Concurrency

ExecutorService

Overview

- *ExecutorService*
  - types of executor service
- *Callable<V>* (*Callable<V>* versus *Runnable*)
- *Future<V>*
- Code (for the above)
- Scheduling tasks; code

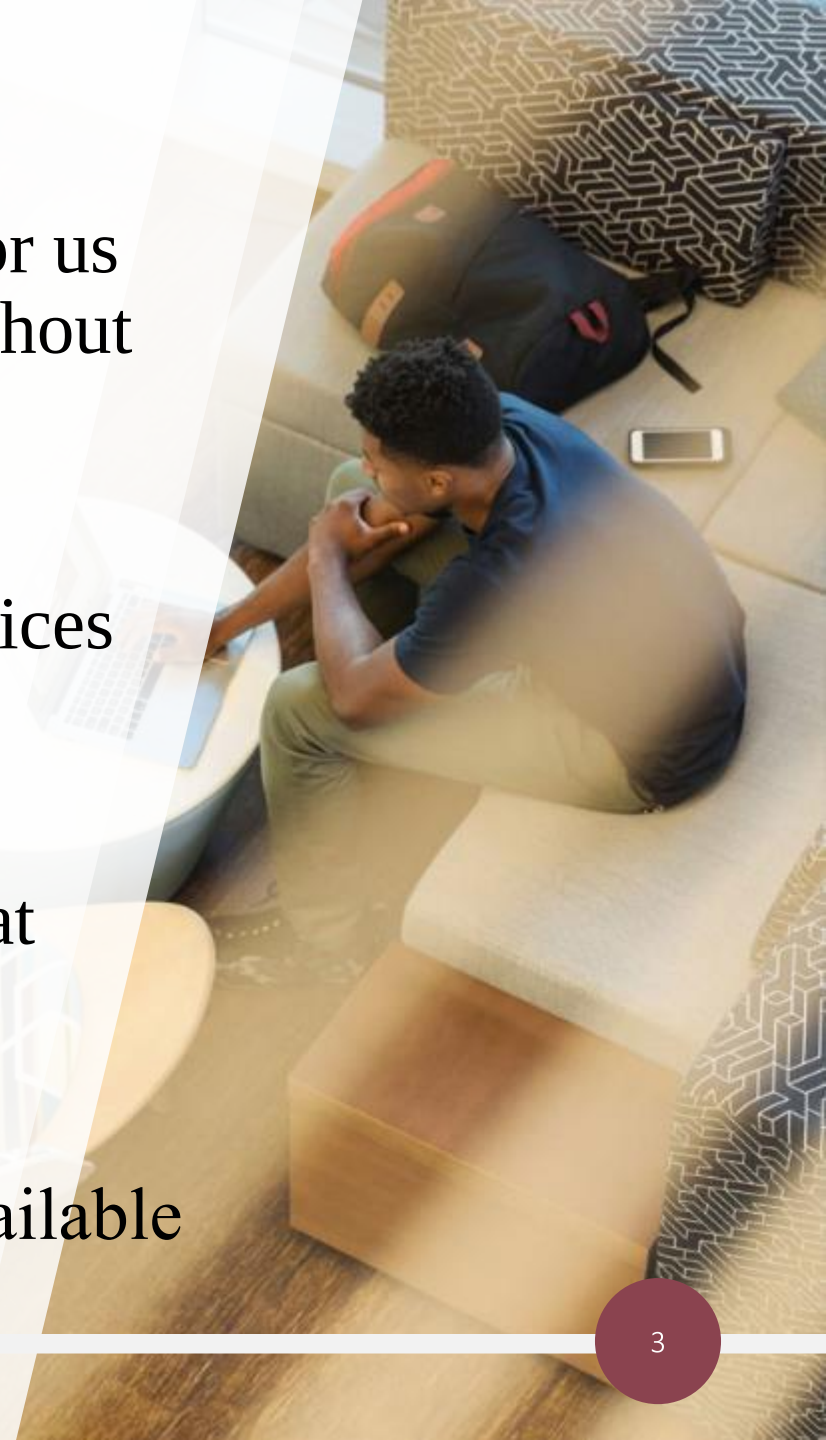
ExecutorService interface

- The Concurrency API abstracts thread management for us i.e. it enables complex processing involving threads without us having to manage threads directly.

- The *ExecutorService* is an interface that provides services for the creation and management of threads.

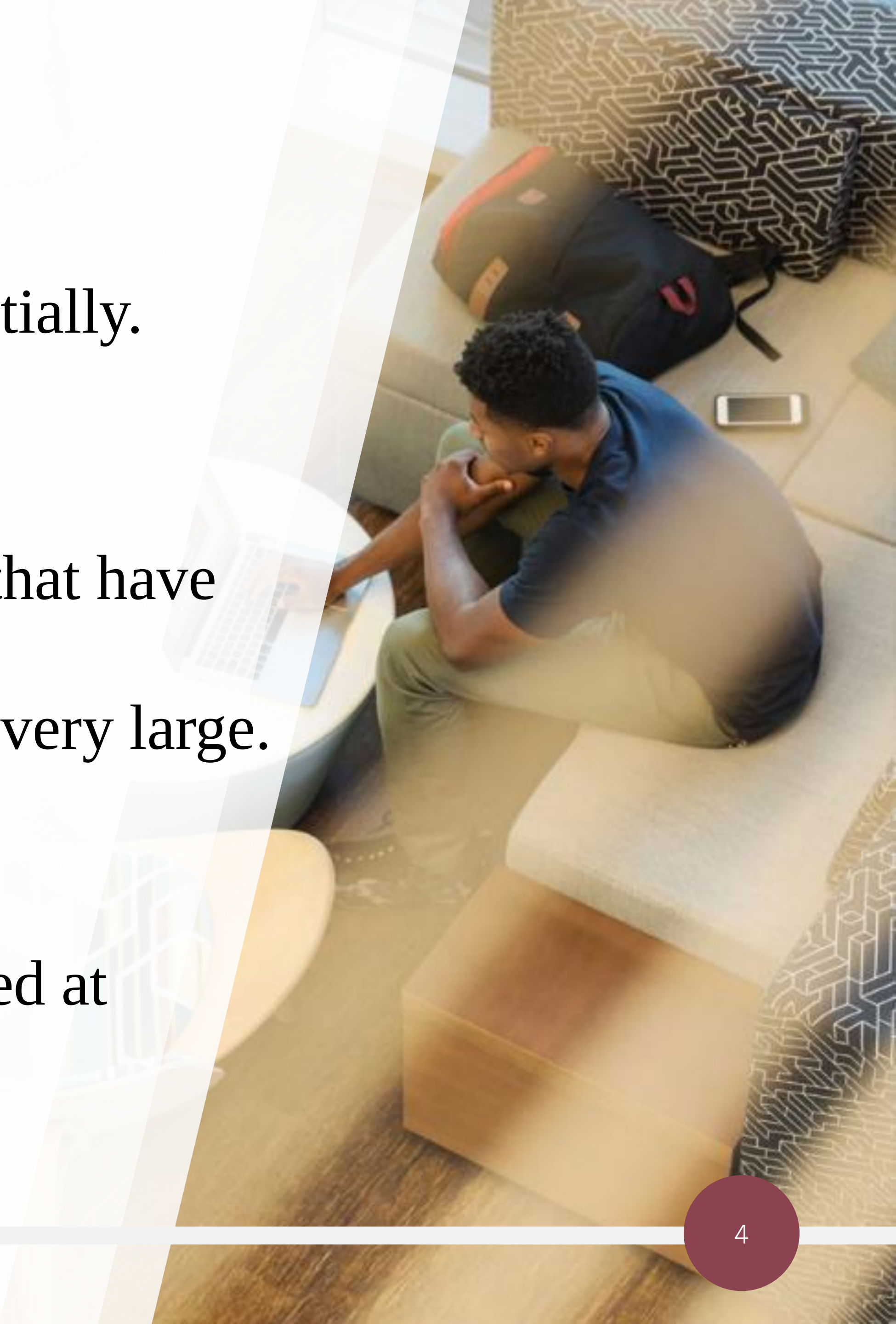
- The *Executors* utility class provides static methods that return *ExecutorService* implementation.

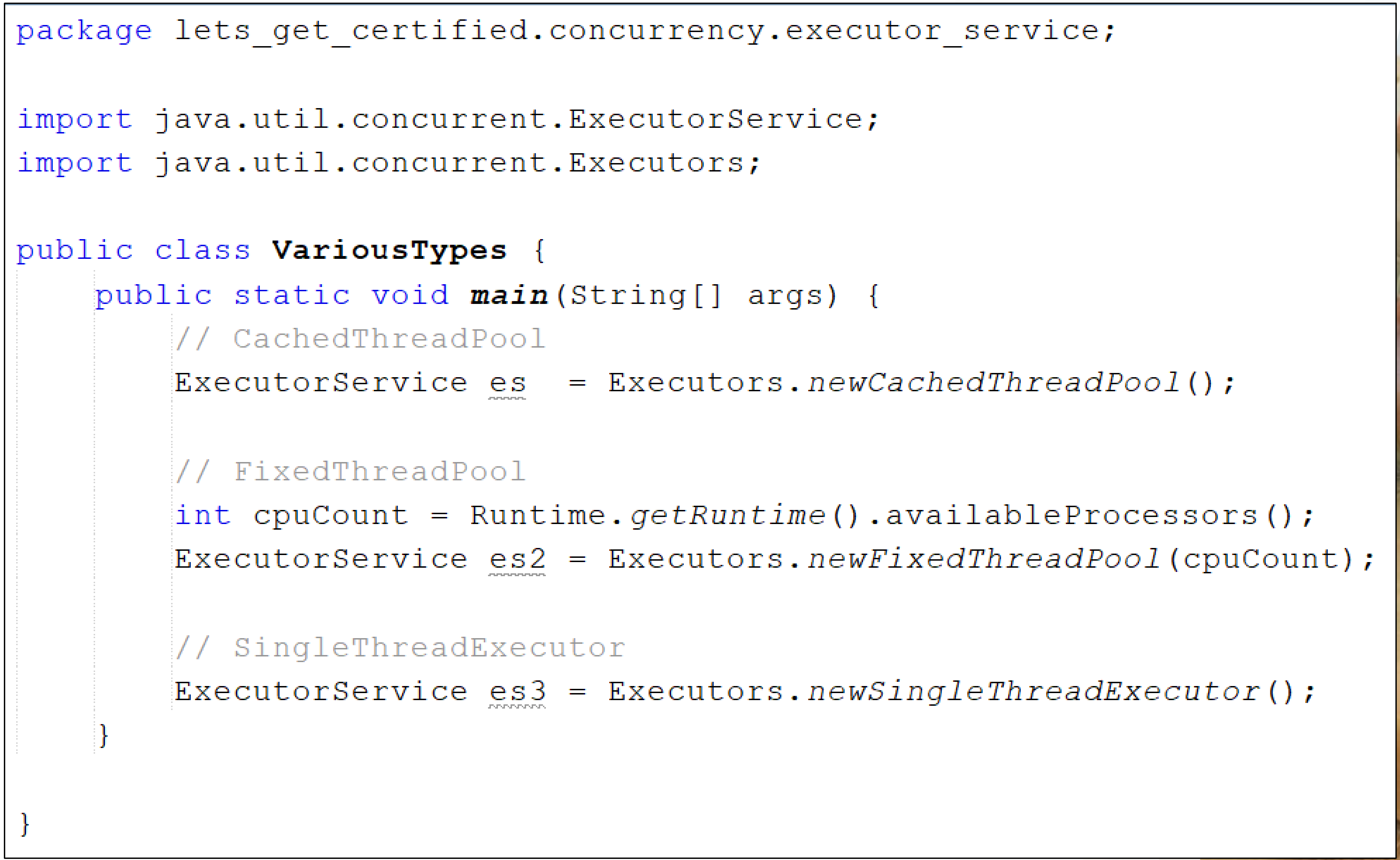
- A "thread pool" is a set of reusable worker threads available to execute tasks.



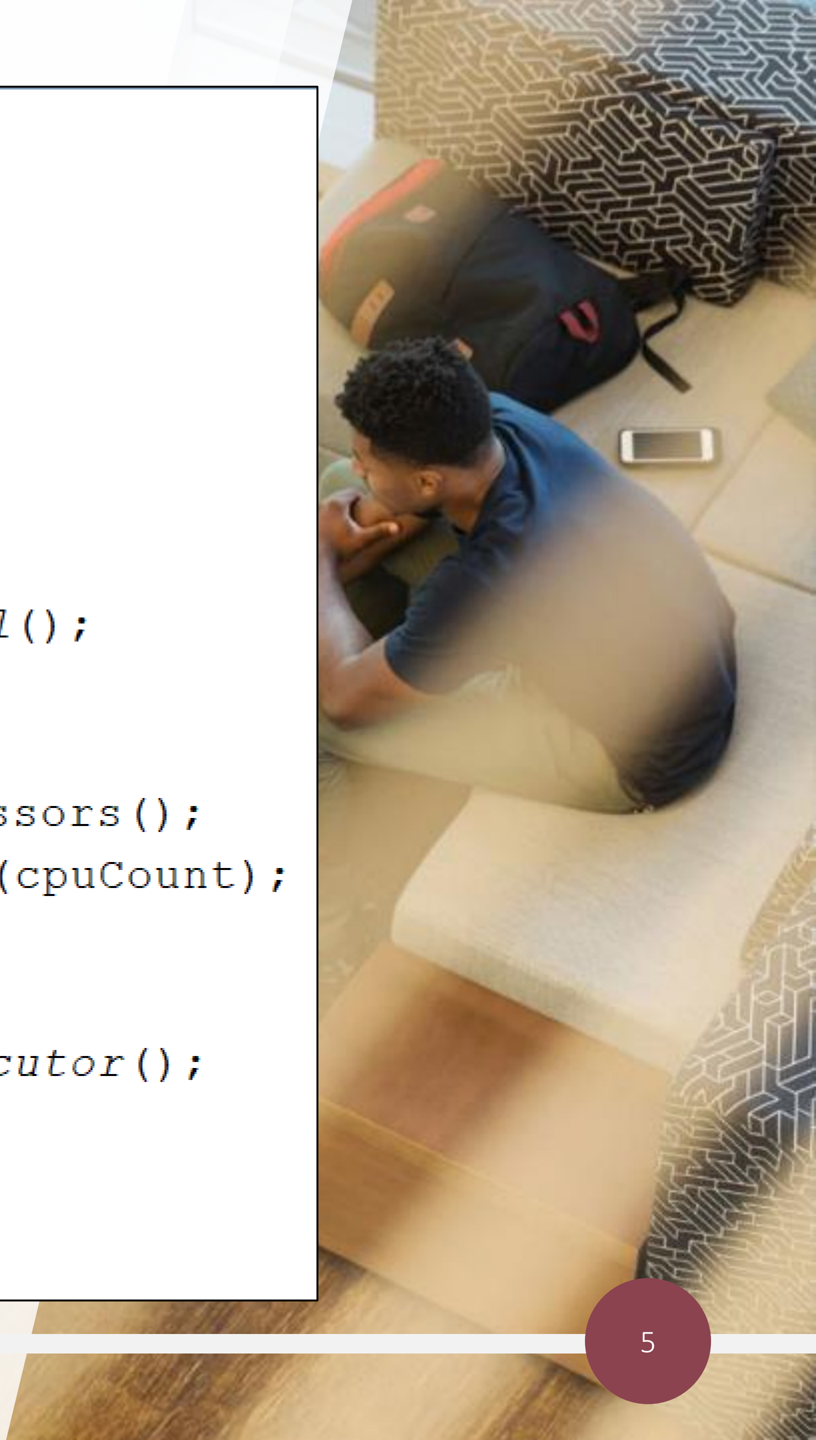
Types of *ExecutorService*

- Single thread pool executor
  - a single thread is used; tasks are processed sequentially.
- Cached thread pool executor
  - creates new threads as needed and reuses threads that have become free.
  - care needed as the number of threads can become very large.
- Fixed thread pool executor
  - creates a fixed number of threads which is specified at the start.





```java
package lets_get_certified.concurrency.executor_service;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class VariousTypes {
    public static void main(String[] args) {
        // CachedThreadPool
        ExecutorService es  = Executors.newCachedThreadPool();

        // FixedThreadPool
        int cpuCount = Runtime.getRuntime().availableProcessors();
        ExecutorService es2 = Executors.newFixedThreadPool(cpuCount);

        // SingleThreadExecutor
        ExecutorService es3 = Executors.newSingleThreadExecutor();
    }

}
```



Submitting tasks to an *ExecutorService*

- A *Callable<V>* is very similar to a *Runnable* except that a *Callable* can return a result and throw a checked exception.

	Runnable	Callable<V>
Asynchronous	Yes	Yes
Represents a task to be executed by thread	Yes	Yes
Functional interface	Yes	Yes
Functional method	void run()	**V call() throws Exception**
ExecutorService	void execute(Runnable) Future<?> submit(Runnable)	Future<T> submit(Callable<T>)

Future<V> interface

- A *Future<V>* is used to obtain the results from a *Callable*’s *call()* method.
- A *Future<V>* object represents the result of an asynchronous computation. Methods are provided to check if the computation is complete (*isDone()*) and to retrieve the result of that computation (*get()*).
- The result can only be retrieved using the method *V get()* when the computation has completed, blocking if necessary until it is ready.

Code:

RunnableTest.java, CallableTest.java, SubmittingTaskCollections.java

Scheduling tasks

- Executors exists that enable us to schedule tasks to be performed at some time in the future.

- In addition, tasks can be scheduled to occur repeatedly at a particular interval.

- To create scheduled executors, use the *Executors* utility class:
    - *ScheduledExecutorService newSingleThreadScheduledExecutor()*
    - *ScheduledExecutorService newScheduledThreadPool()*

Scheduling tasks

- The *ScheduledExecutorService* interface provides 4 methods to schedule tasks:
  - schedule(Runnable task, long delay, TimeUnit unit)
  - schedule(Callable<V> task, long delay, TimeUnit unit)
  - scheduleAtFixedRate(Runnable task, long initialDelay, long periodToWait, TimeUnit unit)
  - scheduleWithFixedDelay(Runnable task, long initialDelay, long delayBetweenEndOfOneAndStartOfNext, TimeUnit unit)

Code:

ScheduledExecutors.java