JAVA Interview Questions

**1. Why is Java a platform independent language?**

[**Java language**](https://www.scaler.com/topics/java/) was developed in such a way that it does not depend on any hardware or software due to the fact that the **compiler** compiles the code and then converts it to platform-independent byte code which can be run on multiple systems.

* The only condition to run that byte code is for the machine to have a runtime environment (JRE) installed in it

2.What do you know about Java?

Java is a high-level programming language originally developed by Sun Microsystems and released in 1995. Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX

Java is a popular programming language.

Java is used to develop mobile apps, web apps, desktop apps, games and much more.

3. What are the supported platforms by Java Programming Language?

Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX/Linux like HP-Unix, Sun Solaris, Redhat Linux, Ubuntu, CentOS, etc.

4. List any five features of Java?

**5. What are the main features of OOPs?**

The main feature of the OOPs, also known as 4 pillars or basic principles of OOPs are as follows:

1. Encapsulation
2. Data Abstraction
3. Polymorphism
4. Inheritance

**6. What is Encapsulation?**

Encapsulation is the binding of data and methods that manipulate them into a single unit such that the sensitive data is hidden from the users  
It is implemented as the processes mentioned below:

1. **Data hiding:** A language feature to restrict access to members of an object. For example, private and protected members in C++.
2. **Bundling of data and methods together:** Data and methods that operate on that data are bundled together. For example, the data members and member methods that operate on them are wrapped into a single unit known as a class.

### ****7. What is Abstraction?****

Abstraction is similar to data encapsulation and is very important in OOP. It means showing only the necessary information and hiding the other irrelevant information from the user. Abstraction is implemented using classes and interfaces.

**abstract** **class** Animal {

**abstract** **void** walk();

**void** eat()

    {

        System.out.println("The animal is eating.");

    }

    Animal()

    {

        System.out.println(

            "An Animal is going to be created.");

    }

}

**class** Cow **extends** Animal {

    Cow() { System.out.println("You have created a Cow"); }

**void** walk() { System.out.println("Cow is walking."); }

}

**class** Goat **extends** Animal {

    Goat()

    {

        System.out.println("You have created a Goat");

    }

**void** walk() { System.out.println("Goat is walking."); }

}

**public** **class** OOPS {

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

    {

        Cow cow = **new** Cow();

        cow.walk();

        cow.eat();

        Goat goat = **new** Goat();

        goat.walk();

        goat.eat();

    }

}

### ****8. What is Polymorphism?****

The word “**Polymorphism**” means having many forms. It is the property of some code to behave differently for different contexts. For example, in C++ language, we can define multiple functions having the same name but different working depending on the context.

Polymorphism can be classified into two types based on the time when the call to the object or function is resolved. They are as follows:

A. Compile Time Polymorphism  
B. Runtime Polymorphism

**A) Compile-Time Polymorphism**

Compile time polymorphism, also known as static polymorphism or early binding is the type of polymorphism where the binding of the call to its code is done at the compile time. Method overloading or operator overloading are examples of compile-time polymorphism.

**B) Runtime Polymorphism**

Also known as dynamic polymorphism or late binding, runtime polymorphism is the type of polymorphism where the actual implementation of the function is determined during the runtime or execution. Method overriding is an example of this method.

5. List two Java IDE’s?

Netbeans, Eclipse, etc, Intelij

## **Java - What is OOP?**

OOP stands for **Object-Oriented Programming**.

Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both data and methods.

Object-oriented programming has several advantages over procedural programming:

* OOP is faster and easier to execute
* OOP provides a clear structure for the programs
* OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
* OOP makes it possible to create full reusable applications with less code and shorter development time

# **Java Classes and Objects**

Java is an object-oriented programming language.

Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake.

A Class is like an object constructor, or a "blueprint" for creating objects.

### 3. What is a Class?

A **class** is a building block of Object Oriented Programs. It is a user-defined data type that contains the data members and member functions that operate on the data members. It is like a blueprint or template of objects having common properties and methods.

An **object** is an instance of a class. Data members and methods of a class cannot be used directly. We need to create an object (or instance) of the class to use them. In simple terms, they are the actual world entities that have a state and behavior.

## **Java Constructors**

A constructor in Java is a **special method** that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes:

# **Java Modifiers**

We divide modifiers into two groups:

* **Access Modifiers** - controls the access level
* **Non-Access Modifiers** - do not control access level, but provides other functionality

|  |  |  |
| --- | --- | --- |
| **Modifier** | **Description** | **Try it** |
| public | The class is accessible by any other class | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_mod_public) |
| *default* | The class is only accessible by classes in the same package. This is used when you don't specify a modifier. |  |
| private | The code is only accessible within the declared class | |
| protected | The code is accessible in the same package and subclasses | |

## **Non-Access Modifiers**

For **classes**, you can use either final or abstract:

|  |  |  |
| --- | --- | --- |
| final | The class cannot be inherited by other classes (You will learn more about inheritance in the [Inheritance chapter](https://www.w3schools.com/java/java_inheritance.asp)) | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_inherit_final) |
| abstract | The class cannot be used to create objects (To access an abstract class, it must be inherited from another class. You will learn more about inheritance and abstraction in the [Inheritance](https://www.w3schools.com/java/java_inheritance.asp) and [Abstraction](https://www.w3schools.com/java/java_abstract.asp) chapters) |  |

## 6. Java Data Types

Data types are divided into two groups:

* Primitive data types - includes byte, short, int, long, float, double, boolean and char
* Non-primitive data types - such as [String](https://www.w3schools.com/java/java_strings.asp), [Arrays](https://www.w3schools.com/java/java_arrays.asp) and [Classes](https://www.w3schools.com/java/java_classes.asp) (you will learn more about these in a later chapter)

# **Java Encapsulation**

## **Encapsulation**

The meaning of **Encapsulation**, is to make sure that "sensitive" data is hidden from users. To achieve this, you must:

* declare class variables/attributes as private
* provide public **get** and **set** methods to access and update the value of a private variable

# **Java Packages**

## **Java Packages & API**

A package in Java is used to group related classes. Think of it as **a folder in a file directory**. We use packages to avoid name conflicts, and to write a better maintainable code. Packages are divided into two categories:

* Built-in Packages (packages from the Java API)
* User-defined Packages (create your own packages)

# **Java Inheritance**

## **Java Inheritance (Subclass and Superclass)**

In Java, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories:

* **subclass** (child) - the class that inherits from another class
* **superclass** (parent) - the class being inherited from

To inherit from a class, use the extends keyword.

# **Java Polymorphism**

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

Like we specified in the previous chapter; [**Inheritance**](https://www.w3schools.com/java/java_inheritance.asp) lets us inherit attributes and methods from another class. **Polymorphism** uses those methods to perform different tasks. This allows us to perform a single action in different ways.

# **Java Abstraction**

## **Abstract Classes and Methods**

Data **abstraction** is the process of hiding certain details and showing only essential information to the user.  
Abstraction can be achieved with either **abstract classes** or [**interfaces**](https://www.w3schools.com/java/java_interface.asp) (which you will learn more about in the next chapter).

The abstract keyword is a non-access modifier, used for classes and methods:

* **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

* **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

An abstract class can have both abstract and regular methods:

abstract class Animal {

public abstract void animalSound();

public void sleep() {

System.out.println("Zzz");

}

}

# 7. **Java Operators**

Operators are used to perform operations on variables and values.

# **Java Keywords**

|  |  |
| --- | --- |
| [abstract](https://www.w3schools.com/java/ref_keyword_abstract.asp) | A non-access modifier. Used for classes and methods: An abstract class cannot be used to create objects (to access it, it must be inherited from another class). An abstract method can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from) |
| assert | For debugging |
| [boolean](https://www.w3schools.com/java/ref_keyword_boolean.asp) | A data type that can only store true and false values |
| [break](https://www.w3schools.com/java/ref_keyword_break.asp) | Breaks out of a loop or a switch block |
| [byte](https://www.w3schools.com/java/ref_keyword_byte.asp) | A data type that can store whole numbers from -128 and 127 |
| [case](https://www.w3schools.com/java/ref_keyword_case.asp) | Marks a block of code in switch statements |
| [catch](https://www.w3schools.com/java/ref_keyword_catch.asp) | Catches exceptions generated by try statements |
| [char](https://www.w3schools.com/java/ref_keyword_char.asp) | A data type that is used to store a single character |
| [class](https://www.w3schools.com/java/ref_keyword_class.asp) | Defines a class |
| [continue](https://www.w3schools.com/java/ref_keyword_break.asp) | Continues to the next iteration of a loop |
| const | Defines a constant. Not in use - use [final](https://www.w3schools.com/java/ref_keyword_final.asp) instead |
| [default](https://www.w3schools.com/java/ref_keyword_default.asp) | Specifies the default block of code in a switch statement |
| [do](https://www.w3schools.com/java/ref_keyword_do.asp) | Used together with while to create a do-while loop |
| [double](https://www.w3schools.com/java/ref_keyword_double.asp) | A data type that can store whole numbers from 1.7e−308 to 1.7e+308 |
| [else](https://www.w3schools.com/java/ref_keyword_else.asp) | Used in conditional statements |
| [enum](https://www.w3schools.com/java/ref_keyword_enum.asp) | Declares an enumerated (unchangeable) type |
| exports | Exports a package with a module. New in Java 9 |
| [extends](https://www.w3schools.com/java/ref_keyword_extends.asp) | Extends a class (indicates that a class is inherited from another class) |
| [final](https://www.w3schools.com/java/ref_keyword_final.asp) | A non-access modifier used for classes, attributes and methods, which makes them non-changeable (impossible to inherit or override) |
| [finally](https://www.w3schools.com/java/ref_keyword_finally.asp) | Used with exceptions, a block of code that will be executed no matter if there is an exception or not |
| [float](https://www.w3schools.com/java/ref_keyword_float.asp) | A data type that can store whole numbers from 3.4e−038 to 3.4e+038 |

## **Java Lambda Expressions**

Lambda Expressions were added in Java 8.

A lambda expression is a short block of code which takes in parameters and returns a value. Lambda expressions are similar to methods, but they do not need a name and they can be implemented right in the body of a method.

The simplest lambda expression contains a single parameter and an expression:

*parameter* -> *expression*

To use more than one parameter, wrap them in parentheses:

*(*parameter1*,* parameter2*)* -> expression

Use a lambda expression in the ArrayList's forEach() method to print every item in the list:

import java.util.ArrayList;

public class Main {

public static void main(String[] args) {

ArrayList<Integer> numbers = new ArrayList<Integer>();

numbers.add(5);

numbers.add(9);

numbers.add(8);

numbers.add(1);

numbers.forEach( (n) -> { System.out.println(n); } );

}

}

# Java Enums

An enum is a special "class" that represents a group of **constants** (unchangeable variables, like final variables).

To create an enum, use the enum keyword (instead of class or interface), and separate the constants with a comma. Note that they should be in uppercase letters:

# Java HashMap

## Java HashMap

In the [ArrayList](https://www.w3schools.com/java/java_arraylist.asp) chapter, you learned that Arrays store items as an ordered collection, and you have to access them with an index number (int type). A HashMap however, store items in "**key**/**value**" pairs, and you can access them by an index of another type (e.g. a String).

One object is used as a key (index) to another object (value). It can store different types: String keys and Integer values, or the same type, like: String keys and String values

# **Java Methods**

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.

## **Create a Method**

A method must be declared within a class. It is defined with the name of the method, followed by parentheses **()**. Java provides some pre-defined methods, such as System.out.println(), but you can also create your own methods to perform certain actions

# **Java Method Parameters**

## **Parameters and Arguments**

Information can be passed to methods as parameter. Parameters act as variables inside the method.

Parameters are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.