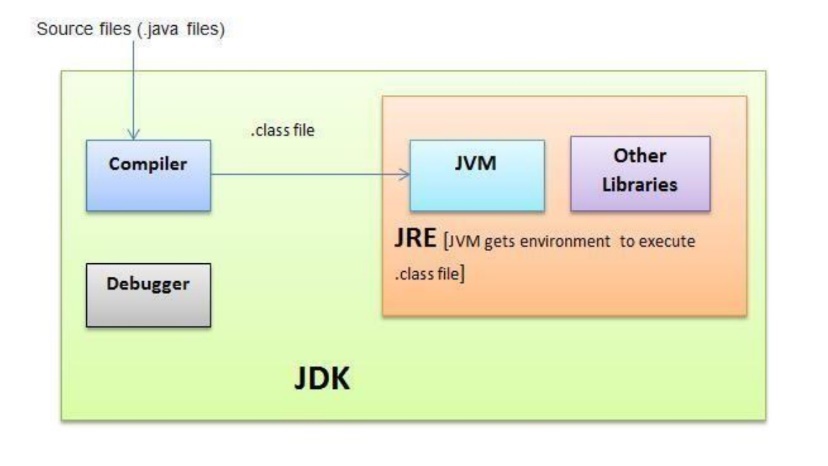
1. ***JAVA là gì ?***

**Java** là ngôn ngữ lập trình hướng đối tượng ( OOP ), có khả năng thực thi ở nhiều loại thiết bị, được sử dụng rộng rãi và được sở hữu bởi Oracle.

1. ***JDK, JRE, JVM***

* **JDK** ( **Java Development Kit**) : Bộ công cụ hỗ trợ lập trình Java

Bao gồm :

* Trình biên dịch compiler ( Javac )
* Debugger
* JRE : Môi trường thực thi
* **JRE** ( **Java Runtime Environment** )

**Môi trường thực thi Java**

Bao gồm :

* Thư viện ( Library )
* JVM : Máy ảo Java
* **JVM** ( **Java Virtual Machine** )

**Máy ảo Java**

Thông dịch mã bytecode sang ngôn ngữ máy.

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

* **Biên dịch** : dịch 1 file sang file mới, khi chạy chương trình thì không cần phải dịch lại.
* **Thông dịch** : dịch từ dòng code để thực thi, khi chạy cần phải dịch lại.

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

=>Java là ngôn ngữ vừa biên dịch ( file.java ) 🡪 file.class ( bytecode) và vừa thông dịch ( JVM thông dịch từ file.class 🡪 ngôn ngữ máy )

1. ***Java Platform là gì?***

* **Java Platform** (**nền tảng Java**) là một tập hợp các chương trình giúp phát triển và chạy các chương trình được viết bằng ngôn ngữ lập trình Java.

***-***

1. ***Các kiểu dữ liệu trong java, và giá trị mặc định khi khai báo.***

* Kiểu dữ liệu : **có 2 kiểu**
* Kiểu số nguyên : 4 kiểu số nguyên ( byte, short, int, long ) ; 2 kiểu số thực ( float, double ) ; 2 kiểu kí tự ( bloolean, char ).

+ **byte** (mặc định 0)

+ **short** (mặc định 0

+ **int** (mặc định 0)

+ **long** (mặc định 0L)

+ **float** (mặc định 0.0f)

+ **double** (mặc định 0.0d)

+ **boolean** (mặc định false)

+ **char** (mặc định \u0000)

* Kiểu đối tượng :

+ **String** (mặc định null)

+ **Array** (tuỳ thuộc data khai báo)

+ **Object :** - Do Java tự định nghĩa : Scanner

- Do lập trình viên tự định nghĩa : Student, customer

1. ***Sự khác nhau giữa bộ nhớ heap và stack trong java?***

|  |  |
| --- | --- |
| **Stack** | **Heap** |
| + Bộ nhớ nhỏ  + Để lưu biến cục bộ, method, biến tham chiếu giá trị nguyên thủy  + Tự động xóa khi hàm kết thúc | + Bộ nhớ lớn, có thể thay đổi dc  + Lưu đối tượng( tạo bằng new )  + Với Java có cơ chế Garbage Collection để dọn rác khi ko còn ai tham chiếu tới địa chỉ ô nhớ. |

Java chỉ truyền tham trị không truyền tham chiếu. Vì:

+ Kiểu nguyên thủy copy giá trị

+ Kiểu tham chiếu copy giá trị địa chỉ ô nhớ

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

### Khởi tạo Mảng :

### DataType [ ] arrayName = { value1, value2, …. }

### DataType [ ] arrayName = new DataType [ size ]

### //size : kích thước của mảng

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 là gì ?***

### Lập trình hướng đối tượng là kĩ thuật ánh xạ các đối tượng từ thực tế vào bên trong ngôn ngữ 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, có đặc điểm và hành vi cụ thể

|  |  |
| --- | --- |
| **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***

### Contructor là 1 method đặc biệt dùng để khởi tạo các 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 :

### Phải cùng tên với class.

### Không có biến kiểu dữ liệu trả về.

### Trong 1 class có thể tạo nhiều contructor ( với nhiều tham số khác nhau )

### Nếu trong class không có bất kì contructor nào thì java sẽ tự cung cấp cho ta 1 contructor mặc định không tham số.

### Contructor được gọi khi khởi tạo đối tượng ( sử dụng từ khoá new )

### 1 contructor chỉ được gọi 1 contructor khác thông qua từ khoá this ( lưu ý là phải đặt trên đầu contructor ).

1. ***Phân biệt constructor và method***

|  |  |
| --- | --- |
| Contructor | Method |
| Phải cùng tên với classKhông có kiểu dữ liệu trả vềĐể khởi tạo ObjectĐược gọi thông qua từ khoá newKo sử dụng từ khoá abtract, final, static | Có thể trùng tên với class => khuyến cáo không nên dùng.Có kiểu dữ liệu trả vềĐể thực thi nv cho ObjectĐược gọi thông qua tên của methodCó thể sử dụng abtract, final, static |

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

* Là kỹ thuật che dấu thông tin, chỉ hiển thị những thông tin cần thiết

### Để có được tính bao đóng : sử dụng ( getter/setter và access modifier )

1. ***Tham trị, tham chiếu***

### Khi gọi hàm và truyền đối số vào cho hàm

### Nếu đối số là kiểu nguyên thuỷ ( primitive ) => tham trị

### Nếu đối số là kiểu đối tượng ( refence ) => tham chiếu

### Trong Java 100% truyền tham trị không có truyền tham chiếu .

### VÌ : Kiểu nguyên thuỷ copy giá trị

### Kiểu đối tượng coppy giá trị địa chỉ ô nhớ.

1. ***Từ khóa static dùng để làm gì ?***

* Từ khóa Static dùng để quản trị bộ nhớ.
* Static là từ khoá dùng để khai báobiến, phương thức và khối .
* Các thành phần static thuộc về class.
* Mục đích sử dụng static:

+ Biến static: + Tạo ra biến chung cho tất cả các đối tượng

+ Lấy bộ nhớ chỉ 1 lần , tác dụng biến đếm

+ Phương thức static: Tạo ra các class tiện ích ( vì ko cần tạo đối tượng ko tốn vùng nhớ ) Class.method ví dụ Math, Caculator

+ Khối static

1. Được sử dụng để khởi tạo thành viên dữ liệu static.
2. Nó được thực thi trước phương thức main tại lúc tải lớp.
3. ***Ràng buộc khi sử dụng static***

### + Phương thức static có thể gọi được phương thức static, không gọi được phương thức thường ( muốn gọi được thì phải khởi tạo đối tượng )

### + Phương thức static có thể truy cập / thay đổi giá trị biến static.

1. ***Các loại biến trong JAVA***

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

**- Global ( instance ) :** là biến được khai báo trong Lớp và ở NGOÀI method, block, constructor.

+ Có giá trị mặc định khi khởi tạo

+ Có thể sử dụng ở bất cứ phương thức, hàm tạo hay khối lệnh thuộc lớp chứa nó

+ Lưu ở Heap

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

+ Ko có giá trị mặc định

+ Sẽ hủy khi kết thúc phương thức hay khối lệnh

+ Lưu ở Stack

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

+ Tham chiếu thuộc tính chung của tất cả các đối tượng

+ Lưu ở vùng nhớ class area

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

***Quy định phạm vi truy cập***

* ***Private:***

+ Private Contrustor thì ko thể tạo đối tượng bên ngoài class.

+ Một lớp ko thể Private hoặc Protected trừ khi lớp lồng nhau.

* ***Default:***

+ Truy cập cùng Package

* ***Protected:***

+ Truy cập trong Package và ngoài package với điều kiện là kế thừa

+ Ko áp dụng cho lớp

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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** |

1. ***Kế thừa 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)).
* Lớp con ko thê truy cập vào thuộc tính private của lớp cha.

1. ***Lớp Object là gì***

### Object là lớp cha của tất cả các class trong java ( kể cả class do LTV tự định nghĩa )

* 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 1 đối tượng có thể hiện/ hành vi theo những cách khác nhau tuỳ thuộc vào ngữ cảnh.

### Trong java cung cấp 2 cơ chế thể hiện tính đa hình

### + overloading ( nạp chồng )

### + overriding ( ghi đè )

1. ***Phân biệt Overloading và Overriding***

|  |  |
| --- | --- |
| Overloading | Overriding |
| Là cơ chế cho phép trong 1 class có thể định nghĩa ra nhiều phương thức cùng tên nhưng khác tham số truyền vào.Quy tắc để có Overloading :+ 2 method cùng tên+ Khác tham số truyền vàoXảy ra trong cùng 1 classĐa hình tại compiler ( thời điểm biên dịch ) | Là cơ chế cho phép lớp con định nghĩa lại nội dung các phương thức đã có ở lớp cha.Quy tắc để có method overriding :+ phải có cùng tên, cùng tham số, cùng kiểu dữ liệu trả về.+ Access modifier phải lớn hơn or = access class của method ở lớp cha.Xảy ra trong MQH kế thừa ( ở lớp con )Đa hình tại runtime ( thời điểm thực thi)Ko ghi đè dc pthuc static vì static thuộc class còn pthuc instance thuộc đối tượng |

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

### Ép kiểu là cơ chế chuyển 1 tham chiếu đến đối tượng thuộc loại này thành tham chiếu đến đối tượng thuộc loại khác.

### Có 2 loại ép kiểu :

### + Implicit Casting : ngầm định => diễn ra tự động / ngầm định bởi hệ thống.

**+ Explicit Casting : tường minh => không tự động mà do LTV thực hiện.**

**Lưu ý: Phải có ép kiểu ngầm định mới có ép kiểu tường minh.**

**+ Ép kiểu phải thuộc chuỗi kế thừa.**

1. ***Tính trừu tượng là gì ?***

### Tính trừu tượng là khả năng ẩn chi tiết của tiến trình triển khai, chỉ hiện thị tính năng, kết quả cho người dùng ( chỉ quan tâm đến kết quả, không quan tâm đến cách thực hiện )

### Trong java tính trừu tượng thể hiện qua abtract class & interface

**=> Cụ thể là abtract method ( phương thức trừu tượng )**

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

***Chung: Không thể tạo đối tượng***

|  |  |
| --- | --- |
| **ABSTRACT CLASS ( 10 – 100% ABS)** | **INTERFACE (100% ABSTRACT)** |
| Chứa **ABSTRACT** **METHOD** hoặc **các thuộc tính hay hành vi như class bình thường** | Chỉ chứa **ABSTRACT** **METHOD** và thuộc tính là **CONSTANT** |
| Có Constructor  Là 1 class bình thường | Không Có Constructor  Không phải là class |
| Một **ABSTRACT CLASS** khác **kế thừa** 1 **ABSTRACT CLASS** thì **không cần OVERRIDING** tất cả các **ABSTRACT METHOD**  **Class bình thường** **kế thừa** **ABSTRACT CLASS** thì cần **OVERRIDING** tất cả các **ABSTRACT METHOD** | Khi khai báo **METHOD** mặc định có **PUBLIC ABSTRACT** và **CONSTANT** thì mặc định **PUBLIC STATIC FINAL** |
| **1 class chỉ có thể extend dc 1 abstract class**  **Ko hỗ trợ đa kế thừa**  **Mối quan hệ is-a** | 1 **CLASS** có thể **implements nhiều INTERFACE** khác (Khắc phục đơn kế thừa CLASS)  Hỗ trợ đa kế thừa  Mối quan hệ can-do  Một INTERFACE có thể kế thừa nhiều INTERFACE khác |
| **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 |
| Độ dài cố địnhChỉ có thuộc tính lengthTốc độ thao tác nhanhCó thể lưu được kiểu nguyên thuỷ vào đối tượng.Không có generic < > | Độ dài thay đổi đượcCó method size( ) và các method : get( ) , add( )Tốc độ chậm hơnChỉ có thể lưu kiểu đối tượng Từ JDK5 : có cơ chế Autoboxing có thể lưu kiểu nguyên thuỷ. Int => Interger ( Wrapper class )Có generic < > |

1. ***So sánh ArrayList và LinkedList***

|  |  |
| --- | --- |
| Array List | Linked List |
| Dựa trên cơ chế mảng động để lưu phần tửTruy xuất ngẫu nhiên nhanh hơnXoá, chèn chậm hơnTốn ít bộ nhớ hơn | Sử dụng danh sách liên kết dựa trên các nodeTruy xuất ngẫu nhiên chậm hơnXoá, chèn nhanh hơnTốn nhiều bộ nhớ hơn vì có thêm các Node. |

1. ***So sánh List, Set và Queue ?***

***Java Collection FrameWork là 1 khuôn mẫu cung cấp các Interface và Class để lưu trữ và thao tác với nhóm các đối tượng.***

**Collections trong java** là 1 class tiện ích, gồm những static method, để lưu trữ và xử lý các collection.Như Collections.sort, addAll vv.vv.vv

**Collection trong java** là một root interface trong hệ thống cấp bậc Collection. Java Collection cung cấp nhiều interface (Set, List, Queue, Deque vv) và các lớp (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet vv).

### + List : danh sách có cấu trúc dữ liệu tuyến tính. Duy trì thứ tự thêm vào và lưu được các phần tử trùng lặp.

### + Set : Tập hợp các phần tử không duy trì thứ tự thêm vào và lưu các phần tử không trùng lập.

### + Queue : Hàng đợi => Hoạt động theo cơ chế FIFO ( first in – first out )

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 < >** là cơ chế tham số hoá dữ liệu , cho phép chúng ta tạo ra các lớp, interface hoặc method hoạt động với nhiều kiểu dữ liệu khác nhau

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

### Ưu điểm :

### + Pháp hiện được lỗi tại thời điểm biên dịch

### + Không cần ép kiểu

### + Xây dựng được các lớp tổng quát để tái sử dụng mã nguồn.

### Nhược điểm :

### + Không thể khởi tạo generic với dữ liệu kiểu nguyên thủy

### + Không thể tạo instance cho kiểu dữ liệu

### + Không thể là static trong class

### + Không thể tạo class ngoại lệ là generic

### Một số ký tự đại diện :

### + E : Element ( phần tử )

### + T : Type

### + K, V : key – value

### + N : Number

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

### Stack là cấu trúc dữ liệu dạng ngăn xếp. Thêm và lấy phần tử theo qui tắc FILO ( first in – last out ) => vào trước – ra sau.

### Java thì Stack là 1 class

### Các method của stack :

### + push ( ) : thêm vào vị trí trên cùng

### + pop ( ) : lấy ra và xoá phần tử trên cùng

### + peek ( ) : lấy ra để xem và không xoá phần tử

### + is Empty ( ) : kiểm tra xem Stack đã hết phần tử chưa

### + size ( ) : trả về số lượng ( kích thước của stack )

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

### Queue hàng đợi là 1 cấu trúc dữ liệu thêm và lấy dữ liệu theo quy tắc FIFO ( vào trước ra trước ).

### Queue trong java là interface : các class triển khai.

### + LinkedList :

### + ArrayDeque :

### + PriorityQueue :

1. ***Phương thức cơ bản của queue ?***

### Các method của queue

### + add / offer ( ) : thêm phần tử

### + remove / poll ( ) : lấy ra và xoá

### + element / peek ( ) : lấy ra xem nhưng không xoá.

### Remove, element sinh ra ngoại lệ khi queue trống

### Poll và peek thì trả về null

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

|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| Bạn phải implements giao tiếp Comparable cho **class** đối tượng cần được so sánh. | Không cần implements ở 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 ( interface ) : dùng để lưu và truy xuất dữ liệu theo < key - value >

### Mỗi < key – value > gọi là entry ( phần tử lưu trữ )

### Không cho phép key trùng lặp

### Mỗi key tương ứng với 1 value

### Các class triển khai của map : Hash Map, Linked Hash Map, Tree Map

1. ***Phương thức equals() và hashcode() trong Java là gì?***

Equals( ) dùng để so sánh 2 đối tượng

Hashcode( ) là mã băm,

Để so sánh 2 đối tượng ta dùng equals và hashcode :

Nếu equals và hashcode giống nhau => đối tượng giống nhau, còn nếu equals và hashcode khác nhau => đối tượng khác nhau

1. ***Cây nhị phân là gì***

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

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

### Cách duyệt BST

### + Inorder : trái – cha – phải

### + Postorder : trái – phải – cha

### + Preorder : cha – trái – phải

+ Breath – first : duyệt theo level

1. ***Ngoại lệ (Exception là gì) ?***

### Exception ( ngoại lệ )

### Là sự kiện bất thường xảy ra trong quá trình thực thi chương trình. Nó phá vỡ luồng thực thi của chương trình có thể làm chết chương trình => có thể xử lý được (handle)

1. ***Phân loại Exception***

### Phân loại :

### + Uncheck Exception : xảy ra trong quá trình Runtime không chắc chắn xảy ra, không bắt buộc xử lý

### VD : Number Format Exception, Array Index Out Of Bound Exception

### + Checked Exception : xảy ra trong quá trình Compile không thể bỏ qua, bắt buộc phải xử lý

### VD : File Not Fornd Exception , IO Exception

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** |

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

### Xử lý ngoại lệ có 2 cách :

### Cách 1: dùng try - catch

### + Xử lý tại chỗ những đoạn code nghi ngờ xảy ra lỗi bỏ trong khối try{ }

### + Khối catch : những đoạn code cần xử lý ( những công việc cần làm khi -xảy ra exception )

* Cách 2 dùng thows : ném ra bên ngoài cho nơi gọi method xử
* **1, *Throws* được dùng để khai báo các exception của một method trong khi *Throw* được dùng để ném một exception cụ thể.**
* **2, Theo sau từ khoá *Throw* là một instance của exception trong khi theo sau *Throws* là tên class của các exception.**
* **3, Từ khoá Throw được dùng bên trong method trong khi Throws được dùng chung với phần khai báo của method.**

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

### Lưu ý khi sử dụng try – catch :

### + Khối try có thể có 1 or nhiều khối catch nhưng chỉ có 1 khối finaly

### + Tại 1 thời điểm thì chỉ có 1 exception xãy ra và chỉ có 1 khối catch

### + Các khối catch phải được sắp xếp thì cụ thể đến chung nhất ( từ exception con đến exception cha )

### + Khối finaly luôn luôn chạy cho dù Exception có xảy ra or không xảy ra

1. ***Serialization là gì ? lưu ý khí sử dụng Serialization***

### Serialization là cơ chế chuyển đổi trạng thái của đối tượng vào 1 chuỗi byte stream, ngược lại đổi từ byte sang object gọi là Deserialization

### Byte Stream

|  |  |  |  |
| --- | --- | --- | --- |
| Abtract Class | Class hay dùng | Class đọc ghi | Class tạo bộ nhớ đệm |
| InputStreamOutputStream | FileInputStreamFileOutputStream | ObjectInputStreamObjectQutputStream | BufferInputStreamBufferOutputStream |

### 1 số lưu ý triển khai Serialization trong Java

### Class cha implement Serialization thì class con không cần implement Serialization

### Nếu 1 thuộc tính áp dụng từ khoá transient thì không được Serialization

### Thuộc tính static thuộc về class không thể Serialization

1. ***SOLID là gì? Phát biểu của từng nguyên lý***

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 nên đảm nhận 1 nhiệm vụ duy nhất

* **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. ***Design pattern là gì? Chia làm bao nhiêu loại. Tổng số là bao nhiêu?***

### Design Pattern ( mẫu thiết kế ) là 1 giải pháp tổng thể cho các vấn đề chung trong thiết kế phần mềm.

### Design Pattern không phải là ngôn ngữ lập trình

### Được sử dụng phổ biến trong lập trình OOP

### Chia làm 3 loại :

### + Creational (khởi tạo) : giải quyết các vấn đề liên quan tới khởi tạo đối tượng ( Singleton, Factory method )

### + Structural ( cấu trúc ) : sử dụng các thiết kế các thành phần của lớp và đối tượng ( Proxy, Facade )

### + Behavioral ( hành vi ) : sử dụng giải quyết các vấn đề giao tiếp của các đối tượng ( Obsever, Strategy )

### Tổng số là 6

|  |  |
| --- | --- |
| ****.**** | So khớp với bất kỳ ký tự đơn nào |
| ****^**** | So khớp phần đầu của chuỗi hay dòng |
| ****$**** | So khớp phần cuối của chuỗi hay dòng |
| ****(…)**** | So khớp với các ‘nhóm’ ký tự bên trong |
| ****[…]**** | So khớp với bất kỳ ký tự đơn nào trong dấu ngoặc vuông |
| ****[^…]**** | So khớp với bất kỳ ký tự đơn nào ngoại trừ các ký tự trong dấu ngoặc vuông |
| ****[m-n]**** | So khớp từ ký tự m đến ký tự n theo thứ tự trong ASCII |
| ****XY**** | So khớp với ‘X theo sau là Y’, ví dụ: [a-e][i-u] |
| ****X|Y**** | So khớp với X hoặc Y |
| ****\d**** | So khớp với ký tự là chữ số, viết tắt của [0-9] |
| ****\D**** | So khớp với ký tự không phải là chữ số, viết tắt của [^0-9] |
| ****\s**** | So khớp với bất kỳ ký tự trống nào (dấu cách, tab, xuống dòng), viết tắt của [\t\n\x0B\f\r] |
| ****\S**** | So khớp với bất kỳ ký tự không phải ký tự trống, viết tắt của [^\s] |
| ****\w**** | So khớp bất kỳ ký tự chữ nào (chữ cái và chữ số), viết tắt của [a-zA-Z0-9] |
| ****\W**** | So khớp bất kỳ ký tự nào không phải chữ cái và chữ số, viết tắt của [^\w] |
| ****\b**** | Ranh giới của một từ |
| ****\B**** | Không phải là ranh giới của một từ |
| ****\A**** | So khớp phần đầu của đầu vào |
| ****\G**** | So khớp phần cuối của đầu vào |
| ****X\***** | So khớp với 0 hoặc nhiều sự xuất hiện của X, viết gọn cho X{0,} |
| ****X+**** | So khớp với 1 hoặc nhiều sự xuất hiện X,  viết gọn cho X{1,} |
| ****X?**** | So khớp 0 hoặc 1 sự xuất hiện của X, viết gọn cho X{0,1} |
| ****X{n}**** | So khớp chính xác n lần sự xuất hiện của X |
| ****X{n,}**** | So khớp n hoặc nhiều hơn số lần xuất hiện của X |
| ****X{n,m}**** | So khớp với ít nhất n và nhiều nhất m lần xuất hiện của X |