06. Retornos no Futuro

Date	@08/09/2022
Categoria	Java
Curso	Threads em Java 2: programação concorrente avançada

Tópicos

1. Comandos
2. Implementando
3. Submetendo tarefas
4. Retornos no Futuro
5. Runnable e Callable
6. De Runnable para Callable
7. A interface Future
8. O resultado futuro de uma ação!
9. E quando o futuro não chega?
10. Mãos à obra: Usando Callable
11. Mãos à obra: Juntando resultados
12. Para saber mais: FutureTask
13. Download do código fonte

Revisão

- Thread e Runnable
- ExecutorService - Pool de Threads
  - CachedThreadPool
  - FixedThreadPool
- thread.join()
- volatile e AtomicBoolean
- UncaughtExceptionHandler
- ThreadFactory

---

4. Retornos no Futuro

O que aprendemos?

- A interface `Callable` é uma *alternativa* à interface `Runnable`
  - As interfaces `Runnable` e `Callable` servem para *definir uma tarefa* de uma thread.
  - A diferença do `Callable` é que pode *retornar algo* e jogar *qualquer exceção*.
- Para usar um `Callable` com threads, sempre precisamos de um pool de threads (`ExecutorService`).
  - O método `submit(..)` do pool recebe um `Callable` retorna um `Future`
  - O `Future` representa o resultado da execução que talvez não tenha terminado ainda.

---

5. Runnable e Callable

O `Runnable` pode ser passado para uma thread:

```
Thread thread = new Thread(tarefa);// tarefa só pode implementar `Runnable`
```

O `Callable` precisa de um pool de threads (`ExecutorService`).

Todas as outras alternativas foram corretas:

- Ambas são interfaces.
- Ambas podem ser executadas com um pool (`ExecutorService`)
- Ambas representam uma execução para ser feita em paralela
- `Runnable` tem um método que retorna `void` e `Callable` um método que retorna um valor

---

12. Para saber mais: FutureTask

Temos duas interfaces para definir a tarefa de uma thread: a antiga `Runnable` e a mais recente `Callable`.

A diferença é que `Callable` possui um retorno e trabalha com exceções do tipo *checked*. Além disso, um `Callable` só pode ser executado através de um *pool de threads*.

Como já falamos, não se pode usar uma thread com um `Callable`! Veja o código abaixo que **não compila** pois o construtor da classe `Thread` só funciona com `Runnable`:

```
Callable<String> tarefa = new Tarefa();//Tarefa implementa Callable
new Thread(tarefa).start();//não compila!!
```

Isso significa que somos obrigados de usar um pool com `Callable`? E se no meu código não houver a possibilidade de usar um pool de threads?

Felizmente, há uma solução para resolver esse impasse que se chama FutureTask!

Veja o código:

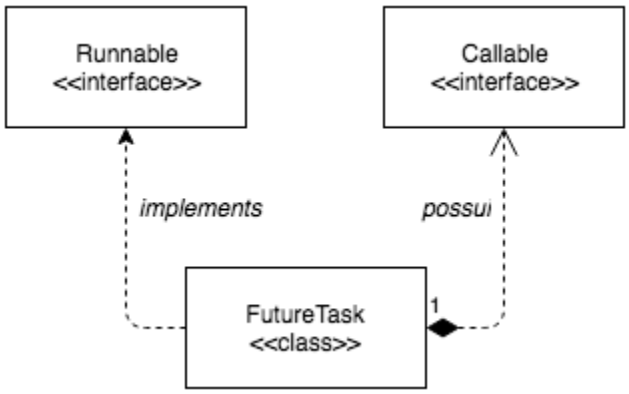
```
Callable<String> tarefa = new Tarefa();//Tarefa implementa Callable
FutureTask<String> futureTask = new FutureTask<String>(tarefa);
new Thread(futureTask).start();//usando Thread puro!!
String resultado = futureTask.get();
```

Isso funciona pois o `FutureTask` também é um `Runnable` (implementa a interface `Runnable`)! Repare também que o `FutureTask` recebe no construtor o `Callable` (a tarefa).

Podemos concluir que o `FutureTask` é um intermediário entre `Callable` e `Runnable`.

Apesar de o autor desse curso não ter conseguido confirmar através da documentação da classe, podemos considerar a classe `FutureTask` como sendo um *Adapter*.

Um *Adapter* é um *Design Pattern* do livro GoF que representa uma ponte ou intermediário entre duas interfaces incompatíveis.



Novamente, padrões de projetos estão em todo lugar e nós nem sempre percebemos quando estamos usando-os.