- Bthg ở cùng class thì ông sẽ dùng this. Để truy xuất đến biến nhưng khi ra class khác (or cha private) thì phải tạo đối tg r trỏ tới hàm set get

- lớp abstract thì sẽ k new đc đối tượng, phg thức abs thì chỉ có cái tên hàm thôi để các lớp con override lại,

cha obj= new con();// khi viết thì lấy thuộc tính vs method trong Class Cha, khi chạy(runtime) thì lấy trong Class Con,

- interface để đa kế thừa và luôn là public abstract, k có biến ( attribute) hay hàm (method) bình thường, chí có hàm abstract (trừu tượng) và hằng số, cũng k new đc đối tượng

\* Interface vs class là implement(khác loại) , Interface vs interface là extends(cùng loại)

* try (BufferedReader br = new BufferedReader(new FileReader("nhanvien.txt")))

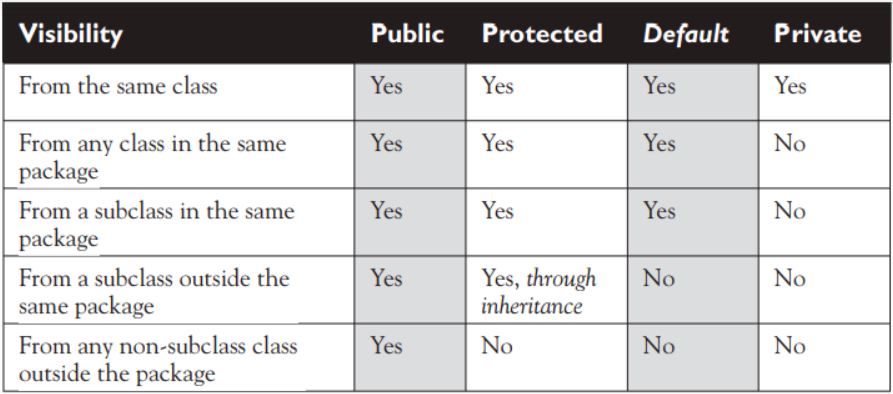
để tự động đóng file luôn

* try { lệnh}

catch { nếu lệnh trên lỗi thì thực hiện// thường in ra lỗi}

finally{ luôn thực hiện lệnh này}

\* biến static là biến của đtượng(class) nên new các đối tượng sẽ dùng chung biến static



**4 tính chất của lập trình hướng đối tượng trong Java**

**Tính đóng gói (encapsulation) và che giấu thông tin (information hiding)** :

* Tức là trạng thái của đối tượng được bảo vệ không cho các truy cập từ code bên ngoài như thay đổi trong thái hay nhìn trực tiếp. Việc cho phép môi trường bên ngoài tác động lên các dữ liệu nội tại của một đối tượng theo cách nào là hoàn toàn tùy thuộc vào người viết mã. Đây là tính chất đảm bảo sự toàn vẹn, bảo mật của đối tượng Trong Java, tính đóng gói được thể hiện thông qua phạm vi truy cập (access modifier). Ngoài ra, các lớp liên quan đến nhau có thể được gom chung lại thành package.

**Tính kế thừa** là khả năng cho phép ta xây dựng một lớp mới dựa trên các định nghĩa của một lớp đã có. Lớp đã có gọi là lớp Cha, lớp mới phát sinh gọi là lớp Con và đương nhiên kế thừa tất cả các thành phần của lớp Cha, có thể chia sẻ hay mở rộng các đặc tính sẵn có mà không phải tiến hành định nghĩa lại.

**Tính đa hình (polymorphism)**:

* Khi một tác vụ được thực hiện theo nhiều cách khác nhau được gọi là tính đa hình.
* Đối với tính chất này, nó được thể hiện rõ nhất qua việc gọi phương thức của đối tượng. Các phương thức hoàn toàn có thể giống nhau, nhưng việc xử lý luồng có thể khác nhau. Nói cách khác: Tính đa hình cung cấp khả năng cho phép người lập trình gọi trước một phương thức của đối tượng, tuy chưa xác định đối tượng có phương thức muốn gọi hay không. Đến khi thực hiện (run-time), chương trình mới xác định được đối tượng và gọi phương thức tương ứng của đối tượng đó. Kết nối trễ giúp chương trình được uyển chuyển hơn, chỉ yêu cầu đối tượng cung cấp đúng phương thức cần thiết là đủ.
* Trong Java, chúng ta sử dụng nạp chồng phương thức (method overloading) và ghi đè phương thức (method overriding) để có tính đa hình.
* **Nạp chồng (Overloading)**: Đây là khả năng cho phép một lớp có nhiều thuộc tính, phương thức cùng tên nhưng với các tham số khác nhau về loại cũng như về số lượng. Khi được gọi, dựa vào tham số truyền vào, phương thức tương ứng sẽ được thực hiện.
* **Ghi đè (Overriding)**: là hai phương thức cùng tên, cùng tham số, cùng kiểu trả về nhưng thằng con viết lại và dùng theo cách của nó, và xuất hiện ở lớp cha và tiếp tục xuất hiện ở lớp con. Khi dùng override, lúc thực thi, nếu lớp Con không có phương thức riêng, phương thức của lớp Cha sẽ được gọi, ngược lại nếu có, phương thức của lớp Con được gọi.

**Tính trừu tượng (abstraction)**:

* Tính trừu tượng là một tiến trình ẩn các chi tiết trình triển khai và chỉ hiển thị tính năng tới người dùng. Tính trừu tượng cho phép bạn loại bỏ tính chất phức tạp của đối tượng bằng cách chỉ đưa ra các thuộc tính và phương thức cần thiết của đối tượng trong lập trình.
* Tính trừu tượng giúp bạn tập trung vào những cốt lõi cần thiết của đối tượng thay vì quan tâm đến cách nó thực hiện.
* Trong Java, chúng là sử dụng abstract class và abstract interface để có tính trừu tượng.

## Sự khác nhau giữa lập trình hướng đối tượng với lập trình cấu trúc (OOP vs POP)

* **Định nghĩa:** OOP tập trung vào dữ liệu hơn là thuật toán còn POP tập trung vào việc xây dựng các hàm, thuật toán hơn là dữ liệu.
* **Chương trình:** OOP chia chương trình thành các đối tượng còn POP chia chương trình thành các hàm.
* **Khả năng truy cập:** OOP chia ra các modifier: Private, Public, Protected, Default còn POP thì không.
* **Thực thi:** với OOP các chức năng có thể chạy đồng thời trong khi POP thì các hàm, chức năng chạy lần lượt.
* **Điều khiển dữ liệu:**với OOP thì dữ liệu và hàm của một đối tượng giống như một thành phần riêng biệt và bị hạn chế truy cập bởi các đối tượng khác. Với POP, dữ liệu có thể truy cập một cách tự do giữa các hàm.
* **Bảo mật:** OOP bảo mật hơn POP nhờ việc giới hạn truy cập dữ liệu.
* **Chỉnh sửa:** với OOP dữ liệu có thể thêm mới một cách dễ dàng từ các đối tượng trong khi với POP thì rất khó.

## Lập trình hướng đối tượng là gì?

* Lập trình hướng đối tượng (Object Oriented Programming – OOP): là một kỹ thuật lập trình, trong đó chương trình sẽ được chia ra thành các phần nhỏ được gọi là đối tượng (Object).
* Đối tượng trong lập trình hướng đối tượng sẽ tương ứng với các thực thể trong bài toán, nó sẽ có các thuộc tính (attribute), các hành động (method).
* Các đối tượng có thể tương tác với nhau.
* Các tính chất cơ bản của lập trình hướng đối tượng: tính trừu tượng, tính đóng gói, tính đa hình, tính kế thừa
* Lập trình hướng đối tượng được đánh giá là dễ học, năng suất, đơn giản, dễ bảo trì, dễ mở rộng…

## Lập trình hướng cấu trúc là gì?

* Lập trình hướng cấu trúc hay còn gọi là lập trình hướng thủ tục (Procedure Oriented Programming – POP): là một kỹ thuật lập trình, trong đó chương trình được chia thành các hàm (chương trình con)
* Mỗi chương trình còn có thể được chia ra nhiều chương trình con khác để đơn giản hóa công việc của chúng. (Quá trình làm mịn)

– Ví dụ : nhập và hiện thị chia thành 2 hàm, có thể chia nhỏ tiếp nếu nó phức tạp

* Trong lập trình hướng cấu trúc ta thường quan tâm đến việc phát triển các hàm mà ít quan tâm tới dữ liệu – thứ mà chúng dùng để xử lý công việc. (Điều này khiến cho dữ liệu khó kiểm soát)
* Để liên kết giữa các hàm với nhau ta thường dùng biến toàn cục hoặc con trỏ.
* Các tính chất cơ bản của lập trình hướng cấu trúc là:

– Tập chung vào công việc cần thực hiện (thuật toán)

– Chương trình lớn được chia thành các hàm nhỏ hơn

– Phần lớn các hàm sử dụng dữ liệu chung

– Dữ liêu trong hệ thống được chuyển động từ hàm này sang hàm khác.

– Hàm biến đổi dữ liệu từ dạng này sang dạng khác

– Sử dụng cách tiếp cận top-down trong thiết kế chương trình

|  |  |
| --- | --- |
| **Constructor** | **Phương thức** |
| Constructor được sử dụng để khởi tạo trạng thái của một đối tượng. | Phương thức được sử dụng để thể hiện hành động của một đối tượng. |
| Constructor không có kiểu trả về. | Phương thức có kiểu trả về. |
| Constructor được gọi ngầm. | Phương thức được gọi tường minh. |
| Trình biên dịch Java tạo ra constructor mặc định nếu bạn không có constructor nào. | Phương thức không được tạo ra bởi trình biên dịch Java. |
| Tên của constructor phải giống tên lớp. | Tên phương thức có thể giống hoặc khác tên lớp. |

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Phương thức** | **Mô tả** | |
| 1 | [char charAt(int index)](https://viettuts.vn/java-string/phuong-thuc-charat) | Trả về giá trị char cho chỉ số cụ thể. | |
| 2 | [int length()](https://viettuts.vn/java-string/phuong-thuc-length) | Trả về độ dài chuỗi. | |
| 3 | [static String format(String format, Object… args)](https://viettuts.vn/java-string/phuong-thuc-format) | Trả về chuỗi được format. | |
| 4 | [static String format(Locale l, String format, Object… args)](https://viettuts.vn/java-string/phuong-thuc-format) | Trả về chuỗi được format theo vùng miền(quốc gia). | |
| 5 | [String substring(int beginIndex)](https://viettuts.vn/java-string/phuong-thuc-substring) | Trả về chuỗi con bắt đầu từ chỉ số index. | |
| 6 | [String substring(int beginIndex, int endIndex)](https://viettuts.vn/java-string/phuong-thuc-substring) | Trả về chuỗi con từ chỉ số bắt đầu đến chỉ số kết thúc. | |
| 7 | [boolean contains(CharSequence s)](https://viettuts.vn/java-string/phuong-thuc-contains) | Kiểm tra chuỗi chứa chuối không, kết quả trả về là giá trị boolean. | |
| 8 | [static String join(CharSequence delimiter, CharSequence… elements)](https://viettuts.vn/java-string/phuong-thuc-join) | Trả về chuỗi được nối từ nhiều chuỗi. | |
| 9 | [static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)](https://viettuts.vn/java-string/phuong-thuc-join) | Trả về chuỗi được nối từ nhiều chuỗi. | |
| 10 | [boolean equals(Object another)](https://viettuts.vn/java-string/phuong-thuc-equals) | kiểm tra sự tương đương của chuỗi với đối tượng. | |
| 11 | [boolean isEmpty()](https://viettuts.vn/java-string/phuong-thuc-isempty) | Kiểm tra chuỗi rỗng. | |
| 12 | [String concat(String str)](https://viettuts.vn/java-string/phuong-thuc-concat) | Nối chuỗi cụ thể. | |
| 13 | [String replace(char old, char new)](https://viettuts.vn/java-string/phuong-thuc-replace) | Thay thế tất cả giá trị char cụ thể bằng một giá trị char mới. | |
| 14 | [String replace(CharSequence old, CharSequence new)](https://viettuts.vn/java-string/phuong-thuc-replace) | Thay thế tất cả các chuỗi bằng một chuỗi mới. | |
| 15 | [static String equalsIgnoreCase(String another)](https://viettuts.vn/java-string/phuong-thuc-equalsignorecase) | So sanh chuỗi, không phân biệt chữ hoa hay chữ thường. | |
| 16 | [String[] split(String regex)](https://viettuts.vn/java-string/phuong-thuc-split) | Trả về mảng các chuỗi được tách ra theo giá trị regex. | |
| 17 | [String[] split(String regex, int limit)](https://viettuts.vn/java-string/phuong-thuc-split) | Trả về mảng các chuỗi được tách ra theo giá trị regex và có giới hạn. | |
| 18 | [String intern()](https://viettuts.vn/java-string/phuong-thuc-intern) | Trả về chuỗi interned. | |
| 19 | [int indexOf(int ch)](https://viettuts.vn/java-string/phuong-thuc-indexof) | Trả về vị trí của ký tự ch cụ thể. | |
| 20 | [int indexOf(int ch, int fromIndex)](https://viettuts.vn/java-string/phuong-thuc-indexof) | Trả về vị trí của ký tự ch tính từ từ vị trí fromIndex. | |
| 21 | [int indexOf(String substring)](https://viettuts.vn/java-string/phuong-thuc-indexof) | Trả về vị trí của chuỗi con substring. | |
| 22 | [int indexOf(String substring, int fromIndex)](https://viettuts.vn/java-string/phuong-thuc-indexof) | Trả về chuỗi con bắt đầu từ vị trí substring đến vị trí fromIndex. | |
| 23 | [String toLowerCase()](https://viettuts.vn/java-string/phuong-thuc-tolowercase) | Trả về chuỗi chữ thường. | |
| 24 | [String toLowerCase(Locale l)](https://viettuts.vn/java-string/phuong-thuc-tolowercase) | Trả về chuỗi chữ thường bằng việc sử dụng locale cụ thể. | |
| 25 | [String toUpperCase()](https://viettuts.vn/java-string/phuong-thuc-touppercase) | Trả về chuỗi chữ hoa. | |
| 26 | [String toUpperCase(Locale l)](https://viettuts.vn/java-string/phuong-thuc-touppercase) | Trả về chuỗi chữ hoa bằng việc sử dụng locale cụ thể. | |
| 27 [String trim()](https://viettuts.vn/java-string/phuong-thuc-trim) | Xóa khoảng trẳng ở đầu và cuối của chuỗi. |  |
| 28 | [static String valueOf(int value)](https://viettuts.vn/java-string/phuong-thuc-valueof) | Chuyển đổi giá trị kiểu dữ liệu đã cho thành chuỗi. | |

**ArrayList**

|  |  |  |  |
| --- | --- | --- | --- |
| **Phương thức** | | **Mô tả** | |
| boolean add(Object o) | | Nó được sử dụng để nối thêm phần tử được chỉ định vào cuối một danh sách. | |
| void add(int index, Object element) | | Nó được sử dụng để chèn các phần tử được chỉ định tại các chỉ số vị trí quy định trong một danh sách. | |
| boolean addAll(Collection c) | | Nó được sử dụng để nối tất cả các phần tử trong collection được chỉ định vào cuối của danh sách này, theo thứ tự chúng được trả về bởi bộ lặp iterator. | |
| boolean addAll(int index, Collection c) | | Nó được sử dụng để chèn tất cả các phần tử trong collection được chỉ định vào danh sách này, bắt đầu từ vị trí đã chỉ định. | |
| void retainAll(Collection c) | | Nó được sử dụng để xóa những phần tử không thuộc collection c và không thuộc list hiện tại khỏi list hiện tại. | |
| void removeAll(Collection c) | | Nó được sử dụng để xóa những phần tử thuộc collection c và thuộc list hiện tại khỏi list hiện tại. | |
| int indexOf(Object o) | | Nó được sử dụng để trả về chỉ mục trong danh sách với sự xuất hiện đầu tiên của phần tử được chỉ định, hoặc -1 nếu danh sách không chứa phần tử này. | |
| int lastIndexOf(Object o) | | Nó được sử dụng để trả về chỉ mục trong danh sách với sự xuất hiện cuối cùng của phần tử được chỉ định, hoặc -1 nếu danh sách không chứa phần tử này. | |
| Object[] toArray() | | Nó được sử dụng để trả về một mảng chứa tất cả các phần tử trong danh sách này theo đúng thứ tự. | |
| Object[] toArray(Object[] a) | | Nó được sử dụng để trả về một mảng chứa tất cả các phần tử trong danh sách này theo đúng thứ tự. | |
| Object clone() | | Nó được sử dụng để trả về một bản sao của ArrayList. | |
| void clear() | | Nó được sử dụng để xóa tất cả các phần tử từ danh sách này. | |
| void trimToSize() | | Nó được sử dụng để cắt dung lượng của thể hiện ArrayList này là kích thước danh sách hiện tại. | |
| // Creating arraylist | |

|  |
| --- |
| ArrayList<String> list = new ArrayList<String>(); |

|  |
| --- |
| // Add objects to arraylist |

|  |
| --- |
| list.add("Java"); |

|  |
| --- |
| list.add("C++"); |

|  |
| --- |
| list.add("PHP"); |

|  |
| --- |
| list.add("Java"); |

|  |
| --- |
| // Show list through Iterator |

|  |
| --- |
| Iterator<String> itr = list.iterator(); |

|  |
| --- |
| while (itr.hasNext()) { |

|  |
| --- |
| System.out.print(itr.next() + ", "); |

|  |
| --- |
| } |

|  |
| --- |
| // Show list through for-each |

|  |
| --- |
| System.out.println(); |

|  |
| --- |
| for (String obj : list) { |

|  |
| --- |
| System.out.print(obj + ", "); |

|  |
| --- |
| } |

|  |
| --- |
| // Show list through index |

|  |
| --- |
| System.out.println(); |

|  |
| --- |
| int size = list.size(); |

|  |
| --- |
| for (int i = 0; i < size; i++) { |

|  |
| --- |
| System.out.print(list.get(i) + ", "); |

|  |
| --- |
| } |
| // Create list ----------------------------------------- |

|  |
| --- |
| ArrayList<String> list = new ArrayList<String>(); |

|  |
| --- |
| // Add objects to list |

|  |
| --- |
| list.add("Java"); |

|  |
| --- |
| list.add("C++"); |

|  |
| --- |
| list.add("PHP"); |

|  |
| --- |
| list.add("Java"); |

|  |
| --- |
|  |

|  |
| --- |
| // addAll() |

|  |
| --- |
| ArrayList<String> listA = new ArrayList<String>(); |

|  |
| --- |
| listA.addAll(list); |

|  |
| --- |
| System.out.print("listA:"); |

|  |
| --- |
| showList(listA); |

|  |
| --- |
|  |

|  |
| --- |
| // retainAll() listA |

|  |
| --- |
| ArrayList<String> listB = new ArrayList<String>(); |

|  |
| --- |
| listB.add("Java"); |

|  |
| --- |
| listA.retainAll(listB); |

|  |
| --- |
| System.out.print("listA:"); |

|  |
| --- |
| showList(listA); |

|  |
| --- |
|  |

|  |
| --- |
| // removeAll() listA |

|  |
| --- |
| list.removeAll(listA); |

|  |
| --- |
| System.out.print("listA:"); |

|  |
| --- |
| showList(list); |
| // define enum with 3 variable------------------- |

|  |
| --- |
| PAGE\_001("PAGE\_001", "Login", "/login"), |

|  |
| --- |
| PAGE\_002("PAGE\_002", "Add new user", "/addUser"), |

|  |
| --- |
| PAGE\_003("PAGE\_003", "List user", "/listUser"); |

|  |
| --- |
|  |

|  |
| --- |
| private String pageID; //constructor |

|  |
| --- |
| private String pageTitle; |

|  |
| --- |
| private String pageURL; |

|  |
| --- |
|  |

|  |
| --- |
| private PageInfo(String pageID, String pageTitle, String pageURL) { |

|  |
| --- |
| this.pageID = pageID; |

|  |
| --- |
| this.pageTitle = pageTitle; |

|  |
| --- |
| this.pageURL = pageURL; |

|  |
| --- |
| } |

|  |
| --- |
| public static void main(String[] args) { |

|  |
| --- |
| System.out.println("pageID: " + PageInfo.PAGE\_001.getPageID()); |

|  |
| --- |
| System.out.println("pageTitle: " + PageInfo.PAGE\_001.getPageTitle()); |

|  |
| --- |
| System.out.println("pageURL: " + PageInfo.PAGE\_001.getPageURL()); |

|  |
| --- |
| } |