Các đặc tính của lập trình hướng đối tượng:

- Tính đóng gói (Encapsulation)
- Tính kế thừa (Inheritance)
- Tính đa hình (Polymorphism)
- Tính trừu tượng (Abstraction)

1. Tính đóng gói

- Tính đóng gói (Encapsulation): đảm bảo những dữ liệu, cài đặt quan trọng được ẩn đi khỏi người dùng. Dữ liệu và các hành động liên quan tới dữ liệu của lớp nào thì gói gọn bên trong lớp đó. Định nghĩa thuộc tính sử dụng từ khóa private. Cung cấp phương thức public là **get** và **set** để truy cập và chỉnh sửa giá trị **private** của biến. (Ví dụ về đóng gói: Một viên thuốc hạ sốt, ta chỉ biết chức năng là hạ sốt và một số thành phần chính, còn cụ thể bên trong có những gì thì không biết)

Tại sao phải đóng gói:

- Kiểm soát tốt hơn thuộc tính và phương thức của class
- Thuộc tính của class có thể **read-only** (nếu chỉ sử dụng phương thức **get**), hoặc **write-only** (nếu chỉ sử dụng phương thức **set**)
- Linh hoạt (flexible): người lập trình có thể thay đổi một một phần của code mà không làm ảnh hưởng đến các phần khác
- Tăng độ bảo mật dữ liệu

2. Tính kế thừa (Inheritance)

- Tính kế thừa: kế thừa các thuộc tính và phương thức từ một class này sang class khác. Các lớp con không phải định nghĩa lại, ngoài ra có thể mở rộng các thành phần kế thừa và bổ sung thêm các thành phần mới.
    - **subclass** (class con): class kế thừa từ class cha
    - **superclass** (class cha): class cha
- Để kế thừa từ một class, sử dụng từ khóa **extends.** Sử dụng access modifier là **protected** cho thuộc tính của class cha để cho phép class con có thể kế thừa các thuộc tính đó.

Tại sao phải sử dụng kế thừa:

- Tăng khả năng tái sử dụng code: sử dụng các thuộc tính và phương thức của class đã có khi tạo một class mới.

Tính đa hình sử dụng các phương thức kế thừa để thực hiện các tác vụ khác nhau

- VD: Với 2 lớp Android, iPhone, mỗi lớp đều đại diện cho một loại smartphone khác nhau nhưng lại có những phương thức giống nhau như gọi điện, nhắn tin, chụp hình. THay vì sao chép những thuộc tính này, ta đặt chúng vào một lớp chung gọi là lớp cha. Có thể định nghĩa lớp cha là SmartPhone có những lớp con kế thừa từ nó.

3. Tính đa hình (Polymorphism)

- **Tính đa hình là với một hành động có thể được thực hiện bằng nhiều cách khác nhau. Hiểu đơn giản, đa hình là khái niệm nhiều lớp có những phương thức giống nhau nhưng có thể thực thi theo những cách khác nhau. Ví dụ: mỗi smartphone kế thừa từ lớp cha SmartPhone nhưng có thể lưu trữ dữ liệu trên cloud theo những cách khác nhau. Android lưu trữ bằng Google Drive, iPhone lưu trên iCloud.**
- Tính đa hình xảy ra khi ta có nhiều lớp có liên quan đến nhau bởi kế thừa.
- **Tính kế thừa cho phép chúng ta kế thừa các thuộc tính và phương thức từ một class khác. Đa hình sử dụng các phương thức đó để thực hiện các nhiệm vụ khác nhau. CHo phép thực hiện một hành động với nhiều cách khác nhau.**

- VD: superclass là Animal có phương thức là animalSound(). Các subclass của Animal là Pigs, Cats, Dogs… và chúng cũng có phương thức animalSound() của chúng
- Tại sao và khi nào sử dụng tính kế thừa và đa hình? Vì hữu dụng cho tái sử dụng code: sử dụng lại thuộc tính và phương thức từ một class đã có khi tạo một class mới.

```
class Animal {
  public void animalSound() {
    System.out.println("The animal makes a sound");
  }
}
class Pig extends Animal {
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
}
class Dog extends Animal {
  public void animalSound() {
    System.out.println("The dog says: bow wow");
  }
}
class Main {
  public static void main(String[] args) {
    Animal myAnimal = new Animal();  // Create a Animal object
    Animal myPig = new Pig();  // Create a Pig object
    Animal myDog = new Dog();  // Create a Dog object
    myAnimal.animalSound();
    myPig.animalSound();
    myDog.animalSound();
  }

}
```

4. **Tính trừu tượng**

- **Trừu tượng có nghĩa là tổng quát hóa một cái gì đó, không cần chú ý chi tiết bên trong. Nó không màng đến chi tiết bên trong là gì và người ta vẫn hiểu mỗi khi nghe về nó. VD: Khi đi xe có hành động tăng ga, thì chức năng tăng ga đại diện cho trừu tượng (abstraction). Người dùng chỉ cần biết là tăng ga thì xe tăng tốc, không cần biết bên trong nó hoạt động thế nào.**
- **VD: Với bài toán quản lý sinh viên, chỉ cần quản lý các thông tin như: Họ tên, ngày sinh, giới tính, điểm thi… mà không cần quản lý thêm các thông tin: màu tóc, sở thích, chiều cao…**
- Trừu tượng hóa dữ liệu là quá trình ẩn đi chi tiết nhất định và chỉ thể hiện những thông tin cần thiết đến người dùng.
- Tính trừu tượng có thể đạt được với **abstract class** hoặc **interface.**
- Từ khóa **abstract** là một non-access modifier, sử dụng cho class và phương thức

- **Abstract class:** là một class hạn chế không dùng để tạo object (để sử dụng, phải kế thừa từ một class khác)
- **Abstract method:** chỉ sử dụng trong một **abstract class**, và không có phần nội dung phương thức. Nội dung phương thức được cung cấp bởi subclass (lớp kế thừa).

VD: Abstract class và abstract method

```
abstract class Animal {
  public abstract void animalSound();
  public void sleep() {
    System.out.println("Zzz");
  }
}
// Subclass (inherit from Animal)
class Pig extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}
class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

Tại sao và khi nào sử dụng Abstract class và methods? Để đạt tính bảo mật, ẩn các chi tiết nhất định và chỉ hiển thị các chi tiết quan trọng của một đối tượng.

5. Interfaces

- Một cách khác để trừu tượng hóa trong Java đó là sử dụng interfaces
- Interfaces là một lớp trừu tượng hoàn toàn được sử dụng để nhóm các phương thức liên quan không có định nghĩa.

- Để sử dụng interface methods (các phương thức giao diện), interface phải được "implemented" (thực hiện) giống như kế thừa, bởi một class khác với từ khóa **implements (**thay vì extends). Body của phương thức interface được cung cấp bởi implement class
- Lưu ý với interface:
    - Giống như **abstract class, interface** không được dùng để tạo đối tượng
    - Interface method không có phần định nghĩa, phần định nghĩa được cung cấp bởi implement class.
    - Một implementation của một interface, phải ghi đè tất cả phương thức của nó
    - **Interface methods** mặc định là **abstract** và **public**
    - **Interface attributes** mặc định là **public, static, final**
    - Một **interface** không thể chứa một constructor (vì thế không sử dụng để tạo object)
- Tại sao mà khi nào sử dụng Interfaces
    - Để đạt tính bảo mật, ẩn các chi tiết nhất định và chỉ hiển thị các chi tiết quan trọng của một đối tượng (interface)
    - Java không hỗ trợ "đa kế thừa" (một class có thể kế thừa từ nhiều class cha). Tuy nhiên, điều đó có thể được với interfaces, vì class có thể implement nhiều interfaces, ngăn cách nhau bởi dấu , (VD: `class DemoClass implements FirstInterface, SecondInterface)`

VD: Một interface với các phương thức chưa được định nghĩa

```
// interface
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void run(); // interface method (does not have a body)
}
// Pig "implements" the Animal interface
class Pig implements Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
  public void sleep() {
    // The body of sleep() is provided here
    System.out.println("Zzz");
  }
}
class Main {
  public static void main(String[] args) {
```

```
    Pig myPig = new Pig();  // Create a Pig object

    myPig.animalSound();

    myPig.sleep();

  }

}
```

==========

Modifiers chia thành 2 nhóm:

- **Access Modifiers:** kiểm soát mức độ truy cập
- **Non-Access Modifiers:** không kiểm soát mức độ truy cập nhưng cung cấp chức năng khác

6. Access Modifiers

- Với Class, có thể sử dụng **public** hoặc *default*
  - **public:** Class có thể được truy cập từ bất cứ class nào khác
  - *default:* Class chỉ có thể truy cập bởi class khác cùng package
- Với attributes, methods và constructor, có thể sử dụng:
  - **public:** code có thể truy cập cho tất cả các class
  - **private:** chỉ có thể truy cập nội bộ bên trong class đã khai báo
  - **protected:** chỉ có thể truy cập bởi class có cùng package và **subclass**
  - *default:* chỉ có thể truy cập trong cùng pakage

Access Modifier	Truy cập bên trong class?	Truy cập bên trong package?	Truy cập bên ngoài package bởi class con?	Truy cập bên ngoài class và không thuộc class con?
private	Y			
Default	Y	Y		
protected	Y	Y	Y	
public	Y	Y	Y	Y

7. Non-Access Modifiers

- Với Class:
  - **final:** Class không cho phép kế thừa bởi class khác
  - **abstract:** Class không thể tạo object (để sử dụng abstract class, nó phải được kế thừa từ class khác)
- Với attributes, methods:
  - **final:** thuộc tính, phương thức không cho phép ghi đè, sửa
  - **static:** thuộc tính và phương thức thuộc về lớp chứ không thuộc về đối tượng
  - **abstract:** sử dụng trong một abstract class, và sử dụng trên methods. Methods không có phần body, phần body được cung cấp bởi subclass (class con kế thừa)

- **transient (tạm thời):** thuộc tính và phương thức bị bỏ qua khi tuần tự hóa đối tượng chứa chúng
- **synchronized (đồng bộ):** phương thức chỉ có thể truy cập bởi một luồng tại một thời điểm
- **volatile:** giá trị của thuộc tính không được lưu trong cache mà luôn đọc từ bộ nhớ chính

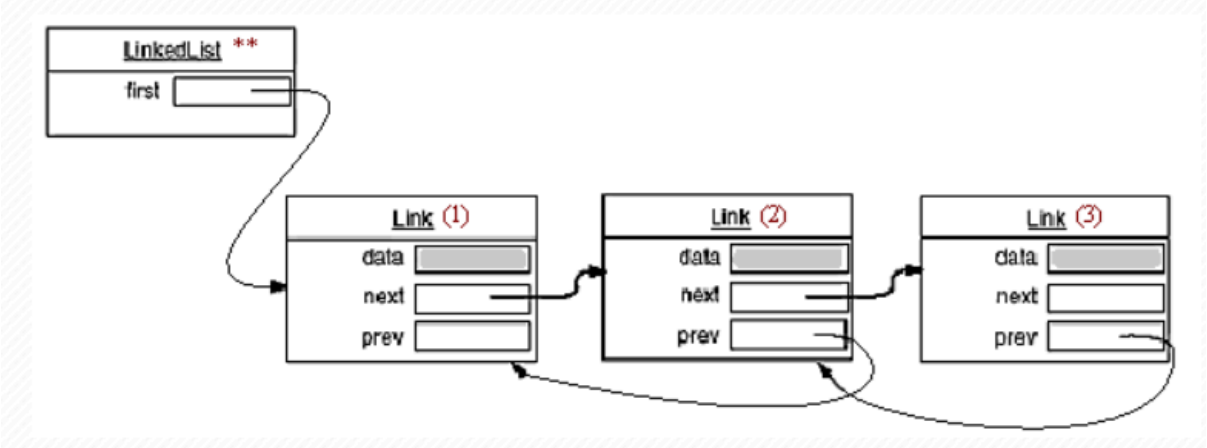
8. Khác biệt giữa ArrayList và Array

- **Array** không thể thay đổi kích thước (nếu muốn thêm hoặc xóa phần tử từ mảng thì phải tạo một Array mới)
- **ArrayList** có thể thêm và loại phần từ. Phần tử của ArrayList là objects. Để sử dụng ArrayList với các phần tử là số nguyên int, phải dùng wrapper class Integer, Boolean cho boolean, Character cho char, Double cho double

Array	**ArrayList**
Kích thước cố định	Kích thước có thể thay đổi được
Có thể lưu trữ dữ liệu kiểu nguyên thủy và đối tượng	Chỉ có thể lưu trữ dữ liệu kiểu đối tượng. Để sử dụng ArrayList với các phần tử là kiểu nguyên thủy, phải dùng wrapper class Integer cho int, Boolean cho boolean, Character cho char, Double cho double
Tốc độ lưu trữ và thao tác nhanh hơn	Tốc độ lưu trữ và thao tác chậm hơn
Chỉ có thuộc tính length	Có nhiều phương thức để thao tác với dữ liệu

9. ArrayList và LinkedList

- **LinkedList** và **ArrayList** giống nhau là cùng chứa các object cùng loại. Các phương thức của ArrayList và LinkedList giống nhau vì cùng implement List interface (thêm, sửa, xóa và xóa list)
- Lớp **ArrayList** có một array thông thường bên trong. Khi thêm một phần tử vào, nó được đặt vào trong mảng. Nếu mảng không đủ lớn, một mảng mới lớn hơn được tạo thay thế cho mảng cũ và mảng cũ bị xóa.
- **LinkedList** chứa các phần tử dưới dạng “Node”.


LinkedList **
first
Link (1)
data
next
prev
Link (2)
data
next
prev
Link (3)
data
next
prev

-

- Khi nào sử dụng **LinkedList** và **ArrayList?**
    - Sử dụng **ArrayList** để lưu trữ và truy cập dữ liệu
    - Sử dụng **LinkedList** để thao tác với dữ liệu

	ArrayList	**LinkedList**
Cấu trúc dữ liệu	ArrayList sử dụng mảng động để lưu trữ các phần tử ArrayList là một cấu trúc dữ liệu dựa trên chỉ mục (index), trong đó mỗi phần tử (element) được liên kết với một chỉ mục	LinkedList sử dụng danh sách liên kết để lưu trữ các phần tử Các phần tử trong LinkedList được gọi là node, mỗi node cần lưu trữ 3 thông tin: tham chiếu phần tử trước nó, giá trị của phần tử và tham chiếu tới phần tử kế tiếp
Thao tác thêm và xóa	**Chậm hơn**, vì nó sử dụng nội bộ mảng. Bởi vì sau khi thêm hoặc xoá các phần tử cần sắp xếp lại	**Nhanh hơn**. Bởi vì nó không cần sắp xếp lại các phần tử sau khi thêm hoặc xóa. Nó chỉ cần cập nhật lại tham chiếu tới phần tử trước và sau nó.
Thao tác tìm kiếm hoặc truy xuất phần tử	**Nhanh hơn** LinkedList. Vì các phần tử trong ArrayList được lưu trữ dựa trên chỉ mục (index)	**Chậm hơn,** Vì LinkedList phải duyệt qua lần lượt các phần tử từ đầu tiên đến cuối cùng
Truy xuất ngẫu nhiên	ArrayList có thể truy xuất ngẫu nhiên phần tử	LinkedList **Không** thể truy xuất ngẫu nhiên. Nó phải duyệt qua tất cả các phần tử từ đầu tiên đến cuối cùng để tìm phần tử
Trường hợp sử dụng	ArrayList chỉ có thể hoạt động như một list vì nó chỉ implements giao tiếp List	LinkedList có thể hoạt động như một ArrayList, Stack, Queue, Singly Linked List và Doubly Linked List vì nó implements các giao tiếp List và Deque
Sử dụng bộ nhớ	**Ít bộ nhớ hơn** so với LinkedList. Vì ArrayList chỉ lưu dữ liệu và chỉ mục của nó	**Nhiều bộ nhớ hơn** so với ArrayList vì LinkedList lưu trữ thông tin của nó và tham chiếu tới phần tử trước và sau
Khi nào sử dụng	**ArrayList** tốt hơn trong việc lưu trữ và truy xuất	**LinkedList** tốt hơn trong việc thao tác dữ liệu (thêm, xóa)

10. Các thuật toán sắp xếp

Bubble Sort (Sắp xếp nổi bọt). Sắp xếp dựa trên việc so sánh cặp phần tử liền kề nhau và tráo đổi nếu chúng không theo thứ tự. Không thích hợp với tập dữ liệu lớn vì tốc độ chậm nhất

Selection Sort (Sắp xếp lựa chọn). Phần tử nhỏ nhất lần lượt được chọn ra trong mảng chưa được sắp xếp và thêm vào mảng đã được sắp xếp

Insertion Sort (Sắp xếp chèn). Một danh sách con luôn được duy trì dưới dạng đã qua sắp xếp. Sắp xếp chèn là chèn thêm một phần tử vào danh sách con đã qua sắp xếp. Phần tử được chèn vào vị trí thích hợp sao cho vẫn đảm bảo danh sách con vẫn sắp xếp theo thứ tự

Quick Sort (Sắp xếp nhanh) (Left Right Pointer). Giải thuật hiệu quả cao dựa trên việc chia mảng thành các mảng nhỏ hơn. Giải thuật chia mảng thành hai phần bằng cách so sánh từng phần tử của mảng với một phần tử gọi là phần tử chốt (Pivot). Một mảng bao gồm các phần tử nhỏ hơn hoặc bằng phần tử chốt và mảng còn lại bao gồm các phần tử lớn hơn hoặc bằng phần tử chốt

Merge Sort (Sắp xếp trộn). Một giải thuật sắp xếp dựa trên giải thuật chia để trị. Đầu tiên, giải thuật sắp xếp trộn chia mảng thành hai nửa và sau đó kết hợp chúng lại với nhau thành một mảng đã được sắp xếp

11. Các giải thuật tìm kiếm

Tìm kiếm tuyến tính. Mỗi phần tử đều được kiểm tra và nếu tìm thấy bất kỳ kết nối nào thì phần tử cụ thể đó được trả về, nếu không tìm thấy thì quá trình tìm kiếm tiếp tục diễn ra cho tới khi tìm kiếm hết dữ liệu

Tìm kiếm nhị phân. (Mảng cần phải được sắp xếp) Tìm kiếm một phần tử cụ thể bằng cách so sánh phần tử tại vị trí giữa nhất của tập dữ liệu. Nếu tìm thấy kết quả thì trả về. Nếu phần tử cần tìm là lớn hơn giá trị phần tử giữa thì phần tử cần tìm được tìm trong mảng con nằm ở bên phải phần tử giữa và ngược lại. Tiến trình sẽ tiếp tục như vậy trên mảng con cho tới khi tìm hết mọi phần tử trên mảng con này.

Tìm kiếm nội suy. biến thể của tìm kiếm nhị phân. Để giải thuật tìm kiếm này làm việc chính xác thì tập dữ liệu phải được sắp xếp

12. Phân biệt Abstract - Interface

- **Inheritance:** class abstract có thể kế thừa được một class khác. Trong khi interface có thể kế thừa nhiều Interface khác
- **Accessibility:** các thành viên trong Interface kiểu mặc định là public. Trong khi class abstract có thể là private, protected,...

Abstract. tức một cái gì đó không hoàn toàn cụ thể, chỉ là một ý tưởng hoặc ý chính của một cái gì đó mà không có bản triển khai cụ thể. Vì vậy class abstract chỉ là một cấu trúc hoặc hướng dẫn được tạo cho các class cụ thể khác. Class abstract là linh hồn của một class cụ thể, và một cơ thể (class) không thể có hai linh hồn. Vì thế Java không hỗ trợ nhiều kế thừa cho các class abstract

Interface. (giao diện) là một hình thức, giống như một hợp đồng, nó không thể tự làm bất cứ điều gì. Nhưng khi có một class ký kết hợp đồng (implement interface) này, thì class đó

phải tuân theo interface (hợp đồng). Trong interface định nghĩa các hành vi của một class sẽ thực hiện. Một class có thể có một số cách hành vi khác nhau, cũng giống như nó có thể ký kết với nhiều hợp đồng khác nhau. Đó là lý do tại sao Java cho phép implement nhiều interface.

Interface khác với abstract class vì nó không phải là class. Nó đơn giản chỉ là một dạng thiết yếu có thể được thỏa mãn bởi bất cứ class nào implement interface đó. Bất cứ class nào implement một interface phải implement toàn bộ method được liệt ra trong định nghĩa của interface

Abstract class có thể có implementation code, còn interface thì không có

Khi nào nên dùng?

- Tạo một class abstract khi bạn đang cung cấp các hướng dẫn cho một class cụ thể
- Tạo interface khi chung ta cung cấp các hành vi bổ sung cho class cụ thể và những hành vi này không bắt buộc với class đó
- Một class nhất định phải được biểu thị bằng abstract nếu nó có nhiều hơn một abstract method. Một method được biểu thị là abstract nếu nó có method heading tuy nhiên phần thân là rỗng - có nghĩa là một abstract method sẽ không có implementation code bên trong dấu ngoặc như những method khác

```
/* Figure class phải được biểu thị là abstract do nó có chứa  abstract method

public abstract class Figure
{
        /* do đây là abstract method cho nên phần thân sẽ rỗng  */
        public abstract float getArea();
}

public class Circle extends Figure
{
        private float radius;

        public float getArea()
        {
                return (3.14 * (radius ^ 2));
        }
}

public class Rectangle extends Figure
{
        private float length, width;

        public float getArea(Figure other)
        {
                return length * width;
        }
```

Trong ví dụ trên bản thân Figure class không phải là một dạng nhất định giống như Circle hay Rectangle, chính vì vậy chúng ta không thể có một định nghĩa cụ thể nào cho getArea method trong Figure class. Điều đó lý giải tại sao phải biểu thị getArea method là abstract trong Figure class

- Sử dụng abstract class nếu dự định sử dụng kế thừa và cung cấp base class phổ thông cho những class bắt nguồn từ nó
- Abstract class rất hữu dụng nếu muốn có thể biểu thị những phần thử không public. Trong một interface, tất cả các method phải là public
- Nếu cần phải thêm method trong tương lai thì nên chọn abstract class. Vì nếu thêm những method rỗng vào trong interface thì tất cả các class vốn dĩ đã implement interface này phải đổi sang hết implement những method mới
- Interface rất tốt nếu muốn có kiểu đa kế thừa trong hệ thống khi có thể tạo nhiều interface và bắt class implement chúng

13. **Overloading (Nạp chồng phương thức) và Overriding (Ghi đè phương thức**

Nạp chồng phương thức	Ghi đè phương thức
Nạp chồng phương thức giúp code dễ đọc hơn	Ghi đè phương thức được sử dụng để cung cấp cài đặt cụ thể cho phương thức được khai báo ở lớp cha
Nạp chồng được thực hiện bên trong một class	Ghi đè phương thức xảy ra trong 2 class có quan hệ kế thừa
Nạp chồng phương thức thì tham số phải khác nhau	Ghi đè phương thức thì tham số phải giống nhau
Nạp chồng phương thức là ví dụ về đa hình lúc biên dịch	Ghi đè phương thức là ví dụ về đa hình lúc runtime
Trong java, nạp chồng phương thức không thể được thực hiện khi chỉ thay đổi kiểu giá trị trả về của phương thức. Kiểu giá trị trả về có thể giống hoặc khác nhau, nhưng tham số phải khác nhau	Giá trị trả về phải giống nhau

14. Collection trong Java

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

Comparable	Comparator
Nó cung cấp phước thức compareTo()	Nó cung cấp phương thức compare()
Đặt trong java.lang package	Đặt trong java.util package
Nếu một lớp được implement Comparable interface thì lớp đó phải được sửa đổi	Lớp không bị sửa đổi

15. Bộ nhớ Heap và Stack

- **Stack.** là một vùng nhớ được sử dụng để lưu trữ các tham số, các biến local của phương thức mỗi khi phương thức được gọi ra
- **Heap.** là một vùng nhớ được sử dụng để lưu trữ các đối tượng khi từ khóa new được gọi ra, các biến static, và các biến toàn cục (biến instance)

16. Từ khóa static

Biến static.

- Biến static dùng để tham chiếu thuộc tính chung của tất cả đối tượng (mà không phải duy nhất cho mỗi đối tượng).
- Biến static lấy bộ nhớ chỉ một lần tại thời gian tải lớp

Phương thức static.

- Một phương thức static thuộc lớp chứ không phải đối lượng của lớp
- Phương thức static có thể truy cập biến static và thay đổi giá trị
- Một phương thức static gọi mà không cần tạo instance của một lớp

Tại sao phương thức main là static

- Vì không cần thiết phải tạo đối tượng để gọi phương thức static.

Khối static.

- Được sử dụng để khởi tạo thành viên dữ liệu static
- Được thực thi trước phương thức main tại lúc tải lớp

17. Các câu hỏi phỏng vấn kế thừa

this trong java.

this là từ khóa tham chiếu đến đối tượng hiện tại

Kế thừa là gì?

Kế thừa là cơ chế trong đó một đối tượng được thừa hưởng tất cả các thuộc tính, phương thức của đối tượng khác của lớp khác. Nó được sử dụng để tái sử dụng code và ghi đè phương thức

Lớp nào là là cha cho tất cả các lớp?

Lớp Object

Tại sao đa kế thừa không được hỗ trợ trong java

GIảm thiểu sự phức tạp và đơn giản hóa ngôn ngữ

Tại sao java không hỗ trợ con trỏ?

Con trỏ là một biến tham chiếu tới địa chỉ ô nhớ. Nó không được sử dụng trong java thì không an toàn và phức tạp

super trong java là gì?

super là một từ khóa tham chiếu trực tiếp đến đối tượng của lớp cha

18. Các câu hỏi phỏng vấn về đa hình

Đa hình tại runtime là gì?

Đa hình tại runtime là quá trình gọi phương thức đã được ghi đè (overriding) trong thời gian thực thi chương trình. Trong quá trình này, một phương thức được ghi đè được gọi thông qua biến tham chiếu của một lớp cha

```
class Bike {
    void run() {
        System.out.println("running");
    }
}

public class Splender extends Bike {
    void run() {
        System.out.println("running safely with 60km");
    }

    public static void main(String args[]) {
        Bike b = new Splender();// upcasting
        b.run();
    }
}
```

Có thể thực hiện runtime với các thành viên dữ liệu không?

Không

Sự khác nhau giữa ràng buộc tĩnh và ràng buộc động là gì?

Kiểu ràng buộc tĩnh của đối tượng được xác định tại lúc biên dịch, còn kiểu ràng buộc động của đối tượng được xác định tại runtime

19. Câu hỏi phỏng vấn về trừu tượng

Trừu tượng là gì?

Tính trừu tượng là tiến trình ẩn đi các cài đặt chi tiết và chỉ hiển thị tính năng với người dùng. Sự trừu tượng khiến bạn tập trung vào đối tượng đó làm gì thay vì đối tượng đó làm như thế nào

Sự khác nhau giữa trừu tượng và đóng gói?

Trừu tượng là ẩn đi cài đặt chi tiết, đóng gói là gói code và data vào một khối duy nhất

Lớp trừu tượng là gì?

Một lớp được khai báo với từ khóa abstract là lớp trường tượng trong java. Cần có một lớp khác kế thừa nó và cài đặt phương thức của nó.

Có phương thức trừu tượng không nằm trong lớp trừu tượng không?

Không, nếu có bất kỳ phương thức trừu tượng nào trong lớp, thì lớp đó phải là lớp trừu tượng

Có thể sử dụng cả abstract và final cho một phương thức không?

Không, vì phương thức trừu tượng cần phải được ghi đè, trong khi không thể ghi đè phương thức final

Có thể tạo thể hiện của lớp trừu tượng không?

Không, lớp trừu tượng không có thể hiện

Interface là gì?

Một Interface trong Java là một bản thiết kế của một lớp. **Nó chỉ có các phương thức trừu tượng.** Interface là một kỹ thuật để thu được tính trừu tượng hoàn toàn và đa kế thừa trong Java

Có thể khai báo một phương thức của Interface với từ khóa static được không?

Không. Vì mặc định các phương thức của một Interface là trừu tượng, từ khóa static và abstract không thể được sử dụng cùng nhau

Một Interface có thể là final không?

Không. Vì các cài đặt phải được cung cấp bởi một lớp khác

20. Khác nhau giữa Abstract class và Interface

Lớp Abstract	**Interface**
Các phương thức có thể có nội dung	Chỉ có các phương thức trừu tượng
Có thể có các biến instance (biến khai báo trong một lớp)	Không thể có các biến instance
có thể có constructor	Không thể có Constructor
Có thể có các phương thức static	Không thể có các phương thức static
Một lớp chỉ có thể extends một lớp abstract	Một lớp có thể implement nhiều interface

Có thể định nghĩa private hoặc protected cho các biến trong interface không?

Không, phải là public

21. Các câu hỏi phỏng vấn overloading (nạp chồng) phương thức

Overloading (nạp chồng) phương thức là gì?

Một lớp có nhiều phương thức có tên giống nhau nhưng các tham số khác nhau được gọi là overloading phương thức. Giúp code rõ ràng, dễ hiểu

Tại sao overloading phương thức không xảy ra khi thay đổi kiểu giá trị trả về?

Bởi vì đó là sự không rõ ràng, không biết gọi phương thức nào khi runtime

Có thể overloading phương thức main() không?

Có thể

22. Các câu hỏi phỏng vấn overriding (ghi đè) phương thức

Overriding (ghi đè) phương thức là gì?

Nếu lớp con có phương thức giống lớp cha được gọi là ghi đè (overriding) phương thức trong java. Nói cách khác, nếu lớp con cung cấp sự cài đặt cụ thể cho phương thức đã được cung cấp bởi lớp cha được gọi là ghi đè phương thức trong java

Có thể ghi đè phương thức static không?

Không. Vì phương thức static thuộc về class chứ không thuộc về đối tượng, trong khi phương thức instance bị ràng buộc với đối tượng, static được lưu trong vùng nhớ Class và instance được lưu trong vùng nhớ heap

Có thể ghi đè phương thức đã nạp chồng?

Có

Có thể ghi đè biến instance không?

Không

23. Sự khác nhau giữa nạp chồng và ghi đè

Nạp chồng phương thức (overloading)	**Ghi đè phương thức (overriding)**
Giúp code dễ đọc hơn	Cung cấp các cài đặt cụ thể cho phương thức được khai báo ở lớp cha
Thực hiện bên trong một class	Xảy ra trong 2 class có quan hệ kế thừa
Nạp chồng phương thức thì **tham số phải khác nhau**	Ghi đè phương thức **tham số phải giống nhau**
Nạp chồng phương thức là ví dụ về đa hình lúc biên dịch	Ghi đè phương thức là ví dụ về đa hình lúc runtime

Trong java, nạp chồng không để được thực hiện khi chỉ thay đổi kiểu giá trị trả về của phương thức	Giá trị trả về phải giống nhau

24. Từ khóa final

Biến final là gì?

Nếu một biến được tạo với từ khóa final thì không thể thay đổi giá trị của biến đó

Phương thức final

Phương thức final không thể được ghi đè

Lớp final

Lớp không thể được kế thừa

Biến final blank

Một biến final không được khởi tạo giá trị lúc khai báo được gọi là biến final blank

Có thể khởi tạo giá trị cho biến final blank không?

Có, nếu biến đó là non-static thì chỉ khởi tạo được trong constructor. Nếu biến đó là static thì chỉ khởi tạo được trong khối static

Có thể khai báo phương thức main là final không?

Có, public static final void main(String[] args){}

25. Câu hỏi phỏng vấn xử lý ngoại lệ trong Java

Xử lý ngoại lệ (Handling exception) là gì?

Exception Handling trong Java là một cơ chế xử lý lỗi runtime để có thể duy trì luồng bình thường của ứng dụng

Có phải mỗi khối try phải đi kèm với một khối catch?

Mỗi khối try phải được theo sau bởi một khối catch hoặc một khối finally. Và bất kỳ trường hợp ngoại lệ có thể bị ném sẽ được khai báo với từ khóa throws của phương thức

Khối finally là gì?

- Khối lệnh finally trong java được sử dụng để thực thi các lệnh quan trọng như đóng kết nối,...
- Khối lệnh finally trong java luôn được thực thi cho dù có ngoại lệ xảy ra hay không
- Khối lệnh finally trong java được khai báo sau khối lệnh try hoặc sau khối lệnh catch

Khối finally có thể được sử dụng mà không cần khối catch không?

Có. Bởi khối try, khối finally phải theo sau khối try hoặc catch

Có trường hợp nào khối finally không được thực thi không?

Khối finally không được thực thi nếu chương trình bị thoát (bằng cách gọi System.exit() hoặc lỗi phần cứng)

26. Sự khác nhau giữa throw và throws

throw	**throws**
từ khóa throw trong java được sử dụng để ném ra một ngoại lệ rõ ràng	Từ khóa throws trong java được sử dụng để khai báo một ngoại lệ
Ngoại lệ checked không được truyền ra nếu chỉ sử dụng từ khóa throw	Ngoại lệ checked được truyền ra ngay cả khi chỉ sử dụng từ khóa throws
Sau throw là một instance	Sau throws là một hoặc nhiều class
Throw được sử dụng trong phương thức	Throws được khai báo ngay sau dấu đóng ngoặc đơn của phương thức
Không thể throw nhiều exceptions	Có thể khai báo nhiều exceptions. Ví dụ public void method() throws IOException, SQLException

27. Câu hỏi phỏng vấn String trong Java

Ý nghĩa của immutable (bất biến) trong String là gì?

Ý nghĩa của immutable là không thể sửa đổi khi đối tượng String đã được tạo ra

Có bao nhiêu cách để tạo ra một đối tượng String trong java

Có 2 cách:

- Sử dụng string literal (VD: String s = “hello”)
- Sử dụng từ khóa new (VD: String s = new String(“hello”))

Có bao nhiêu đối tượng được tạo ra trong đoạn code

String s = new String(“Hello”)

Có 2 đối tượng được tạo ra. Một đối tượng được tạo ra trong string constant pool và một đối tượng được tạo ra trong bộ nhớ heap bởi từ khóa new

Sự khác nhau giữa String và StringBuffer?

String là một đối tượng bất biến (immutable). StringBuffer là một đối tượng có thể biến đổi (mutable)

Sự khác nhau giữa StringBuffer và StringBuilder?

StringBuffer là đồng bộ còn StringBuilder là không đồng bộ

Làm thế nào để tạo lớp immutable trong Java

Định nghĩa lớp với từ khóa final

28. **Các câu hỏi phỏng vấn về Java Collection**
    Sự khác nhau giữa ArrayList và LinkedList

ArrayList	**LinkedList**
Sử dụng mảng động	Sử dụng danh sách liên kết doubly
Không hiệu quả với thao tác vì cần nhiều chuyển đổi	Hiệu quả cho thao tác
Tốt hơn cho lưu trữ và lấy dữ liệu	Tốt hơn để thao tác dữ liệu

SỰ khác nhau giữa List và Set

List có thể chứa các phần tử trùng lặp, trong khi Set chỉ chứa các phần tử duy nhất

Sự khác nhau giữa HashSet và TreeSet

HashSet không duy trì thứ tự nào, trong khi TreeSet duy trì thứ tự tăng dần

Sự khác nhau giữa Set và Map

Set chỉ chứa giá trị, trong khi Map chứa cặp key và value

29. Sự khác nhau giữa String, StringBuffer, StringBuilder

- **String**. 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, chỉ khác biệt tình huống sử dụng có liên quan đến đa luồng => Tốc độ xử lý. (StringBuilder > StringBuffer > String)