Java Language Keywords

* You cannot use any of the following as identifiers in your programs.
* The keywords const and goto are reserved, even though they are not currently used.
* true, false, and null might seem like keywords, but they are actually literals; you cannot use them as identifiers in your programs.

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

The following table shows the keywords grouped by category:

|  |  |
| --- | --- |
| **Category** | **Keywords** |
| Access specifiers | private, protected, public |
| Class, method, variable modifiers | abstract, class, extends, final, implements, interface, native, new, static, strictfp, synchronized, transient, volatile |
| Flow control | break, case, continue, default, do, else, for, if, instanceof, return, switch, while |
| Package control | import, package |
| Primitive types | boolean, byte, char, double, float, int, long, short |
| Error handling | assert, catch, finally, throw, throws, try |
| Enumeration | enum |
| Others | super, this, void |
| Unused | const, goto |

**Private:**

* If a method marked as private, it cannot be invoked from outside of the class it is declared. In other words, the private method is available to the enclosing class.
* Like private method, a private variable can be accessed only from within its enclosing class.
* An inner class can be declared as private thus it can be accessed only from within the enclosing class, like private method and private variable.

**Protected:**

When a method or a variable is marked as protected, it can be accessed from:

* Within the enclosing class.
* Other classes in the same package as the enclosing class.
* Sub classes, regardless of packages.

**Public:**

* The public keyword is an access modifier for class, method and variable:
* When a class is marked as public, it can be accessed from anywhere, including outside packages.
* When a method is marked as public, it can be invoked not only from the enclosing class, but also from outside classes.
* When a variable is marked as public, it can be accessed and updated from outside classes.

**Final:**

* Final class: if a class is marked as final, it cannot be subclassed/inherited by another class.
* Final method: when a method is final, that means it cannot be overriden, neither by methods in the same class or in sub class.
* Final variable: if a variable is marked as final, its reference cannot be changed to refer to another object, once initialized.
* Final parameter: If you declare any parameter as final, you cannot change the value of it inside the method.

Note**:**

* a class cannot be both abstract and final.
* constructor can never be final because it is never inherited.

**Blank or uninitialized final variable:**

* A final variable that is not initialized at the time of declaration is known as blank final variable.
* If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed, it is useful.
* It can be initialized only in constructor without which it will throw error.
* Similarly static blank final variable can be initialized in a static block alone and without which it will throw an error.

**Native:**

* The native keyword is used to declare a method which is implemented in platform-dependent code such as C or C++.
* When a method is marked as native, it cannot have a body and must ends with a semicolon instead.
* The Java Native Interface (JNI) specification governs rules and guidelines for implementing native methods, such as data type conversion between Java and the native application.

**Example:**

public class NativeExample {

public native void fastCopyFile(String sourceFile, String destFile);

}

**New:**

* In Java, all arrays and objects are allocated on the heap. The only things that are ever allocated on the stack in Java are object references and primitives.
* The new keyword is used to create a new instance (object) of a class. So always reserves heap memory

**Static:**

* In java is used for memory management mainly.
* Static methods and variables are shared among all instances of a class.
* They can be invoked and accessed without creating new instances of the class.
* A regular class cannot be static, only inner class can be static.

The static can be:

* variable (also known as class variable) - gets memory only once in class area at the time of class loading.
* method (also known as class method)
* block
* nested class

**Restrictions for static method:**

* The static method can not use non static data member or call non-static method directly.
* this and super cannot be used in static context.

**Java static block**

* Is used to initialize the static data member.
* It is executed before main method at the time of class loading.

**Can we execute a program without main() method?**

Yes, one of the way is static block but in previous version of JDK not in JDK 1.7.

**Static Import:**

Using static import we can invoke static methods by using method name only and not specify class name.

Eg:

import static java.lang.Math.\*;

and can use

System.out.println("Square root 25: " + sqrt(25));

**Strictfp:**

* The floating point precision depends on target platform’s hardware, i.e. CPU’s floating point processing capability.
* Strictfp ensures that you get exactly the same results from your floating point calculations on every platform.

**Rules**

* strictfp cannot be applied for constructors.
* If an interface or class is declared with strictfp, then all methods and nested types within that interface or class are implicitly strictfp.
* Strictfp cannot be applied for interface methods.

**Volatile:**

* The volatile keyword can be applied for only member variables (fields).
* When a volatile variable is accessed concurrently by threads, its value is updated consistently among threads.

Eg: With non-volatile variables there are no guarantees about when the Java Virtual Machine (JVM) reads data from main memory into CPU caches, or writes data from CPU caches to main memory.

By declaring the shared variable volatile the JVM guarantees that every read of the variable will always be read from main memory, and that all writes to the variable will always be written back to main memory.

**Rules**

* The volatile keyword cannot be applied for class, method and local variable.
* A final variable cannot be declared volatile.

## Performance Considerations of volatile

Reading and writing of volatile variables causes the variable to be read or written to main memory. Reading from and writing to main memory is more expensive than accessing the CPU cache. Accessing volatile variables also prevent instruction reordering which is a normal performance enhancement technique. Thus, you should only use volatile variables when you really need to enforce visibility of variables.

**New Java 5 functionality for volatile variables:**

* truly atomic get-and-set operations are permitted.
* an efficient means of accessing the nth element of a volatile array (and performing atomic get-and-set on the element) is provided.

**Break:**

* The break statement has two forms: labeled and unlabeled.
* use an unlabeled break to terminate a for, while, or do-while loop
* The execution goes to next statement after the construct is broken.
* An unlabeled break statement terminates the innermost switch, for, while, or do-while statement, but a labeled break terminates an outer statement.
* The break statement terminates the labeled statement; it does not transfer the flow of control to the label. Control flow is transferred to the statement immediately following the labeled (terminated) statement.

**Continue:**

* The continue statement skips the current iteration of a for, while, or do-while loop.
* The unlabeled form skips to the end of the innermost loop's body and evaluates the Boolean expression that controls the loop.
* A labeled continue statement skips the current iteration of an outer loop marked with the given label.

**Illegal Way of Writing Continue :**

* Continue as a Part of Normal Java Statement
* Don’t Use Continue Inside If unless it is part of any Loop.

**Return:**

The return statement exits from the current method, and control flow returns to where the method was invoked.

**The return statement has two forms:**

* one that returns a value
  + used to stop execution of a method and return a value for the caller.
* one that doesn't
  + used to stop execution of a void method and return to the caller

**Default:**

* If there is no access specifier for a class or method then it is consider as default specifier
* The default keyword can optionally be used in a switch statement to label a block of statements to be executed if no case matches the specified value
* the default keyword can also be used to declare default values in a [Java annotation](https://en.wikipedia.org/wiki/Java_annotation)
* From Java 8 onwards, the default keyword is also used to specify that a method in an interface provides the default implementation of an optional method.(Evolving Interface or Defender Methods)

**Default methods have two important use cases:**

* Evolving existing interfaces: Stream<E> stream() and default Stream<E> parallelStream() methods of java.util.Collection interface
* Increasing design flexibility: Default methods offer greater flexibility because you can implement an interface at any point in the class hierarchy and access the interface's default methods from the implementing classes which is similar to abstract classes. Because when using abstract classes we can extend only one class.

Instanceof:

* The java instanceof operator is used to test whether the object is an instance of a specified class or its super class.
* The instanceof in java is also known as type comparison operator because it compares the instance with type.
* If we apply the instanceof operator with any variable that has null value, it returns false

## Downcasting with java instanceof operator:

Downcasting can also be performed without the use of instanceof operator. But without which there is a possibility of getting ClassCastException.

Eg: static void method(Animal a) {

if(a instanceof Dog3){

Dog3 d=(Dog3)a;//downcasting

}

}

**Import:**

* It declares a Java class to use in the code below the import statement.
* Once a Java class is declared, then the class name can be used in the code without specifying the package the class belongs to.
* The static import construct can be used to invoke static methods by using method name only and need not specify class name.

#### Note: If you import a package, subpackages will not be imported.

**Package:**

* It declares a 'name space' for the Java class.
* It must be put at the top of the Java file, it should be the first Java statement line.
* To ensure that the package name will be unique across vendors, usually the company url is used starting in backward.
* There can be only one package statement in each source file, and it applies to all types in the file.
* If a package statement is not used then the class, interfaces, enumerations, and annotation types will be put into an unnamed package.

**Assert:**

* Assertion enables developers to test assumptions in their programs as a way to defect and fix bugs.
* An assert statement is used to declare an expected Boolean condition in a program.
* If the program is running with assertions enabled, then the condition is checked at run-time.
* If the condition is false, the Java run-time system throws a [AssertionError](https://en.wikibooks.org/w/index.php?title=Java_Programming/API/java.lang.AssertionError&action=edit&redlink=1).

Eg: assert number <=10;

assert value>=18:"Not valid";

By default, assertion is disabled at runtime

To enable assertion, specify the switch –enableassertions or -ea at command line of java program.

**Super:**

The super keyword in java is a reference variable that is used to refer immediate parent class object.

## Usage of java super Keyword

* super is used to refer immediate parent class instance variable.
* super() is used to invoke immediate parent class constructor.
* super is used to invoke immediate parent class method.

#### Note: super() is added in each class constructor automatically by compiler.

**This:**

* this keyword can be used to refer current class instance variable.
* this() can be used to invoke current class constructor.
* this keyword can be used to invoke current class method (implicitly)
* this can be passed as an argument in the method call.
* this can be passed as argument in the constructor call.
* this keyword can also be used to return the current class instance.

#### Note: Call to this() must be the first statement in constructor.