|  |  |  |
| --- | --- | --- |
| Stt | Câu hỏi | Trả lời |
| 1 | Sự khác nhau giữa JDK,JRE và JVM? | |  |  |  |  | | --- | --- | --- | --- | |  | JDK | JRE | JVM | | Chức năng | Phát triển và chạy chương trình java | Chạy chương trình java | Chạy mã bytecode | | Thành phần | Trình biên dịch javac: Biên dịch mã Java (.java) thành bytecode (.class).  javadoc: Tạo tài liệu từ mã nguồn.  jdb: Công cụ gỡ lỗi.  JRE | JVM, Thư viện Java | Công cụ chuyển đổi bytecode | | Người sử dụng | nhà phát triển | Người dùng chạy ứng dụng | Chạy ứng dụng nội bộ | |
| 2 | Sự khác nhau giữa bộ nhớ heap và stack trong java? | |  |  |  | | --- | --- | --- | | Đặc điểm | Stack | Heap | | Mục đích | Lưu biến cục bộ, tham số hàm, lời gọi hàm  Nó hoạt động theo nguyên tắc LIFO (Last In, First Out). | Lưu đối tượng, biến toàn cục  Là một vùng nhớ động được quản lý linh hoạt | | Kích thước | Nhỏ hơn | Lớn hơn | | Tốc độ | Nhanh hơn | Chậm hơn | | Quản lý bộ nhớ | Tự động (phương thức kết thúc) | Dựa và Garbage Collector | | Phạm vi | Mỗi luồng có Stack riêng | Chia sẻ giữa tất cả các luồng | | Thời gian sống | Tồn tại trong thời gian phương thức thực thi | Tồn tại cho đến khi bị thu hồi bởi Garbage Collector | | Dữ liệu lưu trữ | Tham chiếu và biến cục bộ | Đối tượng và biến tĩnh |   Ví dụ:  public class MemoryExample {  public static void main(String[] args) {  int a = 10; // Lưu trong Stack  int b = 20; // Lưu trong Stack  MyClass obj = new MyClass(); // `obj` (tham chiếu) trong Stack, đối tượng ‘MyClass()’ trong Heap  obj.value = 30; // Dữ liệu của đối tượng trong Heap } } class MyClass { int value; // Lưu trong Heap } |
| 3 | Trình biên dịch JIT là gì? | [Trình biên dịch Just-In-Time (JIT) là một loại trình biên dịch động được sử dụng trong nhiều môi trường runtime hiện đại như .NET và Java](https://kb.pavietnam.vn/kien-thuc-co-ban-ve-trinh-bien-dich-jit.html" \t "https://edgeservices.bing.com/edgesvc/_blank). [Thay vì biên dịch toàn bộ mã nguồn trước khi chạy chương trình, JIT biên dịch mã ngay khi cần thiết trong quá trình thực thi](https://itigic.com/vi/just-in-time-jit-compile-how-it-works-in-programming/" \t "https://edgeservices.bing.com/edgesvc/_blank).  [Điều này giúp tối ưu hóa hiệu suất bằng cách chỉ biên dịch những phần mã thực sự cần thiết tại thời điểm đó](https://viblo.asia/p/trinh-bien-dich-javascript-jit-just-in-time-jvElaXRdZkw" \t "https://edgeservices.bing.com/edgesvc/_blank). [JIT kết hợp ưu điểm của cả trình biên dịch và trình thông dịch, giúp chương trình khởi động nhanh hơn và vẫn duy trì được hiệu suất cao trong quá trình chạy](https://viblo.asia/p/trinh-bien-dich-javascript-jit-just-in-time-jvElaXRdZkw" \t "https://edgeservices.bing.com/edgesvc/_blank). |
| \* | Phân biệt trình biên dịch và thông dịch | Trình biên dịch và trình thông dịch đều có nhiệm vụ chuyển đổi mã nguồn từ ngôn ngữ lập trình cấp cao sang mã máy, nhưng chúng hoạt động theo cách khác nhau: Trình biên dịch (Compiler)  * [Cách hoạt động: Chuyển đổi toàn bộ mã nguồn thành mã máy trước khi chương trình chạy](https://www.guru99.com/vi/difference-compiler-vs-interpreter.html" \t "https://edgeservices.bing.com/edgesvc/_blank). * [Tốc độ: Chương trình biên dịch chạy nhanh hơn vì mã đã được chuyển đổi hoàn toàn trước khi thực thi](https://www.guru99.com/vi/difference-compiler-vs-interpreter.html" \t "https://edgeservices.bing.com/edgesvc/_blank). * [Phát hiện lỗi: Hiển thị tất cả các lỗi sau khi quá trình biên dịch hoàn tất](https://www.guru99.com/vi/difference-compiler-vs-interpreter.html" \t "https://edgeservices.bing.com/edgesvc/_blank). * [Bộ nhớ: Yêu cầu nhiều bộ nhớ hơn để lưu trữ mã đối tượng trung gian](https://viblo.asia/p/su-khac-nhau-giua-trinh-bien-dich-va-trinh-thong-dich-Eb85oxG2K2G" \t "https://edgeservices.bing.com/edgesvc/_blank). * [Ngôn ngữ phổ biến: C, C++, C#, Java](https://viblo.asia/p/su-khac-nhau-giua-trinh-bien-dich-va-trinh-thong-dich-Eb85oxG2K2G" \t "https://edgeservices.bing.com/edgesvc/_blank).  Trình thông dịch (Interpreter)  * [Cách hoạt động: Chuyển đổi từng dòng mã nguồn thành mã máy và thực thi ngay lập tức](https://www.guru99.com/vi/difference-compiler-vs-interpreter.html" \t "https://edgeservices.bing.com/edgesvc/_blank). * [Tốc độ: Chạy chậm hơn vì mã được chuyển đổi và thực thi từng dòng một](https://www.guru99.com/vi/difference-compiler-vs-interpreter.html" \t "https://edgeservices.bing.com/edgesvc/_blank). * [Phát hiện lỗi: Hiển thị lỗi từng dòng một trong quá trình thực thi](https://www.guru99.com/vi/difference-compiler-vs-interpreter.html" \t "https://edgeservices.bing.com/edgesvc/_blank). * [Bộ nhớ: Yêu cầu ít bộ nhớ hơn vì không tạo mã đối tượng trung gian](https://www.guru99.com/vi/difference-compiler-vs-interpreter.html" \t "https://edgeservices.bing.com/edgesvc/_blank). * [Ngôn ngữ phổ biến: Python, Ruby, PHP](https://www.guru99.com/vi/difference-compiler-vs-interpreter.html" \t "https://edgeservices.bing.com/edgesvc/_blank). |
| 4 | Platform là gì? | Platform (Nền tảng) là một môi trường hoặc hệ thống mà các ứng dụng, dịch vụ, hoặc sản phẩm khác có thể được xây dựng, triển khai và vận hành. Nó cung cấp các công cụ, dịch vụ, và cơ sở hạ tầng cần thiết để hỗ trợ sự phát triển và thực thi của các ứng dụng hoặc dịch vụ.  Các loại Platform chính:  Operating System (Hệ điều hành)  Application Platform (Nền tảng ứng dụng)  Cloud Platform (Nền tảng đám mây)  Web Platform (Nền tảng web)  Hardware Platform (Nền tảng phần cứng)  Mobile Platform (Nền tảng di động)  Social Media Platform (Nền tảng mạng xã hội)  Business Platform (Nền tảng kinh doanh) |
| 5 | Sự khác nhau giữa Java platform và các platform khác? | Java Platform có một số đặc điểm khác biệt so với các nền tảng khác: Java Platform  1. [Độc lập nền tảng: Java Platform dựa trên phần mềm, cho phép các ứng dụng Java chạy trên nhiều nền tảng phần cứng khác nhau nhờ vào Máy ảo Java (JVM)](https://quizlet.com/vn/964929770/cau-hoi-phong-van-java-flash-cards/" \t "https://edgeservices.bing.com/edgesvc/_blank) . 2. [Bytecode: Mã nguồn Java được biên dịch thành bytecode, một ngôn ngữ trung gian, giúp nó có thể chạy trên bất kỳ nền tảng nào có JVM](https://quizlet.com/vn/964929770/cau-hoi-phong-van-java-flash-cards/" \t "https://edgeservices.bing.com/edgesvc/_blank) . 3. [API phong phú: Java cung cấp một bộ API phong phú, hỗ trợ nhiều chức năng từ cơ bản đến nâng cao](https://quizlet.com/vn/964929770/cau-hoi-phong-van-java-flash-cards/" \t "https://edgeservices.bing.com/edgesvc/_blank) . 4. [Viết một lần, chạy mọi nơi: Tính chất “Write Once, Run Anywhere” giúp mã Java có thể chạy trên bất kỳ hệ điều hành nào mà không cần thay đổi 1](https://quizlet.com/vn/964929770/cau-hoi-phong-van-java-flash-cards/" \t "https://edgeservices.bing.com/edgesvc/_blank).  Các nền tảng khác  1. [Nền tảng cụ thể: Nhiều nền tảng khác như .NET, iOS, Android thường được thiết kế để chạy trên một hệ điều hành hoặc phần cứng cụ thể 1](https://quizlet.com/vn/964929770/cau-hoi-phong-van-java-flash-cards/" \t "https://edgeservices.bing.com/edgesvc/_blank). 2. [Ngôn ngữ biên dịch trực tiếp: Các nền tảng như C++ thường biên dịch mã nguồn trực tiếp thành mã máy, phụ thuộc vào hệ điều hành và phần cứng cụ thể 1](https://quizlet.com/vn/964929770/cau-hoi-phong-van-java-flash-cards/" \t "https://edgeservices.bing.com/edgesvc/_blank). 3. [API giới hạn: Một số nền tảng có thể có bộ API giới hạn hơn so với Java, tùy thuộc vào mục đích sử dụng và phạm vi hỗ trợ 1](https://quizlet.com/vn/964929770/cau-hoi-phong-van-java-flash-cards/" \t "https://edgeservices.bing.com/edgesvc/_blank). 4. [Tính tương thích: Các nền tảng khác có thể yêu cầu điều chỉnh mã nguồn khi chuyển đổi giữa các hệ điều hành hoặc phần cứng khác nhau](https://quizlet.com/vn/964929770/cau-hoi-phong-van-java-flash-cards/" \t "https://edgeservices.bing.com/edgesvc/_blank) . |
| 6 | Tính chất "viết một lần chạy nhiều nơi" của java là gì? | là một đặc điểm nổi bật của ngôn ngữ Java, thể hiện khả năng của Java trong việc viết mã nguồn một lần và chạy được trên nhiều hệ điều hành hoặc nền tảng khác nhau mà không cần chỉnh sửa mã. |
| 7 | Classloader trong java là gì? | ClassLoader trong Java là một thành phần của Java Virtual Machine (JVM), chịu trách nhiệm tải các lớp Java (Java classes) vào bộ nhớ trong quá trình runtime. Nó là một phần của Java Runtime Environment (JRE) và đảm bảo rằng tất cả các lớp cần thiết để chạy một ứng dụng Java được tìm thấy, tải, và liên kết đúng cách. |
| 8 | File có tên trống ".java" có hợp lệ không? | Không hợp lệ |
| 9 | Các từ delete, next, main, exit và null có phải là từ khóa trong java không? | Không phải |
| 10 | Nếu không cung cấp bất kỳ đối số nào trên command line, thì mảng String của hàm main là empty hay null? | Là mảng rỗng |
| 11 | Chuyện gì xảy ra nếu khai báo static public void thay vì public static void? | Không ảnh hưởng |
| 12 | Giá trị mặc định của các biến local là gì? | Không có giá trị mặc định, nếu cứ dùng sẽ bị lỗi biên dịch |
| 13 | Khác nhau Biến tham trị và Biến tham chiếu? | Biến tham trị truyền giá trị của biến (copy của dữ liệu gốc) vào phương thức. Phương thức chỉ thao tác trên bản sao, không ảnh hưởng đến dữ liệu gốc.  Biến tham chiếu truyền địa chỉ tham chiếu của đối tượng (không phải bản sao của dữ liệu) vào phương thức. Bất kỳ thay đổi nào thực hiện trên tham chiếu sẽ ảnh hưởng trực tiếp đến dữ liệu gốc |
| 14 | Bạn hiểu gì về Đối tượng? | Đối tượng (Object) là một khái niệm cốt lõi trong lập trình hướng đối tượng (Object-Oriented Programming - OOP). Nó đại diện cho một thực thể trong thế giới thực hoặc một khái niệm trừu tượng, có các thuộc tính (properties) và hành vi (behaviors). |
| 15 | Định nghĩa Lớp (class)? | Nó là một khuôn mẫu (template) hoặc bản thiết kế dùng để tạo ra các đối tượng (objects). |
| 16 | Kiểu biến nào một lớp có thể chứa? | 1. Biến thành viên (Instance Variables)  2. Biến lớp (Class Variables) - static  3. Biến cục bộ (Local Variables)  4. Biến tham số (Parameters)  5. Hằng số (Constant Variables) - final |
| 17 | Biến local hay biến cục bộ là gì? | Biến cục bộ (Local Variable) là một biến được khai báo bên trong một phương thức, constructor, hoặc khối lệnh (block) và chỉ tồn tại trong phạm vi của phương thức, constructor, hoặc khối lệnh đó. Không có giá trị mặc định, Không thể sử dụng từ khóa static |
| 18 | Biến instance là gì? | Biến instance (Instance Variable) là một loại biến trong Java được khai báo bên trong lớp (class) nhưng bên ngoài mọi phương thức, constructor, hoặc khối lệnh. Biến này thuộc về mỗi đối tượng (instance) của lớp, nghĩa là mỗi đối tượng sẽ có một bản sao riêng của biến instance. |
| 19 | Sự khác biệt giữa ngôn ngữ lập trình hướng đối tượng và ngôn ngữ lập trình dựa trên đối tượng là gì? |  |
| 20 | Giá trị khởi tạo của biến tham chiếu đối tượng được định nghĩa là biến instance là gì? | Trong Java, khi một biến tham chiếu đối tượng được định nghĩa là một biến instance nhưng không được khởi tạo cụ thể, giá trị mặc định của nó sẽ là null. Điều này đúng với mọi kiểu tham chiếu (chẳng hạn như các lớp, giao diện, mảng). |
| 21 | Constructor là gì? | Constructor (hàm khởi tạo) là một phương thức đặc biệt trong một lớp, được sử dụng để khởi tạo đối tượng khi đối tượng đó được tạo ra. Constructor có tên giống hệt tên lớp và không có kiểu trả về, kể cả void. |
| 22 | Mục đích của constructor là gì? | để khởi tạo đối tượng khi đối tượng đó được tạo ra |
| 23 | Constructor trả về kiểu giá trị gì? | không có kiểu trả về |
| 24 | Constructor được kế thừa không? | Constructor không được kế thừa trong Java. |
| 25 | Có thể tạo constructor final không? | Không thể tạo constructor final trong Java.  Trong Java, final là một từ khóa có thể được sử dụng để ngăn chặn việc thay đổi giá trị của biến, ghi đè phương thức, hoặc kế thừa lớp |
| 26 | Biến static là gì? | Biến static trong Java là một biến thuộc về lớp thay vì thuộc về đối tượng của lớp đó. Điều này có nghĩa là biến static chỉ có một bản sao duy nhất cho toàn bộ lớp, thay vì mỗi đối tượng có một bản sao riêng của biến đó. Biến static có thể được chia sẻ giữa tất cả các đối tượng của lớp. |
| 27 | Phương thức static là gì? | Phương thức static trong Java là một phương thức thuộc về lớp (class) chứ không thuộc về đối tượng (instance) của lớp. Điều này có nghĩa là bạn có thể gọi phương thức static mà không cần tạo một đối tượng của lớp đó. Không thể truy cập trực tiếp thuộc tính hoặc phương thức không static. Không thể sử dụng từ khóa this hoặc super. Có thể được gọi trong ngữ cảnh static. Không thể bị ghi đè (override) trong kế thừa, vì phương thức static gắn với lớp, không gắn với đối tượng. Nếu bạn định nghĩa lại phương thức static trong lớp con, đó là phương thức ẩn (hiding), không phải ghi đè. |
| 28 | Tại sao phương thức main là static? | 1. Không cần tạo đối tượng để gọi phương thức main  2. Dễ dàng truy cập mà không cần phụ thuộc vào trạng thái đối tượng  3. Chuẩn định nghĩa của JVM  4. Tính nhất quán trong thiết kế |
| 29 | Khối static là gì? | Khối static (static block) trong Java là một đoạn mã được thực thi chỉ một lần duy nhất khi lớp được tải vào bộ nhớ bởi JVM, trước khi bất kỳ đối tượng nào của lớp được tạo hoặc bất kỳ phương thức nào của lớp được gọi.  - đặc điểm  Thực thi tự động  Thực thi một lần duy nhất  Dùng để khởi tạo tài nguyên tĩnh (static members)  Không thể sử dụng this hoặc super  Ứng dụng của khối static:  Khởi tạo biến static: Khối static thường được dùng để khởi tạo các biến hoặc tài nguyên dùng chung trong lớp.  Tải tài nguyên: Sử dụng để tải các tệp cấu hình, kết nối cơ sở dữ liệu, hoặc chuẩn bị tài nguyên cần thiết khi lớp được tải.  Kiểm tra điều kiện: Thực hiện các kiểm tra trước khi lớp có thể được sử dụng. |
| 30 | Chúng ta có thể thực thi một chương trình không có phương thức main() không? | 1. Sử dụng khối static:  Trong Java, khối static được thực thi ngay khi lớp được tải vào bộ nhớ. Do đó, bạn có thể viết logic trong khối static để chạy chương trình mà không cần main().  2. Sử dụng công cụ hoặc framework đặc biệt:  Một số framework (như Spring, JavaFX, hoặc Servlet) có cơ chế khởi chạy chương trình riêng mà không yêu cầu phương thức main() rõ ràng trong mã nguồn của bạn. Trong trường hợp này, bạn chỉ cần cung cấp các cấu hình hoặc lớp khởi động mà framework yêu cầu  3. Sử dụng kỹ thuật bytecode manipulation:  Bạn có thể chỉnh sửa bytecode của lớp để thực thi chương trình mà không cần khai báo phương thức main(). Điều này thường được thực hiện bằng cách sử dụng các thư viện như ASM hoặc Javassist.  4. Dùng các công cụ thử nghiệm:  Khi chạy các bài kiểm thử (test) bằng các framework như JUnit hoặc TestNG, bạn có thể thực thi các phương thức khác mà không cần có main() trong mã của mình. Framework này sẽ quản lý việc khởi chạy chương trình.  5. Sử dụng IDE hoặc trình quản lý runtime:  Một số môi trường phát triển tích hợp (IDE) hoặc công cụ runtime có thể chạy chương trình với các cấu hình đặc biệt, không yêu cầu main(). |
| 31 | Chuyện gì xảy ra khi phương thức main không có static? | Nếu phương thức main không có từ khóa static, JVM sẽ không thể gọi trực tiếp phương thức main khi chương trình khởi chạy, và chương trình sẽ không thực thi được. Điều này dẫn đến lỗi runtime. |
| 32 | Sự khác nhau giữa phương thức static và phương thức instance? |  |
| 33 | this trong java là gì? | this trong Java là một từ khóa đặc biệt được sử dụng để tham chiếu đến đối tượng hiện tại của lớp. Nó có vai trò quan trọng trong việc truy cập và xử lý các thành phần của đối tượng trong các tình huống cụ thể.  Công dụng của this trong Java  a. Tham chiếu đến biến instance của đối tượng hiện tại  b. Gọi phương thức của đối tượng hiện tại  c. Gọi constructor khác trong cùng một lớp  d. Truyền đối tượng hiện tại làm tham số  e. Trả về đối tượng hiện tại |
| 34 | Kế thừa là gì? | Kế thừa là một khái niệm quan trọng trong lập trình hướng đối tượng (OOP - Object-Oriented Programming). Nó cho phép một lớp (class) mới được tạo ra từ một lớp hiện có bằng cách thừa kế các thuộc tính và phương thức của lớp đó. Lớp được kế thừa gọi là lớp cha (hoặc lớp cơ sở, superclass), còn lớp mới gọi là lớp con (hoặc lớp dẫn xuất, subclass). |
| 35 | Lớp nào là lớp cha cho tất cả các lớp. | Object |
| 36 | Tại sao đa kế thừa không được hỗ trợ trong java. | Vấn đề Diamond (Kim cương)  Được biết đến là "vấn đề kim cương", nếu Java hỗ trợ đa kế thừa trực tiếp, có thể xảy ra tình huống một lớp kế thừa từ hai lớp cha có cùng một phương thức hoặc thuộc tính. Khi đó, nếu có sự thay đổi trong các lớp cha, việc xác định lớp con sẽ kế thừa từ đâu (lớp cha nào) có thể gây nhầm lẫn và khó kiểm soát. |
| 37 | Composition là gì? | Composition (kết hợp) là một trong những khái niệm quan trọng trong lập trình hướng đối tượng, đặc biệt là trong thiết kế phần mềm. Nó chỉ mối quan hệ giữa các đối tượng, trong đó một đối tượng "chứa" các đối tượng khác như một phần của chính nó. Composition được coi là một mối quan hệ "has-a" (có một) giữa các lớp, có nghĩa là một đối tượng có thể sử dụng các đối tượng khác để thực hiện các chức năng của mình. |
| 38 | Sự khác nhau giữa aggregation và composition? |  |
| 39 | Tại sao java không support con trỏ? |  |
| 40 | super trong java là gì? | super trong Java là một từ khóa được sử dụng để tham chiếu đến lớp cha (superclass) của một đối tượng. Nó có thể được sử dụng trong các trường hợp sau:  1. Gọi Constructor của lớp cha  2. Truy cập đến phương thức của lớp cha  3. Truy cập thuộc tính của lớp cha |
| 41 | Có thể sử dụng cả this() và super() trong một constructor? | Trong Java, bạn **không thể sử dụng cả** this() **và** super() **trong cùng một constructor**. Điều này là vì trong Java, **chỉ có thể gọi một constructor khác trong cùng một constructor**, và bạn không thể gọi cả this() (constructor trong cùng lớp) và super() (constructor trong lớp cha) cùng một lúc. |
| 42 | Object cloning là gì? | Object cloning (sao chép đối tượng) là quá trình tạo ra một bản sao của đối tượng trong lập trình. Điều này có nghĩa là bạn tạo ra một đối tượng mới có cùng trạng thái (giá trị của các thuộc tính) như đối tượng ban đầu. Có hai loại cloning chính:  1. \*\*Shallow Clone (Sao chép nông)\*\*: Bản sao chỉ sao chép các tham chiếu đến các đối tượng con (nếu có), không sao chép sâu các đối tượng con. Nghĩa là, nếu đối tượng con thay đổi, đối tượng sao chép sẽ bị ảnh hưởng.  2. \*\*Deep Clone (Sao chép sâu)\*\*: Bản sao sẽ sao chép tất cả các đối tượng con và các đối tượng con của chúng, tạo ra một bản sao hoàn toàn độc lập với đối tượng gốc.  Cloning thường được sử dụng trong các ngữ cảnh khi bạn cần một bản sao của đối tượng mà không muốn thay đổi đối tượng gốc, ví dụ như trong các mô hình thiết kế phần mềm như Prototype Pattern. |
| 43 | Overloading (nạp chồng) phương thức là gì? | là một kỹ thuật cho phép nhiều phương thức trong cùng một lớp có cùng tên nhưng khác nhau về danh sách tham số. Các phương thức nạp chồng có thể khác nhau ở:   1. Số lượng tham số. 2. Kiểu dữ liệu của tham số. 3. Thứ tự của các tham số (nếu chúng có kiểu khác nhau). |
| 44 | Tại sao overloading phương thức không xảy ra khi thay đổi kiểu giá trị trả về? | Overloading không xảy ra khi chỉ thay đổi kiểu giá trị trả về là vì nó không đủ để trình biên dịch phân biệt các phương thức. Để tránh nhầm lẫn, overloading dựa vào sự khác biệt ở danh sách tham số, giúp mã rõ ràng và dễ bảo trì hơn. |
| 45 | Có thể overload phương thức main() không? | có thể overload phương thức main() trong Java, nhưng chỉ phương thức main(String[] args) có vai trò đặc biệt là điểm vào của chương trình và được JVM gọi tự động. Các phiên bản khác của main chỉ hoạt động như các phương thức bình thường. |
| 46 | Ghi đè (overriding) phương thức là gì? | cho phép một lớp con cung cấp cách triển khai riêng của một phương thức đã được định nghĩa trong lớp cha. Khi ghi đè, phương thức trong lớp con sẽ thay thế phương thức tương ứng trong lớp cha.  Đặc điểm   1. Tên phương thức giống nhau: Phương thức trong lớp con phải có cùng tên với phương thức trong lớp cha. 2. Danh sách tham số giống nhau: Số lượng, kiểu dữ liệu, và thứ tự của các tham số phải giống hoàn toàn với phương thức trong lớp cha. 3. Kiểu trả về tương thích:   - Kiểu trả về của phương thức ghi đè phải giống hoặc là kiểu con (có thể thay thế) của kiểu trả về của phương thức trong lớp cha (áp dụng từ Java 5 trở đi với covariant return type).   1. Phạm vi truy cập không được thu hẹp:   Phạm vi truy cập của phương thức ghi đè phải bằng hoặc lớn hơn phương thức trong lớp cha.  Ví dụ: Nếu phương thức trong lớp cha là protected, phương thức ghi đè không thể là private và Default. |
| 47 | Có thể ghi đè phương thức static không? |  |
| 48 | Tại sao không thể ghi đè phương thức static? | Phương thức static không thể ghi đè (vì chúng thuộc về lớp, không phải đối tượng). |
| 49 | Có thể ghi đè phương thức đã nạp chồng? | Có  Khi một phương thức đã được nạp chồng trong lớp cha, lớp con vẫn có thể ghi đè từng phiên bản của các phương thức đó. Tuy nhiên, ghi đè chỉ áp dụng cho các phương thức có cùng chữ ký (signature) giữa lớp cha và lớp con. |
| 50 | Có thể ghi đè biến instance không? | Trong Java, bạn **không thể ghi đè (override)** một biến instance. Ghi đè chỉ áp dụng cho các phương thức, không áp dụng cho các biến instance hoặc biến tĩnh.  Tuy nhiên, bạn có thể **ẩn (hide)** một biến instance trong lớp con bằng cách khai báo một biến với cùng tên trong lớp con. Khi đó, hai biến này sẽ độc lập với nhau, và biến nào được sử dụng phụ thuộc vào loại tham chiếu (reference type-kiểu được khai báo) được sử dụng. |
| 51 | Sự khác nhau giữa nạp chồng và ghi đè là gì? |  |
| 52 | Biến final là gì? | Biến được khai báo với từ khóa final được gọi là **hằng số** (constant). |
| 53 | Phương thức final là gì? | Phương thức được khai báo với từ khóa final được gọi là phương thức final. Phương thức này **không thể bị ghi đè (override)** bởi các lớp con. |
| 54 | Lớp final là gì? | Lớp được khai báo với từ khóa final **không thể bị kế thừa (extends) bởi bất kỳ lớp nào khác**. Điều này có nghĩa là bạn không thể tạo một lớp con từ lớp final. |
| 55 | Biến final blank là gì? | **Final blank variable** là biến được khai báo với từ khóa final nhưng **chưa được khởi tạo ngay lập tức** khi khai báo.Điều này có nghĩa là **giá trị của nó sẽ được gán sau đó, nhưng chỉ một lần duy nhất.**  Có hai loại biến **final blank** trong Java:   * **Biến instance (**final instance variable**)**: Được gán giá trị trong **constructor**. * **Biến static (**final static variable**)**: Được gán giá trị trong **static block**. |
| 56 | Có thể khởi tạo giá trị cho biến final blank không? | Có thể khởi tạo giá trị cho **biến** final blank, nhưng chỉ **một lần duy nhất** |
| 57 | Có thể khai báo phương thức main là final không? | Trong Java, bạn **có thể** khai báo phương thức main là final. Tuy nhiên, việc này không có ý nghĩa đặc biệt nào đối với phương thức main vì nó là một phương thức tĩnh (static) và không thể bị ghi đè (override) dù có khai báo final hay không. |
| 58 | Đa hình tại runtime là gì? | Đa hình tại runtime là một cơ chế trong lập trình hướng đối tượng cho phép phương thức được gọi dựa trên **đối tượng thực tế** (thể hiện của lớp con) thay vì kiểu tham chiếu của nó. Điều này đạt được thông qua **method overriding (ghi đè phương thức)**. |
| 59 | Có thể thực hiện đa hình lúc runtime với các thành viên dữ liệu không? | Không! **Đa hình tại runtime (Runtime Polymorphism) không áp dụng cho các biến dữ liệu (fields) trong Java.**  Trong Java, chỉ **các phương thức** (methods) được ghi đè (overridden) mới hỗ trợ đa hình tại runtime. Còn **các thành viên dữ liệu (fields)** sẽ bị ràng buộc (binding) tại thời điểm biên dịch (compile-time), **không phụ thuộc vào đối tượng thực tế**. |
| 60 | Sự khác nhau giữa ràng buộc tĩnh và ràng buộc động là gì? |  |
| 61 | Trừu tượng là gì? | **Trừu tượng (Abstraction)** là một trong bốn nguyên tắc cơ bản của lập trình hướng đối tượng (OOP), giúp **che giấu các chi tiết không cần thiết** và **chỉ hiển thị những chức năng cốt lõi** cho người dùng.  Trong Java, trừu tượng được thực hiện bằng cách sử dụng: ✔ **Lớp trừu tượng (**abstract class**)** ✔ **Giao diện (**interface**)** |
| 62 | Sự khác nhau giữa trừu tượng và đóng gói là gì? |  |
| 63 | Lớp trừu tượng là gì? | **\* Lớp trừu tượng (**abstract class**)** là một lớp trong Java có thể chứa **các phương thức trừu tượng (**abstract method**)** (không có phần thân) và **các phương thức bình thường (**concrete method**)**.  \* Lớp trừu tượng không thể khởi tạo trực tiếp mà **phải được kế thừa (extends) bởi lớp con** để sử dụng. |
| 64 | Có phương thức trừu tượng không nằm trong lớp trừu tượng không? | Không, trong Java **một phương thức trừu tượng (**abstract method**) luôn phải nằm trong một lớp trừu tượng (**abstract class**) hoặc trong một giao diện (**interface**)**. |
| 65 | Có thể sử dụng cả abstract và final cho một phương thức không? | **Không thể** |
| 66 | Có thể tạo thể hiện của lớp trừu tượng không? | Không, bạn **không thể tạo thể hiện (instance) trực tiếp của một lớp trừu tượng (abstract class)** trong Java. Lớp trừu tượng được thiết kế để làm lớp cơ sở cho các lớp con kế thừa và triển khai các phương thức trừu tượng của nó |
| 67 | Interface là gì? | **Interface** trong Java là một khái niệm quan trọng trong lập trình hướng đối tượng (OOP). Nó định nghĩa một bộ các phương thức (methods) mà các lớp cụ thể (concrete classes) phải triển khai. Interface giúp đạt được tính trừu tượng (abstraction) và đa hình (polymorphism), đồng thời hỗ trợ việc thiết kế các hệ thống linh hoạt và dễ bảo trì.  **\* Interface** là một kiểu dữ liệu tham chiếu (reference type) trong Java, tương tự như một lớp (class), nhưng nó chỉ chứa các phương thức trừu tượng (abstract methods), hằng số (constants), và các phương thức mặc định (default methods) hoặc phương thức tĩnh (static methods) (từ Java 8 trở đi).  \* Một lớp có thể **triển khai (implement)** một hoặc nhiều interface, và phải cung cấp thân hàm (implementation) cho tất cả các phương thức trừu tượng của interface đó. |
| 68 | Có thể khai báo một phương thức của interface với từ khóa static không? | Từ **Java 8**, Java cho phép khai báo phương thức static trong interface.  · **Phương thức** static **trong** interface **phải có phần thân** (giống như trong lớp bình thường).  · **Không thể override phương thức** static **của interface trong lớp triển khai (**implements**)**.  · **Gọi phương thức** static **của interface bằng cách dùng tên interface, không cần đối tượng**. |
| 69 | Một interface có thể là final không? | ****Không thể khai báo một**** interface ****là**** final ****trong Java**** Trong Java, **một** interface **không thể có từ khóa** final, vì final ngăn chặn việc kế thừa (inheritance), trong khi **bản chất của** interface **là để được triển khai (**implements**) bởi các lớp khác**. |
| 70 | Marker interface là gì? | **Marker Interface** là một interface **không chứa bất kỳ phương thức hoặc thuộc tính nào**. Nó chỉ đóng vai trò **đánh dấu (marker)** cho các lớp triển khai nó, giúp trình biên dịch hoặc runtime biết rằng lớp đó có một đặc tính cụ thể. |
| 71 | Sự khác nhau giữa lớp abstract và interface là gì? |  |
| 72 | Có thẻ định nghĩa private hoặc protected cho các biến trong interface không? | **Không thể, các biến trong interface được định nghĩa mặc định là “public static final”** |
| 73 | Khi nào một tham chiếu đối tượng có thể được ép sang kiểu interface tham chiếu? |  |
| 74 | Package là gì? | **Package** trong Java là một cách để **nhóm các lớp (classes), interface, và các thành phần khác lại với nhau** nhằm tổ chức mã nguồn tốt hơn và tránh xung đột tên lớp. |
| 75 | Có cần import package import java.lang không? tại sao? | ✅ **Có, nếu muốn sử dụng lớp từ một package khác!** ❌ **Không cần nếu lớp nằm cùng package.** |
| 76 | Có thể import package/lớp giống nhau hai lần không? JVM sẽ tải package hai lần khi chạy không? | **Có thể** import cùng một package hoặc cùng một lớp **nhiều lần**, nhưng điều đó **không có tác dụng** và **không gây lỗi**.  Java **chỉ tải (load) một lần** dù bạn import bao nhiêu lần.  import java.util.\*;  import java.util.ArrayList; // Không có lỗi, nhưng ArrayList đã có trong java.util.\* |
| 77 | Static import là gì? | **Static Import** là một tính năng trong Java (từ Java 5) cho phép bạn **import trực tiếp các thành viên static (biến, phương thức) của một lớp**, thay vì phải gọi chúng thông qua tên lớp. |
| 78 | Xử lý ngoại lệ (handling exception) là gì? | **Xử lý ngoại lệ** là cơ chế giúp chương trình **phát hiện, xử lý lỗi một cách an toàn mà không làm gián đoạn quá trình thực thi**. |
| 79 | Sự khác biệt giữa checked exception và unchecked exception là gì? |  |
| 80 | Có phải mỗi khối try phải đi kèm với một khối catch? |  |