1. **JAVA là gì?**

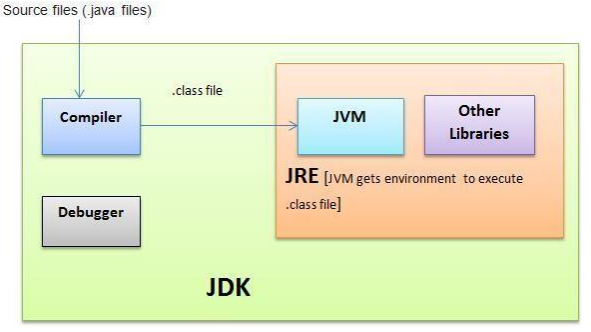
* Là ngôn ngữ lập trình hướng đối tượng, chạy được trên nhiều nền tảng khác nhau (gồm JE, EE, ME)

1. **JDK, JRE, JVM**

**JDK:** gồm Complier, Debugger, JRE

* **Complier :** chuyển đổi mã nguồn từ file java sang class
* **JRE:** gồm JVM và Other Libraries

**+ JVM:** thực thi file class được chuyển đổi từ Complier



1. **Thông dịch, biên dịch khác nhau như thế nào ?**

* **Biên dịch:** thực hiện **biên dịch từ source code sang byte code** (được thực thi ở Complier)
* **Thông dịch:** thực hiện **thông dịch từ byte code sang ngôn ngữ máy** ( được thực thi ở JVM)

1. **JAVA là ngôn ngữ thông dịch hay biên dịch ?**

* Java là ngôn ngữ vừa **thông dịch và biên dịch**

Các kiểu dữ liệu trong java, và giá trị mặc định khi khai báo. Từ khóa static dùng để làm gì ? Ràng buộc khi sử dụng static ?

* Có 2 kiểu dữ liệu là **kiểu nguyên thuỷ** (Primitive) và **kiểu đối tượng**
* **Kiểu nguyên thuỷ** (Primitive) gồm

+ **byte** (mặc định 0) : kiểu sô nguyên (1 byte = 8 bit) từ -128 đến 127

+ **short** (mặc định 0): kiểu sô nguyên (2 byte) từ -216 – 1 đến 216 – 1 – 1

+ **int** (mặc định 0) : kiểu số nguyên (4 byte) từ -232 – 1 đến 232 – 1 – 1

+ **long** (mặc định 0L) : kiểu số nguyên (8 byte) -264 – 1 đến 264 – 1 – 1

+ **float** (mặc định 0.0f): kiểu số thực (4 byte)

+ **double** (mặc định 0.0d) : kiểu số thực (8 byte)

+ **boolean** (mặc định false) : kiểu logic (1 bit)

+ **char** (mặc định \u0000) : kiểu ký tự Unicode (2 byte) từ ‘\u0000’ (0) đến ‘\uffff’ (216  – 1)

* **Kiểu đối tượng** :

+ **String** (mặc định null)

+ **Array** (tuỳ thuộc data khai báo)

+ **Object**

**Từ khóa static dùng để làm gì ?**

* **Kiểu static**: Dùng để khai báo thuộc tính, method, block static của class
* Công dụng : Dùng để khai báo các thuộc tính và phương thức chung của class (Khai báo lớp tiện ích mà không cần khởi tạo nhiều lần)

**Ràng buộc khi sử dụng ?**

* Đặc điểm:

+ Static chỉ **thuộc lớp chứ không phải thuộc đối tượng** của lớp

+ Phương thức Static **được gọi mà không cần instance** của lớp

+ Trình tự khởi tạo main -> static -> obj ( vì vậy static **không tương tác được với biến non-static** hoặc **gọi trực tiếp phương thức non-static** vì static tạo trước khi non static được tạo nên không thể tương tác được nhưng **non static có thể tương tác được với static**)

+ Chỉ phương thức static mới truy cập được tới thành phần static (***phương thức thường có thể truy cập đến thành phần static thông qua class)***

***+*** Khối static được sử dụng để khởi tạo biến thành viên và được thực thi trước phương thức main tại lúc tải lớp.

1. **Có bao nhiêu loại mệnh đề if**

* Có 2 loại là if ( if…else, if…else if…else) và switch…case

1. **So sánh if và switch-case**

|  |  |
| --- | --- |
| **If**   * Dùng so sánh **>, <** * Mỗi câu lệnh có 1 biểu thức điều kiện và trả về true/false * Biểu thức điều kiện trả về true/false * Một khối lệnh thực thi nếu điều kiện đúng | **Switch…case**   * Dùng so sánh = hoặc khác * Tất cả các trường hợp đều so sánh với 1 biểu thức điều kiện duy nhất * Biểu thức điều kiện trả về byte, short, char, int, String * Nếu điều kiện đúng (không break) tất cả lệnh ở phía sau sẽ thực thi. |

1. **Khi nào dùng for, while, do-while?**

* **For :** Khi xác định được số lần lặp
* **Do – while:** Khi thực hiện ít nhất một câu lệnh rồi thực hiện lặp
* **While:** Khi không xác định được số lần lặp

1. **Phân biệt for-i và for-each**

|  |  |
| --- | --- |
| **For - i**   * Lặp theo **biểu thức điều kiện** có sẵn * Có thể có **0 hoặc 3 biểu thức** điều kiện * Duyệt và **chỉnh sửa** **được phần tử** trong mảng * Có thể **tìm được vị trí** phần tử * Chạy **xuôi** hoặc **ngược** hoặc **tại một vị trí** **bất kỳ** | **For - each**   * Thực hiện **đến cuối mảng** hoặc Collection * Bắt buộc **phải có điều kiện** để duyệt * Chỉ dùng để duyệt các phần tử trong mảng, **không thể chỉnh sửa** được giá trị * **Không thể tìm được vị trí** phần tử * **Chỉ có thể chạy xuôi** các phần tử trong mảng |

1. **Khác nhau giữa while, do-while. Cho ví dụ khi nào dùng ?**

|  |  |
| --- | --- |
| **While**   * **Không** * **Biểu thức điều kiện đầu** vòng lặp * Duyệt dữ liệu khi **không biết số lượng phần tử** trong danh sách. | **Do…While**   * **Thực hiện ít nhất một khối lệnh** trước khi lặp * **Biểu thức điều kiện cuối** vòng lặp * Sử dụng **kiểm tra đầu vào** |

1. **Break, Continue có tác dụng gì trong mệnh đề lặp ?**

* **Break**: Dừng vòng lặp ngay tức thì.
* **Continue**: Bỏ qua các khối lệnh bên dưới nó và tiếp tục vòng lặp mới.

1. **Trình bày các cách khởi tạo một mảng trong JAVA ?**

* **Lưu trữ nhiều giá trị cùng 1 lúc** và có dữ liệu ở **vị gần nhau trong 1 vùng nhớ**
* Có 2 cách khởi tạo một mảng trong Java:

+ Datatype[] arr = new Datatype[];

+ Datatype arr[] = new Datatype[];

1. **Phần tử của mảng có thể dùng kiểu dữ liệu nào, và có giá trị mặc định là gì ?**

* Gán trực tiếp thông qua {a,b}
* Toán tử new Datatype[size];
* Lưu trữ kiểu **nguyên thuỷ** hoặc **đối tượng**
* Giá trị mặc định **theo kiểu khai báo Datatype**
* Kích thước không thể thay đổi được

1. **OOP(Oriented Object Programming) là gì?**

* Là kỹ thuật lập trình, mô phỏng các **đối tượng ngoài thực tế** vào lập trình

1. **Phân biệt class và object**

* **Class :** là **khuôn mẫu mô tả thuộc tính** và **hành vi chung** của 1 nhóm đối tượng
* **Object:** thể hiện cụ thể của class

|  |  |
| --- | --- |
| **Class**   * Chỉ tạo được 1 class cùng tên trong 1 package * Khai báo bằng từ khoá access modifier + **class** | **Object**   * Có thể tạo nhiều đối tượng trong cùng 1 class * Khai báo bằng từ khoá **new** |

1. **Constructor là gì**

* Là phương thức đặc biệt khởi tạo đối tượng.

1. **Cách khai báo constructor và đặc điểm constructor trong JAVA**

* Cú pháp khai báo constructor : access modifier + name class + ( parameter ) + {}

**Đặc điểm:**

* **Cùng tên với Class**
* **Không có kiểu trả về**
* Có thể **có nhiều constructor trong 1 class** (Thể hiện của Đa hình) khác tham số
* Được gọi thông qua từ khoá **new**
* Java mặc định có **1 constructor không tham số** nhưng với trường hợp không có constructor nào được khai báo
* Có thể gọi 1 constructor khác thông qua từ khoá **this**(parameter) **được đặt ở trên cùng của body constructor**.

1. **Phân biệt constructor và method**

|  |  |
| --- | --- |
| **Constructor**   * Không có kiểu trả về * Được dùng khởi tạo đối tượng * Cùng tên class * Được gọi thông qua từ khoá new * Không thể đi cùng | **Method**   * Có kiểu trả về * Được dùng để thực hiện một công việc cố định * Không cần cùng tên với class * Được gọi thông qua instance * Có thể đi cùng với từ khoá final, static, abstract |

1. **Tính bao đóng là gì ? Làm sao để thực hiện được tính bao đóng trong java ?**

* Là kỹ thuật che dấu thông tin, chỉ hiển thị những thông tin cần thiết được thể hiện qua (getter/setter và access modifier)

1. **Tham trị, tham chiếu**

* Tham trị (Pass by value) : tạo bản sao lưu giá trị gốc và chỉ thao tác được với bản copy
* Tham chiếu (Pass by preference) : thao tác trên địa chỉ bộ nhớ ‘
* ***Java 100% tham trị (Pass by value)***

1. **Các loại biến trong JAVA**

* Bao gồm 3 loại : **Global, Local, Static**

**+ Global:** là biến được khai báo ở NGOÀI method, block, constructor.

**+ Local:** là biến được khai báo ở TRONG method, block, constructor.

**+ Static:** là biến của CLASS (Không phải của đối tượng).

1. **Trình bày các loại access modifier, và phạm vi truy cập.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Modifier** | **Class** | **Package** | **Subclass** | **World** |
| **Public** | **Y** | **Y** | **Y** | **Y** |
| **Protected** | **Y** | **Y** | **Y** | **N** |
| **Default** | **Y** | **Y** | **N** | **N** |
| **Private** | **Y** | **N** | **N** | **N** |

**Class (public & default)**

1. **Kế thừa (Inheritance) trong JAVA là gì ?**

* Là cơ chế cho phép lớp con sử dụng lại các phương thức, thuộc tính của lớp cha (Super Class) giúp rút gọn mã nguồn.
* Sử dụng từ khoá **final** để ngăn chặn kế thừa
* Quan hệ kế thừa là Is - a

1. **Lớp con kế thừa được những tài sản nào (thuộc tính, phương thức) của lớp cha ?**

* Các thuộc tính của lớp cha.
* Chỉ kế thừa được các METHOD có access modifier lớn hơn hoặc bằng **Protected**
* Không thể kế thừa Constructor (có thể gọi construtor lớp cha thông qua từ khoá super(parameter)).

1. **Lớp Object là gì**

* Lớp Object là lớp cha của tất cả các lớp trong Java. Nói cách khác nó là một lớp cao nhất trong Java.
* Sử dụng lớp Object là hữu ích nếu muốn tham chiếu bất kỳ đối tượng nào mà chưa biết kiểu dữ liệu của đối tượng đó

1. **Khái niệm đa hình**

* Là khả năng đối tượng thể hiện hành vi theo nhiều cách khác nhau ở những ngữ cảnh khác nhau

1. **Phân biệt Overloading và Overriding**

|  |  |
| --- | --- |
| **OVERRIDING** | **OVERLOADING (Constructor)** |
| Lớp con **định nghĩa lại phương thức** của lớp cha | Có thể **tạo nhiều phương thức cùng tên** nhưng **KHÁC** tham số. |
| **Cùng tên**, **cùng tham số**, **cùng kiểu dữ liệu trả về** | Nếu **cùng tham số sẽ xét kiểu dữ liệu trả về** |
| **Access Modifier** phải **lớn hơn** hoặc **bằng** lớp cha | Nếu **khác tham số** thì có thể **khác kiểu dữ liệu trả về** |
| Xảy ra ở 2 class có **mối quan hệ kế thừa** | Xảy ra ở trong **cùng 1 class** |
| Xảy ra ở quá trình **RUNTIME** | Xảy ra ở quá trình **COMPILE** |

1. **Từ khoá Final?**

* **Class :** Ngăn chặn kế thừa
* **Method:**  Ngăn chặn overriding.
* **Variable:** Khai báo hằng số.

1. **Ép kiểu là gì ? các loại ép kiểu**

* **Ép kiểu :** là thay đổi kiểu dữ liệu của 1 đối tượng gồm 2 loại **Ngầm định (Implicit)** và **Tường minh (Explicit).**

**+ Ngầm định:** thực hiện tự động thể hiện qua phép gán (**SUB->SUPER**). VD: 10 = a Int

**+ Tường minh:** tự thực hiện, phải chỉ rõ kiểu dữ liệu muốn ép kiểu (**SUPER->SUB**). VD: 130(byte) = a byte.

* *Ép kiểu dữ liệu từ lớn -> nhỏ có thể gây mất dữ liệu. =>* ***a = -126***

1. **Tính trừu tượng là gì ?**

* Là tính chất ẩn đi những thông tin của hành vi, chỉ quan tâm đến kết quả KHÔNG quan tâm đến quá trình.
* Được thể hiện qua ABSTRACT CLASS và INTERFACE.

1. **Phân biệt abstract class và interface**

|  |  |
| --- | --- |
| **ABSTRACT CLASS ( 10 – 100% ABS)** | **INTERFACE (100% ABSTRACT)** |
| **Không** thể tạo **INSTANCE** | **Không** thể tạo **INSTANCE** |
| Chứa **ABSTRACT** **METHOD** hoặc **NON**-**ABSTRACT** **METHOD** | Chỉ chứa **ABSTRACT** **METHOD** và **CONSTANT** |
| Một **ABSTRACT CLASS** khác **kế thừa** 1 **ABSTRACT CLASS** thì **không cần OVERRIDING** tất cả các **ABSTRACT METHOD** | Khi khai báo **METHOD** mặc định có **PUBLIC ABSTRACT** và **CONSTANT** thì mặc định **PUBLIC STATIC FINAL** |
| **NON-ABSTRACT** **kế thừa** **ABSTRACT CLASS** thì cần **OVERRIDING** tất cả các **ABSTRACT METHOD** | Một **INTERFACE** có thể **kế thừa** **nhiều INTERFACE khác**, 1 **CLASS** có thể **implements nhiều INTERFACE** khác (Khắc phục đơn kế thừa CLASS) |
| **NON-ABSTRACT CLASS** **không** thể **chứa** các **ABSTRACT METHOD** | Kể từ **Java 8** cho phép **khai báo** **ABSTRACT METHOD** bằng từ khoá **DEFAULT** |

1. **So sánh Array và ArrayList**

|  |  |
| --- | --- |
| **Array** | **Array List** |
| Kích thước **cố định** | Kích thước có thể **thay đổi** |
| Lưu kiểu dữ liệu **nguyên thuỷ** và **đối tượng** | Chỉ lưu kiểu dữ liệu **đối tượng** (Kể từ Java 5, kiểu dữ liệu nguyên thuỷ được tự động chuyển đổi sang đối tượng bằng cơ chế **auto-boxing**) |
| Tốc độ lưu trữ và thao tác **nhanh hơn** | Tốc độ lưu trữ và thao tác **chậm hơn** |
| Chỉ có thuộc tính **length** | Cung cấp nhiều method để thao tác |

* **Auto-boxing:** là quá trình biên dịch tự động chuyển đổi giữa kiểu dữ liệu nguyên thuỷ sang kiểu tham chiếu. (biến kiểu tham trị sẽ **khởi tạo một đối tượng** trong **vùng nhớ Heap** và **copy giá trị của biến tham trị vào đối tượng mới này**, được thực hiện nhờ quá trình chuyển đổi ngầm định)
* **Unboxing:** là quá trình ngược lại với Boxing (quá trình được thực hiện một cách tường minh).
* **Wrapper Class:** cung cấp cơ chế để chuyển đổi kiểu dữ liệu nguyên thuỷ thành kiểu đối tượng và từ đối tượng thành kiểu dữ liệu nguyên thuỷ.

1. **So sánh ArrayList và LinkedList**

|  |  |
| --- | --- |
| **Array List** | **Linked List** |
| Sử dụng **mảng động** để lưu trữ các phần tử | Sử dụng **danh sách liên kết đôi** để lưu trữ các phần tử |
| Một cấu trúc dữ liệu **dựa trên chỉ số index** và **mỗi phần tử được liên kết** với 1 chỉ số **index** | Các phần tử trong LinkedList là các **node,** mỗi **node** gồm 3 phần: địa chỉ trước nó, giá trị node, địa chỉ kế tiếp |
| Tốc độ truy xuất **nhanh** (do truy xuất **thông qua chỉ số index**) | Tốc độ truy xuất **chậm** (do duyệt **tuần tự** từ node đầu đến node cuối) |
| Chèn, xoá **chậm** (do việc **dịch các phần tử** trong mảng được diễn ra khi thực hiện) | Chèn, xoá **nhanh** (chỉ việc trỏ **tham chiếu lại địa chỉ trước và sau node** đó) |
| **Có thể** truy xuất phần tử ngẫu nhiên | **Không** thể truy xuất ngẫu nhiên |
| Hoạt động như một **List** | Hoạt động như một **List**, **Stack**, **Queue** |
| Cần **ít** bộ nhớ (chỉ lưu data và index) | Cần **nhiều** bộ nhớ |
| Lưu trữ dữ liệu **ít** thay đổi | Lưu trữ dữ liệu thay đổi **nhiều** |

1. **Set là gì, các lớp triển khai của Set**

* **Set là** một **INTERFACE** kế thừa **COLLECTION INTERFACE** trong Java. **Không thể chứa các phần tử trùng lặp.**
* **Các triễn khai của Set gồm:** HashSet, LinkedHashSet, TreeSet

**+ HashSet:** lưu trữ các phần tử dưới dạng bảng **“BĂM”. Không duy trì thứ tự chèn vào**

**+ LinkedHashSet:** lưu trữ các phần tử dưới dạng mã **“BĂM”** theo cấu trúc **danh sách liên kết**. **Duy trì thứ tự chèn vào**

**+ TreeSet:** Lưu trữ dữ liệu dưới dạng **TREE**, **các phần tử được sắp xếp** (mặc định từ bé -> lớn)

**+ *Bản chất của HashSet và LinkedHashSet là HashMap***

1. **Generic là gì**

* **Generic** được hiểu là **tham số hoá kiểu dữ liệu.**
* Giúp cho việc **bắt lỗi các kiểu dữ liệu không hợp lệ** (VD: List<Employee> không thể thêm Customer).
* Giúp dễ dàng hơn cho việc tạo và sử dụng các class, interface, method với nhiều kiểu dữ liệu khác nhau (VD: List <T> đa dạng kiểu dữ liệu)

1. **Ưu điểm và hạn chế khi dùng generic ?**

* **Ưu điểm:**

**+ Dùng để viết những thuật toán tổng quá :** rút gọn mã nguồn.

**+ Kiểu đối tượng an toàn:** Chỉ lưu một kiểu đối tượng duy nhất. Không cho phép lưu trữ 2 đối tượng có kiểu khác nhau

**+ Hạn chế ép kiểu**

**+ Kiểm tra lúc biên dịch:** giúp hạn chế bị lỗi lúc Runtime

* **Hạn chế:**

+ Không sử dụng được với **Static**

+ Không thể tạo **Instance**

+ Không sử dụng được với **KDL nguyên thuỷ**

+ Không dùng được **overload**

1. **Stack là gì, các phương thức của stack ?**

* **Stack là** cấu trúc dữ liệu dạng danh sách. Thêm và lấy phần tử theo qui tắc FILO (First In Last Out)
* **Các phương thức thao tác:**

**+** Thêm : push, add

**+** Lấy ra và xoá : pop (lấy ra phần tử trên cùng. Trả về phần tử đó)

**+** Lấy ra không xoá: peek (lấy ra phần tử trên cùng nhưng không xoá).

**+** Tìm kiếm: search

**+** Chứa đối tượng: constains

1. **Queue là gì, các class triển khai của queue ?**

* **Queue là** cấu trúc dạng danh sách. Thêm và lấy phần tử theo qui tắc FIFO (First In First Out)
* **Class triển khai Queue:** LinkedList, ArrayDeque, PriorityQueue.

1. **Phương thức cơ bản của queue ?**

**Các Phương thức thao tác:**

**+ add / offer(return true** nếu add thành công, nếu không **false)**

**+ peek / element:** lấy ra phần tử ở đầu hàng đợi không xoá (peek nếu rỗng trả về null; element ném ra exception)

**+ poll / remove :** lấy ra phần tử ở đầu hàng đợi và xoá (poll rỗng trả về null; remove ném ra exception)

1. **So sánh Comparable và Comparator, khi nào dùng cái nào?**

|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| Triển khai interface **trong cùng class** muốn sắp xếp | Không cần triển khai interface ở class **khác class** cần sắp xếp |
| Override **compareTo()** | Override **compare()** |
| Thư viện **java.lang** | Thư viện **java.util** |
| Ảnh hưởng **thay đổi class** muốn sắp xếp | Không ảnh hưởng |
| Collections.sort(List) | Collections.sort(List, Comparator) |

1. **Map là gì, các class triển khai**

* **Map là** cấu trúc dữ liệu theo dạng key – value
* Các cặp key – value còn gọi là **entry**
* Lưu trữ **key không trùng** và **value có thể trùng**
* **Map** thuộc **Java Collection Framework**
* **Các class triển khai:**

**+ HashMap:**

* Cho phép 1 key có giá trị null
* Không duy trì thứ tự chèn

**+ LinkedHashMap:**

* Tương tự HashMap nhưng duy trì thứ tự chèn vào

**+ TreeMap:**

* Tương tự HashMap nhưng sắp xếp theo thứ tự tăng dần (đối với KDL nguyên thuỷ)
* Không cho phép key null

1. **Cây nhị phân là gì**

* **Cây nhị phân là** một cấu trúc dữ liệu trong đó **mỗi nút** có thể có **nhiều nhất 2 nhánh con** được chia 2 nhánh là left-subtree và right-subtree. **Các nút anh em** nhau gọi là **Sibling**. Các nút không có node con gọi là **Leaf node**. **Nút trên cùng của cây nhị phân được gọi là nút gốc** hoặc **nút cha** và **các nút dưới từ nút gốc** được gọi là **nút con. (Giá trị được sắp xếp left < center < right )**

1. **Trình bày các cách duyệt cây nhị phân**

* **Có 4 cách duyệt cây nhị phân:**

**+ Inorder**: Trái -> Giữa -> Phải

**+ Postorder**: Trái -> Phải -> Giữa

**+ Preorder**: Root -> Left (hết) -> Right (hết)

**+ Breath-first**: Root -> Từng tầng (Left - > Right)

1. **Ngoại lệ (Exception là gì) ?**

* **Ngoại lệ là** sự kiện thất thường xảy ra trong quá trình thực thi chương trình. Phá vỡ luồng bình thường của chương trình, gây chết chương trình.

1. **Phân loại Exception**

* **Gồm 2 loại : Checked và Unchecked**

**+ Checked**: xảy ra trong quá trình **Compile** không thể bỏ qua, bắt buộc phải xử lý (VD: FileNotFound, IOException….)

**+ Unchecked**: xảy ra trong quá trình **Runtime** không chắc chắn xảy ra, không bắt buộc xử lý (VD: InputMissMatch, NullPoint.., ArrayIndexOutOfBound, NegativeArraySize,…)

1. **Phân biệt Error và Exception**

|  |  |
| --- | --- |
| **Error** | **Exception** |
| Xảy ra do **hệ thống** | Xảy ra do **source code** |
| Không thể phục hồi | Có thể phục hồi, sửa chữa được. |
| Không có cách xử lý | Sử dụng **từ khoá try…catch** và **throw** |
| Khi lỗi chương trình sẽ **dừng** | Ngoại lệ được phát hiện bằng **throw** và xử lý trong **khối catch tương ứng** |
| Được phân loại là loại **không kiểm tra được** | **Checked** và **Unchecked** |
| Trong thư viện **java.lang.error** | Trong thư viện **java.lang.exception** |
| OutOfMemory, StackOverFlow | **Checked**: NoSuchMethod, ClassNotFound. **UnChecked**: NullPoint, IndexOutOfBound. |

1. **Có bao nhiêu cách để xử lý ngoại lệ (handle)**

* **Try/catch (finally) :** xử lý exception ngay tại nơi gặp phải
* **Throw/Throws:** Ném ngoại lệ đến nơi khác để xử lý

1. **Một số lưu ý khi dùng try-catch**

* **Lưu ý:**

**+** Khi thực thi **chỉ có 1 exception xảy ra** và **1 khối catch được thực thi** (có nhiều catch)

**+** Nếu **dùng nhiều khối catch**, phải **sắp xếp theo thứ tự cụ thể đến chung nhất** (con -> cha)

**+ Finally**: luôn được thực thi dù có xảy ra ngoại lệ hay không (chỉ có 1 khối finally)

**+** Khối **try** **bắt buộc phải có**.

**+** Có thể không có **catch** nhưng phải có **Finally**.

1. **SOLID**

* Là **nguyên lý thiết kế trong lập trình**. Giúp **xây dựng**, **phát triển ứng dụng** tốt hơn.

**+ S: Single Responsibility principle –** nguyên lý **trách nhiệm duy nhất**

* Mỗi class **chỉ nên làm 1 nhiệm vụ duy nhất** => có thể **phát triển độc lập từng thành phần**
* **Chia ứng dụng thành nhiều phần** rõ ràng chức năng
* **Dễ dàng** phát triển, maintain

**+ O:** **Open/closed principle –** nguyên lý **đóng mở**

* Có thể **thoải mái mở rộng class** nhưng **không được sửa đổi** bên trong **class**

**+ L:** **Liskov substitution principle –** nguyên lý **thay thế Liskov**

* Các **object của class con có thể thay thế class cha** mà không làm thay đổi tính đúng đắn của chương trình

**+ I:** **Interface segregation principle –** nguyên **lý phân tách interface**

* Thay vì **dùng 1 interface lớn** => nên **tách** ra **nhiều** **interface** **nhỏ**, gồm các **method** **liên** **quan** đến nhau => **dễ** **dàng** **quản** **lý** hơn.

**+ D:** **Dependency Inversion principle –** nguyên **lý đảo ngược phụ thuộc**

* Các **module** **cấp** **cao** **không** nên **phụ** **thuộc** vào **module** **cấp** **thấp** , cả 2 **nên phụ thuộc** vào **abstraction**
* **Interface** (abstraction) **không** nên **phụ** **thuộc** vào **chi** **tiết**, mà ngược lại. (các giao tiếp với nhau thông qua interface, không phải thông qua implementation)

1. **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** có thể **quăng ra Exception** ở **bất kỳ dòng nào** trong phương thức (sau đó dùng try-catch để bắt hoặc throws cho thằng khác sử lý) | Throws được **khai báo ngay sau dấu đóng ngoặc đơn của phương thức**. Khi một phương thức có throw bên trong mà không bắt lại (try – catch) thì phải ném đi (throws) cho thằng khác xử lý |
| **Không** thể **throw** **nhiều** **exceptions** | Có thể **khai** **báo** **nhiều** **exceptions**, Ví dụ:  public void method() throws IOException, SQLException { } |

1. **Collection và Collections**

* **Collection** là một interface cấp cao nhất nhất của Collection Framework.
* **Collections** là một lớp tiện ích. **Collections** bao gồm các phương thức static được sử dụng để thao tác trên các đối tượng của Collection (List, ArrayList, LinkedList, Map, Set, …).

**Những điểm quan trọng về lớp Collections trong java là:**

* Lớp Collections hỗ trợ các thuật toán đa hình (polymorphic algorithms) hoạt động trên các collection.
* Lớp Collections ném một ngoại lệ NullPointerException nếu các collection hoặc các đối tượng lớp cung cấp cho chúng là null.