Module III: Methods, Arrays and Recursions

Course: Object oriented Programming in JAVA

Dr. Vishwa Pratap Singh

NextGen Academy

February 10, 2024

NextGen Academy

Towards fulfilling a million dreams

NextGen Academy

Towards fulfilling a million dreams

Defining a Method

- A method is a block of code in Java that performs specific actions when called.
- Primary uses include code reusability, breaking down complex programs, and improving code readability.
- Method declaration includes access specifier, return type, method name, parameter list, method signature, and method body.
- Syntax: `public int addNumbers(int a, int b) { /* method body */ }`

Declare Methods in Java

- Access specifiers: Public, Private, Protected, Default.
- Return type: Defines the type of value the method returns.
- Method name: Gives a unique name to the method.
- Parameter list: Arguments used in the method.
- Method signature: Combination of method name and parameter list.
- Method body: Set of instructions within curly brackets.
- Example: `public int addNumbers(int x, int y) { /* method body */ }`

How to Name a Method?

- Use names corresponding to functionality (e.g., `add()`, `sum()`).
- Start with a verb in lowercase (e.g., `sum()`, `divide()`).
- For multi-word names, use the first word as a verb, followed by a noun or adjective (e.g., `addIntegers()`, `areaOfSquare`).

Calling a Method

- To execute and use a method's functionalities, call it using its name followed by parentheses and a semicolon.
- Syntax: `add();`
- Example: `exMethod();`

Passing Arguments by Values

- Java is strictly pass by value.
- Different mechanisms for passing parameters: value, reference, result, value-result, name.
- Java supports pass by value and does not support pass by reference.

About the Parameters Passed in Java

- In Java, pass by reference concept is not supported.
- Primitive variables hold actual values; non-primitive variables hold reference variables.
- During method invocation, a copy of each argument is created.
- Demonstration using examples with primitive and non-primitive data types.

Java Pass by Value Example

- Example demonstrating that passing parameters by values does not affect the original variable.
- Syntax and output of a simple program illustrating pass by value.

- Definition of Modularity in Java.
- Modules as independent partitions of software.
- Similarities to microservices in MVC architecture.
- Benefits: optimization, reduced coupling.
- Facilitation of functionality testing during development.
- Reduction of development time with testing on the fly.

Example: Creating a modular calculator in Java.

```
public interface MathUtil  // Mathematical operations int add(int
a, int b); int subtract(int a, int b);
```

“begin–frame”–Why Java Modularity?”
“begin–itemize”
“item Reusability: Save time by reusing code.
“item Stability: Maintain software stability during changes.
“item Parallel Development: Develop modules simultaneously.
“item Introduction to Java 11 modularity.
“item Example: Creating a calculator with modules.
“end–itemize”
“textbf–Syntax:” Module declaration in Java 11.
“begin–verbatim”
module calculator –
requires math.util;
”

NextGen Academy
Towards fulfilling a million dreams

Creating Modules - Step 1

- Two distinct modules: Math.util and Calculator.
- Math.util module has an API for mathematical calculations.
- Calculator module launches the calculator.

Syntax: Defining modules in Java.

```
module calculator  requires math.util;
```

Executing Modules - Step 2

- Example of executing API module in the browser.
- Functions in Math.util module: isPrime, isEven, sumOfFirstNEvens, sumOfFirstNOdds.
- Observations from executing functions.
- Refactoring the API using computeFirstNSum.

Example: Executing functions in Math.util module.

```
public static Integer sumOfFirstNEvens(Integer count)  return com
i -> isEven(i));
```

NextGen Academy
Towards fulfilling a million dreams

“begin–frame”–Inserting Utility Class Step 3”

“begin–itemize”

“item Conventions for inserting utility class in the module.

“item Creating moduleinfo.java in the root folder.

“item Placing packages and codes in the root directory.

“end–itemize”

“textbf–Syntax:” Structure of moduleinfo.java.

“begin–verbatim”

module math.util –

exports com.upgrad.math;

”

NextGen Academy

Towards fulfilling a million dreams

Creating Module Definition - Step 4

- Creating a module definition for the Calculator.
- Dependencies with Math.util module.
- Compiling the code and creating dependencies.

Syntax: Compiling code with dependencies.

```
javac -d mods --module-source-path . (find. – name" ∗ .java")
```

NextGen Academy
Towards fulfilling a million dreams

“begin–frame″–Execute the Code Step 5″

“begin–itemize″

“item Having both modules in the mods directory.

“item Executing the code to bring the calculator into action.

“end–itemize″

“textbf–Syntax:″ Running the modularized application.

“begin–verbatim″

java modulepath mods m calculator/com.upgrad.calculator.Calculator

- Definition of Method Overloading.
- Also known as Compile-time Polymorphism.
- Example with the Sum class demonstrating overloading.
- Output of the example.

Syntax: Overloaded methods in Java.

```
public class Sum  public int sum(int x, int y)  return (x + y);
public int sum(int x, int y, int z)  return (x + y + z);
```

“begin–frame”–Different Ways of Method Overloading”
“begin–itemize”
“item Changing the Number of Parameters.
“item Changing Data Types of the Arguments.
“item Changing the Order of the Parameters.
“end–itemize”
“end–frame”
“begin–frame”–Changing the Number of Parameters”
“begin–itemize”
“item Example of method overloading by changing the number of parameters.
“item Implementation with Product class and multiplication.
“item Output of the example.
“end–itemize”
“textbf–Syntax:” Overloading by changing the number of parameters.
“begin–verbatim”
public class Product –
public int multiply(int a int b) – return a * b; ”
public int multiply(int a int b int c) – return a * b * c; ”
”

Changing Data Types of the Arguments

- Example of method overloading by changing data types.
- Implementation with Product class and different data types.
- Output of the example.

Syntax: Overloading by changing data types.

```
public class Product  public int multiply(int a, int b, int c)  r
a * b * c;  public double multiply(double a, double b, double c)
return a * b * c;
```

NextGen Academy
Towards fulfilling a million dreams

“begin–frame”–Changing the Order of the Parameters”

“begin–itemize”

“item Example of method overloading by changing the order of parameters.

“item Implementation with Student class and student ID.

“item Output of the example.

“end–itemize”

“textbf–Syntax:” Overloading by changing the order of parameters.

“begin–verbatim”

```
public class Student –
public void studentId(String name int roll˙no) –
// Implementation
”
public void studentId(int roll˙no String name) –
// Implementation
”
”
```

What if the Exact Prototype does not Match?

- Compiler's steps for handling prototype mismatches.
- Type conversion in the same family.
- Type conversion to the next higher family.
- Example with the Demo class and show method.

Example: Handling prototype mismatches in Java.

```
class Demo  public void show(int x)  /* Implementation */  public
void show(String s)  /* Implementation */
```

NextGen Academy
Towards fulfilling a million dreams

“begin–frame”–Advantages of Method Overloading”
“begin–itemize”
“item Improves readability and reusability.
“item Reduces program complexity.
“item Enables efficient task performance.
“item Access methods with slightly different arguments and types.
“end–itemize”
“textbf–Example:” Constructor overloading for object initialization.
“begin–verbatim”
public class MyClass –
public MyClass(int a) – /* Implementation */ ”
public MyClass(int a int b) – /* Implementation */ ”
”

Member Variables (Class Level Scope)

- Member variables are declared inside a class and are accessible throughout the class.
- They are typically initialized with default values if not explicitly initialized.
- Syntax:

```
public class MyClass  int a; // member variable private String
b; // private member variable public static void main(String[]
args)  // Accessing member variables MyClass obj = new MyClass
obj.a = 10; // assigning value to 'a' System.out.println(obj.a
// accessing 'a'
```

NextGen Academy
Towards fulfilling a million dreams

“begin–frame”–Local Variables (Method Level Scope)”

“begin–itemize”

“item Local variables are declared inside methods and have methodlevel scope.

“item They must be initialized before use.

“item Syntax:

“begin–verbatim”

public class MyClass –

public void method1() –

int x = 5; // local variable

System.out.println(x); // accessing $'x'$

”

”

Predicting Output Example

- Example program:

```
public class Test  static int x = 11; private int y = 33; public
void method1(int x)  Test t = new Test(); this.x = 22; y = 44; Sy
" + Test.x); System.out.println("t.x: " + t.x); System.out.printl
" + t.y); System.out.println("y: " + y);  public static void main
args[])  Test t = new Test(); t.method1(5);
```

NextGen Academy
Towards fulfilling a million dreams

“begin–frame”–Loop Variables (Block Scope)”

“begin–itemize”

“item Variables declared inside a block have blocklevel scope.

“item They are only accessible within the block.

“item Syntax:

“begin–verbatim”

public class MyClass –

public static void main(String[] args) –

–

int x = 10; // block scope

System.out.println(x);

”

// Uncommenting below line would produce error

// System.out.println(x);

”

”

NextGen Academy

Towards fulfilling a million dreams

Tricky Loop Scope Example

- Nested loop with redeclaration of variable `a`.
- Java throws an error due to the redeclaration.
- Syntax:

```
public class Test  public static void main(String args[])  int a
= 5; for (int a = 0; a < 5; a++)  System.out.println(a);
```

NextGen Academy
Towards fulfilling a million dreams

“begin–frame”–Important Points about Variable Scope in Java”
“begin–itemize”
“item Scope defined by curly brackets.
“item Accessing variables within the same or nested brackets.
“item Classlevel variables accessible by all member methods.
“item Usage of ”this” keyword for samenamed local and class variables.
“item Declaration placement for loop variables affects readability.
“end–itemize”
“end–frame”
“begin–frame”–Array Basics”
“begin–itemize”
“item A collection of similar elements with contiguous memory.
“item Java array is an object with elements of a similar data type.
“item Indexbased starting at 0.
“item Array length obtained using the “texttt–length” member.
“item Java array is an object of a dynamically generated class.
“end–itemize”
“end–frame”
“begin–frame”–Array Basics (contd.)”

“begin–itemize”
“item Advantages:
“begin–itemize”
“item Code Optimization for efficient data retrieval.
“item Random access to data at an index position.
“end–itemize”
“item Disadvantages:
“begin–itemize”
“item Size Limit: Fixed size doesn$'tgrowatruntime$.
“end–itemize”
“end–itemize”
“end–frame”
“begin–frame”–Types of Array in Java”
“begin–itemize”
“item Single Dimensional Array
“item Multidimensional Array
“end–itemize”
“end–frame”
“begin–frame”–Single Dimensional Array in Java”

“begin–itemize″
“item “textbf–Syntax to Declare:″ “texttt–dataType[] arr;″ or “texttt–dataType []arr;″ or “texttt–dataType arr[];″
“item “textbf–Instantiation:″ “texttt–arrayRefVar = new datatype[size];″
“item Example:
“begin–itemize″
“item “lstinline–int a[]=new int[5];″
“end–itemize″
“item Declaration instantiation and initialization in a single line: “lstinline–int a[]=–33345″;″
“item Foreach Loop: “lstinline–for(int i:arr)″
“end–itemize″
“end–frame″
“begin–frame″[fragile]–Foreach Loop Example″
“begin–lstlisting″[language=Java]
class Testarray1–
public static void main(String args[])–
int arr[]=–33345″;
for(int i:arr)

```
System.out.println(i);
″
″
```

“end–lstlisting″

Output:

“begin–verbatim″

```
33
3
4
5
```

Passing Arrays to Methods

- Pass Java array to method for reusing logic.
- Example: Find the minimum number in an array.
- Anonymous Array in Java.

Anonymous Array Example

```
class TestAnonymousArray{
  static void printArray(int arr[]){
    for(int i=0; i<arr.length; i++)
      System.out.println(arr[i]);
  }

  public static void main(String args[]){
    printArray(new int[]{10,22,44,66});
  }
}
```

Output:

```
10
22
44
66
```

NextGen Academy
Towards fulfilling a million dreams

Returning an Array from a Method

- Return an array from a method in Java.
- Example: Return an array with values.

Returning an Array Example

```
class TestReturnArray{
  static int[] get(){
    return new int[]{10,30,50,90,60};
  }

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

NextGen Academy
Towards fulfilling a million dreams

Copying Arrays

- Copy an array to another using `System.arraycopy()`.
- Syntax: `System.arraycopy(Object src, int srcPos, Object dest, int destPos, int length)`
- Example: Copying a source array into a destination array.

Copying Arrays Example

```
class TestArrayCopyDemo {
  public static void main(String[] args) {
    char[] copyFrom = {'d', 'e', 'c', 'a', 'f', 'f', 'e', 'i',
    char[] copyTo = new char[7];
    System.arraycopy(copyFrom, 2, copyTo, 0, 7);
    System.out.println(String.valueOf(copyTo));
  }
}
```

Output:

```
caffein
```

NextGen Academy
Towards fulfilling a million dreams

Linear Search Method

- Sequentially traverses the array
- Uses a for-each loop to check each element
- **Syntax:**

```
for (int element : arr) {
  if (element == toCheckValue) {
    return true;
  }
}
```

- Example code and output provided
- **Complexity:** Time - $O(N)$, Space - $O(1)$

NextGen Academy
Towards fulfilling a million dreams

Binary Search Method

- Searches a sorted array using divide and conquer
- Utilizes `Arrays.binarySearch()` method
- **Syntax:**

```
int res = Arrays.binarySearch(arr, toCheckValue);
boolean test = res >= 0 ? true : false;
```

- Example code and output provided
- **Complexity:** Time - $O(n \log(n))$, Space - $O(1)$

NextGen Academy
Towards fulfilling a million dreams

List.contains() Method

- Checks if an element exists in a given list
- Utilizes `List.contains()` method
- **Syntax:**

```
boolean test = Arrays.asList(arr).contains(toCheckValue);
```

- Example code and output provided
- **Complexity:** Time - $O(N)$, Space - $O(1)$

Stream.anyMatch() Method

- Checks if any elements in a stream match a given predicate
- Utilizes `Stream.anyMatch()` method
- Two examples provided with different stream creation methods
- **Syntax:**

```
boolean test = IntStream.of(arr).anyMatch(x -> x == toCheckValue)
```

- **Complexity:** Time - $O(N)$, Space - $O(1)$

NextGen Academy
Towards fulfilling a million dreams

Using the sort() Method

- The `sort()` method is provided by the `Arrays` class in `java.util`.
- It uses the Dual-Pivot Quicksort algorithm for sorting.
- Complexity: $O(n \log n)$.
- `public static void sort(int[] a)` - sorts an array in ascending order.
- Example program using `sort()` method.
- `Arrays.sort(array)` sorts the array in-place.
- Works for primitive types like `int`, `float`, etc.

SortArrayExample1.java

```
import java.util.Arrays;

public class SortArrayExample1 {
    public static void main(String[] args) {
        int[] array = {90, 23, 5, 109, 12, 22, 67, 34};
        Arrays.sort(array);
        System.out.println("Elements in ascending order:");
        for (int i = 0; i < array.length; i++) {
            System.out.println(array[i]);
        }
    }
}
```

NextGen Academy
Towards fulfilling a million dreams

Using the for Loop

- Sorting without using `sort()` method.
- Nested loops for comparing and swapping elements.
- Example program using a for loop.
- Logic to sort in ascending order.
- Output displays sorted array elements.

SortArrayExample2.java

```
public class SortArrayExample2 {
    public static void main(String[] args) {
        int[] arr = {78, 34, 1, 3, 90, 34, -1, -4, 6, 55, 20, -65
        System.out.println("Array elements after sorting:");
        for (int i = 0; i < arr.length; i++) {
            // Sorting logic
            for (int j = i + 1; j < arr.length; j++) {
                int tmp = 0;
                if (arr[i] > arr[j]) {
                    tmp = arr[i];
                    arr[i] = arr[j];
                    arr[j] = tmp;
                }
            }
            // Prints the sorted element of the array
            System.out.println(arr[i]);
        }
    }
```

NextGen Academy
Towards fulfilling a million dreams

Using User Defined Method

- Define a method (`sortArray()`) to sort an array.
- Logic similar to the for loop approach.
- Invoking the user-defined method for sorting.
- Example program with a user-defined sorting method.
- Output displays array elements before and after sorting.

SortArrayExample3.java

```
public class SortArrayExample3 {
    public static void main(String[] args) {
        int[] array = {12, 45, 1, -1, 0, 4, 56, 23, 89, -21, 56,
        System.out.print("Array elements before sorting:\n");
        for (int i = 0; i < array.length; i++) {
            System.out.println(array[i]);
        }
        sortArray(array, array.length);
        System.out.print("Array elements after sorting:\n");
        for (int i = 0; i < array.length; i++) {
            System.out.println(array[i]);
        }
    }

    private static void sortArray(int array[], int n) {
        // User-defined sorting logic
    }
}
```

Sort Array in Descending Order

- Descending order arranges elements from highest to lowest.
- Methods for sorting in descending order:
- Using `reverseOrder()` method.
- Using the for loop.
- Using a user-defined method.

Using `reverseOrder()` Method

- Java Collections class provides `reverseOrder()`.
- Static method, invokes by using the class name.
- Sorts array in reverse-lexicographic order.
- Example program with `Arrays.sort(array, Collections.reverseOrder())`.
- Output displays array elements in descending order.

SortArrayExample4.java

```
import java.util.Arrays;
import java.util.Collections;

public class SortArrayExample4 {
    public static void main(String[] args) {
        Integer[] array = {23, -9, 78, 102, 4, 0, -1, 11, 6, 110,
        Arrays.sort(array, Collections.reverseOrder());
        System.out.println("Array elements in descending order: "
    }
}
```

NextGen Academy
Towards fulfilling a million dreams

Without Using the Method

- Sorting in descending order without `reverseOrder()`.
- Example program using a for loop.
- Nested loops for comparing and swapping elements.
- Output displays array elements in descending order.

SortArrayExample6.java

```
public class SortArrayExample6 {
    public static void main(String[] args) {
        int temp;
        int a[] = {12, 5, 56, -2, 32, 2, -26, 9, 43, 94, -78};
        for (int i = 0; i < a.length; i++) {
            for (int j = i + 1; j < a.length; j++) {
                if (a[i] < a[j]) {
                    temp = a[i];
                    a[i] = a[j];
                    a[j] = temp;
                }
            }
        }
        System.out.println("Array elements in descending order:")
        for (int i = 0; i <= a.length - 1; i++) {
            System.out.println(a[i]);
        }
    }
```

NextGen Academy
Towards fulfilling a million dreams

Using User Defined Method

- Define a method (`sortArray()`) to sort an array in descending order.
- Logic similar to the for loop approach.
- Invoking the user-defined method for sorting.
- Example program with a user-defined sorting method.
- Output displays array elements before and after sorting.

SortArrayExample7.java

```
import java.util.Scanner;

public class SortArrayExample7 {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int n, temp;
        System.out.print("Enter the number of elements: ");
        n = s.nextInt();
        int a[] = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            a[i] = s.nextInt();
        }
        // User-defined method for sorting in descending order
        sortArray(a, n);
        System.out.println("Array elements in descending order:")
        for (int i = 0; i < n - 1; i++) {
            System.out.println(a[i]);
```

Two-Dimensional Array Basics

- Array - fixed-length container for values of a common data type.
- Two-dimensional array - collection of arrays arranged in rows and columns (tabular format).
- Access elements using row and column indices.

Introduction to 2D Array in Java

- Data represented in tabular form using rows and columns.
- Effective for organizing various types of information.
- Java implements this using a two-dimensional array.
- A linear data structure storing data in a tabular format.

In Java, an array is...

- Homogeneous collection of fixed values.
- Stored in contiguous memory locations.
- Elements of the same type (homogeneous data).
- Two-dimensional array treated as an array of one-dimensional arrays.

Java's 2D Array Implementation

- 2D array in Java is a collection of pointers.
- Each pointer refers to a one-dimensional array representing a row.
- Created using the `new` keyword.

2D Array Declaration in Java

```
DataType[][] ArrayName;
```

- `DataType` - type of values the array can store.
- `ArrayName` - reference variable for the 2D array object.

Create Two-Dimensional Array in Java

```
// Declaring 2D array
DataType[][] ArrayName;

// Creating a 2D array
ArrayName = new DataType[r][c];
```

- `new DataType[r][c]` - creates a 2D array with `r` rows and `c` columns.
- JVM allocates memory and initializes with default values.

NextGen Academy
Towards fulfilling a million dreams

Example: Creating a 2D Array

```
// Declaring 2D array
int[][] a;

// Creating a 2D array
a = new int[3][3];
```

- `a` - reference variable for the 2D array with 3 rows and 3 columns.

Java 2D Array - Primitive Type

- Arrays are collections of elements with similar data types.
- 2D arrays in Java for primitive types: int, byte, short, long, float, double, boolean, char.
- Syntax: `DataType[][] ArrayName = new DataType[r][c].`
- Access using indices: `ArrayName[i][j].`
- Index starts from 0; ArrayIndexOutOfBoundsException for out-of-bounds.

Java 2D Array - Objects

- 2D array of objects: collection of arrays of reference variables.
- Syntax: `ClassName[][] ArrayName`.
- Accessing elements: `ArrayName[i][j]`.
- Useful for dealing with String data objects.

Initializing 2D Arrays

- Declaring with both dimensions: `int[][] a = new int[2][2].`
- Declaring with one dimension: `int[][] a = new int[2][].`
- Position of square brackets matters.
- Variable column length: `int a = new int[4][]; a[0] = new int[1];`
  `....`
- Heterogeneous 2D array: `Object[][] a = new Object[3][]; ....`

NextGen Academy
Towards fulfilling a million dreams

- Data stored in row and column-based index.
- Syntax for declaration: `dataType[][] arrayRefVar`.
- Example instantiation and initialization.

- Process in which a method calls itself.
- Recursive method syntax.
- Example 1: Infinite recursion.
- Example 2: Finite recursion.

Recursion Examples

- Example 3: Factorial Number calculation.
- Showcase of recursive calls.