**MultiThreading**

**Concurrency** is when two or more tasks can start, run, and complete in overlapping time periods. It doesn't necessarily mean they'll ever both be running at the same instant. For example, multitasking on a single-core machine.

Machine or software application executing multiple tasks -***concurrency***

**Parallelism** is when tasks literally run at the same time, e.g., on a multicore processor.

Running tasks exactly at the same time -***parallelism***

A **process** is a collection of code, memory, data and other resources. A **thread** is a sequence of code that is executed within the scope of the process. You can (usually) have multiple threads executing concurrently within the same process.

**Java SE supports concurrency through multithreading.**

Java SE supports multithreading programming through its basic and high-level API’s.

***Basic API’s*** : Thread and Runnable – all of the API’s are in java.lang package.

***High-Level-API’s*** : Executor Frameworks – all of the API’s are in the java.util.concurrent package.

**Thread**:

Thread class present in java.lang package.

Need to create a instance of this class to run an asynchronous task.

Thread class has a run method that defines the job of the thread spawned.

**Runnable** :

Runnable interface present in java.lang package.

Runnable interface is preferred over Thread class.

Runnable interface has a run method that defines the job of the thread spawned.

**States of the Threads** :

* Whenever we create a thread then the state of the thread is NEW.
* When we start the thread the state of the thread is RUNNABLE. Runnable means it is a indication given to JVM that this thread is ready to run.
* When the JVM executes the run method then then the state of thread is RUNNING.
* When the job is completed then the state of the thread is TERMINATED/DEAD.

**Limitations of Using Thread and Runnable** :

* Thread and Runnable API’s are sufficient for low-level and basic tasks.
* Developer responsible for all the threads for the system/application in addition to bussiness logic.
* Basic API’s need solid parallelism logic.

**Executor Framework** :

This framework provides standardized invocation, scheduling, execution and control of asynchronous tasks in parallel threads.

Composed of many classes and interfaces that provide creation and management of threads.

**Thread Pool** :

A thread pool helps mitigate the issue of performance by reducing the number of threads needed and managing their lifecycle. Essentially, threads are kept in the thread pool until they're needed, after which they execute the task and return the pool to be reused later.

Graphical user interface

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Executor Interface** :

Executor : an object that executes submitted tasks with a singe method execute(). Instead of new Thread(new Runnable()).start() , executor object could be used as : ***executor.execute(new Runnable())***

***ExecutorService Interface*** :

ExecutorService is a sub-interface of Executor. Manages lifecycle of all individual threads and also of the Executor. It provides submit method which accepts both Runnable and callable Objects. It helps to execute a collection of Runnable or callable objects. It helps to shuts down the executor.

**Java EE Concurrency**

Java EE application follows some important rules like :

* Java EE applications work within an underlying container or an application server.
* Containers provide runtime support for application components like EJB’s and servelts
* Java EE servers provide central source management.
* Application integrity is important.

So in this kind of environment it is excepted that a Thread should have access for enterprice services like messaging services etc. So contextual information of the containers should be provided to thread, contextual information is class loader information, java naming directory interface naming and security context.

In this case, if we tried to create threads that container don’t aware of contextual information accessble to threads. That’s why it is advisible that container to manage the threads.. so that why Java EE concurrency utilities were born.

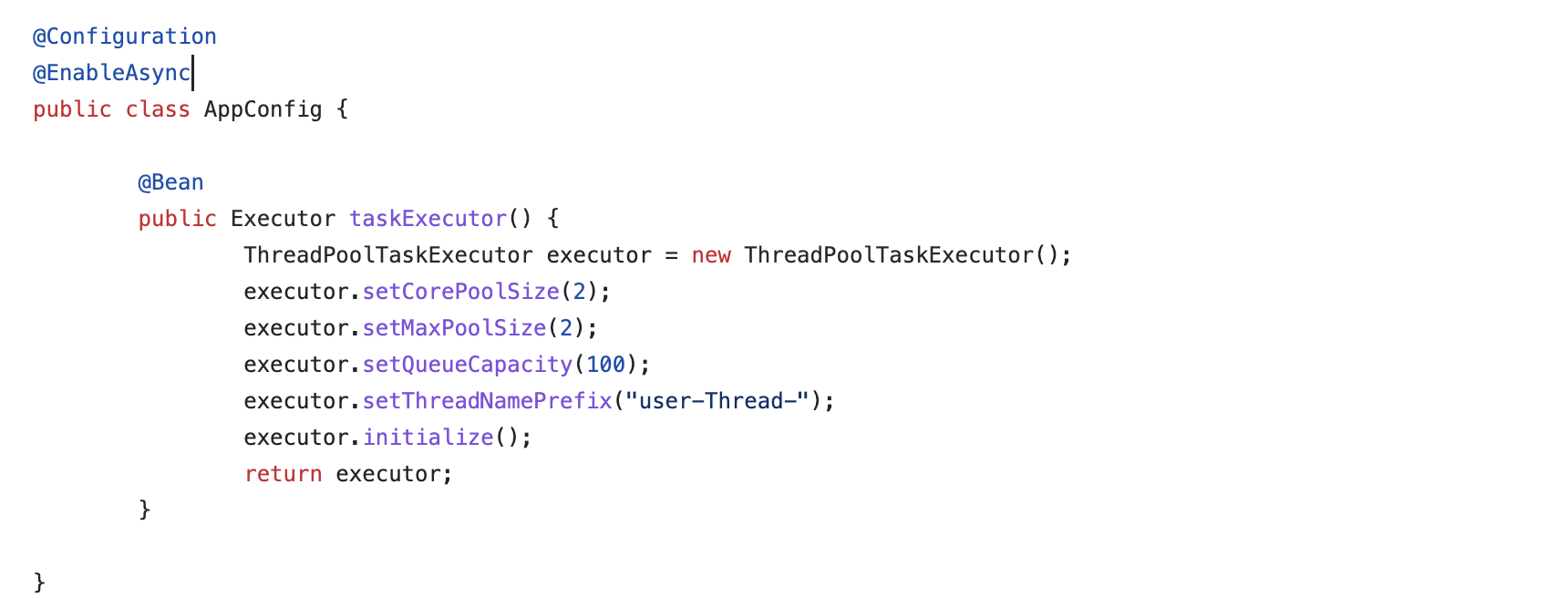
A picture containing graphical user interface

Description automatically generated

**Implementaion multi thread env in spring boot.**

Add a config file

In that file annotation @EnableAsync and create bean of Type Executor.



In service class to make a method as asynchros add @Async Annotation and return type should be CompletableFuture.



Controller class

