I'm not a robot

Updated on October 18, 2021 by Arpit Mandliya 5.3K Do you need help preparing for your Java OOPS interview? Object-oriented programming (OOP) is a core aspect of Java, and a strong understanding of its principles is essential. This guide covers the top 50 Java OOPS interview questions and answers, addressing both fundamental and advanced topics. With clear explanations and practical examples, these questions will help you approach technical rounds with confidence. Fun Fact: According to the TIOBE Index (2025), Java remains a top 5 programming language, with over 9 million developers using it. A major reason for its popularity is its strong OOPS foundation. Here is a list of basic Java Object Oriented Programming questions and answers for interviews: What are the four main principles of Object-Oriented Programming? The four main principles of OOPS are: Encapsulation – Wrapping data and methods into a single unit (class) to restrict direct access to data. Abstraction – Hiding implementation details and exposing only necessary functionalities using abstract classes or interfaces. Inheritance – Allowing one class (child) to inherit properties and methods from another (parent) to promote reusability. Polymorphism – Allowing a single method or operator to have multiple implementations (method overloading and method overriding). How is abstraction different from encapsulation in Java? Abstraction hides unnecessary details and exposes only the essential parts. It is implemented using abstract classes and interfaces. Encapsulation, on the other hand, restricts direct access to an object’s data by using access modifiers like private, protected, and public. While abstraction is about hiding implementation, encapsulation is about data security and integrity. What is method overloading and method overriding? Provide examples. Method Overloading: When multiple methods in the same class have the same name but different parameter lists. Example: class MathOperations {   int add(int a, int b) { return a + b; }   double add(double a, double b) { return a + b; } } Method Overriding: When a subclass provides a specific implementation of a method already defined in its superclass. Example: class Parent {   void display() { System.out.println(“Parent class method”); } } class Child extends Parent {   @Override   void display() { System.out.println(“Child class method”); } } Why is multiple inheritance not supported in Java? Multiple inheritance is not supported in Java to avoid ambiguity issues caused by the diamond problem. If two parent classes have the same method, the compiler cannot determine which one to inherit. Instead, Java provides interfaces, allowing a class to implement multiple interfaces without ambiguity. Here are some common Java Object Oriented Programming interview questions for freshers: What is the difference between a class and an object? A class is a blueprint for creating objects. It defines attributes (variables) and behaviors (methods). An object is an instance of a class with specific values assigned to its attributes. Example: class Car {   String brand;   void drive() { System.out.println(“Car is driving”); } } Car myCar = new Car(); // Object creation What is the significance of the ‘this’ keyword in Java? See also Top 25+ Computer Vision Interview Questions and AnswersThe this keyword refers to the current instance of a class. It is used to: Differentiate instance variables from local variables when they have the same name. Call another constructor in the same class. Pass the current instance as a parameter. Example: class Employee {   String name;   Employee(String name) { this.name = name; } } How does Java achieve runtime polymorphism? Java achieves runtime polymorphism through method overriding. The overridden method in a subclass is called at runtime based on the object type, even when referenced by a parent class. Example: class Animal {   void sound() { System.out.println(“Animal makes a sound”); } } class Dog extends Animal {   void sound() { System.out.println(“Dog barks”); } } Animal obj = new Dog(); obj.sound(); // Outputs: Dog barks What is the purpose of the ‘super’ keyword in Java? The super keyword is used to refer to the immediate parent class. It can be used to: Call the parent class constructor. Access parent class methods. Access parent class variables. Example: class Parent {   void display() { System.out.println(“Parent method”); } } class Child extends Parent {   void show() { super.display(); } } Let’s go through important Java OOPS programming interview questions and answers for experienced candidates: How does Java manage memory with respect to objects? Java uses automatic memory management with the help of the Garbage Collector (GC). When an object is no longer referenced, the GC removes it to free up memory. Java memory consists of: Heap (stores objects). Stack (stores method calls and local variables). Method area (stores class structures). What are the different types of constructors in Java? Default Constructor – No parameters, initializes objects with default values. Parameterized Constructor – Takes arguments to initialize instance variables. Copy Constructor – Copies values from one object to another. Example: class Student {   String name;   Student(String name) { this.name = name; } } How is object cloning implemented in Java? Java supports shallow cloning using the clone() method from the Cloneable interface. Example: class Employee implements Cloneable {   String name;   Employee(String name) { this.name = name; }   protected Object clone() throws CloneNotSupportedException {   return super.clone();   } } Deep cloning requires manual copying of referenced objects. What is a shallow copy and deep copy in Java? Shallow Copy: Copies field values but does not create new referenced objects. Changes in the original object affect the copied object. Deep Copy: Creates a new copy of referenced objects, making them independent. Example of deep copy: class Address {   String city;   Address(String city) { this.city = city; } } class Person {   String name;   Address address;   Person(String name, Address address) {   this.name = name;   this.address = new Address(address.city); // Deep Copy   } } If you have 2 years of experience, you might come across such Java and OOPS interview questions: Why did you choose Java for your career? Describe a situation where you had to debug a complex object-oriented issue. How did you solve it? If you had to redesign an existing system to improve reusability, which OOPS principles would you focus on and why? These interview questions for OOPS in Java are for candidates with three years of experience: What is the most challenging Java project you have worked on? How do you handle a situation where your team disagrees on the best OOPS approach for a project? If you need to implement a flexible payment system, which OOPS concepts would you apply and how? See also Top 45+ Database Testing Interview Questions and Answers These Java Object Oriented interview questions are for candidates with 5 years of experience: What design patterns have you used in your Java projects, and why? How do you mentor junior developers on OOPS principles? You need to refactor a monolithic Java application into a microservices-based architecture. How would you approach the OOPS design? If you are at a senior level and have around 10 years of experience, you might come across such Java OOPS interview questions: How has your understanding of OOPS evolved over the years? Have you ever had to optimize an object-oriented Java system for performance? How did you do it? Given an existing Java application with tight coupling, how would you refactor it to follow SOLID principles? You might also come across OOPS concepts in Java interview questions like these: What is the difference between an interface and an abstract class? An interface defines a contract that classes must follow. It contains only abstract methods (before Java 8) and allows default and static methods (from Java 8). Interfaces support multiple inheritance since a class can implement multiple interfaces. An abstract class can have both abstract and concrete methods. It can include constructors and instance variables. Unlike interfaces, abstract classes can have method implementations but do not support multiple inheritance. How does Java implement multiple inheritance? Java does not support multiple inheritance through classes to avoid ambiguity (diamond problem). Instead, it uses interfaces. A class can implement multiple interfaces, allowing it to inherit behaviors from different sources without conflicts. What is dynamic method dispatch in Java? Dynamic method dispatch, also called runtime polymorphism, is the process where method calls are resolved at runtime based on the object’s actual type, not the reference type. These are some core Java OOP questions and answers for interviews: What is an association, aggregation, and composition in Java? Association: A relationship between two classes where both objects exist independently (e.g., Student and Teacher). Aggregation: A weaker relationship where the child can exist independently, but the parent owns it (e.g., Department and Employee). Composition: A strong relationship where the child object’s existence depends on the parent (e.g., Car and Engine). What is the role of access modifiers in OOPS? Access modifiers control visibility of class members: private – Accessible only within the class. default – Accessible within the same package. protected – Accessible in the same package and subclasses. public – Accessible from anywhere. Let’s go through some advanced Java Object Oriented interview questions and answers: What are the different types of design patterns in Java? Creational Patterns – Singleton, Factory, Builder. Structural Patterns – Adapter, Composite, Proxy. Behavioral Patterns – Strategy, Observer, Command. How do lambda expressions fit into Java’s object-oriented model? Lambda expressions provide a concise way to implement functional interfaces (interfaces with a single abstract method). They allow inline implementations without creating a separate class. Example: interface Calculator {   int operate(int a, int b); } Calculator add = (a, b) -> a + b; System.out.println(add.operate(5, 3)); // Outputs: 8 What is the function of reflection in Java OOPS? Reflection allows introspection and manipulation of classes, methods, and fields at runtime. It is useful in frameworks, serialization, and dependency injection. See also Top 25+ Performance Testing Interview Questions and AnswersExample: Class obj = Class.forName(“java.util.ArrayList”); System.out.println(obj.getMethods()); Also Read - Top 25+ Python OOPs Interview Question (2025) You should also take a look at these OOPS in JavaScript interview questions: How is object-oriented programming implemented in JavaScript? JavaScript is prototype-based, meaning objects inherit from other objects instead of classes. Objects can be created using constructors, prototypes, or ES6 classes. What is prototypal inheritance, and how does it differ from classical inheritance? Prototypal inheritance allows objects to inherit properties directly from another object using the prototype chain. Unlike classical inheritance, which relies on class hierarchies, JavaScript objects inherit dynamically. Example: let parent = { greet: function() { console.log(“Hello”); } }; let child = Object.create(parent); child.greet(); // Outputs: Hello Also Read - Top 20 PHP OOPs Interview Questions and Answers Here are some coding Java object oriented interview questions: Write a Java program to demonstrate method overriding. class Parent {   void show() { System.out.println(“Parent class method”); } } class Child extends Parent {   @Override   void show() { System.out.println(“Child class method”); } } public class Test {   public static void main(String[] args) {   Parent obj = new Child();   obj.show(); // Outputs: Child class method   } } Implement a singleton class in Java. class Singleton {   private static Singleton instance;   private Singleton() {} // Private constructor   public static Singleton getInstance() {   if (instance == null) {   instance = new Singleton();   }   return instance;   } } Create an interface and implement it in multiple classes with different behaviors. interface Animal {   void sound(); } class Dog implements Animal {   public void sound() { System.out.println(“Dog barks”); } } class Cat implements Animal {   public void sound() { System.out.println(“Cat meows”); } } public class Test {   public static void main(String[] args) {   Animal a1 = new Dog();   Animal a2 = new Cat();   a1.sound();   a2.sound();   } } Also Read - Top 20 C++ OOPs Interview Questions and Answers Implement a real-world example of polymorphism in Java. Create a Java class that follows the principle of encapsulation. Write a Java program to demonstrate the Factory Design Pattern. Implement an abstract class with a concrete method and abstract methods. Also Read - Top 30+ C# OOPs Interview Questions and Answers What is the difference between instance and static methods? How does Java handle object destruction? Can a constructor be private? If yes, when would you use it? What is the difference between early binding and late binding in Java? Also Read - Top 20 OOPs ABAP Interview Questions and Answers Here are some common interview questions for OOPS in Java in MCQ form: Which of the following is not an OOPS principle? a) Encapsulationb) Inheritancec) Compilationd) Polymorphism Programming paradigm based on the concept of objects "Object-oriented" redirects here. For other meanings of object-oriented, see Object-orientation. UML notation for a class. This Button class has variables for data, and functions. Through inheritance, a subclass can be created as a subset of the Button class. Objects are instances of a class. Object-oriented programming (OOP) is a programming paradigm based on the concept of objects.[1] Objects can contain data (called fields, attributes or properties) and have actions they can perform (called procedures or methods and implemented in code). In OOP, computer programs are designed by making them out of objects that interact with one another.[2][3] Many of the most widely used programming languages (such as C++, Java,[4] and Python) support object-oriented programming to a greater or lesser degree, typically as part of multiple paradigms in combination with others such as imperative programming and declarative programming. Significant object-oriented languages include Ada, ActionScript, C++, Common Lisp, C#, Dart, Eiffel, Fortran 2003, Haxe, Java,[4] JavaScript, Kotlin, Logo, MATLAB, Objective-C, Object Pascal, Perl, PHP, Python, R, Raku, Ruby, Scala, SIMSCRIPT, Simula, Smalltalk, Swift, Vala and Visual Basic.NET. The idea of "objects" in programming started with the artificial intelligence group at MIT in the late 1950s and early 1960s. Here, "object" referred to LISP atoms with identified properties (attributes).[5][6] Another early example was Sketchpad created by Ivan Sutherland at MIT in 1960–1961. In the glossary of his technical report, Sutherland defined terms like "object" and "instance" (with the class concept covered by "master" or "definition"), albeit specialized to graphical interaction.[7] Later, in 1968, AED-0, MIT's version of the ALGOL programming language, connected data structures ("plexes") and procedures, prefiguring what were later termed "messages", "methods", and "member functions".[8][9] Topics such as data abstraction and modular programming were common points of discussion at this time. Meanwhile, in Norway, Simula was developed during the years 1961–1967.[8] Simula introduced essential object-oriented ideas, such as classes, inheritance, and dynamic binding.[10] Simula was used mainly by researchers involved with physical modelling, like the movement of ships and their content through cargo ports.[10] Simula is generally accepted as being the first language with the primary features and framework of an object-oriented language.[11] I thought of objects being like biological cells and/or individual computers on a network, only able to communicate with messages (so messaging came at the very beginning – it took a while to see how to do messaging in a programming language efficiently enough to be useful). Alan Kay, [1] Influenced by both MIT and Simula, Alan Kay began developing his own ideas in November 1966. He would go on to create Smalltalk, an influential object-oriented programming language. By 1967, Kay was already using the term "object-oriented programming" in conversation.[1] Although sometimes called the "father" of object-oriented programming,[12] Kay has said his ideas differ from how object-oriented programming is commonly understood, and has implied that the computer science establishment did not adopt his notion.[1] A 1976 MIT memo co-authored by Barbara Liskov lists Simula 67, CLU, and Alphard as object-oriented languages, but does not mention Smalltalk.[13] In the 1970s, the first version of the Smalltalk programming language was developed at Xerox PARC by Alan Kay, Dan Ingalls and Adele Goldberg. Smalltalk-72 was notable for use of objects at the language level and its graphical development environment.[14] Smalltalk was a fully dynamic system, allowing users to create and modify classes as they worked.[15] Much of the theory of OOP was developed in the context of Smalltalk, for example multiple inheritance.[16] In the late 1970s and 1980s, object-oriented programming rose to prominence. The Flavors object-oriented Lisp was developed starting 1979, introducing multiple inheritance and mixins.[17] In August 1981, Byte Magazine highlighted Smalltalk and OOP, introducing these ideas to a wide audience.[18] LOOPS, the object system for Interlisp-D, was influenced by Smalltalk and Flavors, and a paper about it was published in 1982.[19] In 1986, the first Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) was attended by 1,000 people. This conference marked the beginning of efforts to consolidate Lisp object systems, eventually resulting in the Common Lisp Object System. In the 1980s, there were a few attempts to design processor architectures that included hardware support for objects in memory, but these were not successful. Examples include the Intel iAPX 432 and the Linn Smart Rekursiv. In the mid-1980s, new object-oriented languages like Objective-C, C++, and Eiffel emerged. Objective-C was developed by Brad Cox, who had used Smalltalk at ITT Inc. Bjarne Stroustrup created C++ based on his experience using Simula for his PhD thesis.[14] Bertrand Meyer produced the first design of the Eiffel language in 1985, which focused on software quality using a design by contract approach.[20] In the 1990s, object-oriented programming became the main way of programming, especially as more languages supported it. These included Visual FoxPro 3.0,[21][22] C++,[23] and Delphi[citation needed]. OOP became even more popular with the rise of graphical user interfaces, which used objects for buttons, menus and other elements. One well-known example is Apple's Cocoa framework, used on Mac OS X and written in Objective-C. OOP toolkits also enhanced the popularity of event-driven programming.[citation needed] At ETH Zürich, Niklaus Wirth and his colleagues created new approaches to OOP. Modula-2 (1978) and Oberon (1987), included a distinctive approach to object orientation, classes, and type checking across module boundaries. Inheritance is not obvious in Wirth's design since his nomenclature looks in the opposite direction: It is called type extension and the viewpoint is from the parent down to the inheritor. Many programming languages that existed before OOP have added object-oriented features, including Ada, BASIC, Fortran, Pascal, and COBOL. This sometimes caused compatibility and maintainability issues, as these languages were not originally designed with OOP in mind. In the new millenium, new languages like Python and Ruby have emerged that combine object-oriented and procedural styles. The most commercially important "pure" object-oriented languages continue to be Java, developed by Sun Microsystems, as well as C# and Visual Basic.NET (VB.NET), both designed for Microsoft's .NET platform. These languages show the benefits of OOP by creating abstractions from implementation. The .NET platform supports cross-language inheritance, allowing programs to use objects from multiple languages together. See also: Comparison of programming languages (object-oriented programming) and List of object-oriented programming terms Object-oriented programming focuses on working with objects, but not all OOP languages have every feature linked to OOP. Below are some common features of languages that are considered strong in OOP or support it along with other programming styles. Important exceptions are also noted.[24][25][26][27] Christopher J. Date pointed out that comparing OOP with other styles, like relational programming, is difficult because there isn't a clear, agreed-upon definition of OOP.[28] Further information: Imperative programming and Structured programming Features from imperative and structured programming are present in OOP languages and are also found in non-OOP languages. Variables hold different data types like integers, strings, lists, and hash tables. Some data types are built-in while others result from combining variables using memory pointers. Procedures – also known as functions, methods, routines, or subroutines – take input, generate output, and work with data. Modern languages include structured programming constructs like loops and conditionals. Support for modular programming lets programmers organize related procedures into files and modules. This makes programs easier to manage. Each modules has its own namespace, so items in one module will not conflict with items in another. Object-oriented programming (OOP) was created to make code easier to reuse and maintain.[29] However, it was not designed to clearly show the flow of a program's instructions—that was left to the compiler. As computers began using more parallel processing and multiple threads, it became more important to understand and control how instructions flow. This is difficult to do with OOP.[30][31][32][33] Main article: Object (computer science) An object is a type of data structure that has two main parts: fields and methods. Fields may also be known as members, attributes, or properties, and hold information in the form of state variables. Methods are actions, subroutines, or procedures, defining the object's behavior in code. Objects are usually stored in memory, and in many programming languages, they work like pointers that link directly to a contiguous block containing the object instances's data. Objects can contain other objects. This is called object composition. For example, an Employee object might have an Address object inside it, along with other information like "first_name" and "position". This type of structures shows "has-a" relationships, like "an employee has an address". Some believe that OOP places too much focus on using objects rather than on algorithms and data structures.[34][35] For example, programmer Rob Pike pointed out that OOP can make programmers think more about type hierarchy than composition.[36] He has called object-oriented programming "the Roman numerals of computing".[37] Rich Hickey, creator of Clojure, described OOP as overly simplistic, especially when it comes to representing real-world things that change over time.[35] Alexander Stepanov said that OOP tries to fit everything into a single type, which can be limiting. He argued that sometimes we need multisorted algebras—families of interfaces that span multiple types, such as in generic programming. Stepanov also said that calling everything an "object" doesn't add much understanding.[34] Sometimes, objects represent real-world things and processes in digital form.[38] For example, a graphics program may have objects such as "circle", "square", and "menu". An online shopping system might have objects such as "shopping cart", "customer", and "product". Niklaus Wirth said, "This paradigm [OOP] closely reflects the structure of systems in the real world and is therefore well suited to model complex systems with complex behavior".[39] However, more often, objects represent abstract entities, like an open file or a unit converter. Not everyone agrees that OOP makes it easy to copy the real world exactly or that doing so is even necessary. Bob Martin suggests that because classes are software, their relationships don't match the real-world relationships they represent.[40] Bertrand Meyer argues in Object-Oriented Software Construction, that a program is not a model of the world but a model of some part of the world; "Reality is a cousin twice removed".[41] Steve Yegge noted that natural languages lack the OOP approach of strictly prioritizing things (objects/nouns) before actions (methods/verbs), as opposed to functional programming which does the reverse.[42] This can sometimes make OOP solutions more complicated than those written in procedural programming.[43] Most OOP languages allow reusing and extending code through "inheritance". This inheritance can use either "classes" or "prototypes", which have some differences but use similar terms for ideas like "object" and "instance". In class-based programming, the most common type of OOP, every object is an instance of a specific class. The class defines the data format, like variables (e.g., name, age) and methods (actions the object can take). Every instance of the class has the same set of variables and methods. Objects are created using a special method in the class known as a constructor. Here are a few key terms in class-based OOP: Class variables – belong to the class itself, so all objects in the class share one copy. Instance variables – belong to individual objects; every object has its own version of these variables. Member variables – refers to both the class and instance variables that are defined by a particular class. Class methods – linked to the class itself and can only use class variables. Instance methods – belong to individual objects, and can use both instance and class variables Classes may inherit from other classes, creating a hierarchy of "subclasses". For example, an "Employee" class might inherit from a "Person" class. This means the Employee object will have all the variables from Person (like name variables) plus any new variables (like job position and salary). Similarly, the subclass may expand the interface with new methods. Most languages also allow the subclass to override the methods defined by superclasses. Some languages support multiple inheritance, where a class can inherit from more than one class, and other languages similarly support mixins or traits. For example, a mixin called UnicodeConversionMixin might add a method unicode_to_ascii() to both a FileReader and a WebPageScraper class. Some classes are abstract, meaning they cannot be directly instantiated into objects; they're only meant to be inherited into other classes. Other classes are utility classes which contain only class variables and methods and are not meant to be instantiated or subclassed.[44] In prototype-based programming, there aren't any classes. Instead, each object is linked to another object, called its prototype or parent. In Self, an object may have multiple or no parents,[45] but in the most popular prototype-based language, Javascript, every object has exactly one prototype link, up to the base Object type whose prototype is null. The prototype acts as a model for new objects. For example, if you have an object fruit, you can make two objects apple and orange, based on it. There is no fruit class, but they share traits from the fruit prototype. Prototype-based languages also allow objects to have their own unique properties, so the apple object might have an attribute sugar_content, while the orange or fruit objects do not. Some languages, like Go, don't use inheritance at all.[46] Instead, they encourage "composition over inheritance", where objects are built using smaller parts instead of parent-child relationships. For example, instead of inheriting from class Person, the Employee class could simply contain a Person object. This lets the Employee class control how much of Person it exposes to other parts of the program. Delegation is another language feature that can be used as an alternative to inheritance. Programmers have different opinions on inheritance. Bjarne Stroustrup, author of C++, has stated that it is possible to do OOP without inheritance.[47] Rob Pike has criticized inheritance for creating complicated hierarchies instead of simpler solutions.[48] See also: Object-oriented design People often think that if one class inherits from another, it means the subclass "is a" more specific version of the original class. This presumes the program semantics are that objects from the subclass can always replace objects from the original class without problems. This concept is known as behavioral subtyping, more specifically the Liskov substitution principle. However, this is often not true, especially in programming languages that allow mutable objects, objects that change after they are created. In fact, subtype polymorphism as enforced by the type checker in OOP languages cannot guarantee behavioral subtyping in most if not all contexts. For example, the circle-ellipse problem is notoriously difficult to handle using OOP's concept of inheritance. Behavioral subtyping is undecidable in general, so it cannot be easily implemented by a compiler. Because of this, programmers must carefully design class hierarchies to avoid mistakes that the programming language itself cannot catch. When a method is called on an object, the object itself—not outside code—decides which specific code to run. This process, called dynamic dispatch, usually happens at run time by checking a table linked to the object to find the correct method. In this context, a method call is also known as message passing, meaning the method name and its inputs are like a message sent to the object for it to act on. If the method choice depends on more than one type of object (such as other objects passed as parameters), it's called multiple dispatch. Dynamic dispatch works together with inheritance: if an object doesn't have the requested method, it looks up to its parent class (delegation), and continues up the chain until it finds the method or reaches the top. Data abstraction is a way of organizing code so that only certain parts of the data are visible to related functions (data hiding). This helps prevent mistakes and makes the program easier to manage. Because data abstraction works well, many programming styles, like object-oriented programming and functional programming, use it as a key principle. Encapsulation is another important idea in programming. It means keeping the internal details of an object hidden from the outside code. This makes it easier to change how an object works on the inside without affecting other parts of the program, such as in code refactoring. Encapsulation also helps keep related code together (decoupling), making it easier for programmers to understand. In object-oriented programming, objects act as a barrier between their internal workings and external code. Outside code can only interact with an object by calling specific public methods or variables. If a class only allows access to its data through methods and not directly, this is called information hiding. When designing a program, it's often recommended to keep data as hidden as possible. This means using local variables inside functions when possible, then private variables (which only the object can use), and finally public variables (which can be accessed by any part of the program) if necessary. Keeping data hidden helps prevent problems when changing the code later.[49] Some programming languages, like Java, control information hiding by marking variables as private (hidden) or public (accessible).[50] Other languages, like Python, rely on naming conventions, such as starting a private method's name with an underscore. Intermediate levels of access also exist, such as Java's protected keyword, (which allows access from the same class and its subclasses, but not objects of a different class), and the internal keyword in C#, Swift, and Kotlin, which restricts access to files within the same module.[51] Abstraction and information hiding are important concepts in programming, especially in object-oriented languages.[52] Programs often create many copies of objects, and each one works independently. Supporters of this approach say it makes code easier to reuse and intuitively represents real-world situations.[53] However, others argue that object-oriented programming does not enhance readability or modularity.[54][55] Eric S. Raymond has written that object-oriented programming languages tend to encourage thickly layered programs that destroy transparency.[56] Raymond compares this unfavourably to the approach taken with Unix and the C programming language.[56] One programming principle, called the "open/closed principle", says that classes and functions should be "open for extension, but closed for modification". Luca Cardelli has stated that OOP languages have "extremely poor modularity properties with respect to class extension and modification", and tend to be extremely complex.[54] The latter point is reiterated by Joe Armstrong, the principal inventor of Erlang, who is quoted as saying:[55] The problem with object-oriented languages is they've got all this implicit environment that they carry around with them. You wanted a banana but what you got was a gorilla holding the banana and the entire jungle. Leo Brodie says that information hiding can lead to copying the same code in multiple places (duplicating code),[57] which goes against the don't repeat yourself rule of software development.[58] Polymorphism is the use of one symbol to represent multiple different types.[59] In object-oriented programming, polymorphism more specifically refers to subtyping or subtype polymorphism, where a function can work with a specific interface and thus manipulate entities of different classes in a uniform manner.[60] For example, imagine a program has two shapes: a circle and a square. Both come from a common class called "Shape." Each shape has its own way of drawing itself. With subtype polymorphism, the program doesn't need to know the type of each shape, and can simply call the "Draw" method for each shape. The programming language runtime will ensure the correct version of the "Draw" method runs for each shape. Because the details of each shape are handled inside their own classes, this makes the code simpler and more organized, enabling strong separation of concerns. In object-oriented programming, objects have methods that can change or use the object's data. Many programming languages use a special word, like this or self, to refer to the current object. In languages that support open recursion, a method in an object can call other methods in the same object, including itself, using this special word. This allows a method in one class to call another method defined later in a subclass, a feature known as late binding. This section does not cite any sources. Please help improve this section by adding citations to reliable sources. Unsourced material may be challenged and removed. (August 2009) (Learn how and when to remove this message) See also: List of object-oriented programming languages OOP languages can be grouped into different types based on how they support and use objects: Pure OOP languages: In these languages, everything is treated as an object, even basic things like numbers and characters. They are designed to fully support and enforce OOP. Examples: Ruby, Scala, Smalltalk, Eiffel, Emerald,[61] JADE, Self, Raku. Mostly OOP languages: These languages focus on OOP but also include some procedural programming features. Examples: Java, Python, C++, C#, Delphi/Object Pascal, VB.NET. Retrofitted OOP languages: These were originally designed for other types of programming but later added some OOP features. Examples: PHP, JavaScript, Perl, Visual Basic (derived from BASIC), MATLAB, COBOL 2002, Fortran 2003, ABAP, Ada 95, Pascal. Unique OOP languages: These languages have OOP features like classes and inheritance but use them in their own way. Examples: Oberon, BETA. Object-based languages: These support some OOP ideas but avoid traditional class-based inheritance in favor of direct manipulation of objects. Examples: JavaScript, Lua, Modula-2, CLU, Go. Multi-paradigm languages: These support both OOP and other programming styles, but OOP is not the predominant style in the language. Examples include Tcl, where TclOO allows both prototype-based and class-based OOP, and Common Lisp, with its Common Lisp Object System. The TIOBE programming language popularity index graph from 2002 to 2023. In the 2000s the object-oriented Java (orange) and the procedural C (dark blue) competed for the top position. Many popular programming languages, like C++, Java, and Python, use object-oriented programming. In the past, OOP was widely accepted,[62] but recently, some programmers have criticized it and prefer functional programming instead.[63] A study by Potok et al. found no major difference in productivity between OOP and other methods.[64] Paul Graham, a well-known computer scientist, believes big companies like OOP because it helps manage large teams of average programmers. He argues that OOP adds structure, making it harder for one person to make serious mistakes, but at the same time restrains smart programmers.[65] Eric S. Raymond, a Unix programmer and open-source software advocate, argues that OOP is not the best way to write programs.[56] Richard Feldman says that, while OOP features helped some languages stay organized, their popularity comes from other reasons.[66] Lawrence Krubner argues that OOP doesn't offer special advantages compared to other styles, like functional programming, and can make coding more complicated.[67] Luca Cardelli says that OOP is slower and takes longer to compile than procedural programming.[54] In recent years, object-oriented programming (OOP) has become very popular in dynamic programming languages. Some languages, like Python, PowerShell, Ruby and Groovy, were designed with OOP in mind. Others, like Perl, PHP, and ColdFusion, started as non-OOP languages but added OOP features later (starting with Perl 5, PHP 4, and ColdFusion version 6). On the web, HTML, XHTML, and XML documents use the Document Object Model (DOM), which works with the JavaScript language. JavaScript is a well-known example of a prototype-based language. Instead of using classes like other OOP languages, JavaScript creates new objects by copying (or "cloning") existing ones. Another language that uses this method is Lua. When computers communicate in a client-server system, they send messages to request services. For example, a simple message might include a length field (showing how big the message is), a code that identifies the type of message, and a data value. These messages can be designed as structured objects that both the client and server understand, so that each type of message corresponds to a class of objects in the client and server code. More complex messages might include structured objects as additional details. The client and server need to know how to serialize and deserialize these messages so they can be transmitted over the network, and map them to the appropriate object types. Both clients and servers can be thought of as complex object-oriented systems. The Distributed Data Management Architecture (DDM) uses this idea by organizing objects into four levels: Basic message details - Information like message length, type, and data. Objects and collections - Similar to how objects work in Smalltalk, storing messages and their details. Managers - Like file directories, these organize and store data, as well as provide memory and processing power. They are similar to IBM i Objects. Clients and servers - These are full systems that include managers and handle security, directory services, and multitasking. The first version of DDM defined distributed file services. Later, it was expanded to support databases through the Distributed Relational Database Architecture (DRDA). Design patterns are common solutions to problems in software design. Some design patterns are especially useful for object-oriented programming, and design patterns are typically introduced in an OOP context. The following are notable software design patterns for OOP objects.[68] Function object: Class with one main method that acts like an anonymous function (in C++, the function operator, operator()) Immutable object: does not change state after creation First-class object: can be used without restriction Container object: contains other objects Factory object: creates other objects Metaobject: Used to create other objects (similar to a class, but an object) Prototype object: a specialized metaobject that creates new objects by copying itself Singleton object: only instance of its class for the lifetime of the program Filter object: receives a stream of data as its input and transforms it into the object's output A common anti-pattern is the God object, an object that knows or does too much. Main article: Design pattern (computer science) Design Patterns: Elements of Reusable Object-Oriented Software is a famous book published in 1994 by four authors: Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. People often call them the "Gang of Four". The book talks about the strengths and weaknesses of object-oriented programming and explains 23 common ways to solve programming problems. These solutions, called "design patterns," are grouped into three types: Creational patterns (5): Factory method pattern, Abstract factory pattern, Singleton pattern, Builder pattern, Prototype pattern Structural patterns (7): Adapter pattern, Bridge pattern, Composite pattern, Decorator pattern, Facade pattern, Flyweight pattern, Proxy pattern Behavioral patterns (11): Chain-of-responsibility pattern, Command pattern, Interpreter pattern, Iterator pattern, Mediator pattern, Memento pattern, Observer pattern, State pattern, Strategy pattern, Template method pattern, Visitor pattern Main articles: Object-relational impedance mismatch, Object-relational mapping, and Object database Both object-oriented programming and relational database management systems (RDBMSs) are widely used in software today. However, relational databases don't store objects directly, which creates a challenge when using them together. This issue is called object-relational impedance mismatch. To solve this problem, developers use different methods, but none of them are perfect.[69] One of the most common solutions is object-relational mapping (ORM), which helps connect object-oriented programs to relational databases. Examples of ORM tools include Visual FoxPro, Java Data Objects, and Ruby on Rails ActiveRecord. Some databases, called object databases, are designed to work with object-oriented programming. However, they have not been as popular or successful as relational databases. Date and Darwen have proposed a theoretical foundation that uses OOP as a kind of customizable type system to support RDBMSs, but it forbids objects containing pointers to other objects.[70] In responsibility-driven design, classes are built around what they need to do and the information they share, in the form of a contract. This is different from data-driven design, where classes are built based on the data they need to store. According to Wirfs-Brock and Wilkerson, the originators of responsibility-driven design, responsibility-driven design is the better approach.[71] SOLID is a set of five rules for designing good software, created by Michael Feathers: Single responsibility principle: A class should have only one reason to change. Open/closed principle: Software entities should be open for extension, but closed for modification. Liskov substitution principle: Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it. Interface segregation principle: Clients should not be forced to depend upon interfaces that they do not use. Dependency inversion principle: Depend upon abstractions, not concretes. GRASP (General Responsibility Assignment Software Patterns) is another set of software design rules, created by Craig Larman, that helps developers assign responsibilities to different parts of a program:[72] Creator Principle: allows classes create objects they closely use. Information Expert Principle: assigns tasks to classes with the needed information. Low Coupling Principle: reduces class dependencies to improve flexibility and maintainability. High Cohesion Principle: designing classes with a single, focused responsibility. Controller Principle: assigns system operations to separate classes that manage flow and interactions. Polymorphism: allows different classes to be used through a common interface, promoting flexibility and reuse. Pure Fabrication Principle: create helper classes to improve design, boost cohesion, and reduce coupling. See also: Formal semantics of programming languages In object-oriented programming, objects are things that exist while a program is running. An object can represent anything, like a person, a place, a bank account, or a table of data. Many researchers have tried to formally define how OOP works. Records are the basis for understanding objects. They can represent fields, and also methods, if function literals can be stored. However, inheritance presents difficulties, particularly with the interactions between open recursion and encapsulated state. Researchers have used recursive types and co-algebraic data types to incorporate essential features of OOP.[73] Abadi and Cardelli defined several extensions of System F n % 2 == 0) .collect(Collectors.toList()); In this example, the stream takes the list of numbers, applies a filter to keep only the even ones, and then collects the result into a new list. Streams are particularly useful when I want to chain multiple operations, as they are designed for lazy evaluation—meaning operations are only performed when necessary. In Java, exceptions are categorized into two main types: checked and unchecked exceptions. Checked exceptions are exceptions that are checked at compile time. This means that when I write code that throws a checked exception, I must either handle it with a try-catch block or declare it with the throws keyword in the method signature. Examples of checked exceptions include IOException and SQLException . These exceptions typically represent conditions that are outside of the program’s control, such as file handling or network issues. Unchecked exceptions, on the other hand, are not checked at compile time and are typically the result of programming errors, such as logic flaws or incorrect data. These exceptions extend the RuntimeException class and include exceptions like NullPointerException and ArrayIndexOutOfBoundsException . Since they are unchecked, Java doesn’t force me to handle them, although it’s still a good practice to catch them when necessary. Unchecked exceptions usually indicate that something has gone wrong in the program logic that should be fixed by the developer. Read More: TCS Java Interview Questions Multithreading in Java is the process of executing multiple threads concurrently to maximize the CPU’s utilization. Each thread represents an independent path of execution, allowing me to perform multiple tasks simultaneously, which can improve the performance of my applications. For example, in a web server, multithreading allows the server to handle multiple client requests at the same time. Java makes it easy to implement multithreading using the Thread class or the Runnable interface, as well as higher-level concurrency APIs from the java.util.concurrent package. However, when multiple threads access shared resources or data, issues like data inconsistency and race conditions can arise. To manage this, I can use thread synchronization mechanisms in Java, such as the synchronized keyword or locks from the java.util.concurrent.locks package. By synchronizing critical sections of code, I ensure that only one thread can execute that section at a time, preventing conflicts and ensuring data consistency. Here’s an example of synchronized code: public synchronized void incrementCounter() { counter++; } In this example, the incrementCounter method is synchronized, which means that only one thread can modify the counter variable at a time. This prevents race conditions where multiple threads attempt to update counter simultaneously, which could lead to inconsistent results. Read More: Java Projects with Real-World Applications Java manages memory through a combination of automatic memory allocation and garbage collection. The memory in Java is divided into two main areas: the heap and the stack. The heap is where objects are stored, while the stack holds method calls and local variables. When I create an object, it’s allocated memory on the heap, and once it’s no longer referenced by any part of the program, the garbage collector removes it. This process prevents manual memory management, which is a common source of bugs in other programming languages. Despite Java’s efficient memory management system, memory leaks can still occur if objects are unintentionally retained in memory due to lingering references. A memory leak happens when an object that is no longer needed remains in memory because some part of the code still holds a reference to it. This can cause the heap to fill up, leading to performance degradation or even an OutOfMemoryError . To avoid memory leaks, I ensure that unused references are cleared and use tools like profilers and the Eclipse Memory Analyzer to track down issues in large applications. import java.util.ArrayList; import java.util.List; public class MemoryLeakExample { private List list = new ArrayList(); public void addToList(String item) { list.add(item); // Adding items to the list } public static void main(String[] args) { MemoryLeakExample example = new MemoryLeakExample(); // Simulating memory leak by adding items in a loop for (int i = 0; i < 100000; i++) { example.addToList("Item " + i); } // Intentionally not clearing the list to retain references // This can lead to a memory leak } } In this snippet, the MemoryLeakExample class maintains a list that continuously adds strings. By not clearing the list after usage, the program retains references to many strings, leading to a memory leak. If this pattern continues in a long-running application, it can exhaust available memory. Java 8 introduced several powerful features that significantly improved the language’s performance and ease of use. One of the most notable features is lambda expressions, which allow me to write more concise and readable code by treating functions as first-class objects. Lambdas are particularly useful in reducing boilerplate code, especially when working with collections and functional interfaces. Additionally, the Streams API was introduced in Java 8, enabling functional-style operations on collections, such as filtering, mapping, and reducing, to be performed efficiently. Another key feature introduced in Java 8 is the Optional class, which helps in handling null values more safely. Instead of dealing with NullPointerException , I can use Optional to represent the possibility of a value being present or absent. Java 8 also introduced default and static methods in interfaces, allowing me to provide default behavior without breaking existing implementations. These features, combined with enhancements like the new Date and Time API , improve code readability and maintainability, making Java 8 a significant milestone in the evolution of the language. import java.util.Arrays; import java.util.List; public class Java8FeaturesExample { public static void main(String[] args) { List names = Arrays.asList("Alice", "Bob", "Charlie", "David"); // Using lambda expression to filter and print names that start with 'C' names.stream() .filter(name -> name.startsWith("C")) .forEach(System.out::println); } } In this example, a list of names is created. The stream() method is called to process the list as a stream of data. The filter method uses a lambda expression to select names that start with “C”. Finally, forEach is used to print the filtered names. This concise syntax improves code readability and allows for functional-style operations, demonstrating the power of Java 8’s features. Read More: Object-Oriented Programming Java The Java memory model defines how the JVM manages memory and how threads interact with it. Java divides memory into two main areas: heap and stack memory. Heap memory is used to store objects, and it’s shared among all threads in an application. Objects created using the new keyword are stored on the heap, and garbage collection helps in freeing up memory occupied by objects that are no longer in use. In contrast, stack memory is used for thread-specific data, such as method calls, local variables, and references to objects. One key difference between heap and stack memory is scope and lifetime. Stack memory is much faster and is automatically managed by the JVM, but it’s limited in size and tied to the lifecycle of a method. When a method completes, the stack frame for that method is popped off the stack, and the memory is freed. Heap memory, on the other hand, is larger and more flexible but slower because it’s managed through garbage collection. Objects stored in the heap persist until they are no longer referenced, making heap memory suitable for objects that need to exist beyond a method’s execution. public class MemoryModelExample { public static void main(String[] args) { int localVariable = 10; // Stored in stack memory MyObject obj = new MyObject(); // Object reference in stack, object in heap obj.display(localVariable); // Method call } } class MyObject { public void display(int value) { System.out.println("Value: " + value); // 'value' is a local variable in stack } } In this example, the MemoryModelExample class demonstrates both stack and heap memory usage. The variable localVariable is stored in the stack as a local variable within the main method. The object obj is created using the new keyword, so the reference to it is stored in the stack, while the actual MyObject instance is allocated in the heap. The display method is called, where the parameter value is also stored in the stack. This snippet illustrates how Java manages memory allocation for local variables and objects within its memory model. Read More: Java and Cloud Integration The Producer-Consumer problem is a classic synchronization problem in multithreaded programming where one or more producers generate data and add it to a shared buffer, while one or more consumers retrieve and process the data from the buffer. The challenge lies in ensuring that the producers and consumers don’t access the buffer at the same time, which could lead to data inconsistency. In Java, I can solve this problem by using synchronization techniques like wait() , notify() , and notifyAll() to coordinate access between producers and consumers. A more modern and efficient way to solve the Producer-Consumer problem is by using the BlockingQueue class from the java.util.concurrent package. The BlockingQueue handles synchronization for me and ensures that producers wait when the buffer is full, and consumers wait when the buffer is empty. Here’s a simple example: BlockingQueue queue = new ArrayBlockingQueue(10); Runnable producer = () -> { try { queue.put(1); } catch (InterruptedException e) { Thread.currentThread().interrupt(); } }; Runnable consumer = () -> { try { queue.take(); } catch (InterruptedException e) { Thread.currentThread().interrupt(); } }; new Thread(producer).start(); new Thread(consumer).start(); In this example, the ArrayBlockingQueue ensures thread-safe access to the shared buffer, and the put() and take() methods block the threads until the buffer has space or an item to consume, respectively. Read more: Accenture Java interview Questions and Answers In Java, the volatile keyword is used to indicate that a variable’s value can be modified by multiple threads. When a variable is declared as volatile , any changes made to it by one thread are immediately visible to all other threads. This prevents issues related to caching, where one thread may read a stale value from a local cache instead of fetching the most recent value from main memory. The volatile keyword ensures that updates to the variable are always written to and read from main memory, maintaining consistency across threads. However, it’s important to note that volatile does not provide atomicity or mutual exclusion. While it ensures visibility, it doesn’t protect against race conditions when multiple threads are updating the variable simultaneously. For example, a simple increment operation x++ is not atomic and could lead to incorrect results when performed by multiple threads. In such cases, I would need to use synchronization techniques like synchronized blocks or locks to ensure thread safety. The volatile keyword is best suited for cases where multiple threads read and write to a single variable without complex operations or dependencies. class VolatileExample { private volatile boolean running = true; public void run() { System.out.println("Thread started."); while (running) { // Simulate some work } System.out.println("Thread stopped."); } public void stop() { running = false; // Set the flag to stop the thread } public static void main(String[] args) throws InterruptedException { VolatileExample example = new VolatileExample(); Thread thread = new Thread(example::run); thread.start(); // Let the thread run for a moment Thread.sleep(1000); example.stop(); // Signal the thread to stop thread.join(); // Wait for the thread to finish } } In this example, the VolatileExample class uses a volatile boolean variable running to control the execution of a thread. The run method starts a loop that continues as long as running is true . The stop method sets running to false . The main method starts the thread, sleeps for a second, and then calls stop , signaling the thread to terminate. Because running is declared as volatile , any changes made by one thread (in this case, setting running to false) are immediately visible to the other thread, ensuring proper termination of the loop. Read More: Java Interview Questions for Freshers Part 1 Preparing for a Java interview with five years of experience is important. You should be ready to answer questions about advanced Java concepts. Topics like object-oriented programming, design patterns, and data structures are key. Understanding frameworks like Spring and Hibernate is also crucial. Practicing coding problems and discussing real-life projects can help you shine. Being confident and clear in your answers makes a big difference. Remember to show your passion for Java and your problem-solving skills. Good luck! Premium Read: Access my best content on Medium member-only articles — deep dives into Java, Spring Boot, Microservices, backend architecture, interview preparation, career advice, and industry-standard best practices. Some premium posts are free to read — no account needed. Follow me on Medium to stay updated and support my writing. Top 10 Udemy Courses (Huge Discount): Explore My Udemy Courses — Learn through real-time, project-based development. ▶ Subscribe to My YouTube Channel (172K+ subscribers): Java Guides on YouTube Are you preparing for a Java interview and have around 5 years of experience in the field? Here is a list of 50 questions to help you brush up on your knowledge and impress your interviewers. Answer: Component Description Purpose JDK (Java Development Kit) Includes JRE, a compiler (javac), an archiver (jar), and other tools needed for Java development. Used to develop and compile Java applications. JRE (Java Runtime Environment) Includes the JVM and standard libraries needed to run Java applications. Provides the necessary environment to run Java applications. JVM (Java Virtual Machine) An abstract machine that converts Java bytecode into machine code. Executes Java bytecode and provides a runtime environment for Java applications. 2. Explain the concept of Object-Oriented Programming (OOP) in Java. Answer: OOP is a programming paradigm based on the concept of "objects", which can contain data and code. The four main principles of OOP are: Encapsulation: Wrapping data and methods into a single unit. Inheritance: Mechanism where one class acquires the properties and behaviors of a parent class. Polymorphism: The ability of an object to take on many forms, typically through method overriding and overloading. Abstraction: Hiding the complex implementation details and showing only the essential features of the object. 3. What are Java Generics? Give an example. Answer: Generics enable types (classes and interfaces) to be parameters when defining classes, interfaces, and methods. It allows for code reusability and type safety. public class Box { private T t; public void set(T t) { this.t = t; } public T get() { return t; } } 4. Explain the difference between ‘==’ and ‘equals()’ method in Java. Answer: ‘==’ operator: Compares the reference of the objects. ‘equals()’ method: Compares the content of the objects. 5. What is a Java String Pool? Answer: Java String Pool is a special memory region where Java stores literal string values. When a new string is created using a literal, the JVM checks the string pool first. If the string already exists, the reference is returned; otherwise, a new string is created in the pool. 6. How does Java handle memory management? Answer: Java uses an automatic memory management system called garbage collection. The garbage collector automatically deletes unused objects to free up memory, thus preventing memory leaks. 7. Explain the concept of Multithreading in Java. Answer: Multithreading in Java is the process of executing multiple threads simultaneously. Java provides built-in support for multithreaded programming. It allows performing multiple operations independently in parallel, improving application performance. 8. What is the difference between ArrayList and LinkedList in Java? Answer: ArrayList: Resizable array implementation of the List interface. It's better for storing and accessing data. LinkedList: Doubly-linked list implementation of the List and Deque interfaces. It's better for manipulating data, like adding or removing elements. 9. Describe the use of the transient keyword. Answer: The transient keyword in Java is used to indicate that a field should not be serialized. When an object is serialized, the fields marked as transient are ignored and not included in the serialized representation. 10. Explain the concept of Exception Handling in Java. Answer: Exception Handling in Java is a powerful mechanism for handling runtime errors and maintaining the normal flow of the application. It uses try, catch, finally, and throw statements to handle exceptions gracefully. 11. What is the final keyword in Java? Answer: final variable: Its value cannot be changed once assigned. final method: Cannot be overridden by subclasses. final class: Cannot be subclassed. 12. Explain the difference between Checked and Unchecked Exceptions. Answer: Checked Exceptions: Exceptions that are checked at compile-time (e.g., IOException). Unchecked Exceptions: Exceptions that are not checked at compile-time (e.g., ArithmeticException, NullPointerException). 13. What is the use of the synchronized keyword? Answer: The synchronized keyword in Java is used to control the

access of multiple threads to any shared resource. Synchronization ensures that only one thread can access the resource at a time. 14. How does the hashCode() method work in Java? Answer: The hashCode() method returns an integer value generated by a hashing algorithm. It is used in hashing-based collections like HashMap, HashSet, and Hashtable to determine the bucket location for storing objects. 15. What are Java Annotations? Answer: Annotations provide metadata about the code. They are used to provide additional information to the compiler and are not part of the program logic. Examples include @Override, @Deprecated, and @SuppressWarnings. 16. What is a Singleton class in Java? Answer: A Singleton class in Java ensures that only one instance of the class is created. It provides a global point of access to the instance. It is typically implemented using a private constructor and a static method. 17. Explain the difference between wait(), notify(), and notifyAll() methods. Answer: wait(): Causes the current thread to wait until another thread invokes notify() or notifyAll() on the same object. notify(): Wakes up a single thread that is waiting on the object's monitor. notifyAll(): Wakes up all the threads that are waiting on the object's monitor. 18. What is Java Reflection API? Answer: Java Reflection API allows the inspection and modification of the runtime behavior of applications. It can be used to inspect classes, interfaces, fields, and methods at runtime, even if they are not accessible during compile time. 19. What are Java Streams? Answer: Java Streams are a new abstraction introduced in Java 8. They allow functional-style operations on collections of elements, such as map-reduce transformations. Streams can be sequential or parallel. 20. What is a volatile keyword in Java? Answer: The volatile keyword in Java is used to mark a variable as "stored in main memory." Every read of a volatile variable will be read from the computer's main memory and not from the CPU cache. 21. Explain the concept of Dependency Injection. Answer: Dependency Injection is a design pattern used to implement IoC (Inversion of Control). It allows the creation of dependent objects outside of a class and provides those objects to a class through different ways (e.g., constructor injection, setter injection). 22. What is the difference between StringBuilder and StringBuffer? Answer: StringBuilder: Non-synchronized, faster, and used in a single-threaded environment. StringBuffer: Synchronized, slower, and used in a multi-threaded environment. 23. Explain the purpose of the default keyword in Java 8 interfaces. Answer: The default keyword in Java 8 allows the creation of default methods in interfaces. These methods can have a body and provide a default implementation that can be overridden by implementing classes. 24. What is the Fork/Join framework in Java? Answer: The Fork/Join framework is designed for parallelism. It allows breaking a task into smaller tasks (forking) and then joining the results of the subtasks. It is used to exploit the multiple processors available in a system. 25. How does Java handle memory leaks? Answer: Java handles memory leaks using garbage collection, which automatically removes unused objects from memory. However, poorly written code (e.g., holding onto references longer than necessary) can still cause memory leaks. 26. Explain the difference between Callable and Runnable interfaces. Answer: Runnable: Represents a task that can be executed by a thread. It does not return any result and cannot throw checked exceptions. Callable: Similar to Runnable but can return a result and throw a checked exception. It is part of the java.util.concurrent package. 27. What is the Future interface in Java? Answer: The Future interface represents the result of an asynchronous computation. Methods provided include isDone(), get(), cancel(), and isCancelled(). It is used in conjunction with Callable to get the result of an asynchronous task. 28. Explain the Java Memory Model (JMM). Answer: JMM defines how threads interact through memory and what behaviors are legal in concurrent executions. It specifies the visibility of variables across threads and ordering of reads and writes to variables. 29. What are Java Atomic classes? Answer: Atomic classes (like AtomicInteger, AtomicLong, AtomicReference) provide a way of updating variables atomically without using synchronization. They are part of the java.util.concurrent.atomic package and use low-level concurrency primitives. 30. How does the volatile keyword differ from synchronized? Answer: volatile: Ensures visibility of changes to variables across threads but does not provide atomicity or mutual exclusion. synchronized: Provides both mutual exclusion and visibility, ensuring that only one thread can access a block of code or method at a time. 31. Explain what a ThreadLocal variable is. Answer: ThreadLocal provides thread-local variables. Each thread accessing such a variable has its own independent copy of the variable. It is useful for maintaining per-thread context. 32. What is the use of Phaser in Java? Answer: Phaser is a more flexible and reusable synchronization barrier that supports adjustable phases, where threads can wait for others to reach a common barrier point. It's part of java.util.concurrent. 33. How does Java implement polymorphism? Answer: Polymorphism in Java is implemented through method overriding (runtime polymorphism) and method overloading (compile-time polymorphism). It allows objects to be treated as instances of their parent class or interface. 34. Explain Java's Stream API and its benefits. Answer: The Stream API, introduced in Java 8, allows for functional-style operations on streams of elements. Benefits include cleaner and more readable code, ease of parallel processing, and powerful operations like map, filter, and reduce. 35. What is the difference between HashMap and ConcurrentHashMap? Answer: HashMap: Not thread-safe and can be used in a single-threaded environment. ConcurrentHashMap: Thread-safe and allows concurrent read and write operations. It divides the map into segments to reduce contention. 36. How does the CompletableFuture class enhance concurrency in Java? Answer: CompletableFuture in Java 8 enhances concurrency by providing a way to write non-blocking, asynchronous code. It supports combinatory operations like thenApply(), thenAccept(), and thenCombine() for chaining multiple async tasks. 37. What is the difference between yield(), sleep(), and wait()? Answer: yield(): Hints the thread scheduler to give other threads of the same priority a chance to run. sleep(): Pauses the thread execution for a specified period. wait(): Causes the current thread to wait until another thread invokes notify() or notifyAll() on the same object. 38. Explain the concept of ReentrantLock. Answer: ReentrantLock is a lock implementation that allows the same thread to acquire the lock multiple times. It provides more flexibility than synchronized blocks, including timed lock waits, and interruptible lock acquisition. 39. What is the ForkJoinPool in Java? Answer: ForkJoinPool is a specialized implementation of the ExecutorService that supports the creation and processing of tasks using the fork/join paradigm, where tasks can be recursively split into smaller sub-tasks. 40. Explain the purpose of the CountDownLatch. Answer: CountDownLatch is a synchronization aid that allows one or more threads to wait until a set of operations being performed in other threads completes. It works by having a counter that threads decrement and wait on. 41. What are SoftReference, WeakReference, and PhantomReference? Answer: SoftReference: References that are cleared at the discretion of the garbage collector in response to memory demand. WeakReference: References that are cleared as soon as the garbage collector detects they are only weakly reachable. PhantomReference: References that are used to track the garbage collection process. They are enqueued after the collector determines that an object is phantom reachable. 42. Describe the Optional class introduced in Java 8. Answer: Optional is a container object which may or may not contain a non-null value. It provides methods to check for the presence of a value, return a value if present, or return a default value otherwise. It helps avoid null checks and NullPointerException. 43. How does try-with-resources statement work in Java? Answer: The try-with-resources statement ensures that each resource declared within it is closed at the end of the statement. It works with any object that implements the AutoCloseable interface. 44. What is the role of the Default method in interfaces? Answer: Default methods, introduced in Java 8, allow methods to have a body in interfaces. They enable interfaces to evolve by adding new methods without breaking existing implementations. 45. Explain the concept of Functional Interfaces in Java. Answer: A functional interface is an interface with exactly one abstract method. They can be implemented using lambda expressions, method references, or constructor references. Examples include Runnable, Callable, and custom interfaces annotated with @FunctionalInterface. 46. What is the purpose of the Collectors class in Java Streams? Answer: Collectors provides a series of utility methods for accumulating elements of streams into collections, summarizing statistics, and concatenating strings. It supports common operations like toList(), toSet(), groupingBy(), and partitioningBy(). 47. Explain the concept of immutable classes in Java. Answer: Immutable classes are classes whose instances cannot be modified after creation. All fields are final and private, and no setters are provided. Examples include String and wrapper classes like Integer and Double. 48. What is the difference between wait() and sleep()? Answer: wait(): Releases the lock held by the thread and puts the thread into waiting state until another thread invokes notify() or notifyAll() on the same object. sleep(): Puts the thread into a sleeping state for a specified duration without releasing the lock. 49. Explain what a java.util.concurrent.Executor is. Answer: Executor is an interface that represents an object which executes submitted Runnable tasks. It provides a way to decouple task submission from the details of how each task will be run, including thread use and scheduling. 50. How does Deadlock occur and how can it be avoided? Answer: Deadlock occurs when two or more threads are blocked forever, waiting for each other to release resources. It can be avoided by following practices like acquiring locks in a consistent order, using timeout for lock acquisition, and avoiding unnecessary locks. Conclusion These 50 questions cover a wide range of topics and concepts that are essential for a Java developer with 5 years of experience. Make sure to understand the underlying principles and be ready to provide examples or elaborate further if needed. Good luck with your interview preparation! 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. image_credit — EducativeHello guys, if you are preparing for Java interviews and need some questions to revise Object Oriented concepts then you have come to the right place. In the past, I have shared the best OOP books and courses as well multiple Java interview questions and in this article, I am going to share 50+ object-oriented programming concept questions for you.OOP Interview question or Object-oriented programming interview question is an integral part of any Java interview. Since Java is an Object-oriented programming language, it’s expected from Java developers that he is good in Object-oriented analysis and design and familiar with essential OOP concepts like Abstraction, Encapsulation, and Polymorphism.OOP, Interview question in Java is mainly based around fundamental OOPS concepts, and How those are implemented in Java, like Abstraction OOP concept is implemented using an interface and abstract class, Encapsulation is using private keyword, etc.Question from OOP is also asked as part of the Java design pattern question on the Senior level Java interview. On Freshers and Beginners level interview, OOP Interview Questions are mostly based on fundamentals only with some tricky Java questions like Why Java doesn’t support multiple inheritances, etc.In this Java article, we will see some frequently asked OOP interview questions and answers.Some question is very fundamental, and some Object-oriented programming question is difficult to answer, but this mix helps to learn more. If you love to learn more about the OOPS design principle, read 10 OOP design principles for Java programmers.without wasting any more of your time, here is my list of frequently asked Object-Oriented Programming questions for Java programmers. These questions have been collected from many Java interviews, both telephonic and face-bot-face from different levels of Java programmers.What is Class in Object-oriented programming? (answer)A class is a blueprint to create objects.What is Object in OOP? (answer)Created from classes, represent a particular state of the class.What is the abstraction in Java? (answer)An OOP technique to hide complexities from clients.What is Inheritance in Java? (answer)An object-oriented technique to reuse code and functionalities.What is Encapsulation or data hiding in Java? (answer)An oop way to hide data so that you can change it later without impacting others.What is Polymorphism in Java or OOP? (answer)provides flexibility to choose different codes to run at runtime.What is the difference between Polymorphism, Overloading, and Overriding? (answer)Why Java doesn’t support Multiple Inheritance in Java? (answer)When do you use interface and abstract class in Java?(Answer)What is the difference between static and dynamic binding in Java? (answer)What is the difference between abstraction and polymorphism in Java?** (answer)What is the difference between the IS-A relationship and HAS-A? (answer)IS-A represents Inheritance and HAS-A represents composition.Why Java doesn’t support operator overloading? (answer)What is an abstract class in Java? (answer)What is an interface in Java? (answer)What is the difference between Inheritance and Composition? (answer)What is the difference between coupling and cohesion? (answer)coupling is the dependency between different parts of code while cohesion is about the same part of code.What is a constructor in Java? (answer)A special construct that creates objects.What is the difference between Class and Object in Java or OOP? (answer)What is the difference between Inheritance and Polymorphism in Java? (answer)What are SOLID Design Principles? Explain any three of them (answer)What is the difference between Factory and Abstract Factory design patterns? (answer)What is the difference between Overloading, hiding, shadowing and Overriding in OOP? (answer)What is the difference between instance and object in Java? (answer)What is the difference between static and dynamic binding in Java? (answer)What are SOLID Object-oriented principles?** (answer)Difference between Abstract Class and Interface in Java? (answer)Difference between private, protected, and public modifiers in Java? (answer)What is constructor chaining? (answer)Difference between pass by value and pass by reference? (answer)Difference between abstraction and encapsulation? (answer)Difference between association, composition, and aggregation? (answer)Can you explain Open Closed Design Principle? (answer)What is an Observer design pattern? When should you use it? (answer)What is the difference between hiding and shadowing in OOP?** (answer)Can you override a static method in Java? (answer)What is the difference between state and strategy design patterns? (answer)What is the difference between a class and an instance? (answer)What is the difference between Factory and Abstract Factory design patterns? (answer)What is the difference between Dependency injection and Factory Pattern? (answer)What is method overloading in OOP or Java? (answer)What is the method overriding in OOP or Java?** (answer)Is Java a pure object-oriented language? if not why? (answer)What are the rules of method overloading and overriding in Java?** (answer)The difference between method overloading and overriding? (answer)Can we overload a static method in Java? (answer)Can we override the static method in Java?** (answer)Can we override a private method in Java?** (answer)What is the covariant method overriding in Java?** (answer)Can we change the argument list of an overriding method?** (answer)Can we override a method that throws runtime exception without throws clause? (answer)Can we override the final method in Java? (answer)What is the default method of Java 8?** (answer)What is an abstract class in Java? (answer)**What is an interface in Java? What is the real use of an interface?** (answer)What are the differences between Abstract class and interface?** (answer)Can we make a class abstract without an abstract method?** (answer)Can we make a class both final and abstract at the same time?** (answer)Can we overload or override the main method in Java?** (answer)What problem is solved by the Strategy pattern in Java?** (answer)Which OOP concept Decorator design Pattern is based upon?** (answer)When to use the Singleton design pattern in Java?** (answer)What is the difference between State and Strategy Patterns?** (answer)What is the difference between Association, Aggregation, and Composition in OOP? (answer)What is the difference between Decorator, Proxy, and Adapter patterns in Java? (answer)What is the difference between Composition and Inheritance in OOP?(answer)That’s all about some of the frequently asked OOP Interview Questions for Java Programmers. Most likely you already know answers to these fundamental Object-oriented programming, design, and pattern-related questions.If you don’t see the links and revise those concepts. You can use this list to revise all key object oriented programming concepts before your interview. If you think a popular OOP question is missing from the list, feel free to share in the comments and I will include it in this list.Most of these questions are also useful for JavaScript and Python developers because both JavaScript and Python are also object-oriented programming languages.Other Interview Questions you may like to Prepare35 Python Interview Questions for Beginners (python questions)50+ SQL and Database Phone Interview questions (SQL questions)130+ Java Interview Questions with Answers (list)17 Spring AOP Interview Questions with Answers (list)40+ Object-Oriented Interview Questions with Answers (questions)15 Spring Data JPA Interview Questions (list)20+ JUnit Interview Questions for Java developers (questions)10 Dynamic Programming Problems for Coding interviews (questions)25 Spring Security Interview Questions with Answers (questions)5 Best Courses to learn Java Programming (best courses)20 Spring MVC Interview Questions with answers (spring questions)Thanks for reading this article so far. If you find these Object-Oriented Programming interview questions useful then please share them with your friends and colleagues. If you have any questions or feedback then please drop a note.P. S. — If you are new to object oriented programming and need some resources to learn OOP then you can also check-out these free object oriented programming courses to start with. 5.3K Do you need help preparing for your Java OOPS interview? Object-oriented programming (OOP) is a core aspect of Java, and a strong understanding of its principles is essential. This guide covers the top 50 Java OOPS interview questions and answers, addressing both fundamental and advanced topics. With clear explanations and practical examples, these questions will help you approach technical rounds with confidence. Fun Fact: According to the TIOBE Index (2025), Java remains a top 5 programming language, with over 9 million developers using it. A major reason for its popularity is its strong OOPS foundation. Here is a list of basic Java Object Oriented Programming questions and answers for interviews: What are the four main principles of Object-Oriented Programming? The four main principles of OOPS are: Encapsulation – Wrapping data and methods into a single unit (class) to restrict direct access to data. Abstraction – Hiding implementation details and exposing only necessary functionalities using abstract classes or interfaces. Inheritance – Allowing one class (child) to inherit properties and methods from another (parent) to promote reusability. Polymorphism – Allowing a single method or operator to have multiple implementations (method overloading and method overriding). How is abstraction different from encapsulation in Java? Abstraction hides unnecessary details and exposes only the essential parts. It is implemented using abstract classes and interfaces. Encapsulation, on the other hand, restricts direct access to an object’s data by using access modifiers like private, protected, and public. While abstraction is about hiding implementation, encapsulation is about data security and integrity. What is method overloading and method overriding? Provide examples. Method Overloading: When multiple methods in the same class have the same name but different parameter lists. Example: class MathOperations {   int add(int a, int b) { return a + b; }   double add(double a, double b) { return a + b; } } Method Overriding: When a subclass provides a specific implementation of a method already defined in its superclass. Example: class Parent {   void display() { System.out.println(“Parent class method”); } } class Child extends Parent {   @Override   void display() { System.out.println(“Child class method”); } } Why is multiple inheritance not supported in Java? Multiple inheritance is not supported in Java to avoid ambiguity issues caused by the diamond problem. If two parent classes have the same method, the compiler cannot determine which one to inherit. Instead, Java provides interfaces, allowing a class to implement multiple interfaces without ambiguity. Here are some common Java Object Oriented Programming interview questions for freshers: What is the difference between a class and an object? A class is a blueprint for creating objects. It defines attributes (variables) and behaviors (methods). An object is an instance of a class with specific values assigned to its attributes. Example: class Car {   String brand;   void drive() { System.out.println(“Car is driving”); } } Car myCar = new Car(); // Object creation What is the significance of the ‘this’ keyword in Java? See also "What Are Your Hobbies" Interview Question with Sample AnswersThe this keyword refers to the current instance of a class. It is used to: Differentiate instance variables from local variables when they have the same name. Call another constructor in the same class. Pass the current instance as a parameter. Example: class Employee {   String name;   Employee(String name) { this.name = name; } } How does Java achieve runtime polymorphism? Java achieves runtime polymorphism through method overriding. The overridden method in a subclass is called at runtime based on the object type, even when referenced by a parent class. Example: class Animal {   void sound() { System.out.println(“Animal makes a sound”); } } class Dog extends Animal {   void sound() { System.out.println(“Dog barks”); } } Animal obj = new Dog(); obj.sound(); // Outputs: Dog barks What is the purpose of the ‘super’ keyword in Java? The super keyword is used to refer to the immediate parent class. It can be used to: Call the parent class constructor. Access parent class methods. Access parent class variables. Example: class Parent {   void display() { System.out.println(“Parent method”); } } class Child extends Parent {   void show() { super.display(); } } Let’s go through important Java OOPS programming interview questions and answers for experienced candidates: How does Java manage memory with respect to objects? Java uses automatic memory management with the help of the Garbage Collector (GC). When an object is no longer referenced, the GC removes it to free up memory. Java memory consists of: Heap (stores objects). Stack (stores method calls and local variables). Method area (stores class structures). What are the different types of constructors in Java? Default Constructor – No parameters, initializes objects with default values. Parameterized Constructor – Takes arguments to initialize instance variables. Copy Constructor – Copies values from one object to another. Example: class Student {   String name;   Student(String name) { this.name = name; } } How is object cloning implemented in Java? Java supports shallow cloning using the clone() method from the Cloneable interface. Example: class Employee implements Cloneable {   String name;   Employee(String name) { this.name = name; }   protected Object clone() throws CloneNotSupportedException {   return super.clone();   } } Deep cloning requires manual copying of referenced objects. What is a shallow copy and deep copy in Java? Shallow Copy: Copies field values but does not create new referenced objects. Changes in the original object affect the copied object. Deep Copy: Creates a new copy of referenced objects, making them independent. Example of deep copy: class Address {   String city;   Address(String city) { this.city = city; } } class Person {   String name;   Address address;   Person(String name, Address address) {   this.name = name;   this.address = new Address(address.city); // Deep Copy   } } If you have 2 years of experience, you might come across such Java and OOPS interview questions: Why did you choose Java for your career? Describe a situation where you had to debug a complex object-oriented issue. How did you solve it? If you had to redesign an existing system to improve reusability, which OOPS principles would you focus on and why? These interview questions for OOPS in Java are for candidates with three years of experience: What is the most challenging Java project you have worked on? How do you handle a situation where your team disagrees on the best OOPS approach for a project? If you need to implement a flexible payment system, which OOPS concepts would you apply and how? See also Top 70+ Selenium Interview Questions and Answers These Java Object Oriented interview questions are for candidates with 5 years of experience: What design patterns have you used in your Java projects, and why? How do you mentor junior developers on OOPS principles? You need to refactor a monolithic Java application into a microservices-based architecture. How would you approach the OOPS design? If you are at a senior level and have around 10 years of experience, you might come across such Java OOPS interview questions: How has your understanding of OOPS evolved over the years? Have you ever had to optimize an object-oriented Java system for performance? How did you do it? Given an existing Java application with tight coupling, how would you refactor it to follow SOLID principles? You might also come across OOPS concepts in Java interview questions like these: What is the difference between an interface and an abstract class? An interface defines a contract that classes must follow. It contains only abstract methods (before Java 8) and allows default and static methods (from Java 8). Interfaces support multiple inheritance since a class can implement multiple interfaces. An abstract class can have both abstract and concrete methods. It can include constructors and instance variables. Unlike interfaces, abstract classes can have method implementations but do not support multiple inheritance. How does Java implement multiple inheritance? Java does not support multiple inheritance through classes to avoid ambiguity (diamond problem). Instead, it uses interfaces. A class can implement multiple interfaces, allowing it to inherit behaviors from different sources without conflicts. What is dynamic method dispatch in Java? Dynamic method dispatch, also called runtime polymorphism, is the process where method calls are resolved at runtime based on the object’s actual type, not the reference type. These are some core Java OOP questions and answers for interviews: What is an association, aggregation, and composition in Java? Association: A relationship between two classes where both objects exist independently (e.g., Student and Teacher). Aggregation: A weaker relationship where the child can exist independently, but the parent owns it (e.g., Department and Employee). Composition: A strong relationship where the child object’s existence depends on the parent (e.g., Car and Engine). What is the role of access modifiers in OOPS? Access modifiers control visibility of class members: private – Accessible only within the class. default – Accessible within the same package. protected – Accessible in the same package and subclasses. public – Accessible from anywhere. Let’s go through some advanced Java Object Oriented interview questions and answers: What are the different types of design patterns in Java? Creational Patterns – Singleton, Factory, Builder. Structural Patterns – Adapter, Composite, Proxy. Behavioral Patterns – Strategy, Observer, Command. How do lambda expressions fit into Java’s object-oriented model? Lambda expressions provide a concise way to implement functional interfaces (interfaces with a single abstract method). They allow inline implementations without creating a separate class. Example: interface Calculator {   int operate(int a, int b); } Calculator add = (a, b) -> a + b; System.out.println(add.operate(5, 3)); // Outputs: 8 What is the function of reflection in Java OOPS? Reflection allows introspection and manipulation of classes, methods, and fields at runtime. It is useful in frameworks, serialization, and dependency injection. See also Top 30+ Pega Interview Questions and Answers for 2025Example: Class obj = Class.forName(“java.util.ArrayList”); System.out.println(obj.getMethods()); Also Read - Top 25+ Python OOPs Interview Question (2025) You should also take a look at these OOPS in JavaScript interview questions: How is object-oriented programming implemented in JavaScript? JavaScript is prototype-based, meaning objects inherit from other objects instead of classes. Objects can be created using constructors, prototypes, or ES6 classes. What is prototypal inheritance, and how does it differ from classical inheritance? Prototypal inheritance allows objects to inherit properties directly from another object using the prototype chain. Unlike classical inheritance, which relies on class hierarchies, JavaScript objects inherit dynamically. Example: let parent = { greet: function() { console.log(“Hello”); } }; let child = Object.create(parent); child.greet(); // Outputs: Hello Also Read - Top 20 PHP OOPs Interview Questions and Answers Here are some coding Java object oriented interview questions: Write a Java program to demonstrate method overriding. class Parent {   void show() { System.out.println(“Parent class method”); } } class Child extends Parent {   @Override   void show() { System.out.println(“Child class method”); } } public class Test {   public static void main(String[] args) {   Parent obj = new Child();   obj.show(); // Outputs: Child class method   } } Implement a singleton class in Java. class Singleton {   private static Singleton instance;   private Singleton() {} // Private constructor   public static Singleton getInstance() {   if (instance == null) {   instance = new Singleton();   }   return instance;   } } Create an interface and implement it in multiple classes with different behaviors. interface Animal {   void sound(); } class Dog implements Animal {   public void sound() { System.out.println(“Dog barks”); } } class Cat implements Animal {   public void sound() { System.out.println(“Cat meows”); } } public class Test {   public static void main(String[] args) {   Animal a1 = new Dog();   Animal a2 = new Cat();   a1.sound();   a2.sound();   } } Also Read - Top 20 C++ OOPs Interview Questions and Answers Implement a real-world example of polymorphism in Java. Create a Java class that follows the principle of encapsulation. Write a Java program to demonstrate the Factory Design Pattern. Implement an abstract class with a concrete method and abstract methods. Also Read - Top 30+ C# OOPs Interview Questions and Answers What is the difference between instance and static methods? How does Java handle object destruction? Can a constructor be private? If yes, when would you use it? What is the difference between early binding and late binding in Java? Also Read - Top 20 OOPs ABAP Interview Questions and Answers Here are some common interview questions for OOPS in Java in MCQ form: Which of the following is not an OOPS principle? a) Encapsulationb) Inheritancec) Compilationd) Polymorphism Object-oriented programming, or OOPs, is a programming paradigm that implements the concept of objects in the program. It aims to provide an easier solution to real-world problems by implementing real-world entities such as inheritance, abstraction, polymorphism, etc. in programming. OOPs concept is widely used in many popular languages like Java, Python, C++, etc.List of 30 Best OOPs Interview Questions with AnswersIn the upcoming section, you will get hands-on with the most asked interview questions on Object-oriented programming with their perfect answers. So, if you are a beginner and experienced in programming go through the questions and ace your upcoming interviews.1. What is Object Oriented Programming (OOPs)?Object Oriented Programming is a programming paradigm where the complete software operates as a bunch of objects talking to each other. An object is a collection of data and the methods which operate on that data.2. Why OOPs?The main advantage of OOP is better manageable code that covers the following:The overall understanding of the software is increased as the distance between the language spoken by developers and that spoken by users.Object orientation eases maintenance by the use of encapsulation. One can easily change the underlying representation by keeping the methods the same.The OOPs paradigm is mainly useful for relatively big software.The programming paradigm is referred to the technique or approach of writing a program. The programming paradigms can be classified into the following types: 1. Imperative Programming ParadigmIt is a programming paradigm that works by changing the program state through assignment statements. The main focus in this paradigm is on how to achieve the goal. The following programming paradigms come under this category:Procedural Programming Paradigm: This programming paradigm is based on the procedure call concept. Procedures, also known as routines or functions are the basic building blocks of a program in this paradigm.Object-Oriented Programming or OOP: In this paradigm, we visualize every entity as an object and try to structure the program based on the state and behavior of that object.Parallel Programming: The parallel programming paradigm is the processing of instructions by dividing them into multiple smaller parts and executing them concurrently.2. Declarative Programming ParadigmDeclarative programming focuses on what is to be executed rather than how it should be executed. In this paradigm, we express the logic of a computation without considering its control flow. The declarative paradigm can be further classified into:Logical Programming Paradigm: It is based on formal logic where the program statements express the facts and rules about the problem in the logical form.Functional Programming Paradigm: Programs are created by applying and composing functions in this paradigm.Database Programming Paradigm: To manage data and information organized as fields, records, and files, database programming models are utilized.4. What is the difference between Structured Programming and Object-Oriented Programming?Structured Programming is a technique that is considered a precursor to OOP and usually consists of well-structured and separated modules. It is a subset of procedural programming. The difference between OOPs and Structured Programming is as follows:Object-Oriented ProgrammingStructural ProgrammingProgramming that is object-oriented is built on objects having a state and behavior.A program's logical structure is provided by structural programming, which divides programs into their corresponding functions.It follows a bottom-to-top approach.It follows a Top-to-Down approach.Restricts the open flow of data to authorized parts only providing better data security.No restriction to the flow of data. Anyone can access the data.Enhanced code reusability due to the concepts of polymorphism and inheritance.Code reusability is achieved by using functions and loops.Methods work dynamically, making calls based on object behavior and the need of the code at runtime.Functions are called sequentially, and code lines are processed step by step.Modifying and updating the code is easier.Modifying the code is difficult as compared to OOPs.Data is given more importance in OOPs.Code is given more importance.5. What are some commonly used Object-Oriented Programming Languages?OOPs paradigm is one of the most popular programming paradigms. It is widely used in many popular programming languages such as:C++JavaPythonJavaScriptC#Ruby6. What are the advantages and disadvantages of OOPs?Advantages of OOPsDisadvantages of OOPsOOPs provides enhanced code reusability. The programmer should be well-skilled and should have excellent thinking in terms of objects as everything is treated as an object in OOPs.The code is easier to maintain and update.Proper planning is required because OOPs is a little bit tricky.It provides better data security by restricting data access and avoiding unnecessary exposure.OOPs concept is not suitable for all kinds of problems.Fast to implement and easy to redesign resulting in minimizing the complexity of an overall program.The length of the programs is much larger in comparison to the procedural approach.7. 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.8. What is an Object?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 behaviour. C++ #include using namespace std; // defining class class Student { public: string name; }; int main() { // creating object Student student1; // assigning member some value student1.name = "Rahul"; cout