**SOLID Principle:**

1) **Single responsibility principle:**

class should have one and only one **reason** to change

2) **Open for extension close for modification**

you should be able to extend a class behavior without modifying it.

operation example use, add operation calculate(a1,a2), new add operation, division etc

class added

3) **Linskov substitution**

Derived classes must be substitutable for their base classes.

example:

Vehicle -> startengine();

car --> startengine();

but bicycle doesn't have startEngine() method.// un necessary add

it will create a problem

We should be able to substitute child in place of parent object.

Vehicle -> VehicleWithEngine() , VehicleWithoutEngine().

Car with with and , bicycle without.

4) **Interface segregation**

Make fine grained interfaces that are client specific.

example:

Shape -> calculateArea(), calcualteVolume();

class should not implement extra method which is not relevant for them

(I)Two dimentional -> calculate , (I) TrheDementional -> calclateVolume();

square impl Two, cuboid imp th,two

5) **Dependency inversion.**

High level modules should not depend on low level modules, both should depend on

abstractions.

Abstractions should not depend on details, details should depend on abstraction

example

Desktop -> monitor,keyboard. Desktop() {this.monitor = new monitor; this.keyboard= keyboard;}

java 8 Feature:

=======================

stream: is used to process collection of objects.

is not a data structure instead it takes input from the collections, Arrays

or I/O channels.

streams don't change the original data structure, they only

segregate, apply and combine (SAC)

--------------------------------------

**volatile vs atomic**

flag = false; t-1 , t-2

core1, local cache, shared cache

if t-1 update flag to true then t-2 not get as it's visibility problem

**volatile** -- solved visibility problem

**atomic**:

int value =1

t-1,t-2 ,, value++;

incremented value should be in both thread but not work even if we change to volatile integer . it's possible is not visibility problem it's synchronization problem

Thread two different operation reading and writing.

compound operation.

solve problem, we can use **synchronize** second way use **atomicinteger**.

AtomicInteger value = new AtomicInteger(1);

value.increment();

=============================

Optional.**ofNullable**(nullName).orElse("John"):

**Microservice**:

PUT - update the content logically

POST - create the content

Delete - delete the resource

Safe method - Get, unsafe method- delete, post, put

**Idempotent**- put, delete

non-idempotent - post