**Section 1:**

**A Java project in Eclipse and a Maven project? What are the difference**

Ans: The primary differences between lie in how they manage dependencies, handle the build process, and organize the project structure:

The key difference between a **Java Project** and a **Maven Project** lies in how they are structured, built, and managed:

### **Java Project**

* A **Java Project** is a basic structure where you manually manage dependencies, compilation, and builds.
* It typically consists of:
  + .java files (source code)
  + .class files (compiled bytecode)
  + A lib folder for manually adding external JAR files
  + No predefined directory structure (you create your own)
* Compilation and execution are done manually using **javac** and **java** commands or through an IDE like Eclipse/IntelliJ.
* Dependency management is **manual**, requiring JARs to be downloaded and added to the classpath.

### **Maven Project**

* A **Maven Project** is a structured Java project that uses **Apache Maven**, a build automation tool.
* It provides:
  + **Convention over configuration** (follows a predefined structure)
  + **Automated dependency management** (using pom.xml)
  + **Build lifecycle management** (mvn clean install, mvn package, etc.)
* Structure follows a standard:
* my-maven-project/
* ├── src/
* │ ├── main/java/ (Source Code)
* │ ├── main/resources/ (Configuration files)
* │ ├── test/java/ (Test Cases)
* ├── pom.xml (Maven Configuration)
* ├── target/ (Compiled files & JARs)
* Dependencies are managed automatically using **Maven Central Repository** (pom.xml).
* Common Maven commands:
  + mvn compile – Compiles the source code
  + mvn package – Builds the JAR/WAR
  + mvn install – Installs the package in the local repository

### **Summary**

| **Feature** | **Java Project** | **Maven Project** |
| --- | --- | --- |
| Build Management | Manual | Automated (Maven Lifecycle) |
| Dependency Management | Manual (JARs added manually) | Automated (pom.xml) |
| Project Structure | Custom | Standard (src/main/java, src/test/java, etc.) |
| Compilation & Execution | Manual (javac, java) | mvn compile, mvn package |
| Reproducibility | Less structured | Easily reproducible across different systems |

### **When to Use What?**

* **Use a Java Project** if you're working on a small, simple program that doesn't require dependency management.
* **Use a Maven Project** for **medium to large applications** where dependency management, structured builds, and automation are needed.

**Difference between Maven and Ant**

* Difference between Maven and Ant

1. Maven : Maven is a powerful project management tool based on the Project Object Model. It helps in managing project builds, documentation, dependency, releases, etc.

2. Ant : Ant is a command-line toolbox without any coding conventions or project structures, making it flexible and more manageable to use. It is most commonly used to build Java applications.

**Keywords and variables:**

1. keywords are reserved word

2. Meaning of word, is known to complier

3. Keywords cannot be used as name of variable

## List of Java keywords

[[edit](https://en.wikipedia.org/w/index.php?title=List_of_Java_keywords&action=edit&section=1)]

**\_**

Added in Java 9, the underscore has become a keyword and cannot be used as a variable name anymore.[[3]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-3)

[**abstract**](https://en.wikipedia.org/wiki/Abstract_type)

A method with no definition must be declared as abstract and the class containing it must be declared as abstract. Abstract classes cannot be instantiated. Abstract methods must be implemented in the sub classes. The abstract keyword cannot be used with variables or constructors. Note that an abstract class isn't required to have an abstract method at all.

[**assert**](https://en.wikipedia.org/wiki/Assertion_(software_development))**(added in**[**J2SE 1.4**](https://en.wikipedia.org/wiki/J2SE_1.4)**)**[[4]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-keywords_date-4)

Assert describes a predicate (a true–false statement) placed in a Java program to indicate that the developer thinks that the predicate is always true at that place. If an assertion evaluates to false at run-time, an assertion failure results, which typically causes execution to abort. Assertions are disabled at runtime by default, but can be enabled through a command-line option or programmatically through a method on the class loader.

[**boolean**](https://en.wikipedia.org/wiki/Boolean_data_type)

Defines a boolean variable for the values "true" or "false" only. By default, the value of boolean primitive type is false. This keyword is also used to declare that a method returns a value of the primitive type [boolean](https://en.wikipedia.org/wiki/Boolean_data_type" \o "Boolean data type).

[**break**](https://en.wikipedia.org/wiki/Break_statement)

Used to end the execution in the current loop body.

Used to break out of a [switch](https://en.wikipedia.org/wiki/List_of_Java_keywords#switch) block.

[**byte**](https://en.wikipedia.org/wiki/Byte)

The byte keyword is used to declare a field that can hold an 8-bit signed two's complement integer.[[5]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-primitive-5)[[6]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-FOOTNOTEFlanagan200522-6) This keyword is also used to declare that a method returns a value of the primitive type byte.[[7]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-return-7)[[8]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-FOOTNOTEFlanagan200566%E2%80%9367-8)

[**case**](https://en.wikipedia.org/wiki/Switch_statement)

A statement in the [switch](https://en.wikipedia.org/wiki/List_of_Java_keywords#switch) block can be labeled with one or more [case](https://en.wikipedia.org/wiki/List_of_Java_keywords#case) or [default](https://en.wikipedia.org/wiki/List_of_Java_keywords#default) labels. The [switch](https://en.wikipedia.org/wiki/List_of_Java_keywords#switch) statement evaluates its expression, then executes all statements that follow the matching [case](https://en.wikipedia.org/wiki/List_of_Java_keywords#case) label; see [switch](https://en.wikipedia.org/wiki/List_of_Java_keywords#switch).[[9]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-switch-9)[[10]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-FOOTNOTEFlanagan200546%E2%80%9348-10)

[**catch**](https://en.wikipedia.org/wiki/Exception_handling_syntax#Java)

Used in conjunction with a try block and an optional finally block. The statements in the catch block specify what to do if a specific type of exception is thrown by the try block.

[**char**](https://en.wikipedia.org/wiki/Character_(computing))

Defines a character variable capable of holding any character of the java source file's character set.

[**class**](https://en.wikipedia.org/wiki/Class_(computer_science)#Java)

A type that defines the implementation of a particular kind of object. A class definition defines [instance](https://en.wikipedia.org/wiki/Object_(computer_science)) and class [fields](https://en.wikipedia.org/wiki/Field_(computer_science)), [methods](https://en.wikipedia.org/wiki/Method_(computer_science)), and [inner classes](https://en.wikipedia.org/wiki/Inner_class) as well as specifying the [interfaces](https://en.wikipedia.org/wiki/Interface_(computer_science)) the class implements and the immediate [superclass](https://en.wikipedia.org/wiki/Superclass_(computer_science)) of the class. If the superclass is not explicitly specified, the superclass is implicitly [Object](https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/lang/Object.html). The class keyword can also be used in the form Class**.class** to get a Class object without needing an instance of that class. For example, **String.class** can be used instead of doing **new String().getClass()**.

[**continue**](https://en.wikipedia.org/wiki/Continue_(Java))

Used to resume program execution at the end of the current loop body. If followed by a label, continue resumes execution at the end of the enclosing labeled loop body.

**default**

The default keyword can optionally be used in a [switch statement](https://en.wikipedia.org/wiki/Switch_statement) to label a block of statements to be executed if no case matches the specified value; see [*switch*](https://en.wikipedia.org/wiki/List_of_Java_keywords#switch).[[9]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-switch-9)[[10]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-FOOTNOTEFlanagan200546%E2%80%9348-10) Alternatively, 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 can be used to allow an interface to provide an implementation of a method.

[**do**](https://en.wikipedia.org/wiki/Do_while_loop)

The do keyword is used in conjunction with [while](https://en.wikipedia.org/wiki/List_of_Java_keywords#while) to create a [do-while loop](https://en.wikipedia.org/wiki/Do-while_loop), which executes a block of statements associated with the loop and then tests a boolean expression associated with the while. If the expression evaluates to true, the block is executed again; this continues until the expression evaluates to false.[[11]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-do-while-11)[[12]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-FOOTNOTEFlanagan200548%E2%80%9349-12)

[**double**](https://en.wikipedia.org/wiki/Double_precision)

The double keyword is used to declare a variable that can hold a 64-bit [double precision](https://en.wikipedia.org/wiki/Double_precision) [IEEE 754](https://en.wikipedia.org/wiki/IEEE_754) [floating-point number](https://en.wikipedia.org/wiki/Floating-point_number).[[5]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-primitive-5)[[6]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-FOOTNOTEFlanagan200522-6) This keyword is also used to declare that a method returns a value of the primitive type double.[[7]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-return-7)[[8]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-FOOTNOTEFlanagan200566%E2%80%9367-8)

[**else**](https://en.wikipedia.org/wiki/Conditional_(programming))

The else keyword is used in conjunction with [if](https://en.wikipedia.org/wiki/List_of_Java_keywords#if) to create an [if-else statement](https://en.wikipedia.org/wiki/Conditional_(programming)), which tests a [boolean expression](https://en.wikipedia.org/wiki/Boolean_expression" \o "Boolean expression); if the expression evaluates to true, the block of statements associated with the if are evaluated; if it evaluates to false, the block of statements associated with the else are evaluated.[[13]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-if-else-13)[[14]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-FOOTNOTEFlanagan200544%E2%80%9346-14)

[**enum**](https://en.wikipedia.org/wiki/Enumerated_type)**(added in**[**J2SE 5.0**](https://en.wikipedia.org/wiki/J2SE_5.0)**)**[[4]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-keywords_date-4)

A Java keyword used to declare an [enumerated type](https://en.wikipedia.org/wiki/Enumerated_type). Enumerations extend the base class [Enum](https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/lang/Enum.html).

[**extends**](https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming))

Used in a class declaration to specify the superclass; used in an interface declaration to specify one or more superinterfaces. Class X extends class Y to add functionality, either by adding fields or methods to class Y, or by overriding methods of class Y. An interface Z extends one or more interfaces by adding methods. Class X is said to be a subclass of class Y; Interface Z is said to be a subinterface of the interfaces it extends.

Also used to specify an upper bound on a type parameter in Generics.

[**final**](https://en.wikipedia.org/wiki/Final_(Java))

Define an entity once that cannot be changed nor derived from later. More specifically: a final class cannot be subclassed, a final method cannot be overridden, and a final variable can occur at most once as a left-hand expression on an executed command. All methods in a final class are implicitly final.

[**finally**](https://en.wikipedia.org/wiki/Exception_handling_syntax#Java)

Used to define a block of statements for a block defined previously by the try keyword. The finally block is executed after execution exits the try block and any associated catch clauses regardless of whether an exception was thrown or caught, or execution left method in the middle of the try or catch blocks using the return keyword.

[**float**](https://en.wikipedia.org/wiki/Single_precision)

The float keyword is used to declare a variable that can hold a 32-bit [single precision](https://en.wikipedia.org/wiki/Single_precision) IEEE 754 floating-point number.[[5]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-primitive-5)[[6]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-FOOTNOTEFlanagan200522-6) This keyword is also used to declare that a method returns a value of the primitive type float.[[7]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-return-7)[[8]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-FOOTNOTEFlanagan200566%E2%80%9367-8)

[**for**](https://en.wikipedia.org/wiki/For_loop)

The for keyword is used to create a [for loop](https://en.wikipedia.org/wiki/For_loop), which specifies a variable initialization, a [boolean expression](https://en.wikipedia.org/wiki/Boolean_expression" \o "Boolean expression), and an incrementation. The variable initialization is performed first, and then the boolean expression is evaluated. If the expression evaluates to true, the block of statements associated with the loop are executed, and then the incrementation is performed. The boolean expression is then evaluated again; this continues until the expression evaluates to false.[[15]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-for-15)

As of [J2SE 5.0](https://en.wikipedia.org/wiki/J2SE_5.0), the for keyword can also be used to create a so-called "[enhanced for loop](https://en.wikipedia.org/wiki/Foreach)",[[16]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-FOOTNOTEFlanagan200550%E2%80%9354-16) which specifies an [array](https://en.wikipedia.org/wiki/Array_data_type) or [Iterable](https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/lang/Iterable.html) object; each iteration of the loop executes the associated block of statements using a different element in the array or Iterable.[[15]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-for-15)

[**if**](https://en.wikipedia.org/wiki/If_statement)

The if keyword is used to create an [if statement](https://en.wikipedia.org/wiki/If_statement), which tests a [boolean expression](https://en.wikipedia.org/wiki/Boolean_expression" \o "Boolean expression); if the expression evaluates to true, the block of statements associated with the if statement is executed. This keyword can also be used to create an [if-else statement](https://en.wikipedia.org/wiki/Conditional_(programming)); see [*else*](https://en.wikipedia.org/wiki/List_of_Java_keywords#else).[[13]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-if-else-13)[[14]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-FOOTNOTEFlanagan200544%E2%80%9346-14)

**implements**

Included in a class declaration to specify one or more [interfaces](https://en.wikipedia.org/wiki/Interface_(Java)) that are implemented by the current class. A class inherits the types and abstract methods declared by the interfaces.

**import**

Used at the beginning of a [source file](https://en.wikipedia.org/wiki/Source_file) to specify classes or entire [Java packages](https://en.wikipedia.org/wiki/Java_package) to be referred to later without including their package names in the reference. Since J2SE 5.0, import statements can import static members of a class.

**instanceof**

A [binary operator](https://en.wikipedia.org/wiki/Operator_(programming)) that takes an object reference as its first operand and a class or interface as its second operand and produces a boolean result. The instanceof operator evaluates to true if and only if the runtime type of the object is assignment compatible with the class or interface.

[**int**](https://en.wikipedia.org/wiki/Integer_(computer_science))

The int keyword is used to declare a variable that can hold a 32-bit signed two's complement integer.[[5]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-primitive-5)[[6]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-FOOTNOTEFlanagan200522-6) This keyword is also used to declare that a method returns a value of the primitive type int.[[7]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-return-7)[[8]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-FOOTNOTEFlanagan200566%E2%80%9367-8)

[**interface**](https://en.wikipedia.org/wiki/Interface_(Java))

Used to declare an [interface](https://en.wikipedia.org/wiki/Interface_(Java)) that only contains abstract or default methods, constant (static final) fields and static interfaces. It can later be implemented by classes that declare the interface with the implements keyword. As [multiple inheritance](https://en.wikipedia.org/wiki/Multiple_inheritance) is not allowed in Java, interfaces are used to circumvent it. An interface can be defined within another interface.

[**long**](https://en.wikipedia.org/wiki/Long_integer)

The long keyword is used to declare a variable that can hold a 64-bit signed two's complement integer.[[5]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-primitive-5)[[6]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-FOOTNOTEFlanagan200522-6) This keyword is also used to declare that a method returns a value of the primitive type long.[[7]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-return-7)[[8]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-FOOTNOTEFlanagan200566%E2%80%9367-8)

[**native**](https://en.wikipedia.org/wiki/Java_Native_Interface)

Used in method declarations to specify that the method is not implemented in the same Java source file, but rather in another language.[[8]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-FOOTNOTEFlanagan200566%E2%80%9367-8)

[**new**](https://en.wikipedia.org/wiki/Object_lifetime#Java)

Used to create an instance of a class or array object. Using keyword for this end is not completely necessary (as exemplified by [Scala](https://en.wikipedia.org/wiki/Scala_(programming_language))), though it serves two purposes: it enables the existence of different namespace for methods and class names, it defines statically and locally that a fresh object is indeed created, and of what runtime type it is (arguably introducing dependency into the code).

[**package**](https://en.wikipedia.org/wiki/Java_package)

Java package is a group of similar classes and interfaces. Packages are declared with the package keyword.

[**private**](https://en.wikibooks.org/wiki/Java_Programming/Classes,_Objects_and_Types)

The private keyword is used in the declaration of a method, field, or inner class; private members can only be accessed by other members of their own class.[[17]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-access-17)

[**protected**](https://en.wikibooks.org/wiki/Java_Programming/Classes,_Objects_and_Types)

The protected keyword is used in the declaration of a method, field, or inner class; protected members can only be accessed by members of their own class, that class's [subclasses](https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)) or classes from the same [package](https://en.wikipedia.org/wiki/Java_package).[[17]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-access-17)

[**public**](https://en.wikibooks.org/wiki/Java_Programming/Classes,_Objects_and_Types)

The public keyword is used in the declaration of a class, method, or field; public classes, methods, and fields can be accessed by the members of any class.[[17]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-access-17)

[**return**](https://en.wikipedia.org/wiki/Method_(computer_science))

Used to finish the execution of a method. It can be followed by a value required by the method definition that is returned to the caller.

[**short**](https://en.wikipedia.org/wiki/Short_integer)

The short keyword is used to declare a field that can hold a 16-bit signed two's complement integer.[[5]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-primitive-5)[[6]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-FOOTNOTEFlanagan200522-6) This keyword is also used to declare that a method returns a value of the primitive type short.[[7]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-return-7)[[8]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-FOOTNOTEFlanagan200566%E2%80%9367-8)

[**static**](https://en.wikipedia.org/wiki/Static_variable)

Used to declare a field, method, or inner class as a class field. Classes maintain one copy of class fields regardless of how many instances exist of that class. static also is used to define a method as a class method. Class methods are [bound](https://en.wikipedia.org/wiki/Name_binding) to the class instead of to a specific instance, and can only operate on class fields. Classes and interfaces declared as static members of another class or interface are behaviorally top-level classes.[[18]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-18)

[**super**](https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming))

Inheritance basically used to achieve dynamic binding or run-time polymorphism in java. Used to access members of a class inherited by the class in which it appears. Allows a subclass to access [overridden](https://en.wikipedia.org/wiki/Method_overriding_(programming)) methods and hidden members of its superclass. The super keyword is also used to forward a call from a constructor to a constructor in the superclass.

Also used to specify a lower bound on a type parameter in Generics.

[**switch**](https://en.wikipedia.org/wiki/Switch_statement)

The switch keyword is used in conjunction with [case](https://en.wikipedia.org/wiki/List_of_Java_keywords#case) and [default](https://en.wikipedia.org/wiki/List_of_Java_keywords#default) to create a [switch statement](https://en.wikipedia.org/wiki/Switch_statement), which evaluates a variable, matches its value to a specific case (including [patterns](https://en.wikipedia.org/wiki/Pattern_matching)), and executes the block of statements associated with that case. If no case matches the value, the optional block labelled by default is executed, if included.[[9]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-switch-9)[[10]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-FOOTNOTEFlanagan200546%E2%80%9348-10) The switch keyword can also be used with the non-reserved keyword yield to create switch expressions.

[**synchronized**](https://en.wikipedia.org/wiki/Mutual_exclusion)

Used in the declaration of a method or code block to acquire the [mutex](https://en.wikipedia.org/wiki/Mutex" \o "Mutex) lock for an object while the current [thread](https://en.wikipedia.org/wiki/Thread_(computer_science)) executes the code.[[8]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-FOOTNOTEFlanagan200566%E2%80%9367-8) For static methods, the object locked is the class's Class. Guarantees that at most one thread at a time operating on the same object executes that code. The mutex lock is automatically released when execution exits the synchronized code. Fields, classes and interfaces cannot be declared as *synchronized*.

[**this**](https://en.wikipedia.org/wiki/This_(Java))

Used to represent an instance of the class in which it appears. this can be used to access class members and as a reference to the current instance. The this keyword is also used to forward a call from one constructor in a class to another constructor in the same class.

[**throw**](https://en.wikipedia.org/wiki/Exception_handling_syntax#Java)

Causes the declared exception instance to be thrown. This causes execution to continue with the first enclosing exception handler declared by the catch keyword to handle an assignment compatible exception type. If no such exception handler is found in the current method, then the method returns and the process is repeated in the calling method. If no exception handler is found in any method call on the stack, then the exception is passed to the thread's uncaught exception handler.

[**throws**](https://en.wikipedia.org/wiki/Exception_handling_syntax#Java)

Used in method declarations to specify which exceptions are not handled within the method but rather passed to the next higher level of the program. All uncaught exceptions in a method that are not instances of RuntimeException must be declared using the throws keyword.

[**transient**](https://en.wikipedia.org/wiki/Transient_(computer_programming))

Declares that an instance field is not part of the default [serialized](https://en.wikipedia.org/wiki/Serialization) form of an object. When an object is serialized, only the values of its non-transient instance fields are included in the default serial representation. When an object is deserialized, transient fields are initialized only to their default value. If the default form is not used, e.g. when a *serialPersistentFields* table is declared in the class hierarchy, all transient keywords are ignored.[[19]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-serialSpec-19)[[20]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-transient1-20)

[**try**](https://en.wikipedia.org/wiki/Exception_handling_syntax#Java)

Defines a block of statements that have exception handling. If an exception is thrown inside the try block, an optional catch block can handle declared exception types. Also, an optional finally block can be declared that will be executed when execution exits the try block and catch clauses, regardless of whether an exception is thrown or not. A try block must have at least one catch clause or a finally block.

[**void**](https://en.wikipedia.org/wiki/Void_type)

The void keyword is used to declare that a method does not return any value.[[7]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-return-7)

[**volatile**](https://en.wikipedia.org/wiki/Volatile_variable)

Used in field declarations to guarantee visibility of changes to variables across threads. Every read of a volatile variable will be read from main memory, and not from the CPU cache, and that every write to a volatile variable will be written to main memory, and not just to the CPU cache.[[21]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-Java_Volatile_Keyword-21) Methods, classes and interfaces thus cannot be declared *volatile*, nor can local variables or parameters.

[**while**](https://en.wikipedia.org/wiki/Do_while_loop)

The while keyword is used to create a [while loop](https://en.wikipedia.org/wiki/While_loop), which tests a [boolean expression](https://en.wikipedia.org/wiki/Boolean_expression" \o "Boolean expression) and executes the block of statements associated with the loop if the expression evaluates to true; this continues until the expression evaluates to false. This keyword can also be used to create a [do-while loop](https://en.wikipedia.org/wiki/Do-while_loop); see [*do*](https://en.wikipedia.org/wiki/List_of_Java_keywords#do).[[11]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-do-while-11)[[12]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-FOOTNOTEFlanagan200548%E2%80%9349-12)

## Reserved Identifiers

[[edit](https://en.wikipedia.org/w/index.php?title=List_of_Java_keywords&action=edit&section=2)]

The following identifiers are contextual keywords, and are only restricted in some contexts:

**exports**

**module**

**non-sealed**

Used to declare that a class or interface which extends a sealed class can be extended by unknown classes.[[22]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-sealed-22)

**open**

**opens**

**permits**

The permits clause specifies the classes that are permitted to extend a sealed class.[[22]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-sealed-22)

**provides**

**record**

**requires**

**sealed**

A sealed class or interface can only be extended or implemented by classes and interfaces permitted to do so.[[22]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-sealed-22)

**to**

**transitive**

**uses**

**var**

A special identifier that cannot be used as a type name (since Java 10).[[23]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-23)

**when**

used as an additional check for a case statement. [[24]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-24)

**with**

**yield**

Used to set a value for a switch expression, when using labelled statement groups (for example, case L:).[[25]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-25)

## Reserved words for literal values

[[edit](https://en.wikipedia.org/w/index.php?title=List_of_Java_keywords&action=edit&section=3)]

[**true**](https://en.wikipedia.org/wiki/Truth_value)

A boolean literal value.

[**false**](https://en.wikipedia.org/wiki/Truth_value)

A boolean literal value.

[**null**](https://en.wikipedia.org/wiki/Null_pointer)

A reference literal value.

## Unused

[[edit](https://en.wikipedia.org/w/index.php?title=List_of_Java_keywords&action=edit&section=4)]

[**const**](https://en.wikipedia.org/wiki/Constant_(programming))

Although reserved as a keyword in Java, const is not used and has no function.[[2]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-keywords-2)[[26]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-FOOTNOTEFlanagan200520-26) For defining constants in Java, see the final keyword.

[**goto**](https://en.wikipedia.org/wiki/GOTO)

Although reserved as a keyword in Java, goto is not used and has no function.[[2]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-keywords-2)[[26]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-FOOTNOTEFlanagan200520-26)

[**strictfp**](https://en.wikipedia.org/wiki/Strictfp)**(added in**[**J2SE 1.2**](https://en.wikipedia.org/wiki/J2SE_1.2)**)**[[4]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-keywords_date-4)

Although reserved as a keyword in Java, strictfp is obsolete, and no longer has any function.[[27]](https://en.wikipedia.org/wiki/List_of_Java_keywords#cite_note-27) Previously this keyword was used to restrict the precision and rounding of floating point calculations to ensure portability.

Note: true, false and null are not keywords, but they are literals and reserved words that cannot used as identifiers.

Variable: is used initialize and store data.

**Type casting in Java:**

**Implicit casting(widening)**

Byte -> short->int->long->float->double ->

byte = 8bit (integer type)

short = 16bit (integer type)

int = 32bit(integer type)

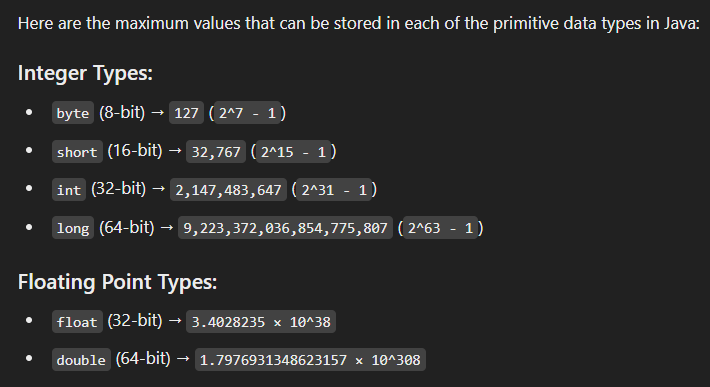
long = 64bit(integer type)

float= 32 bit (floating type)

double = 64 bit(floating type)

**Explicitly casting (Narrowing)**

double -> float-> long -> int -> short -> byte ->



### **Parsing vs. Type Conversion in Java**

Both **parsing** and **type conversion** deal with changing data types, but they are used in different contexts.

### **1. Parsing**

**Parsing** is the process of converting a String into a different data type (e.g., int, double, etc.).

* It is done using methods like Integer.parseInt(), Double.parseDouble(), etc.
* Commonly used when reading input from a file, user input, or APIs where numbers are received as strings.

#### **Example of Parsing:**

String numberStr = "123";

int num = Integer.parseInt(numberStr); // Converts String to int

System.out.println(num); // Output: 123

### **2. Type Conversion (Type Casting)**

**Type conversion** is the process of changing a variable from one primitive type to another. It is mainly used between numeric data types.

#### **Two Types of Type Conversion:**

1. **Implicit (Widening) Conversion** → No data loss
   * Smaller type → Larger type
   * Happens automatically by Java

**Example:**

int num = 10;

double d = num; // int to double (implicit)

System.out.println(d); // Output: 10.0

1. **Explicit (Narrowing) Conversion** → Possible data loss
   * Larger type → Smaller type
   * Requires **explicit** casting

**Example:**

double d = 10.99;

int num = (int) d; // double to int (explicit)

System.out.println(num); // Output: 10

### **Key Differences:**

| **Feature** | **Parsing** | **Type Conversion** |
| --- | --- | --- |
| **Definition** | Converts a String to another data type | Converts one data type to another |
| **Use Case** | Used when reading user input or files | Used when working with numeric data types |
| **Method** | Uses parseXxx() methods (e.g., Integer.parseInt()) | Uses implicit or explicit casting |
| **Example** | "123" → 123 (String to int) | 10.5 → 10 (double to int) |

### **Conclusion**

* **Use parsing when converting a String to a number.**
* **Use type conversion when changing between numeric data types.**

Let me know if you need more examples! 🚀

Here's an example demonstrating proper naming conventions in Java:

// Class names should be nouns and describe their purpose

class ReverseString {

// Method names should be verbs, describing what they do

public String reverse(String input) {

StringBuilder reversed = new StringBuilder(input);

return reversed.reverse().toString();

}

}

// Interface names should be adjectives, often ending in "able"

interface Readable {

void readData();

}

// A class implementing the Readable interface

class FileReader implements Readable {

@Override

public void readData() {

System.out.println("Reading data from a file...");

}

}

// Constants should be in uppercase with underscores

class MathConstants {

public static final double PI = 3.141592653589793;

public static final int MAX\_INT = Integer.MAX\_VALUE;

}

public class NamingConventionExample {

public static void main(String[] args) {

// Using ReverseString class

ReverseString reverser = new ReverseString();

System.out.println("Reversed: " + reverser.reverse("Java"));

// Using Readable interface

Readable reader = new FileReader();

reader.readData();

// Using constants

System.out.println("Value of PI: " + MathConstants.PI);

System.out.println("Max Integer: " + MathConstants.MAX\_INT);

}

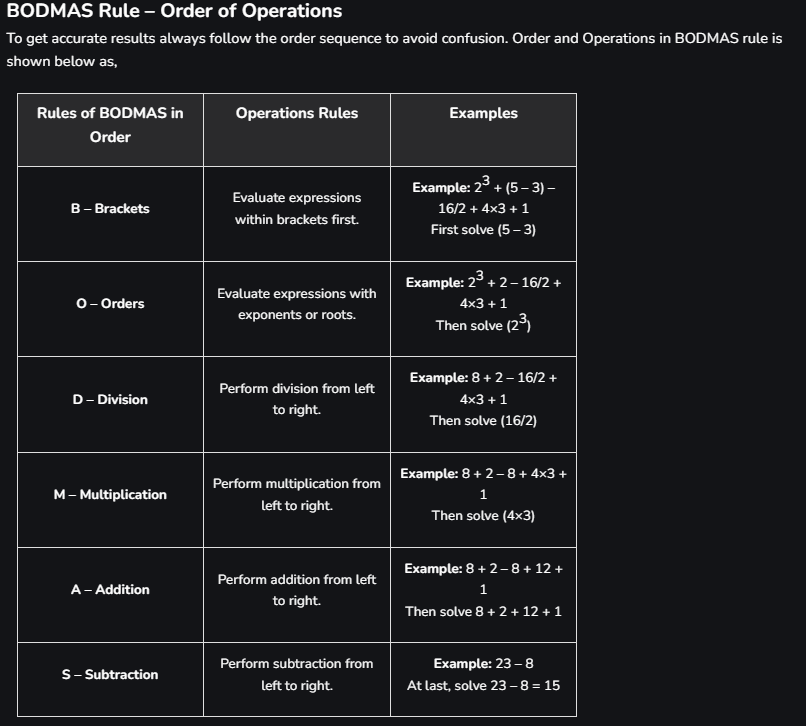
}

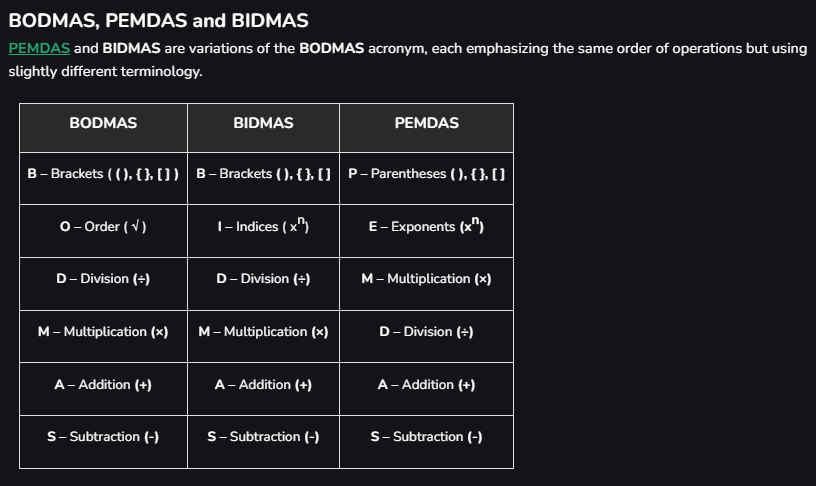
### Breakdown:

1. **Class (ReverseString)** - A noun that describes its function.
2. **Interface (Readable)** - An adjective, indicating objects that can be read.
3. **Method (reverse(), readData())** - Verbs that describe the action performed.
4. **Constants (PI, MAX\_INT)** - Uppercase and underscores for readability.

This follows Java’s best practices for naming conventions. Let me know if you need more examples! 🚀

**Arithmetic Operations:**

****

****

# Section 2:

# Decision Making in Java (if, if-else, switch, break, continue, jump)

**Decision-making statements in Java**execute a block of code based on a condition. Decision-making in programming is similar to decision-making in real life. In programming, we also face situations where we want a certain block of code to be executed when some condition is fulfilled.

## **Types of Decision-Making Statements**

1. [if](https://www.geeksforgeeks.org/java-if-statement-with-examples/)
2. [if-else](https://www.geeksforgeeks.org/java-if-else-statement-with-examples/)
3. [nested-if](https://www.geeksforgeeks.org/nested-if-in-java/)
4. if-else-if
5. [switch-case](https://www.geeksforgeeks.org/switch-statement-in-java/)
6. jump – break, continue, return

### **1. Java if Statement:** The**if statement** is the most simple decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e. if a certain condition is true then a block of statements is executed otherwise not.

**Syntax**:

*if(condition) {*

*// Statements to execute if*

*// condition is true*

*}*

**Code:**

// if condition

int a = 10;

int b = 20;

if (a > b) {

System.***out***.println("a is greater than b");

}

### **2. Java if-else Statement:** The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won’t. But what if we want to do something else if the condition is false? Here, comes the “else” statement. We can use the else statement with the if statement to execute a block of code when the condition is false.

**Syntax**:

*if(condition){*

*// Executes this block if*

*// condition is true*

*}else{*

*// Executes this block if*

*// condition is false*

*}*

**Code:**

// if else condition

if (a > b) {

System.***out***.println("a is greater than b");

} else {

System.***out***.println("b is greater than a");

}

### **3.**Java **nested-if Statement:** A nested if is an if statement that is the target of another if or else. Nested if statements mean an if statement inside an if statement. Yes, java allows us to nest if statements within if statements. i.e, we can place an if statement inside another if statement.

### 

**Syntax:**

*if (condition1) {*

*// Executes when condition1 is true*

*if (condition2)*

*{*

*// Executes when condition2 is true*

*}*

*}*

**Code:**

// Nested if else condition

int x = 30;

int y = 40;

if (x > y) {

if (x == 30) {

System.***out***.println("x is 30");

} else {

System.***out***.println("x is greater than y");

}

} else {

System.***out***.println("x is less than y");

}

### **4. Java if-else-if ladder :** Here, a user can decide among multiple options.The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that ‘if’ is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed. There can be as many as ‘else if’ blocks associated with one ‘if’ block but only one ‘else’ block is allowed with one ‘if’ block.

**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 all conditions are false*

*}*

**Code:**

// if else if condition

if (a > b) {

System.***out***.println("a is greater than b");

} else if (a < b) {

System.***out***.println("b is greater than a");

} else {

System.***out***.println("a is equal to b");

}

### **5. Java Switch Case:** The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.

**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 no cases match*

*}*

// Switch case

int day = 3;

switch (day) {

case 1: {

System.***out***.println("Sunday");

break;

}

case 2: {

System.***out***.println("Monday");

break;

}

case 3: {

System.***out***.println("Tuesday");

break;

}

case 4: {

System.***out***.println("Wednesday");

break;

}

case 5: {

System.***out***.println("Thursday");

break;

}

case 6: {

System.***out***.println("Friday");

break;

}

case 7: {

System.***out***.println("Saturday");

break;

}

default:

System.***out***.println("Invalid day");

}

### **6. jump Statements :** Java supports three jump statements: **break, continue** and **return**. These three statements transfer control to another part of the program.

* **Break:** In Java, a break is majorly used for:
  + Terminate a sequence in a switch statement (discussed above).
  + To exit a loop.
  + Used as a “civilized” form of goto.
* **Continue:**Sometimes it is useful to force an early iteration of a loop. That is, you might want to continue running the loop but stop processing the remainder of the code in its body for this particular iteration. This is, in effect, a goto just past the body of the loop, to the loop’s end. The continue statement performs such an action.

// Jump Statements : break, continue, return

// break

for (int i = 1; i <= 10; i++) {

if (i == 5) {

break;

}

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

}

// continue

for (int i = 1; i <= 10; i++) {

if (i == 5) {

System.***out***.println("five found and skpied");

continue;

}

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

}

### **Return Statement**

The return statement is used to explicitly return from a method. That is, it causes program control to transfer back to the caller of the method.

/ return

// The return statement is used to exit a method and optionally return a value to the caller.

// It plays a crucial role in controlling the flow of a program. When a return statement is encountered,

// he current method's execution stops, and the specified value (if any) is passed back to the calling code.

public int add(int a1, int b1) {

return a1 + b1;

}

// System.out.println() Statement

// System.out.println() is a method used to print text or data to the console (standard output).

// It's primarily used for displaying information to the user or for debugging purposes. It doesn't return any value and doesn't affect the program's control flow in the same way as return

public void printMessage(String message) {

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

}

// return

DecisionMaking decisionMaking = new DecisionMaking();

decisionMaking.printMessage("Hello, World!");

System.***out***.println(decisionMaking.add(50, 50));

# Ternary Operator

There is also a short-hand [if else](https://www.w3schools.com/java/java_conditions.asp), which is known as the **ternary operator** because it consists of three operands.

It can be used to replace multiple lines of code with a single line, and is most often used to replace simple if else statements:

variable *= (*condition*) ?* expressionTrue *:*  expressionFalse*;*

//Ternary Operator

int age = 20;

String eligibleAge = (age > 18)? "Eligible to vote ": "Not eligible to vote";

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

**Section 3: Flow Control**

**Decision Making** → Helps **choose** an execution path based on conditions (if-else, switch).

**Flow Control** → Manages the **execution sequence** (loops, jumps).

**Loops:**

1. for loop
2. for each
3. while loop
4. do-while loop

**for loop**

In Java, [for loop](https://www.javatpoint.com/java-for-loop) is similar to [C](https://www.javatpoint.com/c-programming-language-tutorial) and [C++](https://www.javatpoint.com/cpp-tutorial). It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code. We use the for loop only when we exactly know the number of times, we want to execute the block of code.

**Syntax**

**for**(initialization, condition, increment/decrement) {

//block of statements

}

1. // for loop
2. for (int i = 0; i < 10; i++) {
3. System.***out***.println(i);
4. }

### **for-each loop**

Java provides an enhanced for loop to traverse the data structures like array or collection. In the for-each loop, we don't need to update the loop variable. The syntax to use the for-each loop in java is given below.

**Syntax**

**for**(data\_type var : array\_name/collection\_name){

//statements

}

// for each loop

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

for (int i : arr) {

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

}

### **while loop**

The [while loop](https://www.javatpoint.com/java-while-loop) is also used to iterate over the number of statements multiple times. However, if we don't know the number of iterations in advance, it is recommended to use a while loop. Unlike for loop, the initialization and increment/decrement doesn't take place inside the loop statement in while loop.

It is also known as the entry-controlled loop since the condition is checked at the start of the loop. If the condition is true, then the loop body will be executed; otherwise, the statements after the loop will be executed.

**Syntax:**

**while**(condition){

//looping statements

}

// while loop

while (b < 15) {

System.***out***.println("While loop " + b);

b++;

}

### **do-while loop**

The [do-while loop](https://www.javatpoint.com/java-do-while-loop) checks the condition at the end of the loop after executing the loop statements. When the number of iteration is not known and we have to execute the loop at least once, we can use do-while loop.

It is also known as the exit-controlled loop since the condition is not checked in advance. The syntax of the do-while loop is given below.

**do**

{

//statements

} **while** (condition);

int a = 10;

do {

System.***out***.println("Do while loop " + a);

a++;

} while (a < 15);

**Section 4: Methods in Java**

**Package :** Package is a folder structure where we keep java files.

A **method** is a block of code which only runs when it is called.

You can pass data, known as parameters, into a method.

Methods are used to perform certain actions, and they are also known as **functions**.

Why use methods? To reuse code: define the code once, and use it many times.

public class MethodsInJava {

public static void loops(int start, int end) {

for (int i = start; i <= end; i++) {

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

}

}

// befor return type we can use access modifier like public, private, protected, default

// and static keyword is used to call method without creating object

// double is return type of method (we should specify while creating method)

// return type is used to return value from method

// method name is areaOfCircle

// method parameter is radius

public static double areaOfCircle(int radius) {

double area = Math.***PI*** \* radius \* radius;

System.***out***.println("Area of circle is: " + area);

return area;

}

// for this below method we are not using static keyword so we need to create object of class to call this method

public double squareOfTwoNumbers(int n1, int n2) {

return Math.*pow*(n1, n2);

}

public static void main(String[] args) {

System.***out***.println("Methods in Java");

//Calling a method

*loops*(1, 10);

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

*areaOfCircle*(5);

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

MethodsInJava sqareOfTwoNumbers = new MethodsInJava();

System.***out***.println(sqareOfTwoNumbers.squareOfTwoNumbers(2, 3));

}

}

## **cc**

With**method overloading**, multiple methods can have the same name with different parameters:

public class MethodOverloading {

static int plusMethod(int x, int y) {

return x + y;

}

static double plusMethod(double x, double y) {

return x + y;

}

public static void main(String[] args) {

System.***out***.println("Method Overloading");

int myNum1 = *plusMethod*(8, 5);

double myNum2 = *plusMethod*(4.3, 6.26);

System.***out***.println("int: " + myNum1);

System.***out***.println("double: " + myNum2);

}

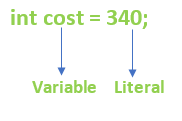
}

**Method Overriding**

**Java Literals:**

## **Literals**

In Java, **literals** are the constant values that appear directly in the program. It can be assigned directly to a variable. Java has various types of literals. The following figure represents a literal.



## **Types of Literals in Java**

There are the majorly **four** types of literals in Java:

1. Integer Literal
2. Character Literal
3. Boolean Literal
4. String Literal

**Section 7 OOPS:**

**MVC Architecture:**

When structuring a Java project, especially following the **MVC (Model-View-Controller) architecture**, you should organize your project into different folders/packages to maintain clarity, scalability, and maintainability. Here are the main folders/packages you should consider:

### 1. **Model (Data Layer)**

* **Folder Name:** model or entity
* **Purpose:** Defines the data structures and entity classes.
* **Example Classes:**
  + User.java
  + Product.java
  + Order.java
* **Additional Folders:**
  + dto (Data Transfer Objects)
  + vo (Value Objects)

Declare all you input fields using instance variable, generate getter and setter, generate hashcode, and create a constructor.

### 3. **Repository (Data Access Layer)**

* **Folder Name:** repository or dao
* **Purpose:** Handles database interaction using JPA, Hibernate, or JDBC.
* **Example Classes:**
  + UserRepository.java
  + ProductRepository.java

### 5. **Service (Business Logic Layer)**

* **Folder Name:** service
* **Purpose:** Contains business logic and interacts with repositories.
* **Example Classes:**
  + UserService.java
  + OrderService.java
* **Additional Folders:**
  + impl (For service implementations)

### 4. **Controller (Presentation Layer)**

* **Folder Name:** controller
* **Purpose:** Handles HTTP requests and calls the service layer.
* **Example Classes:**
  + UserController.java
  + OrderController.java

### 2. **Configuration (Configuration & Utility Classes)**

* **Folder Name:** config
* **Purpose:** Contains configuration files like database setup, security, and global settings.
* **Example Classes:**
  + DatabaseConfig.java
  + SecurityConfig.java

### 6. **Exception Handling (Custom Exceptions)**

* **Folder Name:** exception
* **Purpose:** Custom exception classes and global exception handlers.
* **Example Classes:**
  + UserNotFoundException.java
  + GlobalExceptionHandler.java

### 7. **Utility (Helper Classes)**

* **Folder Name:** util
* **Purpose:** Helper methods like string manipulation, date conversion, etc.
* **Example Classes:**
  + DateUtil.java
  + JwtUtil.java

### 8. **Security (Authentication & Authorization)**

* **Folder Name:** security
* **Purpose:** Security configurations, authentication filters, and JWT handling.
* **Example Classes:**
  + JwtAuthFilter.java
  + SecurityService.java

### 9. **Test (Unit & Integration Testing)**

* **Folder Name:** test
* **Purpose:** Contains JUnit and integration test cases.
* **Example:**
* src/
* ├── test/
* ├── controller/
* ├── service/
* ├── repository/

### **Example Folder Structure**

src/

│── main/

│ ├── java/com/example/project/

│ │ ├── config/

│ │ ├── controller/

│ │ ├── exception/

│ │ ├── model/

│ │ ├── repository/

│ │ ├── service/

│ │ │ ├── impl/

│ │ ├── security/

│ │ ├── util/

│── test/

│ ├── java/com/example/project/

│ │ ├── controller/

│ │ ├── service/

│ │ ├── repository/

│── resources/

│ ├── application.properties

│ ├── static/

│ ├── templates/

Would you like a more specific structure based on a particular framework like **Spring Boot**? 🚀

**Abstract class:**

**For interface abstract class is mandatory, for other its not**