**Câu 1: Sự khác nhau giữa JDK,JRE và JVM?**

JDK: Java Development Kit.  được sử dụng để tạo các applet và ứng dụng Java. JDK giúp viết mã và chạy các chương trình Java

JRE: Java Runtime Environment. là một gói phần mềm cho phép chương trình Java chạy. Nó chứa các thư viện lớp (class libraries), lớp bộ nạp (loader class) và JVM

JVM: Java Virtual Machine: máy ảo Java: chức năng để compile 1 class Java thành file .java để chạy. Nó chuyển đổi Java bytecode thành ngôn ngữ máy. Nó cung cấp các chức năng java cơ bản như quản lý bộ nhớ, bảo mật, thu gom rác

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

**Stack** là một vùng nhớ được sử dụng để lưu trữ các tham số và các biến local của phương thức mỗi khi một phương thức được gọi ra.

**Heap** là một vùng nhớ trong bộ nhớ được sử dụng để lưu trữ các đối tượng khi từ khóa new được gọi ra, các biến static và các biến toàn cục (biến instance).

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

Java code được biên dịch thành một byte code, là ngôn ngữ trung gian giữa mã nguồn và mã máy. Byte code này không phải là nền tảng cụ thể và do đó có thể được thông dịch cho bất kỳ nền tảng nào.

**Câu 7: Classloader trong java là gì?**

Classloader là một hệ thống con của JVM được sử dụng để tải các lớp và các interface.

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

Các biến local không được khởi tạo với bất kỳ giá trị mặc định nào, bất kể là nguyên thủy hay tham chiếu đối tượng.

**Câu 13: Khác nhau Biến tham trị và Biến tham chiếu?**

|  |
| --- |
| **Câu 14: Bạn hiểu gì về Đối tượng?**  Đối tượng là thực thể có thuộc tính và hành vi |
| **Câu 15: Định nghĩa Lớp (class)?**  Lớp là một nhóm các đối tượng có đặc điểm chung |
|  |
| **Câu 17: Biến local hay biến cục bộ là gì?**  Biến được khai báo trong phương thức |
| **Câu 18: Biến instance là gì?**  Biến trong 1 class và ngoài phương thức |
| **Câu 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 dựa trên đối tượng thực hiện theo tất cả các tính năng của OOPs ngoại trừ tính kế thừa |
| **Câu 20: Giá trị khởi tạo của biến tham chiếu đối tượng được định nghĩa là biến instance là gì?**  null |
| **Câu 21: Constructor là gì?**  một phương thức được sử dụng để khởi tạo một đối tượng |
| **Câu 22: Mục đích của constructor là gì?**  Constructor mặc định cung cấp các giá trị mặc định cho các đối tượng. Trình biên dịch java tạo ra một constructor mặc định chỉ khi không có constructor nào trong lớp. |
| **Câu 23: Constructor trả về kiểu giá trị gì?**  Object |
| **Câu 24: Constructor được kế thừa không?**  Không |
| **Câu 25: Có thể tạo constructor final không?**  Không, có thể overload |
| **Câu 26: Biến static là gì?**  Các biến được khai báo với từ khóa static tham chiếu đến thuộc tính chung cho tất cả các đối tượng của lớp. |
| **Câu 27: Phương thức static là gì?**  Một phương thức static thuộc lớp chứ không phải đối tượng của lớp.  Một phương thức static gọi mà không cần tạo một instance của một lớp.  Phương thức static có thể truy cập biến static và có thể thay đổi giá trị của nó. |
| **Câu 28: Tại sao phương thức main là static?**  Bởi vì không cần thiết phải tạo đối tượng để gọi phương thức static. Nếu nó là phương thức non-static, JVM đầu tiên tạo đối tượng và sau đó gọi phương thức main() mà có thể gây ra vấn đề về cấp phát bộ nhớ bộ nhớ phụ. |
| **Câu 29: Khối static là gì?**  Được sử dụng để khởi tạo thành viên dữ liệu static.  Nó được thực thi trước phương thức main tại lúc tải lớp |
| **Câu 30: Chúng ta có thể thực thi một chương trình không có phương thức main() không?**  Có, một cách khác là sử dụng khối static |
| **Câu 31: Chuyện gì xảy ra khi phương thức main không có static?**  Chương trình được biên dịch, nhưng gặp lỗi ngoại lệ tại runtime là "NoSuchMethodError". |
| **Câu 32: Sự khác nhau giữa phương thức static và phương thức instance?**   * Phương thức non-static thì gọi được các biến static và non-static * Phương thức static thì không gọi được biến non-static |
| **Câu 33: this trong java là gì?**  this là từ khóa trong java tham chiếu đến đối tượng hiện tại. |
| **Câu 34: Kế thừa là gì?**  Kế thừa là một cơ chế trong đó một đối tượng được thừa hưởng tất cả thuộc tính và phương thức của đối tượng khác của lớp khác. Nó được sử dụng để tái sử dụng code và ghi đè phương thức. |
| **Câu 35: Lớp nào là lớp cha cho tất cả các lớp.** |

Object

|  |  |
| --- | --- |
| 36. Tại sao đa kế thừa không được hỗ trợ trong java. | Để giảm thiểu sự phức tạp và đơn giản hóa ngôn ngữ, đa kế thừa không được support trong java |
| 37. Composition là gì? | Khai báo biến tham chiếu của một class trong một class khác được gọi là composition(sự hợp thành). |
| 38. Sự khác nhau giữa aggregation và composition? | Aggregation biểu diễn mối quan hệ yếu, còn composition biểu diễn quan hệ chặt chẽ. Ví dụ: xe máy có một công tơ mét (aggregation), nhưng xe máy có một động cơ (composition). |
| 39. Tại sao java không support con trỏ? | Con trỏ là một biến tham chiếu tới địa chỉ ô nhớ. Nó không được sử dụng trong java vì nó không an toàn và phức tạp. |
| 40. Super trong java là gì ? | -Nó là một từ khóa mà tham chiếu trực tiếp đến biến instance của lớp cha gần nhất.  -super() được sử dụng để gọi trực tiếp Constructor của lớp cha  -Từ khóa super được sử dụng để gọi trực tiếp phương thức của lớp cha. |

|  |  |
| --- | --- |
| 41. Có thể sử dụng cả this() và super() trong một constructor ? | Không, vì this() gọi đến một constructor khác trong lớp hiện tại, còn super() gọi constructor của lớp cha. |
| 42. Object cloning là gì ? | Object cloning được sử dụng để tạo ra một bản sao giống hệt đối tượng ban đầu. |
| 43. Overloading (nạp chồng) phương thức là gì ? | Overloading là nạp chồng phương thức, đơn giản là có vài phương thức trùng tên nhưng khác nhau về tham số trong cùng 1 class, dùng để thu được tính đa hình của lớp |
| 44.  Tại sao overloading phương thức không xảy ra khi thay đổi kiểu giá trị trả về ? | Bởi vì đó là sự không rõ ràng, không biết gọi phương thức nào khi runtime. |
| 45. Có thể overload phương thức main() không ? | Có, có thể overload phương thức main(). |
| 46. Ghi đè (overriding) phương thức là gì ? | -Cơ chế cho phép lớp con định nghĩa lại các phương thức đã được định nghĩa trước đó ở lớp cha.  -Phương thức override ở lớp con có cùng tên, cùng danh sách tham số và kiểu dữ liệu trả về so với phương thức ở lớp cha  -Phương thức ở lớp con phải có access modifier có level bằng hoặc cao hơn so với phương thức ở lớp cha  -@Override có thể sử dụng để đánh dấu phương thức ghi đè |
| 47. Có thể ghi đè phương thức static không ? | Bạn không thể ghi đè phương thức static vì chúng thuộc về class chứ không thuộc về đối tượng. |
| 48. Tại sao không thể ghi đè phương thức static? | Bởi vì phương thức static là một phần của lớp và nó bị ràng buộc với lớp, trong khi phương thức instance là bị ràng buộc với đối tượng, static được lưu trong vùng nhớ Class và instance được lưu trong bộ nhớ heap. |
| 49. Có thể ghi đè phương thức đã nạp chồng ? | Có. |
| 50. Có thể ghi đè biến instance không ? | Không. |
| 51. Sự khác nhau giữa nạp chồng và ghi đè là gì ? | Nạp chồng phương thức (Overloading):   * Được sử dụng để giúp code của chương trình dễ đọc hơn * Nạp chồng được thực hiện bên trong một class. * Nạp chồng phương thức thì tham số phải khác nhau * Nạp chồng phương thức là ví dụ về đa hình lúc biên dịch * Trong java, nạp chồng phương thức không thể được thực hiện khi chỉ thay đổi kiểu giá trị trả về của phương thức. Kiểu giá trị trả về có thể giống hoặc khác. *Giá trị trả về có thể giống hoặc khác*, nhưng tham số phải khác nhau.   Ghi đè phương thức (overriding):   * Được sử dụng để cung cấp cài đặt cụ thể cho phương thức được khai báo ở lớp cha * Ghi đè phương thức xảy ra trong 2 class có quan hệ kế thừa * Ghi đè phương thức thi tham số phải giống nhau * Ghi đè phương thức là ví dụ về đa hình lúc runtime * Giá trị trả về phải giống nhau |
| 52.  Biến final là gì ? | Từ khóa final được dùng để tạo hằng số, có nghĩa là nó được gán một lần duy nhất. |
| 53. Phương thức final là gì ? | Phương thức không thể được ghi đè. |
| 54. Lớp final là gì ? | Lớp không thể được kế thừa. |
| 55. Biến final blank là gì ? | Một biến final không được khởi tạo giá trị lúc khai báo được gọi là biến final blank. |
| 56.  Có thể khởi tạo giá trị cho biến final blank không ? | Có, nếu biến đó là non-static thì chỉ khởi tạo được trong constructor. Nếu biến đó là static thì chỉ khởi tạo được trong khối static. |
| 57. Có thể khai báo phương thức main là final không ? | Có, giống như: public static final void main(String[] args){} |
| 58. Đa hình tại runtime là gì ? | Đa hình tại runtime là quá trình gọi phương thức đã được ghi đè trong thời gian thực thi chương trình. Trong quá trình này, một phương thức được ghi đè được gọi thông qua biến tham chiếu của một lớp cha.  Ví dụ: Upcasting, Downcasting |
| 59. Có thể thực hiện đa hình lúc runtime với các thành viên dữ liệu không ? | Không. |
| 60. Sự khác nhau giữa ràng buộc tĩnh và ràng buộc động là gì ? | Kiểu ràng buộc tĩnh của đối tượng được xác định tại lúc biên dịch, còn kiểu ràng buộc động của đối tượng được xác định tại runtime. |

|  |  |
| --- | --- |
| 61. Trừu tượng là gì ? | Tính trừu tượng là một tiến trình ẩn các cài đặt chi tiết và chỉ hiển thị tính năng tới người dùng.  Nói cách khác, nó chỉ hiển thị các thứ quan trọng tới người dùng và ẩn các chi tiết nội tại, ví dụ: để gửi tin nhắn, người dùng chỉ cần soạn text và gửi tin. Bạn không biết tiến trình xử lý nội tại về phân phối tin nhắn.  Sự trừu tượng khiến bạn tập trung vào đối tượng đó làm những gì thay vì đối tượng đó làm như thế nào. |
| 62. Sự khác nhau giữa trừu tượng và đóng gói là gì ? | Trừu tượng là ẩn đi cài đặt chi tiết còn đóng gói là gói code và data vào một khối duy nhất. |
| 63. Trừu tượng là gì ? | Một lớp được khai báo với từ khóa abstract là lớp abstract trong Java. Lớp abstract có nghĩa là lớp trừu tượng, nó có thể có các phương thức abstract hoặc non-abstract.  Có 2 cách để đạt được sự trừu tượng hóa trong java:   1. Sử dụng lớp abstract 2. Sử dụng interface |
| 64. Có phương thức trừu tượng không nằm trong lớp trừu tượng không ? | Không, nếu có bất kỳ phương thức trừu tượng nào trong lớp, thì lớp đó phải là lớp trừu tượng. |
| 65. Có thể sử dụng cả abstract và final cho một phương thức không ? | Không, vì phươngthức trừu tượng (abstract) cần phải được ghi đè, trong khi đó không thể ghi đè được phương thức final. |
| 66. Có thể tạo thể hiện của lớp trừu tượng không ? | Không, lớp trừu tượng không có thể hiện. |
| 67. Interface là gì ? | Là một bản thiết kế các phương thức (hàm) hoặc có thể hiểu dễ hơn nữa nó là một khuôn mẫu cố định mà khi nhìn vào nó chúng ta có thể biết được những hằng, phương thức mà class implements nó chắc chắn phải triển khai.  Các tính chất quan trọng của interface:   * Interface không phải là một lớp * Trong interface chỉ được khai báo hằng không khai báo biến * Các phương thức trong interface chỉ được gọi tên chứ không được định nghĩa * Không thể khởi tạo đối tượng từ interface * Các interface có thể kế thừa lẫn nhau * Một class có thể implements nhiều interface |
| 68. Có thể khai báo một phương thức của interface với từ khóa static không? | Không. Vì mặc định các phương thức của một interface là trừu tượng, từ khóa static và abtract không thể được sử dụng chung với nhau. |
| 69. Một interface có thể là final không? | Không. Vì cài đặt phải được cung cấp bởi một lớp khác. |
| 70. Marker interface là gì ? | Một giao tiếp (interface) không có thành viên dữ liệu và phương thức được biết đến như là một giao tiếp marker. Ví dụ Serializable, Cloneable, ... |
| 71. Sự khác nhau giữa lớp abstract và interface là gì? | Lớp Abstract:   * Một lớp abstract có thể có phương thức, có nội dung (các phương thức tường minh). * Một lớp abstract có thể có các biến instance. * Một lớp abstract có thể có constructor * Một lớp abstract có thể có các phương thức static. * Một lớp chỉ có thể extends một lớp abstract.   Interface:   * Interface chỉ có các phương thức trừu tượng. * Một interface không thể có các biến instance. * Một interface không thể có constructor. * Một interface không thể có các phương thức static. * Một lớp có thể implement nhiều interface. |
| 72. Có thẻ định nghĩa private hoặc protected cho các biến trong interface không? | Không. chúng phải là public. |
| 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 implements interface được tham chiếu. |
| 74. Package là gì? | -Java sử dụng package để tránh sự xung đột trong điều khiển truy cập các lớp.  -Một package có thể hiểu như là một tập hợp các thứ như: các lớp, các package con. |
| 75. Có cần import package import java.lang không? tại sao? | Không. Vì nó được tải ngầm bởi JVM. |
| 76. Có thể import package/lớp giống nhau hai lần không? JVM sẽ tải package hai lần khi chạy không? | Chúng ta có thể import package/lớp giống nhau nhiều lần, trình biên dịch và JVM sẽ không phàn nàn gì về vấn đề này. Nhưng JVM sẽ tải lớp đó một lần. |
| 77. Static import là gì? | Với static import, chúng ta có thể truy cập đến các thành viên của một lớp một cách trực tiếp. Không cần phải sử dụng tên class cũng có thể truy cập. |
| 78. Xử lý ngoại lệ (handling exception) là gì? | **-Exception Handling** trong java hay xử lý ngoại lệ trong java là một cơ chế mạnh mẽ để xử lý các lỗi runtime để có thể duy trì luồng bình thường của ứng dụng.  -Exception Handling (xử lý ngoại lệ) là một cơ chế xử lý các lỗi runtime như ClassNotFound, IO, SQL, Remote, vv |
| 79.Sự khác biệt giữa checked exception và unchecked exception là gì? | Checked exception:  Các lớp được extends lớp Throwable ngoại trừ RuntimeException và Error được gọi là các checked exception. Ví dụ: IOException, SQLException, ... Checked exception được kiểm tra tại thời gian biên dịch.  Unchecked exception:  Các lớp được extends lớp RuntimeException được gọi là unchecked exception. Ví dụ như ArithmeticException, NullPointerException, ... Unchecked exception không được kiểm tra tại thời gian biên dịch. |
| 80. Có phải mỗi khối try phải đi kèm với một khối catch? | Nó không phải là cần thiết mà mỗi khối thử phải được theo sau bởi một khối catch. Nó phải được theo sau bởi một khối [catch](https://viettuts.vn/exception-handling/khoi-lenh-try-catch-trong-java) hoặc một khối [finally](https://viettuts.vn/exception-handling/khoi-lenh-finally-trong-java). Và bất kỳ trường hợp ngoại lệ có thể bị ném sẽ được khai báo với [từ khóa throws](https://viettuts.vn/exception-handling/tu-khoa-throws-trong-java) của phương thức. |

|  |
| --- |
| **Câu 81: Khối finally là gì?**  Khối lệnh finally trong java luôn được thực thi cho dù có ngoại lệ xảy ra hay không. |
| **Câu 82: Khối finally có thể được sử dụng mà không cần khối catch không?**  Có. Bởi khối try. Khối finally phải theo sau khối try hoặc catch. |
| **Câu 83: Có trường hợp nào khối finally không được thực thi không?**  Khối finally không được thực thi nếu chương trình bị thoát (bằng cách gọi System.exit() |
| **Câu 84: Sự khác nhau giữa throw và throws là gì?**   |  |  |  | | --- | --- | --- | | No | Throw | Throws | | 1) | Từ khóa throw trong java được sử dụng để ném ra một ngoại lệ rõ ràng. | Từ khóa throws trong java được sử dụng để khai báo một ngoại lệ. | | 2) | Ngoại lệ checked không được truyền ra nếu chỉ sử dụng từ khóa throw. | Ngoại lệ checked được truyền ra ngay cả khi chỉ sử dụng từ khóa throws. | | 3) | Sau throw là một instance. | Sau throws là một hoặc nhiều class. | | 4) | Throw được sử dụng trong phương thức. | Throws được khai báo ngay sau dấu đóng ngoặc đơn của phương thức. | | 5) | Bạn không thể throw nhiều exceptions. | Bạn có thể khai báo nhiều exceptions, Ví dụ: public void method()throws IOException,SQLException. | |
| **Câu 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?**  Được nhưng chỉ đối với unchecked exception còn checked exception thì không. |
| **Câu 86: Việc tuyên truyền ngoại lệ là gì?**  Việc chuyển tiếp đối tượng ngoại lệ sang phương thức gọi được gọi là sự truyền ngoại lệ |
| **Câu 87: Có vấn đề gì không với thứ tự của các lệnh catch được viết cho FileNotFoundException và IOException?**  Có **FileNotFoundException** kế thừa IOException. Lớp con của exception phải được bắt đầu tiên |
| **Câu 88: Điểm khác nhau giữa Error và Exception?**  - Error là lỗi của chương trình, còn exception là lỗi chương trình được phát hiện và có thể xử lý. |
| **Câu 89: Ý nghĩa của immutable (bất biến) trong String là gì?**  Ý nghĩa đơn giản của immutable là không thể sửa đổi hoặc không thể thay đổi được. Một khi đối tượng String đã được tạo ra, giá trị của nó không thể thay đổi |
| **Câu 90: Tại sao các đối tượng String trong java là immutable?**  Bởi vì java sử dụng khái niệm về string literal. Giả sử có 5 biến tham chiếu, tất cả các tham chiếu đến một đối tượng "hello". Nếu một biến tham chiếu thay đổi giá trị của đối tượng, nó sẽ bị ảnh hưởng đến tất cả các biến tham chiếu khác. Đó là lý do tại sao đối tượng String trong java immutable (bất biến). |
| **Câu 91: Có bao nhiêu cách để tạo ra một đối tượng String trong java?**  Có 2 cách để tạo đối tượng String:  Sử dụng string literal  Sử dụng từ khóa new |
| **Câu 92: Có bao nhiêu đối tượng String được tạo ra trong đoạn code sau?** |
| **Câu 93: Tại sao java sử dụng khái niệm string literal?**  Vì điều này khiến Java sử dụng bộ nhớ hiệu quả hơn (vì không có đối tượng String mới được tạo ra trong string constant pool). |
| **Câu 94: Có bao nhiêu đối tượng được tạo ra trong đoạn code sau?** |
| **Câu 95: Sự khác nhau giữa String và StringBuffer là gì?**  string là một đối tượng immutable (bất biến). StringBuffer là môt đối tượng mutable (có thể biến đổi). |
| **Câu 96: Sự khác nhau giữa StringBuffer và StringBuilder là gì?**  StringBuffer là đồng bộ còn StringBuilder là KHÔNG đồng bộ. |
| **Câu 97: Làm thế nào để tạo lớp immutable trong java?**  Chúng ta có thể tạo ra lớp immutable như lớp String bằng cách định lớp với từ khóa final |
| **Câu 98: Mục đích của phương thức toString() trong java là gì?**  Phương thức toString() trả về biểu diễn chuỗi của bất kỳ đối tượng nào. |
| **Câu 99: Em biết gì về String Pool?** |
| **Câu 100: Nested class (lớp lồng nhau) là gì?**  Một lớp được khai báo bên trong một lớp khác được gọi là lớp lồng nhau. Có 4 loại lớp bên trong lớp lồng nhau: member inner class, local inner class, annonymous inner class và static nested. |
| **Câu 101: Có sự khác nhau giữa nested class và inner class không?**  Có. Inner class là non-static nested class nghĩa là inner class là một kiểu nested class. |
| **Câu 102: Nested interface là gì?**  Bất kỳ interface nào được khai báo bên trong interface hoặc class, được biết đến nested interface. Nó có thể là static hoặc default. |
| **Câu 103: Có thể khai báo interface trong class không?**  Có. Nó được biết đến như nested interface. |
| **Câu 104: Có thể khai báo một class trong interface không?**  Có. Nhưng chúng không liên quan đến nhau. |
| **Câu 105: Sự khác nhau giữa Array và ArrayList là gì?**   |  |  |  | | --- | --- | --- | |  | **Array** | **ArrayList** | | 1 | 1) Kích thước **cố định**. | Kích thước có thể **thay đổi được**. | | 2 | 2) Có thể lưu trữ dữ liệu kiểu **nguyên thủy** và **đối tượng**. | Chỉ có thể lưu trữ dữ liệu kiểu **đối tượng**. Kể từ Java 5, kiểu nguyên thủy được tự động chuyển đổi trong các đối tượng được gọi là **auto-boxing**. | | 3 | 3) Tốc độ lưu trữ và thao tác **nhanh hơn**. | Tốc độ lưu trữ vào thao tác **chậm hơn**. | | 4 | 4) Chỉ có thuộc tính **length**. | Có nhiều phương thức để thao tác với dữ liệu. | |
| **Câu 106: Lợi thế của ArrayList so với các mảng?**   * Kích thước của ArrayList có thể thay đổi tăng hoặc giảm tùy vào số lượng phần tử trong nó. * Thêm và xóa phần tử ở bất kì vị trí nào trong mảng một cách nhanh chóng, hiệu quả. * Có nhiều phương thức hỗ trợ các thao tác như removeAll-xóa tất cả, searching-tìm kiếm, iterations-duyệt mảng, retainAll-sửa phần tử,… * Một ArrayList có thể chứa nhiều loại đối tượng như Integer, String, Scanner, hay một đối tượng do người dùng tự khai báo. |
| **Câu 107: Hoạt động xóa trong LinkedList là nhanh hơn trong ArrayList, tại sao?**  Thao tác thêm và xóa phần tử với LinkedList nhanh hơn ArrayList. Bởi vì nó không cần sắp xếp lại các phần tử sau khi thêm hoặc xóa. Nó chỉ cần cập nhật lại tham chiếu tới phần tử phía trước và sau nó |
| **Câu 108: Bạn quyết định khi nào sử dụng ArrayList và LinkedList?**  Add, delete thì dùng linkedList  Duyệt thì dùng arrayList |
| **Câu 109: Sự khác nhau giữa ArrayList và LinkedList là gì?**   |  |  |  | | --- | --- | --- | | No. | ArrayList | LinkedList | | 1) | ArrayList sử dụng một mảng động. | LinkedList sử dụng danh sách liên kết doubly. | | 2) | ArrayList không hiệu quả với thao tác vì cần nhiều chuyển đổi. | LinkedList là hiệu quả cho thao tác. | | 3) | ArrayList là tốt hơn để lưu trữ và lấy dữ liệu. | LinkedList là tốt hơn để thao tác dữ liệu. | |
| **Câu 110: Sự khác nhau giữa ArrayList và Vector là gì?** |
| **Câu 111: Sự khác nhau giữa Iterator và ListIterator là gì?** |
| **Câu 112: Sự khác nhau giữa List và Set là gì?**  List có thể chứa các phần tử trùng lặp (dublicate), trong khi Set chỉ chứa các phần tử duy nhất. |
| **Câu 113: Sự khác nhau giữa Set và Map là gì?**  Set chỉ chứa giá trị, trong khi Map chứa cặp key và value. |
| **Câu 114: Sự khác nhau giữa Stack và Queue là gì?**  Stack: FILO  Queue: FIFO |
| **Câu 115: Sự khác nhau giữa Collection và Collections là gì?** |
| **Câu 116: Sự khác nhau giữa Comparable và Comparator là gì?**   |  |  |  | | --- | --- | --- | | **No.** | **Comparable** | **Comparator** | | 2) | Nó cung cấp phương thức compareTo(). | Nó cung cấp phương thức compare(). | | 3) | Nó được đặt trong java.lang package. | Nó được đặt trong java.util package. | | 4) | implement Comparable interface, thì lớp đó phải được sửa đổi. | Lớp không bị sửa đổi. | |
| **Câu 117: Phương thức hashCode() là gì?**  Phương thức hashCode() trả về một giá trị mã băm (một số nguyên). |
| **Câu 118: Tại sao chúng ta phải ghi đè phương thức equals()?**  Phương thức equals() được sử dụng để kiểm tra xem hai đối tượng có giống nhau hay không. Nó cần phải được ghi đè nếu chúng ta muốn kiểm tra các đối tượng dựa trên thuộc tính của chúng. |
| **Câu 119: Làm thế nào để đồng bộ List, Set và Map?** |
| **Câu 120: Lợi ích của generic collection là gì?**  Nếu chúng ta sử dụng lớp generic, chúng ta không cần typecasting. Nó là typesafe và kiểm tra tại thời gian biên dịch. |
| **Câu 121: Tại sao Generic được sử dụng trong Java?**  Java Generics cho phép chúng ta viết một phương thức duy nhất có thể thực hiện các hoạt động trong nhiều loại đối tượng hỗ trợ phương thức đó. Sử dụng các lớp và phương thức Java Generic, lập trình viên có thể chỉ định một tập hợp các phương thức có liên quan với một khai báo phương thức chung / chung hoặc với một khai báo một lớp. |
| **Câu 122: Bạn hiểu gì về Access Modifier?**  Có hai loại Access Modifier trong Java, đó là: Access Modifier và Non-access Modifier.  Access Modifer trong Java xác định phạm vi có thể truy cập của biến, phương thức, constructor hoặc lớp. |
| **Câu 123: Phân biệt public, protected, private, default?**  - Access modifier: 3 access: private, protected, public  + private: trong class  + default (không có từ khóa): trong package  + protected: trong package; ngoài package nhưng là trong lớp con  + public: toàn bộ trong src |
| **Câu 124: Bạn hiểu gì về Đa luồng (Multi-Thread)?**  Trong Java, các luồng được quản lý bởi máy ảo Java (JVM). Đa luồng – Multi Thread là một tiến trình thực hiện nhiều luồng đồng thời. Trong một chương trình, ngoài luồng chính thì có thể có các luồng khác thực thi đồng thời làm ứng dụng chạy nhanh và hiệu quả hơn |
| **Câu 125: Phân biệt Multitasking, multiprocessing, multithreading**  **Multitasking**  Một máy tính có thể thực thi cùng một lúc nhiều chương trình khác nhau được gọi là Multitasking. Hệ thống sẽ điều khiển mỗi CPU luân phiên thực thi mỗi chương trình trong một khoảng thời sau đó chuyển sang thực thi các chương trình khác. Chúng ta không còn giả định được chương trình của mình chiếm toàn bộ thời gian CPU mà nó chỉ chiếm một khoảng được hệ thống cấp cho.  **Multithreading**  Multithreading nghĩa là có nhiều thread thực thi cùng lúc trong một chương trình nhầm tối ưu tài nguyên của CPU, tăng tốc độ xử lý và trải nghiệm người dùng. |
| **Câu 126: Thread có thể được tạo bằng hai cách nào? So sánh**  Có hai cách để tạo ra một thread: Bởi extends lớp Thread. Bởi implements Runnable interface. |
| **Câu 127: Phân biệt run(), start()**  - Khi chúng ta gọi phương thức start(), 1 thread mới sẽ được tạo và code trong phương thức run() sẽ được thực thi trong thread mới này. Nếu chúng ta gọi phương thức run(), không có thread mới nào được tạo và code trong run() sẽ được thực thi tại thread hiện tại. |
| **Câu 128: Vòng đời Thread** |
| **Câu 129: Giải thích trình dọn rác (Garbage Collector) trong Java?**  Garbage Collection (Bộ thu gom rác) trong Java được định nghĩa như một quá trình tự động thực thi nhiệm vụ quản lý bộ nhớ. |
| **Câu 130: Các cách mà một Thread có thể đi vào trạng thái đợi (waiting state)?**  phương thức sleep() |
| **Câu 131: Cách mà Đa luồng (Multi-Thread) diễn ra trên một máy tính với một CPU đơn?**  Scheduler của hệ điều hành cấp phát thời gian thực thi cho các Task. Bằng việc nhanh nhóng chuyển đổi giữa các Task đang thực thi, nó tạo cho chúng ta cảm tưởng rằng các Task này được thực thi đồng thời. |
| **Câu 132: Một Thread đã chết có thể được restart lại như thế nào?**  Không thể restart |
| **Câu 133: Điểm khác nhau giữa yield và sleep?**  Khi một tác vụ triệu hồi phương thức yield() của nó, nó chuyển thành trạng thái sẵn sàng. Khi một tác vụ triệu hồi phương thức sleep() của nó, nó chuyển sang trạnh thái đợi. |
| **Câu 134: Phương thức sleep() và wait() khác nhau ở điểm nào?**  sleep() method nó sẽ tự động đánh thức sau một khoảng thời gian được chỉ định nếu không bị gián đoạn. Đối với wait(), khi wait() method được gọi, thread hiện tại sẽ rơi vào trạng thái chờ cho đến khi một thread khác gọi notify() hay notifyAll() trên cùng một object. |