Language Fundamentals

1. **Identifiers**
2. **Literals**
3. **Keywords/Reserved words**
4. **Operators**

1. Identifiers

Identifiers are symbolic names used to uniquely identify programming elements such as variables, functions, classes, and other entities in a program. They play a crucial role in making the code more readable, maintainable, and understandable by providing meaningful names to different elements, aiding both programmers and the compiler in effectively communicating the program's logic and structure.

OR

Identifier is a name assigned to the programming elements like variables, methods, classes, abstract classes, interfaces.

Rules and regulations For Identifiers:

- Identifiers should not be started with any number
- identifiers may be started with an alphabet, '_' symbol, '$' symbol,

Examples:

Valid Identifiers

1. userName
2. calculateSum
3. _counter
4. $totalAmount
5. StudentInfo
6. MAX_VALUE
7. isPrime
8. userInput
9. PI
10. Book

Invalid Identifiers

11. 123numbers (starts with a digit)
12. my-variable (contains a hyphen)
13. illegal@char (contains special characters)
14. class (a reserved keyword)
15. 3DArray (starts with a digit)
16. my space (contains whitespace)
17. void (a reserved keyword)
18. if (a reserved keyword)
19. @symbol (starts with a special character)
20. return (a reserved keyword)

Questions(Home Work):

int salary=10000
int 11salary=999
String _addr="Agra"
float $sal=10000.0f
String student11No="V132-1111"
String std_Name="Ashwani Upadhyay"
float emp$Sal=50000.0f

NOTE: Identifiers are not allowing all operators and all special symbols except '_' and '$' symbols

More Examples:

int stdNo=111; ========= valid
int std+No=111;=========Invalid
String std*Name="Ashwani";===== Invalid
String #stdAddr="Delhi";====Invalid
String std@id="123";===Invalid
float std.Fee=50000.0f;=====Invalid
String std-Addr="Hyd";======Invalid
String std_Addr="Hyd";======Valid

Note 2: Identifiers are not allowing spaces in the middle.

Valid Identifiers:

- myVariable
- totalAmount
- userInput
- className
- isPrime

Invalid Identifiers:

- my Variable (contains a space)
- illegal char (contains a space)
- user input (contains a space)
- class name (contains a space)
- invalid identifier (contains a space)

Note 3: Identifiers should not be duplicated with in the same scope, identifiers may be duplicated in two different scopes.

```
class Student {
        int num=10;-----> Class level
        short num=20;----> Error
        double sal=33.33---> No Error
        void print() {
                float sal=22.22f; -----> local variable
                double fsal=33.33;---> Error
                long num=30;---> No Error
        }
}
```

Note 4: In java applications, we can use all predefined class names and interface names as identifiers.

Examples:

1. String
2. List
3. Scanner
4. Math
5. Runnable
6. Exception
7. String

2. Literals

literals are fixed values that are directly used in your code. They are used to represent specific data types like numbers, characters, strings, and Boolean values. Literals make your code more readable and help you initialize variables and constants with specific values. Let's explore the different types of literals in Java.

OR

Literal is a constant assigned to the variables

Example
int a=10;
int ----> data types
a ------> variables/ identifier
= ------> Operator
10 -----> constant [Literal].
; ------> Special symbol.

Numeric Literals

Numeric literals represent numerical values. They can be categorized into integers and floating-point numbers.

Integers

An integer literal is a whole number without a decimal point. It can be written in different number bases

Decimal: Default base. Example: 123

Binary: Prefixed with 0b or 0B. Example: 0b1010 represents 10.

Octal: Prefixed with 0. Example: 075 represents 61.

Hexadecimal: Prefixed with 0x or 0X. Example: 0x1A represents 26.

Examples:

```
int decimal = 123;

int binary = 0b1010;

int octal = 075;

int hexadecimal = 0x1A;
```

Floating-Point Numbers

A floating-point literal represents a real number with a decimal point. It can be written in standard or scientific notation.

Examples:

double normal = 3.14;

float small = 2.0f;

Character and String Literals

Character literals represent single characters enclosed in single quotes ' '.

Examples:

char letterA = 'A';

char digit5 = '5';

String Literals

String literals represent sequences of characters enclosed in double quotes " ".

Examples:

String message = "Hello, World!";

String empty = "";

Boolean Literals

Boolean literals represent the two truth values: true and false.

Examples:

boolean isRaining = true;

boolean hasCoffee = false;

Escape Sequences

Escape sequences are used to represent special characters within character and string literals. They are written as a backslash \ followed by a character.

Common escape sequences include:

\' **for single quote**

\" **for double quote**

\\ **for backslash**

\n **for newline**

\t **for tab**

\r **for carriage return**

\b **for backspace**

Examples:

```
char singleQuote = '\'';
String quoteExample = "She said, \"Hello!\"";
String newLine = "First line.\nSecond line.";
```

Null Literal

The null literal represents the absence of a value. It is often used to indicate that a reference does not refer to any object.

Example:

String name = null;

Java Keywords (Reserved Words)

In Java programming, keywords are special words that have predefined meanings and functionalities. They are reserved and cannot be used as identifiers (such as variable names, class names, etc.). Keywords play a crucial role in defining the structure and behavior of your Java programs.

Data Types

3. **byte**: Represents a small integer data type.
4. **short**: Represents a short integer data type.
5. **int**: Represents an integer data type.
6. **long**: Represents a long integer data type.
7. **float**: Represents a floating-point number data type.
8. **double**: Represents a double-precision floating-point number data type.
9. **char**: Represents a character data type.
10. **boolean**: Represents a boolean data type.

Control Flow

1. **if**: Used for conditional branching in decision-making.
2. **else**: Part of the conditional branching for alternative paths.
3. **switch**: Enables multiway selection based on different values.
4. **case**: Specifies different cases within a switch statement.
5. **default**: Provides a default case within a switch statement.
6. **while**: Creates a loop that executes while a condition is true.
7. **do**: Starts a loop that executes at least once and then checks the condition.
8. **for**: Initiates a loop with initialization, condition, and increment/decrement.
9. **break**: Terminates the current loop or switch statement.
10. **continue**: Skips the rest of the loop iteration and starts the next one.

Classes and Objects

1. **class**: Declares a class.
2. **new**: Creates a new instance of a class.
3. **this**: Refers to the current instance of a class.
4. **super**: Refers to the parent class of a subclass.
5. **instanceof**: Checks if an object is an instance of a particular class.

Methods

1. **void**: Specifies that a method does not return any value.
2. **return**: Exits a method and returns a value to the caller.
3. **static**: Declares a method or variable as static (belonging to the class rather than instances).
4. **public**: Specifies the accessibility of a class, method, or variable as public (accessible from anywhere).
5. **private**: Specifies the accessibility of a class, method, or variable as private (accessible only within the same class).
6. **protected**: Specifies the accessibility of a class, method, or variable as protected (accessible within the same package and subclasses).

Exception Handling

1. try: Defines a block of code to be tested for exceptions.
2. catch: Catches an exception and specifies how to handle it.
3. finally: Defines a block of code to be executed after try or catch, regardless of whether an exception occurred.

Miscellaneous

1. **package**: Declares a package (a collection of related classes).

2. **import**: Imports classes from other packages for use in the current program.
3. **interface**: Declares an interface (a contract for classes to implement).
4. **extends**: Indicates inheritance, where a class inherits from another class.
5. **implements**: Specifies that a class implements an interface.
6. **throws**: Specifies that a method may throw a specific type of exception.

Java Operators

In Java programming, **operators** are special symbols or keywords that perform operations on operands (variables, values, or expressions). Operators play a critical role in manipulating data and controlling program flow. They allow you to perform tasks such as arithmetic operations, comparisons, and logical operations.

OR

Operator is a symbol; it will perform a particular operation over the provided operands.

Arithmetic Operators

Arithmetic operators perform basic mathematical operations.

Operator	Description	Example
+	Addition	int sum = 10 + 13;

Operator	Description	Example
-	Subtraction	int difference = 12 - 4;
*	Multiplication	int product = 2 * 2;
/	Division	int quotient = 12 / 2;
%	Modulus (remainder)	int remainder = 6 % 3;

Increment and Decrement Operators

Increment and decrement operators modify a variable's value by 1.

Operator	Description	Example
++	Increment	int incremented = x++;
--	Decrement	int decremented = y--;

Assignment Operators

Assignment operators assign values to variables.

Operator	Description	Example
=	Assign	int x = 10;
+=	Add and assign	x += 5; // x is now 15
-=	Subtract and assign	x -= 3; // x is now 12

Operator	Description	Example
*=	Multiply and assign	x *= 2; // x is now 24
/=	Divide and assign	x /= 4; // x is now 6
%=	Modulus and assign	x %= 3; // x is now 0

Comparison Operators

Comparison operators compare values and return a Boolean result.

Operator	Description	Example
==	Equal to	boolean isEqual = x == y;
!=	Not equal to	boolean notEqual = x != y;
<	Less than	boolean less = x < y;
>	Greater than	boolean greater = x > y;
<=	Less than or equal	boolean lessEqual = x <= y;
>=	Greater than or equal	boolean greaterEqual = x >= y;