Menu

- Callable
- Future
- CompletableFuture
- CompletionStage

Callable

- Callable – chaqirish mumkin.
- Future – kelajak, kelgusi.

Callable

- In a case of Runnable job Thread will not return anything after completing the job.
- If a thread is required to return some result after executing the we should go for a Callable.
- Runnable/Thread holatida Thread o’z ishini tugatgandan keyin qiymat return qilmaydi.
- Agar Thread nimadir return qilishni hohlasa biz Callable interface dna foydalanishimiz kerak.

Callable interface

- Callable interface contains only one method.
  - public Object call() throws Exception
- If we submit a callable Object to execute the after completion job thread returns an object of type Future.
- That is future Object can be used to retrieve the result from callable job.

- Agar Callable ob’ektini ishlatib yuborsak, u ishini tugatganidan keyin Future ob’ekt retrurn qiladi.
- Future ob’ekti orqali callable orqali bajarilgan ishning natijasini olish mumkin.

Callable Example 1

- Callable class

```
public class MyCallable implements Callable {
    int num;

    public MyCallable(int num) {
        this.num = num;
    }

    @Override
    public Object call() throws Exception {
        int sum = 0;
        for (int i = 0; i <= num; i++) {
            sum = sum + i;
        }
        Thread.sleep(1000);
        return sum;
    }
}
```

Callable example 2

- Main Class which container Executor

```
public class MyMain {
    public static void main(String[] args) {
        MyCallable[] jobs = {new MyCallable(10),
                             new MyCallable(20),
                             new MyCallable(30)};
        ExecutorService service = Executors.newFixedThreadPool(2);

        for (MyCallable job : jobs) {
            Future<Integer> future = service.submit(job);
            System.out.println(future.get());
        }

        service.shutdown();
    }
}
```

Runnable vs Callable

- If a thread is not required to return anything after completion the job then we should go for Runnable
- If thread required retunr something after completing the job then we should go for Callable.
  - Runnable – public void run();
  - Callable – public void call() throwsException();
- A thread cannot be created using the Callable interface.
- A thread can be created using the Runnable interface.
- Agar thread o’z ishini tugatganidan keyin hech narsa return qilishi kerak bo’lmasa Runnable ishlatishimiz kerak.
- Agar thread o’z ishini tugatganidan keyin nimadir return qilishi kerak bo’lsa Callable ishlatishimiz kerak.
- Callable Interface orqali yangi Thread yaratib bo’lmaydi.
- Runnable interface orqali yangi Thread yatarish mumkin.

Callable links

- https://www.javatpoint.com/java-callable-example
- https://www.baeldung.com/java-runnable-callable

Future

Future

- In Java, **Future** is an interface that belongs to **java.util.concurrent** package. It is used to represent the result of an asynchronous computation. The interface provides the methods to check if the computation is completed or not, to wait for its completion, and to retrieve the result of the computation. Once the task or computation is completed one cannot cancel the computation.
- Once the asynchronous task completes, the result can be accessed via the Future object returned when the task was started.

- Javada Future interface ni **java.util.concurrent** da joylashgan. U asinxron hisoblash natijasini ifodalash uchun ishlatiladi. Interface hisoblashni tugaganini yoki bajarilayotganini bilish, uni tugashini kutib turish va hisoblashni natijasini olish uchun methodlar tagdim etadi. Vazifa yoki hisoblash tugallangandan so'ng, hisoblashni bekor qilish mumkin emas.
- Asynchronous ish tugaganidan keyin u boshlangan paytdan return qilgan Future ob’ekt dan olish mumkin.

Methods of the Future Interface

- The interface provides the following five methods:
- cancel() - It tries to cancel the execution of the task. Vazifaning bajarilishini bekor qilishga harakat qiladi
- get() - The method waits if necessary, for the computation to complete, and then retrieves its result. Agar kerak bo'lsa method hisoblashni fazifasini tugashini kutadi va natijasni oladi.
- isCancelled() - It returns true if the task was cancelled before its completion. Hisoblash tugashidan oldin bekor qilingan bo'lsa true return qiladi.
- isDone() - It returns true if the task is completed. Agar hisoblash tugagan bo'lsa true return qiladi.

Future Interface Shortcoming

- There was some shortcoming of the Future interface that are as follows:
  - Using Future, the computation cannot be completed manually.
  - It does not notify once the commutation is completed.
  - Its chain cannot be created and combined.
- Future interfeysida quyidagi kamchiliklar mavjud edi:
  - Future yordamida hisoblashni qo'lda yakunlab bo'lmaydi.
  - Kommutatsiya tugallangandan keyin u xabar bermaydi.
  - Uning zanjirini yaratish va birlashtirish mumkin emas.

- In order to overcome the above limitations, Java 8 introduced **CompletableFuture**.
- Shu muommolarni hal qilish uchun Java 8 dan CompletableFuture tagdim etildi.

Future Example 1

- Thread Class

```
public class MyThread implements Callable<Integer> {
    @Override
    public Integer call() throws Exception {
        Thread.sleep(1000);
        return ThreadLocalRandom.current().nextInt(100);
    }
}
```

Future Example 2

- Main

```
public class FutureDemoMain {
    public static void main(String[] args) throws ExecutionException, InterruptedException {
        ExecutorService service = Executors.newFixedThreadPool(2);

        Future<Integer> r1 = service.submit(new MyThread());
        System.out.println(r1.get());

        Future<Integer> r2 = service.submit(new MyThread());
        System.out.println(r2.get());

        service.shutdown();
    }
}
```

Future Links

- https://www.javatpoint.com/java-future-example
- https://www.baeldung.com/java-future
- https://jenkov.com/tutorials/java-util-concurrent/java-future.html
- https://howtodoinjava.com/java/multi-threading/java-callable-future-example/

CompletableFuture

CompletableFuture 1

- A **CompletableFuture** is used for asynchronous programming. Asynchronous programming means writing non-blocking code.
- It runs a task on a separate thread than the main application thread and notifies the main thread about its progress, completion or failure.
- In this way, the main thread does not block or wait for the completion of the task. Other tasks execute in parallel. Parallelism improves the performance of the program.
- A CompletableFuture is a class in Java. It belongs to java.util.cocurrent package. It implements CompletionStage and Future interface.

- CompletableFutue asinxron dasturlash uchun ishlatiladi. Asinxron dasturlash bloklanmaydigan kod yozishni anglatadi.
- U fazifani main(asosiy) Thread dan alohida boshqa bir Thread da ishlatadi va main Thread ga mofaqiyatli yoki hatolik bilan tugagani haqida habar beradi.
- Shunday qilib main Thread blocklanmaydi yoki Fazifasini tugashini kutmaydi. Boshqa task paraller ravishda ishlaydi. Parallellik dasturning ishlashini yaxshilaydi.
- CompaletableFuture Java classi dir. U java.util.cocurrent package da joylashgan. U CompletionStage va Future interface larini implements qilgan.

CompletionStage Interface

- java.util.concurrent.CompletionStage<T> interface represents a commutation task (either synchronous or asynchronous).
- As all methods declared in this interface return an instance of CompletionStage itself, multiple CompletionStages can be chained together in different ways to complete a group of tasks.
- java.util.concurrent.CompletionStage<T> interfeysi kommutatsiya vazifasini bajaradi (sinxron yoki asinxron thread lar uchun).
- Ushbu interfeysdagi barcha methodlar CompletionStage ni o’zini qaytaradi. Shu sababdan bir nechta CompletionStage vazifalar bitta guruhga bog’lash mumkin.

Using *CompletableFuture* as a Simple *Future*

- First of all, the *CompletableFuture* class implements the *Future* interface, so we can **use it as a *Future* implementation, but with additional completion logic**.
- For example, we can create an instance of this class with a no-arg constructor to represent some future result, hand it out to the consumers, and complete it at some time in the future using the *complete* method. The consumers may use the *get* method to block the current thread until this result is provided.
- Javada *CompletableFuture* classi Future interface dan nasil olgan shu sababdan uni Future ni realizatsiyasi sifatida va qo'shimcha tugatish mantig‘ini tag’dim etadigan class sifatida ko’rsak bo’ladi.
- *CompletableFuture* class dan ob’ekt yaratish uchun default konstructordan foydalansak bo’ladi. complete() metodi orqali
- Istemolchi get methodi orqali hozirgi thread ni block qilib natija kelishini kutadi.

CompletableFuture – supplyAsync()

- **supplyAsync():** It complete its job asynchronously. The result of supplier is run by a task from ForkJoinPool.commonPool() as default. The supplyAsync() method returns CompletableFuture on which we can apply other methods.
- U o'z ishini asinxron tarzda yakunlaydi. Bu ish (Task) ForkJoinPool.commonPool() tomonidan Yangi Thread da bajariladi va natija yetkazib beriladi. SupplementAsync() methodi CompletableFuture-ni qaytaradi, bunda biz boshqa usullarni qo'llashimiz mumkin.

CompletableFuture – supplyAsync() example

```
public static void main(String[] args) throws InterruptedException, ExecutionException {
        CompletableFuture<String> cf = CompletableFuture.supplyAsync(() -> getDataById(10));

        cf.get();
    }

    private static String getDataById(int id) {
        System.out.println("getDataById: " + Thread.currentThread().getName());
        return "Data:" + id;
    }
```

Result: getDataById: ForkJoinPool.commonPool-worker-1

CompletableFuture – **thenApply()**

- **thenApply():** The method accepts function as an arguments. It returns a new CompletableStage when this stage completes normally. The new stage use as the argument to the supplied function.
- Bu method argument sifatida function qabul qiladi. U bosqish tugaganidan keyin u yangi CompletableState (tuganlangna boshqish) return qiladi. Yangi boshqish yetkazib beruvchi funksiyasining argumenti sifatida ishlatiladi.

- **join():** the method returns the result value when complete. It also throws a CompletionException (unchecked exception) if completed exceptionally.
- Bu method Tugaganidan so’ng natijani return qiladi. Method CompletionException sodir qiladi aga ish hatolik bilan tugasa.

CompletableFuture - *completedFuture Method*

- **completedFuture()** is a static method of the CompletableFuture class and is used to get a new CompletableFuture that is in the completed stage, with the passed value as the result of the future.

- **completedFuture()** methodi CompletableFuture classining static methodi va u yakunlangan bosqich dagi Yangi CompletableFuture ob’ektini return qiladi. Berib yuborilgan qiymatni natija sifatida return qiladi.

CompletableFuture - *completedFuture Example*

- Example

```
public static void main(String[] args) throws ExecutionException, InterruptedException {
        CompletableFuture<String> cf = CompletableFuture.completedFuture("message");
        System.out.println(cf.get());

        CompletableFuture<Thread> cf2 = CompletableFuture.completedFuture(new Thread(){
            @Override
            public void run() {
                System.out.println("Inner Thread");
            }
        });

        cf2.get().start();
    }
```

Links

- https://www.concretepage.com/java/java-8/java-completablefuture-supplyasync
- https://dzone.com/articles/20-examples-of-using-javas-completablefuture
- https://www.logicbig.com/tutorials/core-java-tutorial/java-multi-threading/completion-stage-and-completable-future.html
- https://www.baeldung.com/java-completablefuture
- https://www.javatpoint.com/completablefuture-in-java
- https://www.callicoder.com/java-8-completablefuture-tutorial/

- **ReentrantReadWriteLock**
- ***StampedLock***

General Links

- https://www.youtube.com/watch?v=Ft8D_Toqa0k&list=PL786bPIlqEjRFPH8Z9IOwJWseG6Dq_Qxb&index=24