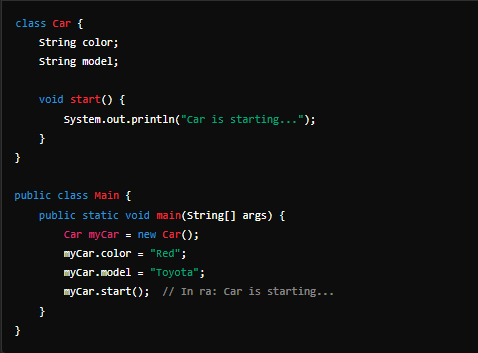
# Object Oriented Programming

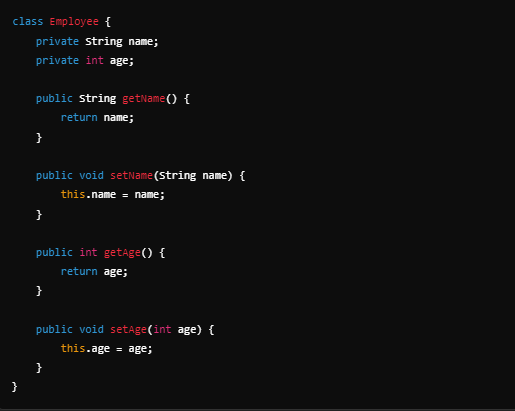
### 1. **Lớp (Class) và Đối tượng (Object)**

* **Lớp (Class)** là một bản thiết kế mô tả các đặc điểm và hành vi chung của đối tượng. Một lớp bao gồm các thuộc tính (biến) và phương thức (hàm).
* **Đối tượng (Object)** là một thể hiện của lớp. Khi lớp được định nghĩa, ta có thể tạo ra nhiều đối tượng từ lớp đó.
* Ví dụ:



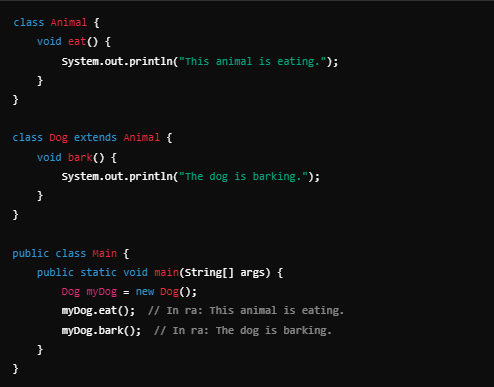
### 2. **Đóng gói (Encapsulation)**

* Đóng gói là việc nhóm các thuộc tính và phương thức có liên quan vào trong một lớp. Các thuộc tính này thường được ẩn (private) và chỉ có thể truy cập thông qua các phương thức getter và setter.
* Điều này giúp bảo vệ dữ liệu khỏi việc bị sửa đổi trực tiếp từ bên ngoài và tạo ra một giao diện rõ ràng để tương tác với lớp.
* Ví dụ:



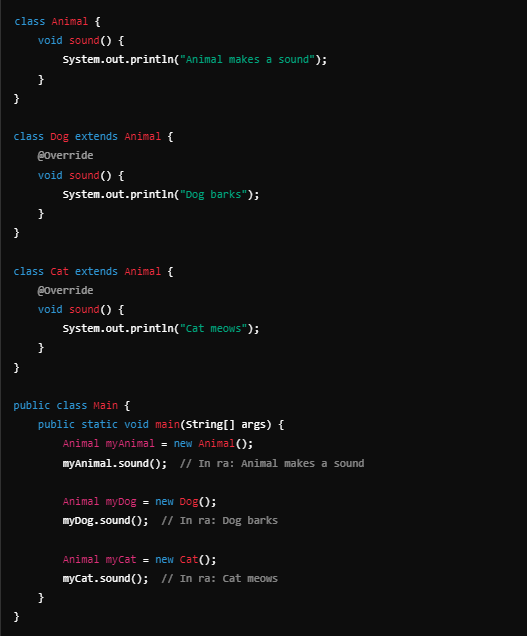
### 3. **Kế thừa (Inheritance)**

* Kế thừa cho phép một lớp con (subclass) kế thừa các đặc điểm và hành vi từ lớp cha (superclass). Điều này giúp tái sử dụng mã nguồn và dễ dàng mở rộng các chức năng.
* Lớp con có thể ghi đè (override) các phương thức của lớp cha để thay đổi hành vi.
* Ví dụ:



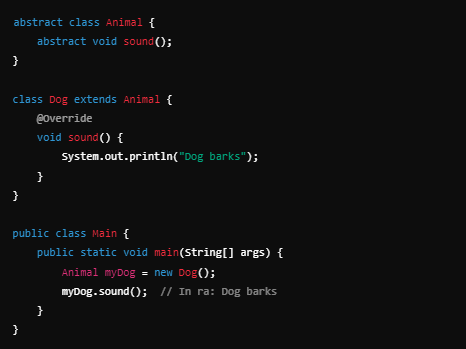
### 4. **Đa hình (Polymorphism)**

* Đa hình cho phép các đối tượng khác nhau có thể có hành vi khác nhau khi gọi cùng một phương thức. Đa hình có thể được thực hiện thông qua:
  + **Nạp chồng phương thức (Method Overloading):** Khi cùng tên phương thức nhưng khác tham số.
  + **Ghi đè phương thức (Method Overriding):** Khi lớp con ghi đè lại phương thức của lớp cha.
* Ví dụ:



### 5. **Trừu tượng (Abstraction)**

* Trừu tượng là việc ẩn chi tiết cài đặt và chỉ cung cấp các phương thức cần thiết cho người dùng. Điều này được thực hiện thông qua các lớp trừu tượng (abstract classes) hoặc giao diện (interfaces).
* Các lớp trừu tượng không thể tạo đối tượng trực tiếp, nhưng các lớp con phải cài đặt các phương thức trừu tượng.
* Ví dụ:



# Cấu trúc dữ liệu và giải thuật

## A.Cấu trúc dữ liệu

### 1.Mảng (Array)

Mảng là một cấu trúc dữ liệu cơ bản, trong đó các phần tử có cùng kiểu dữ liệu được lưu trữ liên tiếp trong bộ nhớ. Mảng có chỉ số (index) giúp truy xuất các phần tử nhanh chóng.

* **Ưu điểm:**
  + Truy cập ngẫu nhiên (random access) tới bất kỳ phần tử nào.
  + Quản lý bộ nhớ đơn giản và hiệu quả.
* **Nhược điểm:**
  + Kích thước mảng phải được xác định trước khi sử dụng (cứng nhắc).
  + Chèn hoặc xóa phần tử vào giữa mảng có thể tốn thời gian (O(n)).
* **Ứng dụng:** Dùng trong việc lưu trữ dữ liệu liên tục như danh sách các số, bảng dữ liệu.

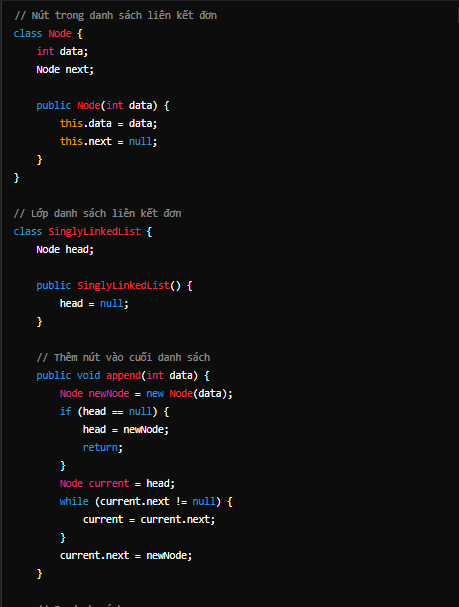
### 2.Singly Linked List

1. Danh sách liên kết đơn là một tập hợp các phần tử, được gọi là nút (node), được liên kết với nhau theo một thứ tự tuyến tính. Mỗi nút chứa hai thành phần:

* **Dữ liệu (Data)**: Lưu trữ giá trị của phần tử.
* **Con trỏ (Next)**: Lưu trữ địa chỉ của nút tiếp theo trong danh sách.

Nút cuối cùng trong danh sách có con trỏ Next trỏ đến NULL (hoặc null trong Java), biểu thị kết thúc danh sách.

* **Ưu điểm**:
  + Kích thước động: Có thể dễ dàng thêm hoặc xóa nút mà không cần phải xác định kích thước ban đầu.
  + Tiết kiệm bộ nhớ: Chỉ sử dụng bộ nhớ cần thiết cho các nút hiện có.
  + Chèn và xóa hiệu quả: Việc chèn và xóa nút ở giữa danh sách có độ phức tạp O(1) nếu đã biết vị trí.
* **Nhược điểm**:
  + Truy cập ngẫu nhiên chậm: Để truy cập một nút ở vị trí bất kỳ, cần phải duyệt từ đầu danh sách (độ phức tạp O(n)).
  + Tốn thêm bộ nhớ cho con trỏ: Mỗi nút cần thêm bộ nhớ để lưu trữ con trỏ.



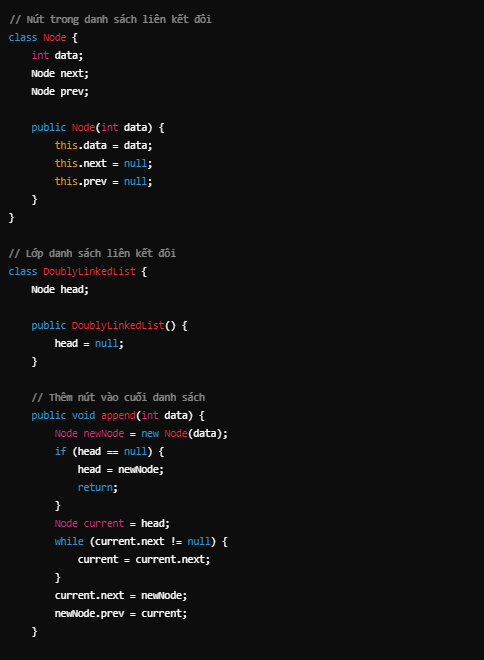
### 3.Doubly Linked List

Danh sách liên kết đôi tương tự như danh sách liên kết đơn, nhưng mỗi nút trong danh sách liên kết đôi chứa ba thành phần:

* **Dữ liệu (Data)**: Lưu trữ giá trị của phần tử.
* **Con trỏ Next**: Trỏ đến nút tiếp theo trong danh sách.
* **Con trỏ Prev (Previous)**: Trỏ đến nút trước nút hiện tại.

Nút đầu tiên trong danh sách có con trỏ Prev là NULL, và nút cuối cùng có con trỏ Next là NULL.

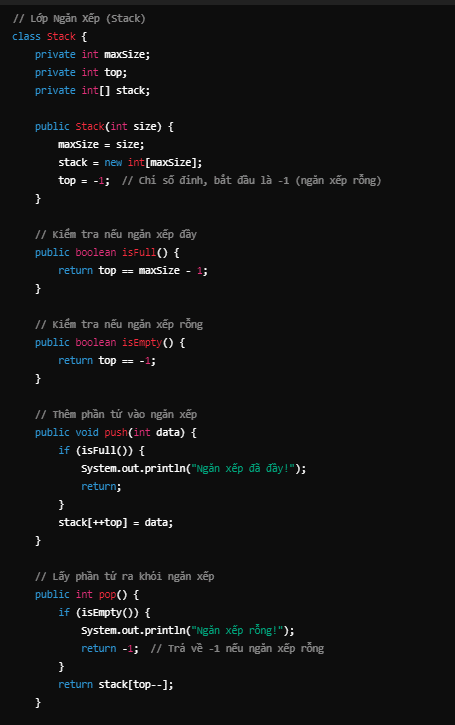
* **Ưu điểm**:
  + Duyệt theo cả hai chiều: Cho phép duyệt danh sách theo cả chiều xuôi và chiều ngược, giúp thực hiện một số thao tác hiệu quả hơn.
  + Xóa nút dễ dàng hơn: Không cần phải duyệt tìm nút trước nút cần xóa nếu đã có con trỏ đến nút đó.
* **Nhược điểm**:
  + Tốn thêm bộ nhớ: Mỗi nút cần thêm một con trỏ Prev, do đó tốn nhiều bộ nhớ hơn so với danh sách liên kết đơn.
  + Các thao tác phức tạp hơn: Việc thêm và xóa nút phức tạp hơn một chút do cần cập nhật cả hai con trỏ.



### 4.Stack

Ngăn xếp là một cấu trúc dữ liệu theo nguyên lý "LIFO" (Last In, First Out). Phần tử được thêm vào và loại bỏ từ cùng một đầu gọi là "đỉnh" của ngăn xếp.

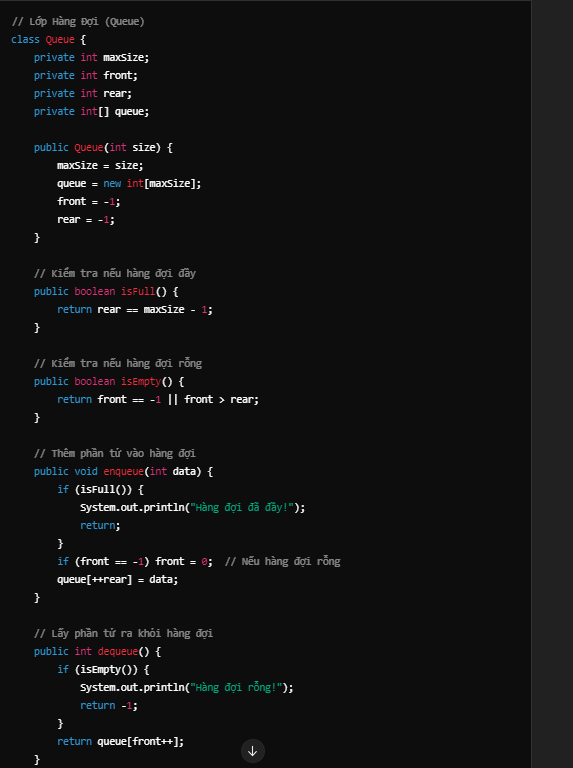
* **Ưu điểm:**
  + Dễ dàng thực hiện các thao tác đẩy (push) và lấy (pop) phần tử.
  + Tốt cho việc lưu trữ và truy xuất các phần tử theo thứ tự ngược lại.
* **Nhược điểm:**
  + Không thể truy xuất phần tử ngoài cùng (top) mà không phải loại bỏ phần tử khác.
* **Ứng dụng:**
  + Kiểm tra biểu thức toán học (trong các biểu thức dạng hậu tố, tiền tố).
  + Thực thi các cuộc gọi hàm trong ngôn ngữ lập trình.
  + Duyệt đồ thị (DFS - Depth First Search).



### 5.Queue

Hàng đợi là một cấu trúc dữ liệu theo nguyên lý "FIFO" (First In, First Out). Phần tử được thêm vào cuối hàng đợi và loại bỏ từ đầu hàng đợi.

* **Ưu điểm:**
  + Tốt cho các tình huống yêu cầu xử lý theo thứ tự (như xử lý các tác vụ theo thứ tự).
* **Nhược điểm:**
  + Không thể truy xuất trực tiếp các phần tử ngoài cùng mà không loại bỏ phần tử khác.
* **Ứng dụng:**
  + Xử lý các tác vụ trong hệ thống, chẳng hạn như quản lý tác vụ trong hệ điều hành.
  + Cây quyết định trong mạng máy tính.



6.Cấu trúc dữ liệu cây  
Cây nhị phân (**Binary Tree**) là một cấu trúc dữ liệu dạng cây, trong đó mỗi nút có tối đa **hai nút con**:

* **Nút con trái (Left Child)**
* **Nút con phải (Right Child)**

Cây nhị phân thường được sử dụng để biểu diễn dữ liệu phân cấp, tìm kiếm, và thực hiện các thuật toán hiệu quả.

+Duyệt trung thứ tự trong cây nhị phân

-Trong cách duyệt này, cây con bên trái được truy cập đầu tiên, sau đó là nút gốc và sau đó là cây con bên phải. Bạn nên luôn luôn ghi nhớ rằng mỗi nút đều có thể biểu diễn một cây con.

-Nếu một cây nhị phân được duyệt trung thứ tự, kết quả tạo ra sẽ là các giá trị khóa được sắp xếp theo thứ tự tăng dần.



***D → B → E → A → F → C → G***

+Duyệt tiền thứ tự trong cây nhị phân

Trong cách thức duyệt tiền thứ tự trong cây nhị phân, nút gốc được duyệt đầu tiên, sau đó sẽ duyệt cây con bên trái và cuối cùng sẽ duyệt cây con bên phải.



***A → B → D → E → C → F → G***

+Duyệt hậu thứ tự trong cây nhị phân

Trong cách thức duyệt hậu thứ tự trong cây nhị phân, nút gốc của cây sẽ được truy cập cuối cùng, do đó bạn cần chú ý. Đầu tiên, chúng ta duyệt cây con bên trái, sau đó sẽ duyệt cây con bên phải và cuối cùng là duyệt nút gốc.



***D → E → B → F → G → C → A***

### 7.Cây tìm kiếm nhị phân Cây tìm kiếm nhị phân (BST) là một tập hợp bao gồm các nút được sắp xếp theo cách để chúng có thể duy trì hoặc tuân theo các đặc điểm của cây tìm kiếm nhị phân. Mỗi một nút thì đều có một khóa và giá trị liên kết với nó. Trong khi tìm kiếm, khóa cần tìm được so sánh với các khóa trong cây tìm kiếm nhị phân (BST) và nếu tìm thấy, giá trị liên kết sẽ được thu nhận.

-Ví dụ chèn :  


### 8.Cây AVL

Cây AVL là một loại cây nhị phân tìm kiếm (**Binary Search Tree**) được cân bằng về chiều cao (**Height-Balanced Binary Search Tree**).  
Nó đảm bảo rằng sự chênh lệch chiều cao giữa hai nhánh con của bất kỳ nút nào không vượt quá 1.

Cây AVL được đặt tên theo hai nhà khoa học người Nga **Adelson-Velsky** và **Landis**, những người đã phát minh ra nó vào năm 1962.