JAVA KEYWORDS CHEAT SHEET

Bitelligence
Programming Training & Continuing Education.

PACKAGES AND IMPORTING

Organize code into namespaces and allow it to access.

`package`	Used to declare the package to which a class belongs.
`import`	Used to import classes, interfaces, and other packages into a program.

DECLARATIONS

Define the names, types and accessibility of elements.

`enum`	Declares a new enumerated type, which is a special type of class used to represent a fixed number of constant values.
`record`	Compact and immutable way to declare classes for storing data, similar to a struct in C. It automatically generates the getters, toString, equals and hashCode methods.
`class`	Declares a new class, which can contain fields, constructors, and methods.
`interface`	Declares a new interface, which defines a contract that classes can implement.
`extends`	Is used to create a subclass that inherits properties and methods from a parent class or interface.
`implements`	Indicates that a class is implementing one or more interfaces, providing method implementations for each.

ACCESS MODIFIERS

Keyword that specifies the level of access or visibility for classes, methods, and variables.

`public`	Indicates that a class, method, or variable is accessible to all other classes in the same package or in other packages.
`protected`	Indicates that a class member is accessible within the same package or in a subclass of the class in which it is defined.
`private`	Indicates that a class member (field, method, or inner class) is accessible only within the same class.

PRIMITIVE TYPES

Fundamental data types, representing simple values like integers, floating-point numbers, characters or booleans.

`byte`	8-bit signed integer -2^7 to 2^7-1.
`short`	16-bit signed integer -2^{15} to $2^{15}-1$.
`int`	32-bit signed integer -2^{31} to $2^{31}-1$.
`long`	64-bit signed integer -2^{63} to $2^{63}-1$.
`float`	32-bit floating point number.
`double`	64-bit floating point number.
`char`	16-bit Unicode character.
`boolean`	true or false value.

FLOW CONTROL

Language constructs that enables the control of program flow, like branching, looping and conditional statements.

`if`	Test a condition and execute a block of code if the condition is true.
`else`	Executes a block of code if the condition in the if statement is false.
`for`	Executing a block of code repeatedly for a fixed amount or until a certain condition is met.
`do`	Executes a block of code repeatedly until a certain condition is met, runs at least once.
`while`	Executing a block of code repeatedly as long as a condition is true.
`switch`	Selecting one of several code blocks to be executed, depending on the value of an expression.
`case`	Specify one of the code blocks to be executed, depending on the value of the expression.
`default`	Gets executed if none of the case statements match the expression's value.
`yield`	Returns a value from a switch expression branch, without requiring a break statement.

JAVA KEYWORDS CHEAT SHEET

Bitelligence
Programming Training & Continuing Education.

OBJECT RELATED

Fundamental concepts of object-oriented programming, such as object creation, object reference and type checking.

`new`	Creates an instance of a class.
`super`	Refers to the parent class of a subclass.
`this`	Refers to the current object.
`instanceof`	Checks if an object is an instance of a specific class or interface.

LITERALS

Values assigned to variables or used in expressions, such as numbers, strings and boolean values.

`true`	Represents a boolean value that is used to indicate a logical true condition.
`false`	Represents the boolean value that indicates a logical false condition.
`null`	Indicates that a reference variable does not refer to any object. It is a special value that represents the absence of a value.

JUMP STATEMENTS

Allow to transfer control to another part of the program, such as a loop or method, based on a certain condition.

`continue`	Skips the rest of the loop and starts the next iteration.
`break`	Exits the loop or switch statement.
`return`	Exits the current method and returns a value.

THREADS

Allow for concurrent execution of multiple parts of a program and related concepts include synchronization, locks, race conditions and deadlocks.

`synchronized`	Controls access to shared resources by allowing only one thread to execute the synchronized code block at a time.
`volatile`	Indicates that a variable's value may be modified by multiple threads and should not be cached by a single thread. Accesses to a volatile variable are guaranteed to be atomic.

MODIFIERS

Used to modify the properties or behavior of classes, variables, methods and other program entities.

`abstract`	Indicates that a class or method does not have an implementation and must be subclassed or overridden.
`native`	Refers to a method whose implementation is written in a language other than Java, such as C or C++.
`final`	Used to make a variable's value unchangeable, a method un-overridable, or a class un-extendable.
`static`	Refers to a variable or method that belongs to the class itself, rather than to any instance of the class.
`transient`	Indicates that a variable should not be serialized when the object is written to a stream.
`void`	Used as the return type of a method that does not return any value.
`sealed`	Restricts the set of classes that can extend or implement a superclass or interface.
`non-sealed`	Default Java access modifier. No restrictions on the set of classes that can extend or implement a superclass or interface.
`permits`	Explicitly specify which classes or interfaces are allowed to implement the sealed class or interface.
`default`	Provide a default implementation for a method in an interface, allowing interface evolution and backward compatibility.

DYNAMIC TYPING

Words related to the declaration and initialization of variables, enabling flexible and efficient coding with greater readability and ease of maintenance.

`var`	Is used for type inference, allowing the compiler to determine the type of a variable based on its value.
`_`	Variable name or parameter placeholder for ignored values, i.e., values that are not used in the code.

JAVA KEYWORDS CHEAT SHEET

Bitelligence
Programming Training & Continuing Education.

EXCEPTION HANDLING

Handle errors and other exceptional conditions during program execution. It uses try-catch-finally blocks to gracefully recover from errors and avoid program termination.

`try`	Syntax structure that enables handling of exceptions and trying out potentially error-prone code.
`catch`	Used for handling exceptions that may occur within a try block by executing specific code for each exception type.
`finally`	Code block that executes regardless of errors or exceptions, for cleanup or releasing resources.
`throw`	A statement used to explicitly raise an exception or error within a program, with a specific message or type.
`throws`	Specifies the exceptions a method might throw, enabling callers to handle them or pass them up the call stack.
`assert`	Check assumptions in code, throwing an error if the condition fails, aiding in debugging and testing.

UNUSED & DEPRECATED

Currently unused as they were deemed unnecessary or problematic. One for declaring constants, one for jumping to labeled statements and one for enforcing floating-point arithmetic standards.

`const`	Was originally reserved in Java as a possible future addition to the language. Proposed for Java constants, but not used due to compatibility issues.
`goto`	Present in early versions of the Java language specification as a reserved keyword, but it was never actually implemented as a feature of the language due to readability, maintainability concerns and potential security risks.
`strictfp`	Enforces precise floating-point calculations, but is rarely used due to its potential impact on performance and limited usefulness in modern computing. Additionally, the keyword has been marked as deprecated since Java 16.

ANNOTATIONS

Keywords specifically designed for annotations, which are used to provide metadata and other annotations-related functionality to code.

`@interface`	Allows programmers to define their own custom annotations in Java for use in code documentation and runtime processing.
`default`	In the context of annotations, default is used to specify a default value for an annotation element.

MODULE RELATED

Java module system keywords are used to declare module dependencies, encapsulate code, and control module access, promoting modularity and maintainability.

`module`	Defines a module, a self-contained unit of code that explicitly declares its dependencies on other modules.
`exports`	A Java feature that limits access to module classes and methods, promoting modularity and secure encapsulation.
`open`	A feature that allows selected code entities to be accessed from other modules, promoting controlled visibility and modularity.
`opens`	Permits controlled access to internal classes and members of a module, promoting encapsulation and secure modularity.
`provides`	Used to declare a service provider and its implementation, allowing service clients to dynamically discover and use the provider.
`requires`	Declare a module dependency on another module, ensuring proper resolution and loading of required modules at runtime.
`to`	Used in module declarations to specify which modules can access a package or a class, promoting controlled visibility and modularity.
`uses`	Declare a service interface used by a module, allowing the Java platform to dynamically locate and link a service implementation at runtime.
`permits`	Allows specified modules to access a package or a class, promoting controlled visibility and modularity.