03

O método take

No último vídeo passamos rapidamente pelo código da API de chat. Esse código é avançado e se você ainda tiver uma ou outra dúvida, por favor, fique tranquilo, pois a apresentação sai um pouco do foco do curso.

A classe principal nesse projeto é a `PollingController`. Nela está implementada o recebimento e o envio das mensagens. Mostramos no vídeo que existem dois atributos nessa classe:

```
private BlockingQueue<Message> messages = new LinkedBlockingQueue<>();
private Queue<DeferredResult<Message>> clients = new ConcurrentLinkedQueue<>();
```

Repare que usamos implementações da interface `Queue` (*Fila*), pois as mensagens, assim como os clientes, ficam enfileiradas no lado do servidor. Além disso usamos implementações *Thread-Safe* (`LinkedBlockingQueue` e `ConcurrentLinkedQueue`). Isso significa que várias *Threads* podem acessar essas filas ao mesmo tempo de maneira segura.

Logo após a declaração dos atributos, vimos o método inicial da classe `PollingController`, que cria uma nova *thread* com um *loop* infinito (`while(true)`) para verificar a fila de mensagens:

```
@PostConstruct
public void init()  {
    new Thread(new Runnable() {
        @Override
        public void run() {
            while(true) {
                try {
                    Message message = messages.take();
                    sendToClients(message);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }).start();
}
```

Repare a linha que tira uma mensagem da fila:

```
Message message = messages.take();
```

O que aprendemos sobre o método `take` na aula?

Marque 1 alternativa

A O método `take` finaliza a *thread* atual.

B O método `take` é não bloqueante.

C O método `take` é bloqueante.

D O método `take` dispara uma nova *thread*.