**1.Sự khác nhau giữa JDK,JRE và JVM**

JDK (Java Development Kit) là một bộ công cụ phát triển cho việc phát triển ứng dụng Java. Nó bao gồm các công cụ như trình biên dịch, các thư viện cần thiết, và các công cụ khác để phát triển ứng dụng Java.

JRE (Java Runtime Environment) là một môi trường thực thi để chạy các ứng dụng Java. Nó bao gồm các thư viện và các công cụ cần thiết để chạy các ứng dụng Java.

JVM (Java Virtual Machine) là một máy ảo Java, thực thi mã bytecode Java và cung cấp môi trường để chạy các ứng dụng Java.

**2. Sự khác nhau giữa bộ nhớ heap và stack trong java?**

Trong Java, bộ nhớ heap được sử dụng để lưu trữ đối tượng được tạo trong quá trình chạy của chương trình. Các đối tượng được lưu trữ trong heap được quản lý bởi garbage collector và tồn tại cho đến khi không còn tham chiếu nào trỏ đến chúng.

Trong khi đó, stack được sử dụng để lưu trữ các biến cục bộ và các tham chiếu đến các đối tượng trong các phương thức của chương trình. Mỗi thread có một stack riêng, và nó chứa các khuôn của các phương thức được gọi

**3.Trình biên dịch JIT là gì?**

Trình biên dịch JIT (Just-In-Time) trong Java là một phần của JVM. Nó hoạt động bằng cách chuyển đổi mã bytecode Java thành mã máy cụ thể của nền tảng khi chương trình được thực thi. Điều này giúp cải thiện hiệu suất bằng cách tối ưu hóa mã khi chương trình đang chạy.

**4. Platform là gì?**

Platform là một môi trường phần cứng hoặc phần mềm trong đó một chương trình hoạt động. Nó bao gồm tất cả các thành phần cần thiết để chạy một ứng dụng, bao gồm cả hệ điều hành, phần cứng và các thư viện hỗ trợ.

**5. Sự khác nhau giữa Java platform và các platform khác?**

Java platform là một môi trường phát triển và thực thi dành cho việc phát triển ứng dụng Java. Nó bao gồm JDK để phát triển và JRE để thực thi các ứng dụng Java. Các platform khác có thể là các môi trường khác nhau như .NET Framework, Node.js, Android, v.v., mỗi platform có các tính năng và cấu trúc riêng biệt.

**6. Tính chất "viết một lần chạy nhiều nơi" của java là gì?**

Tính chất "viết một lần chạy mọi nơi" của Java đề cập đến khả năng viết mã một lần và chạy trên mọi nền tảng hỗ trợ Java, miễn là nền tảng đó có JVM tương thích.

**7. Classloader trong java là gì?**

Classloader trong Java là một phần của JVM, nó được sử dụng để tải các lớp và gói từ không gian tên lớp Java vào bộ nhớ của JVM. Classloader tải các lớp cần thiết khi chương trình được thực thi và chịu trách nhiệm quản lý các phụ thuộc lớp.

**8. File có tên trống ".java" có hợp lệ không?**

File có tên trống ".java" không hợp lệ trong Java. Tên file .java phải khớp với tên của lớp chứa trong file đó.

**9. Các từ delete, next, main, exit và null có phải là từ khóa trong java không?**

Trong Java, "delete", "next", "main", "exit" không phải là từ khóa. Tuy nhiên, "null" là một từ khóa trong Java, đại diện cho một tham chiếu không trỏ đến đối tượng nào.

**10. Nếu không cung cấp bất kỳ đối số nào trên command line, thì mảng String của hàm main là empty hay null?**

Nếu không cung cấp bất kỳ đối số nào trên command line, thì mảng String của hàm main không phải là null mà là một mảng rỗng (empty).

**11. Chuyện gì xảy ra nếu khai báo static public void thay vì public static void?**

Trong Java, thứ tự từ khóa "static" và "public" không ảnh hưởng đến tính hợp lệ của phương thức. Vì vậy, khai báo "static public void" hoặc "public static void" đều là hợp lệ và có cùng ý nghĩa.

**12. Giá trị mặc định của các biến local là gì?**

Giá trị mặc định của các biến local trong Java không tồn tại. Trước khi sử dụng, các biến local phải được khởi tạo bởi giá trị tương ứng.

**13. Khác nhau Biến tham trị và Biến tham chiếu?**

Biến tham trị là biến mà tham số được truyền vào một phương thức là một bản sao của giá trị ban đầu của biến đó. Bất kỳ thay đổi nào đối với biến tham trị không ảnh hưởng đến biến ban đầu.q

Biến tham chiếu là biến mà tham số được truyền vào một phương thức là một tham chiếu đến đối tượng thực sự trong bộ nhớ. Bất kỳ thay đổi nào đối với biến tham chiếu sẽ ảnh hưởng đến đối tượng được tham chiếu.

**14. Bạn hiểu gì về Đối tượng?**

Trong lập trình hướng đối tượng, đối tượng là một thể hiện của một lớp. Đối tượng bao gồm dữ liệu (các trường) và các phương thức để làm việc với dữ liệu đó.

**15. Định nghĩa Lớp (class)?**

. Lớp (class) trong lập trình hướng đối tượng là một mô hình hoặc bản thiết kế cho các đối tượng. Nó định nghĩa các thuộc tính (trường) và hành vi (phương thức) mà các đối tượng của nó sẽ có.

**16. Kiểu biến nào một lớp có thể chứa?**

Một lớp trong Java có thể chứa các kiểu biến sau:

Các biến instance (thuộc tính).

Biến static (biến lớp).

Biến local (trong các phương thức).

Tham số của phương thức.

**17. Biến local hay biến cục bộ là gì?**

Biến local (hay còn gọi là biến cục bộ) là biến được khai báo bên trong một phương thức hoặc một khối mã và chỉ có phạm vi sử dụng trong phạm vi đó.

**18. Biến instance là gì?**

Biến instance là biến được khai báo trong một lớp và không được đánh dấu là "static". Mỗi thể hiện của lớp (đối tượng) có một bản sao riêng của các biến instance này.

**19. Sự khác biệt giữa ngôn ngữ lập trình hướng đối tượng và ngôn ngữ lập trình dựa trên đối tượng là gì?**

Ngôn ngữ lập trình hướng đối tượng (OOP) là một phương pháp lập trình mà mọi thứ được hiểu như các đối tượng có thể tương tác với nhau. Trong khi ngôn ngữ lập trình dựa trên đối tượng (object-based) không nhất thiết phải tuân theo tất cả các nguyên tắc của OOP và có thể thiếu một số tính năng như kế thừa hoặc đa hình.

**20. Giá trị khởi tạo của biến tham chiếu đối tượng được định nghĩa là biến instance là gì?**

Giá trị khởi tạo của biến tham chiếu đối tượng được định nghĩa là biến instance là null.Top of Form

**21. Constructor là gì?**

Constructor là một phương thức đặc biệt trong Java có tên giống với tên của lớp và không có kiểu trả về. Nó được sử dụng để khởi tạo đối tượng mới của lớp đó.

**22. Mục đích của constructor là gì?**

Mục đích chính của constructor là khởi tạo các giá trị ban đầu cho các biến instance của đối tượng khi nó được tạo.

**23. Constructor trả về kiểu giá trị gì?**

1. Constructor không trả về bất kỳ kiểu giá trị nào. Thay vào đó, nó được sử dụng để khởi tạo đối tượng.

**24. Constructor được kế thừa không?**

Constructor không được kế thừa trong Java. Mỗi lớp con phải định nghĩa các constructor của riêng mình. Tuy nhiên, constructor của lớp cha có thể được gọi từ constructor của lớp con bằng cách sử dụng từ khóa "super".

**25. Có thể tạo constructor final không?**

Constructor không thể được khai báo là "final" trong Java vì không có ý nghĩa khi một constructor được kế thừa hoặc ghi đè.

**26. Biến static là gì?**

Biến static trong Java là một biến được liên kết với lớp, không phải với mỗi đối tượng cụ thể của lớp đó. Nó được chia sẻ trên tất cả các thể hiện của lớp và tồn tại cho đến khi chương trình kết thúc.

**27. Phương thức static là gì?**

Phương thức static là một phương thức được liên kết với lớp, không phải với mỗi đối tượng cụ thể của lớp đó. Nó có thể được gọi mà không cần tạo thể hiện của lớp đó.

**28. Tại sao phương thức main là static?**

Phương thức main là static vì nó được gọi bởi JVM mà không cần tạo một đối tượng của lớp chứa nó.

**29. Khối static là gì?**

Khối static (static block) là một khối mã được sử dụng để khởi tạo các biến static. Nó được thực thi chỉ một lần khi lớp được nạp vào bộ nhớ.

**30. Chúng ta có thể thực thi một chương trình không có phương thức main() không?**

Không, trong Java, một chương trình phải có một phương thức main() static để bắt đầu thực thi. Một chương trình không có phương thức main() sẽ không thể thực thi.

**31. Chuyện gì xảy ra khi phương thức main không có static?**Khi phương thức main không có static, JVM sẽ không biết cách để gọi phương thức main để bắt đầu thực thi chương trình. Điều này sẽ dẫn đến lỗi và chương trình sẽ không thể chạy được.

**32.Sự khác nhau giữa phương thức static và phương thức instance?**

Phương thức static:

Được gắn liền với lớp, không phụ thuộc vào việc tạo thể hiện của lớp.

Có thể được gọi trực tiếp từ lớp mà không cần tạo đối tượng.

Không thể truy cập các biến instance và không thể gọi phương thức không static từ bên trong phương thức static.

Phương thức instance:

Phụ thuộc vào việc tạo thể hiện của lớp và làm việc trên các biến instance của lớp.

Phải được gọi thông qua một đối tượng cụ thể của lớp.

Có thể truy cập cả biến instance và phương thức instance của lớp.

**33. this trong java là gì?**

Trong Java, this là một tham chiếu đến đối tượng hiện tại. Nó được sử dụng để tham chiếu đến các biến instance và phương thức của lớp hiện tại.

**34. Kế thừa là gì?**

Kế thừa trong lập trình hướng đối tượng là một quá trình cho phép một lớp (lớp con) kế thừa các thuộc tính và phương thức của một lớp khác (lớp cha). Lớp con có thể sử dụng lại các thành phần đã được định nghĩa trong lớp cha mà không cần phải viết lại chúng.

**35. Lớp nào là lớp cha cho tất cả các lớp.**

Lớp Object là lớp cha cho tất cả các lớp trong Java. Mọi lớp đều trực tiếp hoặc gián tiếp kế thừa từ lớp Object.

**36. Tại sao đa kế thừa không được hỗ trợ trong java.**

Java không hỗ trợ đa kế thừa vì nó có thể dẫn đến những vấn đề phức tạp như xung đột tên, mâu thuẫn giữa các phương thức cùng tên từ các lớp cha khác nhau, gây khó khăn trong việc hiểu và bảo trì mã.

**37. Composition là gì?**

Composition là một mối quan hệ trong lập trình hướng đối tượng mà một đối tượng được tạo bằng cách sử dụng các đối tượng khác làm thành phần của nó. Đối tượng này sở hữu các thành phần và quản lý vòng đời của chúng.

**38. Sự khác nhau giữa aggregation và composition?**

Aggregation là một loại của composition, trong đó một đối tượng có thể tồn tại mà không cần tồn tại đối tượng khác. Nói cách khác, quan hệ giữa các đối tượng trong aggregation là một mối quan hệ "có" ("has-a").

Composition là một mối quan hệ mạnh hơn, trong đó đối tượng tồn tại chỉ trong phạm vi của đối tượng khác. Quan hệ giữa các đối tượng trong composition là một mối quan hệ "là" ("is-a").

**39. Tại sao java không support con trỏ?**

Java không hỗ trợ con trỏ vì con trỏ có thể dẫn đến các vấn đề bảo mật và quản lý bộ nhớ không an toàn, dễ dẫn đến lỗi như truy cập đến vùng nhớ không hợp lệ.

**40. super trong java là gì?**

Trong Java, super là một từ khóa được sử dụng để tham chiếu đến lớp cha gần nhất của một lớp. Nó được sử dụng để gọi phương thức của lớp cha, truy cập các biến instance của lớp cha và thực hiện các tác vụ khác liên quan đến lớp cha.

**41.Có thể sử dụng cả this() và super() trong một constructor?**

Có, bạn có thể sử dụng cả this() và super() trong một constructor, nhưng không thể sử dụng cả hai trong cùng một constructor. this() được sử dụng để gọi một constructor khác trong cùng một lớp, trong khi super() được sử dụng để gọi constructor của lớp cha.

**42.Object cloning là gì?**

Object cloning là quá trình tạo ra một bản sao của một đối tượng hiện có trong Java. Quá trình này thực hiện bằng cách sử dụng phương thức clone(), nó tạo ra một bản sao của đối tượng gốc và trả về một tham chiếu đến bản sao đó.

**43.Overloading (nạp chồng) phương thức là gì**

Overloading (nạp chồng) phương thức là quá trình định nghĩa nhiều phương thức cùng tên trong một lớp, nhưng có các danh sách tham số khác nhau hoặc kiểu tham số khác nhau.

**44.Tại sao overloading phương thức không xảy ra khi thay đổi kiểu giá trị trả về?**

Overloading phương thức không xảy ra khi chỉ thay đổi kiểu giá trị trả về vì kiểu giá trị trả về không được xem xét khi quyết định phương thức nào sẽ được gọi. Java chỉ xem xét các tham số của phương thức để quyết định xem phương thức nào sẽ được gọi.

**45.Có thể overload phương thức main() không?**

Không, phương thức main() không thể được overload vì JVM chỉ gọi phương thức main() có định dạng chính xác là "public static void main(String[] args)" để bắt đầu thực thi chương trình.

**46. Ghi đè (overriding) phương thức là gì?**

Ghi đè (overriding) phương thức là quá trình định nghĩa lại một phương thức từ lớp cha trong lớp con với cùng một chữ ký (tên phương thức, danh sách tham số và kiểu trả về).

**47. Có thể ghi đè phương thức static không?**

Không, không thể ghi đè phương thức static trong Java. Khi một phương thức được đánh dấu là static, nó không thể được ghi đè vì nó được gắn liền với lớp chứ không phải với đối tượng cụ thể.

**48. Có thể ghi đè phương thức static không?**

Phương thức static không thể ghi đè vì nó liên kết với lớp, không phải với đối tượng cụ thể. Trong ghi đè, phương thức được ghi đè của lớp con cần phải có cùng chữ ký với phương thức được ghi đè của lớp cha. Tuy nhiên, trong trường hợp phương thức static, không có khái niệm kế thừa, do đó không thể ghi đè.

**49. Có thể ghi đè phương thức đã nạp chồng?**

Có, bạn có thể ghi đè phương thức đã nạp chồng trong Java. Ghi đè và nạp chồng là hai khái niệm khác nhau. Overriding liên quan đến việc định nghĩa lại một phương thức trong lớp con có cùng tên và chữ ký với một phương thức trong lớp cha, trong khi overloading liên quan đến việc định nghĩa nhiều phương thức cùng tên nhưng có danh sách tham số khác nhau trong cùng một lớp.

**50. Có thể ghi đè biến instance không?**

Không, không thể ghi đè biến instance trong Java. Biến instance được kế thừa từ lớp cha và không thể định nghĩa lại trong lớp con. Điều này có nghĩa là một biến instance trong lớp con có thể được truy cập từ lớp con, nhưng nó không thể ghi đè hoặc định nghĩa lại.Top of Form

**51. Sự khác nhau giữa nạp chồng và ghi đè là gì?**

Nạp chồng (overloading) và ghi đè (overriding) là hai khái niệm quan trọng trong lập trình hướng đối tượng:

Nạp chồng là quá trình định nghĩa nhiều phương thức cùng tên trong một lớp, nhưng có các danh sách tham số khác nhau hoặc kiểu tham số khác nhau.

Ghi đè là quá trình định nghĩa lại một phương thức từ lớp cha trong lớp con với cùng một chữ ký (tên phương thức, danh sách tham số và kiểu trả về). Ghi đè cho phép lớp con cung cấp một triển khai khác cho một phương thức đã được định nghĩa trong lớp cha.

**52.Biến final là gì?**

Biến final trong Java là một biến mà giá trị của nó không thể thay đổi sau khi được gán giá trị ban đầu.

**53. Phương thức final là gì?**

Phương thức final trong Java là một phương thức mà không thể ghi đè trong lớp con. Nếu một phương thức được đánh dấu là final, thì không thể ghi đè lại nó trong lớp con.

**54. Lớp final là gì?**

Lớp final trong Java là một lớp mà không thể được kế thừa. Điều này có nghĩa là không thể tạo ra một lớp con từ một lớp final.

**55. Biến final blank là gì?**

Biến final blank trong Java là một biến final mà không được khởi tạo giá trị ban đầu ngay khi nó được khai báo.

**56. Có thể khởi tạo giá trị cho biến final blank không?**

Không, không thể khởi tạo giá trị cho biến final blank khi nó được khai báo. Giá trị cho biến final blank phải được gán trong constructor hoặc qua biểu thức khởi tạo.

**57. Có thể khai báo phương thức main là final không?**

Không, không thể khai báo phương thức main là final trong Java vì JVM cần có thể gọi phương thức main mà không bị hạn chế.

**58. Đa hình tại runtime là gì?**

Đa hình tại runtime là khả năng của Java để quyết định phương thức nào sẽ được gọi cho một đối tượng dựa trên loại đối tượng của đối tượng tại thời điểm runtime.

|  |
| --- |
| **59.Có thể thực hiện đa hình lúc runtime với các thành viên dữ liệu không?** |

Không, đa hình tại runtime chỉ áp dụng cho phương thức, không áp dụng cho các thành viên dữ liệu.

**60. Sự khác nhau giữa ràng buộc tĩnh và ràng buộc động là gì?**

Ràng buộc tĩnh (static binding) xảy ra khi liên kết giữa một phương thức và một đối tượng xác định được thực hiện tại thời điểm biên dịch. Trong khi ràng buộc động (dynamic binding) xảy ra khi liên kết giữa một phương thức và một đối tượng xác định được thực hiện tại thời điểm runtime.

**61. Trừu tượng là gì?**

Trừu tượng là một khái niệm trong lập trình hướng đối tượng, mô tả việc tách rời cấu trúc và triển khai của một đối tượng. Đối tượng trừu tượng chỉ mô tả các hành vi mà đối tượng đó cần thực hiện mà không cung cấp triển khai cụ thể cho các hành vi đó.

**62. Sự khác nhau giữa trừu tượng và đóng gói là gì?**

Sự khác biệt giữa trừu tượng và đóng gói:

Trừu tượng: Tách rời cấu trúc và triển khai, chỉ định các hành vi mà đối tượng phải có mà không cung cấp triển khai cụ thể.

Đóng gói: Ẩn thông tin và che giấu chi tiết triển khai bên trong một đối tượng, chỉ expose các phương thức công cộng.

**63. Lớp trừu tượng là gì?**

Lớp trừu tượng là một lớp không thể khởi tạo trực tiếp, mà chỉ có thể được sử dụng thông qua kế thừa. Lớp này thường chứa ít nhất một phương thức trừu tượng.

**64. Có phương thức trừu tượng không nằm trong lớp trừu tượng không?**

Có thể có phương thức trừu tượng nằm trong lớp trừu tượng. Việc triển khai các phương thức này sẽ được thực hiện bởi các lớp con kế thừa từ lớp trừu tượng.

**65. Có thể sử dụng cả abstract và final cho một phương thức không?**

Không, một phương thức không thể được đồng thời khai báo là abstract (trừu tượng) và final. Từ khóa final đánh dấu rằng phương thức không thể được ghi đè (override), trong khi từ khóa abstract yêu cầu phải có triển khai (override) ở các lớp con.

**66. Có thể tạo thể hiện của lớp trừu tượng không?**

Không, không thể tạo thể hiện của một lớp trừu tượng. Lớp trừu tượng chỉ được sử dụng cho mục đích kế thừa và cung cấp các phương thức trừu tượng cho các lớp con triển khai.

**67. Interface là gì?**

Interface là một tập hợp các phương thức trừu tượng và các hằng số (constants). Nó không chứa bất kỳ triển khai cụ thể nào của các phương thức.

**68. Có thể khai báo một phương thức của interface với từ khóa static không?**

Không, không thể khai báo một phương thức của interface với từ khóa static. Interface chỉ chứa phương thức trừu tượng, không thể có các phương thức static.

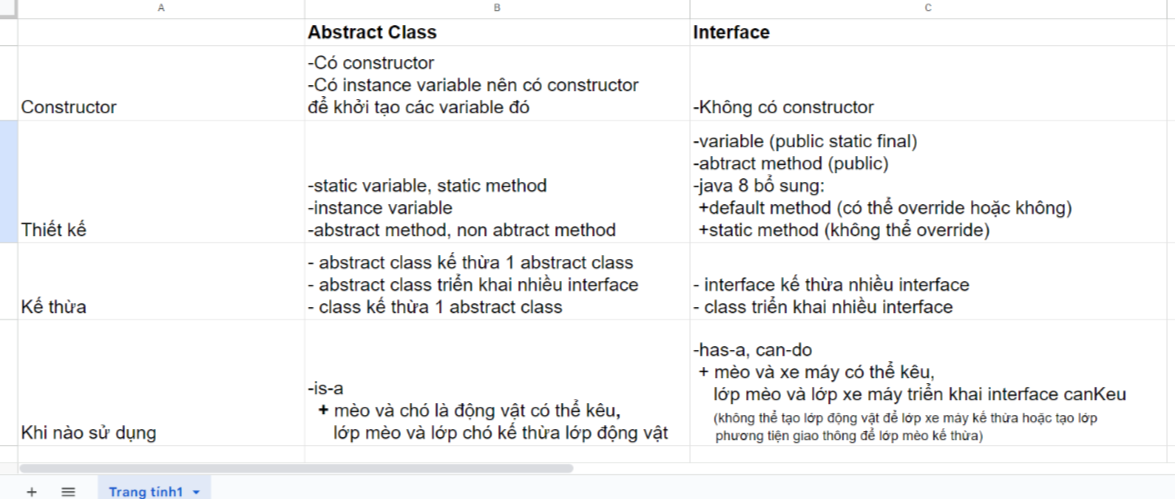
**69. Một interface có thể là final không?**

Không, một interface không thể được khai báo là final. Một interface thường được sử dụng để định nghĩa các hành vi, và nó thường được mở rộng bởi các interface khác.

**70. Marker interface là gì?**

Marker interface là một interface không có bất kỳ phương thức nào, chỉ đơn giản là đánh dấu một lớp hoặc một đối tượng với một tính chất hoặc một tình trạng cụ thể. Ví dụ, interface Serializable trong Java là một marker interface, chỉ đơn giản đánh dấu các lớp có thể được serialize.

**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, trong interface không thể định nghĩa các biến private hoặc protected. Các biến trong interface mặc định là public, static, và final.

**73.Khi nào một tham chiếu đối tượng có thể được ép sang kiểu interface tham chiếu?**

Một tham chiếu đối tượng có thể được ép sang kiểu interface tham chiếu khi đối tượng đó triển khai (implements) interface đó. Điều này cho phép gọi các phương thức của interface từ tham chiếu đối tượng đó.

**74. Package là gì?**

Trong lập trình Java, package là một cách để nhóm các lớp liên quan với nhau và kiểm soát việc truy cập giữa chúng. Nó cung cấp một không gian tên để tránh xung đột tên và giúp tổ chức code.

**75. Có cần import package import java.lang không? tại sao?**

Không cần import package java.lang vì Java tự động import các lớp từ java.lang vào mọi file mã nguồn mặc định.

**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 lớp hai lần, nhưng chỉ có tác dụng làm cho code dễ đọc hơn. JVM chỉ tải package/lớp một lần khi chạy và bỏ qua các lệnh import thừa.

**77. Static import là gì?**

Static import là một tính năng trong Java cho phép bạn import các phần tử static từ một lớp vào một file mã nguồn, giúp rút ngắn cú pháp khi sử dụng các phần tử static đó.

78. Xử lý ngoại lệ (handling exception) là gì?

Xử lý ngoại lệ là quá trình kiểm soát và xử lý các tình huống ngoài dự tính trong quá trình thực thi chương trình.

**79. Sự khác biệt giữa checked exception và unchecked exception là gì?**

Checked exception là các ngoại lệ mà trình biên dịch yêu cầu phải được xử lý thông qua việc sử dụng khối try-catch hoặc throws clause, trong khi unchecked exception là các ngoại lệ mà trình biên dịch không yêu cầu phải xử lý.

**80. Có phải mỗi khối try phải đi kèm với một khối catch?**

Không, một khối try không nhất thiết phải đi kèm với một khối catch. Khối try có thể đi kèm với một hoặc nhiều khối catch hoặc một khối finally hoặc cả hai.

**81. Khối finally là gì?**

Khối finally trong Java là một khối mã được sử dụng để định nghĩa các hành động cần thực thi bất kể có xảy ra ngoại lệ hay không trong khối try-catch. Khối finally thường được sử dụng để giải phóng tài nguyên hoặc thực hiện các hành động dọn dẹp sau khi xử lý ngoại lệ.

**82. Khối finally có thể được sử dụng mà không cần khối catch không?**

Có, khối finally có thể được sử dụng mà không cần khối catch. Nếu không có khối catch đi kèm, khối finally sẽ được thực thi sau khi khối try kết thúc, dù có xảy ra ngoại lệ hay không.

**83. Có trường hợp nào khối finally không được thực thi không?**

Có một số trường hợp mà khối finally không được thực thi, bao gồm khi chương trình bị kết thúc bằng cách gọi phương thức System.exit() hoặc khi máy ảo Java (JVM) bị tắt.

**84. Sự khác nhau giữa throw và throws là gì?**

Sự khác biệt giữa throw và throws:

throw: Dùng để ném một ngoại lệ cụ thể.

throws: Sử dụng trong khai báo phương thức để chỉ ra rằng phương thức đó có thể ném một hoặc nhiều loại ngoại lệ cụ thể.

**85. Có thể khai báo phương thức overriding của lớp con một ngoại lệ nếu phương thức của lớp cha không throw một ngoại lệ?**

Có, bạn có thể khai báo một phương thức overriding của lớp con mà không cần throw một ngoại lệ nếu phương thức của lớp cha không throw ngoại lệ. Tuy nhiên, nếu phương thức của lớp cha throw ngoại lệ, phương thức overriding trong lớp con chỉ có thể throw ngoại lệ con của nó hoặc không throw ngoại lệ nào.

**86. Việc tuyên truyền ngoại lệ là gì?**

Việc tuyên truyền ngoại lệ là quá trình thông báo cho lời gọi phương thức hoặc khối mã hiện tại rằng nó cần xử lý ngoại lệ hoặc chuyển tiếp ngoại lệ đến nơi gọi.

**87. Có vấn đề gì không với thứ tự của các lệnh catch được viết cho FileNotFoundException và IOException?**

Vấn đề với thứ tự các lệnh catch cho FileNotFoundException và IOException là FileNotFoundException là một loại con của IOException. Do đó, khi bạn đặt lệnh catch cho FileNotFoundException trước, nó sẽ bắt các ngoại lệ FileNotFoundException trước và các ngoại lệ IOException sẽ không bao giờ được bắt được.

**88. Điểm khác nhau giữa Error và Exception?**

Error là một loại ngoại lệ chỉ ra các vấn đề nghiêm trọng không thể khắc phục được trong quá trình thực thi, trong khi Exception là một loại ngoại lệ chỉ ra các vấn đề có thể được xử lý trong quá trình thực thi.

**89. Ý nghĩa của immutable (bất biến) trong String là gì?**

Trong Java, immutable (bất biến) trong String có nghĩa là một khi một đối tượng String đã được tạo, nó không thể thay đổi. Bất kỳ thay đổi nào đối với đối tượng String sẽ tạo ra một đối tượng mới thay vì thay đổi đối tượng ban đầu.

**90. Có bao nhiêu cách để tạo ra một đối tượng String trong java?**

Các đối tượng String trong Java được thiết kế là immutable để tăng hiệu suất và bảo vệ dữ liệu. Bằng cách này, các đối tượng String có thể được chia sẻ an toàn giữa nhiều luồng mà không cần lo lắng về sự thay đổi không mong muốn.

**91. Có bao nhiêu cách để tạo ra một đối tượng String trong java?**

Có hai cách để tạo một đối tượng String trong Java:

Sử dụng string literal: String str = "Hello";

Sử dụng constructor: String str = new String("Hello");

**93. Tại sao java sử dụng khái niệm string literal?**

Java sử dụng khái niệm string literal để tối ưu bộ nhớ và tăng hiệu suất. Khi bạn sử dụng string literal, JVM sẽ tạo một chuỗi trong String Pool và tái sử dụng chuỗi đó nếu cùng một chuỗi đã tồn tại.

**95. Sự khác nhau giữa String và StringBuffer là gì?**

Sự khác nhau giữa String và StringBuffer:

String là immutable, nghĩa là không thể thay đổi giá trị của chuỗi sau khi đã được tạo.

StringBuffer là mutable, nghĩa là có thể thay đổi giá trị của chuỗi sau khi đã được tạo.

**96. Sự khác nhau giữa StringBuffer và StringBuilder là gì?**

Sự khác nhau giữa StringBuffer và StringBuilder:

StringBuffer là thread-safe, có thể sử dụng trong môi trường đa luồng mà không gây ra vấn đề về đồng bộ hóa.

StringBuilder không thread-safe, hiệu suất cao hơn StringBuffer nhưng không an toàn khi sử dụng trong môi trường đa luồng.

**97. Làm thế nào để tạo lớp immutable trong java?**

Để tạo một lớp immutable trong Java, bạn cần:

Khai báo lớp là final để ngăn chặn việc kế thừa.

Khai báo tất cả các trường dữ liệu là private và final.

Không cung cấp bất kỳ phương thức nào để thay đổi trạng thái của đối tượng.

**98. Mục đích của phương thức toString() trong java là gì?**

Phương thức toString() trong Java được sử dụng để chuyển đổi một đối tượng thành một chuỗi, thường được sử dụng để in ra thông tin của đối tượng để debugging hoặc hiển thị cho người dùng.

**99. Em biết gì về String Pool?**

String Pool là một khu vực trong bộ nhớ heap trong Java được sử dụng để lưu trữ các chuỗi literal. Khi bạn tạo một chuỗi bằng cách sử dụng string literal, JVM sẽ kiểm tra xem chuỗi đó đã tồn tại trong String Pool chưa. Nếu có, nó sẽ tái sử dụng chuỗi đó thay vì tạo một đối tượng chuỗi mới.

**100. Nested class (lớp lồng nhau) là gì?**

Nested class (lớp lồng nhau) trong Java là một lớp được định nghĩa bên trong một lớp hoặc một phương thức của một lớp khác. Có hai loại nested class: static nested class và inner class (non-static nested class).

**101. Có sự khác nhau giữa nested class và inner class không?**

Có, có sự khác nhau giữa nested class và inner class:

Nested class là một lớp được định nghĩa bên trong một lớp khác.

Inner class là một loại nested class đặc biệt, có thể truy cập các thành viên (fields và methods) của lớp bao bọc nó.

**102. Nested interface là gì?**

Nested interface là một interface được định nghĩa bên trong một lớp hoặc một interface khác.

**103. Có thể khai báo interface trong class không?**

Có, bạn có thể khai báo một interface trong một class trong Java.

**104. Có thể khai báo một class trong interface không?**

Không, không thể khai báo một class trong một interface trong Java.

**105. Sự khác nhau giữa Array và ArrayList là gì?**

Sự khác nhau giữa Array và ArrayList:

Array là một cấu trúc dữ liệu cố định với kích thước cố định, trong khi ArrayList là một lớp trong Java Collections Framework, có thể mở rộng tự động khi cần.

**106. Lợi thế của ArrayList so với các mảng?**

Lợi thế của ArrayList so với các mảng:

Có thể thay đổi kích thước linh hoạt: ArrayList có thể tự động mở rộng khi cần.

Hỗ trợ các phương thức tiện ích: ArrayList cung cấp các phương thức sẵn có để thêm, xóa, tìm kiếm, sắp xếp và truy cập phần tử một cách dễ dàng.

**107. Hoạt động xóa trong LinkedList là nhanh hơn trong ArrayList, tại sao?**

Hoạt động xóa trong LinkedList là nhanh hơn trong ArrayList vì khi xóa một phần tử trong LinkedList, chỉ cần cập nhật liên kết giữa các nút, trong khi khi xóa một phần tử trong ArrayList, các phần tử phía sau phải dịch chuyển sang trái.

**108. Bạn quyết định khi nào sử dụng ArrayList và LinkedList?**

Bạn nên sử dụng ArrayList khi:

Cần truy cập ngẫu nhiên nhanh chóng đến các phần tử.

Cần thêm hoặc xóa phần tử ở cuối danh sách thường xuyên.

Không cần đồng bộ hóa.

Bạn nên sử dụng LinkedList khi:

Cần thêm hoặc xóa phần tử ở đầu hoặc giữa danh sách thường xuyên.

Cần thao tác chèn và xóa hiệu quả.

**109. Sự khác nhau giữa ArrayList và LinkedList là gì?**

Sự khác nhau giữa ArrayList và LinkedList:

ArrayList sử dụng một mảng để lưu trữ các phần tử, trong khi LinkedList sử dụng một danh sách liên kết (linked list).

Truy cập ngẫu nhiên nhanh chóng hơn trong ArrayList, trong khi thao tác chèn và xóa nhanh chóng hơn trong LinkedList.

**110. Sự khác nhau giữa ArrayList và Vector là gì?**

Sự khác nhau giữa ArrayList và Vector:

ArrayList không đồng bộ (non-synchronized), trong khi Vector là đồng bộ (synchronized).

ArrayList không an toàn để sử dụng trong môi trường đa luồng, trong khi Vector có thể được sử dụng an toàn.

**111. Sự khác nhau giữa Iterator và ListIterator là gì?**

Sự khác nhau giữa Iterator và ListIterator:

Iterator là một giao diện trong Java Collections Framework được sử dụng để lặp qua các phần tử của một tập hợp (collection).

ListIterator là một giao diện con của Iterator, chỉ dùng cho các cấu trúc dữ liệu danh sách (list), cho phép điều hướng đến phần tử trước hoặc sau vị trí hiện tại và thực hiện các thao tác chỉnh sửa.

**112. Sự khác nhau giữa List và Set là gì?**

Sự khác nhau giữa List và Set:

List là một tập hợp có thứ tự của các phần tử, cho phép phần tử trùng lặp và truy cập bằng chỉ số.

Set là một tập hợp không có thứ tự của các phần tử, không cho phép phần tử trùng lặp và không có cách truy cập bằng chỉ số.

**113. Sự khác nhau giữa Set và Map là gì?**

Sự khác nhau giữa Set và Map:

Set là một tập hợp các phần tử không có thứ tự và không cho phép phần tử trùng lặp.

Map là một tập hợp các cặp key-value, trong đó mỗi key duy nhất được ánh xạ tới một value.

**114. Sự khác nhau giữa Stack và Queue là gì?**

Sự khác nhau giữa Stack và Queue:

Stack là một cấu trúc dữ liệu dạng ngăn xếp (LIFO - Last In, First Out), trong đó phần tử cuối cùng được thêm vào là phần tử đầu tiên được lấy ra.

Queue là một cấu trúc dữ liệu dạng hàng đợi (FIFO - First In, First Out), trong đó phần tử đầu tiên được thêm vào là phần tử đầu tiên được lấy ra.

**115. Sự khác nhau giữa Collection và Collections là gì?**

Sự khác nhau giữa Collection và Collections:

Collection là một giao diện trong Java Collections Framework, đại diện cho một nhóm các đối tượng.

Collections là một lớp tiện ích trong Java cung cấp các phương thức static để thao tác với các cấu trúc dữ liệu collection.

**116. Sự khác nhau giữa Comparable và Comparator là gì?**

Sự khác nhau giữa Comparable và Comparator:

Comparable là một interface trong Java được sử dụng để so sánh các đối tượng của cùng một loại dựa trên một tiêu chí nhất định.

Comparator là một interface trong Java được sử dụng để so sánh các đối tượng của bất kỳ loại nào dựa trên một tiêu chí nhất định không phải là tiêu chí mặc định.

**117. Phương thức hashCode() là gì?**

Phương thức hashCode() trong Java được sử dụng để trả về mã hash của một đối tượng. Mã hash này thường được sử dụng trong việc xây dựng các cấu trúc dữ liệu như bảng băm để tìm kiếm nhanh.

**118. Tại sao chúng ta phải ghi đè phương thức equals()?**

Chúng ta phải ghi đè phương thức equals() để so sánh hai đối tượng dựa trên nội dung của chúng thay vì so sánh tham chiếu đến đối tượng trong bộ nhớ.

**119. Làm thế nào để đồng bộ List, Set và Map?**

Để đồng bộ List, Set và Map trong Java, bạn có thể sử dụng các phương thức trong lớp Collections để lấy các phiên bản đồng bộ hóa của chúng, như synchronizedList(), synchronizedSet(), và synchronizedMap().

**120. Lợi ích của generic collection là gì?**

Lợi ích của generic collection là:

Kiểm tra kiểu tại compile time giúp tránh được lỗi kiểu (type errors) ở runtime.

Tăng tính linh hoạt và tái sử dụng code.

Tạo code dễ đọc hơn và ít lỗi hơn.

**121. Tại sao Generic được sử dụng trong Java?**

Generic được sử dụng trong Java để cung cấp kiểm tra kiểu tại compile time và tạo code linh hoạt và tái sử dụng.

**122. Bạn hiểu gì về Access Modifier?**

Trong Java, Access Modifier là các từ khóa được sử dụng để kiểm soát quyền truy cập vào các thành phần của lớp như biến, phương thức và lớp con. Các Access Modifier bao gồm: public, protected, default (không có từ khóa), và private.

**123. Phân biệt public, protected, private, default?**

Phân biệt giữa public, protected, private và default:

public: Các thành phần được khai báo public có thể truy cập từ bất kỳ đâu, không phụ thuộc vào package hoặc lớp con.

protected: Các thành phần được khai báo protected chỉ có thể truy cập từ cùng một package hoặc từ lớp con của lớp đó (kể cả các lớp con ở các package khác).

private: Các thành phần được khai báo private chỉ có thể truy cập từ bên trong cùng một lớp.

default (không có từ khóa): Các thành phần không được khai báo với bất kỳ từ khóa truy cập nào chỉ có thể truy cập từ cùng một package.

***124.* Bạn hiểu gì về Đa luồng (Multi-Thread)?**

Đa luồng (Multi-Thread) là một kỹ thuật trong lập trình mà cho phép một chương trình thực thi nhiều luồng đồng thời. Mỗi luồng có thể thực hiện các tác vụ riêng biệt mà không cần chờ đợi cho đến khi các tác vụ khác hoàn thành.

**125. Phân biệt Multitasking, multiprocessing, multithreading**

Phân biệt giữa Multitasking, multiprocessing và multithreading:

Multitasking: Là khả năng của hệ điều hành để thực thi nhiều tác vụ (processes) cùng một lúc.

Multiprocessing: Là khả năng của hệ điều hành để thực thi nhiều tiến trình (processes) cùng một lúc trên nhiều CPU hoặc core.

Multithreading: Là khả năng của một chương trình để thực thi nhiều luồng (threads) cùng một lúc trong một tiến trình.

**126. Thread có thể được tạo bằng hai cách nào? So sánh**

Thread có thể được tạo bằng hai cách:

Kế thừa lớp Thread: Tạo một lớp con kế thừa từ lớp Thread và ghi đè phương thức run().

Triển khai giao diện Runnable: Triển khai interface Runnable và truyền instance của lớp triển khai này vào constructor của Thread.

**127. Phân biệt run(), start()**

Phân biệt giữa run() và start():

run(): Phương thức run() chứa mã nguồn của luồng, được gọi khi gọi phương thức start() của luồng.

start(): Phương thức start() được sử dụng để bắt đầu thực thi luồng, nó sẽ gọi phương thức run() của luồng.

**128. Vòng đời Thread**

Vòng đời Thread bao gồm các trạng thái: New, Runnable, Running, Blocked/Waiting, Terminated.

**129. Giải thích trình dọn rác (Garbage Collector) trong Java?**

Trình dọn rác (Garbage Collector) trong Java là một thành phần của JVM được sử dụng để tự động thu hồi bộ nhớ không sử dụng (garbage) để giải phóng tài nguyên và tránh lỗi memory leak.

**130. Các cách mà một Thread có thể đi vào trạng thái đợi (waiting state)?**

Một Thread có thể vào trạng thái đợi (waiting state) thông qua các cách sau:

Gọi phương thức sleep().

Gọi phương thức join() trên một Thread khác.

Sử dụng phương thức wait() và notify() trong synchronized block.

**131. Cách mà Đa luồng (Multi-Thread) diễn ra trên một máy tính với một CPU đơn?**

Trên một máy tính với một CPU đơn, Đa luồng (Multi-Thread) sẽ được thực thi một cách ngẫu nhiên, hệ điều hành sẽ lựa chọn luồng nào được thực thi dựa trên các tiêu chí như tiêu tốn ít CPU nhất hoặc ưu tiên cao nhất.

**132. Một Thread đã chết có thể được restart lại như thế nào?**

Một Thread đã chết không thể được restart lại. Nếu bạn muốn thực hiện một công việc tương tự sau khi một Thread đã kết thúc, bạn cần tạo một Thread mới.

**133. Điểm khác nhau giữa yield và sleep?**

Sự khác nhau giữa yield() và sleep():

yield(): Cho phép một luồng khác có cùng hoặc cao hơn độ ưu tiên có cơ hội thực thi.

sleep(): Tạm dừng thực thi của luồng hiện tại trong một khoảng thời gian nhất định.

**134. Phương thức sleep() và wait() khác nhau ở điểm nào?u**

Phương thức sleep() và wait() khác nhau ở điểm:

sleep(): Là phương thức static trong lớp Thread, tạm dừng thực thi của luồng hiện tại.

wait(): Là phương thức của lớp Object, tạm dừng thực thi của luồng hiện tại và giải phóng khóa của đối tượng.