Let’s come and target FANG together

Day 2 of learning java

Topics we have covers

- **Variables**
- **Data Types**
- **Naming Convention**
- **Literals**
- **Keyword**
- **Scape Sequence**
- **User Input**

Variables: -

In Java, a variable is a named memory location that holds a value. Variables are used to store data that can be manipulated or accessed within a program. Here are some key points about variables in Java.

- **Declaration:** Variables must be declared before they can be used. Declaration involves specifying the variable's name and its data type.

  For example:

  int age; Declaration of an integer variable named 'age'

  double salary; Declaration of a double variable named 'salary'

  String name; Declaration of a String variable named 'name'

- Initialization: Initialization is the process of assigning an initial value to a variable. Variables can be initialized at the time of declaration or later in the code.

For example:

int age = 30; Initialization of 'age' with the value 30

double salary = 50000.50; Initialization of 'salary' with the value 50000.50

String name = "John"; Initialization of 'name' with the value "John"

Int→ type of variables

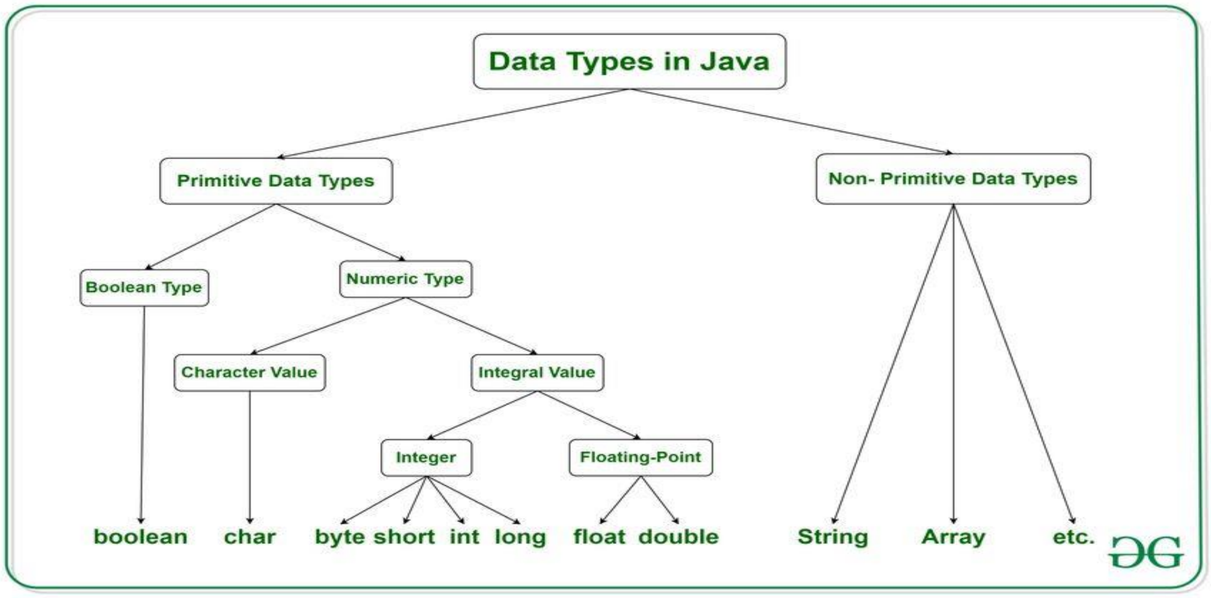
Age → name of variables

= → assigning operator

30 -→ value of to be stored in variable

Data Types: -

In Java, data types define the type of data that a variable can hold. Java supports two categories of data types: primitive data types and reference data types(Non-primitive data type).



Primitive Data Types:

Primitive data types are predefined by the language and are not objects. They are directly supported by the Java programming language and have corresponding wrapper classes (e.g., Integer, Double) that provide methods to work with them in an object-oriented manner.

1) byte: 8-bit signed integer. Range: -128 to 127.
2) short: 16-bit signed integer. Range: -32,768 to 32,767.
3) int: 32-bit signed integer. Range: -2^31 to 2^31-1.
4) long: 64-bit signed integer. Range: -2^63 to 2^63-1.
5) float: 32-bit floating-point number. Used for decimal numbers with single precision.
6) double: 64-bit floating-point number. Used for decimal numbers with double precision.
7) char: 16-bit Unicode character. Represents a single character.
8) boolean: Represents true or false values.

Reference Data Types:

Reference data types, on the other hand, are used to store references to objects in memory. They include types such as classes, interfaces, arrays, and enumerations. Unlike primitive data types, reference data types are created dynamically and are accessed via references.

1) String: Represents a sequence of characters.
2) Array: Represents a collection of elements of the same type.
3) Class: Represents user-defined types, such as objects instantiated from classes.

Naming Convention: - In Java, naming conventions are guidelines that developers follow to name classes, variables, methods, packages, and other elements of their code. Adhering to naming conventions improves code readability and maintainability, making it easier for other developers to understand and work with the codebase.

Packages: Package names are usually in lowercase and follow a reversed domain name convention, such as **com.example.myapp.**

Classes and Interfaces: Class and interface names should be nouns and written in UpperCamelCase, starting with a capital letter, such as **MyClass, UserInterface**.

Methods: Method names should be verbs or verb phrases and written in lowerCamelCase, starting with a lowercase letter, such as **calculateTotal(), getUserById().**

Variables: Variable names should be meaningful and written in lowerCamelCase, starting with a lowercase letter, such as **firstName, totalCount.**

Constants: Constants should be written in uppercase letters with underscores separating words, such as **MAX_SIZE, DEFAULT_TIMEOUT**.

Enums: Enum constants should be written in uppercase letters with underscores separating words, and enum types should be in UpperCamelCase, similar to constants.

Packages, modules, and files: Package names, module names, and file names should be lowercase and may include underscores or hyphens to improve readability, such as **utils, my_module, file_operations.java**.

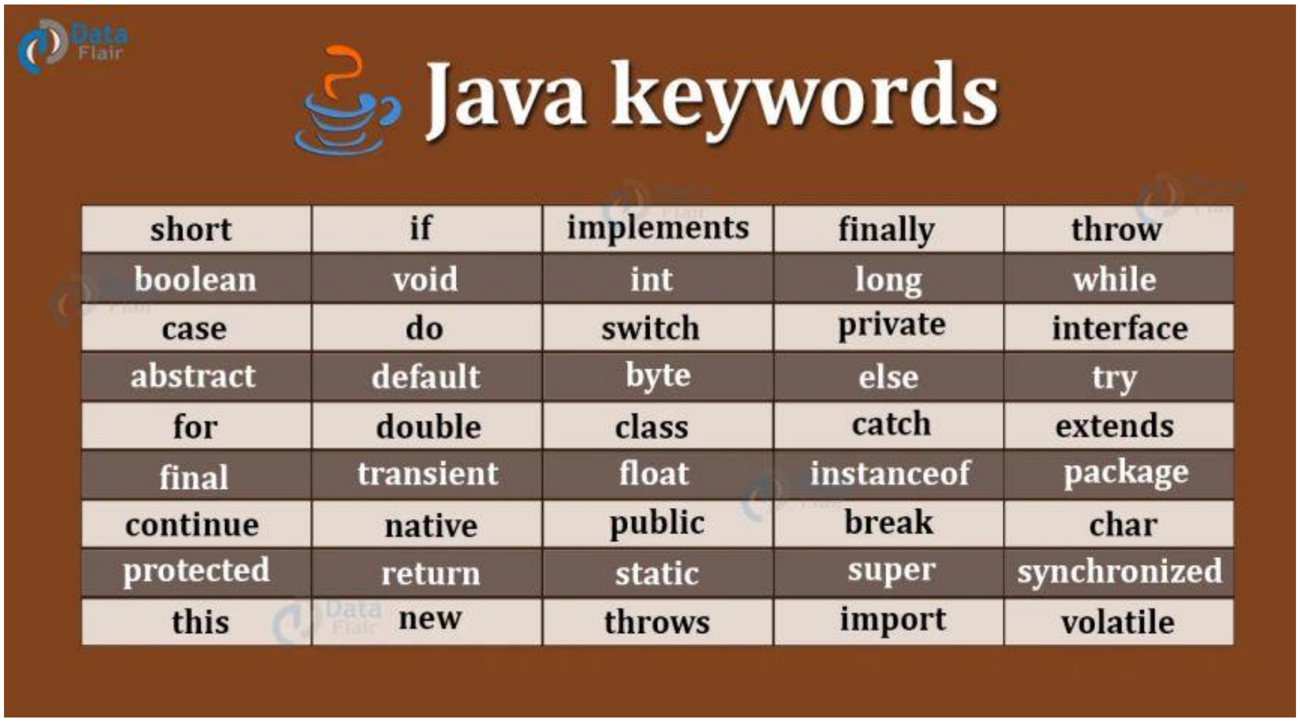
Acronyms and abbreviations: Acronyms and abbreviations should be treated as words in camel case. For example, XMLHttpRequest instead of XmlHTTPRequest.

Booleans: Boolean variables and methods that return a boolean value should typically start with "is", "has", "can", or similar prefixes, such as isActive(), canLogin().

Getter and Setter methods: Getter methods should start with "get" followed by the variable name, and setter methods should start with "set" followed by the variable name, both in lowerCamelCase, such as getName(), setName().

Keywords: -

In Java, keywords are reserved words that have predefined meanings and are part of the language syntax. These words are used to define the structure and logic of Java programs. Keywords cannot be used as identifiers (e.g., variable names, method names, class names) because they have special meanings assigned by the Java language.


Java keywords

short	if	implements	finally	throw
boolean	void	int	long	while
case	do	switch	private	interface
abstract	default	byte	else	try
for	double	class	catch	extends
final	transient	float	instanceof	package
continue	native	public	break	char
protected	return	static	super	synchronized
this	new	throws	import	volatile

These keywords have predefined meanings and specific roles in the Java language. Using them inappropriately or trying to redefine their meanings will result in compilation errors.

Literals: -In Java, a literal refers to a notation representing a fixed value in the source code. It's a way to directly express values within the code without having to compute or evaluate them at runtime. Java supports literals for various data types, including:

1) **Integer literals:** These represent whole numbers and can be written in decimal (base 10), octal (base 8, prefixed with 0), hexadecimal (base 16, prefixed with 0x or 0X), or binary (base 2, prefixed with 0b or 0B) notation. For example:

    Decimal: int num = 42;

    Octal: int octalNum = 052; // Octal representation of 42

    Hexadecimal: int hexNum = 0x2A; // Hexadecimal representation of 42

    Binary: int binaryNum = 0b101010; // Binary representation of 42

2) **Floating-point literals:** These represent fractional numbers and can be expressed in decimal form with or without scientific notation. For example:

    double num = 3.14;

    float pi = 3.14f; // Note the 'f' suffix to indicate it's a float

    Character literals: These represent single characters enclosed within single quotes. For example:

    char letter = 'A';

3) **String literals:** These represent sequences of characters enclosed within double quotes. For example:

```
String message = "Hello, World!";
```

4) **Boolean literals:** These represent the two boolean values, true and false. For example:

```
boolean flag = true;
```

5) **Null literal**: This represents the absence of a value and is denoted by the keyword null. For example:

```
String str = null;
```

Scape Sequence: -

Escape sequences in Java are special sequences of characters that begin with a backslash \ and are used to represent characters that are difficult to include directly in a string literal or to represent non-printable characters. Here are some commonly used escape sequences in Java:

Examples

- \n: Represents a newline character.

  ```
  System.out.println("Hello\nWorld");
  // Output:
  // Hello
  // World
  ```

- \t: Represents a tab character

  ```
  System.out.println("Hello\tWorld");
  // Output:
  // Hello	World
  ```

- \b: Represents a backspace character.

  System.out.println("Hello\bWorld");

  // Output:

  // HellWorld

UserInput: -

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        // Create a Scanner object to read input from the console
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter their name
        System.out.print("Enter your name: ");

        // Read the user's input as a string
        String name = scanner.nextLine();

        // Display a greeting message with the user's name
        System.out.println("Hello, " + name + "!");

        // Close the Scanner to release system resources
        scanner.close();
    }
}
```