* Khuôn mẫu
* Interface vs Abstract class: cần hiểu về trừu tượng, kế thừa
  1. Trừu tượng.
     + Trong lập trình Java, trừu tượng (abstraction) là một khái niệm quan trọng trong việc thiết kế và phát triển các ứng dụng. Abstraction đề cập đến khả năng che giấu chi tiết triển khai của một đối tượng, chỉ hiển thị các tính năng hoặc hành vi chính của đối tượng đó.
     + Trong Java, abtraction được thể hiện thông qua hai cơ chế chính: abstract class và interface.
  2. Kế thừa.
     + Trong Java, kế thừa (inheritance) là một tính năng quan trọng trong việc thiết kế và phát triển ứng dụng. Kế thừa cho phép ta tái sử dụng các thuộc tính và phương thức của một lớp trong lớp khác, đồng thời cho phép ta mở rộng chức năng của lớp đó bằng cách thêm các thuộc tính và phương thức mới hoặc ghi đè các phương thức có sẵn.
  3. Abstract class.

A picture containing text, font, white, receipt

Description automatically generated

* + - Là một bản thiết kế cho các class khác dựa vào.
    - An *abstract class* is a class that is declared abstract—it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed.
    - Là bán trừu tượng.
    - Keyword mặc định của Abstract Class là “public” và “abstract”
    - Một class thường chỉ có thể extends duy nhất một Abstract class.
    - Một abstract class có thể implements nhiều interface. Các method của các Interface có thể được implements ngay trong Abstract Class, điều này giúp cho abstract class đĩnh nghĩa hành vi để sử dụng trên các class khác nhau, hoặc là không. Nếu không thì các subclass mà extends Abstract class đó phải implements các methods trong các Interfaces mà Abstract class đó đã implements.
    - Abstract class không thể được khởi tạo, nghĩa là không thể tạo ra một object từ class đó.
    - **Abstract class là một phân lớp, để cho các class khác có thể kế thừa nó. Các class kế thừa nó phải kế thừa tất cả các phương thức của nó (implements), hoặc các class đó cũng phải là Abstract Class (extends).**
    - Abstract class có thể có hoặc không có các abstract method. Các abstract method không có body phải được triển khai trong các class con implements Abstract class.
    - Abstract class có thể có các Constructor. Tuy nhiên chúng chỉ được gọi khi các class con được khởi tạo (Instantiation).
    - Abstract class in java can’t be instantiated.
    - If a class have abstract methods, then the class should also be abstract using abstract keyword, else it will not compile.
    - Abstract class chỉ có thể được khởi tạo thông qua class con extends nó.
    - Abstract class có thể có các trường và các method static. Chúng nó thể được sử dụng ở lớp con extends Abstract class này hoặc dùng trực tiếp.
    - Có thể lồng Abstract Class trong Abstract Class
    - Lợi ích của việc dung Abstract class: Cho phép định nghĩa các thuộc tính và hành vi chung chung của 1 nhóm các class. Điều này giúp giảm thiểu code trùng lặp trong nhiều class và giúp code nhìn đơn giản hơn.
  1. Interface



* + - Là một blueprint hay một contract để các class follow.
    - Nó định nghĩa một tập các phương thức trừu tượng và constants để các class khác triển khai.
    - Một class có thể implements nhiều Interface, kể cả abstract class.
    - Một Interface không thể extends được một Abstract class.
    - Một interface có thể extend một interface khác.
    - Một Interface có thể có nhiều phương thức trừu tượng, phương thức static, phương thức default, các trường constant.
    - Dùng để đạt được sự trừu tượng và đa kế thừa.
    - Tất cả phương thức trong interface mặc định là public và abstract.
    - Class mà implements interface phải triển khai tất cả các phương thức trong interface đó.
    - Các phương thức default trong một interface có một triển khai trong chính interface đó và có thể bị ghi đè bởi các lớp triển khai.
    - Các trường constant mặc định là static và final.
    - Nhiều class có thể implements 1 interface.
    - Có thể lồng interface trong interface.
    - An interface can be used as a type for variables, method parameters, and return types.
    - An interface can be used to achieve loose coupling in code by allowing classes to interact with each other through their interfaces, rather than their concrete implementations.
    - An interface can be used as a type for variables, method parameters, and return types.
    - Interfaces can’t have constructors because we can’t instantiate them and interfaces can’t have a method with body.
    - We can’t instantiate an interface in java.

A screenshot of a computer

Description automatically generated with low confidence

* 1. Tương tác giữa class với abstract class và interface.
     + Một class có thể extends một Abstract class và một hoặc nhiều Interface cùng một lúc, miễn là không xung đột với nhau.
     + Nếu Abstract class mà Class đang extends không implements các phương thức mà Abstract class đó implements từ các Interfaces thì các subclass mà extends Abstract class đó phải implements các methods trong các Interfaces mà Abstract class đó đã implements.
     + Một Interface có thể khai báo trong một Class.
* Overriding vs Overloading: cần hiểu về đa hình
  1. Đa hình
     + Trong sinh học, một loài vật có thể có nhiều hình dạng và trạng thái khác nhau.
     + Các class con của một class có thể định nghĩa hành vi riêng của chúng và chia sẻ một số chức năng cùng với class cha.
  2. Override
     + Override là một tính năng cho phép một lớp con cung cấp một triển khai cụ thể của phương thức đã được cung cấp bởi một trong các lớp cha của nó. Nói dễ hiểu hơn, nếu lớp con có một hoặc nhiều phương thức giống với một trong các lớp cha của nó, thì đó là ghi đè phương thức.
     + Override được sử dụng để thu được tính đa hình tại runtime.
     + @Override được từ cả class thường hay abstract class và interface, nhưng phải @Override bắt buộc với abstract class và interface.
     + Các quy tắc cho việc @Override: Danh sách tham số phải trùng nhau, kiểu giá trị trả về phải tương thích; Phương thức đè không được giảm quyền truy nhập so với phiên bản của lớp cha; Phải là quan hệ IS-A (kế thừa); Các phương thức **final**, **static**, **private** không thể cài đè.
  3. Overload
     + Nạp chồng phương thức đơn giản là có vài phương thức trùng tên nhưng khác nhau về đối số trong cùng 1 class. Cài chồng phương thức cho phép ta tạo nhiều phiên bản của một phương thức, mỗi phiên bản chấp nhận một danh sách đối số khác nhau, nhằm tạo thuận lợi cho việc gọi phương thức.
     + Nạp chồng phương thức được sử dụng để thu được tinh đa hình tại compile time.
     + Nạp chồng phương thức khởi tạo:

+ Xét trường hợp ta có các hàm khởi tạo chồng với hoạt động khởi tạo giống nhau và chỉ khác nhau ở phần xử lý các kiểu đối số. Ta sẽ không muốn chép đi chép lại phần mã khởi tạo mà các hàm khởi tạo đều có.

+ Cách giải quyết: ta sẽ đặt toàn bộ phần mã đó vào chỉ một trong các hàm khởi tạo. Và ta muốn rằng hàm khởi tạo nào cũng đều gọi đến hàm khởi tạo kia để nó hoàn thành công việc khởi tạo. Để làm việc đó, ta dùng this() để gọi một hàm khởi tạo từ bên trong một hàm khởi tạo khác của cùng một lớp.

* String

1. Mutable vs Immutable, String vs String Buffer vs String Builder
   * + Mutable:

+ Mutable Object: khi khởi tạo 1 đối tượng, tức ta có 1 tham chiếu tới 1 thể hiện của 1 lớp, thì trạng thái của đối tượng có thể thay đổi được sau khi việc khởi tạo đối tượng thành công. (Trạng thái đối tượng ở đây có thể là các trường thông tin mà đối tượng đó nắm giữ. Ví dụ: tên, tuổi của 1 đối tượng sinh viên chẳng hạn)

* + - Immutable:

+ Immutable object: khi khởi tạo 1 đối tượng, thì trạng thái của tối tượng đó không thể thay đổi được sau khi việc khởi tạo đối tượng thành công. (Điều này có nghĩa là, bạn chỉ có thể get mà không thể set).

* + - String, String buffer vs String Builder

+  Trong Java, String là Immutable còn StringBuffer và StringBuilder là mutable.

+ String: Là một Immutable class. Một khi object String được tạo, giá trị của nó sẽ không thay đổi. Bất kì hoạt động hay toán tử nào lên String sẽ tạo một object String mới. Điều này làm cho String là “thread-safe” và có thể dự đoán được, tuy nhiên nó có thể đưa ra kết quả chậm nếu như nối hoặc xử lý một chuỗi dài.

+ StringBuffer: Là một Mutable class, được sử dụng để tạo và xử lý String. StringBuffer có thể được sửa đổi sau khi tạo ra, khiến cho nó hiệu quả và linh hoạt khi xử lý với các chuỗi lớn. StringBuffer là “thread-safe” và đồng bộ, khiến cho việc nhiều luồng truy cập và xử lý nó trở nên an toàn. Tuy nhiên, điều đó có thể tạo ra kết quả chậm hơn so với xử lý đơn luồng.

+ StringBuilder: Giống với StringBuffer, nhưng nó không đồng bộ, nghĩa là nó nhanh hơn khi xử lý đơn luồng. Nhưng vì StringBuilder không phải “thread-safe”, nên không nên được sử dụng trong các ứng dụng đa luồng.

* + - String pool:

+ String pool là một vùng nhớ đặc biệt nằm trong vùng nhớ Heap (Heap memory), dùng để lưu trữ các biến được khai báo theo kiểu String.

+ String pool giúp tối ưu hoá việc lưu trữ và sử dụng vùng nhớ khi khai báo biến String, giúp hạn chế tình trạng tràn bộ nhớ Java Heap Space.

A diagram of a string pool

Description automatically generated with medium confidence

* Class:
  + - Nested Class
    - Local Class
    - Anonymous Classes
    - Lambda Expressions