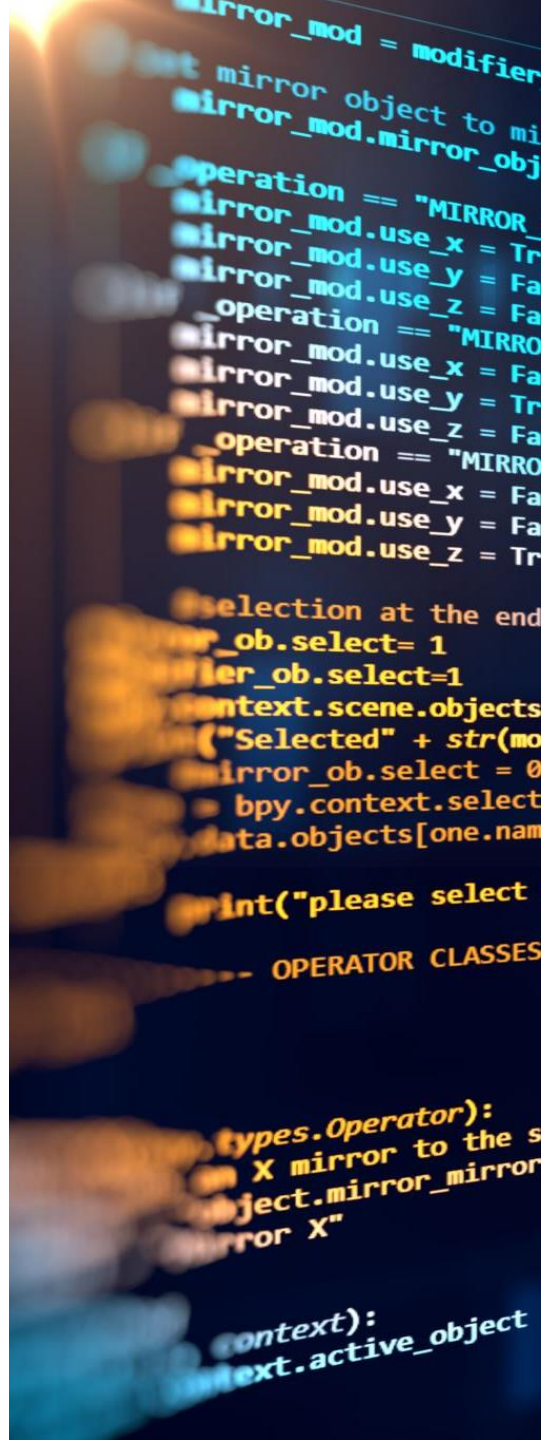
CS 1027
Fundamentals of Computer Science II

Java Foundations: Overview

Ahmed Ibrahim

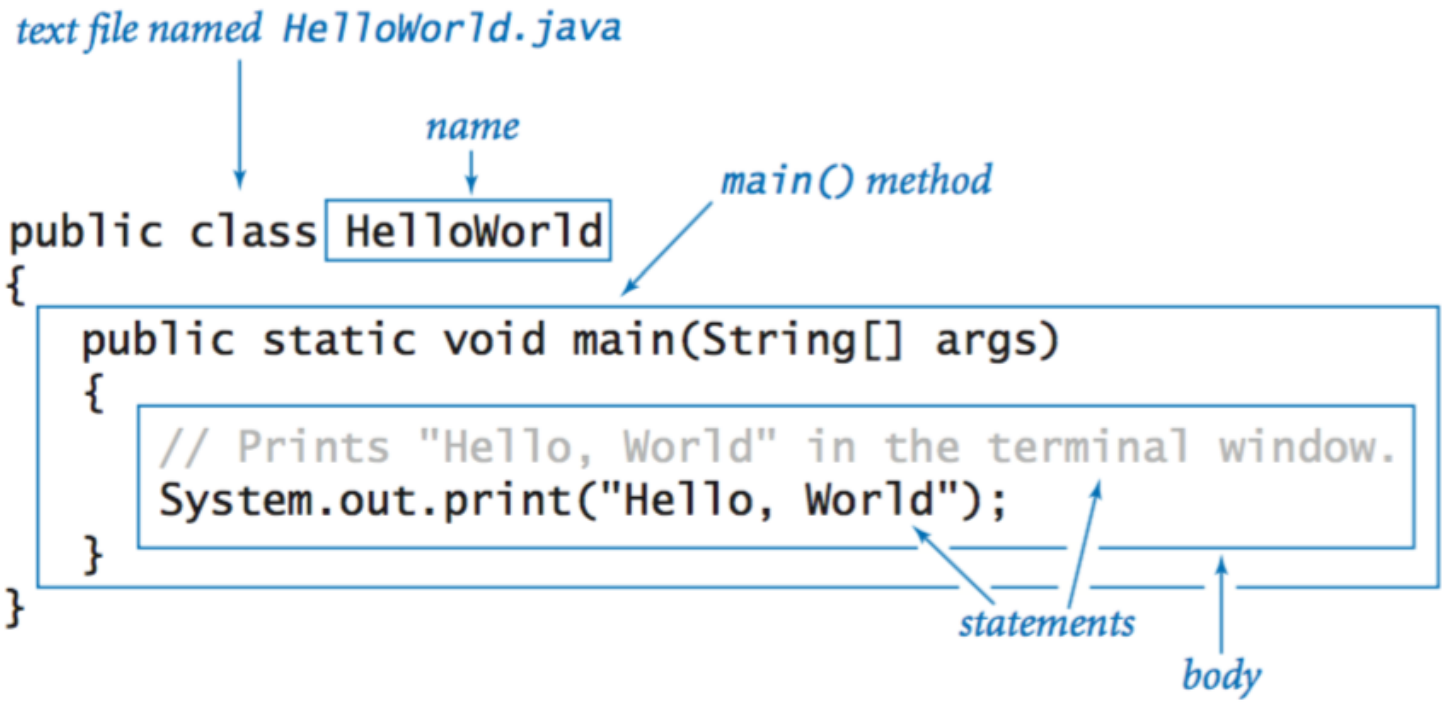


Agenda

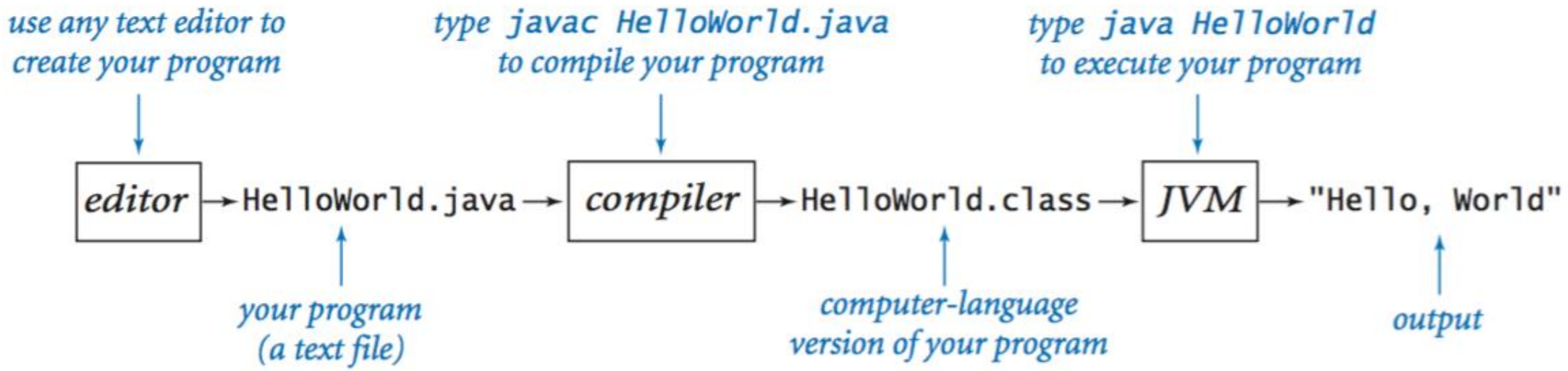
- Program Anatomy
- Data Types and Variables
- Operators and Expressions
- Control Flow Statements
- Methods
- Arrays and Strings
- Object-Oriented Programming (OOP) Basics
- Input/Output (I/O) Handling

Java Program

- A Java program is a collection of classes.



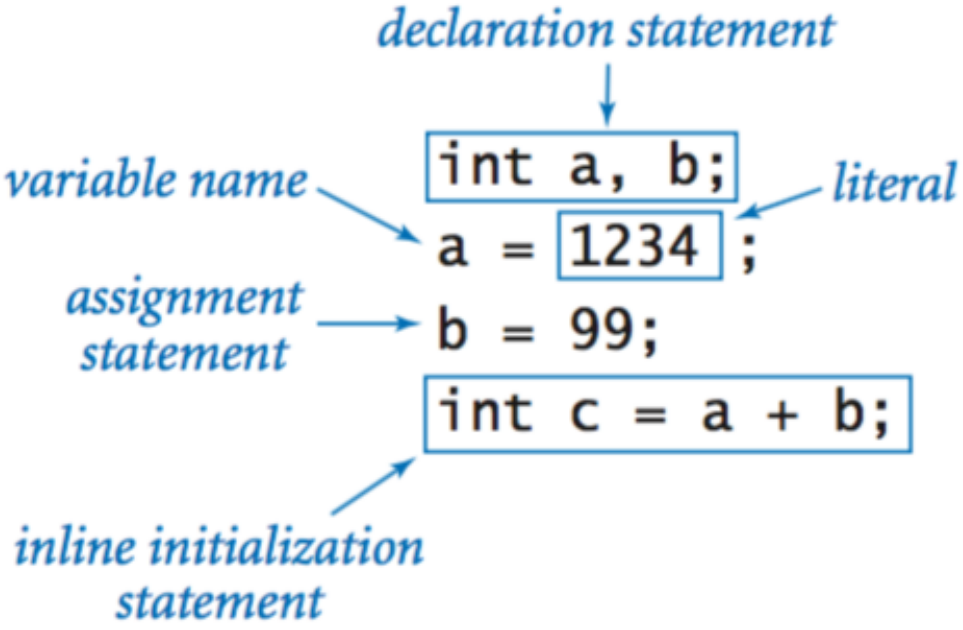
Editing, Compiling, and Executing



Data Types and Variables

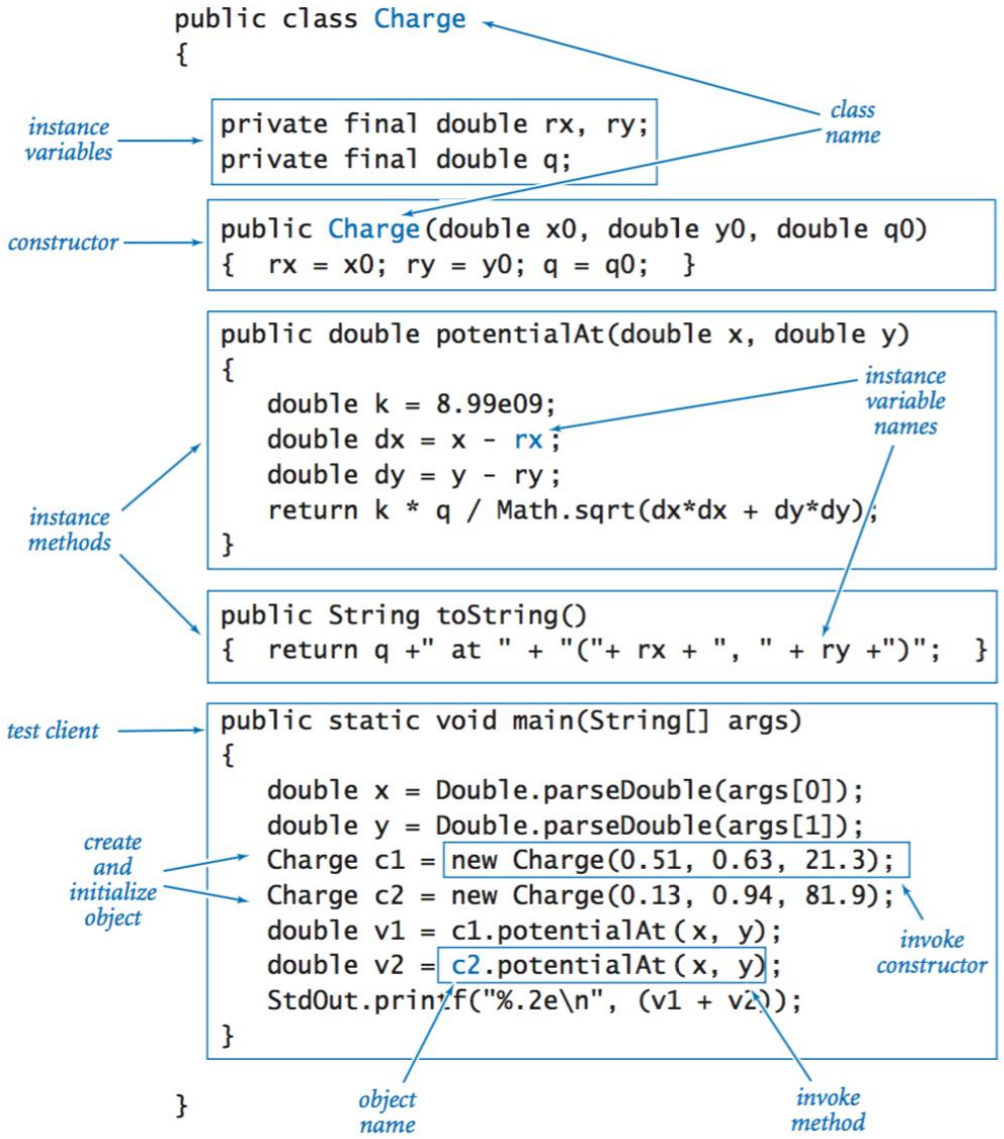
type	*set of values*	*common operators*	*sample literal values*
`int`	integers	`+ - * / %`	`99 12 2147483647`
`double`	floating-point numbers	`+ - * /`	`3.14 2.5 6.022e23`
`boolean`	boolean values	`&& || !`	`true false`
`char`	characters		`'A' '1' '%' '\n'`
`String`	sequences of characters	`+`	`"AB"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

Declaration & Assignment Statements



Class in Java

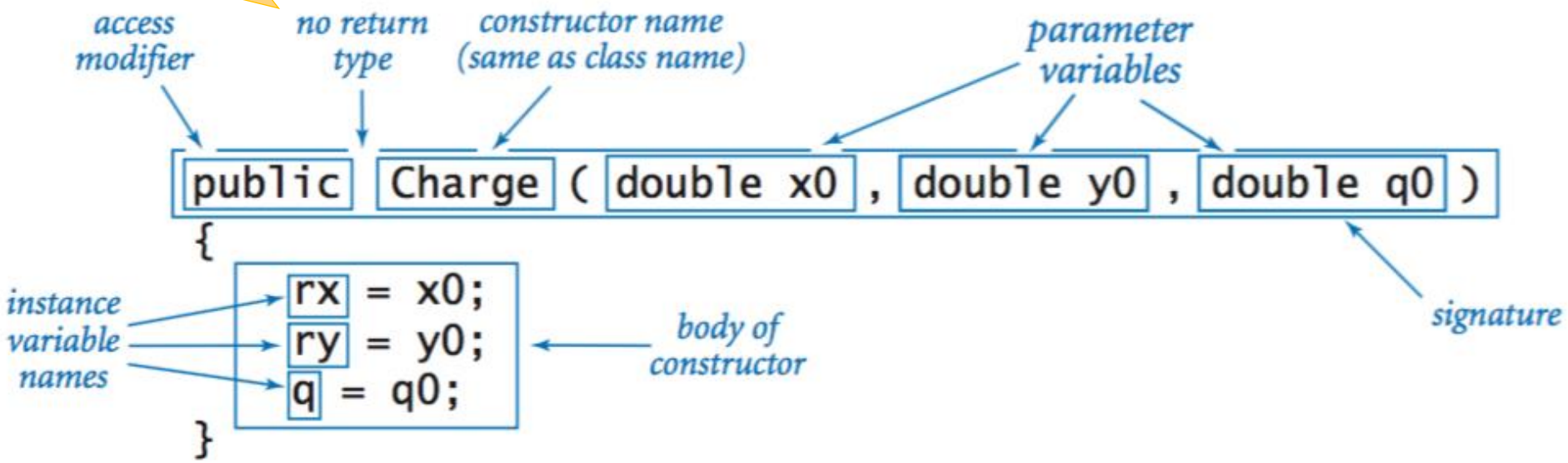
- The figure illustrates a Java class Charge with **constructors**, **instance variables**, **methods**, and a **test** client.



Constructor in Java

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

a Java class constructor **cannot** have a return value



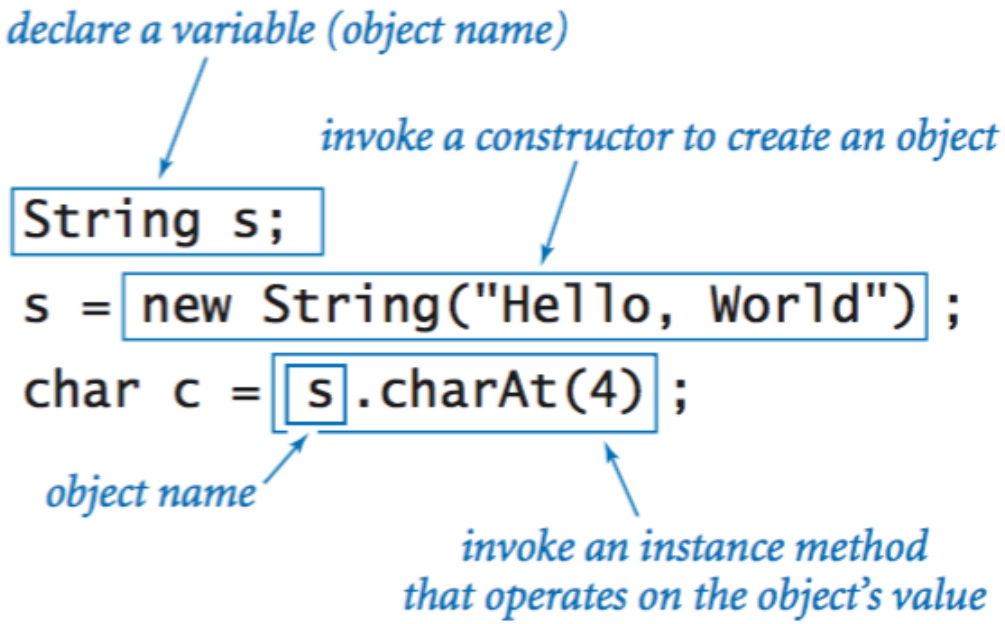
Java Class Example

- 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.

```java
public class Car {

    // Instance variables for the Car class
    String model;
    int year;

    // Constructor to initialize the Car object
    Car(String model, int year) {
        this.model = model;
        this.year = year;
    }

    // Main method to run the program
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        // Create a Car object
        Car myCar = new Car(model:"Toyota", year:2020);

        // Print the model and year of the car
        System.out.println(myCar.model + " " + myCar.year);
    }
}
```

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.



Strings

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

instance method call	*return type*	*return value*
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.

```java
public class methodExample {

    // Method 1: Adds two integers
    public static int add(int a, int b) {
        return a + b;
    }

    // Method 2: Adds two doubles (Overloaded method)
    public static double add(double a, double b) {
        return a + b;
    }

    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        // Using the add method with integers
        int sum = add(a:5, b:3);

        // Output: Sum of integers: 8
        System.out.println("Sum of integers: " + sum);

        // Using the overloaded add method with doubles
        double doubleSum = add(a:2.5, b:3.2);

        // Output: Sum of doubles: 5.7
        System.out.println("Sum of doubles: " + doubleSum);
    }
}
```

Operators & Expressions

```java
public class OPExample {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        int x = 5, y = 10;

        // Arithmetic Operators
        System.out.println("Sum: " + (x + y));
        System.out.println("Difference: " + (y - x));
        System.out.println("Product: " + (x * y));
        System.out.println("Quotient: " + (y / x));
        System.out.println("Remainder: " + (y % x));

        // Relational Operators
        System.out.println("x == y: " + (x == y));
        System.out.println("x != y: " + (x != y));
        System.out.println("x > y: " + (x > y));
        System.out.println("x < y: " + (x < y));

        // Logical Operators
        boolean result = (x < y) && (x > 0);
        System.out.println("Result of (x < y) && (x > 0): " + result);

        result = (x > y) || (x == 5);
        System.out.println("Result of (x > y) || (x == 5): " + result);

        result = !(x == y);
        System.out.println("Result of !(x == y): " + result);
    }
}
```

Conditional Statements

TABLE A.4
Java Control Statements

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

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

Conditional Statements Example

```java
import java.util.Scanner;

public class controlFlowExample {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        // Shortened if-else example
        System.out.print(s:"Enter your score: ");
        int score = input.nextInt();
        System.out.println(score > 90 ? "Grade: A" : score > 80 ? "Grade: B" : "Grade: C");

        // Shortened switch example
        System.out.print(s:"Enter a number for the day (1-7): ");
        switch(input.nextInt()) {
            case 1: System.out.println(x:"Monday"); break;
            case 2: System.out.println(x:"Tuesday"); break;
            case 3: System.out.println(x:"Wednesday"); break;
            case 4: System.out.println(x:"Thursday"); break;
            case 5: System.out.println(x:"Friday"); break;
            case 6: System.out.println(x:"Saturday"); break;
            case 7: System.out.println(x:"Sunday"); break;
            default: System.out.println(x:"Invalid day");
        }

        input.close();
    }
}
```

Conditional Statements

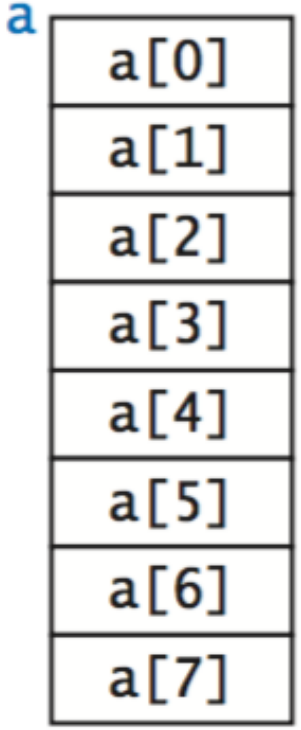
TABLE A.4 (continued)

Control Structure	Purpose	Syntax

`do ... while`	Used to write a loop that specifies the repetition *condition* after the loop body. The *condition* 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.	`do {`
		`  ...`
		`while (`*condition*`) ;`

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

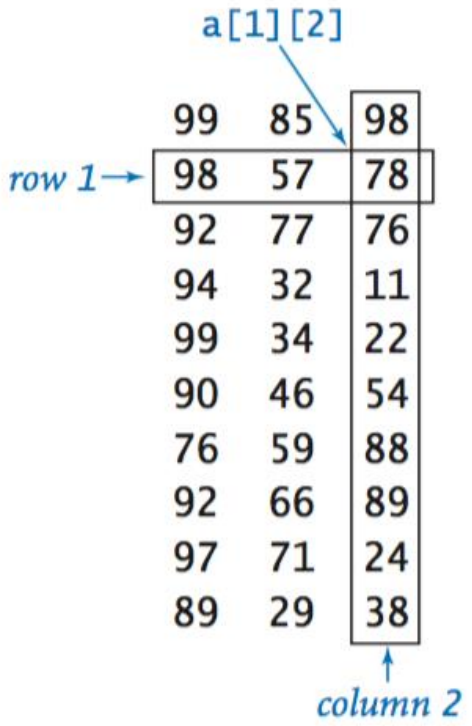
Arrays



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




```
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

```java
import java.util.Scanner;

public class ioJava
{
    Run | Debug
    public static void main(String[] args) {
        // Create a Scanner object to read input
        Scanner input = new Scanner(System.in);

        // Prompt the user to enter a number
        System.out.print(s:"Enter a number: ");

        // Read the entered number
        int num = input.nextInt();

        // Display the entered number
        System.out.println("You entered: " + num);

        // Close the scanner to prevent resource leaks
        input.close();
    }
}
```

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 (3nd ed.) Wiley.
  - Princeton University. (n.d.). Java cheatsheet. https://introcs.cs.princeton.edu/java/11cheatsheet/