**BỘ CÂU HỎI PHỎNG VẤN DOANH NGHIỆP**

**JAVA**

# 1.Switch case?

Mệnh đề switch-case trong java được sử dụng để thực thi 1 hoặc nhiều khối lệnh từ nhiều điều kiện.

- Switch so sánh === hoặc != (không so sánh logic bên trong 1 case: vd >= , =< , >, <)

* 1 biểu thức điều kiện sẽ so sánh vs tất cả các case.
* Switch hỗ trợ các kiểu DL: byte, short, char, int và các Wrapper class tương ứng, enum, String
* Giá trị trả về kiểu Boolean, String,Number.
* Có thể thực hiện nhiều case nếu không gặp lệnh break.
* Lệnh break làm chương trình thoát ra khỏi switch.
* Nếu tất cả các trường hợp đều sai, khối lệnh default sẽ được thực thi. Có thể có hoặc không có khối lệnh default.

# 2.(Trending)Ý nghĩa từ khóa continue, break?

* **Từ khóa break trong java:** được sử dụng để dừng thực thi lệnh trong vòng lặp hoặc trong mệnh đề switch tại điều kiện đã được chỉ định. Đối với vòng lặp bên trong vòng lặp khác, thì nó chỉ stop vòng lặp bên trong đó.
* **Từ khóa continue trong java:** được sử dụng để tiếp tục vòng lặp tại điều kiện đã được xác định. Với điều kiện đó **khối lệnh phía sau từ khóa continue sẽ không được thực thi.** Đối với vòng lặp bên trong một vòng lặp khác, continue chỉ có tác dụng với vòng lặp bên trong đó.

# 3.OOP? Các tính chất và ý nghĩa ? Mỗi tính chất cho một ví dụ.

* **OOP** (viết tắt của Object Oriented Programming) – lập trình hướng đối tượng là một phương pháp lập trình dựa trên khái niệm về lớp và đối tượng. Là 1 kỹ thuật giúp LTV ánh xạ TG thực vào trong ngôn ngữ lập trình.
* Đối tượng **(Object)** trong OOP là 1 thể hiện của class: bao gồm T**huộc tính (Attribute):** là những thông tin, đặc điểm của đối tượng, Phương thức (**Method**): là những hành vi mà đối tượng có thể thực hiện, **Lưu ý:** Đối tượng được khởi tạo bằng từ khóa **new**, đối tượng có thể được khởi tạo nhiều lần. Các **object** khác nhau sẽ có giá trị của **method** và **property** khác nhau để phân biệt với nhau
* **Class** trong **OOP**: là 1 bản thiết kế (**blueprint**): Một lớp là một kiểu dữ liệu bao gồm các thuộc tính và các phương thức được định nghĩa từ trước. Đây là sự trừu tượng hóa của đối tượng. Các đối tượng có các đặc tính tương tự nhau được gom lại thành một lớp đối tượng.VD: ta có lớp **(class) smartphone** gồm 2 thành phần: **Thuộc tính:** màu sắc, bộ nhớ, hệ điều hành… **Phương thức:** gọi điện, chụp ảnh, nhắn tin, ghi âm…**Các đối tượng** của lớp này có thể là: iPhone, Samsung, Oppo, Huawei…Lưu ý: khai báo class = từ khóa Class or bằng function, chỉ đc khai báo 1 lần trong 1 package.

# 4.(Trending)Các tính chất OOP thể hiện qua những thành phần, khái niệm nào?

* **Tính bao đóng (encapsulation):** Tính đóng gói cho phép che giấu thông tin và những tính chất xử lý bên trong của đối tượng. Hạn chế sự truy cập trực tiếp đến dữ liệu bên trong và làm thay đổi trạng thái của đối tượng mà bắt buộc phải thông qua các phương thức công khai do đối tượng đó cung cấp. Tính chất này giúp tăng tính bảo mật cho đối tượng và tránh tình trạng dữ liệu bị hư hỏng ngoài ý muốn. Các từ khóa private, getter/setter sinh ra để thực hiện tính bao đóng. Getter trả về giá trị của object. Setter dùng để thay đổi giá trị thuộc tính cũ của Object. Private dùng để khai báo method, property chỉ đc dùng trong Class đó. **Ví dụ:** một viên thuốc chữa cảm. Chúng ta chỉ biết nó chữa cảm sổ mũi nhức đầu và một số thành phần chính, còn cụ thể bên trong nó có những hoạt chất gì thì hoàn toàn không biết.
* **Tính kế thừa (Inheritance):** Tính kế thừa cho phép xây dựng một lớp mới (lớp Con), kế thừa và tái sử dụng các thuộc tính, phương thức dựa trên lớp cũ (lớp Cha) đã có trước đó. Các lớp Con kế thừa toàn bộ thành phần của lớp Cha và không cần phải định nghĩa lại. Lớp Con có thể mở rộng các thành phần kế thừa hoặc bổ sung những thành phần mới. **Ví dụ:** Lớp Cha là smartphone, có các thuộc tính: màu sắc, bộ nhớ, hệ điều hành…Các lớp Con là iPhone, Samsung, Oppo cũng có các thuộc tính: màu sắc, bộ nhớ, hệ điều hành…và mỗi lớp con có thể có thêm các tính năng khác nhau mà lớp cha smartphone không có.
* **Tính đa hình (Polymorphism):** cho phép các đối tượng khác nhau thực thi hành vi, chức năng giống nhau theo những cách khác nhau tùy theo mỗi thời điểm. VD: chó, mèo đều là lớp động vật.Cùng 1 hành động kêu: con chó sẽ kêu gâu gâu, con mèo sẽ kêu meo meo.
* **Tính trừu tượng (Abstraction):**Tính trừu tượng giúp loại bỏ những thứ phức tạp, không cần thiết của đối tượng và chỉ tập trung vào những gì cốt lõi, quan trọng.Ví dụ: Quản lý nhân viên thì chỉ cần quan tâm đến những thông tin như:Họ tên, Ngày sinh,Giới tính, Trình độ…chứ không cần quan tâm: Chiều cao,Cân nặng, Sở thích,Màu da…
* **Ưu điểm của lập trình hướng đối tượng OOP:**
* OOP mô hình hóa những thứ phức tạp dưới dạng cấu trúc đơn giản.
* Code OOP có thể sử dụng lại, giúp tiết kiệm tài nguyên.
* Giúp sửa lỗi dễ dàng hơn. So với việc tìm lỗi ở nhiều vị trí trong code thì tìm lỗi trong các lớp (được cấu trúc từ trước) đơn giản và ít mất thời gian hơn.
* Có tính bảo mật cao, bảo vệ thông tin thông qua đóng gói.
* Dễ mở rộng dự án.

# 5.(Trending)Phân biệt String & StringBuilder?

Khi làm việc với dữ liệu kiểu text, Java cung cấp 3 class String, StringBuffer và StringBuilder

* String là không thể thay đổi (immutable), và không cho phép có class con + StringBuffer, StringBuilder có thể thay đổi (mutable)
* StringBuilder(không đồng bộ) và StringBuffer(đồng bộ) là giống nhau, nó chỉ khác biệt tình huống sử dụng có liên quan tới đa luồng (Multi Thread).
  + về tốc độ xử lý StringBuilder là tốt nhất, sau đó StringBuffer và cuối cùng mới là String
* Lớp String ghi đè phương thức equals () của lớp Object. Vì thế bạn có thể so sánh nội dung của 2 chuỗi bằng phương thức equals (), Lớp StringBuffer và StringBuilder không ghi đè phương thức equals () của lớp Object.

# 6.So sánh == và equals thì có gì khác nhau?

* Đầu tiên, equals () là một method được định nghĩa bên trong Object class, còn toán tử == là một loại toán tử có thể dùng để so sánh bằng các kiểu dữ liệu nguyên thủy hoặc so sánh bằng địa chỉ vùng nhớ của 2 object
* Khi sử dụng biến kiểu tham trị, JAVA chỉ cho phép bạn sử dụng toán tử so sánh “==”. Bộ xử lý của JAVA sẽ so sánh xem 2 biến tham chiếu này có trỏ đến cùng một đối tượng hay không (So sánh địa chỉ 2 đối tượng và giá trị của các biến nguyên thủy)
* Khi sử dụng biến kiểu tham chiếu, JAVA cho phép sử dụng cả toán tử “==” và equals (). Vì mặc định, equals() được triển khai trong Object class sẽ so sánh địa chỉ object thay vì so sánh giá trị các thuộc tính trong object. Hai đối tượng này là duy nhất và được cấp phát vùng nhớ riêng trong heap. Muốn equals() so sánh giá trị các thuộc tính trong object thay vì so sánh địa chỉ thì phải override lại phương thức equals().

# 7.(Trending)Phân biệt Override, Overload? Ví dụ đã ứng dụng vào bài tập, hoặc project?

**Overriding (ghi đè):** Là cơ chế cho phép lớp con định nghĩa lại nội dung các phương thức trước đó ở lớp cha. Phương thức phải cùng tên, cùng danh sách tham số, cùng kiểu dữ liệu trả về.Access Modifier của lớp con phải có level lớn hơn hoặc bằng lớp cha.Xảy ra trong class có mối quan hệ is-a (quan hệ kế thừa). Đa hình tại runtime. Lưu ý: không ghi đè được phương thức static Vì phương thức static được ràng buộc với class còn phương thức instance được ràng buộc với đối tượng. Static thuộc về vùng nhớ class còn instance thuộc về vùng nhớ heap.

**Overloading (nạp chồng):** Là cơ chế cho phép 1 lớp có khả năng định nghĩa ra nhiều phương thức cùng tên nhưng khác nhau về tham số truyền vào (kiểu dữ liệu, số lượng). Xảy ra trong cùng 1 class. Đa hình tại compile.

* *Câu hỏi 1*: Tại sao không thể nạp chồng phương thức bằng cách chỉ thay đổi kiểu trả về của phương thức? Trong java, không thể nạp chồng phương thức bằng cách chỉ thay đổi kiểu trả về của phương thức bởi vì không biết phương thức nào sẽ được gọi.
* *Câu hỏi 2*: Có thể nạp chồng phương thức main() không? Có, bạn có thể nạp chồng n phương thức main. Nhưng JVM chỉ gọi phương thức main() có tham số truyền vào là một mảng String.

# 8. (Trending)Phân biệt abstract class, interface, class. Ví dụ. Khi nào sử dụng abstract class, interface?

- **Abstract class:** Một lớp được khai báo với từ khóa abstract là lớp **abstract** trong **Java**.Là 1 lớp có tính trừu tượng cao đến mức k thể tạo đc đối tượng.

* + Lớp **abstract** có nghĩa là lớp trừu tượng, nó có thể có các phương thức **abstract** hoặc **non-abstract**
* + Không thể tạo đc đối tượng
* + 1 class có ít nhất 1 phương thức **abstract** thì chắc chắn lớp đó phải là **abstract** (ngược lại thì không)
* + Class **abstract** không thể đi kèm với từ khóa **final**
* + Class **abstract** con **extends** class **abstract** cha thì không cần ghi đè các phương thức **abstract** của cha
* + Từ khóa **private** không dùng cho phương thức **abstract**.

- **Interface:** Một **Interface** trong **Java** là một bản thiết kế của một lớp.

* **+ Interface** là một kỹ thuật để thu được tính trừu tượng hoàn toàn và kế thừa trong **Java**.
* + **Interface** trong **Java** cũng biểu diễn mối quan hệ **IS-A**.
* + Không thể tạo đc đối tượng
* + Các trường ở trong interface đều là hằng số
* + Tất cả các phương thức đều là abstract
* + Hỗ trợ đa kế thừa (1 interface có thể extends từ nhiều interface)
* + Không tạo đc constructor.

[?] So sánh **Abstract** và **Interface**:

* Có tạo đc đối tượng không? => Cả 2 đều không tạo đc đối tượng.
* Trường/ thuộc tính? => Ở Interface đều là hằng số, ở Abstract Class thì như Class thông thường.
* Phương thức (abstract hay non-abstract)? = > Ở Interface đều là abstract, ở Abstract Class có thể là abstract hoặc non-abstract. //trong Java 8, có thể tạo đc phương thức non-abstract trong Interface thông qua default/ static.
* Có constructor không? => Interface không có, Abstract Class có.
* Đa kế thừa? => Interface hỗ trợ đa kế thừa, Abstract Class không hỗ trợ đa kế thừa.

# 9.Sự khác nhau giữa final, finally và finalize

**Final:** Final là một từ khóa. Final được sử dụng để áp dụng các hạn chế về class, phương thức và biến. Lớp final không thể được kế thừa, phương thức final không thể được ghi đè và giá trị biến final không thể thay đổi.

**Finally:** Finally là một khối (block). Finally được sử dụng để thực thi code quan trọng, nó luôn được thực thi cho dù ngoại lệ được xử lý hay không.

**Finalize:** Finalize là một phương thức. Finalize được sử dụng để thực hiện quá trình xử lý xóa trước khi đối tượng thu gom rác được gọi.

# 10.Ý nghĩa từ khóa static?

- Từ khóa static giúp tạo ra các thuộc tính và phương thức sử dụng chung cho tất cả các đối tượng cùng loại, thuộc tính và phương thức static thuộc về class chứ không thuộc về đối tượng.

**Đặc điểm:**

* Khối static sẽ chạy trước hàm “main”, tạo giá trị cho thuộc tính static.
* Có thể áp dụng từ khóa static với thuộc tính (biến của class), phương thức, khối static và static nested class.

**Mục tiêu:**

* Tiết kiệm vùng nhớ.
* Không cần “new” ra đối tượng nhưng vẫn có thể sử dụng phương thức và thuộc tính static
* Giúp xây dựng những tiện ích (util).

**Ràng buộc khi sử dụng static?**

* Phương thức **static** chỉ gọi được các phương thức và thuộc tính **static**.
* Phương thức **non-static** có thể gọi được các phương thức và thuộc tính **static**.
* Phương thức **static** không thể gọi được các phương thức và thuộc tính **non-static**.
* Không thể sử dụng từ khóa **this** & **super** cho phương thức **static**.

# 11.Làm sao để chạy một dòng lệnh trước hàm main? phương thức đó là static.

# 12.interface và abstract khác nhau quan trọng nhất ở chỗ nào?

Cả **interface** và **abstract** class đều là các công cụ quan trọng trong lập trình hướng đối tượng trong Java để giúp tăng tính đa dạng và tái sử dụng của mã. Tuy nhiên, có một số khác biệt chính giữa hai loại này, quan trọng nhất trong số đó là:

**1.Mục đích sử dụng:**

**- Abstract** class được sử dụng khi bạn muốn tạo ra một lớp cha trừu tượng để chứa các phương thức trừu tượng và/hoặc các phương thức có thể được triển khai trong lớp con.

+ Bạn có thể sử dụng abstract class để cung cấp một số mã thực thi chung cho các lớp con của nó.

**- Interface** được sử dụng khi bạn muốn tạo ra một bộ quy tắc hoặc giao diện cho các lớp khác.

+ Một interface chỉ chứa các phương thức trừu tượng, và các lớp khác có thể triển khai các phương thức này theo cách riêng của chúng.

**2.Độ linh hoạt:**

- **Abstract** class cung cấp một mức độ linh hoạt cao hơn cho các lớp con của nó, vì chúng ta có thể triển khai phương thức trừu tượng hoặc không.

+ Ngoài ra, abstract class có thể chứa các thuộc tính và phương thức không trừu tượng.

- **Interface** cung cấp một độ linh hoạt cao hơn cho các lớp khác, vì chúng ta có thể triển khai nhiều interface trong một lớp và các lớp khác có thể triển khai cùng một interface.

+ Một interface không thể chứa bất kỳ phương thức hay thuộc tính cụ thể nào, chỉ có thể chứa các phương thức trừu tượng.

**3.Tính trừu tượng:**

- **Abstract** class có thể chứa các phương thức trừu tượng và các phương thức không trừu tượng.

- **Interface** chỉ có thể chứa các phương thức trừu tượng và các hằng số.

**4.Đa kế thừa**

- **Abstract** không có đa kế thừa

- **Interface** có hỗ trợ đa kế thừa (tuy nhiên trong JV thì interface không được tính là đa kế thừa) + Có constructor không? => **Interface** không có, **Abstract Class** có.

+ Đa kế thừa? => **Interface** hỗ trợ đa kế thừa, **Abstract Class** không hỗ trợ đa kế thừa.

# 13.Phân biệt HashTable và HashMap?

HashMap và Hashtable đều được sử dụng để lưu trữ dữ liệu ở dạng cặp key và value. Cả hai đều đang sử dụng kỹ thuật băm để lưu trữ các khóa duy nhất.

**HashMap**:

1. HashMap cho phép một key là null và nhiều giá trị null
2. HashMap không đồng bộ
3. HashMap là một lớp mới được giới thiệu trong JDK 1.2.
4. HashMap nhanh.
5. HashMap được duyệt bởi Iterator
6. HashMap kế thừa lớp AbstractMap.

**Hashtable:**

1. Hash Table không cho phép bất kỳ key hoặc giá trị null
2. Hashtable là đồng bộ
3. Hashtable là một [lớp legacy](https://viettuts.vn/interview/list-cau-hoi-phong-van-java-collection)
4. Hashtable chậm
5. Hashtable được duyệt bởi Enumerator và Iterator
6. Hashtable kế thừa lớp Dictionary

**Lưu ý:**Phiên bản Java đầu tiên không bao gồm Collection Framework. Nó chỉ định nghĩa một vài lớp và interface cung cấp các phương thức để lưu trữ các đối tượng. Khi Collection Framework được thêm vào trong J2SE 1.2, các lớp gốc đã được tái cấu trúc để hỗ trợ các collection interface. Các lớp này còn được gọi là lớp Legacy. Tất cả các lớp và interface Legacy được thiết kế lại bởi JDK 5 để hỗ trợ Generics.

# 14.Phân biệt TableMap và HashMap?

Trong Java, Table Map và HashMap là hai cấu trúc dữ liệu khác nhau để lưu trữ và quản lý các phần tử dữ liệu. Dưới đây là những khác biệt giữa chúng:

**TableMap:**

1. Là một lớp trừu tượng trong Java.
2. TableMap kế thừa từ AbstractMap và implement Map interface.
3. TableMap cung cấp một bảng giá trị key-value.
4. TableMap được sử dụng để lưu trữ dữ liệu dưới dạng bảng với các hàng và cột.
5. TableMap không phải là một cấu trúc dữ liệu tự động mở rộng, do đó bạn cần phải xác định số lượng các hàng và cột trước khi sử dụng nó.

**HashMap:**

1. Là một cấu trúc dữ liệu bảng băm trong Java.
2. HashMap kế thừa từ AbstractMap và implement Map interface.
3. HashMap cung cấp một bộ đệm băm (hash table) với key-value pairs.
4. HashMap là một cấu trúc dữ liệu tự động mở rộng.
5. HashMap sử dụng hàm băm để tính toán vị trí của key trong bộ đệm băm, do đó độ phức tạp truy cập của HashMap là O(1) trung bình.

**Tóm lại, Table Map và HashMap đều là các cấu trúc dữ liệu để lưu trữ và quản lý các phần tử dữ liệu.**

**Tuy nhiên, TableMap được sử dụng để lưu trữ dữ liệu dưới dạng bảng với các hàng và cột cố định, trong khi HashMap là một cấu trúc dữ liệu tự động mở rộng với độ phức tạp truy cập trung bình là O(1).**

# 15.Quá trình chat giữa hai người trong java xử lý như thế nào?

Trong **Java**, để thực hiện quá trình chat giữa hai người, ta có thể sử dụng một giao thức như **TCP (Transmission Control Protocol** cung cấp một kênh truyền thông đáng tin cậy cho phép ứng dụng client-server sử dụng để giao tiếp với nhau. Để có thể giao tiếp với nhau qua TCP, client và server được thiết lập để kết nối với nhau qua **Socket**.) hoặc **UDP – (User Datagram Protocol:** cung cấp cơ chế vận chuyển không bảo đảm và không nối kết trên mạng IP, ngược với giao thức vận chuyển tin cậy, có nối kết TCP) để thiết lập và duy trì kết nối giữa hai máy tính. Các thư viện Java cung cấp các lớp (như ObjectOutputStream và ObjectInputStream) để hỗ trợ việc tạo kết nối, truyền thông tin và nhận dữ liệu.

**Một số bước thực hiện quá trình chat giữa hai người trong Java có thể là như sau:**

1.Thiết lập kết nối giữa hai máy tính bằng cách sử dụng các lớp của giao thức TCP hoặc UDP.

2.Xác định cách mã hóa và giải mã dữ liệu trao đổi giữa hai máy tính. Các thư viện Java cung cấp các lớp để mã hóa và giải mã dữ liệu, chẳng hạn như ObjectOutputStream và ObjectInputStream.

3.Để bắt đầu quá trình chat, hai máy tính phải xác định ai sẽ gửi và ai sẽ nhận thông điệp. Một trong hai máy tính có thể chủ động gửi thông điệp đầu tiên, hoặc cả hai máy tính có thể đồng thời gửi thông điệp đầu tiên để khởi đầu quá trình trao đổi thông tin.

4.Khi thông điệp được gửi, máy tính gửi sẽ chờ đợi cho đến khi thông điệp được xác nhận và đảm bảo rằng nó đã được gửi thành công. Máy tính nhận sẽ tiếp tục chờ đợi cho đến khi nó nhận được thông điệp từ máy tính gửi và xác nhận rằng thông điệp đã được nhận thành công.

5.Các thông điệp tiếp theo sẽ được gửi và nhận theo cùng một cách thức cho đến khi cuộc trò chuyện kết thúc.

Trên thực tế, các bước thực hiện quá trình chat giữa hai người trong Java có thể phức tạp hơn nếu ta cần xử lý các trường hợp đặc biệt như mất kết nối hoặc xử lý lỗi truyền thông. Tuy nhiên, các thư viện Java cung cấp các lớp và phương thức để xử lý các trường hợp này một cách đáng tin cậy.

# 16.Nếu thay đổi kiểu dữ liệu trả về thì có phải là Overload không? Không.Vì chương trình không biết gọi đến phương thức nào

# 17.(Trending)Các loại access modifier trong java?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Phạm vi** | **Public** | **Protected** | **Default** | **Private** |
| **Từ trong cùng một lớp (class).** | **YES** | **YES** | **YES** | **YES** |
| **Từ bất kì lớp (class) nào trong cùng 1 package** | **YES** | **YES** | **YES** | **NO** |
| **Từ một lớp con (subclass) bên ngoài gói (package)** | **YES** | **YES, thông qua kế thừa** | **NO** | **NO** |
| **Từ bất kì một lớp(class) bên ngoài gói (package)** | **YES** | **NO** | **NO** | **NO** |

=> Một Class chỉ có thể sử dụng 2 cấp độ kiểm soát truy cập (default hoặc public), các thành viên có thể sử dụng cả 4 cấp độ kiểm soát truy cập.

- **Nested class** – là một class được khai báo bên trong một class khác, còn class khác này được gọi là outer class. Có 2 loại: + Inner class. + Static nested class.

# 18.Method có dữ liệu trả về và không có dữ liệu trả về trong Java?

**Return method:** có KDL trả về khi khai báo phương thức VD: Double Area(){ return}.Có thể có hoặc không có tham số. Kết thúc khối block scope phải có từ khóa return về kết quả đúng KDL của method.

**Void method:** được khai báo từ khóa void VD: void Area(){} .Có thể có hoặc không có tham số

# 19.Class là gì? Interface là gì?

**Class:** là 1 bản thiết kế (blueprint) cho đối tượng: Một lớp là một kiểu dữ liệu bao gồm các thuộc tính và các phương thức được định nghĩa từ trước.

+ Đây là sự trừu tượng hóa của đối tượng. Các đối tượng có các đặc tính tương tự nhau được gom lại thành một lớp đối tượng

**Interface:** Một Interface trong Java là một bản thiết kế của một lớp.

+ Nó chỉ có các phương thức trừu tượng (để các lớp triển khai nó dùng chung)

+ Interface là một kỹ thuật để thu được tính trừu tượng hoàn toàn và đa kế thừa trong Java.

+ Interface trong Java cũng biểu diễn mối quan hệ IS-A

# 20.Khi nào thì biến static được cấp phát vùng nhớ?

Một biến đi cùng với từ khóa **static** sẽ chỉ được cấp phát vùng nhớ một lần duy nhất tại thời điểm tải class. Và nó thuộc về class chứ không thuộc về bất kỳ một object nào cụ thể.

# 21.(Trending)Em hiểu gì về constructor?

**Constructor:**

1. Là phương thức đặc biệt giúp tạo ra đối tượng
2. Không có kiểu dữ liệu trả về
3. Trùng tên với tên Class
4. Trong 1 class có thể có nhiều constructor (overload => nạp chồng: trùng tên method, khác kiểu dữ liệu hoặc số lượng tham số)
5. 1 constructor có thể gọi 1 và chỉ 1 constructor khác, nhưng nó phải đc gọi đầu tiên trong body.
6. Nếu như không tạo ra constructor nào thì Java hỗ trợ tạo constructor mặc định.

# 22.Kiểu dữ liệu nguyên thủy và đối tượng được lưu ở đâu? Kiểu dữ liệu String thì lưu ở đâu?

1. Kiểu DL nguyên thủy được lưu ở bộ nhớ STACK
2. Heap memory là bộ nhớ được sử dụng bởi Java Runtime để cấp phát bộ nhớ cho các đối tượng (object) và String.
3. Kiểu đối tượng: biến đc lưu ở vùng nhớ stack, miền giá trị được lưu ở vùng nhớ heap.
4. Kiểu String: biến dc lưu ở vùng nhớ stack. Khởi tạo miền giá trị bằng từ khóa new thì miền giá trị dc lưu ở vùng nhớ Heap, còn khởi tạo giá trị bằng string literal thì miền giá trị lưu ở vùng nhớ String pool nằm trong Heap

# 23.Khái niệm Collections và Collection?

1. **Collections** trong java là một khuôn khổ cung cấp một kiến trúc để lưu trữ và thao tác tới nhóm các đối tượng. Tất cả các hoạt động mà bạn thực hiện trên một dữ liệu như tìm kiếm, phân loại, chèn, xóa, ... có thể được thực hiện bởi Java Collections.
2. **Collection trong java** là một root interface trong hệ thống cấp bậc Collection. Java Collection cung cấp nhiều interface (Set, List, Queue, Deque vv) và các lớp (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet vv).

**Collections Framework** là một bộ sưu tập các lớp và giao diện trong Java cung cấp các cấu trúc dữ liệu để lưu trữ, truy xuất và xử lý các đối tượng. Nó cung cấp một khung làm việc để quản lý các đối tượng như danh sách, tập hợp, bản đồ và hàng đợi.

# 24.Set là gì? Các class triển khai?

**Set là một interface kế thừa Collection interface trong java. Set trong java là một Collection không thể chứa các phần tử trùng lặp.**

**Set được triển khai bởi Hashset, LinkedHashset, Treeset hoặc EnumSet.**

* **HashSet** lưu trữ các phần tử của nó trong bảng băm, là cách thực hiện tốt nhất, tuy nhiên nó không đảm bảo về thứ tự các phần tử được chèn vào.
* **TreeSet** lưu trữ các phần tử của nó trong một cây, sắp xếp các phần tử của nó dựa trên các giá trị của chúng, về cơ bản là chậm hơn HashSet.
* **LinkedHashSet** được triển khai dưới dạng bảng băm với có cấu trúc dữ liệu danh sách liên kết, sắp xếp các phần tử của nó dựa trên thứ tự chúng được chèn vào tập hợp (thứ tự chèn).
* **EnumSet** là một cài đặt chuyên biệt để sử dụng với các kiểu enum.

# 25.Đã dùng Hashtable chưa? Khác gì so với các class khác?

**- Em chưa sử dụng Hashtable.** Trong Java, **Hashtable** là một lớp cũ và đã lỗi thời, thay vào đó nên sử dụng **HashMap** thay thế.

- **Hashtable** là một lớp cài đặt của cấu trúc dữ liệu bảng băm (**hash table**), được sử dụng để lưu trữ và truy xuất các cặp **key-value**.

- **Hashtable** là một lớp đồng bộ (synchronized), điều này có nghĩa là chỉ có một luồng được phép truy cập vào Hashtable cùng một thời điểm.

- Lớp này cung cấp các phương thức để thêm, lấy, cập nhật và xóa các cặp **key-value**.

- Mỗi key phải là duy nhất và không thể là **null**. Giá trị (value) có thể là bất kỳ đối tượng nào, bao gồm **null**.

- **Hashtable** sử dụng hàm băm (hash function) để tính toán một chỉ số cho mỗi key, từ đó lưu trữ giá trị tương ứng vào vị trí tương ứng trong bảng băm.

- Vì lớp **Hashtable** là đồng bộ, nên nó có thể được sử dụng trong các ứng dụng đa luồng (**multithreaded**) mà có nhiều luồng truy cập đến cùng một đối tượng **Hashtable**.

# 26.Hàm dùng để dọn rác những biến không còn sử dụng nữa trong java là gì?

**Phương thức gc():Garbage Collection.** Phương thức gc () được sử dụng để gọi bộ thu gom rác để thực hiện quá trình dọn dẹp.

- Phương thức gc() được cài đặt trong các lớp System và Runtime.

- Garbage Collection được thực hiện bởi một luồng hiểm gọi là Garbage Collector (GC). Luồng này gọi phương thức finalize() trước khi đối tượng thu gom rác.

# 27.So sánh Array với ArrayList?

|  |  |
| --- | --- |
| Array | ArrayList |
| 1/Kích thước cố định  2/Chỉ có thuộc tính **length**  **3/** Có thể lưu kiểu nguyên thủy lẫn đối tượng  4/Tốc độ thao tác nhanh hơn ArrayList | 1/Kích thước có thể thay đổi  2/Có nhiều phương thức hỗ trợ  3/Chỉ cho phép lưu vào kiểu đối tượng. Muốn sử dụng kiểu nguyên thủy thì phải sử dụng Wrapper class (Kể từ Java 5, kiểu nguyên thủy được tự động chuyển đổi trong các đối tượng được gọi là **auto-boxing**)  4/Tốc độ lưu trữ và thao tác chậm hơn (do chứa nhiều phương thức) |

# 28.Khi khởi tạo một đối tượng thì thành phần nào trong class được tạo đầu tiên?

- Khi khởi tạo một đối tượng trong Java, trước tiên các thành phần dữ liệu trong lớp (class) được tạo ra, được khởi tạo bằng giá trị mặc định cho kiểu dữ liệu tương ứng của chúng.

- Sau đó, các phương thức khởi tạo (constructor) trong lớp được gọi để khởi tạo giá trị cho các thành phần dữ liệu này.

# 29.Nếu có ghi kèm phiên bản (java bản 8 hay js ES 6 ) trong CV, thì phải chuẩn bị trước giải thích xem mấy phiên bản đó có gì khác so với các phiên bản trước/sau nó?

**JAVA 8:** Những tính năng mới nhất nổi tiếng và phổ biến nhất của Java SE 8 là gì?

**1/** Giao diện chức năng

**2/** Cải tiến API Bộ sưu tập

**3/** Biểu thức Lambda

**4/** Spliterator

**5/** API luồng và hơn thế nữa.

**Điều gì làm cho Java SE 8 vượt trội hơn các phiên bản khác?**

1/Nó viết mã song song

2/Nó cung cấp nhiều mã có thể sử dụng hơn

3/Nó đã cải thiện các ứng dụng hiệu suất

4/Nó có nhiều mã ngắn gọn và dễ đọc hơn

5/Có hỗ trợ viết cơ sở dữ liệu bao gồm cả các chương trình khuyến mãi.

**JS ES6:** ES6 (là từ viết tắt của ECMAScript 6) được định nghĩa là phiên bản mới nhất chuẩn ECMAScript. ECMAScript được đề xuất bởi hiệp hội sản xuất máy tính Châu Âu để làm tiêu chuẩn cho ngôn ngữ lập trình Javascript.

Lý do nên sử dụng ES6 là gì?ES6 là một trong những phiên bản tập hợp toàn bộ các kỹ thuật nâng cao của Javascript và nó được xem là chuẩn mực để có thể phát triển dựa theo đúng hướng.

**Các tính năng nổi bật của ES6 là gì?**

**1/Arrow Function in ES6:**

**2/ Block-Scoped Constructs Let and Const:** biến chỉ tồn tại trong một phạm vi khối của Block Scope. Những khối lệnh này sẽ được thực hiện định nghĩa thông qua các cặp ngoặc nhọn

**3/Rest Parameter in ES6:**Khi bạn tiến hành khai báo cho một hàm số lượng không xác định những tham số tham gia

**4/Destructuring Assignment:** sẽ tách ra các phần từ Object hoặc Arry chỉ bằng một đoạn code nhất định để chia thành nhiều biến nhỏ

**5/Default Parameters:** Với phiên bản ES6 này thì bạn có thể thực hiện khai báo đối với các giá trị mặc định đơn giản hơn rất nhiều so vs các phiên bản trước

**6/Template Literals in ES6:**hỗ trợ cho các lập trình viên tạo Template HTML rất đơn giản và dễ dàng. Template Literals đối với các ngôn ngữ khác là một dạng hiển thị của các biến ở trong chuỗi. và đặc biệt có thể sử dụng các cú pháp mới ở bên trong của chuỗi

**7/Promises in ES6:** có sẵn một chuẩn Promises

**8/Class:** hỗ trợ lập trình hướng đối tượng và nó cho phép người sử dụng có thể viết Class và kế thừa nó một cách dễ dàng hơn

# 30.Tính đa hình (polymorphism) còn được thể hiện ở đâu, ngoại trừ Overriding và Overloading? ép kiểu

# 31. Các kiểu dữ liệu nguyên thủy trong Java? boolean, byte, short, int, long, float, double, char.

# 32. So sánh Boolean và boolean? Lúc nào thì dùng Boolean, còn lúc nào thì dùng boolean?

1. **boolean** là kiểu dữ liệu nguyên thủy; có 2 giá trị **true**, **false**. Mặc định là **false**.
2. **Boolean** là kiểu dữ liệu **Wrapper Class** của **boolean**; có 3 giá trị **true**, **false**, **null**. Mặc định là **null**.
3. Trong hầu hết các trường hợp, ta nên sử dụng **boolean** để tiết kiệm bộ nhớ và tăng tốc độ xử lý.
4. Tuy nhiên, trong một số trường hợp, ta cần sử dụng **Boolean** để có thể gán giá trị **null** cho biến.
5. Ví dụ: Sử dụng Boolean khi xét giới tính **true** -> nam, nữ -> **false**, khác -> **null**.

# 33. Một biến int x vừa được khai báo thì có giá trị gì? Biến Integer y vừa khai báo có giá trị gì? Tại sao x, y lại có giá trị đó?

1. Biến **int** x có giá trị 0 (mặc định của int).
2. Biến **Integer** y có giá trị là **null** (mặc định của KDL đối tượng)

# 34. Các cách để duyệt mảng và Collection trong Java? Có biết Stream ko?

1. Sử dụng vòng lặp for: duyệt mảng và **Collection**.
2. Sử dụng vòng lặp foreach, Iterator: chỉ dùng cho **Collection**.

**Stream** là một đối tượng được giới thiệu trong Java 8, cho phép thực hiện các thao tác trên **Collection** một cách dễ dàng và linh hoạt.

1. **Stream** cho phép ta thực hiện các thao tác như **filter**, **map**, **reduce**, v.v. trên **Collection** một cách dễ dàng hơn.
2. Cú pháp sử dụng **Stream** thường gồm hai bước: tạo **Stream** và thực hiện các thao tác trên **Stream** đó.

# 35. (Trending)Nêu các thành phần của Collection?

* 1. **Interface**: là một khái niệm trừu tượng đại diện cho các loại cấu trúc dữ liệu khác nhau trong Java Collection Framework, chẳng hạn như List, Set, Queue và Map.
  2. **Class**: là các lớp cụ thể được cài đặt bởi interface Collection, chẳng hạn như ArrayList, LinkedList, HashSet và TreeMap.
  3. **Iterator**: là một interface được sử dụng để duyệt qua các phần tử của Collection, cho phép lấy ra từng phần tử một và thực hiện các thao tác trên nó.
  4. **Set**: là một interface trong Java Collection Framework, đại diện cho một tập hợp các phần tử không có thứ tự và không được phép chứa các phần tử trùng lặp.
  5. **List**: là một interface trong Java Collection Framework, đại diện cho một danh sách các phần tử có thứ tự và có thể chứa các phần tử trùng lặp.
  6. **Map**: là một interface trong Java Collection Framework, đại diện cho một bộ sưu tập các phần tử key-value, trong đó mỗi key tương ứng với một giá trị value.
  7. **Queue**: là một interface trong Java Collection Framework, đại diện cho một hàng đợi các phần tử, nơi các phần tử được thêm vào vào cuối hàng đợi và được lấy ra từ đầu hàng đợi.
  8. **Deque**: là một interface trong Java Collection Framework, đại diện cho một deque (double-ended queue) - một hàng đợi hai đầu có thể thêm và lấy phần tử từ cả hai đầu của hàng đợi.
  9. **Collection utilities**: là các class hỗ trợ sử dụng cho việc thao tác và xử lý Collection, chẳng hạn như Collections, Arrays, và Comparator.

# 36 Phân biệt ArrayList và LinkedList? Có biết về Vector ko?

**ArrayList:**

1. Sử dụng mảng động để lưu trữ phần tử.
2. Là cấu trúc chỉ mục, mỗi phần tử được liên kết với một chỉ mục, các phần tử sắp xếp liền kề nhau trong ô nhớ.
3. Thêm xoá chậm vì khi thực hiện thêm hoặc xoá cần sắp xếp lại mảng (đảm báo tính liền kề)
4. Truy xuất phần tử nhanh hơn vì các phần tử được lưu dựa trên chỉ mục
5. Có thể truy xuất ngẫu nhiên phần tử.
6. Cần ít bộ nhớ hơn.
7. Tốt trong lưu trữ và truy xuất dữ liệu. Dùng vs các bảng ít thay đổi dữ liệu

**Linkedlist**

1. Sử dụng danh sách liên kết (lk đôi) để lưu trữ các phần tử.
2. Các phần tử là node mỗi node chưa tham chiếu phần tử trước nó, giá trị của phần tử và 1 tham chiếu tới phần tử kế tiếp
3. Thêm xoá nhanh hơn vì không cần sắp xếp lại chỉ cần cập nhập lại tham chiếu tới phần tử trước và sau nó (nhanh hơn ở đầu mảng)
4. Truy xuất lâu hơn vì nó phải duyệt qua từng phần tử từ đầu đến cuối.
5. Không thể truy xuất ngẫu nhiên. Phải duyệt qua tất cả các phần tử từ đầu tiên đến cuối cùng để tìm phần tử
6. Cần nhiều bộ nhớ hơn vì phải lưu các tham chiếu theo mỗi phần tử.
7. Tốt trong thao tác với dữ liệu như thêm sửa xoá. Dùng vs bảng có dữ liệu hay thay đổi

**Vector**

1. Vector có các phương thức thực hiện thêm, xóa, sắp xếp, tìm kiếm, truy cập các phần tử trong danh sách, tương tự như các cấu trúc dữ liệu ArrayList và LinkedList.
2. Vector được đồng bộ hóa, nên nó có thể chậm hơn so với các cấu trúc dữ liệu khác khi sử dụng trong các ứng dụng không đa luồng.
3. Do đó, nếu không cần sử dụng tính năng đồng bộ hóa của Vector, thì nên sử dụng ArrayList hoặc LinkedList thay vì Vector.

# 37. Phân biệt Comparable và Comparator?

**- Comparable** chỉ sắp xếp theo 1 tiêu chí. => **Class** phải **implements comparable**<E>. => **Override** phương thức **compareTo**( ).Thư viện **java.lang.**Ảnh hưởng thay đổi class muốn sắp xếp.Cú pháp sử dụng **Collection.sort(List)**

**- Comparator** sắp xếp theo 1 hoặc nhiều tiêu chí (vd: sắp xếp theo tên, điểm, id,...). => **Class** phải **implements comparator**<E> hoặc tạo một class riêng **implement interface Comparator** để so sánh các đối tượng. => **Override** phương thức **compare**( ).Thư viện java,until.Không ảnh hưởng đến class muốn sắp xếp.Cú pháp sử dụng **Collection.sort(List,Comparator).**

# 38. Sự khác nhau giữa kiểu dữ liệu nguyên thủy và kiểu Object?

1. Kiểu dữ liệu nguyên thủy được lưu trữ trực tiếp trong **stack** (lưu trữ tên biến và giá trị).
2. Trong khi đó kiểu **Object** được lưu trữ thông qua một tham chiếu (**reference**), lưu trữ tên biến và giá trị là địa chỉ tham chiếu sang bộ nhớ **heap**.
3. Kiểu dữ liệu nguyên thủy không phải là đối tượng, do đó không có các phương thức và thuộc tính như kiểu **Object**.
4. Kiểu dữ liệu nguyên thủy không thể tham gia vào các hoạt động của **collection framework**, trong khi các đối tượng được kế thừa từ lớp **Object** có thể được thêm vào các **collection**.

# 39. Để so sánh 2 biến kiểu Object? Sử dụng equal và hashcode ()

# 40. Em biết gì về Heap và Stack?

-------------HEAP-------------

* 1. Heap memory là bộ nhớ được sử dụng bởi Java Runtime để cấp phát bộ nhớ cho các đối tượng (object) và String.
  2. Bất kỳ khi nào có một đối tượng được tạo, thì miền giá trị của nó sẽ được tạo lưu ở bộ nhớ Heap.
  3. Bộ dọn rác (Garbage Collection) chạy trên heap memory để giải phóng bộ nhớ nếu không có đối tượng nào tham chiếu tới.

--------------STACK--------------

1. Stack memory bao gồm các giá trị cụ thể của method: các biến local và các tham chiếu tới các đối tượng chứa ở trong heap memory được tham chiếu bởi method.
2. Stack memory được tham chiếu theo thứ tự LIFO (Last In First Out – vào cuối cùng thì ra đầu tiên). Tức là lưu trữ kiểu ngăn xếp (stack).
3. Khi có một method được thực thi, một block được tạo ra trong stack memory để chứa các biến nguyên thủy local và các tham chiếu tới các object.
4. Khi method kết thúc, block đó sẽ không còn được sử dụng và được phục vụ cho method tiếp theo.
5. Stack memory có kích thước rất nhỏ so với Heap memory.

# 41. So sánh do-while và while?

WHILE: - while (điều kiện lặp) {nội dung lặp; } - Dùng khi k biết trước số lần lặp. - Có thể k đc thực thi nếu k thỏa đk từ đầu. DO WHILE: - do {nội dung lặp;} while (điều kiện lặp); - Không biết trước số lần lặp. - Được thực hiện ít nhất 1 lần kể cả khi không thỏa đk.

# 42. Phân biệt List , Set , Map

1. **List:** Collection có lẽ là được dùng nhiều nhất. Là cấu trúc dữ liệu tuyến tính, trong đó các phần tử đc sắp xếp theo 1 thứ tự xác định, cho phép các phần tử đc trùng lặp nhau.. Bởi vì có thứ tự, ta có thể truy cập các phần tử thông qua chỉ số.
2. **Set:** Là cấu trúc dữ liệu tuyến tính, là một tập các phần tử không có thứ tự và không hỗ trợ lưu các phần tử trùng lặp.
3. **Map:** một tập các cặp key - value với key trong map là duy nhất (Map Interface không kế thừa từ Collection Interface mà đây là 1 interface độc lập với các phương thức của riêng mình).
   * List lưu phần tử trùng lặp, Set & Map không lưu phần tử trùng lặp
   * List & Set lưu dưới dạng phần tử, Map lưu dưới dạng key-value

# 43. (Trending)Phân biệt Class và Object?

* Một Class là một Blueprint (kế hoạch) hay Prototype (nguyên mẫu) xác định biến và các phương thức (hay function) chung với tất cả các đối tượng cùng loại.
* Một Object (đối tượng) là một cụ thể, thể hiện của một Class.. Mỗi đối tượng khác nhau có các giá trị thuộc tính và hành vi khác nhau để phân biệt giữa các đối tượng với nhau.
* Khởi tạo class bằng từ khoá Class.Khởi tạo đối tượng bằng từ khoá new.
* Class đc khởi tạo 1 lần trong 1 pakage, còn object thì không giới hạn.

# 44. Từ khóa static dùng được ở đâu?

Trong Java, từ khóa static được sử dụng để xây dựng các thuộc tính (**property**), phương thức (**method**) dùng chung cho các đối tượng hoặc khối mã (**block**).**Static** thuộc lớp, chứ không thuộc về một đối tượng cụ thể của lớp đó.

Từ khóa **static** có thể được sử dụng ở các vị trí sau:

- **Thuộc tính (property)** **static**: Một thuộc tính static là một biến được khai báo với từ khóa static. Thuộc tính static được chia sẻ bởi tất cả các đối tượng của lớp và có thể được truy cập mà không cần tạo đối tượng.

- **Phương thức (method) static:** Một phương thức static là một phương thức được khai báo với từ khóa static. Phương thức static có thể được gọi bằng cách sử dụng tên lớp, không cần tạo đối tượng.

- **Khối mã (block) static:** Một khối mã static là một khối mã được khai báo với từ khóa static. Khối mã static được thực thi khi lớp được tải vào bộ nhớ và chỉ được thực thi một lần trong vòng đời của chương trình.

- Ngoài ra, từ khóa **static** cũng có thể được sử dụng trong các khái niệm khác như lớp lồng nhau **(nested class)** và gói **(package).**

# 45. Phân biệt throw, throws, try catch?

**THROW / THROWS:**

- Sử dụng **throws** không giải quyết được ngoại lệ, nó chỉ bắt được ngoại lệ rồi ném ngoại lệ ra để phương thức khác xử lý ngoại lệ đó.

**Throw**:

+ Nằm bên trong hàm/phương thức.

+ Dùng để trả về một exception được xác định.

+ Nếu là checked exception thì cần đi chung với throws.

+ Nếu là unchecked exception thì không cần đi chung với throws.

+ Theo sau throw là duy nhất một trường hợp/thực thể(instance) ngoại lệ.

**Throws**:

+ Nằm ngay sau phần khai báo tên hàm/phương thức.

+ Dùng để thông báo sự tồn tại của exception.

+ Nếu là checked exception thì cần đi chung với throw.

+ Nếu là unchecked exception thì không cần throw.

+ Theo sau throws có thể là một hoặc nhiều lớp(class).

**TRY - CATCH:** xử lí ngay tại nơi xảy ra **exception**, chủ động hơn trong việc bắt lỗi và fix bug.

- Tại một thời điểm chỉ xảy ra một ngoại lệ và tại một thời điểm chỉ có một khối catch được thực thi. Khi exception đã bị bắt ở một catch thì các catch tiếp theo sẽ không được bắt.

- Tất cả các khối catch phải được sắp xếp từ cụ thể nhất đến chung nhất (từ exception con đến exception cha), tức là phải khai báo khối lệnh catch để xử lý lỗi NullPointerException, ArithmeticException, … trước khi khai báo catch để xử lý lỗi Exception.

- Khối lệnh finally luôn được thực thi dù chương trình có xảy ra ngoại lệ hay không (ngay cả sử dụng lệnh return).

- Đối với mỗi khối try, có thể không có hoặc nhiều khối catch, nhưng chỉ có một khối finally.

- Khối finally sẽ không được thực thi nếu chương trình bị thoát bằng cách gọi System.exit() hoặc xảy ra một lỗi (Error) không thể tránh khiến chương trình bị chết.

# 46. Ý nghĩa của Regex?

- Regular Expression (Regex) là 1 chuỗi được sử dụng để truy định dạng thức của 1 chuỗi khác, đc sử dụng cho validate (xác thực) dữ liệu và tìm kiếm.

- Là một giao diện lập trình để xác định các mô hình dùng để tìm kiếm hoặc thao tác chuỗi.

- Nó được sử dụng rộng rãi để định nghĩa các khuôn mẫu cho chuỗi ví dụ như mật khẩu, email, url,.... Hoặc được dùng để thay thế (replace) các chuỗi ký tự theo mẫu.

# 47. 'Mảng 2 chiều. Đường chéo mảng 2 chiều?

**Mảng 2 chiều:** - Mảng hai chiều là mảng có 2 chỉ số để lưu trữ các giá trị (chẳng hạn giá trị của một bảng có m dòng, n cột).

**Đường chéo mảng 2 chiều:** - Trường hợp mảng hai chiều có đường chéo chính và đường chéo phụ chỉ tồn tại khi số hàng bằng số cột (có nghĩa là ma trận vuông). Đặc điểm của các phần tử nằm trên đường chéo chính của ma trận vuông là chỉ số hàng luôn bằng chỉ số cột.

# 48.(Trending) Map và HashMap: nếu put vào 2 giá trị cùng key vào hashmap thì sao?

Nếu bạn thêm 2 giá trị cùng key vào HashMap, thì giá trị cuối cùng sẽ được giữ lại và ghi đè lên giá trị trước đó.Vì Map không cho chứa key trùng lặp.

# 49. Chẳng hạn chiếc máy tính, đâu là thuộc tính, đâu là hành vi?

1. Thuộc tính: màu sắc, kích thước, trọng lượng, nhà sản xuất, tên máy, kiểu dáng, hệ điều hành
2. Hành vi: lướt web, code, quay video, gập mở

# 50. Ví dụ rõ về triển khai Interface trong thực tế và Override?

Giả sử bạn đang xây dựng một ứng dụng quản lý nhân viên. Bạn có các lớp Employee và Manager, và bạn muốn cho phép tính lương cho cả hai loại nhân viên.

+Bạn có thể tạo một Interface Payable để định nghĩa các phương thức cần thiết để tính lương:

Sau đó, bạn có thể triển khai Interface này trong lớp Employee và Manager bằng cách Override phương thức calculateSalary() để tính lương cho từng loại nhân viên:

+Triển khai Interface này, bạn có thể sử dụng đa hình để tính lương cho một danh sách các đối tượng nhân viên, bao gồm cả Employee và Manager:

# 51.(Trending) Ngoại lệ là gì? có mấy loại ngoại lệ? các loại thông báo lỗi mà ngoại lệ thường hay đưa ra?

1/ Ngoại lệ (Exception) là 1 sự kiện bất thường, làm phá vỡ luông thực thi của chương trình và có thể làm chết chương trình

2/ Phân loại:

* Checked exception: xảy ra tại thời điểm compile, bắt buộc phải xử lý
* Unchecked exception: xảy ra lúc runtime, không chắc chắn xảy ra, không bắt buộc phải xử lý(handle) tại thời điểm compile.
* Error: lỗi do môi trường thực thi (JVM), không thể handle. Chương trình sẽ chết (không được tính là ngoại lệ)

3/ 1 số exception hay xảy ra:

* FileNotFoundException: 1 file được chỉ định với đường dẫn không tồn tại hoặc không có quyền truy cập vào file đó.
* ClassNotFoundException : không thể chuyển kiểu object này sang kiểu object khác hoặc không tìm thấy class muốn tham chiếu tới
* NullPointerException: khi chương trình cố gắng tham chiếu tới một đối tượng nhưng nó không có vị trí nào trên bộ nhớ, tức là có giá trị null
* ArrayIndexOutOfBoundsException: truy nhập mảng với chỉ số không hợp lệ
* StackOverflowError: tràn vùng nhớ stack, stack đạt đến giới hạn tối đa (Error)

# 52.(Trending) Java do công ty nào viết ra? Java 8 khác gì so với Java 7?

1/ **Nguồn gốc java:** Java ban đầu được James Gosling tại Sun Microsystems (sau đó đã được Oracle mua lại) phát triển và được phát hành vào năm 1995 như một thành phần cốt lõi của nền tảng Java của Sun Microsystems

**2/So sánh Java 8 và Java 7:**

**Java 8**

+ phát triển bởi Tập đoàn Oracle 18 Tháng ba, 2014

+ Tên mã: Spider + Java 8 không được hỗ trợ chính thức trên WinXp

+ Đặc trưng: Một số tính năng của Java 8 là Biểu thức Lambda, Mặc định giao diện và Phương thức tĩnh,

+ Nâng cấp chính là biểu thức lambda

**Java 7:**

+ Được phát triển bởi Tập đoàn Oracle vào ngày 27 tháng 1 năm 2010

+ Tên mã: Dolphin

+ hỗ trợ trên WindowsXP

+ Đặc trưng: Một số tính năng của Java 7 là Xử lý nhiều ngoại lệ, Cập nhật kiến ​​trúc trình tải lớp, Suy luận kiểu nâng cao, Rowset 1.1 và JDBC 4.1 đã cập nhật

+ Nâng cấp chính: Hỗ trợ JVM cho các ngôn ngữ kiểu động.

# 53. Để chạy 1 file Java thì mình cần môi trường như thế nào?

* Để chạy một file Java, bạn cần có môi trường Java (JRE hoặc JDK) được cài đặt trên máy tính của mình.
* Môi trường này cung cấp các thư viện và trình biên dịch cần thiết để chạy và thực thi các ứng dụng Java.

# 54. Khách hàng muốn chạy 1 class Java bình thường thì cần những gì?

1/Cài đặt Java Development Kit (JDK) trên máy tính của mình. JDK là bộ công cụ phát triển Java, bao gồm môi trường thực thi Java (JRE), trình biên dịch và các công cụ hỗ trợ phát triển.

2/Tạo một file .java chứa mã nguồn của class Java mà khách hàng muốn chạy.

3/Biên dịch file .java bằng trình biên dịch Java bằng cách sử dụng lệnh javac trong dòng lệnh hoặc IDE của khách hàng.

4/Nếu quá trình biên dịch thành công, một file .class sẽ được tạo ra. Đó là file chứa byte code của class Java mà khách hàng muốn chạy.

5/Chạy file .class bằng cách sử dụng lệnh java trong dòng lệnh hoặc IDE

# 55. Làm thế nào để gọi 1 phương thức static bên ngoài class?

Để gọi một phương thức static từ bên ngoài class, ta sử dụng tên class để truy cập phương thức đó.

# 56. Cách triển khai Comparable?

1/Tạo lớp muốn triển khai Comparable interface và implement phương thức compareTo() của nó. Phương thức này trả về một giá trị nguyên để chỉ ra sự so sánh giữa đối tượng hiện tại với đối tượng được chuyền vào.

2/Trong phương thức compareTo(), ta so sánh đối tượng hiện tại với đối tượng được chuyền vào bằng cách sử dụng các điều kiện so sánh phù hợp để trả về giá trị nguyên thể hiện mối quan hệ giữa hai đối tượng.

1. Nếu giá trị trả về là số dương, đối tượng hiện tại lớn hơn đối tượng được chuyền vào.
2. Nếu giá trị trả về là số âm, đối tượng hiện tại nhỏ hơn đối tượng được chuyền vào.
3. Nếu giá trị trả về là 0, đối tượng hiện tại bằng với đối tượng được chuyền vào.

3/ => Sử dụng phương thức compareTo() để so sánh các đối tượng của lớp đó

# 57. Triển khai comparable và comparator để sắp xếp điểm của học sinh (tăng hoặc giảm). Nói rõ ràng từ lúc tạo class -> add vào list -> sắp xếp.

1/Tạo lớp Student có các thuộc tính như tên, điểm số,có đầy đủ getter/setter,constructor

2/Triển khai Comparable interface trong lớp Student và Overridephương thức compareTo() để so sánh điểm số giữa các học sinh public class Student implements Comparable<Student> {

@Override public int compareTo(Student otherStudent) {

if (this.score > otherStudent.getScore()) { return 1; }

else if (this.score < otherStudent.getScore()) { return -1; }

return 0; } }

3/Tạo lớp SortComparator để triển khai Comparator interface và Override phương thức compare() để so sánh điểm số giữa các học sinh.

public class SortComparator implements Comparator<Student> {

@Override public int compare(Student student1, Student student2) {

if (student1.getScore() < student2.getScore()) { return 1; }

else if (student1.getScore() > student2.getScore()) { return -1; }

return 0; }}

4/Tạo một danh sách List<Student> và thêm các đối tượng Student vào danh sách.

List<Student> students = new ArrayList<>();

students.add(new Student("Alice", 85));

students.add(new Student("Bob", 75));

students.add(new Student("Charlie", 90));

students.add(new Student("David", 80));

5/Sắp xếp danh sách sinh viên theo thứ tự tăng dần hoặc giảm dần bằng cách sử dụng phương thức sort() của Collections và truyền vào một đối tượng Comparator: Collections.sort(students); or Collections.sort(students, new SortComparator)

# 58. Trình bày về static (biến, class, phương thức). Ví dụ phương thức static trong lập trình.

Từ khóa static giúp tạo ra các thuộc tính và phương thức sử dụng chung cho tất cả các đối tượng cùng loại, thuộc tính và phương thức static thuộc về class chứ không thuộc về đối tượng.

- Đặc điểm:

1. Khối static sẽ chạy trước hàm “main”, tạo giá trị cho thuộc tính static.
2. Có thể áp dụng từ khóa static với thuộc tính (biến của class), phương thức, khối static và static nested class.

- Mục tiêu:

1. + Tiết kiệm vùng nhớ
2. + Không cần “new” ra đối tượng nhưng vẫn có thể sử dụng phương thức và thuộc tính static
3. + Giúp xây dựng những tiện ích (util).

Ràng buộc khi sử dụng static?

1. Phương thức static chỉ gọi được các phương thức và thuộc tính static.
2. Phương thức non-static có thể gọi được các phương thức và thuộc tính static.
3. Phương thức static không thể gọi được các phương thức và thuộc tính non-static.
4. Không thể sử dụng từ khóa this & super cho phương thức static.

# 59. Cách xử lí Exception trong Spring? - Có 2 cách, sử dụng try-catch hoặc throw/ throws.

# 60. Generic là gì?

- **Generic** – là cơ chế cho phép truyền dữ liệu vào như là tham số (tham số hóa kiểu dữ liệu), cho phép sử dụng method, class, interface với nhiều kiểu dữ liệu khác nhau.

**- Ưu điểm:**

+ Không cần phải ép kiểu.

+ Xây dựng đc bài toán tổng quát, tái sử dụng đc mã nguồn.

+ Bắt đc lỗi lúc **compile** (biên dịch).

-**Nhược điểm:**

- Không đc sử dụng cho kiểu nguyên thủy.

- Không sử dụng đc từ khóa **static** (vì chưa xác định đc kiểu dữ liệu nên chưa tạo đc bộ nhớ).

- Không thể ép kiểu để kiểm tra (ví dụ: không ép đc Person -> Student).

- Không khởi tạo đc **instance** (ví dụ: <E> e = new <E>();).

- Không thể **overloading** (Ví dụ: phương thức List<String> và List<Integer> thì nó chỉ hiểu là List mà thôi).

- Không thể khởi tạo mảng <E>.

# 61. Nối 2 chuỗi trong Java như thế nào, nêu code cụ thể?

1/Trong Java, để nối 2 chuỗi lại với nhau, chúng ta có thể sử dụng toán tử "+" hoặc phương thức concat() của lớp String.

2/VD: String str1 = "Hello"; String str2 = "World"; String result = str1 + " " + str2;

String str1 = "Hello"; String str2 = "World"; String result = str1.concat(" ").concat(str2);

# 62. Trong Java 8, interface có viết hàm có thân hàm được ko?

- Trong phiên bản Java 8, bạn có thể viết phương thức có thân hàm trong một interface bằng cách sử dụng từ khóa "default". Lớp con triển khai có thể hoặc ko cần overide lại.

# 63. Enum là gì? Mục đích dùng enum?

1/**Enum**: (viết tắt của từ "enumeration") trong Java là một kiểu dữ liệu đặc biệt được sử dụng để đại diện cho một tập hợp các hằng số có thể liệt kê được.

- Định nghĩa một enum trong Java giống như định nghĩa một lớp, với danh sách các hằng số enum được liệt kê trong ngoặc đơn và được phân tách bằng dấu phẩy.

- Mỗi hằng số enum là một phiên bản của enum đó và có thể được tham chiếu bằng cách sử dụng tên enum.

2/**Mục đích:** chính của việc sử dụng enum trong Java là để cung cấp một cách thức tiện lợi để định nghĩa một tập hợp các hằng số liên quan.

* Enum cung cấp tính tự nhiên và rõ ràng hơn trong việc biểu diễn các giá trị cố định, giúp cho mã nguồn dễ đọc hơn và giảm thiểu các lỗi cú pháp.
* Enum cũng cho phép bạn sử dụng các phương thức và thuộc tính cùng với các hằng số enum để thực hiện các hoạt động phù hợp với mục đích của enum đó.

# 64. Viết log ra 1 file trong Java như thế nào?

- Trong Java, để viết log ra một file, chúng ta có thể sử dụng Java Logging API (java.util.logging) có sẵn trong JDK hoặc sử dụng các thư viện logging phổ biến như Log4j, SLF4J hoặc Logback.

- VD: khai báo: private static final Logger logger = Logger.getLogger(MyLogger.class.getName());

# 65.(Trending) Phân biệt về tham chiếu và tham trị trong Java?

**1/Tham trị (pass by value)** khi gọi một phương thức và truyền một giá trị cho phương thức, được gọi là truyền tham trị.

* + 1. Việc thay đổi giá trị chỉ có hiệu lực trong phương thức được gọi, không có hiệu lực bên ngoài phương thức.
    2. Trong Java, truyền tham trị dành cho các tham số có kiểu dữ liệu nguyên thủy là byte, short, int, long, float, double, boolean, char.

**2/Tham chiếu (pass by reference)** khi gọi một phương thức và truyền một tham chiếu cho phương thức, được gọi là truyền tham chiếu.

- Việc thay đổi giá trị của biến tham chiếu bên trong phương thức có thể làm thay đổi giá trị của nó.

- Trong Java, tất các phương thức có tham số là biến có kiểu Object đều là kiểu tham chiếu.

# 66. Trình bày bằng code lưu 1 đối tượng vào file text ?

- Để lưu đối tượng vào file text trong Java, ta có thể sử dụng cơ chế Serialization.

- Cơ chế này cho phép ta chuyển đổi đối tượng sang dạng byte để có thể lưu xuống file, sau đó ta có thể đọc lại và chuyển đổi từ dạng byte sang đối tượng ban đầu. -VD: public class Person implements Serializable{};

# 67. Tạo unit test trong java?

Để tạo unit test trong Java, chúng ta có thể sử dụng framework testing như JUnit, TestNG, hoặc Mockito.

# 68.So sánh equal và hashcode?

Trong java, Object class mặc định là cha của tất cả các class, bên trong nó định nghĩa sẵn 2 method **equals**() và **hashCode**() – đều này có nghĩa rằng tất cả các class đều mặc định sẽ có 2 method này.

- **equal**( ) – dùng để kiểm tra 2 đối tượng bất kỳ có bằng nhau hay không?

* 1. + Toán tử ‘**= =**’ được dùng để so sánh địa chỉ 2 đối tượng và giá trị của các biến nguyên thủy.
  2. + Còn **equals**() được dùng để định nghĩa thế nào là 2 đối tượng trùng nhau, **equals**() chỉ áp dụng cho kiểu đối tượng, không áp dụng cho kiểu nguyên thủy.
  3. + Giả sử rằng 2 đối tượng có các thuộc tính đều bằng nhau, thế nhưng kết quả so sánh từ **equals**() lại trả về **FALSE**? => Vì mặc định, **equals**() được triển khai trong **Object class** sẽ so sánh địa chỉ object thay vì so sánh giá trị các thuộc tính trong **object**. Hai đối tượng này là duy nhất và được cấp phát vùng nhớ riêng trong **heap**.

=> Muốn **equals**() so sánh giá trị các thuộc tính trong **object** thay vì so sánh địa chỉ thì phải **override** lại phương thức **equals**().

- **hashCode**( ) – Mã băm, là giá trị định danh cho 1 đối tượng, **hashCode** trả về của đối tượng được tạo ra sẽ là mã băm của địa chỉ để gom nhóm các đối tượng, giúp tăng tốc độ tìm kiếm của chương trình lên từ 30-50 lần.

+ Nếu hai đối tượng bằng nhau thì chúng phải có cùng **hashCode**(). + Nếu hai đối tượng có cùng **hashCode**() thì chúng có thể bằng nhau hoặc là không.

# 69. Lambda trong java để làm gì?

Trong Java, Lambda Expression là một cú pháp đặc biệt để tạo ra một đối tượng của một functional interface (một interface chỉ có duy nhất một phương thức trừu tượng).

Lambda Expression cho phép bạn viết mã ngắn gọn hơn và dễ hiểu hơn, thường được sử dụng trong các trường hợp sau:

1. Sử dụng trong cấu trúc Stream API của Java để xử lý tập dữ liệu.
2. Sử dụng để triển khai callback trong các ứng dụng đa luồng.
3. Sử dụng để triển khai các phương thức hàm (functional methods) trong các functional interface.

# 70.(Trending) Cú pháp tạo multi thread (đa luồng)?

Để tạo multi-thread (đa luồng) trong Java, bạn có thể sử dụng một trong hai cách sau:

**1. Sử dụng lớp Thread:**

1. Bạn có thể tạo ra một lớp mới kế thừa từ lớp Thread, và ghi đè phương thức run() để triển khai nội dung của luồng.
2. Sau đó, bạn tạo một đối tượng của lớp và gọi phương thức start() để bắt đầu thực thi luồng.

**2. Sử dụng interface Runnable**

1. Bạn cũng có thể tạo class triển khai giao diện Runnable và tạo phương thức run().
2. Sau đó, bạn tạo một đối tượng của lớp Thread và truyền đối tượng của class vào trong phương thức tạo.
3. => Khi chạy chương trình, nội dung của phương thức run() sẽ được thực thi trên một luồng độc lập. Các luồng này chạy đồng thời và độc lập với nhau, giúp tăng tốc độ xử lý các tác vụ đồng thời.

# 71. Autoboxing và Unboxing trong java?

**Autoboxing và unboxing** là hai khái niệm quan trọng trong Java, cho phép chuyển đổi giữa các kiểu dữ liệu nguyên thủy (primitive types) và các kiểu dữ liệu đối tượng (object types) của Java tự động.

**1/Autoboxing** là quá trình tự động chuyển đổi từ kiểu dữ liệu nguyên thủy (primitive type) sang kiểu dữ liệu đối tượng tương ứng (wrapper class), ví dụ như chuyển đổi từ int sang Integer.

- Autoboxing xảy ra khi một giá trị nguyên thủy được gán cho một biến đối tượng hoặc được truyền vào một phương thức yêu cầu tham số là đối tượng.

**2/Unboxing** là quá trình tự động chuyển đổi từ kiểu dữ liệu đối tượng tương ứng (wrapper class) sang kiểu dữ liệu nguyên thủy (primitive type), ví dụ như chuyển đổi từ Integer sang int.

- Unboxing xảy ra khi một giá trị đối tượng được gán cho một biến nguyên thủy hoặc được truyền vào một phương thức yêu cầu tham số là nguyên thủy.

# 72. Trình bày CONCURRENCY trong Java?

**Concurrency** (đồng thời) là khả năng của chương trình để thực thi nhiều tác vụ đồng thời, tận dụng hiệu quả tài nguyên hệ thống và tăng tốc độ thực thi. Trong Java, Concurrency được hỗ trợ thông qua gói java.util.concurrent, cung cấp các lớp và giao diện để quản lý và điều khiển việc thực thi các tác vụ đồng thời. Các thành phần cơ bản trong Concurrency trong Java bao gồm:

**1.Luồng (Thread):** là một đơn vị nhỏ nhất của thực thi đồng thời, được sử dụng để thực hiện các tác vụ đồng thời. Một chương trình Java thường bao gồm ít nhất một luồng chính (main thread) để thực thi các tác vụ chính.

**2.Synchronized:** là một khối mã được đồng bộ hóa, chỉ cho phép một luồng thực thi vào một thời điểm. Điều này giúp tránh xung đột dữ liệu khi nhiều luồng truy cập vào cùng một biến hoặc phương thức.

**3.Lock và Condition:** là cơ chế khác để đồng bộ hóa các luồng, cung cấp khả năng kiểm soát độc quyền truy cập vào tài nguyên và chờ đợi tín hiệu để thực hiện các tác vụ.

**4.Executor:** là một giao diện cung cấp khả năng quản lý và điều khiển việc thực thi các tác vụ đồng thời thông qua một loạt các luồng, giúp tối ưu hiệu suất và tăng tốc độ thực thi.

**5.Semaphore:** là một cơ chế đồng bộ hóa được sử dụng để giới hạn số lượng luồng được phép truy cập vào một tài nguyên cùng một lúc.

**6.Future và Callable:** cung cấp khả năng thực thi một tác vụ đồng thời và trả về kết quả sau khi hoàn thành.

**7.Atomic:** là một tập hợp các lớp và giao diện cung cấp khả năng đồng bộ hóa dữ liệu nguyên thủy, giúp tránh xung đột dữ liệu khi nhiều luồng truy cập vào cùng một biến.

# 73. Khái niệm Immutable?

Trong Java, "**immutable**" (không thay đổi) được sử dụng để chỉ các đối tượng không thể thay đổi sau khi được tạo.

- Một đối tượng không thể thay đổi được hiểu là một đối tượng mà các thuộc tính của nó không thể được sửa đổi sau khi đối tượng đã được tạo.

- Thay vì thay đổi trực tiếp các thuộc tính của đối tượng, một đối tượng không thể thay đổi thường sẽ trả về một đối tượng mới có các giá trị thuộc tính mới khi được thay đổi.

# 74. Khi nào dùng stream, khi nào dùng List?

Stream và List đều là các cấu trúc dữ liệu để lưu trữ và xử lý tập hợp các phần tử.

* List được sử dụng khi ta muốn lưu trữ và truy cập các phần tử theo thứ tự cụ thể.
* Ta có thể truy cập các phần tử trong List bằng cách sử dụng index hoặc iterator.
* Stream được sử dụng khi ta muốn thực hiện các phép xử lý trên tập hợp các phần tử, chẳng hạn như lọc, sắp xếp, phân loại hoặc tính toán.
* Stream cung cấp một loạt các phương thức để thực hiện các phép xử lý này một cách dễ dàng và linh hoạt hơn so với việc sử dụng vòng lặp hoặc List

# 75. Loại bỏ các phần tử trùng lặp mà ko chạy vòng for.

Chuyển đổi mảng hoặc chuỗi cần loại bỏ về **Set** hoặc **Map.Vì Map và Set không chứa phần tử trùng lặp.**

# 76. So sánh 2 Object trong java thì dùng cái gì?

Trong Java, để so sánh hai Object, chúng ta có thể sử dụng hai phương thức **equals() và compareTo().**

* Phương thức equals() được sử dụng để so sánh hai Object về mặt giá trị, tức là so sánh từng thuộc tính của Object để xác định chúng có giống nhau hay không.
* Phương thức compareTo() được sử dụng để so sánh hai Object về mặt thứ tự, tức là so sánh để xác định đối tượng nào lớn hơn hay nhỏ hơn đối tượng kia.
* Phương thức compareTo() được định nghĩa trong giao diện Comparable và được sử dụng để sắp xếp các đối tượng trong một tập hợp.

**Lưu ý:**

1. Khi hai đối tượng được so sánh bằng equals(), nếu kết quả trả về là true, thì theo lý thuyết chúng ta nên có cùng một giá trị hashCode.
2. Tuy nhiên, nếu không override phương thức hashCode(), hai đối tượng có thể có giá trị hashCode khác nhau, dẫn đến việc chúng không thể được xem như là giống nhau khi được thêm vào một cấu trúc dữ liệu dựa trên bảng băm.

# 77. Hiểu exception là gì? Có mấy loại exception? Câu 51.

# 78. Làm thế nào để xóa nhiều mục trong 1 list danh sách?

* Sử dụng vòng lặp for để xóa các mục không mong muốn khỏi danh sách:
* Sử dụng phương thức removeAll() để xóa các phần tử có trong một danh sách khác,hoặc xoá tất cả các phần tử có trong danh sách

# 79. Tại sao sử dụng abstract lấy 1 ví dụ trong project của bạn?

Abstract class thường được sử dụng để định nghĩa một lớp có các phương thức chung cho các lớp con kế thừa và triển khai, giúp cho code dễ bảo trì và mở rộng hơn.

# 80. Kiểu int có bao nhiêu byte? Tại sao có kiểu dữ liệu int mà còn lại có thêm kiểu Integer?

1. Kiểu dữ liệu **int** là một kiểu dữ liệu nguyên thủy (primitive data type) và sử dụng 4 byte để lưu trữ.
2. Kiểu dữ liệu **int** được sử dụng để lưu trữ giá trị số nguyên không dấu trong khoảng từ -2,147,483,648 đến 2,147,483,647.
3. Kiểu dữ liệu **Integer** là một lớp đối tượng (**object class**) được đóng gói (**wrapper class**) cho kiểu dữ liệu **int**. Lớp **Integer** cung cấp một số phương thức và thuộc tính để làm việc với kiểu dữ liệu int như là một đối tượng.

* **intValue():** trả về giá trị int của đối tượng Integer
* **parseInt(String s):** chuyển đổi một chuỗi thành giá trị int
* **MAX\_VALUE**: giá trị lớn nhất có thể được đại diện bởi kiểu int
* **MIN\_VALUE**: giá trị nhỏ nhất có thể được đại diện bởi kiểu int

1. Việc sử dụng lớp **Integer** thường được áp dụng khi ta cần làm việc với kiểu dữ liệu **int** như là một đối tượng
2. Hoặc khi ta muốn sử dụng một số phương thức mà chỉ nhận đối tượng làm tham số đầu vào (chẳng hạn như trong trường hợp của Collections.sort().
3. Hoặc khi ta muốn dùng giá trị null mặc định của Interger vào một đích nào đó

# 81. Có một class bình thường (không dùng abstract, interface), làm cách nào để không cho new class đó? Sử dụng access modifier private cho constructor.

# 82. Integer với long cái nào lớn hơn? Long lớn hơn vì Interger: 4 byte -> 16 bit, Long: 8byte -> 64bit.

# 83. Dùng for(int i=0; i>=0; i++) in i thì nó sẽ ra cái gì? Đến lúc chạy hết phạm vi của kiểu int thì điều gì xảy ra?

In i sẽ ra dãy số từ i = 1 đến số giới hạn của kiểu int là i = 2^31 - 1 = 2147483647. Đến lúc chạy hết phạm vi của kiểu int thì sẽ in ra số 2147483647 và dừng vòng lặp.

# 84. Biến static khác biến thường?

1. **Scope** (Phạm vi): Biến static có phạm vi là local scope (phạm vi cục bộ) trong khi biến thường có thể có phạm vi là local hoặc global (phạm vi toàn cục). Biến static chỉ có thể truy cập trong phạm vi khai báo của chúng.
2. **Lifetime** (Tuổi thọ): Biến static tồn tại trong toàn bộ chương trình khi nó được khởi tạo, trong khi biến thường tồn tại trong phạm vi của nó, và bị hủy khi thoát khỏi phạm vi đó.
3. **Giá trị mặc định:** Biến static sẽ được khởi tạo với giá trị mặc định là 0 hoặc null (đối với kiểu dữ liệu tham chiếu), trong khi biến thường sẽ không có giá trị mặc định.
4. **Truy cập**: Biến static có thể được truy cập trực tiếp bằng tên biến và không cần phải tạo một đối tượng mới, trong khi biến thường thường phải được truy cập thông qua đối tượng chứa nó.
5. **Lưu trữ**: Biến static được lưu trữ trong bộ nhớ tĩnh và không thể thay đổi vị trí của nó trong bộ nhớ trong khi biến thường được lưu trữ trong bộ nhớ động và có thể thay đổi vị trí của nó trong bộ nhớ.

# 85. Cho String a = "5"; String b = "5"; if (a == b) return true or false? Giải thích?

True.Vì cả 2 đều được kháo bằng literor.So sanh == sẽ so sánh giá trị của 2 biến.

# 86. Phân biệt static và final?

* **Static**: được sử dụng để khai báo thuộc tính và phương thứ dùng chung cho lớp là một thành phần của lớp chứ không phải của đối tượng. Những thành phần được khai báo là static có thể được truy cập bằng cách sử dụng tên lớp và không cần tạo một đối tượng của lớp đó.
* **Final**: được sử dụng để khai báo một hằng số, một biến không thể thay đổi giá trị sau khi được khởi tạo hoặc một phương thức không thể được ghi đè bởi lớp con. Biến hoặc phương thức được khai báo là final sẽ được bảo vệ khỏi việc thay đổi giá trị hoặc định nghĩa lại trong lớp con.

# 87. Set, phân biệt Set - List?

**Set** là cấu trúc dữ liệu tuyến tính, trong đó mỗi phần tử chỉ xuất hiện duy nhất 1 lần.

**List** là cấu trúc dữ liệu tuyến tính, trong đó các phần tử được sắp xếp theo 1 thứ tự xác định, cho phép các phần tử được trùng lặp nhau.

# 88. Phân biệt Stack, Queue?

**Stack** (ngăn xếp) là cấu trúc dữ liệu dạng danh sách, thêm và xóa phần tử theo cơ chế FILO. Stack là một lớp con của lớp Vector, (Vector) một lớp triển khai của interface List.

**Queue** (hàng đợi) là cấu trúc dữ liệu dạng danh sách, thêm và xóa phần tử theo cơ chế FIFO. Queue là 1 trong các interface collection.

# 89. Viết method java lấy ra danh sách person có tuổi lớn hơn 15 từ 1 list person cho sẵn

public List<Person> getPersonListAboveAge(List<Person> personList, int age) {

List<Person> result = new ArrayList<>();

for (Person person : personList) {

if (person.getAge() > age) { result.add(person); }}

return result; }

# 90. Vùng nhớ đặc biệt của String?

Trong Java, có hai loại vùng nhớ đặc biệt liên quan đến chuỗi String:

**1.String Pool:** Là một vùng nhớ trong heap, nơi lưu trữ các chuỗi String literal.

Khi bạn tạo một chuỗi String literal, JVM sẽ kiểm tra trong String Pool xem có chuỗi nào giống chuỗi đó không.

* Nếu có, thì tham chiếu của chuỗi mới sẽ được trỏ đến chuỗi cũ đã có trong String Pool.
* Nếu không, thì chuỗi mới sẽ được thêm vào String Pool. Điều này giúp tiết kiệm bộ nhớ và tăng tốc độ thực thi chương trình.

**Ví dụ:** String str1 = "hello";

String str2 = "hello"; // tham chiếu đến chuỗi "hello" trong String Pool

String str3 = new String("hello");

String str4 = new String("hello"); // tạo mới hai đối tượng String khác nhau trong memory heap

Trong ví dụ trên, str1 và str2 sẽ cùng trỏ đến một đối tượng String trong String Pool.

Trong khi đó, str3 và str4 là hai đối tượng String khác nhau được tạo trong memory heap.

**2.StringBuilder và StringBuffer:** Là hai lớp được sử dụng để xây dựng và thay đổi các chuỗi String.

* Chúng cung cấp các phương thức để thêm, xóa, sửa đổi, chèn, đảo ngược, ... các ký tự của chuỗi một cách hiệu quả.
* Các đối tượng StringBuilder và StringBuffer được tạo trong memory heap và có thể thay đổi trực tiếp các giá trị của chuỗi mà chúng đang tham chiếu đến.

# 91. Set là gì, Map là gì, so sánh Set vs Map?

**1.Set** là một tập hợp các giá trị duy nhất, tức là mỗi giá trị chỉ xuất hiện một lần trong Set.

* Set không có thứ tự, nghĩa là các phần tử của Set không được sắp xếp theo bất kỳ cách nào.
* Set được sử dụng để loại bỏ các giá trị trùng lặp trong một danh sách hoặc một tập hợp các giá trị.

**2.Map** là một cấu trúc dữ liệu có thể lưu trữ các giá trị theo cặp key-value (khóa - giá trị).

* Key là một giá trị duy nhất, tương ứng với một giá trị value.
* Map có thể được sử dụng để lưu trữ các thông tin với key tương ứng, giúp truy cập dữ liệu nhanh chóng và dễ dàng.

**3.So sánh Set và Map:**

- Cả Set và Map đều được sử dụng để lưu trữ các giá trị duy nhất.

- Set không có thứ tự, trong khi Map có thể được sắp xếp theo key.

- Set không có key, trong khi Map lưu trữ giá trị theo cặp key-value.

- Set được sử dụng để loại bỏ các giá trị trùng lặp trong một danh sách hoặc một tập hợp các giá trị, trong khi Map được sử dụng để lưu trữ và truy xuất các giá trị dựa trên key.

- Trong một số ngôn ngữ lập trình như Java, Set và Map đều triển khai bằng cách sử dụng một tập hợp hash, tuy nhiên Map sử dụng cặp key-value để lưu trữ giá trị trong khi Set chỉ lưu trữ key.

# 92. Thread trong Java

Thread trong java là luồng, là một đơn vị trong java.

* Mỗi thread thực hiện 1 công việc hoàn toàn riêng biệt và được quản lý bởi máy ảo java(JVM) Java là một ngôn ngữ đã luồng.
* Thread kế thừa ý tưởng đa nhiệm trong các ứng dụng để có thể chia nhỏ các hoạt động riêng biệt trong 1 ứng dụng đơn thành các luồng riêng lẻ.
* Mỗi thread có thể chạy song song riêng biệt với nhau

Vòng đời của thread trong java gồm:

- New: Đây là trạng thái đầu tiên của 1 thread. Thread được khởi tạo nhưng chưa được gọi phương thức strart

- Runnable: Khi 1 thread sinh ra và bắt đầu chạy nó sẽ trở thành runnable. Và trong trạng thái này nó được xem như đang thực hiện tác vụ của chính minh

- Waiting: Khi 1 thread phải chờ 1 thread khác thực hiện một tác vụ nào đó

- Timed Waiting: Ở trạng thái này thread sẽ chờ trong 1 khoảng thời gian nào đó

- Terminated: Thread ở trạng thái này khi nó hoàn thành xong tác vụ của nó.

# 93. Tìm kiếm phần tử trong mảng sử dụng quick sort?

Dùng quick sort để sắp xếp lại mảng theo thứ tự tăng dần (hoặc giảm dần) và dùng for để duyệt mảng tìm giá trị cần tìm.

# 94. So sánh implements với extends?

**Implement**

* Sử dụng từ khoá này 1 class có thể implement interface
* Bắt buộc phải implement toàn bộ method của interface trừ method default và static
* 1 Class có thể implement từ nhiều interface cùng lúc
* 1 Interface không thể implement 1 interface khác

**Extend:**

* Sử dụng từ khoá này 1 class có thể kế thừa 1 class và 1 interface có thể kế thừa 1 interface khác
* Không bắt buộc supclass phải override lại toàn bộ method của superclass
* Chỉ 1 lớp cha được extend từ 1 class(không hỗ trợ đa kế thừa)
* Nhiều Interface có thể được extend từ 1 interface.

# 95. Khi nào dùng class khi nào dùng class abtract?

* + - Class định nghĩa một lớp cụ thể, chứa tất cả các phương thức và thuộc tính chung cho một nhóm đối tượng.
    - Khởi tạo các đối tượng của lớp đó.
    - Abstract class định nghĩa một lớp cha chung cho một số lớp con, không được khởi tạo trực tiếp, nhưng được sử dụng để khai báo đối tượng của lớp con.
    - Định nghĩa các phương thức trừu tượng mà các lớp con phải triển khai.

# 96. List và ArrayList khác nhau ntn?

**List**

-Interface con của collection interface

-Chứa nhóm các đối tượng

-Lưu thứ tự thêm vào

-Các phần tử được trùng nhau

**ArrayList**

-Là 1 class thực thi interface list

-Dữ liệu được lưu theo dạng mảng

-Có các phương thức thao tác với mảng được khai báo trong List và các interface cao hơn

**Câu hỏi audit C0622G1**

**1.Bean là gì?** Bean – là những module chính của chương trình, là 1 đối tượng đc quản lý bởi Spring IoC container (do framework tự khởi tạo, tự quản lý vòng đời của đối tượng).

**Các cách khai báo:**

**+ Dùng annotation: @Controller, @Service, @Repository, @Component.**

**+ Dùng kết hợp @Configuration + @Bean.**

**+ Khai báo Bean bằng XML**

**- Có 5 Bean Scope:**

**+ Singleton – chỉ duy nhất một thể hiện của bean sẽ được tạo cho mỗi container. Đây là scope mặc định cho spring bean.**

**+ Prototype – khác với Singleton, 1 thể hiện của bean sẽ được tạo cho mỗi lần được yêu cầu (request).**

**+ Request – giống với prototype scope, tuy nhiên nó dùng cho ứng dụng web, 1 thể hiện của bean sẽ được tạo cho mỗi HTTP request.**

**+ Session – mỗi thể hiện của bean sẽ được tạo cho mỗi HTTP Session.**

**+ Global-Session – được sử dụng để tạo global sesion bean cho các ứng dụng Portlet.**

**2.Tính trừu tượng là gì? Cho ví dụ?**

Tính trừu tượng (Abstraction):Tính trừu tượng giúp loại bỏ những thứ phức tạp, không cần thiết của đối tượng và chỉ tập trung vào những gì cốt lõi, quan trọng.Ví dụ: Quản lý nhân viên thì chỉ cần quan tâm đến những thông tin như:Họ tên, Ngày sinh,Giới tính, Trình độ…chứ không cần quan tâm: Chiều cao,Cân nặng, Sở thích,Màu da…

**3.Phân biệt Having và Where?**

**- Where –** Mệnh đề điều kiện, dùng để truy xuất dữ liệu theo 1 điều kiện nào đó (một số toán tử theo sau where: <, >, < >, =, or, and, not, between, like, in,…)

- **Having** – Mệnh đề điều kiện, thường sử dụng sau **Group by** và thường sử dụng để làm điều kiện cho các hàm thống kê count(), sum(), avg(), max(), min()…, sử dụng để lọc các bản ghi và chỉ lấy những bản ghi phù hợp với yêu cầu hoặc thực sự cần thiết tương tự như mệnh đề **WHERE**. Tuy nhiên:

+ **WHERE** là câu lệnh điều kiện trả kết quả đối chiếu với từng dòng.

+ **HAVING** là câu lệnh điều kiện trả kết quả đối chiếu cho nhóm được tạo bởi mệnh đề **GROUP BY**.

=> Vì vậy mà sau **GROUP BY** thì sẽ chỉ dùng được **HAVING** còn **WHERE** không dùng được sau **GROUP BY**.

**4.Position là gì? các loại Position ?**

**- Position –** là thuộc tính CSS để xác định cách thức hiển thị vị trí của các thẻ HTML. bao gồm:

**+ Static –** vị trí mặc định của thẻ HTML (không chịu ảnh hưởng bởi các thuộc tính top, left, right, bottom).

**+ Relative** – là vị trí tương đối, lấy chính nó làm gốc tọa độ (chịu ảnh hưởng bởi các thuộc tính top,left, right, bottom).

**+ Absolute (tuyệt đối)** – lấy định vị theo thẻ cha (với điều kiện thẻ cha không đc dùng static).

**+ Fixed** – là vị trí cố định với màn hình (không chịu ảnh hưởng bởi các scrollbar – thanh cuộn).

**+ Sticky** – khi cuộn scrollbar đến 1 vị trí nào đó (do LTV quy định) thì phần tử HTML sẽ dán cố định tại vị trí trên trang web.

**5.Java là gì?**

**- Java** là ngôn ngữ OOP – lập trình hướng đối tượng, phát hành năm 1995 bởi Sun Microsystems.

*- Java có 1 câu slogan nổi tiếng là:* **“Write one, run anywhere”** - ***viết một lần chạy nhiều nơi***, nghĩa là chỉ cần viết một lần trên window chẳng hạn, sau đó vẫn chương trình đó có thể chạy trên Linux, Android, các thiết bị J2ME...

*- Sở dĩ có câu này là vì:* Java code được biên dịch thành 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.

**6.Kiến trúc của Angular?**

- Một dự án Angular được chia làm nhiều module dưới sự quản lý của root module(app.module.ts).

- Component là thành phần quan trọng nhất của ứng dụng Angular. Mỗi chức năng sẽ là 1 component.

- Directive: thành phần mở rộng của HTML(\*ngFor, \*ngIf).

- Service: muốn sử dụng phải nhờ vào đối tượng Injector để DI vào class

**7.Java là ngôn ngữ thông dịch hay biên dịch? giải thích?**

- Biên dịch: Là quá trình dịch toàn bộ source file sang file .class

- Thông dịch: Là quá trình dịch từng câu lệnh sang mã máy.

- Java vừa là ngôn ngữ thông dịch, vừa là ngôn ngữ biên dịch

- Java sẽ tiến hành biên dịch thông qua compiler nằm trong JDK để dịch toàn bộ file .java sang file. class

- Tiếp tục nhờ JVM dịch từng dòng trong file .class ra mã máy, nhờ đó máy hiểu và thực thi.

**8.Em biết gì về Heap và stack?**

**-------------HEAP-------------**

- Heap memory là bộ nhớ được sử dụng bởi Java Runtime để cấp phát bộ nhớ cho các đối tượng (object) và String.

- Bất kỳ khi nào có một đối tượng được tạo, thì miền giá trị của nó sẽ được tạo lưu ở bộ nhớ Heap.

- Bộ dọn rác (Garbage Collection) chạy trên heap memory để giải phóng bộ nhớ nếu không có đối tượng nào tham chiếu tới.

**--------------STACK--------------**

- Stack memory bao gồm các giá trị cụ thể của method: các biến local và các tham chiếu tới các đối tượng chứa ở trong heap memory được tham chiếu bởi method.

- Stack memory được tham chiếu theo thứ tự LIFO (Last In First Out – vào cuối cùng thì ra đầu tiên). Tức là lưu trữ kiểu ngăn xếp (stack). Khi có một method được thực thi, một block được tạo ra trong stack memory để chứa các biến nguyên thủy local và các tham chiếu tới các object. Khi method kết thúc, block đó sẽ không còn được sử dụng và được phục vụ cho method tiếp theo.

- Stack memory có kích thước rất nhỏ so với Heap memory.

**9.Lúc nào thì sử dụng Interface? Lúc nào thì sử dụng Abstract Class?** Khi một nhóm đối tượng có cùng bản chất kế thừa từ một class thì sử dụng abstract class. Khi một nhóm đối tượng không có cùng bản chất nhưng chúng có hành động giống nhau thì sử dụng interface.

**10.Em biết những collection nào trong java?**

Set Interface->Sortset interface(TreeSet : class),HashSet (class), LinkedHashSet(class).

* **Queue interface ->Deque interface** (Array Deque: class, Linked List: class),Priority Queue (class)
* **List interface** -> ArrayList (class),Linked List (class),List (class) .
* **Map interface-> SortMap interface (**TreeMap:Class**),**HashMap ,HashTable ,Linked HashMap(class)

**12.Phân biệt abstract/interface?**

- **Abstract class:** 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 + Không thể tạo đc đối tượng + 1 **class** có ít nhất 1 phương thức **abstract** thì chắc chắn lớp đó phải là **abstract** (ngược lại thì không) + 1 **class abstract** thì có thể chứa phương thức **non-abstract** + **Class abstract** không thể đi kèm với từ khóa **final** + Class abstract con **extends** Class abstract cha thì không cần ghi đè các phương thức **abstract** của cha + Từ khóa **private** không dùng cho phương thức **abstract**.

- **Interface:**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 tính trừu tượng hoàn toàn và đa kế thừa trong Java. Interface trong Java cũng biểu diễn mối quan hệ IS-A.+ Không thể tạo đc đối tượng + Các trường ở trong **interface** đều là hằng số + Tất cả các phương thức đều là **abstract** + Hỗ trợ đa kế thừa (1 **interface** có thể **extends** từ nhiều **interface**) + Không tạo đc **constructor**.

**[?]** So sánh **Abstract Class** và **Interface**:

+ Có tạo đc đối tượng không? => Cả 2 đều không tạo đc đối tượng.

+ Trường/ thuộc tính? => Ở **Interface** đều là hằng số, ở **Abstract Class** thì như Class thông thường.

+ Phương thức (abstract hay non-abstract)?=> Ở **Interface** đều là abstract, ở **Abstract Class** có thể là abstract hoặc non-abstract.//trong Java 8, có thể tạo đc phương thức non-abstract trong Interface thông qua default/ static.

+ Có constructor không? => **Interface** không có, **Abstract Class** có.

+ Đa kế thừa? => **Interface** hỗ trợ đa kế thừa, **Abstract Class** không hỗ trợ đa kế thừa.

**13.Phân biệt String & StringBuilder?** Khi làm việc với dữ liệu kiểu text,Java cung cấp 3 class String, StringBuffer và StringBuilder + String là không thể thay đổi (immutable), và không cho phép có class con + StringBuffer, StringBuilder có thể thay đổi (mutable) + StringBuilder và StringBuffer là giống nhau, nó chỉ khác biệt tình huống sử dụng có liên quan tới đa luồng (Multi Thread). => về tốc độ xử lý StringBuilder là tốt nhất, sau đó StringBuffer và cuối cùng mới là String + Lớp String ghi đè phương thức equals() của lớp Object. Vì thế bạn có thể so sánh nội dung của 2 chuỗi bằng phương thức equals(), Lớp StringBuffer và StringBuilder không ghi đè phương thức equals() của lớp Object.

**14. Cách Spring MVC xử lý request?**

**+ Front controller –** xử lý, lắng nghe toàn bộ request từ bên ngoài ứng dụng, thực hiện điều hướng cho controller xử lý, trả về data, hiển thị lên cho người dùng.

**+ DispatcherServlet (trình điều phối) –** đây chính là front controller.

Dispatcher là thành phần quan trọng nhất trong Spring MVC. Nó sẽ là nơi đầu tiên nhận request từ client sau đó sẽ chuyển request đó tới các controller tương ứng, đồng thời sẽ là chốt chặn cuối cùng trả về kết quả cho client.

**+ Controller –** là file class, xử lý logic nghiệp vụ.

**+ ViewResolver (trình phân giải view)** – là cơ chế ánh xạ “tên” của view sang đối tượng view tương ứng.

**+ Luồng thực thi:**

=> Người dùng điền tên website cần truy cập trên browser => nhấn Enter gửi request lên server (nơi triển khai ứng dụng Spring) =>DispatcherServlet tiếp nhận request => chuyển tới controller tương ứng (thông qua các cơ chế mapping – ánh xạ đường dẫn) =>controller nhận yêu cầu và gọi phương thức thích hợp (GET / POST) =>controller tiến hành render dữ liệu và trả về tên view cho DispatcherServlet. => DispatcherServlet sẽ nhận trợ giúp từ ViewResolver để xác định view cho các yêu cầu. => DispatcherServlet chuyển dữ liệu tới view đó và chuyển tới trình duyệt.

**16. Các tầng validate dữ liệu?**

**UI layer:** - Phía client - Dùng ngôn ngữ front-end (JS, HTML,…) - Validate trực tiếp, tốc độ phản hồi nhanh, mượt, cân bằng tải (giảm tải cho server) - Bảo mật không cao, nhiều dữ liệu không xử lý đc (ví dụ như xử lý trùng lặp, xử lý ngày, giờ,…).

**Business layer:** - Phía server - Dùng ngôn ngữ server(Java, C#, PHP,…) - Ràng buộc chặt chẽ, độ chính xác cao, xử lý đa dạng - Nhiều xử lý khi có nhiều truy cập -> server quá tải.

**Data layer**- Phía **DB server** - Dùng **SQL** để tạo ràng buộc (not null, check, primary,…) - Cú pháp đơn giản, ngắn gọn, dễ sử dụng -> Khó bắt lỗi nhập liệu, tốc độ rất chậm (còn phải xét đến sự liên kết giữa các **table** – **CSDL** quan hệ).

Một ứng dụng **web** thông thường, chỉ cần **validate** kết hợp giữa **UI layer** và **Business layer** (**Spring** hỗ trợ **validation** ở tầng **business**).

**17.So sánh các loại form trong Angular ?**

* **Template driven form:** + Import FormsModule. + Bất đồng bộ(Async) + Tạo form ở HTML và gửi dữ liệu về file .ts + Khó validate & custom validate + Dữ liệu thuộc tính bị thay đổi.
* **Model driven form (Relative form):** + Import ReactiveFormsModule + Đồng bộ(Sync) + Tạo form ở file ts binding dữ liệu sang HTML + Hỗ trợ custom validation tốt hơn + Dữ liệu thuộc tính không thay đổi.

**18.Overload, Override là gì? So sánh?**

**Overriding (ghi đè):** Là cơ chế cho phép lớp con định nghĩa lại nội dung các phương thức trước đó ở lớp cha. + Phương thức phải cùng tên, cùng danh sách tham số, cùng kiểu dữ liệu trả về + Access Modifier của lớp con phải có level lớn hơn hoặc bằng lớp cha + Xảy ra trong class có mối quan hệ is-a (quan hệ kế thừa). Đa hình tại runtime. Lưu ý: không ghi đè được phương thức static Vì phương thức static được ràng buộc với class còn phương thức instance được ràng buộc với đối tượng. Static thuộc về vùng nhớ class còn instance thuộc về vùng nhớ heap.

**Overloading (nạp chồng):** Là cơ chế cho phép 1 lớp có khả năng định nghĩa ra nhiều phương thức cùng tên nhưng khác nhau về tham số truyền vào (kiểu dữ liệu, số lượng). Xảy ra trong cùng 1 class. Đa hình tại compile.

* *Câu hỏi 1*: Tại sao không thể nạp chồng phương thức bằng cách chỉ thay đổi kiểu trả về của phương thức? Trong java, không thể nạp chồng phương thức bằng cách chỉ thay đổi kiểu trả về của phương thức bởi vì không biết phương thức nào sẽ được gọi.
* *Câu hỏi 2*: Có thể nạp chồng phương thức main() không? Có, bạn có thể nạp chồng n phương thức main. Nhưng JVM chỉ gọi phương thức main() có tham số truyền vào là một mảng String.

**19.Generic là gì ? - Generic –** là cơ chế cho phép truyền dữ liệu vào như là tham số (tham số hóa kiểu dữ liệu), cho phép sử dụng method, class, interface với nhiều kiểu dữ liệu khác nhau.

**- Ưu:**+Dùng để viết những thuật toán tổng quát: rút gọn mã nguồn + Kiểu đối tượng an toàn: Chỉ lưu một kiểu đối tượng duy nhất. Không cho phép lưu trữ 2 đối tượng có kiểu khác nhau + Hạn chế ép kiểu + Kiểm tra lúc biên dịch: giúp hạn chế bị lỗi lúc Runtime.

**- Nhược:**+ Không đc sử dụng cho kiểu nguyên thủy + Không sử dụng đc từ khóa static (vì chưa xác định được kiểu dữ liệu nên chưa tạo đc bộ nhớ) +Không thể ép kiểu để kiểm tra (ví dụ: không ép đc Person -> Student) + Không khởi tạo đc instance (ví dụ: <E> e = new <E>();) + Không thể overloading (Ví dụ: phương thức List<String> và List<Integer> thì nó chỉ hiểu là List mà thôi) + Không thể khởi tạo mảng <E>

**20.So sánh List, Set, Map ?**

* List có thể chứa các phần tử trùng lặp (duplicate), trong khi Set,Map chỉ chứa các phần tử duy nhất.
* List, Set chỉ chứa giá trị, trong khi Map chứa cặp key và value.

**21.Các thành phần của 1 trang html, có thể chia thành bao nhiêu nhóm thẻ HTML ?**

**HTML** là ngôn ngữ được sử dụng để tạo ra các Webpage . HTML là viết tắt của chữ Hypertext Markup Language (Ngôn ngữ Đánh dấu Siêu Văn bản)

**<!DOCTYPE html> ; <html><head></head><body></body></html>**

**1 số thẻ cơ bản: <p>**: chiếm hết 1 dòng ; **<span>** : chiếm đúng chiều rộng nội dung ; **<h1> đến <h6>:** thẻ tiêu đề từ nhỏ đến lớn ; **<br>**: xuống dòng ;

**<hr>**: tạo đường ngang khi xuống dòng ; **<img>:** hiển thị hình ảnh; **<ul><li>:** tạo danh sách k có thứ tự ; **<ol><li>:** tạo danh sách k có thứ tự;

**22.Tại sao phải dùng Jquery ? một số ví dụ ?**

**- Jquery –** là thư viện đc xây dựng từ JS, với câu slogan “Write less, do more”.Với các tính năng nổi bật:+ Thao tác với DOM dễ dàng, nhanh chóng (cách truy cập DOM gần giống với cú pháp CSS).+ Xử lý event tốt + Xử lý animation dùng để tạo hiệu ứng chuyển động + Hỗ trợ Ajax + Gọn nhẹ, tích hợp file JS + Hỗ trợ hầu hết các loại trình duyệt.

**23.LocalStorage và cookie là gì? Sự khác nhau của 2 cái này.**

**Local Storage** là một tính năng mới được trang bị cho HTML5, cho phép bạn lưu trữ dữ liệu cục bộ trên máy client thông qua JavaScript. Dữ liệu lưu trong Local Storage được lưu theo cặp key-value và không có thời hạn, nghĩa là mọi thứ vẫn sẽ được giữ lại kể cả khi cửa sổ trình duyệt đã đóng.

**Khác nhau:**

* Về kích thước, Cookie chỉ cho phép lưu trữ đến 4KB dữ liệu, nghĩa là chỉ những thông tin cơ bản như username mới phù hợp. Trong khi đó Local Storage cho phép lưu đến 5MB, nhiều hơn hàng ngàn lần và do đó cũng có nhiều không gian hơn để caching dữ liệu.
* Về phạm vi, Local Storage chỉ lưu dữ liệu cục bộ mà không chuyển về các máy chủ như cookie.
* Về thời hạn, Local Storage không cung cấp mặc định tính năng tự động hủy dữ liệu hết hạn, trong khi bạn có thể đặt thời hạn cho cookie.

**Cookies** là **các tệp** được **trang web người dùng truy cập** tạo ra. Cookie giúp trải nghiệm trực tuyến của bạn dễ dàng hơn bằng cách **lưu thông tin duyệt web**. Với Cookies, các trang web có thể **duy trì trạng thái đăng nhập của bạn**, **ghi nhớ tùy chọn** trang web và cung cấp nội dung phù hợp với vị trí của người dùng.

Như vậy, Cookies thường được sử dụng để **lưu lại thông tin của người dùng** trên trình duyệt web.

**24.Sự khác biệt giữa Java 7 và Java 8**

* Tên mã cho Java 7 là Dolphin trong khi tên mã cho Java 8 là Spider.
* Java 7 là bản cập nhật đầu tiên và lớn cho ngôn ngữ lập trình thuộc quyền sở hữu và quản lý của Tập đoàn Oracle.
* Java 7 đã thêm hỗ trợ ngôn ngữ cho String in switch để người dùng có thể viết lại mã hiệu quả và dễ dàng hơn.
* Java 7 được hỗ trợ trên Win XP. Tuy nhiên, Java 8 không được hỗ trợ chính thức trên Win XP.
* Java 7 được phát hành vào ngày 27 tháng 2010 năm 8 trong khi Java 8 được phát hành vào ngày 2014 tháng XNUMX năm XNUMX.
* Java 8 cũng nhận được một API Ngày / Giờ mới và cải tiến, công cụ JavaScript nâng cao, API phát trực tuyến mới. Bộ tích lũy đồng thời, tạo ngẫu nhiên an toàn và hơn thế nữa.

**25.Phân biệt session và cookie**