

Continue



Search App Architecture Search Software Quality Search Cloud Computing Search Security Search AWS Why doesn't Java have a 'const' feature for defining constants like C++? Java does not include a 'const' feature for defining constants, instead offering the 'final' keyword for this purpose. This design decision reflects Java's philosophy prioritizing readability, maintainability, and clarity in coding. Copied final int MAX_VALUE = 100; enum Days { MONDAY, TUESDAY, WEDNESDAY } Causes Historical Development: Java was developed with an emphasis on simplicity compared to C++, leading to design choices that avoid potential confusion. Final Keyword: The 'final' keyword in Java serves as a way to declare constants, requiring explicit use which improves code clarity. No Need for Multiple Constant Definitions: Unlike C++, Java accomplishes similar goals using other constructs, such as final variables and enumerations. Solutions Use the 'final' keyword to declare constants. This approach improves code clarity and conveys the intention of immutability. Consider using enumerations (enums) in cases where a set of related constants is needed. Mistake: Using 'const' instead of 'final' in Java. Solution: Always use the 'final' keyword to declare constants in Java. Mistake: Assuming final variables can be modified after assignment. Solution: Remember that once assigned, the value of a final variable cannot be changed. Java constants Java const keyword Final keyword in Java Differences between C++ and Java Java programming best practices Copyright 2025 - CodingTechRoom.com Constant in programming is a variable that never changes. Today I'll tell you about declaring constants in Java. Java doesn't have a special keyword to define a constant. const is reserved keyword (you can't use it as a name of a variable for example), but its unused. So to declare a constant in Java you have to add static final modifiers to a class field. Example: public static final String BASE_PATH = "/api/"; You should follow Java constant naming convention all constant variables should be in upper case, words should be separated by the underscore. Sometimes programmers are defining constants in a separate class in Java First of all, its a really bad idea to create a single class for all constants in your project. Constants should be related to each other. For example, I have a UserController, that contains endpoints to work with users. Each endpoint has a binding to specific URL. So I can create a class with Java string constants that contains user controller bindings. Example: package com.explainjava; public final class UserBinding { private UserBinding() { } public static final String BASE_PATH = "/api/users/"; public static final String FIND_ONE = BASE_PATH + "/" + "{id}"; public static final String FIND_PREFERENCES = BASE_PATH + "/" + "{id}/preferences/"; } The best practice of defining constants class in Java is: Add a final attribute to class to restrict inheritance. Add a private no-args constructor to forbid new instance creation. All fields in the interface are constants. By default, each variable in the interface is public static final and you can't change it. I'll change our UserBinding to an interface. Example: package com.explainjava; public interface UserBindings { String BASE_PATH = "/api/users/"; String FIND_ONE = BASE_PATH + "/" + "{id}"; String FIND_PREFERENCES = BASE_PATH + "/" + "{id}/preferences/"; } But defining constants in the interface is a bad practice. Its called Constant Interface Antipattern. Joshua Bloch in his book Effective Java said: The constant interface pattern is a poor use of interfaces. That a class uses some constants internally is an implementation detail. Implementing a constant interface causes this implementation detail to leak into the classes exported API. It is of no consequence to the users of a class that the class implements a constant interface. In fact, it may even confuse them. Worse, it represents a commitment: if in a future release the class is modified so that it no longer needs to use the constants, it still must implement the interface to ensure binary compatibility. If a nonfinal class implements a constant interface, all of its subclasses will have their namespaces polluted by the constants in the interface. There are several constant interfaces in the Java platform libraries, such as java.io.ObjectStreamConstants. These interfaces should be regarded as anomalies and should not be emulated. So my recommendation is to use class for constants. If you're using IntelliJ IDEA to write a code I have 3 live templates for you: psf + TAB generates public static final. psfs + TAB generates public static final String. psfi + TAB generates public static final int. Learn your favorite IDE shortcuts and live templates to increase your productivity. Share copy and redistribute the material in any medium or format for any purpose, even commercially. Adapt remix, transform, and build upon the material for any purpose, even commercially. The licensor cannot revoke these freedoms as long as you follow the license terms. Attribution You must give appropriate credit , provide a link to the license, and indicate if changes were made . You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use. ShareAlike If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original. No additional restrictions You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits. You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation . No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material. Java is a widely-used, object-oriented programming language known for its robustness, portability, and security features. While it shares similarities with languages like C and C++, it also introduces its own conventions and restrictions. Two notable keywords that differ in Java compared to these languages are const and goto. This article explores why these keywords are not directly supported in Java, alternative approaches used in Java programming, and the principles underlying their absence. In languages like C and C++, the const keyword is used to define constant variables whose values cannot be changed once assigned. For example: const int MAX_SIZE = 100; However, in Java, the const keyword is not used for defining constants. Instead, constants are typically declared using the final keyword. For instance: final int MAX_SIZE = 100; The final keyword serves a dual purpose in Java, both for constants and for declaring that a class cannot be subclassed or a method cannot be overridden, depending on where it is used. The Absence of goto in Java The goto statement, popular in languages like C and BASIC, allows for unconditional jumps to another part of the code. For example: int count = 0; loop: while (count < 10) { count++; if (count == 5) goto loop; } // Jump back to the 'loop' label Rupam Yadav Oct 12, 2023 Java Java Constant This article will introduce the topic of the Java constant. A constant value is a value that cannot be changed once it is assigned. Java has the word const reserved, but it does not provide any functionality to implement and use constants. But there are other ways that we can use to use constants, like using the keywords static and final that we are going to see in the following examples. Declare and Use Constant Using private, static, and final Keywords in Java In the first example, we use the two keywords - static and final with the private access modifier. We have a class, and inside it is a constant MIN_VOTING_AGE is declared and initialized. We ask the user to enter his/her age to check the eligibility for voting. Once the user inputs age, we check if it is greater than the constant MIN_VOTING_AGE. Then it shows the appropriate output according to the condition. The private keyword ensures that the constant is not accessible outside the class. import java.util.Scanner; public class JavaConstants { private static final int MIN_VOTING_AGE = 18; public static void main(String[] args) { System.out.println("Enter your age and check if you are old enough to vote."); Scanner scanner = new Scanner(System.in); int age = scanner.nextInt(); if (age > MIN_VOTING_AGE) { System.out.println("Congrats, You are eligible to vote."); } else { System.out.println("You are not eligible to vote yet."); } } } Output: Enter your age and check if you are old enough to vote: 12 You are not eligible to vote yet. Declare and Use Constant Using public, static, and final Keywords in Java This example uses static and final keywords but with the public access modifier. We create a class, AnotherClass, that has the constant MIN_VOTING_AGE defined using public static final int. As the constant is public, we can use the constant in the JavaConstants class to access the constant, and we call it using AnotherClass.MIN_VOTING_AGE as it is static. import java.util.Scanner; public class JavaConstants { public static void main(String[] args) { System.out.println("Enter your age and check if you are old enough to vote."); Scanner scanner = new Scanner(System.in); int age = scanner.nextInt(); if (age > AnotherClass.MIN_VOTING_AGE) { System.out.println("Congrats, You are eligible to vote"); } } } else { System.out.println("You are not eligible to vote yet."); } } } } class AnotherClass { public static final int MIN_VOTING_AGE = 18; } Output: Enter your age and check if you old enough to vote: 23 Congrats, You are eligible to vote As the name suggests, a constant is an entity in programming that is immutable. In other words, the value that cannot be changed. Usually, to accomplish this, the variable is declared using the final keyword. Constants are frequently used to represent stable values, like mathematical constants, configuration settings, or flag values, that do not change while a program is running. A variable's value is guaranteed to stay constant and unintentionally changed if it is declared as a constant. For easier reading, constants are usually defined in uppercase and underscores between words. Furthermore, variables can be specified either at the instance level (non-static constants) or at the class level (static constants). What is constant? Constant is a value that cannot be changed after assigning it. Java does not directly support the constants. There is an alternative way to define the constants in Java by using the non-access modifiers static and final. How to declare constant in Java? In Java, to declare any variable as constant, we use static and final modifiers. It is also known as non-access modifiers. According to the Java naming convention the identifier name must be in capital letters. Static and Final Modifiers The purpose to use the static modifier is to manage the memory. It also allows the variable to be available without loading any instance of the class in which it is defined. The final modifier represents that the value of the variable cannot be changed. It also makes the primitive data type immutable or unchangeable. The syntax to declare a constant is as follows: static final datatype identifier name = value; For example, price is a variable that we want to make constant. static final double PRICE = 432.78; Where static and final are the non-access modifiers. The double is the data type and PRICE is the identifier name in which the value 432.78 is assigned. In the above statement, the static modifier causes the variable to be available without an instance of its defining class being loaded and the final modifier makes the variable fixed. Here a question arises that why we use both static and final modifiers to declare a constant? If we declare a variable as static, all the objects of the class (in which constant is defined) will be able to access the variable and can be changed its value. To overcome this problem, we use the final modifier with a static modifier. When the variable defined as final, the multiple instances of the same constant value will be created for every different object which is not desirable. When we use static and final modifiers together, the variable remains static and can be initialized once. Therefore, to declare a variable as constant, we use both static and final modifiers. It shares a common memory location for all objects of its containing class. Why we use constants? The use of constants in programming makes the program easy and understandable which can be easily understood by others. It also affects the performance because a constant variable is cached by both JVM and the application. Points to Remember: Write the identifier name in capital letters that we want to declare as constant. For example, MAX = 12. If we use the private access specifier before the constant name, the value of the constant cannot be changed in that particular class. If we use the public access specifier before the constant name, the value of the constant can be changed in the program. Let's see some examples in which we have used constants. Example 1: Declaring Constant as Private Constant Example 1 java import java.util.Scanner; public class ConstantExample1 { // declaring constant private static final double PRICE = 234.90; public static void main(String[] args) { int unit; double total; bill; System.out.println("Enter the number of units you have used."); Scanner sc = new Scanner(System.in); unit = sc.nextInt(); total = unit * PRICE; System.out.println("The total amount you have to deposit is: " + total + " bill."); } } Output: Explanation A class with the main() method to determine the total cost based on a fixed price per unit is defined by the Java program ConstantExample1. To guarantee its immutability, it declares a constant PRICE with the value 234.90 by employing the last keyword. The application asks the user to enter the number of units used, uses a scanner to read the input, multiplies the price per unit (PRICE) by the quantity of units used to determine the total bill, and then shows the total amount that needs to be deposited. It illustrates how to describe a fixed value that doesn't change during program execution in Java using a constant. Example 2: Constant Example 2 java public class ConstantExample2 { private static final double PRICE = 2999; public static void main(String[] args) { System.out.println("Old Price of Iron: " + PRICE); ConstantExample obj = new ConstantExample(); obj.showPrice(); } } class ConstantExample { private static final double PRICE = 3599; void showPrice() { System.out.println("New Price of Iron: " + PRICE); } } Output: Explanation In the above program, we have defined two classes named ConstantExample2 and ConstantExample. The PRICE constant that has the value 2999, is declared in the ConstantExample2 class. The main() method of this class uses this constant to print the previous price of iron. The showPrice() function of the ConstantExample class is also called upon the creation of an instance of the class. Another constant, PRICE, is declared in the ConstantExample class with a value of 3599. The showPrice() method uses this constant to output the current iron price. Example 3: Declaring Constant as Public In the following example, we have declared constant PI as public. Inside the main() method, we have assigned 3.15 to the constant PI. After that, we have invoked the printValue() method. When we execute the program, it shows an error cannot assign a value to the final variable PI. ConstantExample3 java public class ConstantExample3 { // declaring PI as constant public static final double PI = 3.14; public static void main(String[] args) { printValue(); } } // trying to assign 3.15 in the constant PIPI = 3.15; printValue(); void printValue() { System.out.println("The value of PI cannot be changed to " + PI); } } Output: Explanation ConstantExample3, a Java program, defines a class that has two methods: printValue() and main. To assure its immutability, it uses the final keyword to declare a constant PI with the value 3.14. To print the initial value of PI, the printValue() method is called in the main method. It then tries to give the constant PI a new value (3.15), which is prohibited because it has been declared final. It would give rise to an error in compilation. In order to emphasize the immutability of constants in Java, the printValue() method is called once more to show that the value of PI cannot be altered. Using Enumeration (enum) as Constant It is the same as the final variables. It is a list of constants. Java provides the enum keyword to define the enumeration. It defines a class type by making enumeration in the class that may contain instance variables, methods, and constructors. Example of Enumeration public class EnumExample { // defining the enum public enum Color { Red, Green, Blue, Purple, Black, White, Pink, Gray } public static void main(String[] args) { // traversing the enum for (Color c : Color.values()) System.out.println(c); } } Output: Explanation The Java program EnumExample defines a class with a nested enum named Color. The predefined colors Red, Green, Blue, Purple, Black, White, Pink, and Grey are represented by this enum. The values() function is used in the main() method for each loop to iterate over the Colour enum's values and print each color to the console. It shows how to use an extended for loop to iterate over the elements of an enum and how to use enums in Java to express a fixed collection of constants. Conclusion To sum up, constants are essential to Java programming since they provide immutable values that do not change while a program runs. Variables can be declared as constants, guaranteeing that their values cannot be changed once assigned by combining the final and static modifiers. It increases the readability of the code, preserves program integrity, and boosts performance because the application and JVM both cache constant variables. Constants also follow the naming guidelines for Java variables that usually include uppercase letters and underscores for easier reading. Additionally, enum give a structured method of representing fixed values in programs by making it easy to establish a group of linked constants. Java programmers can produce codebases that are more reliable and manageable by comprehending and making good use of constants. Next Topic Java program to find the missing element in a given permutation The const keyword is a means of marking a variable as giving "read-only access" in C++. What is the equivalent of const in Java? The Java equivalent of const depends on the type of variable: primitive or object and on what you want to make constant: the object contents or the "pointer" (reference in Java). The situation can be roughly summed up as follows: Nearest equivalents to the C++ const keyword in Java Variable type Constant object/variable contents Constant pointer Primitive final variable final array reference (see below) Object Use an immutable object or create a subclass that forces immutability. final object reference The Java final keyword First, the combinations that are relatively easy to immitate in Java. The Java final modifier marks a variable as unmodifiable once it has been set. If it is a primitive variable such as an int, then this makes the actual value unmodifiable; if it is an object reference, then the reference can't be changed, but the object itself can still be modified if it provides a means to do so: final int x; x = 7; // OK : first time the var is set x = 8; // NOT OK : can't modify once set final Rectangle r = new Rectangle(0, 0, 20, 20); r.x = 10; // OK: it's the reference // that's final = new Rectangle(...); // NOT OK: can't // change the reference If the final variable is an instance variable (a variable that belongs to an instance of an object), then the contract is that it must have a value set (and only once!) by the time the constructor exits. There's no real equivalent to the constant pointer to a primitive, since Java doesn't allow pointers to primitives in the first place. (Strictly speaking, object references aren't quite like pointers either, but we consider them functionally similar here.) The nearest equivalent in Java is to use a single-element primitive array. So the equivalent to this in C++ would be to do this in Java: int[] val = new int[1]; val[0] = 3; Making the array reference final has essentially the same semantics as a const primitive pointer: final int[] val = new int[1]; val[0] = 3; // OK val = new int[1]; // Not OK : can't change final reference Array with unmodifiable contents? Java doesn't provide the notion of a const array; that is, an array that can be swapped for a new array (in C++ terms, "the pointer can be modified"), but whose elements can't be changed. However, if you need this functionality, the solution is generally much like providing read-only access to any other object as discussed below. So a couple of possibilities are: you can create an unmodifiable list by passing a list into Collections.unmodifiableList() (though in this case, the variable would be declared of type List, not of a type that marked it as "unmodifiable" as discussed below, any attempt to modify it would be spotted at runtime); you can create a wrapper object around a private array, and provide public methods to read but not write its elements; you can use a read-only IntBuffer (or FloatBuffer etc): To create a read-only IntBuffer, we first create the buffer, fill it, then call asReadOnlyBuffer() to create a reference to an object that provides read-only access: IntBuffer ib = IntBuffer.allocate(10); ib.put(...). IntBuffer constBuffer = ib.asReadOnlyBuffer(); Immutable objects One feature of the C++ const keyword is that it allows the caller to force read-only access on an object that would otherwise be modifiable. Arguably, such a design makes a bit more sense in C++, which still provides structs that have no way to enforce an access policy on its members. In Java, the general ideology is that access policy to the state of an object is controlled by the class. In other words, the general way to make a field "read-only" is to make it private and then not provide any public method to set it! So what if we have an object type that is normally mutable, but we want to make an instance of it immutable "on this occasion"? Well, we're essentially forced to use or create a subclass of the object in question. In the subclass, any methods that would normally update the state of the object can be "disabled": for collections, various static methods on the Collections class already provide such functionality, such as Collections.unmodifiableList(), for other object types, we'll generally need to create our own subclass and "remove" the set methods by overriding them to throw UnsupportedOperationException. So for example, an equivalent of this: myFunction(const l) { // ... // compiler won't let us accidentally modify l } in Java would be: myFunction(List l) { l = Collections.unmodifiableList(); } // ... / at runtime, we can't modify l } As noted in the comments, a difference is that an attempt to access the unmodifiable object will be caught at runtime rather than compile time (as is the case with the const keyword). Making public fields immutable? If a field on an object is declared public, there's really no way to prevent modification of that field. This means that there's no way to subclass and make immutable an object such as Rectangle, whose fields x, y, width and height are all public (with hindsight, probably bad design!). So if we have a method that uses such an object, we have a couple of options depending on why we wanted const in the first place: If we want to prevent accidentally updating the object in question from our own method, then we could create our own accessor class that is a "read-only wrapper" around the object in question, then only ever use the accessor class. Usually, this is too long-winded to be worthwhile it's a fairly heavy solution to spotting what's often a fairly rare bug. But the idea would be to implement something like this: myFunction(const rectangle r) { // ... } as follows in Java: class RectangleAccessor { private final Rectangle r; RectangleAccessor(Rectangle r) { this.r = r; } public int getX() { return r.x; } public int getY() { return r.y; } public int getWidht() { return r.width; } public int getHeight() { return r.height; } } ... myFunction(Rectangle r) { RectangleAccessor ra = new RectangleAccessor(r); r = null; // so we can't accidentally modify // Now only access via 'ra' // ... } In reality, this isn't idiomatic Java. But you might occasionally want to do it if it really helps you spot a bug. The other situation that can occur is when you aren't sure whether a particular library method will alter a given object that we pass in. In its C++ cousin, if the parameter in question had been declared const, we'd take this as a hint that it the function in question wouldn't modify it. In Java, one form of defensive action is usually a pass in a copy of the object in question if it's something trivial such as a Rectangle: Rectangle r = ... callMethodThatMayAlterIt(new Rectangle(r)); (If the object doesn't provide an explicit constructor to make the copy, then many standard JDK objects are cloneable.) You might think this is inefficient. And it is, just ever so slightly. But on a modern VM it's really not so bad either: the overhead is probably a few machine instructions more than passing by value in C/C++ (where in any case, the whole object would be copied on to the stack). What to read next You may be interested in the Java equivalents of other C/C++ functions and keywords, such as the Java equivalent of pointers or the Java equivalent of malloc. If you enjoy this Java programming article, please share with friends and colleagues. Follow the author on Twitter for the latest news and rants. Follow @BitterCoffey Editorial page content written by Neil Coffey. Copyright Javamex UK 2021. All rights reserved. A constant is a variable whose value cannot be changed once it has been initialized. Java doesn't have built-in support for constants. To define a variable as a constant, we just need to add the keyword "final" in front of the variable declaration. It is not mandatory that we assign values to constants during declaration. final float pi = 3.14f; The above statement declares the float variable "pi" as a constant with a value of 3.14f. We cannot change the value of "pi" at any point in time in the program. Later, if we try to do that by using a statement like "pi=5.25f", Java will throw errors at compile time itself. Example: Defining Constants In the below example, we define the primitive data type (byte, int, double, boolean, and char) variables as constants by just adding the keyword "final" when we declare the variable. public class ConstantsDemo { public static void main(String args[]) { final byte var1 = 2; final byte var2; var2 = -3; final int var3 = 100; final int var4; var4 = -112; final double var5 = 20000.3223; final double var6; var6 = -11223.222; final boolean var7 = true; final boolean var8; var8 = false; final char var9 = 'e'; final char var10; var10 = 't'; // Displaying values of all variables System.out.println("value of var1 : " + var1); System.out.println("value of var2 : " + var2); System.out.println("value of var3 : " + var3); System.out.println("value of var4 : " + var4); System.out.println("value of var5 : " + var5); System.out.println("value of var6 : " + var6); System.out.println("value of var7 : " + var7); System.out.println("value of var8 : " + var8); System.out.println("value of var9 : " + var9); System.out.println("value of var10 : " + var10); } } Output: value of var1 : 2value of var2 : -3value of var3 : 100value of var4 : -112value of var5 : 20000.3223value of var6 : -11223.222value of var7 : truevalue of var8 : falsevalue of var9 : evalue of var10 : tThe final keywordThe final keyword is a non-access modifier and is used with variables, methods, and classes so that they cannot be changed. Methods that are defined as final cannot be overridden, and classes that are defined as final cannot be inherited. And, the value of final variables, as discussed above, cannot be changed as they become constant.

Is main a keyword in java. What is const keyword in javascript. What is const keyword. Is there a const keyword in java.