Paradigmas de Programação
Relatório
Sistema Jukebox

Clarissa Simoyama David 21015712

10 de Dezembro de 2015

1 Sobre a Jukebox

Este projeto tem como objetivo simular um aparelho musical Jukebox, cuja funcionalidade consiste em um cliente escolher uma música e pagar por ela, tendo sua música escolhida reproduzida pela Jukebox. Nesta simulação, um cliente escolhe o número da música que está na playlist pré-determinada e, juntamente com o valor da música (estabelecido como 0.5), é feito o pedido para que ela possa ser tocada. Caso o valor dado no pedido seja menor diferente de 0.5, é imprimido uma mensagem de "Valor incorreto"e o programa passa para o próximo cliente. Para a implementação do código, fora utilizados alguns mecanismos do Scala, como *collections*, funções de alta ordem, *pattern matchings*, *case classes*, entre outros. Para programação concorrente, fora escolhida a abordagem feita por meio da trait de Atores. Nas seções seguintes será melhor explicado a utilização destas ferramentas.

2 Implementação da Jukebox

2.1 Case Classes

```
case class Selecionar(numMusica: Int)
case class Tocar(numMusica: Int, musica: Option[String])
case class Pedido(numMusica: Int, dinheiro: Double)
```

Figura 1: Case Classes

Para futura utilização de *pattern matchings* dentro dos atores Controlador e Tocador, foram criadas três *case classes*:

- Selecionar: recebe como parâmetro um Int chamado *numMusica*, que consiste no número da música escolhida pelo cliente;
- Tocar: recebe dois parâmetros: *numMusica* do tipo Int, que consiste no número da música que será tocada, e *musica* do tipo Option[String], que consiste na música que está dentro do Map, e que é escolhida através do número da música selecionada pelo cliente;
- Pedido: recebe dois parâmetros: *numMusica* do tipo Int, que consiste no número da música pedida, e *dinheiro* do tipo Double, que consiste no valor dado pelo cliente para que a música possa ser tocada pela Jukebox.

2.2 Ator Controlador

```
class Controlador extends Actor{
  private val playlist = Map[Int, String]()
  playlist += (1 -> "Queen - Bohemian Rhapsody")
  playlist += (2 -> "Pink Floyd - Another Brick In The Wall")
  playlist += (3 -> "Beatles - All You Need Is Love")
  playlist += (4 -> "Black Sabbath - I Won't Cry For You")
  playlist += (5 -> "Wesley Safadão - Camarote")
  val queue = Queue[Int]()
  def receive = {
    case Selecionar(m) => queue.enqueue(m)
      sender() ! Tocar(queue.dequeue, playlist.get(m))
  }
}
```

Figura 2: Ator Controlador

O ator Controlador, mais especificamente a classe Controlador que implementa a trait Actor, possui em seu construtor o Map (*collection* do Scala) *playlist*, em private, que possui a chave de Inteiros (no caso, o número atribuído a cada música) e é composto por Strings, com os títulos de cada música.

Em seguida, possui um sistema de fila de inteiros (*collection* Queue do Scala), que possui métodos como enfileirar (*enqueue*) e desenfileirar (*dequeue*).

No método *receive* do Controlador, temos o *case* Selecionar, da *case class Selecionar*, que recebe uma música m (da *case Pedido*, que será explicada mais para frente) e, se estiver de acordo com o padrão, m é enfileirado na fila *queue*. Por fim, o ator manda uma mensagem para o *case Tocar*, desenfileirando de *queue* a música que está em primeiro lugar na fila, e "puxando"do Map da playlist a música designada ao m.

2.3 Ator Tocador

```
class Tocador(servidor: ActorRef) extends Actor{
  private val bohemian = List("Is this the real life?", "Is this just fantasy?")
  private val brick = List("All in all you're just", "Another brick in the wall")
  private val beatles = List("All you need is love, love", "Love is all you need")
  private val black = List("So you lie awake and think about tomorrow",
                           "And you try to justify the things you've done")
  private val camarote = List("Agora assista aí de camarote", "Eu bebendo gela, tomando Cîroc")
  def receive = {
    case Pedido(t, d) => if(d == 0.5) servidor ! Selecionar(t) else println("Valor incorreto")
    case Tocar(m, r @ Some(t)) => println(s"$m - $t")
      if(m == 1) bohemian.foreach{println}
      else if(m == 2) brick.foreach{println}
      else if(m == 3) beatles.foreach{println}
      else if(m == 4) black.foreach{println}
      else camarote.foreach{println}
  }
}
```

Figura 3: Ator Tocador

O ator Tocador, mais especificamente a classe Tocador que implmenta a trait Actor, recebe como parâmetro o *servidor* do tipo *ActorRef*. Em seu construtor foram feitas algumas listas privadas de String, que possuem dois versos de cada música da playlist feita no ator Controlador.

Em seu método *receive*, temos dois *cases*:

- *Case Pedido*, da *case class Pedido*, que recebe uma música t de Int e o valor da música depositado d, do tipo Double. Há uma verificação, para checar se o valor colocado em d é o valor correto (no caso, de 0.5). Caso o valor esteja correto, é mandado uma mensagem para o *case* Selecionar, com a música t passada como parâmetro. Caso contrário, e o valor esteja incorreto, o cliente é notificado, e a fila é passada para o próximo cliente (e a próxima música).

- *Case Tocar*, da *case class Tocar*, recebe como parâmetro uma música m de Int e um item do Map da playlist, com o título da música. Caso o padrão necessário seja atingido, ele imprime o número da música e o título dela. Dependendo da numeração passada em m, é feito uma função de alta ordem *foreach*, na qual percorre a lista dos versos das músicas, as imprimindo na tela, simulando a música sendo reproduzida.

2.4 Método principal

No método principal (*main*), é feita as escolhas das músicas e suas respectivas designações para os clientes mandarem para a Jukebox.É mostrado uma mensagem com as músicas que estão nas playlists, e em seguida são pedidas as músicas que serão tocadas.

Após a escolha da música, é criado o *ActorSystem* do programa, com o nome de "Jukebox", para designação e para instanciar os clientes, sendo instanciado também o servidor como sendo o ator Controlador.

Em seguida, é feito o pedido de cada cliente, mandando mensagem para o *case Pedido*, com o número da música escolhida e o valor pago para que ela seja tocada. Entre cada cliente, é feito um comando de *Thread.sleep()*, para que a simulação fique um pouco mais próxima do real, tendo um tempo entre cada música tocada, além de forçar com que as músicas pedidas sejam colocadas e tiradas da fila em *Controlador* na ordem em que são pedidas.

Por fim, é feito o comando *system.shutdown()*, para "desligar"o *actor system*.

3 Trabalhos Futuros

Algumas implementações que seriam desejadas para a simulassão do Jukebox foram deixadas para serem trabalhadas futuramente. Uma delas é a criação de uma *trait* que consiste em não deixar que uma música seja tocada caso ela ainda esteja na fila do *Controlador*, para evitar repetições indesejadas.

Outra implementação deixada para ser trabalhada futuramente foi os tratamentos com os valores das músicas. No momento, os clientes que pedirem as músicas e darem exatamente 0.5 como passagem de valor terão suas músicas reproduzidas, e em caso de valor menor, será pulada a música e passada para a próxima. Uma implementação futura seria o retorno de troco para casos que o cliente dê dinheiro a mais, e devolução de dinhero caso o cliente dê menos do que o valor estipulado.