Câu hỏi ôn tập Java

1. Hãy so sánh function và method ? Hãy so sánh việc gọi hàm trong lập trình thủ tục và truyền thông điệp trong lập trình hướng đối tượng.

2. Khi đi từ dưới lên, 1 cây phân cấp kế thừa, mức độ tổng quát hóa và mức độ trừu tượng hóa tăng hay giảm ? Tại sao?

3. Trong Java giao diện 1 lớp bao gồm những gì ? Lấy ví dụ

4. Có ý kiến cho rằng Java đưa ra khái niệm giao diện để giải quyết vấn đề đa kế thừa ? Vậy lớp trừu tượng trong Java có bị thừa? Giải thích vấn đề trên.

5. Nêu 4 nguyên lý cơ bản của lập trình hướng đối tượng ? Phân tích vai trò của từng nguyên lý trong quá trình đi từ bài toán tới khi hoàn thành mã nguồn chương trình?

HD: Với mỗi nguyên lý cần nêu nó được áp dụng ở giai đoạn nào, khi nào cần và kết quả là gì ?

6. Nêu khái niệm về thành phần static của 1 lớp ? Giải thích tại sao phương thức static chỉ có thể truy cập vào các thuộc tính static và chỉ có thể gọi các phương thức static trong cùng lớp?

7. Mối quan hệ giữa UML và quá trình phát triển phần mềm hướng đối tượng là như thế nào?

8. Những điểm nổi bật của mô hình xử lý ngoại lệ trong lập trình hướng đối tượng ? So sánh với mô hình xử lý lỗi truyền thống của lập trình hướng cấu trúc ?

9. Tuyên bố “Một phương thức khởi tạo không dùng các từ khóa abstract, static, final” là đúng hay sai ? Giải thích lý do cụ thể tại sao ?

10. Giao diện trong lập trình hướng đối tượng là gì ? Giao diện trong Java khác gì so với các giao diện trong lập trình hướng đối tượng nói chung ? Giao diện có dùng thay được lớp trừu tượng hay không ? Doushite ?

11. So sánh đối tượng với lớp ?

12. Hướng đối tượng với hướng cấu trúc có gì khác nhau ?

15. Trừu tượng hóa là gì ? Vai trò của trừu tượng hóa ?

16. Đóng gói trong lập trình hướng đối tượng là gì ? Đóng gói có phải là che dấu dữ liệu ? Cơ chế che dấu dữ liệu là gì ?

17. Ưu điểm của việc sử dụng GETTER và SETTER thay vì sử dụng thuộc tính public thông thường là gì ? Có thể tạo ra nhiều GETTERS và SETTERS cho các các thuộc tính private tuỳ thuộc vào trường hợp sử dụng không?

18. Biến this trong lớp Java có chức năng gì ?

19. Chồng phương thức là gì? Định nghĩa lại là gì ? So sánh overloading với overriding trong Java ?

20. So sánh instance member và class member ?

21. Lợi ích của package trong java ?

22. So sánh kết tập với kế thừa trong Java ?

23. Đa hình trong lập trình Java là gì ?

24. So sánh abstract class với interface: <https://www.javatpoint.com/difference-between-abstract-class-and-interface>

Link đọc thêm về Java: <https://quizlet.com/333170931/oop-flash-cards/>

Trả lời

**Câu 1.**

So sánh function và method :

|  |  |
| --- | --- |
| Function | Method |
| - Đều là cách giúp chia nhỏ chương trình lớn để dễ dàng xử lý, bảo trì.  - Đều là cách giúp tái sử dụng mã nguồn. | |
| - Function có thể đặt bất kì vị trí nào trong chương trình.  - Các function không được trùng tên. | - Method bắt buộc phải nằm trong 1 class cụ thể.  - Các method có trùng tên thông qua overloading và overriding. |

So sánh call function và send message:

|  |  |
| --- | --- |
| Call function | Send message |
| - Chỉ ra chính xác đoạn mã nguồn nào sẽ được thực hiện thông qua tên hàm.  - Chỉ có duy nhất 1 đoạn mã sẽ được thực thi với 1 tên hàm nào đó. => không thể có 2 hàm trùng tên.  - Thời điểm xác định đoạn mã nào được thực thi là thời điểm biên dịch. | - Yêu cầu 1 dịch vụ từ 1 đối tượng và đối tượng đó sẽ quyết định những việc phải làm.  - Các đối tượng khác nhau sẽ thực thi thông điệp theo những cách khác nhau.  - Đối với overloading thì thời điểm xác định đoạn mã thực thi là thời điểm biên dịch, còn overriding thì là thời điểm chạy. |

**Câu 2.**

Khi đi từ dưới lên của 1 cây phân cấp kế thừa thì mức độ tổng quát hóa và mức độ trừu tượng hóa tăng dần.

Các lớp con kế thừa các thành phần của lớp cha nhưng các lớp con cũng có những đặc điểm riêng biệt hơn, cụ thể hơn, phân biệt nó với các lớp con khác cùng cấp.

**Câu 3.**

Trong Java, giao diện (interface) của 1 lớp bao gồm:

+ Chữ kí các phương thức

+ Các thuộc tính khai báo hằng   
Ví dụ về 1 interface:

public interface iAnimal () {

public void say();

public void move ();

public void sleep ();

}

**Câu 4.**

Lớp trừu tượng trong Java không thừa.

Bản chất của lớp trừu tượng (abstract class) khác hoàn toàn với giao diện (interface).

+ Những lớp kế thừa 1 abstract class có cùng bản chất với abstract class. Quan hệ của của lớp với lớp kế thừa là “is – a”. Trong khi các lớp thực thi interface có thể khác bản chất nhau. Quan hệ với interface là “can – do”.

+ Abstract được sử dụng như 1 bản thiết kế tổng thể của các đối tượng trong khi interface dùng để mô tả các chức năng của đối tượng.

+ Abstract class để tái sử dụng mã nguồn trong khi interface thì không thể.

**Câu 5.**

4 nguyên lý cơ bản của hướng đối tượng là:

**- Trừu tượng hóa:** Quá trình loại bỏ các thuộc tính cụ thể, chỉ tổng hợp các thuộc tính chung của các đối tượng, tập trung vào các đối tượng phân biệt nó với các đối tượng khác.

**- Phân cấp:** Phân chia lớp theo mức độ trừu tượng hóa. Các lớp con thì có mức độ trừu tượng hóa thấp hơn các lớp cha.

*Khi nào cần:* Khi phân tích chương trình, cần tập trung vào các khái niệm/ vấn đề nào đó của bài toán tại 1 thời điểm.

*Kết quả:* Những đặc điểm quan trọng của chương trình cần giải quyết.

**- Đóng gói:** Che dấu dữ liệu bên trong của 1 đối tượng, cung cấp 1 tập các dịch vụ nhất định cho thế giới bên ngoài tương tác với đối tượng. Việc thay đổi bên trong không làm ảnh hưởng đến hoạt động bên ngoài của chương trình.

- **Module hóa:** Chia nhỏ 1 hệ thống lớp thành các thành phần nhỏ có thể quản lý dễ dàng hơn.

*Khi nào cần:* Đóng gói và module hóa áp dụng trong quá trình viết mã chương trình. Các lập trình viên có thể viết mã theo từng module nhỏ để dễ dàng quản lý, đẩy nhanh tiến độ công việc. Việc áp dụng nguyên lý đóng gói giúp tránh mã nguồn có thể dễ dàng sửa đổi mà không làm ảnh hưởng đến chương trình chính.

*Kết quả:* Quản lý tiến độ phát triển chương trình theo các module, mã nguồn dễ dàng sửa đổi.

**Câu 6:**

Các thành phần static của 1 lớp:

1. Thuộc tính tĩnh:

- Là các thuộc tính mang thông tin chung của 1 lớp.

- Thay đổi giá trị của thuộc tính tĩnh trong 1 đối tượng của lớp sẽ làm thay đổi giá trị đó trong toàn bộ các đối tượng khác của lớp đó.

1. Phương thức tĩnh:

- Là các phương thức tiện ích được khai báo để sử dụng trực tiếp trên lớp mà không cần khởi tạo.

- Các phương thức tĩnh không tương tác với các thể hiện (instance) của lớp.

- Phương thức static chỉ có thể truy cập vào các thuộc tính static và chỉ có thể gọi các phương thức static trong cùng lớp.

Giải thích:

Các thành phần static được khởi tạo đồng thời với lớp trong khi các thuộc tính, phương thức non-static chỉ tồn tại khi chúng ta khởi tạo đối tượng của lớp đó. Khi các đối tượng được khởi tạo, nếu phương thức static truy cập vào các thuộc tính, phương thức non-static thì nó không biết phải sử dụng các thuộc tính, phương thức của thể hiện nào của lớp.

**Câu 7:**

Mối quan hệ giữa UML và quá trình phát triển phần mềm là:  
Giai đoạn khảo sát dự án:

* Biểu đồ use-case được sử dụng để nắm bắt, đặc tả yêu cầu của phần mềm. Các chức năng, yêu cầu được mô hình hóa thành các tác nhân và use-case.

Giai đoạn phân tích thiết kế phần mềm:

* Biểu đồ use-case và biểu đồ lớp trong các tài liệu thiết kế hệ thống. Biểu đồ lớp chỉ ra cấu trúc tĩnh của các lớp trong phần mềm.
* Biểu đồ tuần tự, biểu đồ trạng thái để xác định trình tự diễn ra sự kiện, luồng xử lý, trạng thái của các đối tượng trong phần mềm.

Giai đoạn thực hiện:

* Dựa vào tài liệu đặc tả phần mềm và các biểu đồ trên để xây dựng hệ thống theo yêu cầu.

Giai đoạn kiểm thử:

* Thử nghiệm đơn vị sử dụng biểu đồ lớp, đặc tả lớp.
* Thử nghiệm hệ thống sử dụng biểu đồ use-case.
* Thử nghiệm tích hợp sử dụng các biểu đồ thành phần, biểu đồ cộng tác.

Giai đoạn triển khai và bảo trì:

* Sử dụng biểu đồ triển khai để triển khai phần mềm.
* Sử dụng biểu đồ thành phần để viết tài liệu hướng dẫn sử dụng.

**Câu 8:**

Những điểm nổi bật của mô hình xử lý ngoại lệ:

* Tách biệt phần mã xử lý lỗi với phần mã của chương trình. => Dễ dàng trong việc xử lý và bảo trì.
* Gom nhóm và phân loại, phân cấp các ngoại lệ => Xử lý được nhiều loại ngoại lệ, không bỏ sót các ngoại lệ.
* Xử lý trực tiếp ngoại lệ hoặc ủy nhiệm cho các phương thức gọi đến nó.

=> Làm cho chương trình dễ đọc và an toàn hơn.

So sánh với các xử lý thông thường:

* Lỗi được xử lý trực tiếp trong mã thực thi của chương trình => Chương trình trở nên rối, không tách bạch được phần xử lý lỗi, khó đọc và khó bảo trì.
* Không bắt hết các trường hợp xảy ra lỗi.
* Truyền trạng thái lỗi lên mức trên thông qua flag dễ gây nhầm lẫn, khó hiểu.

**Câu 9:**

Nhận xét trên đúng:

* Khi phương thức khởi tạo sử dụng từ khóa abstract thì phương thức này không có phần thân và nó được thực hiện ở 1 lớp con. Nhưng constructor được gọi ngầm khi khởi tạo với từ khóa new, bắt buộc phải có phần thân.
* Khi phương thức khởi tạo sử dụng từ khóa static thì có nghĩa là phương thức ấy thuộc về lớp, không thuộc về đối tượng cụ thể. Nhưng constructor lại được sử dụng để khởi tạo 1 đối tượng, vì vậy không có tác dụng khi có 1 hàm khởi tạo tĩnh.
* Khi phương thức khởi tạo sử dụng từ khóa final thì có nghĩa là phương thức này không được override. Mà theo đặc tả ngôn ngữ Java thì constructor không thể bị kế thừa nên không thể ghi đè. Phương thức tạo với từ khóa final không có tác dụng.

**Câu 10:**

Giao diện trong lập trình hướng đối tượng là là tập các khai báo mà không có phần định nghĩa. Giao diện dùng để xác định tập các hành vi cần có của 1 đối tượng.

Giao diện trong Java không thay thế được lớp trừu tượng vì:

Bản chất của lớp trừu tượng (abstract class) khác hoàn toàn với giao diện (interface).

Các lớp trừu tượng chứa các thuộc tính/ phương thức static và non-static và có thể có các phương thức được thực hiện. Điều này Interface không làm được.

Các lớp trừu tượng không hỗ trợ đa kế thừa như interface để giảm độ phức tập và đơn giản hóa ngôn ngữ.

**Câu 11:**

|  |  |
| --- | --- |
| Lớp (class) | Đối tượng (object) |
| - Lớp là mô hình khái niệm, mô tả các thực thể.  - Lớp là bản mẫu (prototype) định nghĩa các thuộc tính và phương thức chung của đối tượng.  - Một lớp là sự trừu tượng hóa của 1 tập các đối tượng | - Đối tượng là các sự vật thật, thực thể thật sự.  - Đối tượng là 1 thể hiện (instance) của 1 lớp, dữ liệu của các đối tượng khác nhau là khác nhau.  - Mỗi đối tượng có 1 lớp xác định các hành vi và dữ liệu của nó. |

**Câu 12:**

|  |  |
| --- | --- |
| Hướng đối tượng | Hướng cấu trúc |
| - Là sự kết hợp giữa các đối tượng + thông điệp.  - Chương trình được chia thành các đối tượng, gửi thông điệp cho nhau.  - Có 3 mức độ truy cập: public, protect, private.  - Hướng đối tượng có thể thực thi đa luồng.  - Mức độ bảo mật cao hơn do hạn chế truy cập.  - Việc mở rộng hoặc bảo trì hệ thống dễ dàng. | - Là sự kết hợp giữa function + thuật toán.  - Chương trình được chia thành các hàm.  - Không có các mức độ truy cập khác nhau.  - Thực hiện tuần tự các chức năng của chương trình.  - Mức độ bảo mật thấp.  - Khó trong việc mở rộng và bảo trì. |

**Câu 15:**

Trừu tượng hóa là quá trình đơn giản hóa 1 đối tượng mà trong đó chỉ bao gồm các quan tâm, các đặc điểm tổng quát, bỏ qua các đặc điểm chi tiết. Quá trình trừu tượng hóa dữ liệu giúp xác định các thuộc tính và hành vi của 1 đối tượng. Tập hợp các thuộc tính, hành vi chung của 1 nhóm đối tượng thành 1 lớp.

Trừu tượng hóa cho phép quản lý các bài toán phức tạp bằng cách tập trung vào các đặc điểm quan trọng của 1 thưc thể nhằm phân biệt nó với các loại thưc thể khác.

**Câu 16:**

Đóng gói là sự che giấu bên trong dữ liệu riêng của mỗi đối tượng của lớp được khai báo và chỉ được truy xuất thông qua hệ thống các phương thức có sẵn của lớp.

Đặc điểm:

* Tạo ra cơ chế để ngăn ngừa việc gọi phương thức của lớp này tác động hay truy xuất dữ liệu của đối tượng thuộc về lớp khác.
* Dữ liệu riêng (khi được khai báo là private) của mỗi đối tượng được bảo vệ khỏi sự truy xuất không hợp lệ từ bên ngoài.
* Người lập trình có thể dựa vào cơ chế này để ngăn ngừa việc gán giá trị không hợp lệ vào thành phần dữ liệu của mỗi đối tượng.
* Cho phép thay đổi cấu trúc bên trong của một lớp mà không làm ảnh hưởng đến những lớp bên ngoài có sử dụng lớp đó.

**Câu 17:**

Ưu điểm:

* Đóng gói các hành vi liên quan đến getting và setting thuộc tính của lớp. Điều này cho phép thêm các chức năng bổ sung (xác thực dữ liệu) cho các thuộc tính.
* Ẩn đi dữ các thuộc tính private bên trong, chỉ cho truy xuất, sửa qua các phương thức getter/setter
* Cho phép các đối tượng kế thừa có thể định nghĩa lại các hành vi của đối tượng bằng cách ghi đè các phương thức getter/setter.
* Cho phép truy cập getter và setter ở mức độ khác nhau.

Có thể tạo nhiều phương thức getter/setter bằng cách overloading các phương thức đó.

**Câu 18:**

Biến this trong Java là biến tham chiếu đề cập đến đối tượng hiện tại.

Biến this trong Java có 6 cách sử dụng:

1. Tham chiếu đến biến thể hiện của lớp hiện tại.

2. Gọi phương thức của lớp hiện tại.

3. this() để gọi hàm constructor của lớp hiện tại.

4. Như là 1 đối số trọng lệnh gọi phương thức.

5. Như là 1 đối số trong hàm constructor.

6. Trả về thể hiện của lớp hiện tại từ phương thức.

Link tham khảo: <https://www.javatpoint.com/this-keyword>

**Câu 19:**

Overloading là các phương thức trong 1 lớp có cùng tên nhưng khác nhau về số lượng tham số, và kiểu tham số.

Overloading tăng khả năng đọc chương trình.

Overriding là các phương thức của lớp con giống hoàn toàn về chữ ký với phương thức đã được định nghĩa trong lớp cha. Nói cách khác thì lớp con cung cấp cách triển khai cụ thể với các phương thức được khai báo bởi cha nói.

Overriding cung cấp việc triển khai cụ thể của các phương thức được khai báo trong lớp nó kế thừa. Overriding sử dụng cho đa hình trong thời gian chạy.

So sánh overloading và overriding:

|  |  |
| --- | --- |
| Overloading | Overriding |
| - Sử dụng để tăng khả năng đọc chương trình.  - Các phương thức overloading nằm trong cùng 1 class.  - Các phương thức cùng tên nhưng khác nhau về chữ kí phương thức.  - Thời gian xác định đoạn mã được thực hiện là thời gian biên dịch.  - Kiểu dữ liệu trả về của phương thức overloading có thể giống hoặc khác nhau. | - Cung cấp các cách thực hiện cụ thể với các phương thức đã được khai báo trong các lớp cha.  - Các phương thức overriding nằm trong 2 class khác nhau có mối quan hệ kế thừa.  - Chữ kí phương thức phải giống nhau.  - Thời gian xác định đoạn mã được thực hiện là thời gian chạy.  - Kiểu dữ liệu trả về phải giống với phương thức bị ghi đè. |

**Câu 20:**

So sánh class member và instance member:

|  |  |
| --- | --- |
| Instance member | Class member |
| - Thuộc tính/ phương thức truy cập thông qua các đối tượng.  - Mỗi đối tượng có 1 bản sao riêng của 1 thuộc tính đối tượng.  - Giá trị của 1 thuộc tính đối tượng của các đối tượng khác nhau là khác nhau. | - Thuộc tính/ phương thức truy cập thông qua lớp.  - Các đối tượng có chung 1 bản sao của thuộc tính lớp.  - Giá trị của 1 thuộc tính lớp của các đối tượng khác nhau là giống nhau. |

**Câu 21:**

Các lợi ích của package trong Java:

* Định nghĩa 1 nhóm các lớp, các giao diện theo 1 package riêng, dễ dàng quản lý, theo dõi.
* Sử dụng package cung cấp việc kiểm soát truy cập dễ dàng hơn.
* Dễ dàng để xác định vị trí các lớp liên quan.
* Các package được khai báo trong các không gian riêng biệt, tránh xung đột tên với tên trong các package khác nhau.
* Cung cấp khả năng tái sử dụng lại mã nguồn trong các lớp, giao diện cùng package.

**Câu 22:**  
So sánh kết tập và kế thừa:

|  |  |
| --- | --- |
| Kết tập | Kế thừa |
| - Đều là các kĩ thuật tái sử dụng mã nguồn trong OOP | |
| - Tái sử dụng mã nguồn thông qua các đối tượng. Tạo lớp mới là tập các đối tượng của lớp đã có. Lớp mới sử dụng các thuộc tính/ hành vi thông qua các đối tượng thành phần.  - Quan hệ của kết tập là quan hệ “là 1 phần” (“is a part of”). | - Tái sử dụng mã nguồn thông qua các lớp. Các lớp con kế thừa lại các thuộc tính/ hành vi của lớp cha.  - Quan hệ của kế thừa là quan hệ “là 1 loại” (“is a kind of”) |

**Câu 23:**

Đa hình trong Java là cách thực hiện 1 hành động, thông điệp theo nhiều cách khác nhau.

Có 2 kiểu đa hình:

- Đa hình trong thời gian biên dịch (Đa hình phương thức): Các phương thức overloading. Các đối tượng thực hiện thông điệp theo các ngữ cảnh khác nhau.

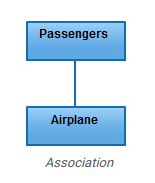
- Đa hình trong thời gian chạy (Đa hình đối tượng): Các phương thức overriding. Các đối tượng khác nhau có cách giải nghĩa thông điệp theo cách khác nhau.

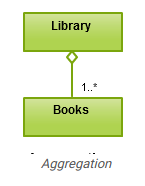
**Câu 24:**

So sánh abstract với interface:

|  |  |
| --- | --- |
| Abstract class | Interface |
| - Đều chứa các phương thức chưa được định nghĩa.  - Đều có tính kế thừa. | |
| - Lớp trừu tượng chứa các phương thức trừu tượng và các phương thức không trừu tượng.  - Sử dụng từ khóa abstract để định nghĩa.  - Các lớp con sử dụng từ khóa ‘extends’ để kế thừa.  - Có các biến final, non-final, static, non-static.  - Phạm vi truy cập: public, private, protect.  - Có thể tái sử dụng mã nguồn.  - Các lớp con kế thừa lớp trừu tượng có cùng bản chất với lớp trừu tượng.  - Không hỗ trợ đa kế thừa. | - Giao diện chỉ chứa các phương thức trừu tượng.  - Sử dụng từ khóa interface để định nghĩa.  - Các lớp con sử dụng từ khóa ‘impliments’ để thực thi interface.  - Chỉ có biến final, static.  - Phạm vi truy cập: public.  - Không thể tái sử dụng mã nguồn.  - Các lớp thực hiện giao diện có thể không cùng bản chất với nhau.  - Hỗ trợ đa kế thừa. |

**Câu 26: Tổng quan về biểu đồ lớp, các mối quan hệ trong class diagram**

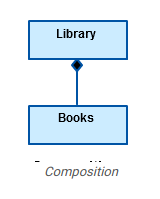
Quan hệ Association: Airplane có thể có hoặc không có Passengers.



Quan hệ Aggregation (Tập hợp): Library là lớp tổng hợp của 1 hoặc nhiều lớp Books.

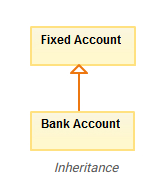
Lớp Books là 1 bộ phận của lớp Library.

Lớp Books và lớp Library không phụ thuộc vào nhau.



Quan hệ Composition (Thành phần): Giống như quan hệ aggregation nhưng

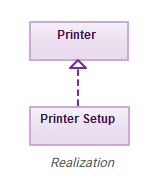
lớp Books và lớp Library phụ thuộc vào nhau. Nghĩa là khi Library biến mất thì lớp Books cũng biến mất theo. Còn aggregation thì lớp Books vẫn tồn tại khi lớp Library biến mất đi.



Quan hệ Inheritance / Generalization (Kế thừa): Là quan hệ kế thừa hay là tổng quát hóa. Dùng cho class con kế thừa abstract class.

VD: Bank Account kế thừa Fixed Account.

extends

Quan hệ realization (hiện thực hóa): Là quan hệ class A hiện thực hóa class B. Dùng cho class A impliments class B.

VD: Class Printer sẽ hiện thực hóa interface Printer Setup.

impliments