By Ayala Berkovich

Java 8

Concurrence

By Ayala Berkovich

CONCURRENCE(WORK IN PARALLEL) API IMPROVEMENTS

CONCURRENCE API from package ***'java.util.concurrent'*** support **asynchronous programming.**

There are two ways to create a thread:

1. By extending **Thread class**

```
class Multi extends Thread{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi t1=new Multi();
t1.start();
 }
}
```

2.By implementing **Runnable interface**

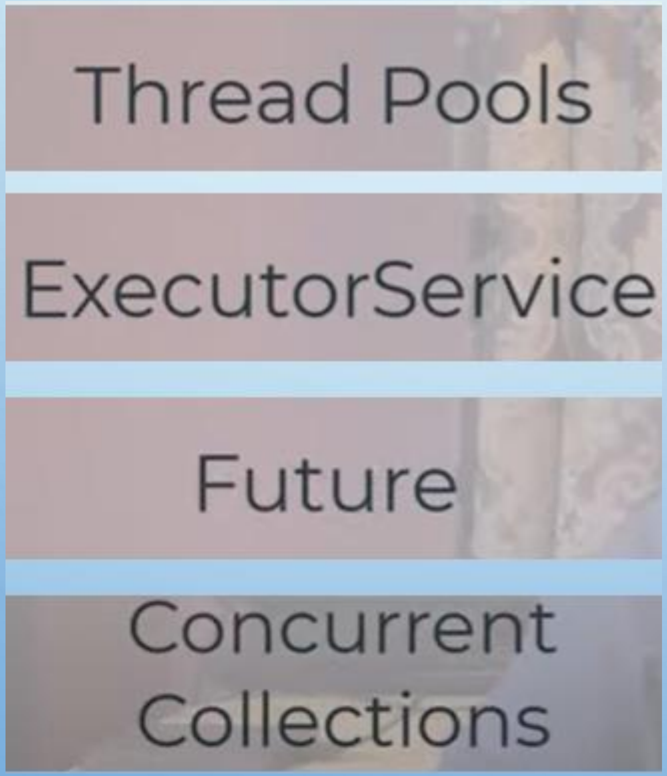
```
class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}

public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);
t1.start();
  }
}
```

By Ayala Berkovich

CONCURRENCE(WORK IN PARALLEL) API IMPROVEMENTS

In **Java5** , was added **Future** 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.

Was added:



Thread Pools

ExecutorService

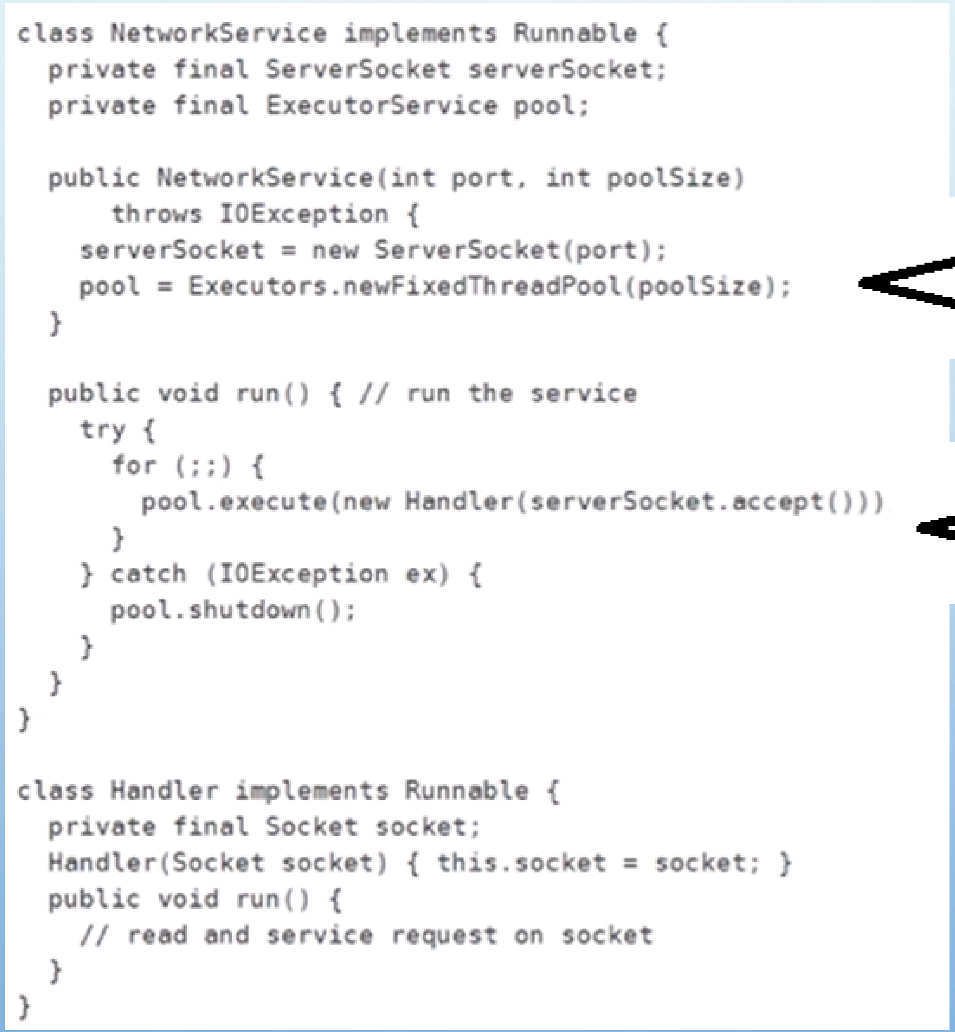
Future

Concurrent Collections

By Ayala Berkovich

CONCURRENCE(WORK IN PARALLEL) API IMPROVEMENTS

ExecutorService



```
class NetworkService implements Runnable {
  private final ServerSocket serverSocket;
  private final ExecutorService pool;

  public NetworkService(int port, int poolSize)
      throws IOException {
    serverSocket = new ServerSocket(port);
    pool = Executors.newFixedThreadPool(poolSize);
  }

  public void run() { // run the service
    try {
      for (;;) {
        pool.execute(new Handler(serverSocket.accept()));
      }
    } catch (IOException ex) {
      pool.shutdown();
    }
  }
}

class Handler implements Runnable {
  private final Socket socket;
  Handler(Socket socket) { this.socket = socket; }
  public void run() {
    // read and service request on socket
  }
}
```

create threads pool(define how many threads will be created)

Get thread from pool and execute

By Ayala Berkovich

CONCURRENCE(WORK IN PARALLEL) API IMPROVEMENTS

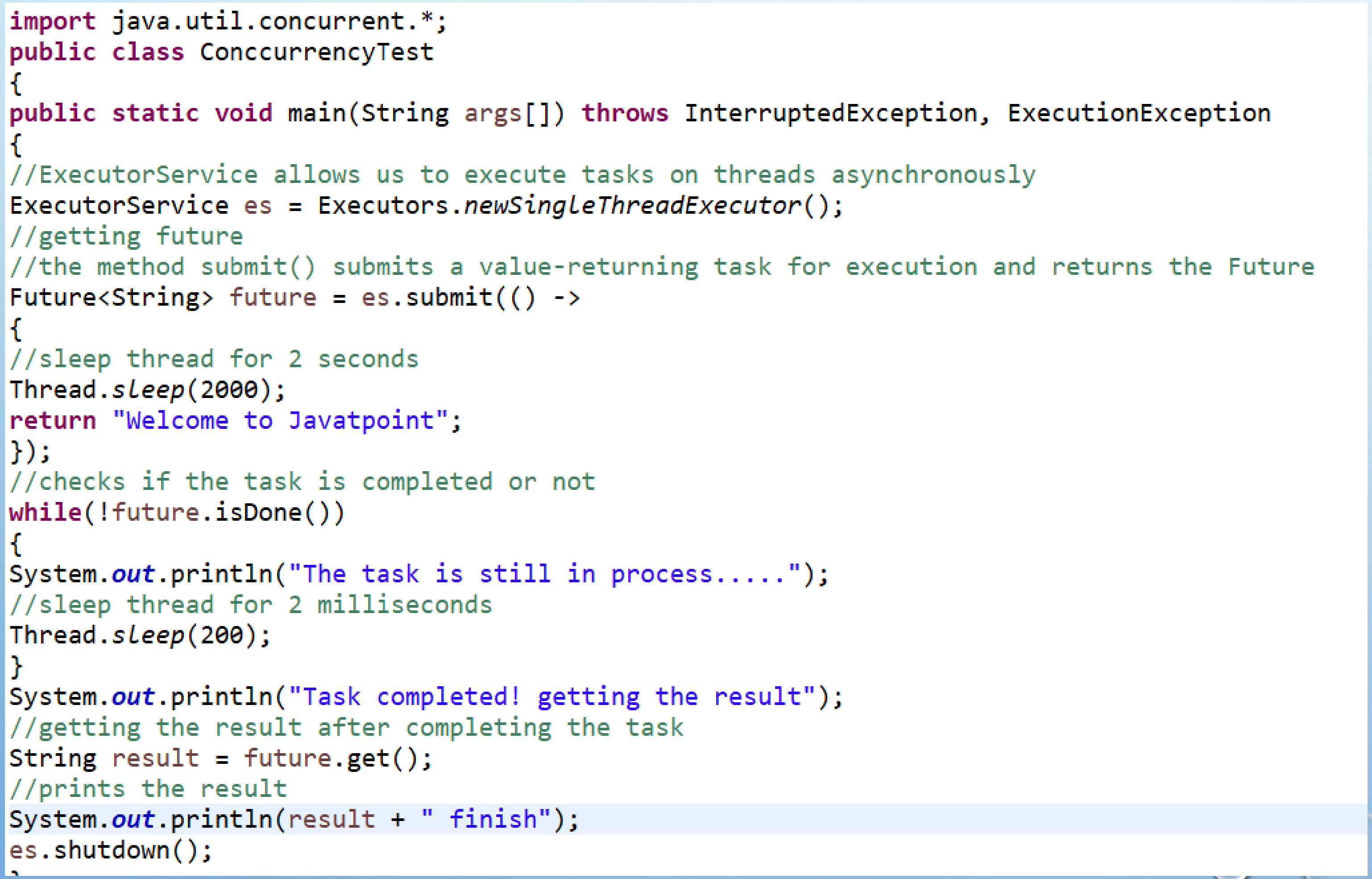
```
public interface Future<V>
```

- Can help to check result of thread working.

Method	Description
cancel()	It tries to cancel the execution of the task.
get()	The method waits if necessary, for the computation to complete, and then retrieves its result.
get()	Waits if necessary, for at most the given time for the computation to complete, and then retrieves its result, if available.
isCancelled()	It returns true if the task was cancelled before its completion.
isDone()	It returns true if the task is completed.

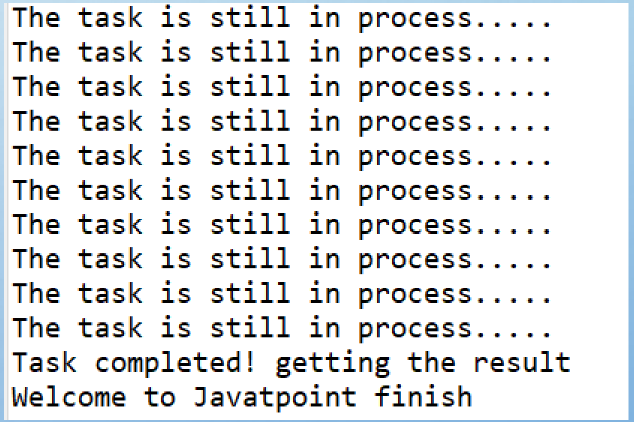
By Ayala Berkovich

CONCURRENCE(WORK IN PARALLEL) API IMPROVEMENTS



```java
import java.util.concurrent.*;
public class ConccurrencyTest
{
public static void main(String args[]) throws InterruptedException, ExecutionException
{
//ExecutorService allows us to execute tasks on threads asynchronously
ExecutorService es = Executors.newSingleThreadExecutor();
//getting future
//the method submit() submits a value-returning task for execution and returns the Future
Future<String> future = es.submit(() ->
{
//sleep thread for 2 seconds
Thread.sleep(2000);
return "Welcome to Javatpoint";
});
//checks if the task is completed or not
while(!future.isDone())
{
System.out.println("The task is still in process.....");
//sleep thread for 2 milliseconds
Thread.sleep(200);
}
System.out.println("Task completed! getting the result");
//getting the result after completing the task
String result = future.get();
//prints the result
System.out.println(result + " finish");
es.shutdown();
```

Output

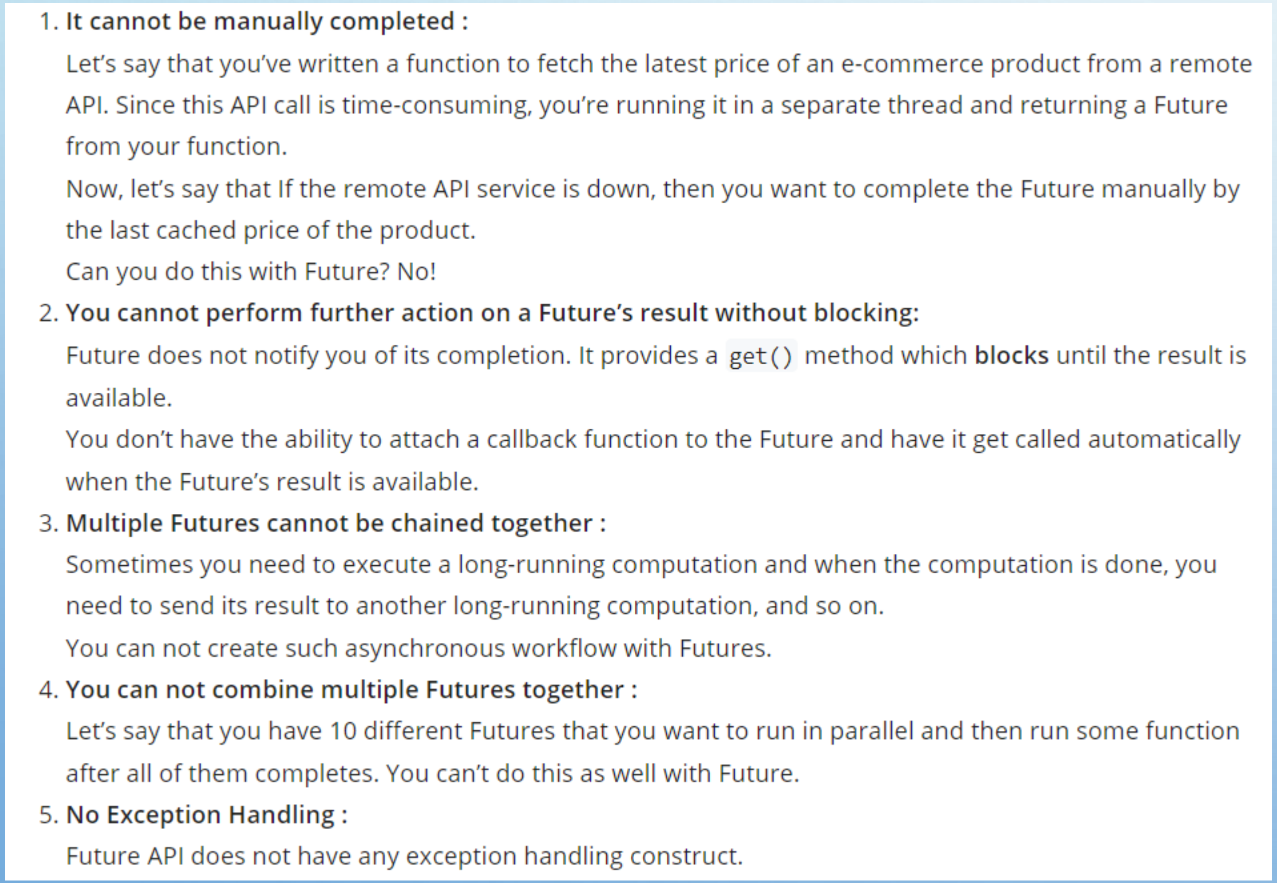


```
The task is still in process.....
The task is still in process.....
The task is still in process.....
The task is still in process.....
The task is still in process.....
The task is still in process.....
The task is still in process.....
The task is still in process.....
The task is still in process.....
The task is still in process.....
Task completed! getting the result
Welcome to Javatpoint finish
```

By Ayala Berkovich

CONCURRENCE(WORK IN PARALLEL) API IMPROVEMENTS

But in Future interface no option to:



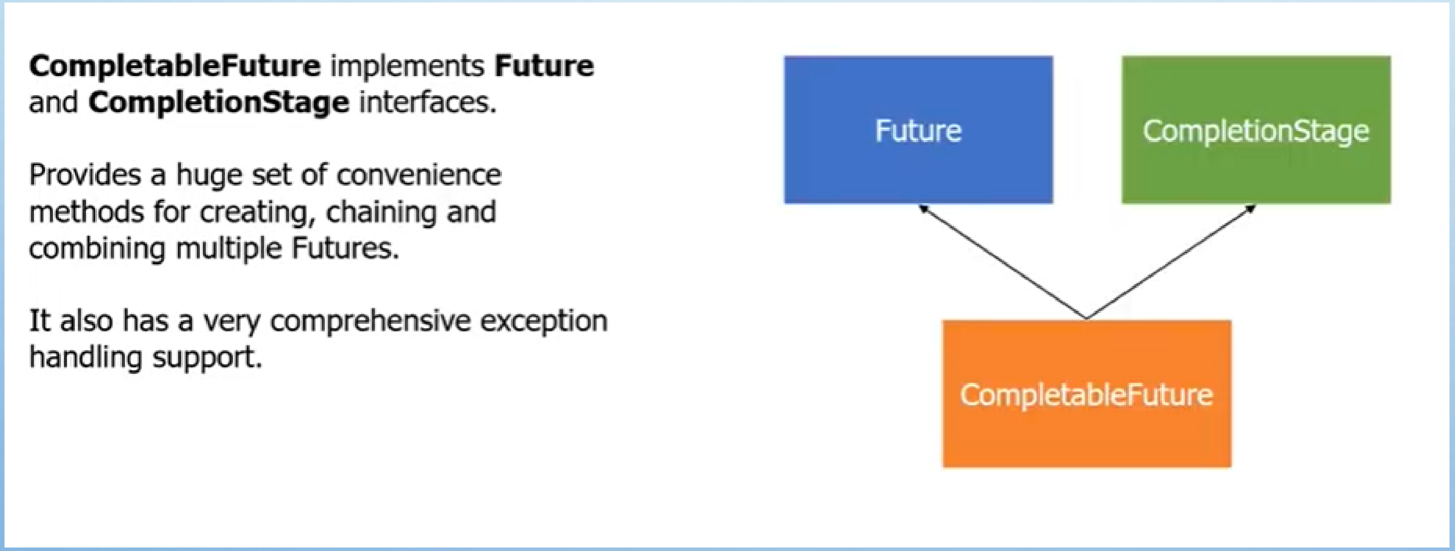
1. **It cannot be manually completed :**
   Let's say that you've written a function to fetch the latest price of an e-commerce product from a remote API. Since this API call is time-consuming, you're running it in a separate thread and returning a Future from your function.
   Now, let's say that If the remote API service is down, then you want to complete the Future manually by the last cached price of the product.
   Can you do this with Future? No!
2. **You cannot perform further action on a Future's result without blocking:**
   Future does not notify you of its completion. It provides a `get()` method which **blocks** until the result is available.
   You don't have the ability to attach a callback function to the Future and have it get called automatically when the Future's result is available.
3. **Multiple Futures cannot be chained together :**
   Sometimes you need to execute a long-running computation and when the computation is done, you need to send its result to another long-running computation, and so on.
   You can not create such asynchronous workflow with Futures.
4. **You can not combine multiple Futures together :**
   Let's say that you have 10 different Futures that you want to run in parallel and then run some function after all of them completes. You can't do this as well with Future.
5. **No Exception Handling :**
   Future API does not have any exception handling construct.

No Manual completion
Can not be Chained
Can not be Combined
No Exception Handling

By Ayala Berkovich

CONCURRENCE(WORK IN PARALLEL) API IMPROVEMENTS

In Java 8 was added new Class CompletableFuture to solve all problems of Future using:


CompletableFuture implements **Future** and **CompletionStage** interfaces.

Provides a huge set of convenience methods for creating, chaining and combining multiple Futures.

It also has a very comprehensive exception handling support.

By Ayala Berkovich

CONCURRENCE(WORK IN PARALLEL) API IMPROVEMENTS

You can create a **CompletableFuture** simply by using the following no-arg constructor

```
CompletableFuture<String> completableFuture = new CompletableFuture<String>();
```

All the clients who want to get the result of this **CompletableFuture** can call get() method:

```
String result = completableFuture.get()
```

This method blocks until the Future is complete.
So, the above call will block forever because the Future is never completed.

```
completableFuture.complete("Future's Result")
```

But we can use complete() method to manually complete a Future

By Ayala Berkovich

CONCURRENCE(WORK IN PARALLEL) API IMPROVEMENTS

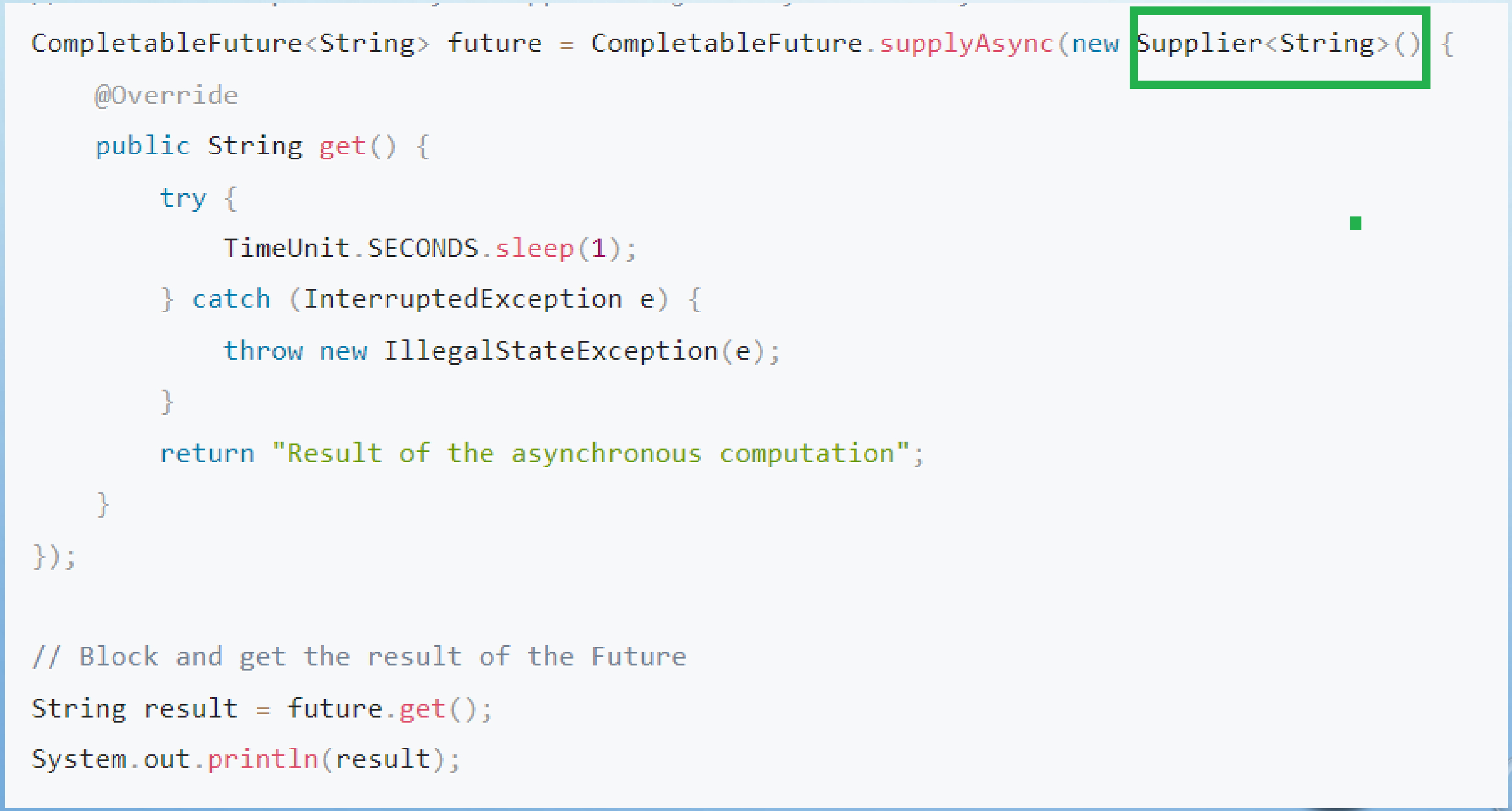
If you want to run **some background** task asynchronously and don’t want to return anything from the task, then you can use **runAsync()** function

```
CompletableFuture<Void> future = CompletableFuture.runAsync(new Runnable() {
    @Override
    public void run() {
        // Simulate a long-running Job
        try {
            TimeUnit.SECONDS.sleep(1);
        } catch (InterruptedException e) {
            throw new IllegalStateException(e);
        }
        System.out.println("I'll run in a separate thread than the main thread.");
    }
});

// Block and wait for the future to complete
future.get()
```

By Ayala Berkovich

CONCURRENCE(WORK IN PARALLEL) API IMPROVEMENTS

if you **want to return some result** from your background task you can use **supplyAsync():**



```
CompletableFuture<String> future = CompletableFuture.supplyAsync(new Supplier<String>() {
    @Override
    public String get() {
        try {
            TimeUnit.SECONDS.sleep(1);
        } catch (InterruptedException e) {
            throw new IllegalStateException(e);
        }
        return "Result of the asynchronous computation";
    }
});

// Block and get the result of the Future
String result = future.get();
System.out.println(result);
```

Supplier<T> is a simple functional interface which represents a supplier of results. It has a single get() method where you can write your background task and return the result.

By Ayala Berkovich

CONCURRENCE(WORK IN PARALLEL) API IMPROVEMENTS

If you don't want to return anything from your callback function and just want to run some piece of code after the completion of the Future, then you can use: **thenAccept()** or **thenRun()** functions:

```
// thenAccept() example
CompletableFuture.supplyAsync(() -> {
    return ProductService.getProductDetail(productId);
}).thenAccept(product -> {
    System.out.println("Got product detail from remote service " + product.getName())
});
```

have access to the Future's result.

Or:

```
// thenRun() example
CompletableFuture.supplyAsync(() -> {
    // Run some computation
}).thenRun(() -> {
    // Computation Finished.
});
```

doesn't even have access to the Future's result.

By Ayala Berkovich

CONCURRENCE(WORK IN PARALLEL) API IMPROVEMENTS

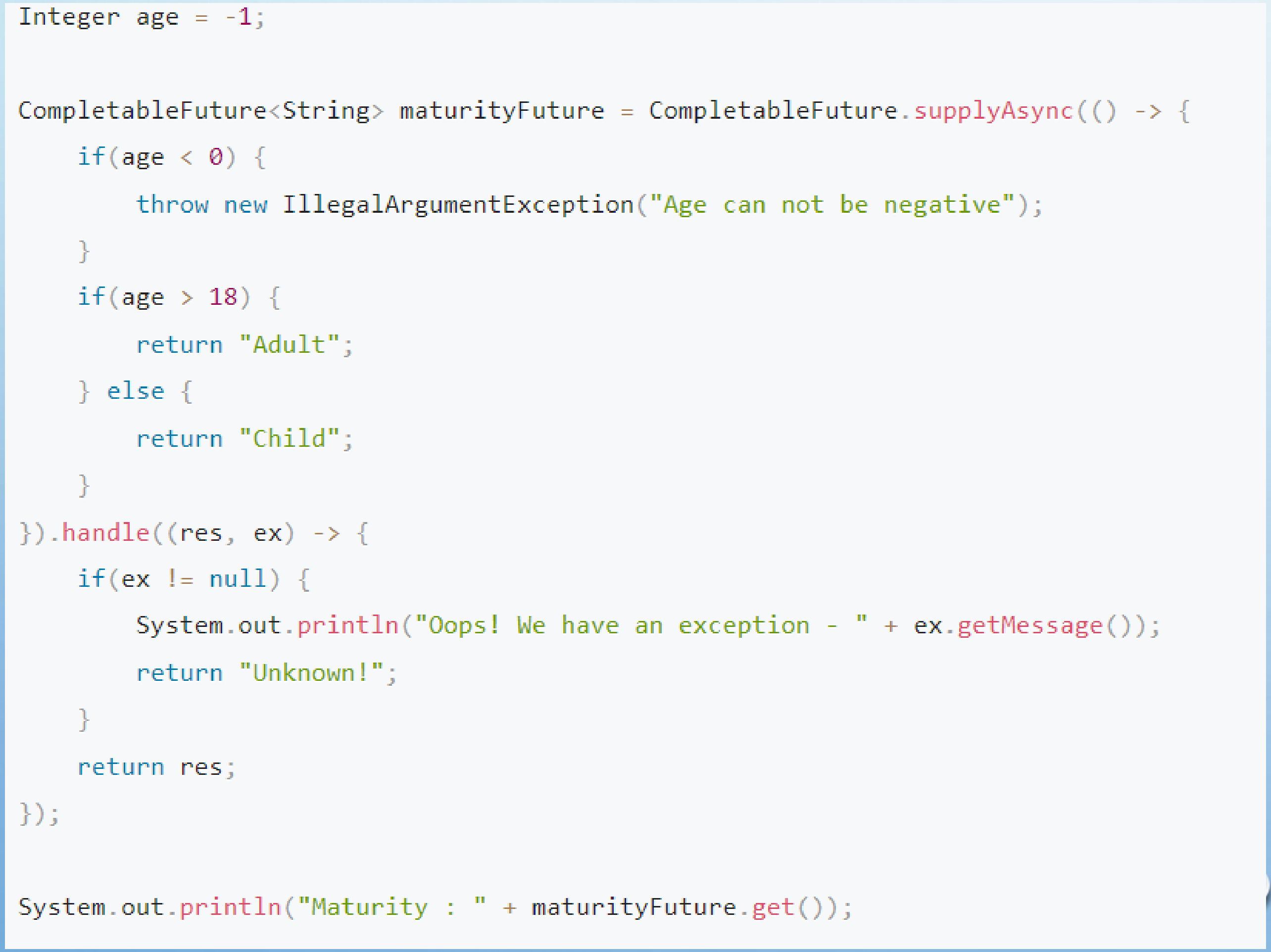
```java
System.out.println("Retrieving weight.");
CompletableFuture<Double> weightInKgFuture = CompletableFuture.supplyAsync(() -> {
    try {
        TimeUnit.SECONDS.sleep(1);
    } catch (InterruptedException e) {
       throw new IllegalStateException(e);
    }
    return 65.0;
});

System.out.println("Retrieving height.");
CompletableFuture<Double> heightInCmFuture = CompletableFuture.supplyAsync(() -> {
    try {
        TimeUnit.SECONDS.sleep(1);
    } catch (InterruptedException e) {
       throw new IllegalStateException(e);
    }
    return 177.8;
});

System.out.println("Calculating BMI.");
CompletableFuture<Double> combinedFuture = weightInKgFuture
        .thenCombine(heightInCmFuture, (weightInKg, heightInCm) -> {
    Double heightInMeter = heightInCm/100;
    return weightInKg/(heightInMeter*heightInMeter);
});

System.out.println("Your BMI is - " + combinedFuture.get());
```

thenCombine() is used when you want two Futures to run independently and do something after both are complete.

By Ayala Berkovich

CONCURRENCE(WORK IN PARALLEL) API IMPROVEMENTS

Handle exceptions using the generic **handle()** method



```java
Integer age = -1;

CompletableFuture<String> maturityFuture = CompletableFuture.supplyAsync(() -> {
    if(age < 0) {
        throw new IllegalArgumentException("Age can not be negative");
    }
    if(age > 18) {
        return "Adult";
    } else {
        return "Child";
    }
}).handle((res, ex) -> {
    if(ex != null) {
        System.out.println("Oops! We have an exception - " + ex.getMessage());
        return "Unknown!";
    }
    return res;
});

System.out.println("Maturity : " + maturityFuture.get());
```

It is called whether or not an exception occurs.

By Ayala Berkovich