# Interface vs Abstract class

|  |  |  |
| --- | --- | --- |
|  | Abstract | Interface |
| Bản chất | Abstract class là một class nên sẽ có đầy đủ tính chất của 1 class (methods, attribute). Khác là có abstract method |  |
| Mục đích | Provide a template for futute specific classes  Sử dụng khi có các đối tượng có tính chất giống nhau | Triển khai multiple inheritances  Khi một chức năng muốn thể hiện **theo nhiều cách khác nhau** à dùng interface  **Achieve abstraction**  **Separate** the definition of a method from the inheritance hierachy  **Multiple** inheritances |
|  | Nếu 1 class có abstract methods thì đó là **abstract class** | Mặc định các phương thức trong interface là abstract and public |
|  | Nếu abstract class ko có method nào đc triển khai à nên dùng interface | Atribute in interface is implicitly public, static and final |
|  | Subclass of abstract class à must implement all the abstract method (nếu ko thì subclass vẫn là abstract class) | Can't have contructors à can't instantiate  Can't have a method with body (only sign)  Use implements to implement an interface (must imlement all methods)  \*Note: interface can **extend an interface**  Slower than abstract class because it requires time to search actual method in the corresponding class |
|  | Can't instantiate | |

# Overriding vs Overloading

Polymorphism: means having forms

Type of polymorphism:

* Compile time polymorphism
* Run time polymorphism
* To achieve polymorphism use **overriding and overloading**

|  |  |
| --- | --- |
| Overriding | Overloading |
| Ways to implement **polymorphism** in java programs | |
| The method **signature (name** and parameters) is the same in the super class and the child class | In a class two or more methods have the same name but **different parameters** |
| Impliments **runtime** polymorphism | Impliments **compile time** polymorphism |
| The method to call is determined at the **run time** based on the object type (since the signature is exactly the same, based on the object referenced) | The method to call is determined at the **compile time** |
| Achived by **function overloading and operator overloading (**java does not support)  \*Note: **không override được static method** vì static method được ràng buộc với class (thuộc về vùng nhớ class)  Override đc các method khác (private, public, protected) vì ràng buộc về đối tượng (thuộc về vùng nhớ heap) à thực thi lúc runtime | Achived by **virtual functions and pointers** |

Why overriding and overloading run-in compile or runtime: <https://dzone.com/articles/how-does-jvm-handle-polymorphism-method-overloadin>

# Mutable vs Immutable, String vs String Buffer vs String Builder

## Mutable vs immutable

|  |  |
| --- | --- |
| Mutable | Immutable |
| Mutable objects are objects whose value can be **changed after initialization** | Immutable objects are objects whose value can not be **changed after initialization** |
| **No new object is formed** when changes are made to an existing object | Whenever an existing object is changed, **a new object is formed** |
| Eg: String Buffer, StringBuilder,.. | Eg: String, primitive objects, … |

|  |  |
| --- | --- |
| StringBuffer | StringBuilder |
| **Mutable** | |
| **Synchronized** à thread safe  (two threads **cant** call the methods of StringBuffer simultaneously) à **slower** | **Non-synchronized** à note thread safe  (two threads **can** call the methods of StringBuffer simultaneously)à **faster** |

Text

Description automatically generated

|  |  |  |
| --- | --- | --- |
| String | StringBuffer | StringBuilder |
| Chậm nhất do mỗi lần thực hiện s.concat("tuan") à thực hiện tạo object mới à gán | Mỗi lần thực hiện append trực tiếp vào sBuff nhưng còn thực hiện **synchronize** | Mỗi lần thực hiện append trực tiếp vào sBuil |
| String < StringBuffer < StringBuilder | | |

# String pool

Vấn đề khi khởi tạo String:

* Có thể tạo ra các string có value giống nhau à String pool

String Pool

* Là **vùng nhớ đặc biệt trong Heap** lưu trữ các biến được khai báo theo kiểu String ( “” )
* Tối ưu hóa việc lưu trữ và sử dụng vùng nhớ khi khai báo String

Text

Description automatically generated

Diagram

Description automatically generated

Cách 1:

* Str1=”abc” java truy cập vào String Pool à tìm trong Pool xem ô nhớ nào có giá trị bằng “abc”?
  + Nếu có thì tham chiếu đến ô nhớ trong Pool
  + Nếu không tạo ô nhớ trong Pool rồi thực hiện tham chiếu
* Str2=”abc” tham chiếu trong Pool do đã có ô nhớ chứa giá trị “abc”

à str1==str2 trả về **true** do cùng tham chiếu đến cùng 1 địa chỉ ô nhớ

Cách 2:

* Tạo đối tượng mới ở Heap và thực hiện tham chiếu

àstr1==str3 trả về true do không cùng tham chiếu đến cùng 1 địa chỉ ô nhớ

**Note:** Mặc định thì toán tử == đối với Object thực hiện so sánh xem 2 biến có cùng tham chiếu đến 1 ô nhớ hay ko

# Passing Mechanism in Java

As far as Java is concerned, **everything is strictly Pass-by-Value.**

|  |  |
| --- | --- |
| Pass-by-value | Pass-by-Reference |
| Parameters passed to the callee method will be **clones of original** parameters à Any modification done in callee method will have **no effect on the original parameters** in caller method | The caller and callee operate on the same object  àany changes tho the parameter’s instance members will **effect to the original value** |
| Primitive variable pass as an argument à the actual parameters are copied to formal arguments and allocate in stack memory | Object reference àan exact copy of the reference variable is created àpoints the same location of the object in heap memory as the original reference variable |
|  |  |

**Pass-By-Value:**

**Text

Description automatically generated**

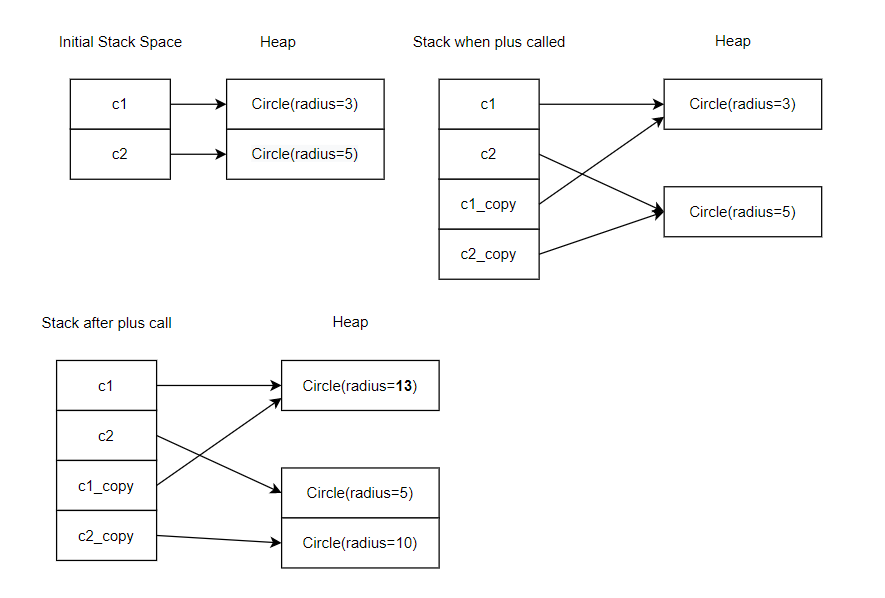
**Diagram

Description automatically generated**

**Passing Object References**

**Text

Description automatically generated**



# Final variable, final method, final class

**Final Class:**

* The **final** keyword in a class declaration does not mean that the objects of this class are **immutable** (can change attribute for object in this class)
* Final class **can not be extended**

**Final variable: Can not be** reassigned

**Final method: Can not be** overridden

**Final Arguments: can be** changed inside a method

|  |  |
| --- | --- |
| **final** | **Immutability** |
| **Cannt change the object’s reference to point to another reference** but its attribute can change (setter method) | **Change its reference but its attribute can not change** |

**Graphical user interface, text

Description automatically generated with medium confidence**

**Text

Description automatically generated**

Biến **s1** là biến final tham chiếu đến đối tượng thuộc lớp Square:

* Có thể thay đổi thuộc tính edge à **không phải immutable**
* Nhưng không thể tham chiếu đến đối tượng khác

# Static variable, static method, static class

## Static fileds (class variables)

When we declare a field static à **a single copy** of that field is **created and shared among all instances of that class**

Static variables are **stored in the heap memory**

**\*Note:**

* **Static** variables belongs to **class** à access by using class name à don’t need any object reference
* Only declare static variables at the class level

**Example:**

**Text

Description automatically generated**

**Graphical user interface, text

Description automatically generated**

**Diagram

Description automatically generated**

Mỗi lần khởi tạo object Student à *numberOfStudent* được tăng 1 đơn vị do đang là **class variable**

## The static Methods (Or Class Methods)

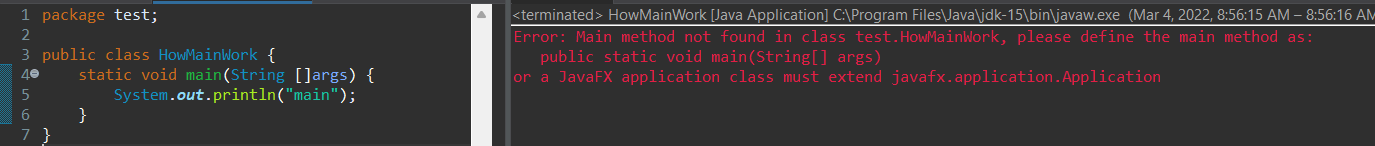
* Use static methods to perform an operation that independent upon instance creation
* In order to share a code across all instances of that class à static method

**Note:**

* **Static can’t be overridden**
* **Abstract method** cant be **static**
* Static methods **cant** use *this* or *super* keywords **(**vì class method cần phải dùng class variable**)**
* Static methods **cant access instance variables and instance methods** directly à need some object reference to do
* Static methods can only **access static variables and static methods**

**Note for main methods:** How main method works in Java ? (**public static void main (String []args)**)

* Accees modifier: **public** à java runtime can execute (**non-public à can’t run**)



* When java runtime starts, there is no object of the class present à **method has to be static** so that JVM can load the class into memory and call the main method

Graphical user interface, text, application

Description automatically generated

* main method doesn’t return anything à **void**. This has been done to keep things simple because once the main method is finished executing, java program terminates (if try to return à error)
* **main** is the name of java main method (fixed)
* **String[] args:** java main method accepts a single argument of type String array à java command line arguments

**Example:**

**Text

Description automatically generated**

## A static Block

* Used for initialzing static variables
* If static variables require additional, multi-statement logic while initialization, then a static block can be used
* A class can have multiple static blocks

Text

Description automatically generated

## A static Class

* Java programming specification **doesn't allow us to declare the top-level class as static**

à only classes within the classes (nested classes) can be made as static

Nested class architechture:

* Nested classes that are declared **static** are called **static nested** classes à access to **static members** of outer class
* Nested classes that are declared **non-static** are called **inner** classes à access to **all member** of outer class

**Text

Description automatically generated**

**SingletonHolder:** nested class

Singleton Design Pattern: **only one object** of a class to be in memory at a given point of time

****

**Diagram

Description automatically generated**

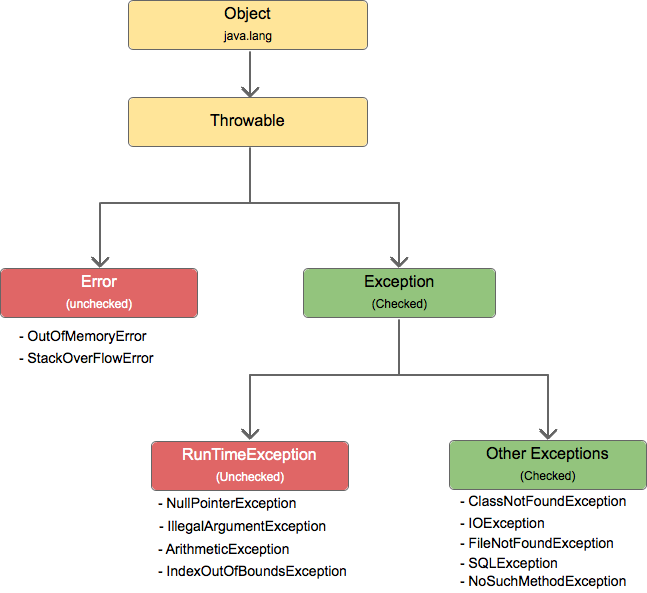
**Text

Description automatically generated**

# Try/catch vs Throw Exception

## Exception

An exception is an **unexpected event** that occurred during the exception of a program and **disrupts the normal flow of intructions**



|  |  |
| --- | --- |
| Checked | Unchecked (RuntimeExceptions) |
| The exceptions that checked in **compile-time** | The exceptions that checked in **run-time** |
|  | Compiler ignores the unchecked excetions |
| Eg: IOException,… | Eg: ArrayIndexOutOfBoundsException, NumberFormatException, … |

**Checked Exception**

**Text

Description automatically generated**

**Unchecked Exception**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Using throws**

**Text

Description automatically generated**

**throw exception**

**Text

Description automatically generated with medium confidence**

**Note:** try-with-resource <https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html>

# Synchronize vs Asynchronize

## Thread

Process can have multiple threads (shared memory)

In java, 2 ways for creating threads

* Extend java.lang.Thread class
* Implement java.lang.Runnable interface

In java, when **main method** runs à JVM starts the execution of a program à creates a **main thread (**automatically**)**

Text

Description automatically generated

* *currentThread*() will return the reference of **main thread**
* The JVM allocates a **stack** to each thread. All method calls store in the stack and each entry is known as a **stack frame (**depend upon the **stack size)**
* **JVM** will create any thread that is considered as **a child of the main thread**

Text

Description automatically generated

Diagram

Description automatically generated

## Thread Pools

* Starting a new thread for each task could limit throughput and cause poor performance
* A thread pool is ideal to **manage the number of tasks executing concurrently**
* **Executor** interface
* **ExecutorService** is a subinterface of **Executor**

To create an Executor object à use static methods in the Executor**s**

## Thread Synchronization

A shared resource may be corrupted if it is accessed simultaneously by multiple threads

àneed **synchronization**

**Race condition:** giả sử 2 thread cùng ghi vào 1 vùng nhớ dẫn đến việc kết quả sau khi thực hiện bị sai

A **thread-safe** class **does not cause a race condition** in the presence of multiple threads

## synchronized key word

Problem: race condition

àgive exclusive access to **one thread at a time** to code that manipulates a shared object

## Asynchronous and Synchronous Callbacks in Java

Callback: là một hàm được truyền vào một hàm khác như một tham số đầu vào và được thực thi dựa trên những sự kiện cụ thể như click, …

### Asynchronous Callbacks

### Synchronous Callbacks

# Lambda, Collection, stream

## Lambda

* Dùng với Functional Interface (interface with a abstract method)

Text

Description automatically generated

## Anonymous

* Anonymous inner class: Inner class là class (non static) được viết trong một class khác (out-class). Anonymous class là Inner class nhưng không có **''class'** đặt trước tên của class.
* Được biên dịch thành lớp thông thường

A screenshot of a computer

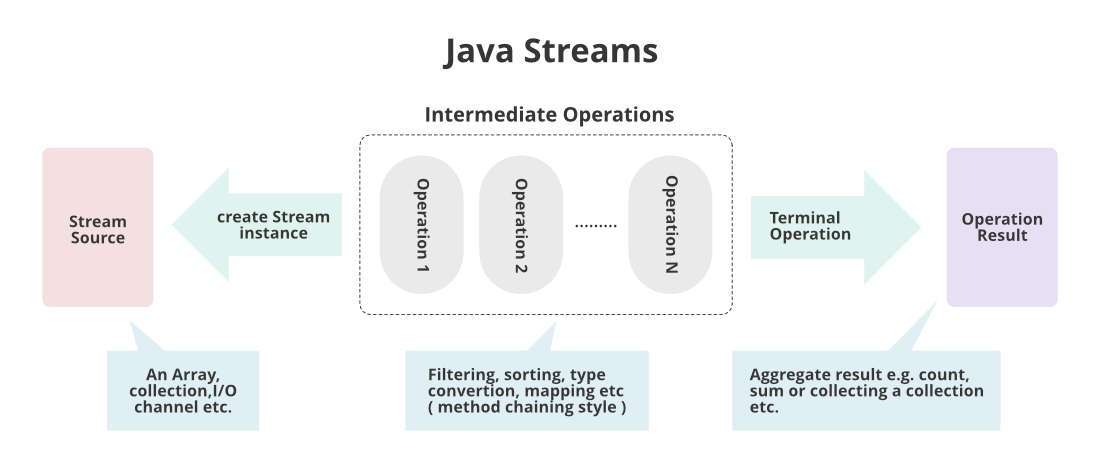
Description automatically generated with medium confidence

## Stream

Đôi khi cần xử lý dữ liệu mà không cần lưu tất cả record in collection à **stream** à phù hợp khi processing data

Stream are divided into two stream:

* **Intermediate**: *filter(), …* (return new stream)
* **Terminal operations**: *forEach(),…* (Mark the stream as consumed à after which point it can no longer be used further)



### Stream internals

<https://theboreddev.com/understanding-java-streams/>

Stream is lazily executed, so when a terminal operation gets executed à Stream selects an **execution plan**

* **Stateless operations: doesn’t need to know about any other element** to be able to emit a result (filter, map, flatMap)
* **Stateful operations**: **need to know about all the elements** before emitting a result (sorted, limit, distinct)

## Collection

## Collection vs Stream

|  |  |
| --- | --- |
| Collection | Stream |
| Mainly used to **store and group** the data | Mainly used to **perform operations** on data |
| **Is** data structure | **Isn’t** data structure |
| **Can** add or remove elements | **Can’t** add or remove elements |
| Iterate **externally** | Iterate **internally** |
| can be **traversed multiple times** | Traverable **only one** |
| **Eagerly construction** (all the elements are computed at the beginning) | **Lazy construction** (imtermediate operations are not evaluated until terminal operation is invoked) |
| Ex: List, Set, Map, … | Ex: filtering, mapping, matching, … |

# Collections, hashmap, hashset

## Collections

## Java Collection Framework Hierarchy HashMap vs HashSet

|  |  |
| --- | --- |
| HashMap | HashSet |
| Thuộc nhánh map nên sẽ có các đặc điểm của map | Thuộc nhánh set nên sẽ có các đặc điểm của set |
| Sử dụng **nested class Node<K,V>** để lưu trữ map entries. Lưu trữ thêm phần tử **next** (lưu địa chỉ của cặp <key,value > tiếp theo , **hash** lưu trữ hashcode của key   * Dùng cấu trúc **hashtable** * **Same bucket** à BST (AVL/RB Tree) | Lưu value (~ key in map)    HashSet chính là HashMap với **key(tương ứng value trong HashSet) của HashMap và value là hằng số**. Từ hình có thể thấy mỗi lần tạo ra đối tượng HashSet đều tạo ra đối tượng map (HashMap) map có key (unique) sử dụng trong Hashset và **value là constant** |
| **Non- synchronized** (nhưng có thể làm đồng bộ bằng cách  Map m = Collections.synchronizedMap(hashMap); ) |  |
| Cho phép **1 null key** và **multiple null values** | Cho phép **1 null value** |
| **Không** duy trì thứ tự | **Không** duy trì thứ tự |
| Dùng mảng các node (bucket) để lưu trữ, mỗi node là một linkedlist |  |
| **O(1)** (get/put/remove/containsKey) |  |
|  |  |

Link HashMap: <https://medium.com/javarevisited/internal-working-of-hashmap-in-java-97aeac3c7beb#:~:text=Working%20Of%20hashmap%3A-,HashMap%20uses%20its%20static%20inner%20class%20Node,new%20entry%20can%20be%20added>.

Link HashSet:

## Example:

### HashMap

**Text

Description automatically generated**

### HashSet

Text

Description automatically generated

## lấy VD so sánh tốc độ khi add 10K-100K phần tử vào collection

# ArrayList vs LinkedList

|  |  |
| --- | --- |
| ArrayList | LinkedList |
| Là array nhưng tốn thêm chi phí duy trì dynamic length. Khi thêm sẽ check xem có chỗ để add. Nếu có thì add, Không thì sẽ mở rộng capacity thêm 50% (default capacity=10) | Doubly linked list |
| Tốn vùng nhớ liên tục | Chiếm vùng nhớ phân mảnh à tối ưu bộ nhớ |
| 0(1) for get  0(n) for add + chi phí duy trì dynamic length | O(1) for add  O(n) for get |

Text

Description automatically generated

Do array list phải duy trì dynamic length à tốn chi phí hơn linked list (doubly linked list)