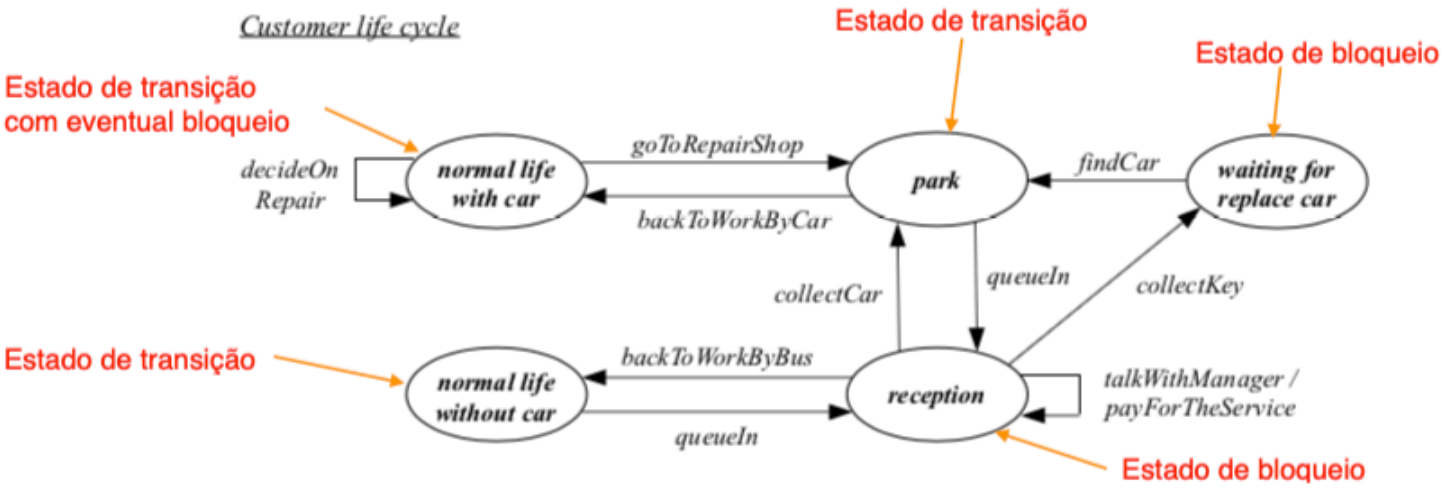
CICLO DE VIDA CUSTOMER



CICLO DE VIDA DE CUSTOMER E SEUS ESTADOS

NORMAL_LIFE_WITH_CAR
É um estado de transição com eventual bloqueio. O customer estará nesse estado em duas possibilidades:

1. *Customer decideOnRepair* e vai à loja de reparos (*goToRepairShop*), entrando no estado PARK;
    - *(a)* Entra no estado RECEPTION e entra na fila (*queueIn*) para falar com o *Manager (talkWithManager);*
    - (b) Indica a peça de reparo e decide se quer um carro substituto ou não;
        - i. Quer carro substituto:
            - (1) Existe Carro substituto:
                - (i) Recebe a chave do carro substituto, e sai do estado RECEPTION;
                - (ii) Vai ao estado PARK (pela ação *collectCar*);
                - (III) Retorna ao estado original (NORMAL_LIFE_WITH_CAR) pela ação *backToWorkByCar*;
            - (2) Não Existe Carro substituto:
                - (i) Entra no estado WAITING_FOR_REPLACE_CAR pela ação *collectKey* e permanece aguardando na fila até ter um carro disponível;
                - (ii) Recebe a chave do carro substituto e passa ao estado PARK pela ação *findCar*;
                - (iii) Retorna ao estado original (NORMAL_LIFE_WITH_CAR) pela ação *backToWorkByCar;*
        - ii. Não quer carro substituto:
            - (1) Vai ao estado NORMAL_LIFE_WITHOUT_CAR pela ação *backToWorkByBus*;
2. *Customer* é informado para retirar o carro:
    - (a) Transita ao estado PARK pela ação *goToRepairShop;*
    - (b) Libera o carro reserva que está em sua posse;
    - (c) Entra no estado RECEPTION;
    - (d) Entra na fila (*queueIn*);
    - (e) Paga pelo serviço (*payForService*);
    - (f) Transita ao estado PARK pela ação *collectCar*;
    - (g) Retorna ao estado **NORMAL_LIFE_WITH_CAR** pela ação *backToWorkByCar.*

NORMAL_LIFE_WITHOUT_CAR
Aplica-se apenas aos *Customers* que não pegaram carro substituto.

1. *Customer* é informado que seu carro está pronto e está **sem** carro substituto;
   (a) Entra na fila (*queueIn*);
   (b) Entra no estado RECEPTION para pagar o serviço;
   (c) Retorna ao estado **PARK** pela ação *collectCar*;
   (d) Retornar ao estado NORMAL_LIFE_WITH_CAR (*backToWorkByCar*);

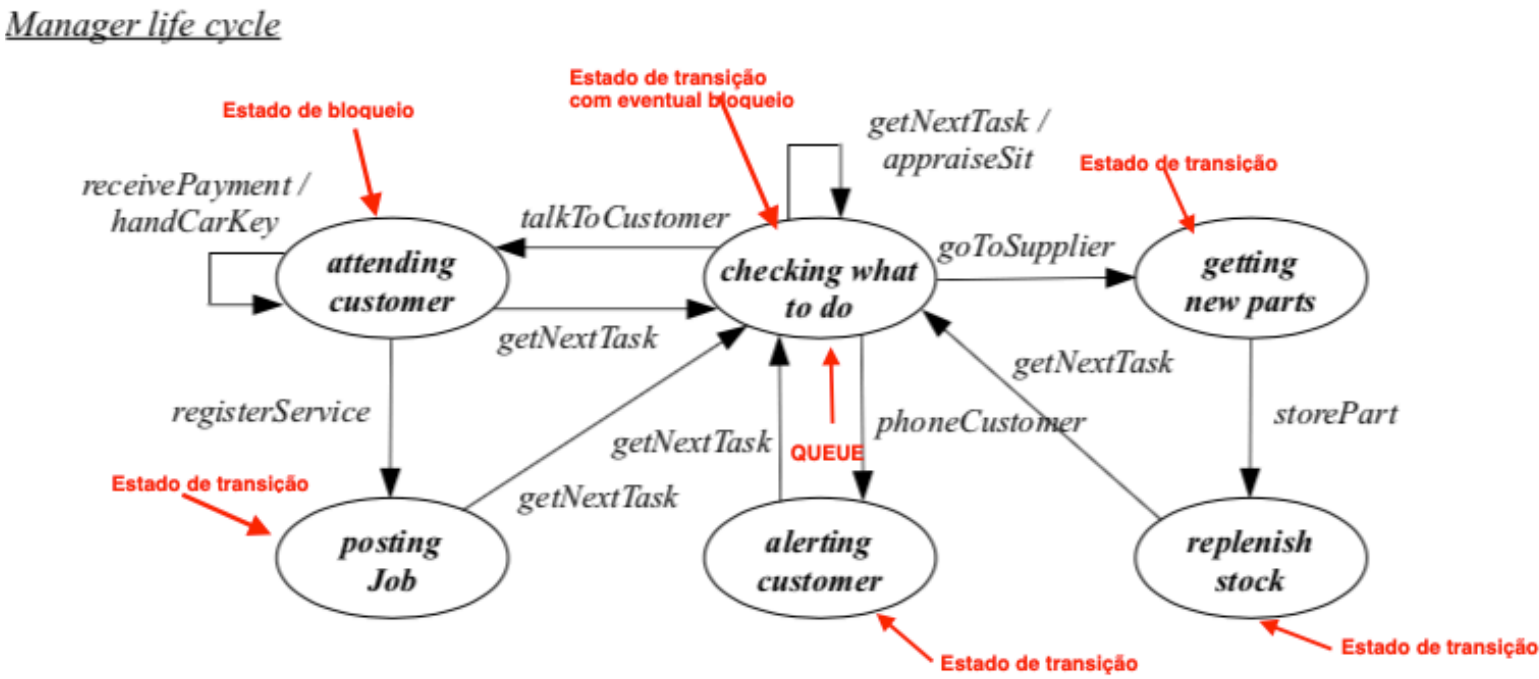
PARK
É um estado independente de transição acessado sempre que:

1. *Customer* tiver NORMAL_LIFE_WITH_CAR como estado de origem (disparado pela ação *goToRepairShop*);
2. *Customer* tiver RECEPTION como estado de origem (disparado pela ação *collectCar*);
3. *Customer* tiver WAITING_FOR_REPLACE_CAR como estado de origem (disparado pela ação *findCar*);

WAITING_FOR_REPLACE_CAR
É um estado de bloqueio para *Customer*. Se aplica apenas aos *Customers* que desejarem carros substitutos (*replaceCars*=true) e *Customers* que tiverem como estado de origem RECEPTION (disparado pela ação *collectKey*)

RECEPTION
É um estado de bloqueio para *Customer*. Se aplica apenas aos *Customers* que estiverem na *queueIn* para deixar o carro para reparo (*talkToManager*) ou para retirar o carro do reparo (*payForService*) aguardando o *Manager*.

CICLO DE VIDA MANAGER



CICLO DE VIDA DO MANAGER E SEUS ESTADOS

ATTENDING_CUSTOMER
É um estado de bloqueio.

1. *Manager* entra nesse estado se e enquanto existirem *Customers* na *queueIn*, disparado pela ação *talkToCustomer;*
2. Atende um *Customer* por vez;
    (a) *Customer* trouxe carro para reparo:
        i. Obtém a chave do carro (*collectKey*);
        ii. Obtém qual peça será reparada (*collectPart?*);
        iii. Informa aos mecânicos a ordem de serviço passando ao estado POSTING_JOB pela ação *registerService*;
        iv. Pergunta ao *Customer* se deseja carro substituto;
            (1) Quer carro substituto:
                (i) Verifica se há carro disponível:
                    a) Existe carro disponível:
                        1) Associa carro ao *Customer*;
                        2) Libera *Customer*;
                    b) Não existe carro disponível:
                        1) Encaminha *Customer* para estado WAITING_FOR_REPLACE_CAR;
                        2) Atende próximo da fila - Retorna ao passo 1:
                            A. Acabou a fila de *Customer*:
                                I. getNextTask -> CHECKING-WHAT_TO_DO;
            (2) Não quer carro substituto
                (i) Libera *Customer*;
                (ii) Atende próximo da fila - Retorna ao passo 1;
                    a) Acabou a fila de *Customer*:
                        1) *getNextTask -> CHECKING_WHAT_TO_DO*
    (b) *Customer* quer pagar pelo serviço;
        i.
    3. Customer veio buscar carro do reparo:
3. Recebe os pagamentos que existirem na *queueIn*;
4. Recebe todas as chaves dos carros com a informação da peça de reparo (*handCarKey*), que existirem na *queueIn*;

POSTING_JOB
Estado de transição.

1. Acessa esse estado após atender todos os *Customers* na *queueIn* e posta todas as ordens de serviço (*handCarKey* com informação de qual a peça deve ser reparada) obtida em ATTENDING_CUSTOMER;
2. Acessa o estado **CHECKING_WHAT_TO_DO** pela ação *getNextTask;*

CHECKING_WHAT_TO_DO
Estado de transição com eventual bloqueio.

1. Entra nesse estado sempre pela ação *getNewTask*;
2. Se a *queueIn* de *Customers* estiver vazia, avalia a situação da repairShop pela ação *appraiseSit*;
3.

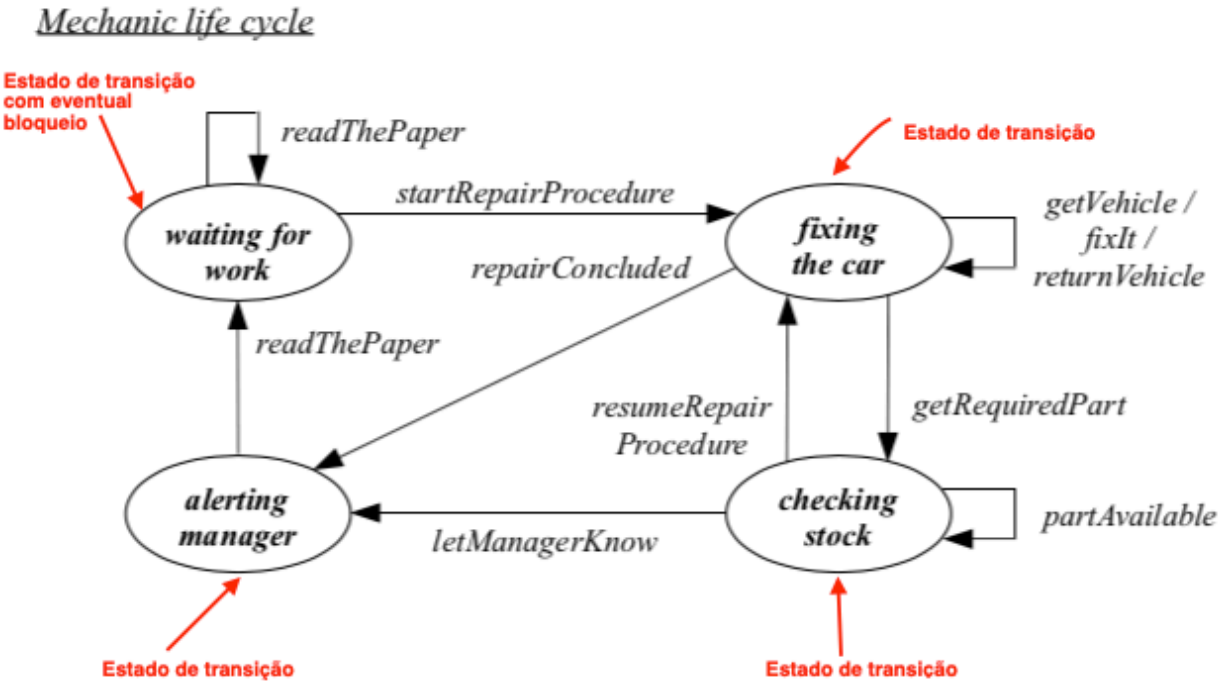
, sai desse estado apenas nas seguintes situações:

1. Se existir *Customers* na *queueIn*, ele volta ao estado ATTENDING_CUSTOMER pela ação *talkToCustomer*;
2. *appraiseSit* sinaliza encaminha ao estado **GETTING_NEW_PARTS** para verificar o estoque, chamando a ação goToSupplier;
3. Se faltarem peças no estoque, passa ao estado **REPLENISH_STOCK**, (reabastece o estoque) pela ação *storePart*, retornar ao estado **CHECKING_WHAT_TO_DO** pela ação *getNewTask*;
4. Se for avisado que um carro está pronto, vai ao estado **ALERTING_CUSTOMER**;

1. O estado **ALERTING_CUSTOMER** possuiu uma *queue* para gerir alertas ao cliente;

B. Permanece nesse estado pela ação *appraiseSit* para verificar e acompanhar as situações na loja, enquanto não houver *Customers* na *queueIn*, ou enquanto não for acionado por outra tarefa (*getNewTask*).

CICLO DE VIDA MECHANIC



CICLO DE VIDA DO MECHANIC COM SEUS ESTADOS

WAITING_FOR_WORK
Estado de transição com eventual bloqueio. Toma conhecimento da ordem de serviço pela ação *readThePaper* e passa ao estado **FIXING_THE_CAR**

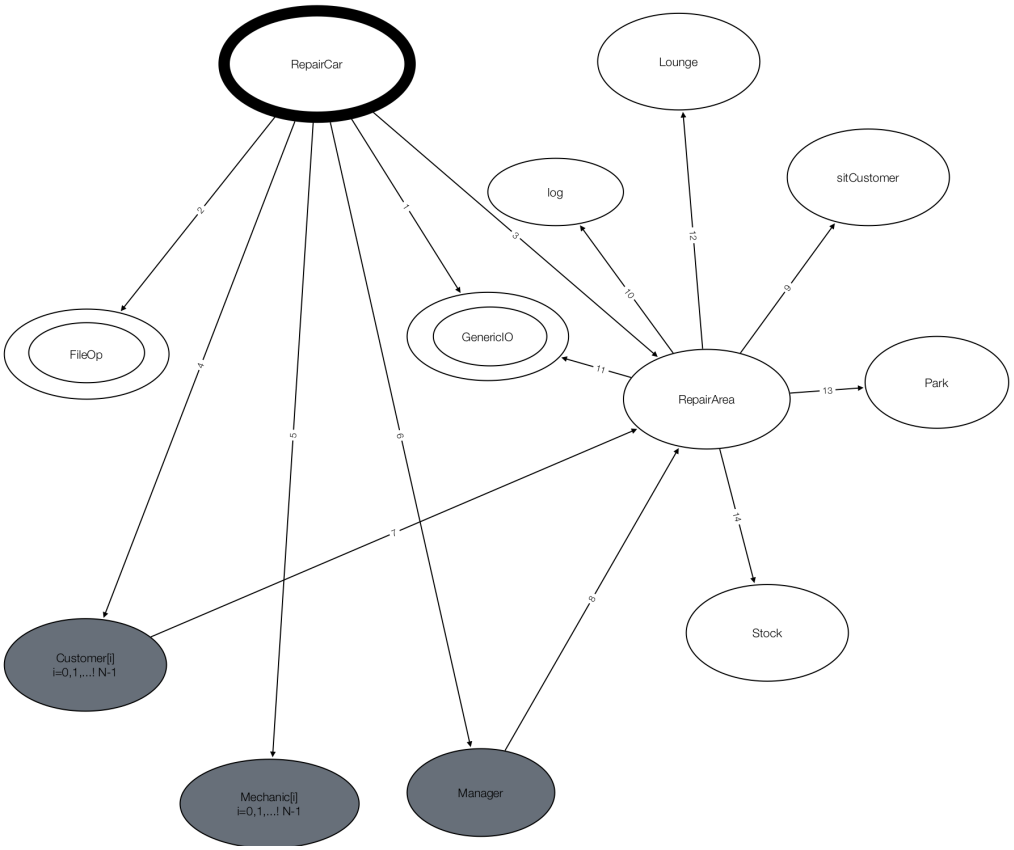
III. Permanece no estado **FIXING_THE_CAR** até terminar as ações a seguir:
   (a) getVehicle - Obtém o veículo de **PARK;**
   (b) *fixIt* - Trabalha no reparo do veículo;
   (c) *returnVehicle* - Devolve o veículo para o **PARK**:
      1) Após *resumeRepairProcedure* (carro reparado), retorna o veículo ao PARK e passa ao estado ALERTING_MANAGER, para informar que um carro já foi reparado, pela ação *repairConcluded*;

IV. CHECKING_STOCK - Passa a esse estado pela ação *getRequiredPart*;
   (a) Obtém nesse estado a informação se a peça está disponível pela ação *partAvailable* e passa ao estado ALERTING_MANAGER em uma das seguintes situações:
      1) ALERTING_MANAGER - Se e peça não estiver disponível, (*letManagertKnow*)**;**
      2) ALERTING_MANAGER - Se e peça estiver disponível, (*resumeRepairProcedure*), volta para FIXING_THE_CAR e vai para ALERTING_MANAGER pela ação repairConcluded**;**

V. ALERTING_MANAGER - Chega a esse estado quando conclui um reparo (*repairConcluded*) ou quando uma peça está em falta (*letManagerKnow*);

VI. **WAITING_FOR_WORK** - Toma conhecimento da ordem de serviço pela ação *readThePaper*, retomando o ciclo. Caso não tenha nova ordem de serviço, retorna ao estado WAITING_FOR_WORK, até existir uma ordem de serviço (POSTING_JOB);

DIAGRAMA DE INTERAÇÃO



1. readln Int, readlnChar, readlnString, writeString,writelnString
2. exists
3. instantiate
4. instantiate, start, join
5. instantiate, start, interrupt, isAlive, join
6. instantiate, start, interrupt, isAlive, join
7. goFixIt
8. returnVehicle, letManagerKnow, readThePaper
9. checkWhatToDo, callCustomer, receivePayment, talkToCustomer, getRequiredPart
10. instantiate, full, write, read
11. instantiate, openForWriting, close, writelnString
12. writelnString
13. instantiate
14. instantiate
15. instantiate
16. instantiate