Java programming

**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.");

}

If-else if- else 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

}

}

08-08-2024

1.Java Methods: A method is a block of code.It’s a way to perform some tasks. It is used to achieve the reusability of code. We write a method once and use it many times.

**Method Declaration:** Provides information about method attributes, such as visibility, return-type, name, and arguments.



**Access Specifier**: Determines the visibility of the method. Common access modifiers include:

* public: The method can be accessed from any other class.
* private: The method can only be accessed within the class it is defined.
* protected: The method can be accessed within the same package or subclasses.
* No modifier (default): The method can be accessed within the same package.

**Return Type**: Indicates the type of value the method will return. If the method doesn’t return any value, it is declared as void.

**Method Name**: A unique identifier for the method. It should be descriptive and follow camelCase naming convention.

**Parameter List**: A set of inputs passed to the method. Parameters are optional, and if there are no parameters, you just use empty parentheses ().

**Method Body**: The block of code that defines what the method does. It is enclosed in curly braces {}.

Types of Method

Predefined user defined

Predefined Methods:Which is already defined in the Java class libraries is known as predefined methods. It is also known as the **standard library method** or **built-in method**. We can directly use these methods just by calling them in the program at any point. Some pre-defined methods are **length(), equals().**predefined method is defined inside a class. Such as print() method is defined in the **java.io.PrintStream** class.

Ex: **public** **class** Demo

{

**public** **static** **void** main(String[] args)

{

// using the max() method of Math class

System.out.print("The maximum number is: " + Math.max(9,7));

}  }

User defined methods: Method written by the user or programmer is known as **a user-defined** method. These methods are modified according to the requirement.

Ex: //user defined method

public static void findEvenOdd(**int** num)

{

//method body

**if**(num%2==0)

System.out.println(num+" is even");

**else**

System.out.println(num+" is odd");

}

How to Call or Invoke a User-defined Method

Instance method static Method

Accessor Method Mutator Method

Instance method: Calling instance method by

1. **Creating an object** of the class
2. **Use the object** to call the method

Ex:

public class Calculator {

public int add(int a, int b) { // This is the user-defined method

return a + b;

}

}

public class Main {

public static void main(String[] args) {

Calculator calc = new Calculator(); // Step 1: Create an object

int sum = calc.add(5, 3); // Step 2: Call the method

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

}

}

Static Method: **Directly call the method** using the class name (no need to create an object).

EX: public class MathUtils {

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

// This is the user-defined method

return a \* b;

}

}

public class Main {

public static void main(String[] args) {

int product = MathUtils.multiply(4, 5);

// Call the method using the class name

System.out.println("Product: " + product);

// Output: Product: 20

}

}

**Accessor Method:** The method(s) that reads the instance variable(s) is known as the accessor method. We can easily identify it because the method is prefixed with the word **get**. It is also known as **getters**. It returns the value of the private field. It is used to get the value of the private field.

**Example**

**public** **int** getId()

{

**return** Id;

}

**Mutator Method:** The method(s) read the instance variable(s) and also modify the values. We can easily identify it because the method is prefixed with the word **set**. It is also known as **setters** or **modifiers**. It does not return anything. It accepts a parameter of the same data type that depends on the field. It is used to set the value of the private field.

**Example**

**public** **void** setRoll(**int** roll)

{

**this**.roll = roll;  }

Recursion in Java:**Recursion** is a process in which a method calls itself to solve a smaller version of the problem until it reaches a base condition.

Ex: public class Factorial {

public int factorial(int n) {

if (n == 0) { // Base condition

return 1;

} else {

return n \* factorial(n - 1); // Recursive call

}

}

public static void main(String[] args) {

Factorial fact = new Factorial();

int result = fact.factorial(5); // 5! = 5 \* 4 \* 3 \* 2 \* 1 = 120

System.out.println("Factorial: " + result);

}

}

Memory updated

**Recursion in Java**

**Recursion** is a process in which a method calls itself to solve a smaller version of the problem until it reaches a base condition.

**Example**: Calculating the factorial of a number.

java

Copy code

public class Factorial {

public int factorial(int n) {

if (n == 0) { // Base condition

return 1;

} else {

return n \* factorial(n - 1); // Recursive call

}

}

public static void main(String[] args) {

Factorial fact = new Factorial();

int result = fact.factorial(5); // 5! = 5 \* 4 \* 3 \* 2 \* 1 = 120

System.out.println("Factorial: " + result);

}

}

The method factorial keeps calling itself with a smaller value of n until n is 0. At that point, it stops and returns the result.

Java Object-Oriented Programming (OOP):

It is based on the Object-Oriented Programming (OOP) paradigm, which organizes software design around objects rather than functions or logic. The main concepts of OOP are:

1. **Encapsulation**
2. **Inheritance**
3. **Polymorphism**
4. **Abstraction**

**Class:** A class is a blueprint or template for creating objects. It defines the properties (attributes) and methods (behaviors)

EX: public class Car { // Class definition

String model; // Class property

int year; // Class property

void startEngine() { // Method

System.out.println("Engine started");

}

}

**Class Properties (Attributes):**Properties are the variables that belong to a class. They represent the state or characteristics of an object.

**Ex** In the Car class above, model and year are properties.

**Objects:**An object is an instance of a class. It has a state (defined by properties) and behavior (defined by methods).

**How to Create an Object**: You create an object using the new keyword.

Ex: public class Main {

public static void main(String[] args) {

Car myCar = new Car(); // Object creation

myCar.model = "Tesla"; // Setting property values

myCar.year = 2023;

myCar.startEngine(); // Calling a method on the object

}

}

**new Keyword:** The new keyword is used to create new objects from a class.

**Ex**: Car myCar = new Car(); creates a new Car object.

**Constructor:** A constructor is a special method used to initialize objects when they are created. It has the same name as the class and no return type.

Purpose: To set up the initial state of an object by assigning values to its properties.

EX:

public class Dog {

String name;

int age;

// Constructor

public Dog(String name, int age) {

this.name = name; // Set the 'name' property

this.age = age; // Set the 'age' property

}

void bark() {

System.out.println(name + " is barking!");

}

}

public class Main {

public static void main(String[] args) {

// Create a Dog object using the constructor

Dog myDog = new Dog("Buddy", 3);

// Use the object

System.out.println("Dog's name: " + myDog.name); // Output: Dog's name: Buddy

System.out.println("Dog's age: " + myDog.age); // Output: Dog's age: 3

myDog.bark(); // Output: Buddy is barking!

}

}

**instanceof Keyword: It** is used to check whether an object is an instance of a specific class or subclass.

Ex: public class Main {

public static void main(String[] args) {

Car myCar = new Car("Tesla", 2023);

System.out.println(myCar instanceof Car); // Returns true

}

}