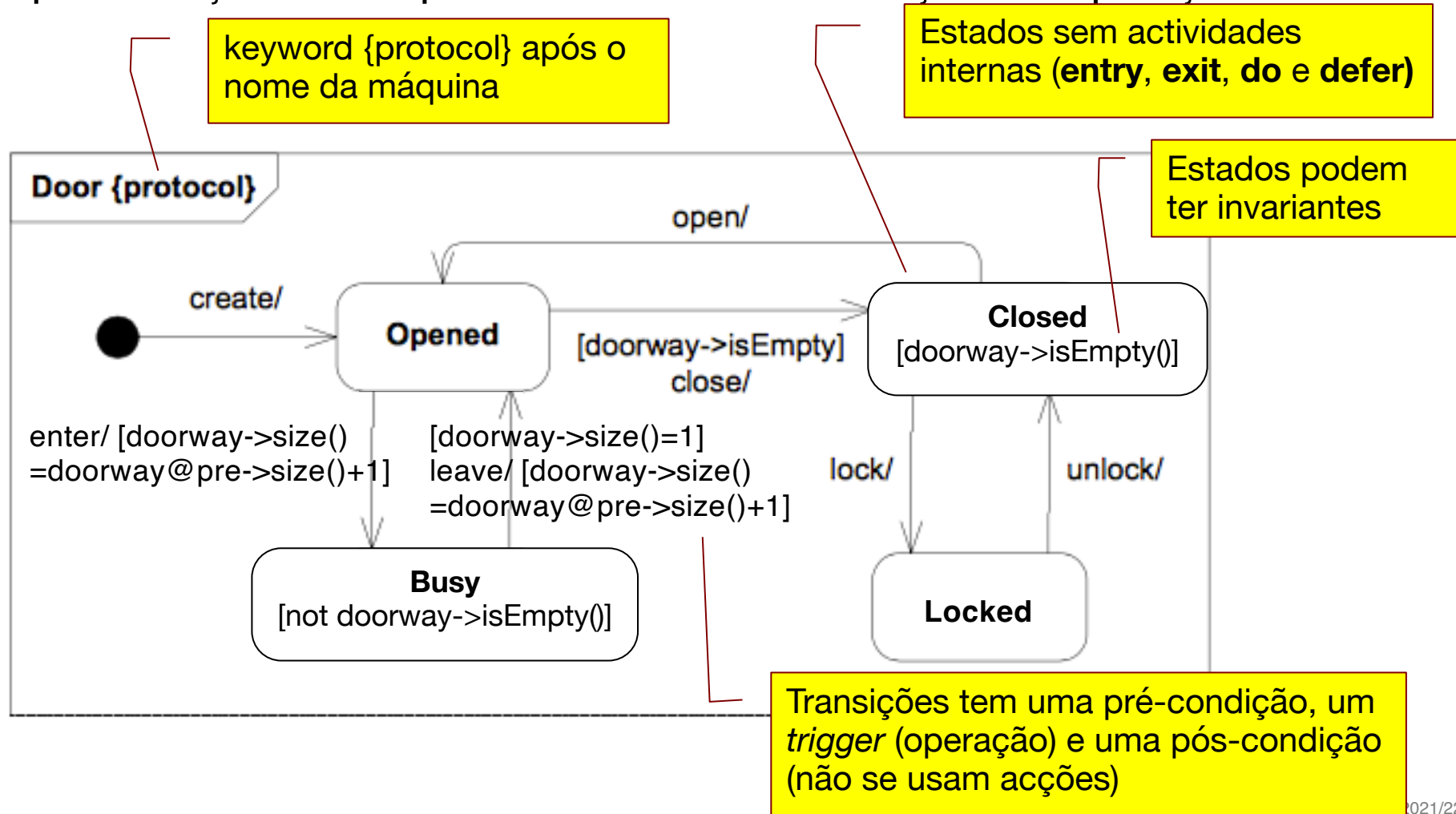




# Protocol State Machines

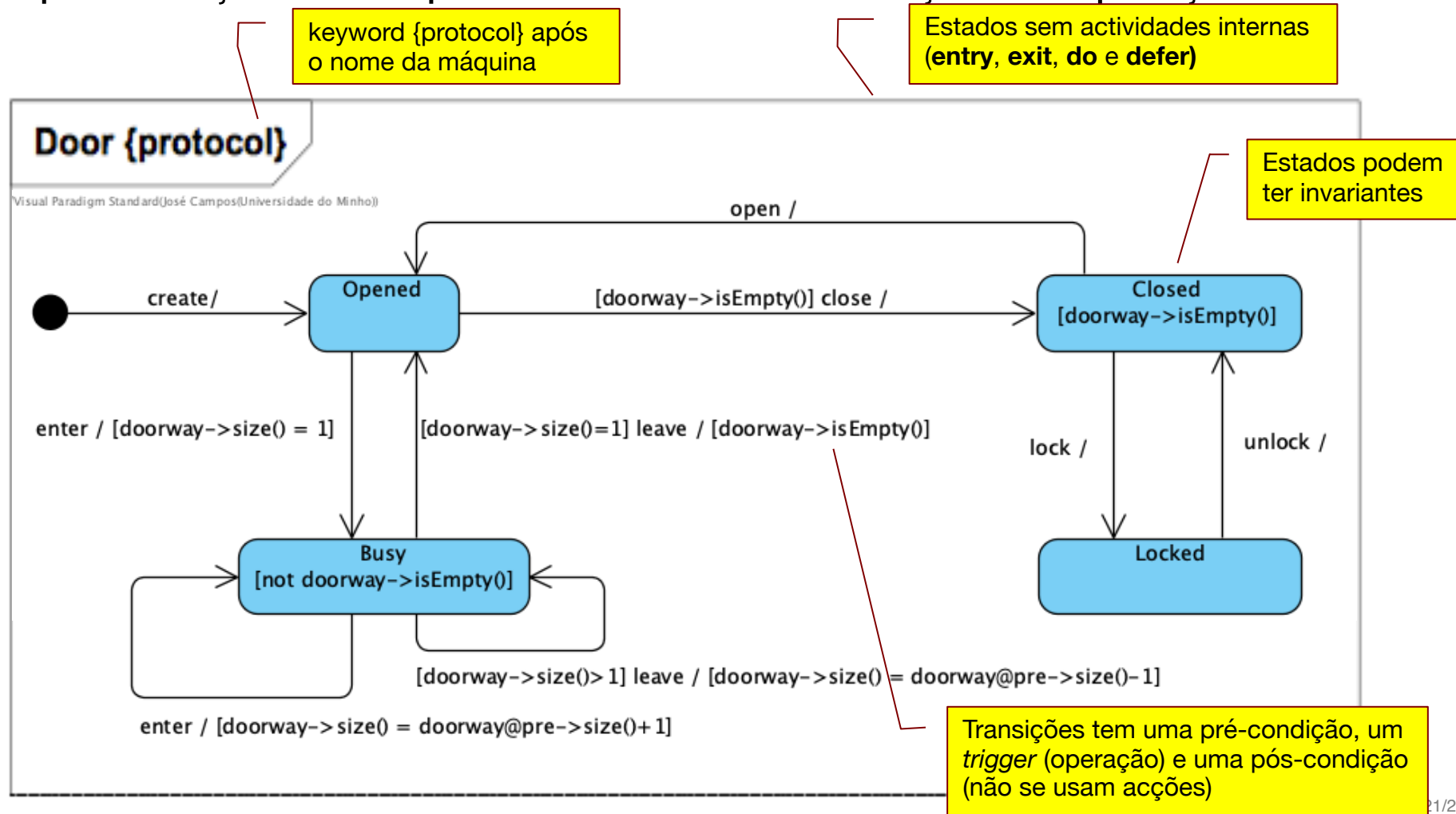
- Especificam que operações podem ser invocadas em cada estado e em que condições - a sequências válidas de invocação das operações.





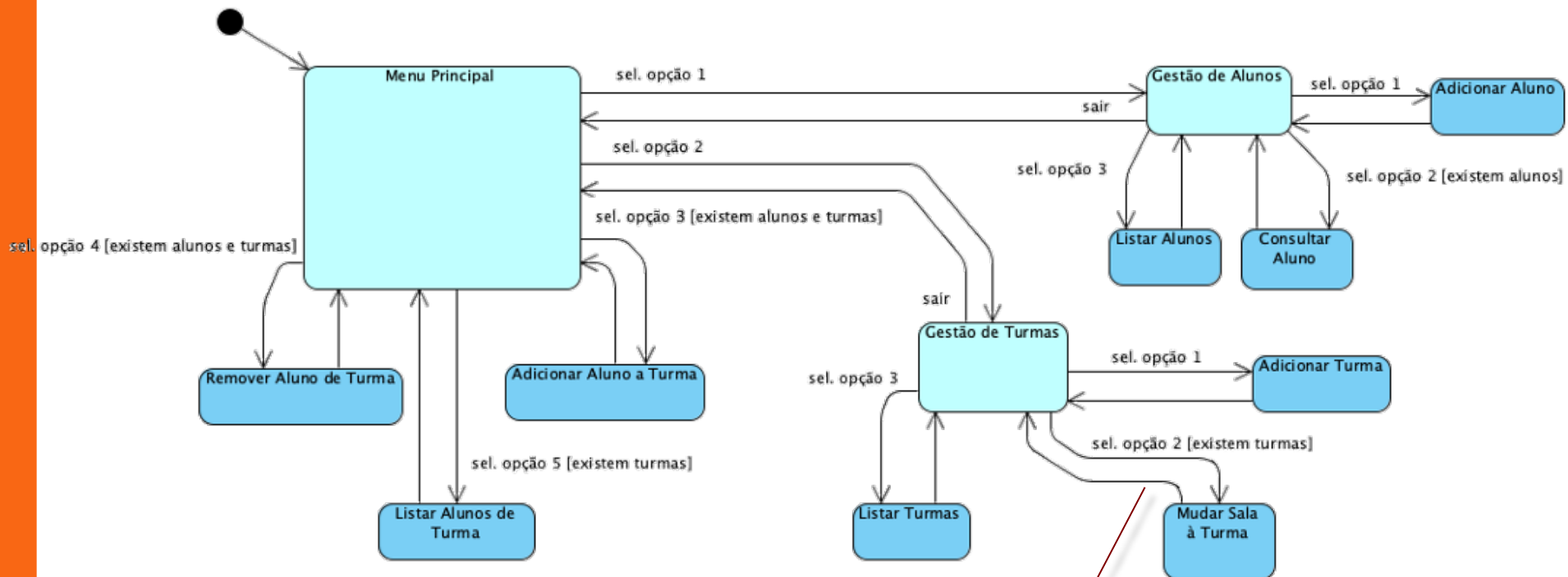
# Protocol State Machines

- Especificam que operações podem ser invocadas em cada estado e em que condições - a sequências válidas de invocação das operações.





# Modelação do controlo de diálogo da interface



```
public TextUI() {
    // Criar o menu
    this.menu = new Menu(new String[]{ "Adicionar Aluno", ...
    this.menu.setHandler(1, this::trataAdicionarAluno);
    this.menu.setHandler(2, this::trataConsultarAluno);
    this.menu.setHandler(3, this::trataListarAlunos);
    this.menu.setHandler(4, this::trataAdicionarTurma);
    this.menu.setHandler(5, this::trataMudarSalaTurma);
    this.menu.setHandler(6, this::trataListarTurmas);
```

*Completion transition*  
Não tem evento, dispara automaticamente logo que possível.



# Modelação do controlo de diálogo da interface

## Use Case: Reservar armação e lentes

**Descrição:** Funcionário regista uma reserva de armação e lentes

**Pós-condição:** Reserva fica registada

### Fluxo normal:

1. Funcionário indica nome e/ou data de nascimento do cliente
2. Sistema procura clientes
3. Sistema apresenta lista de clientes
4. Funcionário selecciona cliente
5. Sistema procura cliente
6. Sistema apresenta detalhes do cliente
7. Funcionário confirma cliente
8. Sistema procura produtos e apresenta lista
9. Funcionário indica Código de armação e lentes
10. Sistema procura detalhes dos produtos
11. Sistema apresenta detalhes dos produtos
12. Funcionário confirma produtos
13. Sistema regista reserva dos produtos
14. <<include>> imprimir talão

### Fluxo de excepção: [cliente não quer produto] (passo 12)

- 11.1. Funcionário rejeita produtos
- 11.2. Sistema termina processo

**Registo de receitas**

Nome Cliente:  Data Nasc.:

Armações... Lentes Esq./Dir.

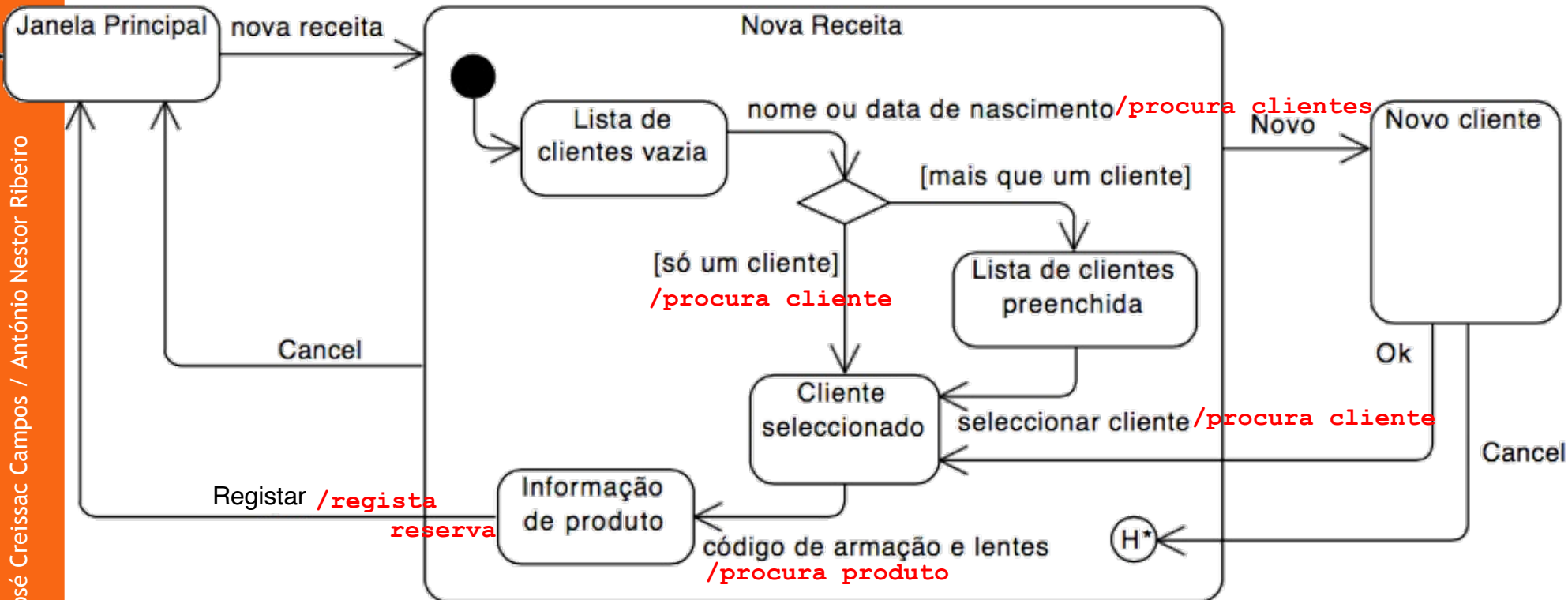
# Modelação do controlo de diá

**Registo de receitas**

Nome Cliente:  Data Nasc.:

---

Armações... Lentes Esq./Dir.



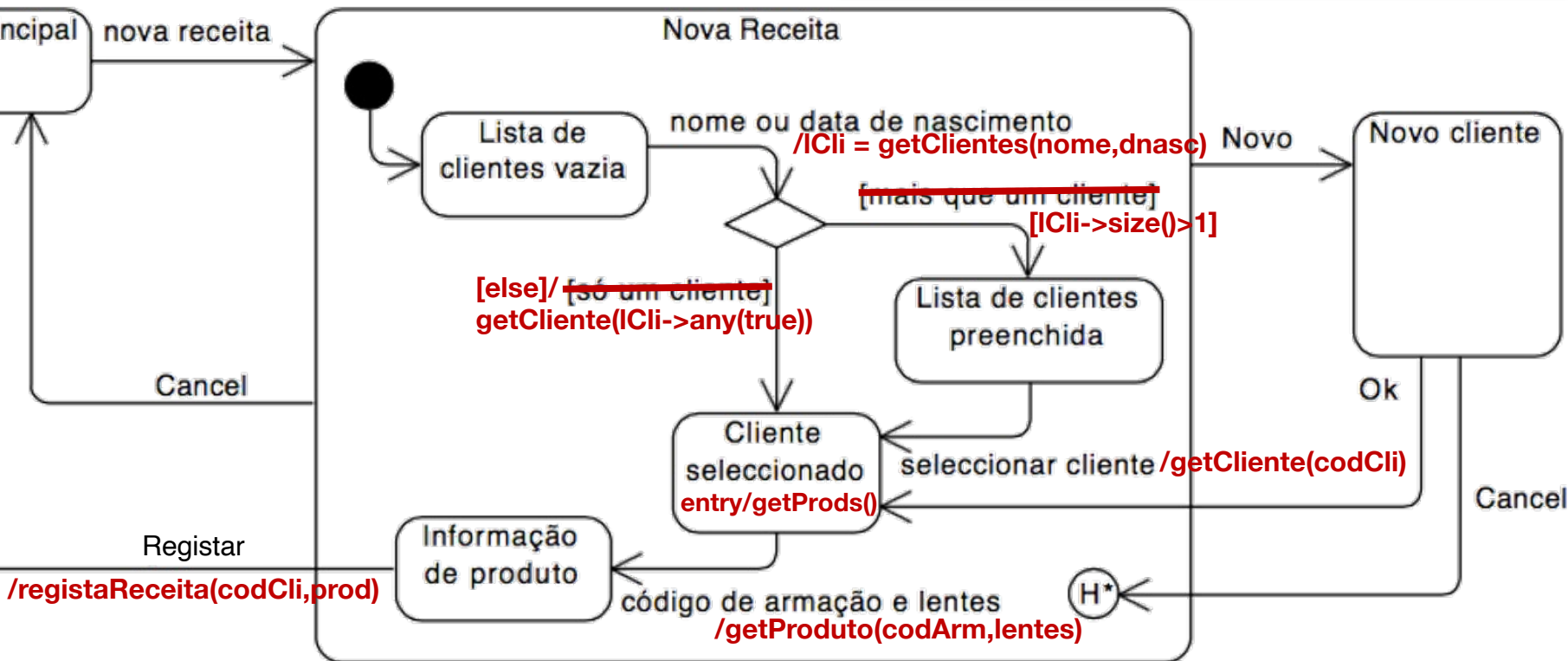
# Modelação do controlo de diá

Registo de receitas

Nome Cliente:  Data Nasc.:

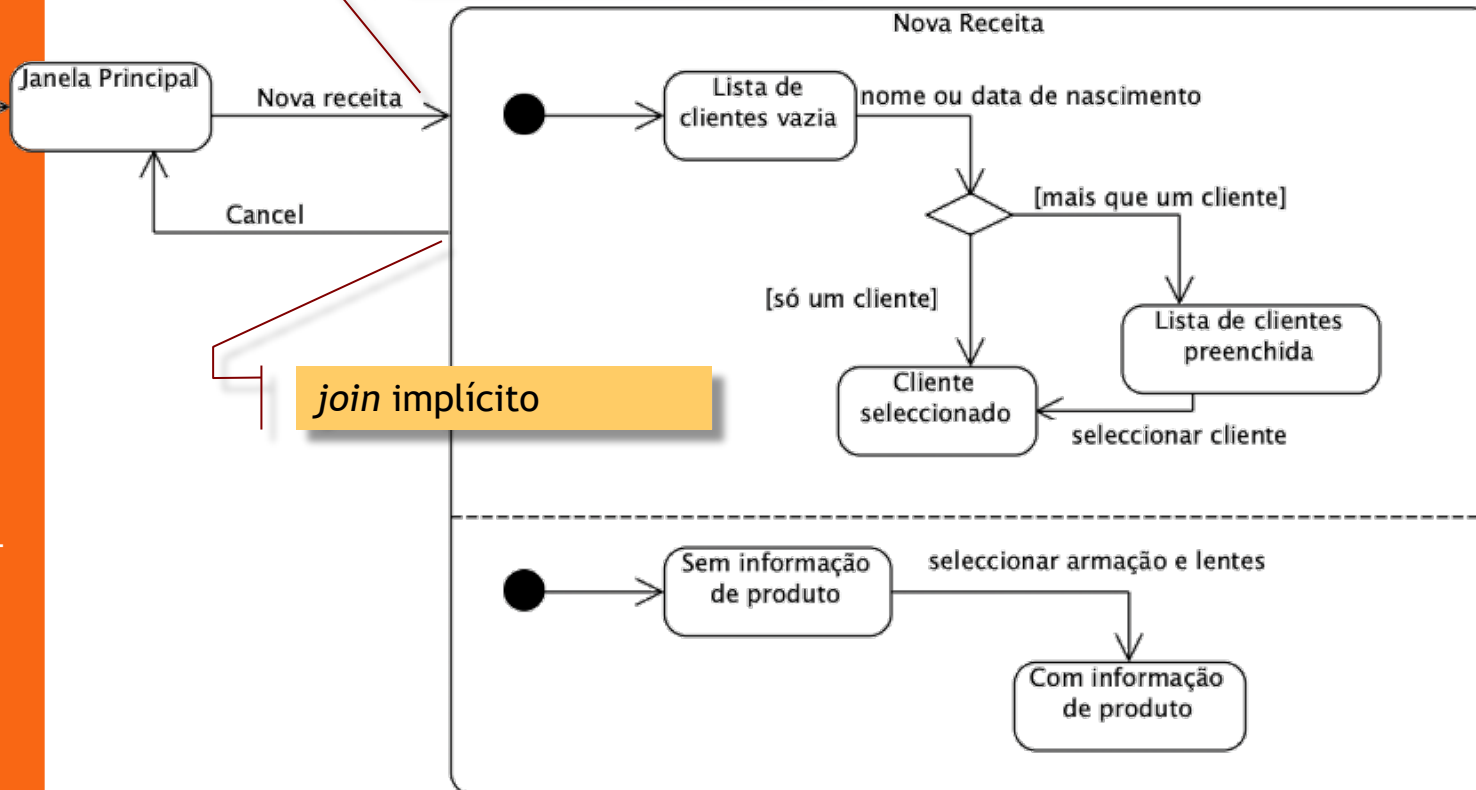
Armações...

Lentes Esq./Dir.



# Estados com concorrência...

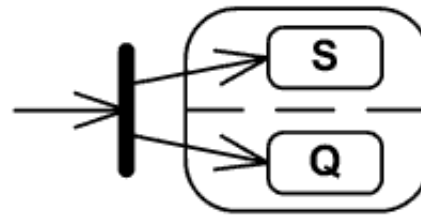
- Um estado pode ser dividido em “regiões” ortogonais
- Cada região contém um sub-diagrama
- Os diagramas das regiões são executados de forma concorrente



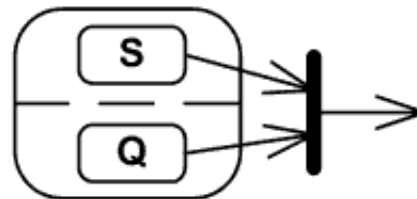


## Pseudoestados *fork* e *join*

- Permitem gerir concorrência.
- Fork - divide uma transição de entrada em duas ou mais transições
  - Transições de saída têm que terminar em regiões ortogonais distintas



- Join - funde duas ou mais transições de entrada numa só transição de saída
  - Transições de entrada têm que originar em regiões ortogonais distinta





# Estados com concorrência...

**Registo de receitas**

Nome Cliente:  Data Nasc.:

Cliente 1  
Cliente 2

202 x 68

Armações...

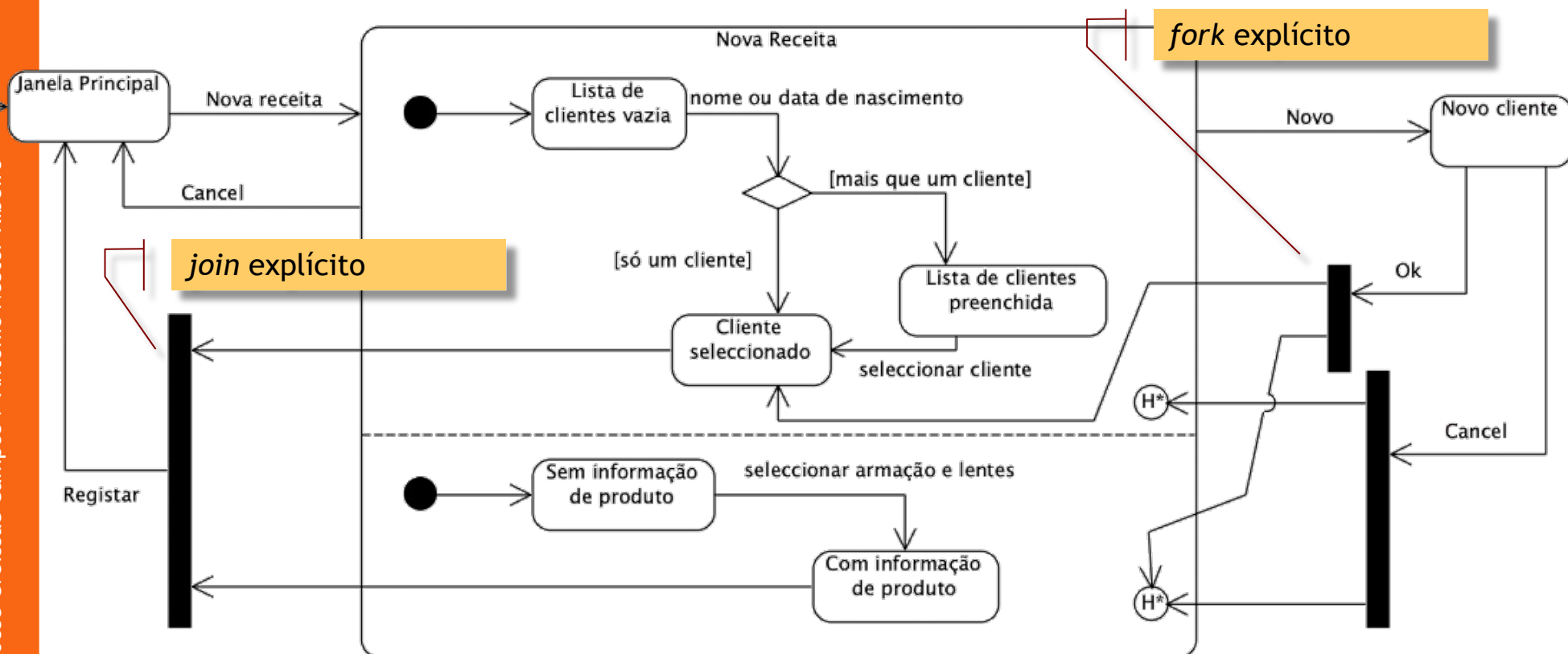
Armação 1  
Armação 2  
Armação 3

Lentes Esq./Dir.

Lente  
Lente

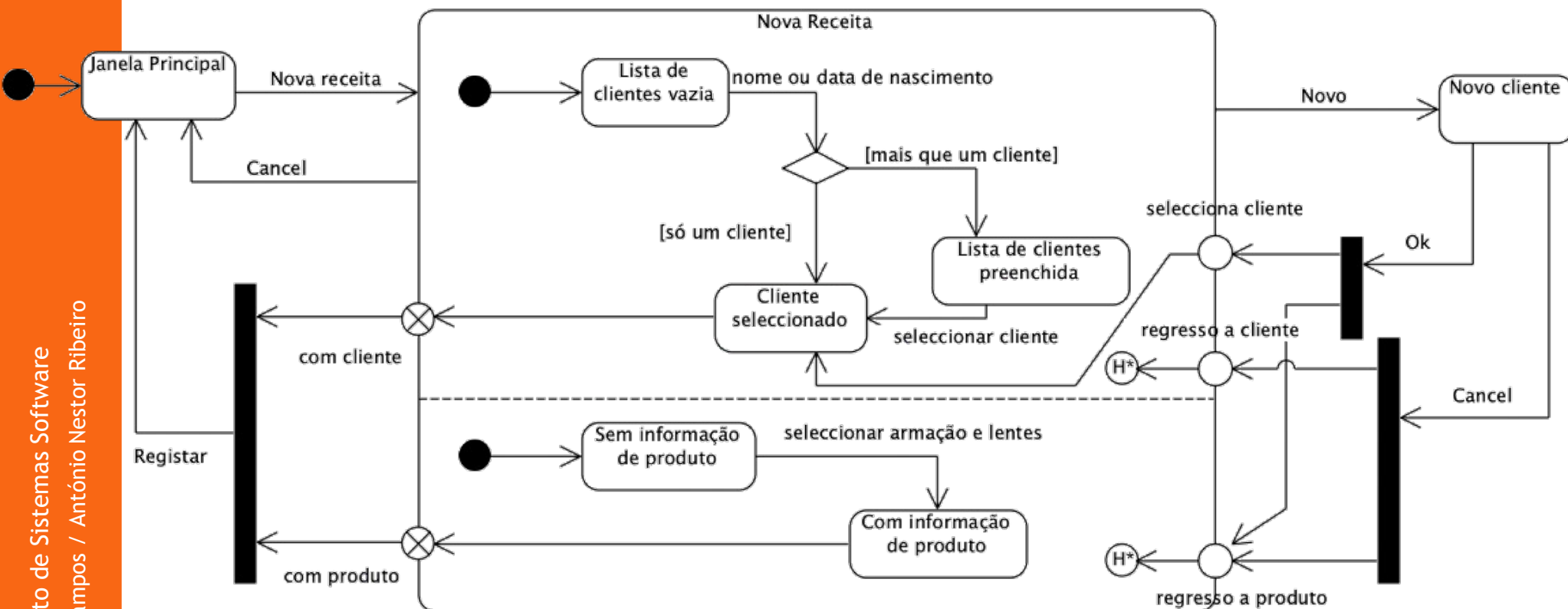
Lente  
Lente

175 x 124



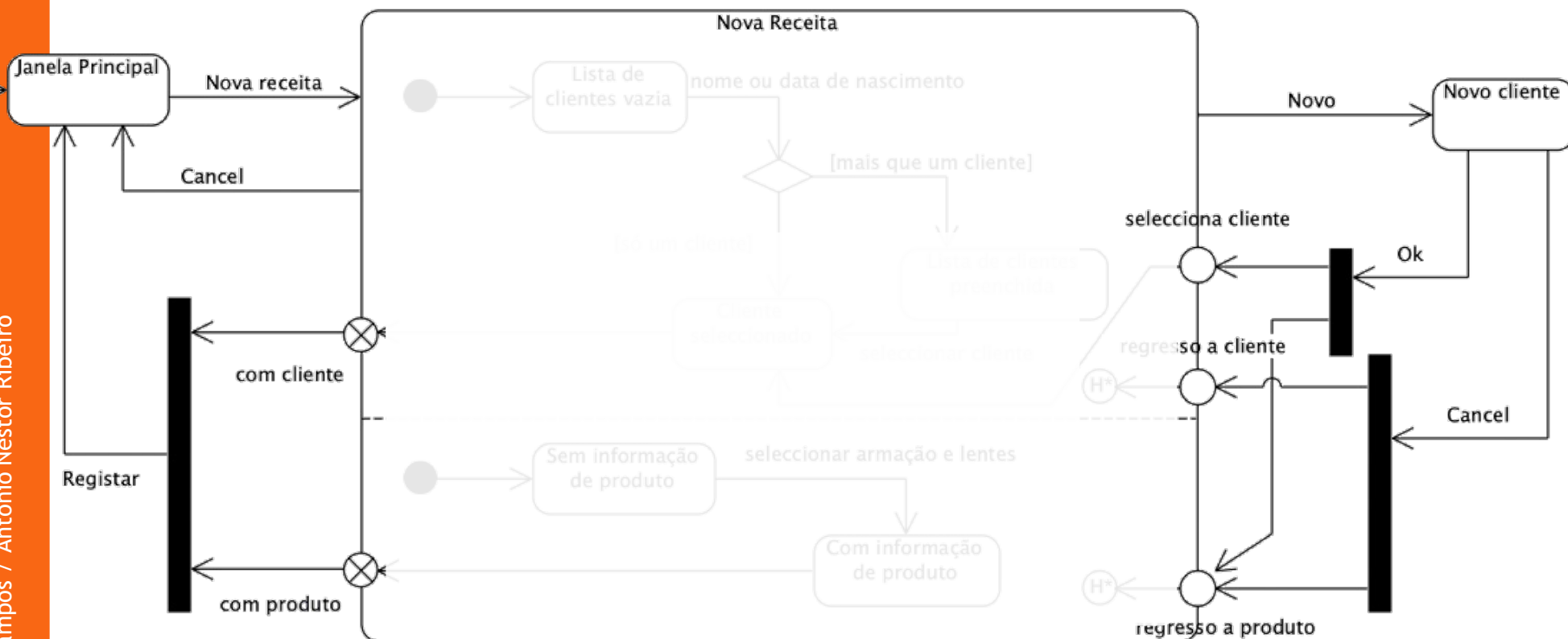


# Pseudo-estados Ponto de entrada e Ponto de saída







# Pseudo-estados Ponto de entrada e Ponto de saída





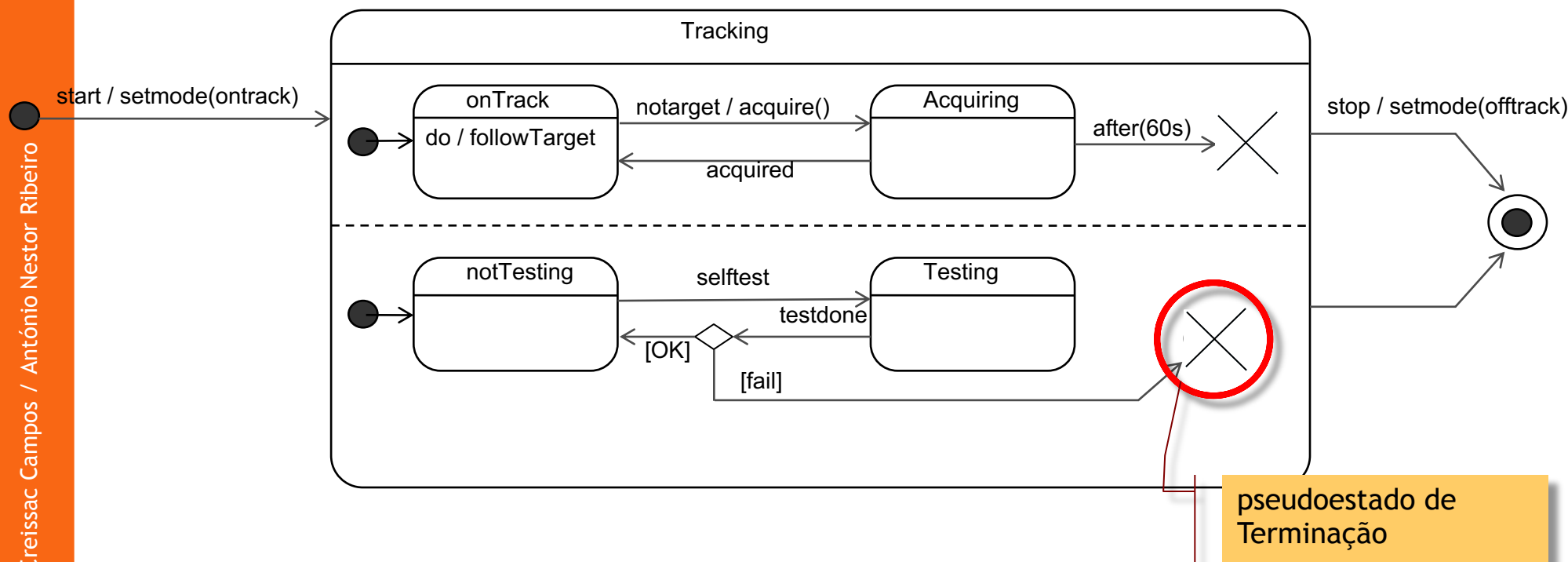
# Pseudoestados Ponto de entrada e Ponto de saída

- Ponto de entrada 
  - Permite definir um ponto de entrada numa máquina de estados ou num estado composto
  - O ponto de entrada é identificado por nome
  - O ponto de entrada transita para um estado interno que poderá ser diferente do definido pelo estado inicial
- Ponto de saída 
  - Permite definir um ponto de saída alternativo ao estado final
  - O ponto de saída é identificado por nome



# Pseudoestado de terminação

- Indica que a execução da máquina de estados termina.
- Não são executadas acções de saída a não ser as da transição para o pseudoestado de terminação





# Diagramas da UML 2.x

