Java programming

Content:

1. About Computer Programming
2. Introduction of Java
3. Java Program Structure
4. Java Program Compilation and Execution
5. Java Comments
6. Literals
7. Keywords and Variables
8. Data Types
9. Java Expressions
10. Expression Evaluation
11. Priority and Associativity
12. Types of Operators and examples

1. About Computer Programming:

Computer programming means the machine completing the work with set of instructions to complete task

1. Introduction of Java:

* Java is a popular, high-level programming language.
* Developed by Sun Microsystems in 1995.
* It allows developers to write code that can run on any device that has a Java Virtual Machine (JVM), making it "write once, run anywhere."
* Java is widely used for building web applications, mobile applications (especially Android apps), and large-scale enterprise systems.
* Its syntax is similar to C++, but it eliminates some of the complexities of C++ and focuses on ease of use and security.

1. Java Program Structure:

It includes:

1. Class Definition
2. Main method
3. Variables
4. Methods
5. Objects and Instantiation
6. Comments

Class Definition: This is a blueprint for creating objects. Every Java program has at least one class.

Ex: public class MyClass {

// Code goes here

}

Main method: This is where the program starts running.

Ex: public static void main(String[] args) {

// Code to run

}

Variables: Used to store data, like numbers or text.

Ex: int number = 10;

String text = "Hello!";

Methods: Methods are blocks of code that perform tasks and can return values.

Ex: public void myMethod() {

// Code to perform a task

}

Objects and Instantiation: Objects are instances of classes. To use a class, you typically create objects from it using the ‘new’ keyword.

Ex: MyClass obj = new MyClass();

**Comments**: Comments are used to explain the code and are not executed. They can be single-line (//) or multi-line (/\* ... \*/).

4.Java Program Compilation and Execution:

* Create a Java source file with a “.java” extension.
* Compile the Java Program using “javac“ command.
* Run the compiled Java program, using “java“ command.

**Note**: Ensure you have the JDK installed on your system.you can check this by running” java -version” and “javac -version” in command prompt.

5.Java Comments:

comments

single line Multi line

//…….// /\*………..\*/

(short explanations) (larger blocks)

6. Literals: Any constant value which can be assigned to the variable.

1) **Integer literals**: Numbers without a decimal point.

2)**Floating literals**: Numbers with a decimal point.

3) **Character literals :** A single letter, number, or symbol

4) **String Literals:** Sequence of characters enclosed with “ ”.

5) **Boolean Literals:** Values representing T or F condition.

6)**Null:** Represents the absence of any value or object.

7.Keywords and Variables: keywords are reserved words that have predefined meanings in the language. They cannot be used as identifiers (like variable names or method names). There are 50 keyword

**Keywords:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Abstract | char | else | goto | long | return |
| assert | class | enum | implements | new | short |
| Boolean | const | extends | if | null | static |
| break | continue | final | import | package | super |
| byte | default | finally | instanceof | private | switch |
| case | do | float | int | protected | this |
| catch | double | for | interface | public | throw |
| throws | transient | try | void | volatile | while |

**Variables**

Local instance static

Local variables:

**Where**: Inside a method or block.

**Scope**: Only within that method or block.

**Lifetime**: Exists only while the method is running.

**Initialization**: Must be initialized before use.

Instance variables:

**Where**: Inside a class but outside any method.

**Scope**: Accessible from any method in the class.

**Lifetime**: Exists as long as the object (instance) exists.

**Initialization**: Automatically initialized to default values.

Static Variables:

**Where**: Inside a class but outside any method, with the static keyword.

**Scope**: Accessible from any method in the class and shared among all instances of the class.

**Lifetime**: Exists as long as the class is loaded.

**Initialization**: Automatically initialized to default values if not explicitly initialized.

8. Data Types:



|  |  |  |
| --- | --- | --- |
| **Data Type** | **Default Value** | **Default size** |
| boolean | false | 1 bit |
| char | '\u0000' | 2 byte |
| byte | 0 | 1 byte |
| short | 0 | 2 byte |
| int | 0 | 4 byte |
| long | 0L | 8 byte |
| float | 0.0f | 4 byte |
| double | 0.0d | 1. byte |

9.Java Expressions: Expressions are combinations of variables, operators, and values that evaluate to a single result.

**Types of Epressions:**

Arithmetic Expression: a + b

Relational Expression: a > b

Logical Expression: (a > b) && (b > c)

Assignment Expression: a = 5

Conditional Expression: int max = (a > b) ? a : b;

10. Expression evaluation: the process of computing the value of an expression. The evaluation of an expression depends on the precedence and associativity.

11. Precedence and Associativity:

|  |  |  |
| --- | --- | --- |
| **Category** | **Operators** | **Associativity** |
| Postfix | ++ - - | Left to right |
| Unary | + - ! ~ ++ - - | Right to left |
| Multiplicative | \* / % | Left to right |
| Additive | + - | Left to right |
| Shift | << >> | Left to right |
| Relational | < <= > >= | Left to right |
| Equality | == != | Left to right |
| Bitwise AND | & | Left to right |
| Bitwise XOR | ^ | Left to right |
| Bitwise OR | | | Left to right |
| Logical AND | && | Left to right |
| Logical OR | || | Left to right |
| Conditional | ?: | Right to left |
| Assignment | = += -= \*= /= %=>>= <<= &= ^= |= | Right to left |

12.operators: Operators in Java are special symbols that perform specific operations on one, two, or three operands and then return a result.

\*There are many types of operators in Java they are:

|  |  |  |
| --- | --- | --- |
| **Operator Type** | **Category** | **Precedence** |
| Unary | postfix | *expr*++ *expr*-- |
| prefix | ++*expr* --*expr* +*expr* -*expr* ~ ! |
| Arithmetic | multiplicative | \* / % |
| additive | + - |
| Shift | shift | << >> >>> |
| Relational | comparison | < > <= >= instanceof |
| equality | == != |
| Bitwise | bitwise AND | & |
| bitwise exclusive OR | ^ |
| bitwise inclusive OR | | |
| Logical | logical AND | && |
| logical OR | || |
| Ternary | ternary | ? : |
| Assignment | assignment | = += -= \*= /= %= &= ^= |= <<= >>= >>>= |

**6-08-2024**

1. Java conditional / Control statements with example

2. Java conditional / Control loops with example

3. Jump statements and examples

1.Conditional and Control Statement:

control statements are used to control the flow of execution of the program based on certain conditions. the main types of conditional statements in Java:

* if
* if-else
* if-else if-else
* Switch

If-statement: The if statement executes a block of code only if a specified condition evaluates to true.

Syntax:

if (condition) {

// code to be executed if condition is true

}

Ex:

int age = 18;

if (age >= 18) {

System.out.println("You are an adult.");

}

If-else Statement: The if-else statement executes one block of code if a condition is true and another block if the condition is false.

Syntax:

if (condition) {

// code to be executed if condition is true

} else {

// code to be executed if condition is false

}

Ex:

int temperature = 30;

if (temperature > 25) {

System.out.println("It's warm outside.");

} else {

System.out.println("It's cool outside.");

}

Nested-If Statement: This structure is used to check multiple conditions. It executes the first true condition's code block and skips the rest.

Syntax:

if (condition1) {

// code to be executed if condition1 is true

} else if (condition2) {

// code to be executed if condition2 is true

} else {

// code to be executed if both condition1 and condition2 are false

}

Ex:

int score = 85;

if (score >= 90) {

System.out.println("Grade: A");

} else if (score >= 75) {

System.out.println("Grade: B");

} else if (score >= 60) {

System.out.println("Grade: C");

} else {

System.out.println("Grade: F");

}

Switch:The switch statement allows a variable to be tested for equality against a list of values, each with associated code.

Syntax:

switch (expression) {

case value1:

// code to be executed if expression == value1

break;

case value2:

// code to be executed if expression == value2

break;

// more cases

default:

// code to be executed if expression doesn't match any case

}

2.conditional / Control loops: control loops are used to execute a block of code multiple times, depending on a condition. Java provides several types of loops:

* For
* While
* do-while
* enhanced for loop.

For loop: used when the number of iterations is known. It consists of three parts: initialization, condition, and update.

**Syntax:** for (initialization; condition; update) {

// code to be executed

}

**Ex**: for (initialization; condition; update) {

// code to be executed

}

While loop: continues to execute a block of code as long as the specified condition is true.

**Syntax:** while (condition) {

// code to be executed

}

**Ex**: int count = 0;

while (count < 3) {

System.out.println("Count = " + count);

count++;

}

do-while loop: similar to the while loop, but it guarantees that the block of code is executed at least once before checking the condition.

Syntax: do {

// code to be executed

} while (condition);

Ex:

int num = 1;

do {

System.out.println("Number = " + num);

num++;

} while (num <= 3);

Enhanced-for loop: also known as the for-each loop, is used to iterate over elements in an array or a collection. It is useful when you do not need to keep track of the index.

Synatx:

for (type element : array/collection) {

// code to be executed

}

Ex: int[] numbers = {1, 2, 3, 4, 5};

for (int nu m : numbers) {

System.out.println(num);

}

3.Jump statements: In Java, jump statements are used to transfer control to other parts of the code.

Break continue return

break Statement: It breaks out of the loop or switch block and continues executing the code.

Syntax: for (int i = 0; i < 10; i++) {

if (i == 5) {

break; // exits the loop when i is 5

}

System.out.println("i = " + i);

}

System.out.println("Loop ended.");

Continue Statement: skips the current iteration of a loop and proceeds to the next iteration.

Ex: for (int i = 0; i < 10; i++) {

if (i % 2 == 0) {

continue; // skips the rest of the loop body when i is even

}

System.out.println("i = " + i);

}

Return: used to exit from the current method and optionally return a value.

Ex: public class Main {

public static void main(String[] args) {

System.out.println(calculateSum(10, 20));

}

public static int calculateSum(int a, int b) {

return a + b; // returns the sum of a and b

}

}