**CÂU HỎI AUDIT MODULE 2**

**BOOTCAMP PREPARATION 2.0**

|  |  |  |
| --- | --- | --- |
| **STT** | | **Câu hỏi** |
| 1 | Java là gì   * Là ngôn ngữ lập trình hướng đối tượng ÔP. * Thực thi được trên nhiều nền tảng và thiết bị. * Slogan: Write one, run anywhere. | | |
| 2 | JDK, JRE, JVM?   * JDK: Java Development Kit - bộ công cụ hỗ trợ lập trình Java. * JRE: Java Runtime Enviromment – môi trường thực thi ứng dụng Java * JVM: Java Virtual Machine – máy ảo Java thực thi (thông dịch) Java byte code sang ngỗn ngữ máy. | | |
| 3 | Thông dịch, biên dịch khác nhau như thế nào?   * Biên dịch: dịch toàn bộ file 1 lần, sau đó sử dụng kết quả biên dịch mà không cần biên dịch lại lần nữa (\*.java -> \*.class). * Thông dịch: dịch từng dòng lệnh, muốn chạy lại thì phải dịch lại lần nữa (JVM thông dịch \*.class ra ngôn ngữ máy). | | |
| 4 | JAVA là ngôn ngữ thông dịch hay biên dịch ? | | |
| 5 | Các kiểu dữ liệu trong java, và giá trị mặc định khi khai báo?   * Trong Java có 8 loại kiểu dữ liệu nguyên thủy và kiểu Object. * Giá trị mặc định khi khai báo:   + Kiểu nguyên thủy:     - Bằng 0 đối với số     - False (đối với kiểu Boolean)     - \0000 (đối với kiểu Char)   + Kiểu Object: nulll (khac với JavaScript là undefined) | | |
| 6 | Có bao nhiêu loại mệnh đề If ? | | |
| 7 | So sánh if và switch-case ? | | |
| 8 | Khi nào dùng for, while, do-while ?   * Vòng lặp for: Dùng khi biết trước số lần lặp. * Vòng lặp while: Dùng khi chưa biết trước số lần lặp   + While : Xét điều kiện trước rồi mới lặp   + Do – while: Thì thực hiện ít nhất 1 lần rồi mới xét điều kiện lặp. | | |
| 9 | Phân biệt for-i và for-each ?   * for-i : có thể điều khiển đc thứ tự duyệt (duyệt mảng ngược, xuôi hoặc từ giữa, …)   + Biểu thức khởi tạo   + Biểu thức điều khiện   + Biểu thức tăng giảm   + Phần body của for * for-each : chỉ duyệt từ đầu tới cuối của 1 danh sách / mảng   + Khai báo biến để duyệt danh sách/mảng   + Mảng, danh sách cần duyệt   + Body | | |
| 10 | **Khác nhau giữa while, do-while. Cho ví dụ khi nào dùng ?**   * While : Xét điều kiện trước rồi mới lặp * Do – while: Thì thực hiện ít nhất 1 lần rồi mới xét điều kiện lặp.   Ví dụ: CHẾ LINH | | |
| 11 | **Break, Continue có tác dụng gì trong mệnh đề lặp ?**   |  |  | | --- | --- | | **Continue** | **Break** | | Khi gặp từ khóa “continue” thì chương trình sẽ bỏ qua tất cả các câu lệnh các câu lệnh dưới trong cùng vòng lặp và nhảy tới vòng lặp tiếp theo | Khi gặp “break” thì chương trình sẽ thoát khỏi vòng lặp gần nhất | | Không sử dụng trong Switch-case | Có thể sử dụng trong Switch-case | | | |
| 12 | **Trình bày các cách khởi tạo một mảng trong JAVA ?**   * Có 2 cách:   + Dùng từ khóa new (ví dụ: String [] text = new String(5); )   + Gán trực tiếp giá trị của 1 mảng có sẵn (ví dụ: int arr[] = {1,2,3}; ) | | |
| 13 | **Phần tử của mảng có thể dùng kiểu dữ liệu nào, và có giá trị mặc định là gì ?**   * Phần tử của mảng có thể là kiểu nguyên thủy hoặc là kiểu đối tượng. * Giá trị mặc định của mảng là: null. * Giá trị mặc định của các phần tử trong mảng: là giá trị mặc định theo kiểu dữ liệu khi khai báo mảng. | | |
| 14 | **OOP là gì?**   * OOP (Object Oriented Programming) – lập trình hướng đối tượng là kỹ thuật lập trình ánh xạ các đối tượng ngoài thực tế vào ngôn ngữ lập trình. * 4 tính chất của OOP:   + Encapsulation (tính bao đóng)   + Inheritance (tính kế thừa)   + Polymorphism (tính đa hình)   + Abstraction (trừu tượng) | | |
| 15 | **Phân biệt class và object**   * Class: là 1 khuôn mẫu mô tả các đặc điểm và hành vi chung cho 1 nhóm các đối tượng tương đồng nhau. * Object: là 1 thể hiện cụ thể của class, có đặc điểm và hành vi cụ thể | | |
| 16 | **Constructor là gì?**   * Constructor là 1 method đặc biệt => dùng để khởi tạo đối tượng. | | |
| 17 | **Cách khai báo constructor và đặc điểm constructor trong JAVA?**   * Cách khai báo Constructor:   new ClassName ( parameter ) {  //Body của Constructor  }   * Đặc điểm Constructor trong Java:    + Phải cùng tên với Class.   + Không có kiểu dữ liệu trả về.   + Trong 1 Class có thể tạo được nhiều Constructor (với các tham số truyền vào khác nhau)   + Nếu trong 1 class không có bất kỳ Constructor nào thì Java sẽ tự cung cấp cho chúng ta 1 Constructor mặc định không có tham số truyền vào.   + Constructor được gọi khi khởi tạo 1 đối tượng (sử dụng từ khóa **new)**   + 1 Constructor có thể gọi 1 Constructor khác thông qua từ khóa **this** (lưu ý: phải đặt đầu của Constructor) | | |
| 18 | **Phân biệt constructor và method?**   |  |  | | --- | --- | | Constructor | Method | | Phải có tên cùng tên Class | Có thể trùng với tên Class (nhưng khuyến cáo không nên) | | Không có kiểu dữ liệu trả về | Có kiểu dữ liệu trả về | | * Được gọi bằng từ khóa **new** * Không sử dụng từ khóa **final** hay **Static** | * Được gọi thông qua tên **method** * Có thể sử dụng **final**, **static** | | | |
| 19 | **Tính bao đóng là gì? Làm sao để thu được tính bao đóng trong Java?**   * Tính bao đóng : Là kỹ thuật ẩn giấu thông tin của đối tượng, chỉ hiện những thông cần thiết. * Mục đích:   + Bảo vệ trạng thái bên trong của đối tượng (không cho phép thay đổi trực tiếp giá trị thuộc tính của đối tượng).   + Giảm độ phức tạp của chương trình * Để thu được tính bao đóng trong Java ta dùng:   + Access modifier   + Getter / setter | | |
| 20 | **Tham trị, tham chiếu là gì?**   * Tham trị (Pass by value): là truyền giá trị   + Áp dụng cho kiểu dữ liệu nguyên thủy.   + Giá trị của biến trước và sau khi gọi hàm không bị thay đổi. * Tham chiếu (Pass by reference): là truyền tham chiếu đến 1 địa chỉ    + Áp dụng cho kiểu dữ liệu Object, biến mảng (Array).   + Giá trị của biến sau khi gọi hàm có thể bị thay đổi hoặc không. * Trong Java thì chỉ có truyền tham trị nhưng thể hiện có thể tham chiếu hoặc tham trị tùy kiểu dữ liệu biến truyền vào (nguyên thủy hoặc đối tượng). | | |
| 21 | **Từ khóa Static dùng để làm gì?**   * Là từ khóa dùng để khai báo thuộc tính và phương thức của Class, không phải của đối tượng. * Có thể truy xuất biến **Static** hoặc phương thức **Static** thông qua tên Class hoặc đối tượng từ Class đó. * Có thể truy xuất biến hoặc phương thức (method) **Static** mà không cần phải khởi tạo đối tượng. * Múc đích dùng **Static** trong Class:    + Định nghĩa các phương thức, biến dùng chung cho toàn bộ đối tượng được tạo từ Class đó.   + Tạo ra các lớp tiện ích (sử dụng mà không cần phải tạo đối tượng). | | |
| 22 | **Ràng buộc khi sử dụng Static ?**   * Phương thức **Static** chỉ có thể sử dụng (truy xuất) được các biến **Static** hoặc các phương thức **Static**. * Chỉ khởi tạo biến **Static** thông qua khối tạo **Static**. | | |
| 23 | **Các loại biến trong Java?**   * Có 3 loại biến:    + Biến toàn cục: Khai báo thì có thể dùng được.   + Biến cục bộ: Khi khai báo phải gán giá trị mới sử dụng được.   + Biến Static: | | |
| 24 | **Trình bày các loại Acces Modifier, và phạm vi truy cập?**   * Có 4 loại **Acces Modifier:**    + Public: Phạm vi truy cập toàn bộ.   + Protected: Phạm vi truy cập chỉ trong packed và ngoài package nếu có thừa kế.   + Default: Phạm vi truy cập chỉ trong packed.   + Private: Phạm vi truy cập chỉ trong class. | | |
| 25 | **Kế thừa trong JAVA là gì ?**   * Kế thừa là cơ chế cho phép 1 lớp con sử dụng lại các đặc điểm và hành vi đã được định nghĩa ở lớp cha. * Mục đích: tái sử dụng được source code. * Mối quan hệ giữa lớp cha và lớp con là: is - a. * Lớp cha: parent class, super class, base class… * Lớp con: subclass, child class, denived class … | | |
| 26 | **Lớp con kế thừa được những tài sản nào(thuộc tính, phương thức) của lớp cha ?**   * Lớp con không thể kế thừa Constructor của lớp cha. * Lớp con không thể kế thừa các thuộc tính và phương thức private của lớp cha. * Java không hỗ trợ đa kế thừa. | | |
| 27 | **Lớp Object là gì?**   * Lớp Object là lớp cha của tất cả các lớp trong Java. * Phương thức toString() là phương thức trả về chuỗi biểu diễn thông tin của đối tượng. Nếu ta không định nghĩa thì mặc định nó sẽ trả về địa chỉ của đối tượng đó trong bộ nhớ. | | |
| 28 | **Khái niệm đa hình?**   * Tính đa hình là khả năng 1 đối tượng có thể thực hiện hành vi theo nhiều cách khác nhau tùy thuộc vào ngữ cảnh khác nhau. | | |
| 29 | Phân biệt Overloading và Overriding   |  |  | | --- | --- | | **Overloading** | **Overriding** | | Là cơ chế cho phép 1 lớp có khả năng định nghĩa ra nhiều phương thức có cùng tên, nhưng khác nhau về tham số truyền vào | Là cơ chế cho phép lớp con định nghĩa lại nội dung các phương thức đã có ở trước đó ở lớp cha.   * PT overriding phải có cùng tên, cùng danh sách tham số, và cùng kiểu trả về (hoặc chuỗi kế thừa). * Acess modifier của phương thức phải có level bằng hoặc cao hơn với phương thức của lớp cha. | | Xảy ra trong cùng 1 class | Xảy ra ở trong class có mối quan hệ kế thừa. | | Đa hình tại runtime | Đa hình tại compile | | | |
| 30 | **Ép kiểu là gì? Các loại ép kiểu?**   * Ép kiểu là cơ chế chuyển đổi một tham chiếu đến đối tượng thuộc loại này thành tham chiếu đến đối tượng thuộc loại khác. * Có 2 loại ép kiểu:    + Ép kiểu ngầm định: Xảy ra khi ép kiểu từ lớp con lên lớp cha. Diễn ra 1 cách tự động (ngầm) bởi hệ thống. Không xảy ra lỗi khi thực hiện ép kiểu loại này.   + Ép kiểu tường minh: Xảy ra khi ép kiểu từ lớp cha về lớp con. Không thể tự động ép kiểu được, ta phải chỉ rõ kiểu dữ liệu cần ép về. Có thể xảy ra lỗi khi ép kiểu tường mình => phải check bằng từ khóa “instance of” trước khi ép kiểu. | | |
| 31 | **Tính trừu tượng là gì?**   * Tính trường tượng là khả năng ẩn các chi tiết của quá trình triển khai chỉ thể hiện tính năng / kết quả cho người dùng => chỉ quan tâm đến kết quả đạt được, không quan tâm cách thực hiện. * Tính trừu tượng trong Java được thể hiện qua **Abstract class** và **interface**. | | |
| 32 | **Phân biệt Abstract class và Abstract interface?**   |  |  | | --- | --- | | **Abstract class** | **Abstract interface** | | **Abstract class** bản chất là 1 class vì vậy có tất cả tính chất của một class và vì có tính trừu tượng cao nên không thể tạo được đối tượng. | **Interface** là 1 bản thiết kế của 1 class, quy định hành vi chung cho class triển khai nó (triển khai bằng từ khóa “implements”). | | Các tính chất của **Abstract class**:   * Không thể tạo được đối tượng * Một class chứa method abstract thì bắt buộc class đó phải là **Abstract class** (ngược lại thì không). * **Abstract class** có thể chứa các thuộc tính và phương thức bình thường. * Không dùng từ khóa “final” cho class abstract và method abstract. * Có thể tạo constructor nhưng thường sẽ không được sử dụng trực tiếp vì nó không thể tạo đối tượng. * Lớp abstract có thể extends từ 1 lớp abstract khác (không cần phải implement method abstract của lớp cha). | Các tính chất của **Interface**:   * Không thể khởi tạo được đối tượng từ interface. * Các trường trong **Interface** mặc định là hằng số (mặc định là pulic static final …). * Các method mặc định toàn bộ là abstract (mặc định là public abstract…). * Không thể tạo được Constructor. * Có thể extends từ 1 hoặc nhiều **Interface** khác. | | | |
| 33 | **So sánh Array và ArrayList**   |  |  | | --- | --- | | **Array** (Có kích thước cố định) | **ArrayList** (Kích thước có thể thay đổi) | | Có thể lưu trữ đươc kiểu nguyên thủy và kiểu đối tượng. | Chỉ có thể lưu được kiểu đối tượng (nhưng với kiểu nguyên thủy sẽ được tự động chuyển qua kiểu đối tượng nhờ cơ chế auto-boxing). | | Chỉ có thuộc tính Length | Có nhiều phương thức thao tác với đối tượng. | | Tốc độ lưu trữ và thao tác nhanh | Tốc độ lưu trữ và thao tác chậm hơn. | | | |
| 34 | **So sánh ArrayList và LinkedList ?**   |  |  | | --- | --- | | **ArrayList** | **LinkedList** | | Có thể sử dụng mảng động để lưu trữ các phần tử. | Sử dụng danh sách liên kết lưu trữ các phần tử. | | Truy xuất ngẫu nhiên nhanh hơn | Truy xuất ngẫu nhiên chậm hơn | | Chèn phần tử vào đầu và giữa danh sách chậm hơn | Chèn phần tử vào đầu và giữa danh sách nhanh hơn | | | |
| 35 | **Set là gì, các lớp triển khai của Set?**   * Set là cấu trúc dữ liệu lưu trữ các phần tử không trùng lặp nhau. * Có 3 lớp triển khai:    + HashSet: Các phần tử được lưu trữ dưới dạng bảng băm, không duy trì thứ tự chèn.   + LinkedHashSet: Các phần tử được lưu trữ dưới dạng bảng băm, với cấu trúc dữ liệu dạng danh sách liên kết, duy trì thứ tự chèn.   + TreeSet: Lưu trữ phần tử dưới dạng cha con, mặc định các phần từ được sắp xếp tăng dần. | | |
| 36 | **Generic là gì?**   * Generic là cơ chế cho phép sử dụng kiểu dữ liệu như là tham số (tham số hóa kiểu dữ liệu. * Generic cho phép chúng ta có thể tạo ra các class, method, interface, hoạt động với nhiều kiểu dữ liệu khác nhau. | | |
| 37 | **Ưu điểm và hạn chế của Generic?**   * Ưu điểm:   + Phát hiện lỗi tại thời điểm compile.   + Không cần phải ép kiểu.   + Xây dựng được cá thuật toán tổng quát, tái sử dụng mã nguồn. * Hạn chế: | | |
| 38 | **Stack là gì, các phương thức của Stack?**   * Stack là 1 cấu trúc dữ liệu dạng danh sách, thêm và lấy các phần tử theo quy tắc First In Last Out. * Các phương thức của Stack:    + Push() : thêm phần tử vào danh sách   + Pop() : xóa phần tử trên cùng ra khỏi danh sách và trả về phần tử đó.   + Peek() : lấy phần tử trên cùng nhưng không xóa phần tử đó. | | |
| 39 | **Queue là gì, các class triển khai của Queue ?**   * Queue là 1 cấu trúc dữ liệu dạng danh sách, thêm và lấy các phần tử theo quy tắc First In First Out. * Các class triển khai của Queue:     + GenericQueue   + ArrayQueue   + PiortyQueue | | |
| 40 | **Phương thức cơ bản của queue ?**   * Enqueue() : thêm phần tử vào danh sách. * Dequeue() : lấy và xóa phần tử vào danh sách. * Add() / offer() : thêm phần tử vào danh sách. * Element() / peek() : lấy phần tử đầu tiên mà không xóa khỏi danh sách. * Remove() / poll() : xóa phần tử đầu tiên khỏi danh sách và trả về phần tử đó. | | |
| 41 | **So sánh Comparable và Comparator, khi nào dùng cái nào?**   |  |  | | --- | --- | | **Comparable** | **Comparator** | | Phải implements giao tiếp Comparable cho lớp đối tượng cần được so sánh. | Không phải implements giao tiếp Comparator cho lớp đối tượng cần được so sánh. | | Comparable cung cấp phương thức compareTo() để sắp xếp các phần tử. |  | | Có thể sắp xếp các phần tử kiểu Comparable bởi phương thức Collections.sort (List). | Có thể sắp xếp các phần từ của kiểu Comparator bởi phương thức Collections.sort (List, Comparator). | | Được dùng khi xác định rõ class và thuộc tính cần so sánh của 2 đối tượng thuộc class đó. Comparable chỉ được override lại một lần và chỉ có thể xét được 1 tiêu chí nhất định | Comparator linh hoạt hơn Comparable ở chỗ phương thức compare() của Comparator có thể được gọi nhiều lần và định nghĩa ngay tại vị trí cần so sánh để sắp xếp theo các tiêu chí tùy ý. | | | |
| 42 | **Map là gì , các class triển khai?**   * Map được sử dụng để lưu trữ và truy xuất dữ liệu theo cặp **key** và **value.** * Mỗi cặp key-value còn được gọi là entry. * Map không cho phép 2 key trùng lặp (key được phép “null” nhưng chỉ duy nhất 1 key “null”). * Mỗi key tương ứng sẽ có 1 value (value được phép null). * Các class triển khai:  |  |  |  | | --- | --- | --- | | **HashMap** | **LinkedHashMap** | **TreeMap** | | * HashMap là lớp triển khai phổ biến nhất của Map. * HashMap lữu trữ dữ liệu dưới dạng cặp **key** và **value.** | * LinkedHashMap: giống với HashMap nhưng có duy trì các phần tử theo thứ tự chèn vào. | * TreeMap giống với Map. * Nhưng có các điểm khác sau:   + Không cho phép key có giá trị null.   + Duy trì các phần tử được thêm vào theo thứ tư key được sắp xếp (mặc đinh là sắp xếp tăng dần). | | | |
| 43 | **Cây nhị phân là gì?**   * Các khái niệm về Binary Tree:    + Binary Tree lưu dữ liệu trên các node.   + Các node có mối quan hệ cha con, node trên cùng gọi là node gốc (root node).   + Binary Tree là tree mà mỗi mỗi node có 0,1 và nhiều nhất là 2 node con (sub note).   + 2 node con lần lượt đc gọi là left – subtree và right - subtree * Binary Search Tree:    + Giá trị của tất cả các node ở cây con bên trái phải nhỏ hơn giá trị của node gốc.   + Giá trị của tất cả các node ở cây con bên phải phải lơn hơn giá trị của node gốc. | | |
| 44 | **So sánh Comparable và Comparator, khi nào dùng cái nào?**   |  |  | | --- | --- | | **Comparable** | **Comparator** | | Phải implements giao tiếp Comparable cho lớp đối tượng cần được so sánh. | Không phải implements giao tiếp Comparator cho lớp đối tượng cần được so sánh. | | Comparable cung cấp phương thức compareTo() để sắp xếp các phần tử. |  | | Có thể sắp xếp các phần tử kiểu Comparable bởi phương thức Collections.sort (List). | Có thể sắp xếp các phần từ của kiểu Comparator bởi phương thức Collections.sort (List, Comparator). | | | |
| 45 | **Ngoại lệ (Exception )là gì?**   * **Exception** là 1 sự kiện bất thường xảy ra trong quá trình thực thi 1 chương trình Java nó phá vỡ luồng xử lý bình thường của chương trình, thậm chí gây chết chương trình. * **Erros** là những lỗi quan trọng liên quan đến môi trường thực thi (JVM) của ứng dụng, hệ thống mà lập trình viên không thể kiểm soát. Lỗi này gây chết chương trình và không thể handle. | | |
| 46 | Phân loại Exception?   * Có 2 loại:    + **Unchecked Exception**: là loại exception xảy ra tại thời điểm thực thi chương trình (lỗi này không chắc chắn xảy ra). Loại exception này được bỏ qua trong quá trình compile, không bắt buộc phải hanhdle. Ví dụ: Null Pointer Exception, Array Index Out Of Bound Exception, …   + **Checked Exception**: là loại Exception xảy ra trong lúc compile và bắt buộc phải handle xử lý nó. Ví dụ: File Not Found Exception, Database Exception, … | | |
| 47 | So sánh Comparable và Comparator, khi nào dùng cái nào?   |  |  | | --- | --- | | **Comparable** | **Comparator** | | Phải implements giao tiếp Comparable cho lớp đối tượng cần được so sánh. | Không phải implements giao tiếp Comparator cho lớp đối tượng cần được so sánh. | | Comparable cung cấp phương thức compareTo() để sắp xếp các phần tử. |  | | Có thể sắp xếp các phần tử kiểu Comparable bởi phương thức Collections.sort (List). | Có thể sắp xếp các phần từ của kiểu Comparator bởi phương thức Collections.sort (List, Comparator). | | | |
| 48 | **Có bao nhiêu cách xử lý ngoại lệ?**   * Dùng try – catch (finally) đễ xử lý ngoại lệ ngay tạy đoạn mã bị lỗi. * Dùng throw / throws để ném ngoại lệ cho logic / method khác xử lý    + Throw : được dùng khi cần phát sinh 1 ngoại lệ nào đó.   + Throws : được dùng ở phần khai báo của hàm dùng để nhường việc xử lý ngoại lệ cho method gọi hàm. | | |
| 49 | Một số lưu ý khi dùng try – catch?   * Tại 1 thời điểm chỉ xảy ra 1 ngoại lệ và tại 1 thời điểm chỉ có 1 catch được thực thi. * Các khối catch phải được sắp xếp từ cụ thể nhất đến chung nhất (từ exception con đến exception cha). * Khối **finally** luôn được thực thi dù chương trình có xảy ra ngoại lệ hay không. * Khối **try** có thể không có hoặc có nhiều khối catch nhưng chỉ có 1 khối finally. | | |