**‘**

**;**

**BỘ CÂU HỎI PHỎNG VẤN MODULE 2**

**ADVANCED PROGRAMMING WITH JAVA**

|  |  |
| --- | --- |
| **STT** | **Câu hỏi** |
| 1 | **Sự khác nhau giữa JDK, JRE và JVM?**   * JDK là một bộ phát triển phần mềm. * JRE là một gói phần mềm cho phép chương trình Java chạy. * JVM là một môi trường để thực thi byte code. * Dạng đầy đủ của JDK là Java Development Kit, trong khi dạng đầy đủ của JRE là Java Runtime Environment, trong khi dạng đầy đủ của JVM là Java Virtual Machine. |
| 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). |
| 3 | **Trình biên dịch JIT là gì?**   * Trình biên dịch JIT (Just-In-Time) được sử dụng để tăng hiệu suất biên dịch chương trình. * Trình JIT biên dịch từng phần của byte code có chức năng giống nhau tại cùng một thời điểm, do đó sẽ giảm được thời gian biên dich. Ở đây, thuật ngữ "trình biên dịch" dùng để chỉ trình dịch từ tập lệnh của một máy ảo Java (JVM) tới tập lệnh của một CPU cụ thể. |
| 4 | **Platform là gì?**   * Bất cứ môi trường phần cứng hoặc phần mềm nào mà trong đó có một chương trình chạy, thì được hiểu như là một Platform. * Với môi trường runtime riêng cho mình (JRE) và API, Java được gọi là Platform. |
| 5 | **Sự khác nhau giữa Java platform và các platform khác?**   * Java platform khác với hầu hết các nền tảng khác theo nghĩa nó là một nền tảng dựa trên phần mềm chạy trên các nền tảng phần cứng khác. Nó có hai thành phần:  1. Môi trường thực thi (Runtime) 2. API(Application Programming Interface) |
| 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. |
| 7 | **Classloader trong java là gì?**   * Classloader là một hệ thống con của JVM được sử dụng để tải các lớp và các interface. * Có rất nhiều loại Classloader ví dụ: Bootstrap classloader, Extension classloader, System classloader, Plugin classloader… |
| 8 | **File có tên trống ".java" có hợp lệ không?**   * Có, bạn có thể lưu file java với tên ".java", sau đó biên dịch bằng lệnh javac .java và chạy. |
| 9 | **Các từ delete, next, main, exit và null có phải là từ khóa trong java không?**   * Không. |
| 10 | **Nếu không cung cấp bất kỳ đối số nào trên command line, thì mảng String của hàm main là empty hay null?**   * Mảng String là empty, không phải null. |
| 11 | **Chuyện gì xảy ra nếu khai báo static public void thay vì public static void?**   * Chương trình được biên dịch và run đúng. |
| 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. |
| 13 | **Khác nhau Biến tham trị và Biến tham chiếu?**   * Tham trị tức là truyền giá trị * Tham chiếu tức là chiếu đến 1 địa chỉ |
| 14 | **Bạn hiểu gì về Đối tượng?**   * Một thực thể có trạng thái và hành vi được gọi là đối tượng |
| 15 | **Định nghĩa Lớp (class)?**   * Lớp (class) là mô tả về các đối tượng sẽ được tạo ra. |
| 16 | **Kiểu biến nào một lớp có thể chứa?**   * Một lớp có thể gồm biến local, biến instance, và biến lớp. |
| 17 | **Biến local hay biến cục bộ là gì?**   * Các biến được định nghĩa bên trong phương thức, constructor hoặc các khối được gọi là biến local. Biến này sẽ được khai báo và khởi tạo bên trong phương thức và nó sẽ bị hủy khi phương thức kết thúc. |
| 18 | **Biến instance là gì?**   * Biến instance là các biến bên trong một lớp nhưng bên ngoài bất cứ phương thức nào. Những biến này được thuyết minh khi lớp được tải. |
| 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. Ví dụ về các ngôn ngữ lập trình dựa trên đối tượng là JavaScript, VBScript,... |
| 20 | **Giá trị khởi tạo của biến tham chiếu đối tượng được định nghĩa là biến instance là gì?**   * Trong java, các tham chiếu của đối tượng được khởi tạo là null. |
| 21 | **Constructor là gì?**   * Constructor(phương thức khởi tạo) là phương thức đặc biệt. Nó cùng tên với lớp của nó, không có kiểu dữ liệu trả về và được gọi để tạo đối tượng mới thông qua toán tử NEW |
| 22 | **Mục đích của constructor là gì?**   * Nó dùng để khởi tạo đối tượng mới thông qua việc sử dụng toán tử NEW. |
| 23 | **Constructor trả về kiểu giá trị gì?**   * Nó không có kiểu dữ liệu trả về. |
| 24 | **Constructor được kế thừa không?**   * Không, constructor không được kế thừa |
| 25 | **Có thể tạo constructor final không?**   * Không, constructor không thể là final. |
| 26 | **Biến static là gì?**   * Biến static có thể được sử dụng để tham chiếu thuộc tính chung của tất cả đối tượng (mà không là duy nhất cho mỗi đối tượng), ví dụ như tên công ty của nhân viên, tên trường học của các sinh viên, … * Biến static lấy bộ nhớ chỉ một lần trong Class Area tại thời gian tải lớp đó. |
| 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 |
| 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. |
| 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. |
| 30 | **Chúng ta có thể thực thi một chương trình không có phương thức main() không?**   * Có, một trong các cách đó là khối static trong phiên bản trước của JDK. Không phải là JDK 1.7 |
| 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" |
| 32 | **Sự khác nhau giữa phương thức static và phương thức instance?**   * Phương thức static:  được khai báo với từ khóa Static, không tạo đối tượng vẫn gọi phương thức static ra dc, CÁC biến non-static thì cũng không gọi ra dc trong phương thức static. * Phương thức instance : là phương thức mà không được khai báo với từ khóa Static, phải tạo đối tượng mới gọi được phương thức instance,biến non-static và static thì dc truy cập trực tiếp trong phương thức instance |
| 33 | **THIS trong java là gì?**   * THIS là một biến tham chiếu được sử dụng để tham chiếu tới đối tượng của lớp hiện tại. |
| 34 | **Kế thừa là gì?**  Kế thừa là cơ chế cho phép một lớp Con sử dụng lại các đặc điểm và hành vi đã được định nghĩa trong lớp Cha. |
| 35 | **Lớp nào là lớp cha cho tất cả các lớp?**   * Lớp Object |
| 36 | **Tại sao đa kế thừa không được hỗ trợ trong java?**   * Java không cho phép đa kế thừa VÌ chúng ta biết rằng java được thiết kế với nhiều cải tiến trong đó sự đơn giản là một trong những điều quan trọng. Việc cho phép đa kế thừa sẽ mang đến nhiều hệ lụy, khiến cho code trở nên mơ hồ. |
| 37 | **Composition là gì?**   * Composition - tạm dịch là tổng hợp - là một kỹ thuật design. * Để thực hiện kỹ thuật Composition, ta cần sử dụng các object ở một class khác làm thuộc tính cho object được tạo bởi class tổng hợp. * Composition được sinh ra nhằm đạt được mục đích là implement mối quan hệ có-một (has-a) giữa các class. Quan hệ has-a của 2 class tức là trong quá trình xây dựng class, class này có tham chiếu đến class khác (xem như là thành viên của nó). |
| 38 | **Sự khác nhau giữa aggregation và composition?**   * **Aggregation** là mối quan hệ mà ở đó thực thể có thể tồn tại độc lập với các lớp khác. Ví dụ: một người nào đó có thể thuộc một công ty hoặc không thuộc một công ty, người đó vẫn có thể tồn tại độc lập, có nghĩa là họ không làm cho một công ty bất kỳ. * **Composition** đề cập đến trường hợp khi một đối tượng (obj1) tồn tại phụ thuộc vào sự tồn tại của một đối tượng khác(obj2). Có nghĩa rằng, nếu đối tượng được phụ thuộc obj2 tồn tại thì đối tượng obj1 mới có thể tồn tại. |
| 39 | **Tại sao java không support con trỏ?**   * Thực ra Java vẫn có con trỏ. Thể hiện qua việc bạn cấp phát bộ nhớ bằng SomeClass a = new SomeClass(). * Nếu bạn lấy SomeClass b = a; thì b đang trỏ đến vùng nhớ của a. * Ps: Con trỏ this trong Java là một con trỏ. |
| 40 | **SUPER trong java là gì?**   * TL: Từ khóa super trong Java là một biến tham chiếu, được sử dụng để tham chiếu trực tiếp đến đối tượng của lớp cha gần nhất. * Bất cứ khi nào bạn tạo ra một thể hiện (hay còn gọi là một instance) của lớp con, một instance của lớp cha được tạo ra ngầm định, nghĩa là được tham chiếu bởi biến super. |
| 41 | **Có thể sử dụng cả this() và super() trong một constructor?**   * Có thể. |
| 42 | **Object cloning là gì?**   * Object cloning là cách để tạo ra một bản sao chính xác của một đối tượng bị clone. Phương thức clone() được sử dụng để tạo ra một object mới. |
| 43 | **Overloading (nạp chồng) phương thức là gì?**   * Nếu một lớp có nhiều phương thức có tên giống nhau nhưng các tham số khác nhau, được gọi là overloading phương thức (nạp chồng phương thức). Nó giúp code rõ ràng, dễ hiểu hơn. |
| 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?**   * Nếu lớp con có phương thức giống lớp cha được gọi là ghi đè (overriding) phương thức trong java. * Nói cách khác, nếu lớp con cung cấp sự cài đặt cụ thể cho phương thức đã được cung cấp bởi một lớp cha của nó được gọi là ghi đè (overriding) phương thức trong java. |
| 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 được sử dụng để giúp code của chương trình dễ đọc hơn. * Ghi đè phương thức được sử dụng để cung cấp cài đặt cụ thể cho phương thức được khai báo ở lớp cha. |
| 52 | **Biến final là gì?**   * Nếu bạn tạo một biến với từ khóa final, bạn sẽ không thể thay đổi được giá trị của biến đó (hằng số). |
| 53 | **Phương thức final là gì?**   * Phương thức final không thể được ghi đè. |
| 54 | **Lớp final là gì?**   * Lớp final 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. |
| 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?**   * Khi kiểu của đối tượng được xác định tại lúc biên dịch được gọi là ràng buộc tĩnh. * Nếu phương thức là private, final hoặc static thì đó là ràng buộc tĩnh. * Khi kiểu của một đối tượng được xác định tại runtime được gọi là ràng buộc động. |
| 61 | **Trừu tượng là gì?**   * Tính trừu tượng trong Java là tính chất không thể hiện cụ thể mà chỉ nêu tên vấn đề. Đó là một quá trình che giấu các hoạt động bên trong và chỉ hiển thị những tính năng thiết yếu của đối tượng tới người dùng. |
| 62 | **Sự khác nhau giữa trừu tượng và đóng gói là gì?**   * Tính trừu tượng chỉ hiển thị dữ liệu hữu ích bằng cách cung cấp các chi tiết cần thiết nhất. Trong khi tính năng Đóng gói bao bọc mã và dữ liệu để có thông tin cần thiết. |
| 63 | **Lớp trừu tượng là gì?**   * Trừu tượng hóa là một khái niệm OOP chỉ tập trung vào dữ liệu có liên quan của một đối tượng. * Nó ẩn các chi tiết nền và nhấn mạnh các điểm dữ liệu cần thiết để giảm độ phức tạp và tăng hiệu quả. * OOP thường chỉ giữ lại thông tin phù hợp nhất cho quá trình cụ thể đó. Phương pháp trừu tượng chủ yếu tập trung vào ý tưởng thay vì vận hành thực tế. |
| 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 phương thức 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?**   * Một Interface trong Java là một bản thiết kế của một lớp. Nó chỉ có các phương thức trừu tượng. Interface là một kỹ thuật để thu được tình trừu tượng hoàn toàn và đa kế thừa trong Java. |
| 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?**   * Một lớp abstract có thể có phương thức có nội dung (các phương thức tường minh). * Interface chỉ có các phương thức trừu tượng. |
| 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?**   * Một package (gói) trong java là một nhóm các kiểu tương tự của các lớp, giao diện và các package con[.](https://viettuts.vn/java/package-trong-java) |
| 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. |
| 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). * 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. |
| 81 | **Khối finally là gì?**   * Khối lệnh finally trong java được sử dụng để thực thi các lệnh quan trọng như đóng kết nối, đóng cá stream,... * Khối lệnh finally trong java luôn được thực thi cho dù có ngoại lệ xảy ra hay không hoặc gặp lệnh return trong khối try. |
| 82 | **Khối finally có thể được sử dụng mà không cần khối catch không?**   * Không. * Khối lệnh finally trong java phải được khai báo sau khối lệnh try hoặc sau khối lệnh catch. |
| 83 | **Có trường hợp nào khối finally không được thực thi không?**   * Khối finally sẽ không được thực thi nếu chương trình bị thoát (bằng cách gọi System. exit() hoặc xảy ra một lỗi không thể tránh khiến chương trình bị chết). |
| 84 | **Sự khác nhau giữa throw và throws là gì?**   * Throw được dùng khi cần phát sinh ngoại lệ trong một phương thức * Throw được dùng với phương thức để gây ra bất cứ checked hay unchecked exception nào. Đồng thời nhường công việc xử lý ngoại lệ đó cho người gọi phương thức này. * Throws là cần thiết cho hầu hết tất cả các ngoại lệ. |
| 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?**   * Nếu phương thức của lớp cha không khai báo ném ra exception, phương thức được ghi đè của lớp cha không thể khai báo ném ra ngoại lệ checked, nhưng ngoại lệ unchecked thì có thể. |
| 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. |
| 87 | **Có vấn đề gì không với thứ tự của các lệnh catch được viết cho FileNotFoundException và IOException?**   * Có. FileNotFoundException được kế thừa từ IOException. Các lớp con của Exception phải được bắt đầu tiên. |
| 88 | **Điểm khác nhau giữa Error và Exception?**   * Lỗi error là do lập trình viên viết code sai trong lúc xây dựng chương trình. Nằm trong tầm kiểm soát của lập trình viên. * Những ngoại lệ (exception) là những lỗi phát sinh trong quá trình thực thi, được phát sinh trong tình huống chạy chương trình thông thường. |
| 89 | **Ý nghĩa của immutable (bất biến) trong String là gì?**   * Trong java đối tượng string là bất biến(immutable). Bất biến có nghĩa là không thể thay đổi. |
| 90 | **Tại sao các đối tượng String trong java là immutable?**   * Immutable đảm bảo cho các String rằng giá trị của chúng sẽ không thay đổi. Vì vậy, phương thức hashCode() được override trong class String để tạo điều kiện sử dụng bộ nhớ đệm. |
| 91 | **Có bao nhiêu cách để tạo ra một đối tượng String trong java?**   * Chúng ta có nhiều cách để khởi tạo một đối tượng String, đó là: * Sử dụng toán tử new * Sử dụng toán tử gán (“=”) * Khai báo trong dấu nháy kép |
| 92 | **Có bao nhiêu đối tượng String được tạo ra trong đoạn code sau?** |
| 93 | **Tại sao java sử dụng khái niệm string literal?**   * Để làm cho bộ nhớ của Java hiệu quả hơn (vì không có đối tượng mới được tạo ra nếu nó đã tồn tại trong string constant pool). |
| 94 | **Có bao nhiêu đối tượng được tạo ra trong đoạn code sau?** |
| 95 | **Sự khác nhau giữa String và StringBuffer là gì?**   * Lớp String là bất biến (immutable). * Lớp StringBuffer là có thể sửa đổi (mutable). * Khi thực hiện nối nhiều chuỗi thì lớp String xử lý chậm và tốn nhiều bộ nhớ hơn, bởi vì mỗi lần nối thêm chuỗi nó tạo ra instance mới. * Khi bạn thực hiện nối nhiều chuỗi thì lớp StringBuffer xử lý nhanh và tốn ít bộ nhớ hơn. |
| 96 | **Sự khác nhau giữa StringBuffer và StringBuilder là gì?**   * StringBuffer là đồng bộ (synchronized) tức là luồng an toàn. * StringBuilder là không đồng bộ (non-synchronized) tức là luồng không an toàn. |
| 97 | **Làm thế nào để tạo lớp immutable trong java?**  Text, letter  Description automatically generated |
| 98 | **Mục đích của phương thức toString() trong java là gì?**   * Đây là phương thức được sử dụng để chuyển đổi một object thành một string (chuỗi) rồi trả về nó. * Phương thức toString() thuộc class Object trong Java và có thể sử dụng nó mà không cần import thư viện nào cả. * Vì thế nó có thể trả về các giá trị của đối tượng mà không cần phải viết nhiều mã. |
| 99 | **Em biết gì về String Pool?**   * String pool là một vùng nhớ đặc biệt nằm trong vùng nhớ Heap (Heap memory), dùng để lưu trữ các biến được khai báo theo kiểu String. * String pool giúp tối ưu hoá việc lưu trữ và sử dụng vùng nhớ khi khai báo biến String |
| 100 | **Nested class (lớp lồng nhau) là gì?**   * Là lớp được khai báo trong một lớp hoặc một interface khác. |
| 101 | **Có sự khác nhau giữa nested class và inner class không?**   * Nested class là một class được khai báo bên trong một class khác. * Nested class được phân chia thành 2 loại: static nested classes và non-static nested classes. * Non-static Nested Classes: bao gồm Inner classes, Local classes and Anonymous Classes. * Inner Classes: là khai báo một class không phải dạng static bên trong một class khác. |
| 102 | **Nested interface là gì?**   * Một interface được khai báo trong một interface hoặc lớp khác được gọi là interface lồng nhau hay nested interface. * Các interface lồng nhau được sử dụng để nhóm các interface liên quan để chúng có thể dễ dàng được bảo trì. Interface lồng nhau phải được bao bọc bên ngoài bởi interface hoặc lớp. Nó không thể truy cập trực tiếp. |
| 103 | **Có thể khai báo interface trong class không?**   * Có. * Vd: Queue,... |
| 104 | **Có thể khai báo một class trong interface không?**   * Tất cả các method trong interface đều được ngầm hiểu là public abstract class * Một interface không thể chứa một trường nào trừ các trường vừa static và final. |
| 105 | **Sự khác nhau giữa Array và ArrayList là gì?**   * Mảng (Array) là một cấu trúc dữ liệu có kích thước cố định, trong khi ArrayList là một lớp Collection có thể thay đổi được kích thước. Nghĩa là chúng ta không thể thay đổi kích thước của mảng khi đã tạo, nhưng ArrayList có thể được thay đổi. * Chúng ta không thể lưu giữ dữ liệu nguyên thủy trong ArrayList, nó chỉ có thể chứa các đối tượng. Nhưng mảng có thể chứa cả hai kiểu dữ liệu nguyên thủy và các đối tượng trong Java. Kể từ Java 5, kiểu nguyên thủy được tự động chuyển đổi trong các đối tượng được gọi là auto-boxing. |
| 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. |
| 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. |
| 108 | **Bạn quyết định khi nào sử dụng ArrayList và LinkedList?**   * ArrayList tốt hơn trong việc lưu trữ và truy xuất dữ liệu (get). * LinkedList tốt hơn trong việc thao tác dữ liệu (thêm/ xóa). |
| 109 | **Sự khác nhau giữa ArrayList và LinkedList là gì?**   * ArrayList là dùng một mảng động (như mảng thường nhưng có thể thay đổi kích thước và các phương thức thêm) để lưu trữ phần tử. * LinkedList sử dụng danh sách liên kết để lưu trữ phần tử. Mỗi phần thử có thể được gọi là 1 node trong danh sách. |
| 110 | **Sự khác nhau giữa ArrayList và Vector là gì?**   * ArrayList là nhanh hơn vì nó là non-synchronized. * Vector là synchronized. * Vector chậm hơn vì nó là synchronized. Tức là, trong môi trường đa luồng, các thread giữ nó ở trong trạng thái runnable hoặc non-runnable cho đến khi thread hiện tại giải phóng đối tượng đó. |
| 111 | **Sự khác nhau giữa Iterator và ListIterator là gì?**   * Giữa Iterator và ListIterator có 2 điểm khác nhau chủ yếu như sau: * Thứ nhất: Iterator được sử dụng để duyệt qua các Interface Collections như List và Set, trong khi ListIterator chỉ được sử dụng để duyệt qua List. * Thứ hai: Iterator chỉ có thể duyệt qua các phần tử theo một hướng (từ đầu đến cuối danh sách), trong khi ListIterator có thể duyệt qua các phần tử theo 2 hướng (từ đầu đến cuối danh sách và ngược lại). |
| 112 | **Sự khác nhau giữa List và Set là gì?**   * List  là một chuỗi các yếu tố được sắp xếp trong khi đó, set là một danh sách các yếu tố riêng biệt không có thứ tự. |
| 113 | **Sự khác nhau giữa Set và Map là gì?**   * Set chỉ chứa giá trị, trong khi đó Map chứa cặp key và value. |
| 114 | **Sự khác nhau giữa Stack và Queue là gì?**   * Sự khác biệt lớn nhất giữa Stack và Queue: * Stack sử dụng phương thức LIFO (last in first out) để truy cập và thêm các phần tử dữ liệu, trong khi Queue sử dụng phương thức FIFO (First in first out) để truy cập và thêm các phần tử dữ liệu. |
| 115 | **Sự khác nhau giữa Collection và Collections là gì?**   * Collection là một interface. * Collections là một lớp. |
| 116 | **Sự khác nhau giữa Comparable và Comparator là gì?**  *Comparable:*   * Bạn phải implements giao tiếp Comparable cho lớp đối tượng cần được so sánh. * Comparable cung cấp phương thức compareTo() để sắp xếp các phần tử. * Comparable thuộc về java.lang package. * Chúng ta có thể sắp xếp các phần tử của kiểu Comparable bởi phương thức Collections.sort(List) .   *Comparator:*   * Bạn KHÔNG phải implements giao tiếp Comparator cho lớp đối tượng cần được so sánh. * Comparator cung cấp phương thức compare() để sắp xếp các phần tử. * Comparator thuộc về java.util package. * Chúng ta có thể sắp xếp các phần tử của kiểu Comparator bởi phương thức Collections.sort(List,Comparator). |
| 117 | **Phương thức hashCode() là gì?**   * Phương thức hashCode() trả về 1 giá trị mã băm (một kiểu số nguyên). * Phương thức hashCode() trả về cùng kiểu số nguyên, nếu 2 keys ( =  phương thức equals()) giống nhau. |
| 118 | Tại sao chúng ta phải ghi đè phương thức equals()? |
| 119 | Làm thế nào để đồng bộ List, Set và Map? |
| 120 | **Lợi ích của generic collection là gì?**   * Kiểu đối tượng an toàn :  Chúng ta chỉ có thể lưu một kiểu đối tương duy nhất trong generics . Nó không cho phép lưu trữ 2 dối tượng khác nhau. * Không cần ép kiểu. |
| 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. |
| 122 | **Bạn hiểu gì về Access Modifier?**   * Access Modifer trong Java xác định phạm vi có thể truy cập của biến, phương thức, constructor hoặc lớp. |
| 123 | **Phân biệt public, protected, private, default?**   * Private Access Modifier chỉ được truy cập trong phạm vi lớp. * Nếu bạn không khai báo modifier nào, thì nó chính là trường hợp mặc định. Default Access Modifier là chỉ được phép truy cập trong cùng package. * Protected access modifier được truy cập bên trong package và bên ngoài package nhưng phải kế thừa. * Public access modifier được truy cập ở mọi nơi. |
| 124 | **Bạn hiểu gì về Đa luồng (Multi-Thread)?**   * Multi-thread (đa luồng) là một tiến trình thực hiện nhiều luồng đồng thời. Một ứng dụng Java ngoài luồng chính có thể có các luồng khác thực thi đồng thời làm ứng dụng chạy nhanh và hiệu quả hơn. |
| 125 | **Phân biệt Multitasking, multiprocessing, multithreading**   * 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** 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. * **Multiprocessing** là khả năng của một hệ thống hỗ trợ nhiều bộ vi xử lý processor cùng một lúc. Các ưng dụng trong hệ thống đa xử lý được chia thành nhiều quy trình nhỏ và chạy độc lập, và hệ điều hành sẽ phân bổ các luồng này cho bộ vi xử lý để cải thiện hiệu suất của hệ thống. |
| 126 | **Thread có thể được tạo bằng hai cách nào? So sánh?**   * Cách thứ nhất là kế thừa class Thread, và đưa các tác vụ vào phương thức run() và override phương thức run(). Các tác vụ sẽ được thực hiện khi gọi start() * Implement từ interface Runnable .Khi triển khai Runnable, nhiều luồng có thể chia sẻ cùng một đối tượng luồng trong khi ở lớp Mở rộng luồng, mỗi luồng có một đối tượng duy nhất được liên kết với nó. |
| 127 | **Phân biệt run(), start()?**  Cả 2 phương thức này đều có chức năng gọi đến phương thức run() của thread và thực thi các dòng code bên trong phương thức này. Vậy sự khác nhau giữa chúng là:   * 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. * Ngoài ra, chúng ta không thể gọi phương thức start() 2 lần trên đối tượng thread, việc này sẽ quăng ra lỗi IllegalStateException trong Java. Trong khi đó, chúng ta có thể gọi phương thức run() 2 lần trên đối tượng thread. |
| 128 | **Vòng đời Thread**   1. *New Thread* : Khi một Thread mới được tạo, nó ở trạng thái mới. Thread vẫn chưa bắt đầu chạy khi Thread ở trạng thái này. Khi một Thread nằm ở trạng thái mới, mã của nó vẫn chưa được chạy và chưa bắt đầu thực thi. 2. *Runnable State* : Một Thread đã sẵn sàng chạy được chuyển sang trạng thái có thể chạy được. Trong trạng thái này, một Thread có thể thực sự đang chạy hoặc nó có thể sẵn sàng chạy bất kỳ lúc nào. Bộ lập lịch luồng có trách nhiệm cung cấp cho Thread, thời gian để chạy. Một chương trình đa luồng phân bổ một lượng thời gian cố định cho từng luồng riêng lẻ. Mỗi và mọi Thread chạy trong một thời gian ngắn, sau đó tạm dừng và chuyển CPU sang một Thread khác để các luồng khác có cơ hội chạy. Khi điều này xảy ra, tất cả các luồng đã sẵn sàng chạy, đang chờ CPU và luồng hiện đang chạy sẽ nằm ở trạng thái có thể chạy được. 3. *Blocked/Waiting state*: Khi một Thread tạm thời không hoạt động, thì nó ở một trong các trạng thái sau  * Bị chặn * Đang chờ đợi  1. *Timed Waiting*: Một Thread nằm ở trạng thái chờ theo thời gian khi nó gọi một phương thức có tham số thời gian chờ. Một Thread nằm ở trạng thái này cho đến khi hết thời gian chờ hoặc cho đến khi nhận được thông báo. Ví dụ: khi một Thread gọi là ngủ hoặc chờ có điều kiện, nó sẽ được chuyển sang trạng thái chờ theo thời gian. 2. *Terminated State:* Một Thread kết thúc vì một trong các lý do sau: Vì nó tồn tại bình thường. Điều này xảy ra khi mã của Thread đã được chương trình thực thi hoàn toàn. Bởi vì đã xảy ra một số sự kiện sai bất thường, như lỗi phân đoạn hoặc một ngoại lệ chưa được khắc phục. |
| 129 | **Giải thích trình dọn rác (Garbage Collector) trong Java?**   * Đầu tiên chúng ta hãy nhớ lại kiến trúc JVM, trong module "Run time area" chúng ta có 2 vùng nhớ là vùng nhớ Stack (dùng để lưu trữ tham số và các biến local) và vùng nhớ Heap (dùng để lưu trữ các đối tượng sau khi khởi tạo bằng từ khóa new và các biến static). Bộ nhớ Heap là bộ nhớ cần dọn dẹp nhất vì các Object không cần dùng nữa phải được xóa bỏ để giải phóng bộ nhớ. * Có nhiều tiến trình thu gom rác khác nhau nhưng phổ biến nhất Oracle HotSpot. Mặc dù HotSpot có nhiều tiến trình thu gom rác được tối ưu cho từng trường hợp khác nhau nhưng tất cả đều theo một phương thức cơ bản nhất. Đầu tiên, các Object không được tham chiếu sẽ được đánh dấu sẵn sàng để được dọn rác. Ở bước thứ hai, trình thu gom rác sẽ tiến hành xóa các Object đó. Ở bước thứ ba, vùng nhớ của các Object còn lại sẽ được nén lại và nằm liền kề nhau trong bộ nhớ Heap. Quá trình này sẽ giúp việc cấp phát bộ nhớ cho Object mới dễ dàng hơn. * Ở vùng nhớ Heap được chia làm 3 vùng nhớ nhỏ hơn, tạm gọi là Young generation, Old generation và Permanent generation. * Young generation: được chia thành nhiều vùng nhớ nhỏ hơn là Eden(khởi tạo) và Survivor(sống sót). Các Object vừa được khởi tạo sẽ nằm ở vùng Eden, sau chu kì quét đầu tiên nếu Object đó còn tồn tại thì sẽ được chuyển sang vùng Survivor. Tại đây cái Object được GC theo dõi liên tục, nếu  như qua nhiều chu kì quét mà Object vẫn còn được sử dụng thì lúc này Object sẽ được chuyển sang vùng nhớ thứ thứ hai. * Old generation: là nơi chứa những Object tòn tại đủ "lâu", còn "lâu" như thế nào thì lại tùy thuộc vào thuật toán của từng bộ GC.Permanent generation: không chứa Object mà đây là nơi để chứa các metadata của JVM như các class và method của ứng dụng. Do đó nếu các class và method không còn được sử dụng nữa thì GC sẽ coi chúng là "rác" và dọn dẹp nó. |
| 130 | **Các cách mà một Thread có thể đi vào trạng thái đợi (waiting state)?**   * Một Thread có thể đi vào trạng thái đợi (Waiting state) bằng việc triệu hồi phương thức sleep() của nó, bằng việc được khóa trên IO, hoặc thất bại trong việc cố gắng thu được lock của đối tượng, hoặc bởi triệu hồi phương thức wait() của đối tượng. * Nó cũng có thể đi vào trạng thái đợi bởi triệu hồi phương thức suspend() của nó (phương thức này đã cũ). |
| 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. |
| 132 | **Một Thread đã chết có thể được restart lại như thế nào?**   * Không restart được. |
| 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. |
| 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. |