* **Basic structure of a Java program**

A Java program can either be a library of static methods or a data type definition.

* Key Components:

+ Primitive Data Types: Define basic types such as integers, real numbers, and boolean values, including their possible values and operations.

+ Statements: Used to perform computations, assign values to variables, and control execution flow. There are six types of statements: declarations, assignments, conditionals, loops, calls, and returns.

+ Arrays: Allow for working with multiple values of the same type.

+ Static Methods: Enable encapsulation and code reuse, developing programs as independent modules.

+ Strings: Sequences of characters with built-in operations in Java.

+ Input/Output: Sets up communication between programs and the outside world.

* Data Abstraction: Extends encapsulation and reuse, allowing for the definition of non-primitive data types, supporting object-oriented programming.
* **Primitive data types and expressions**
* Data Types: A data type is defined by a set of values and the operations that can be performed on those values. The four primary primitive data types in Java are:

+Integers (int): Whole numbers with arithmetic operations.

+ Real Numbers (double): Decimal numbers with arithmetic operations.

+ Booleans (boolean): Values representing true or false, with logical operations.

+ Characters (char): Alphanumeric characters and symbols.

* Variables and Identifiers: Variables are named with identifiers and associated with data types, allowing them to store permissible values. Expressions use identifiers, operator symbols (like +, -, \*, /), literals (like 1 or 3.14), and can combine these to perform operations.
* Expressions: An expression is a combination of literals, variables, and operations that produces a value. Expressions are typically in infix notation and can involve arithmetic and logical operations. Operator precedence rules apply, where multiplication and division take precedence over addition and subtraction.
* Type Conversion: Java automatically promotes smaller data types to larger types (e.g., int to double). Casting is used to convert between types explicitly, but it should be done with care.
* Comparison Operators: These operators (like ==, !=, <, >, etc.) compare two values of the same type and produce a boolean result, crucial for conditional and loop statements.
* Other Primitive Types: Java also includes additional primitive data types such as:

+ long: 64-bit integers

+ short: 16-bit integers

+ byte: 8-bit integers

+ float: 32-bit single-precision real numbers

* **Statements**
* Statements: A Java program consists of statements that define computations by creating and manipulating variables, assigning values to them, and controlling the flow of execution. Statements are often organized into blocks using curly braces {}.
* Declarations: A declaration associates a variable name with a data type. Java requires declarations to specify variable names and types, making it a strongly typed language. The scope of a variable is determined by the block of code that follows its declaration.
* Assignments: An assignment statement assigns a data type value (defined by an expression) to a variable. For example, c = a + b expresses the action of setting the value of c to the sum of a and b, rather than a mathematical equality.
* Conditionals: The if statement allows for different actions based on varying inputs.
* Loops: The while statement enables repeated execution of a block of statements as long as a given condition is true.
* Break and Continue:

+break: Immediately exits the loop.

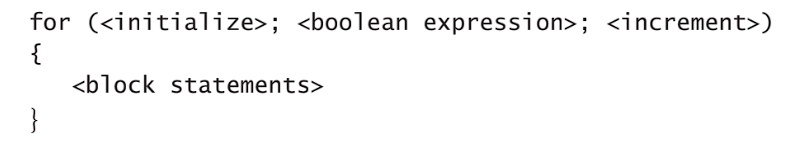
+ continue: Immediately starts the next iteration of the loop.

* **Shortcut notations**
* Initializing Declarations: Combine variable declaration and initialization, e.g., int i = 1;.
* Implicit Assignments:

+ Increment/Decrement Operators: i++ is equivalent to i = i + 1.

+ Compound Operations: Use operators like +=, -=, etc., to modify variables, e.g., i += 1; is the same as i = i + 1;.

* Single-statement Blocks: Curly braces can be omitted for single statements in conditionals or loops.
* For Loops: Use the for notation for compact looping



* Arrays:

+ Store a sequence of values of the same type.

+ Creating an Array: Involves declaring, creating, and initializing the array.

+ Accessing Array Elements: Use a[i] for the ith value (0 to N-1).

* Aliasing: Assigning one array name to another leads to both names referring to the same array.
* Two-Dimensional Arrays: An array of one-dimensional arrays, declared as double[][] a = new double[M][N];.
* Typical Array Processing: Find maximum, compute average, copy arrays, reverse elements, and perform matrix multiplication.
* **Arrays**
* One-Dimensional Arrays:

+ An array stores a sequence of values of the same type, accessed using indices. The indices range from 0 to N-1 (where N is the number of elements).

+ Syntax for accessing: a[i], with i from 0 to N-1.

* Creating and Initializing an Array:

+ Three steps to create an array:

Declare: Specify the name and type of data.

Create: Specify the length of the array (number of elements).

Initialize: Assign values to the elements.

* Short Form Initialization
* Using an Array:

+ The size of the array is fixed after creation. Use a.length to access the length of the array.

+ The last element is accessed using a[a.length - 1].

* Aliasing:

+ If you assign one array to another, both reference the same array, which can lead to aliasing.

+ To copy an array, you need to create a new array and copy each element.

* Two-Dimensional Arrays:

+ A two-dimensional array is an array of one-dimensional arrays

+ Access elements using: a[i][j].

* Array Processing:

+ Find the maximum value

+ Compute the average

+ Copy to another array

+ Reverse the elements

+ Matrix multiplication

* **APIs**
* API Concept:

+ An API (Application Programming Interface) provides documentation for methods in a library that can be utilized by other programs.

+ Client: Refers to a program that calls methods in another library.

+ Implementation: The Java code that implements the methods defined in an API.

* Libraries in Java:

+ Libraries provide useful functionality for various tasks, including mathematical operations and statistical calculations.

+ They enable abstraction, making client code clearer and easier to understand.

* Benefits of Using Libraries:

+ They ensure methods are well-tested and can handle exceptional conditions, promoting reliability in applications.

+ They facilitate the separation of client code from implementation, allowing for easier updates and substitutions of methods.

* Developing Personal Libraries:

+ Every program should be treated as a potential library for future reuse.

+ Creating a clear API is essential for defining how the methods in a library interact with client code.

* API Contract: An API serves as a contract between the client and the implementation, specifying the expected behavior of methods.
* **Strings**
  + String: A sequence of characters; the String data type is not a primitive type.
  + Concatenation: The + operator is used to combine strings.
  + Conversion: Use Integer.parseInt() and Double.parseDouble() to convert between strings and numbers.
  + Automatic Conversion: Java automatically converts data type values to strings when concatenated with an empty string ("").
  + Command-line Arguments: Strings are used to pass information to the main() method via the args array.
* **Input and output**
  + Purpose: Facilitates interaction between Java programs and the external environment through the standard library.
  + Input and Output:
    - Programs receive input from the command line or standard input stream.
    - Output is written to the standard output stream.
  + Interface Between Java and Operating System: Uses command-line arguments, input, and output streams for interaction.
  + StdOut Library: Provides methods like print(), println(), and printf() for output.
  + Standard Input: Utilizes StdIn with methods such as readInt(), readDouble(), and readString() for reading data.
  + Redirection and Piping: Allows output to be redirected to files or read from files.
  + Examples:
    - Write 1000 random values to a file: java RandomSeq 1000 100.0 200.0 > data.txt.
    - Read values from a file: java Average < data.txt.
* **Binary search**
  + Binary Search Algorithm:
    - Implemented in the rank() method, which takes an integer key and a sorted array a.
    - It returns the index of the key if present, or -1 if not.
    - Uses two variables, lo and hi, to narrow down the search interval. It checks the middle element and adjusts the interval based on comparisons.
  + Development Client:
    - The main() method reads integers from a whitelist file and checks against standard input.
    - Outputs integers not found in the whitelist.
  + Whitelisting Example:
    - Illustrates a scenario where a credit card company checks transactions against a list of valid account numbers.
  + Performance:
    - A brute-force approach to search each element would be inefficient for large datasets.
    - Binary search is much faster, especially with sorted data.
  + Data Abstraction and Object-Oriented Programming:
    - The text emphasizes the importance of data abstraction in modern programming.
    - It promotes code reuse, allows for flexible data structures, and helps define algorithmic challenges more clearly.