**Variables in Java**

* Local
* Static
* Instance

In Java, a variable is a container that holds data that can be manipulated and referenced in a program. Java supports various types of variables, including primitive data types and reference types. Here are some common types of variables in Java:

1. \*\*Primitive Data Types:\*\*

- \*\*byte:\*\* 8-bit signed integer.

- \*\*short:\*\* 16-bit signed integer.

- \*\*int:\*\* 32-bit signed integer. (-2,14,74,83,648 to 2,14,74,83,647)

- \*\*long:\*\* 64-bit signed integer.

- \*\*float:\*\* 32-bit floating-point number.

- \*\*double:\*\* 64-bit floating-point number.

- \*\*char:\*\* 16-bit Unicode character.

- \*\*boolean:\*\* Represents true or false.

Example of primitive variable declaration and initialization:

```java

int age = 25;

double salary = 50000.5;

char grade = 'A';

boolean isStudent = true;

```

2. \*\*Reference Types:\*\*

Reference types include objects, arrays, and interfaces. Unlike primitive types, reference types don't store the actual data but store a reference (memory address) to the location where the data is stored.

Example of reference variable declaration and instantiation:

```java

String name; // Declaration

name = "John"; // Initialization

// Object instantiation

MyClass myObject = new MyClass();

```

3. \*\*Arrays:\*\*

An array is a container object that holds a fixed number of values of a single data type.

Example of array declaration and initialization:

```java

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

```

4. \*\*Constants:\*\*

Constants are declared using the `final` keyword, and their values cannot be changed once assigned.

Example of a constant:

```java

final double PI = 3.14;

```

5. \*\*Static Variables:\*\*

Static variables belong to the class rather than instances of the class. They are declared with the `static` keyword.

Example of a static variable:

```java

class Example {

static int count = 0;

}

```

6. \*\*Instance Variables:\*\*

Instance variables belong to instances of a class and are not marked as `static`. Each instance of the class has its own copy of instance variables.

Example of an instance variable:

```java

class Example {

int value;

}

Sample

**public** **class** javabasics {

**int** a=2147483647;//instance

**final** **int** b=9;

**static** **int** *i*=10;//static

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

// **TODO** Auto-generated method stub

javabasics jb=**new** javabasics();

System.***out***.println(*i*);

System.***out***.println(jb.a);

System.***out***.println(Integer.***MAX\_VALUE***);

jb.test();

}

**void** test() {

**int** a=1;

**int** b=2;//local

System.***out***.println(a+b);

}

}

LOOPS

In Java, loops are used to repeatedly execute a block of code as long as a certain condition is true or for a specified number of iterations. Java supports three types of loops: `for`, `while`, and `do-while`. Here's an overview of each:

**### 1. \*\*for Loop:\*\***

The `for` loop is used when the number of iterations is known in advance.

```java

for (initialization; condition; update) {

// code to be repeated

}

```

- \*\*Initialization:\*\* Executed once at the beginning.

- \*\*Condition:\*\* Evaluated before each iteration. If false, the loop exits.

- \*\*Update:\*\* Executed after each iteration.

Example:

```java

for (int i = 1; i <= 5; i++) {

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

}

```

**### 2. \*\*while Loop:\*\***

The `while` loop is used when the number of iterations is not known in advance and depends on a condition.

```java

while (condition) {

// code to be repeated

}

```

Example:

```java

int count = 1;

while (count <= 5) {

System.out.println("Iteration " + count);

count++;

}

```

**### 3. \*\*do-while Loop:\*\***

The `do-while` loop is similar to the `while` loop, but the condition is checked after the block of code is executed. Therefore, the block of code is guaranteed to execute at least once.

```java

do {

// code to be repeated

} while (condition);

```

Example:

```java

int count = 1;

do {

System.out.println("Iteration " + count);

count++;

} while (count <= 5);

```

Loops are essential for automating repetitive tasks and iterating over collections of data. They help in making the code more concise and efficient. It's important to ensure that the loop condition is properly managed to avoid infinite loops.

**### 3. \*\*for-each Loop:\*\***

The enhanced for loop, also known as the "for-each" loop, was introduced in Java 5. It provides a simpler way to iterate over arrays and collections (such as lists, sets, and maps) without the need for explicit indexing. The syntax of the enhanced for loop is as follows:

```java

for (elementType element : iterable) {

// code to be executed for each element

}

```

- \*\*elementType:\*\* The data type of the elements in the collection or array.

- \*\*element:\*\* A variable that represents the current element in the iteration.

- \*\*iterable:\*\* The collection or array being iterated.

Here are a couple of examples to illustrate the usage of the enhanced for loop:

### 1. Iterating over an Array:

```java

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

// Enhanced for loop to iterate over the array

for (int number : numbers) {

System.out.println(number);

}

```

### 2. Iterating over a Collection (List):

```java

import java.util.ArrayList;

import java.util.List;

List<String> names = new ArrayList<>();

names.add("Alice");

names.add("Bob");

names.add("Charlie");

// Enhanced for loop to iterate over the list

for (String name : names) {

System.out.println(name);

}

```

The enhanced for loop simplifies the process of iterating over elements in arrays and collections, making the code more readable and less error-prone. It automatically handles the initialization, condition checking, and iteration over the elements, abstracting away the details of indexing.

It's important to note that the enhanced for loop is suitable when you want to iterate over all elements sequentially. If you need access to the index or want more control over the iteration process, you may still choose the traditional `for` loop.

Arrays

In Java, an array is a data structure that allows you to store multiple values of the same data type under a single variable name. Each element in the array is identified by its index or position, starting from 0. Arrays provide a convenient way to work with collections of data.

Here are the key aspects of arrays in Java:

### 1. \*\*Declaring an Array:\*\*

To declare an array in Java, you specify the data type of the elements followed by the array variable name and square brackets `[]`. For example:

```java

// Declaration of an integer array

int[] numbers;

// Declaration and initialization of a string array

String[] names = new String[5]; // Creates an array of size 5

```

### 2. \*\*Initializing an Array:\*\*

Arrays can be initialized at the time of declaration or later using the `new` keyword:

```java

// Initializing an integer array at the time of declaration

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

// Initializing a double array later

double[] prices = new double[3];

prices[0] = 19.99;

prices[1] = 29.99;

prices[2] = 39.99;

```

### 3. \*\*Accessing Elements:\*\*

Array elements are accessed using their index:

```java

int[] numbers = {10, 20, 30, 40, 50};

int firstElement = numbers[0]; // Accessing the first element (index 0)

```

### 4. \*\*Array Length:\*\*

The length of an array can be obtained using the `length` property:

```java

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

int arrayLength = numbers.length; // Returns 5

```

### 5. \*\*Iterating Through an Array:\*\*

You can use loops to iterate through the elements of an array:

```java

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

// Using a for loop

for (int i = 0; i < numbers.length; i++) {

System.out.println(numbers[i]);

}

// Using an enhanced for loop (for-each)

for (int number : numbers) {

System.out.println(number);