

CS 1027

Fundamentals of Computer
Science II

Java Foundations: Overview

Ahmed Ibrahim

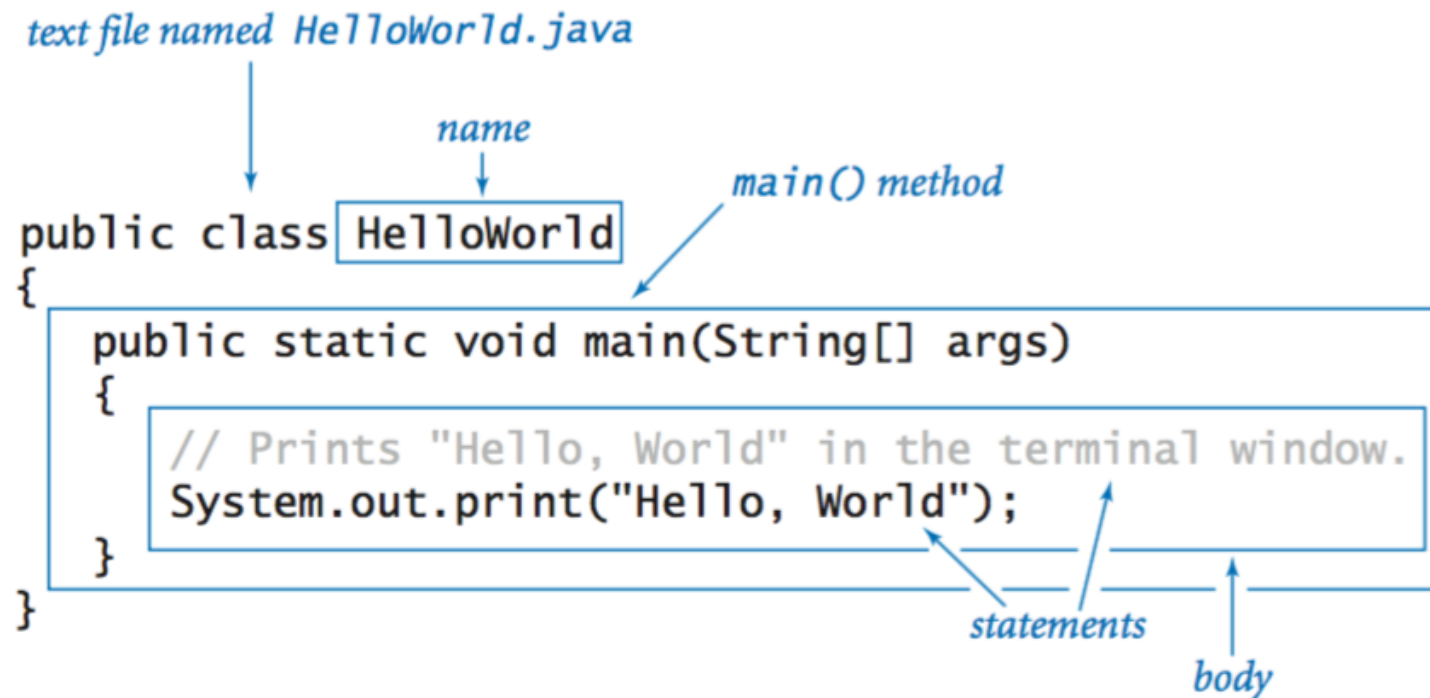


Agenda

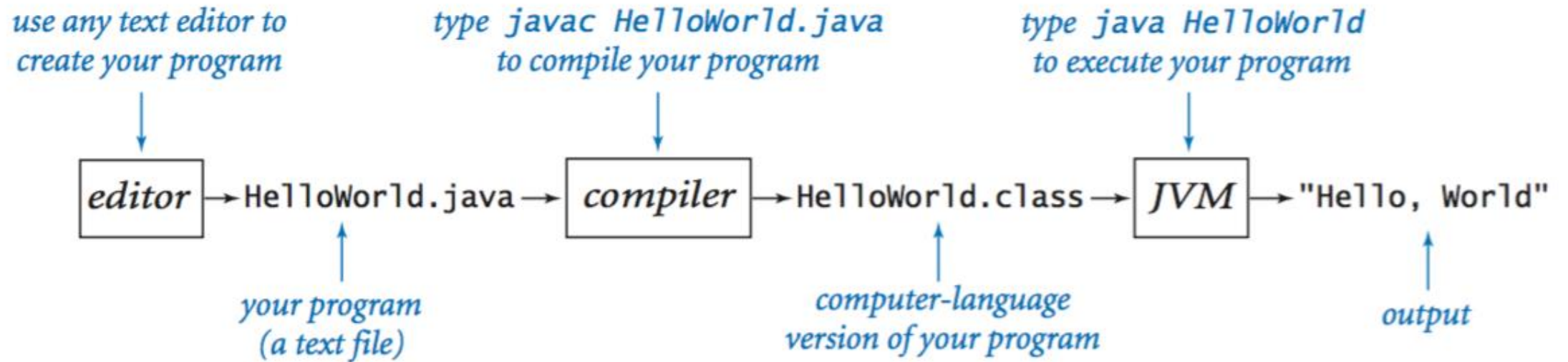
- Java Program Anatomy
- Data Types and Variables
- Classes and Objects in Java
- Object-Oriented Programming (OOP) Basics
- Operators and Expressions
- Conditional Statements
- Arrays and Input/Output (I/O) Handling

Java Program

- A Java program is a collection of classes.



Editing, Compiling, and Executing

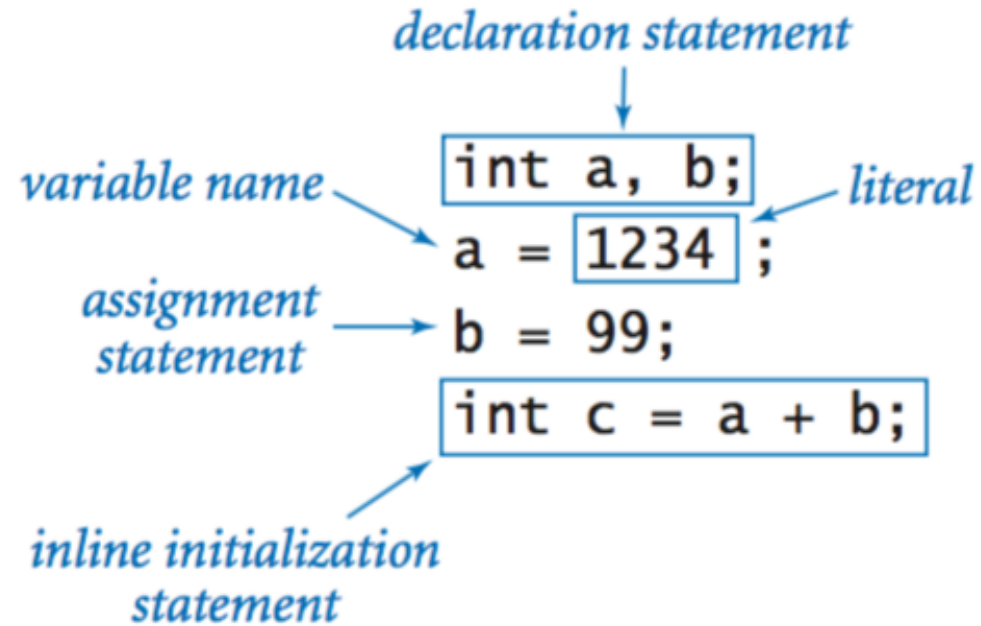


Data Types and Variables



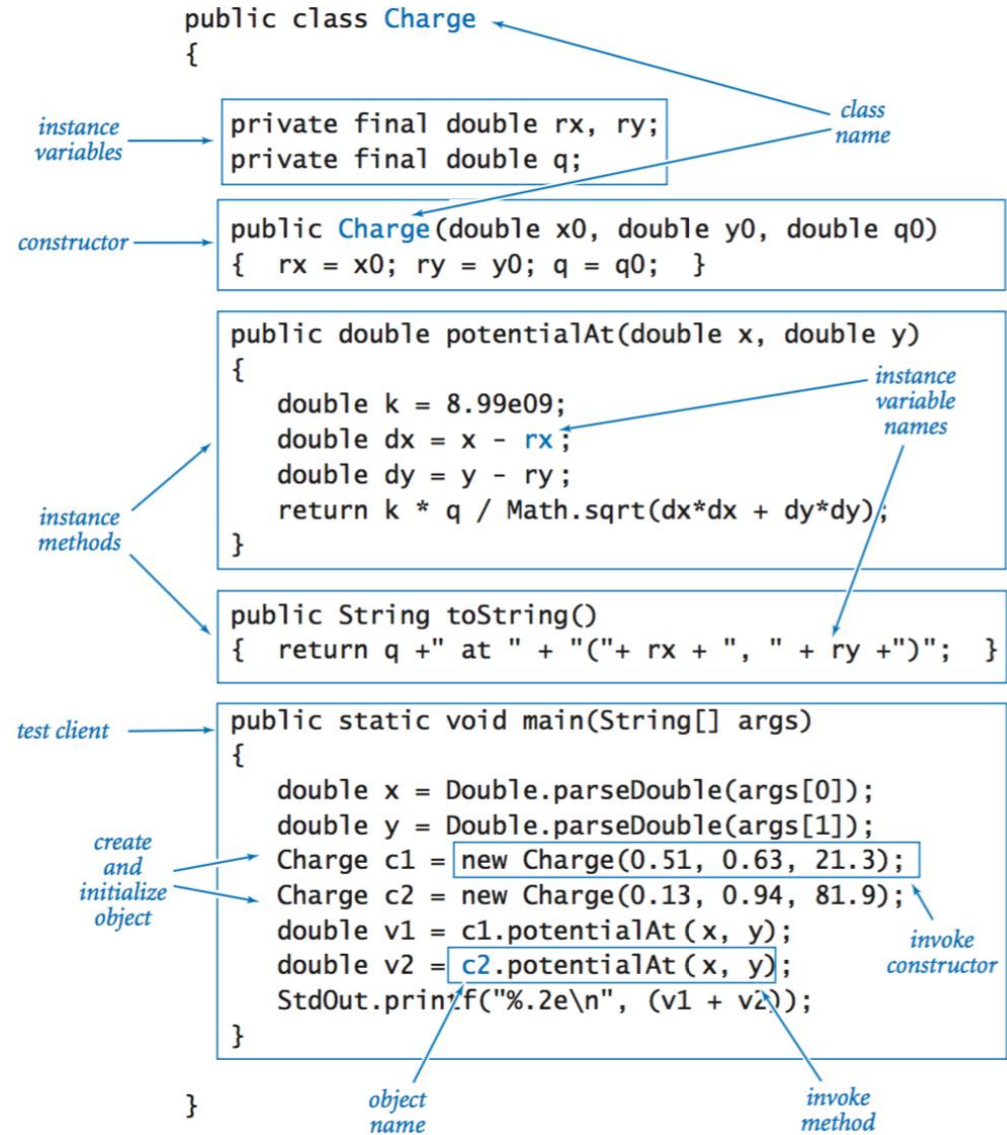
<i>type</i>	<i>set of values</i>	<i>common operators</i>	<i>sample literal values</i>
int	integers	+ - * / %	99 12 2147483647
double	floating-point numbers	+ - * /	3.14 2.5 6.022e23
boolean	boolean values	&& !	true false
char	characters		'A' '1' '%' '\n'
String	sequences of characters	+	"AB" "Hello" "2.5"

Declaration & Assignment Statements



Class in Java

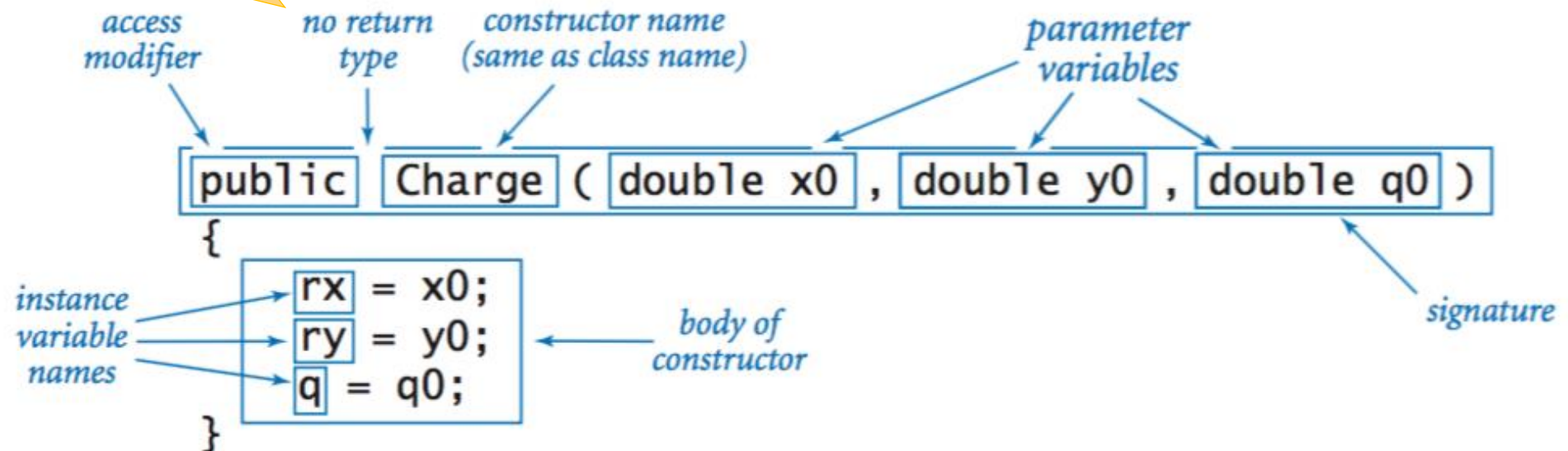
- The figure illustrates a Java class `Charge` with **constructors**, **instance variables**, **methods**, and a **test client**.



Constructor in Java

- The following illustrates a Java class **Charge** constructor.

a Java class constructor
cannot have a return value



Java Class Example

- The code snippet illustrates an example of a Java class (Car).
- It includes the following elements:
 - **Instance variables:** String model and int year;
 - **Constructor:** A method to initialize the model and year of a Car object.
 - **Main method:** Demonstrates the creation of a Car object (myCar) with the model "Toyota" and year 2020, followed by a System.out.println statement to print the car's model and year.

```
1  public class Car {
2
3      // Instance variables for the Car class
4      String model;
5      int year;
6
7      // Constructor to initialize the Car object
8      Car(String model, int year) {
9          this.model = model;
10         this.year = year;
11     }
12
13     // Main method to run the program
14     Run | Debug | Run main | Debug main
15     public static void main(String[] args) {
16         // Create a Car object
17         Car myCar = new Car(model:"Toyota", year:2020);
18
19         // Print the model and year of the car
20         System.out.println(myCar.model + " " + myCar.year);
21     }
```

Object in Java

- **Declares a variable (object name):**
declares a variable `s` of type `String`, which will later reference a `String` object.
- **Invokes a constructor to create an object:** a new `String` object is created with the value "`Hello, World`", and the variable `s` is assigned to reference this object.

```
String s;  
s = new String("Hello, World");  
char c = s.charAt(4);
```

The diagram illustrates the process of creating and using a String object in Java. It shows three lines of code with annotations:

- `String s;`: An annotation "declare a variable (object name)" points to `String`, and another annotation "invoke a constructor to create an object" points to `new String("Hello, World")`.
- `s = new String("Hello, World");`: An annotation "object name" points to `s`, and another annotation "invoke an instance method that operates on the object's value" points to `charAt(4)`.

Strings

```
String a = new String("now is");  
String b = new String("the time");  
String c = new String(" the");
```

<i>instance method call</i>	<i>return type</i>	<i>return value</i>
a.length()	int	6
a.charAt(4)	char	'i'
a.substring(2, 5)	String	"w i"
b.startsWith("the")	boolean	true
a.indexOf("is")	int	4
a.concat(c)	String	"now is the"
b.replace("t", "T")	String	"The Time"
a.split(" ")	String[]	{ "now", "is" }
b.equals(c)	boolean	false

Inheritance & Polymorphism

- **Inheritance:** The Dog class inherits properties and behaviors (methods) from the Animal class.
- **Method Overriding:** The Dog class redefines the sound() method to provide its specific behavior while maintaining the structure of the base class.

```
1  class Animal {  
2      void sound() {  
3          System.out.println(x:"Animal makes a sound");  
4      }  
5  }  
6  
7  class Dog extends Animal {  
8      void sound() {  
9          System.out.println(x:"Dog barks");  
10     }  
11 }  
12
```

Methods Overloading

- **Method Declaration and Return Types:**

- The method `add(int a, int b)` returns an `int`.
- The overloaded method `add(double a, double b)` returns a `double`.

- **Parameters and Arguments:**

- The methods accept two parameters (`int` or `double`).
- When calling the methods, we pass arguments such as 5, 3 and 2.5, 3.2.

- **Overloading Methods:**

- There are two `add` methods: one works with `int` values and the other with `double` values.

```
1 public class methodExample {
2
3     // Method 1: Adds two integers
4     public static int add(int a, int b) {
5         return a + b;
6     }
7
8     // Method 2: Adds two doubles (Overloaded method)
9     public static double add(double a, double b) {
10        return a + b;
11    }
12
13    Run | Debug | Run main | Debug main
14    public static void main(String[] args) {
15        // Using the add method with integers
16        int sum = add(a:5, b:3);
17
18        // Output: Sum of integers: 8
19        System.out.println("Sum of integers: " + sum);
20
21        // Using the overloaded add method with doubles
22        double doubleSum = add(a:2.5, b:3.2);
23
24        // Output: Sum of doubles: 5.7
25        System.out.println("Sum of doubles: " + doubleSum);
26    }
```

Operators & Expressions

```
1 public class OPExample {
    Run | Debug | Run main | Debug main
2     public static void main(String[] args) {
3         int x = 5, y = 10;
4
5         // Arithmetic Operators
6         System.out.println("Sum: " + (x + y));
7         System.out.println("Difference: " + (y - x));
8         System.out.println("Product: " + (x * y));
9         System.out.println("Quotient: " + (y / x));
10        System.out.println("Remainder: " + (y % x));
11
12        // Relational Operators
13        System.out.println("x == y: " + (x == y));
14        System.out.println("x != y: " + (x != y));
15        System.out.println("x > y: " + (x > y));
16        System.out.println("x < y: " + (x < y));
17
18        // Logical Operators
19        boolean result = (x < y) && (x > 0);
20        System.out.println("Result of (x < y) && (x > 0): " + result);
21
22        result = (x > y) || (x == 5);
23        System.out.println("Result of (x > y) || (x == 5): " + result);
24
25        result = !(x == y);
26        System.out.println("Result of !(x == y): " + result);
27    }
28 }
```

Conditional Statements

TABLE A.4
Java Control Statements

Control Structure	Purpose	Syntax
if ... else	Used to write a decision with <i>conditions</i> that select the alternative to be executed. Executes the first (second) alternative if the <i>condition</i> is true (false).	<pre>if (<i>condition</i>) { ... } else { ... }</pre>
switch	Used to write a decision with scalar values (integers, characters) that select the alternative to be executed. Executes the <i>statements</i> following the <i>label</i> that is the <i>selector</i> value. Execution falls through to the next <i>case</i> if there is no return or break . Executes the statements following default if the <i>selector</i> value does not match any <i>label</i> .	<pre>switch (<i>selector</i>) { case <i>label</i> : <i>statements</i>; break; case <i>label</i> : <i>statements</i>; break; ... default : <i>statements</i>; }</pre>
while	Used to write a loop that specifies the repetition <i>condition</i> in the loop header. The <i>condition</i> is tested before each iteration of the loop and, if it is true, the loop body executes; otherwise, the loop is exited.	<pre>while (<i>condition</i>) { ... }</pre>
for	Used to write a loop that specifies the <i>initialization</i> , repetition <i>condition</i> , and <i>update</i> steps in the loop header. The <i>initialization</i> statements execute before loop repetition begins; the <i>condition</i> is tested before each iteration of the loop and, if it is true, the loop body executes; otherwise, the loop is exited. The <i>update</i> statements execute after each iteration.	<pre>for (<i>initialization</i>; <i>condition</i>; <i>update</i>) { ... }</pre>

Appendix A of Koffman, E. B., & Wolfgang, P. A. T. (2016). *Data structures: Abstraction and design using Java* (3rd ed.) Wiley.

Conditional Statements Example

```
1  import java.util.Scanner;
2
3  public class controlFlowExample {
    Run | Debug | Run main | Debug main
4      public static void main(String[] args) {
5          Scanner input = new Scanner(System.in);
6
7          // Shortened if-else example
8          System.out.print(s:"Enter your score: ");
9          int score = input.nextInt();
10         System.out.println(score > 90 ? "Grade: A" : score > 80 ? "Grade: B" : "Grade: C");
11
12         // Shortened switch example
13         System.out.print(s:"Enter a number for the day (1-7): ");
14         switch(input.nextInt()) {
15             case 1: System.out.println(x:"Monday"); break;
16             case 2: System.out.println(x:"Tuesday"); break;
17             case 3: System.out.println(x:"Wednesday"); break;
18             case 4: System.out.println(x:"Thursday"); break;
19             case 5: System.out.println(x:"Friday"); break;
20             case 6: System.out.println(x:"Saturday"); break;
21             case 7: System.out.println(x:"Sunday"); break;
22             default: System.out.println(x:"Invalid day");
23         }
24
25         input.close();
26     }
27 }
```


Conditional Statements

TABLE A.4 (continued)

Control Structure	Purpose	Syntax
do ... while	Used to write a loop that specifies the repetition <i>condition</i> after the loop body. The <i>condition</i> is tested after each iteration of the loop and, if it is true, the loop body is repeated; otherwise, the loop is exited. The loop body always executes at least one time.	<pre>do { ... while (<i>condition</i>) ;</pre>

Appendix A of Koffman, E. B., & Wolfgang, P. A. T. (2016). *Data structures: Abstraction and design using Java* (3rd ed.) Wiley.

Arrays

a

a[0]
a[1]
a[2]
a[3]
a[4]
a[5]
a[6]
a[7]

Inline array initialization

```
String[] SUITS = { "Clubs", "Diamonds", "Hearts", "Spades" };
```

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

2D Arrays

Diagram illustrating a 2D array structure with 10 rows and 3 columns. The first row is highlighted with a blue arrow labeled "row 1". The first column is highlighted with a blue arrow labeled "column 2". The element at the intersection of row 1 and column 2 is labeled `a[1][2]`.

99	85	98
98	57	78
92	77	76
94	32	11
99	34	22
90	46	54
76	59	88
92	66	89
97	71	24
89	29	38

```
double [][] a =  
{  
    { 99.0, 85.0, 98.0, 0.0 },  
    { 98.0, 57.0, 79.0, 0.0 },  
    { 92.0, 77.0, 74.0, 0.0 },  
    { 94.0, 62.0, 81.0, 0.0 },  
    { 99.0, 94.0, 92.0, 0.0 },  
    { 80.0, 76.5, 67.0, 0.0 },  
    { 76.0, 58.5, 90.5, 0.0 },  
    { 92.0, 66.0, 91.0, 0.0 },  
    { 97.0, 70.5, 66.5, 0.0 },  
    { 89.0, 89.5, 81.0, 0.0 },  
    { 0.0, 0.0, 0.0, 0.0 }  
};
```

Input/Output Handling

- Reading Input using
Scanner
- Writing Output to Console

```
1  import java.util.Scanner;
2
3  public class ioJava
4  {
5      Run | Debug
6      public static void main(String[] args) {
7          // Create a Scanner object to read input
8          Scanner input = new Scanner(System.in);
9
10         // Prompt the user to enter a number
11         System.out.print(s:"Enter a number: ");
12
13         // Read the entered number
14         int num = input.nextInt();
15
16         // Display the entered number
17         System.out.println("You entered: " + num);
18
19         // Close the scanner to prevent resource leaks
20         input.close();
21     }
```

Best Practices and Coding Standards

- Use meaningful variable names
- Keep code DRY (Don't Repeat Yourself)
- Commenting and documentation
- Follow Java naming conventions (e.g., camelCase, ClassName)

Takeaway Points

- A Java program is a collection of classes.
- The JVM approach enables a Java program written on one machine to execute on any other machine with a JVM.
- Java defines a set of primitive data types for representing numbers, characters, and Boolean data.
- The control structures of Java are similar to those found in other languages.
- You can declare your own Java classes and create objects of these classes using the new operator.
- A class has data fields and instance methods.

References

- The following references were used in the preparation of this presentation:
 - Appendix A of Koffman, E. B., & Wolfgang, P. A. T. (2016). Data structures: Abstraction and design using Java (3rd ed.) Wiley.
 - Princeton University. (n.d.). Java cheatsheet.
<https://introcs.cs.princeton.edu/java/11cheatsheet/>