending order.
- Use either manual reverse logic or `Arrays.sort()` with `Collections.reverseOrder()`.

First Part Finish!

Summary of What We've Learned So Far:

- Java syntax and structure.
- Control flow statements: if-else, switch, loops.
- Arrays and array manipulation.

Next Steps: Object-Oriented Programming (OOP)

- How do we break down bigger problems into smaller pieces using classes and objects, and other OOP principles?

Outline

Class: the Type of Variables

- Member Variables: Defined in a class, belongs to objects (instances).
- Local Variables: Declared in methods, exist during method execution.
- Static Variables: Shared among all objects of a class, belong to the class itself.
- Final Variables: Constants that cannot change once initialized
- Parameter Variables: Passed into methods to receive values.

Class: Member Variables

Member variables belong to a class and store data for objects created from that class.
These variables can be initialized with a value or left uninitialized, in which case they will have a default value.
Default values:

- Numeric types (e.g., `int`, `double`): 0
- Boolean: `false`
- Object references: `null`

Example

```
public class Student {
    public String name = "John";   // Initialized
        member variable
    private int age;                // Default value: 0
}
```

Class: Local Variables

Definition:

- Local variables are declared inside methods, or constructors.
- They are created when the method or block is executed and destroyed when it finishes.
- Local variables must be initialized before use (unlike member variables).

Scope:

- The scope of a local variable is limited to the method where it is defined.
- Local variables cannot have access modifiers.

Example:

```
public class Example {
    public void displayAge() {
        int age = 20;  // Local variable
        System.out.println("Age: " + age);
    }
}
```

Class: Static Variables (Class Variables)

Definition:

- Static variables (also called class variables) are shared among all instances of a class.
- They are declared using the `static` keyword and belong to the class, not any particular instance.

Example:

```
public class Counter {
    public static int count = 0;  // Static variable
    public Counter() {
        count++;
    }
}
```

Access:

- Static variables can be accessed using the class name, like `Counter.count`.

Class: Final Variables

Definition:

- Final variables are constants, meaning their value cannot be changed once initialized.
- Declared using the `final` keyword, they must be initialized either at the time of declaration or in the constructor.

Example:

```
public class Example {
    public static final int MAX_SCORE = 100;  // Final
        variable
    public final String response; //Final but NOT
       static
    public Answer(String response) {
        this.response = response;  // Each object can
            have a unique final value
    }}
```

Class: Parameter Variables

Definition:

- Parameter variables are used in method declarations to receive values when the method is called.
- These variables are local to the method and only exist for the duration of the method execution.

Example

```
public class Example {
    public void setAge(int age) {// Method with
        parameter 'age'
         System.out.println("Age: " + age);
    }}
```

```
public class Example {
    public void greet() { // Method with no parameters
         System.out.println("Hello!");
    }
}
```

Class: Types of Methods

Member Methods:

- Belong to objects (instances).
- Can access and modify instance variables.

Static Methods:

- Belong to the class, not instances.
- Can access static variables and call static methods.

Constructor Methods:

- Initialize objects, have no return type.

Final Methods:

- Cannot be overridden by subclasses.

Class: Member Methods

Definition:

- Member methods (also called instance methods) belong to objects of a class and can access instance variables.
- Member methods can return values using `return`; or return nothing using `void`

Example:

```
public class Car {
    String model;  // Instance variable

    // Member method
    public void displayModel() {
        System.out.println("Car model: " + model);
    }
}
```

Class: Static Methods

Definition:

- Static methods belong to the class, not to any object of the class.
- These methods are defined using the `static` keyword.
- Static methods cannot directly access instance variables but can access static variables.

Example:

```
public class Utility {
    public static void printMessage() {
        System.out.println("This is a static method");
    }
}
```

Key Points:

- Called using the class name (e.g., `Utility.printMessage()`).
- Cannot access non-static (instance) variables or methods directly.

Class: Constructor Methods

Definition:

- A constructor is a special method used to initialize objects.
- It has the same name as the class and no return type (not even `void`).
- Constructors can be overloaded to initialize objects in different ways.

Example:

```
public class Car {
    String model;
    // Constructor method
    public Car(String model) {
        this.model = model;
    }
}
```

Class: Final Methods

Definition:

- A final method is a method that cannot be overridden by subclasses.
- Final methods are declared using the `final` keyword.

Example:

```
public class Car {
    public final void displayInfo() {
        System.out.println("This method cannot be
            overridden.");
    }
}
```

Key Points:

- Prevents subclasses from modifying the method's behavior.
- Useful when you want to secure the method's functionality.

The this (Without this)

Definition:

- The this keyword refers to the current object (instance) of the class.
- It helps to differentiate between instance variables and local parameters when they have the same name.

Example Without this:

```
public class Student {
    String name;
    // Constructor without 'this'
    public Student(String name) {
        name = name; } // Local variable 'name' is
            assigned to itself
    public void displayInfo() {
        System.out.println("Name: " + name); }
    public static void main(String[] args) {
        Student s = new Student("Alice");
        s.displayInfo();}}  // Outputs: Name: null
```

The `this` Keyword (Without `this`)

Key Points:

- Without `this`, the local variable `name` is assigned to itself.
- The instance variable `name` remains uninitialized, resulting in a `null` output.

The this Keyword (With this)

Example With this:

```
public class Student {
    String name;

    // Constructor with 'this'
    public Student(String nameB) {
        this.name = name;  // Instance variable 'name'
            is assigned
    }
    public void displayInfo() {
        System.out.println("Name: " + this.name);  //
            Outputs the correct name
    }
    public static void main(String[] args) {
        Student s = new Student("Alice");
        s.displayInfo();  // Outputs: Name: Alice
    }
}
```

Object: Creation, Access, and Destroy

Object Creation

Object Creation:

- Objects are instances of a class, created using the `new` keyword.
- The constructor method initializes the object.
- Attributes are accessed using the dot operator on the object.

```
public class Car {
    String model;
    // Constructor
    public Car(String model) {
        this.model = model;  }
    public void displayModel() {
        System.out.println("Model: " + model); }
    public static void main(String[] args) {
        // Object creation
        Car car1 = new Car("Sedan");
        // Accessing method
        car1.displayModel();
    }}
```

Object Destruction: Garbage Collection

Object Destruction:

- Objects are destroyed automatically by Java's **garbage collector**.
- When there are no more references to an object, it becomes eligible for garbage collection.

Example:

```
Car car1 = new Car("SUV");  // Object created
car1 = null;  // Object is no longer referenced
// The object is now eligible for garbage collection.
```

Key Points:

- Garbage collection in Java happens automatically, freeing up memory by destroying unused objects.

Exercise: Cinema Ticket System

Exercise 6: You are designing a class for a cinema ticket system where you will use different types of variables and methods.

Question 1: Create the **CinemaTicket** class with the following variables:

- Member Variables: Store the `movieTitle` and `ticketPrice`.
- Static Variable: `totalTicketsSold` to count how many tickets have been sold.
- Final Variable: Store the `maxSeats`, initialized to 100.

Exercise: Cinema Ticket System

Question 2: Implement Four Types of Methods:

- Create a Constructor that initializes the cinema ticket's details and increments the total number of tickets sold.
- Member Method: `printTicket()` outputs the movie title and ticket price.
- Static Method: `getTotalTicketsSold()` returns the total number of tickets sold.
- Final Method: `closeCinema()` outputs a message saying the cinema is closed.

Exercise: Cinema Ticket System

Question 3: Create and Use an Object

- In the `main()` method, create two new instances of the `CinemaTicket` class.
- Call the following:
  - `printTicket()` method to display the ticket information.
  - `getTotalTicketsSold()` method to display how many tickets have been sold.
  - `closeCinema()` method to announce the cinema's closure.