2022 Java concepts

JTA/XA is a kind of system insurance against data corruption (and the resulting business losses). The most common use cases are:

* Processing JMS messages from a queue and inserting the results in a database: you don't want a crash to lose messages whose results are not yet stored in the database.
* Updating two or more legacy back-end systems in the same transaction
* We have to learn about algorithms and data structures in order to make sure that the running time of the application is going to be as efficient as possible.

The first one is the so called **single responsibility principle.**

Then we have the so called **open closed principle.**

Then the third solid principle is the so called **Liskov substitution principle.**

Then we have **interface segregation principle**, and finally we have this so called **dependency inversion**.

So all of these principles are extremely crucial in software design

interfaces and abstraction in the main are quite powerful tools to achieve loosley coupling,

S- single Responsibiltiy Principle- That every class in a computer program should have responsibility over a single part of that program's

Functionality which it should encapsulate.

 every single class is responsible for a single behavior, a single operation.

OK, then we have a violation checker.

So in this case, we are going to check whether that pair is a valid integer or not.

And finally, we have a given operation.

So why is it crucial?

Because, of course, it is the so-called loosely coupled software architecture.

We have managed to separated the behaviors in two different classes and different methods.

**Creational design patterns**

The next lectures are about creational design patterns. We will consider the following design pattern one by one:

* singleton pattern
* factory pattern
* builder pattern
* prototype pattern

Singleton- eager instantiation, lazy instantiation,

* + what if we have multiple threads, add synchronized keyword, but this will make things very very slow
* Another problem with this implementation that despite the fact that we use a private constructor, new
* instances can be instantiated with the help of reflection attack, which means that with the help of the reflection API, new instances can be instantiated despite the fact that we use a private constructor.

Solution – use database as enum-Because enums are compile time constants basically.

So it is going to be guaranteed that this instance is safe. So if we use something like this, then it is guaranteed that acquiring the instance is safe.

Top of Form

Enum instances are thread-safe by default

Bottom of Form

Factory Pattern- we should separate the static classes in the application from the classes that change

frequently when we instantiate a new class, for example, a dog class or some other animals, basically

we can encapsulate object creation in distinct classes.

These are called factory classes and we can instantiate the classes in these separate factory classes.

Animal dog= AnimalFactory.getDog();