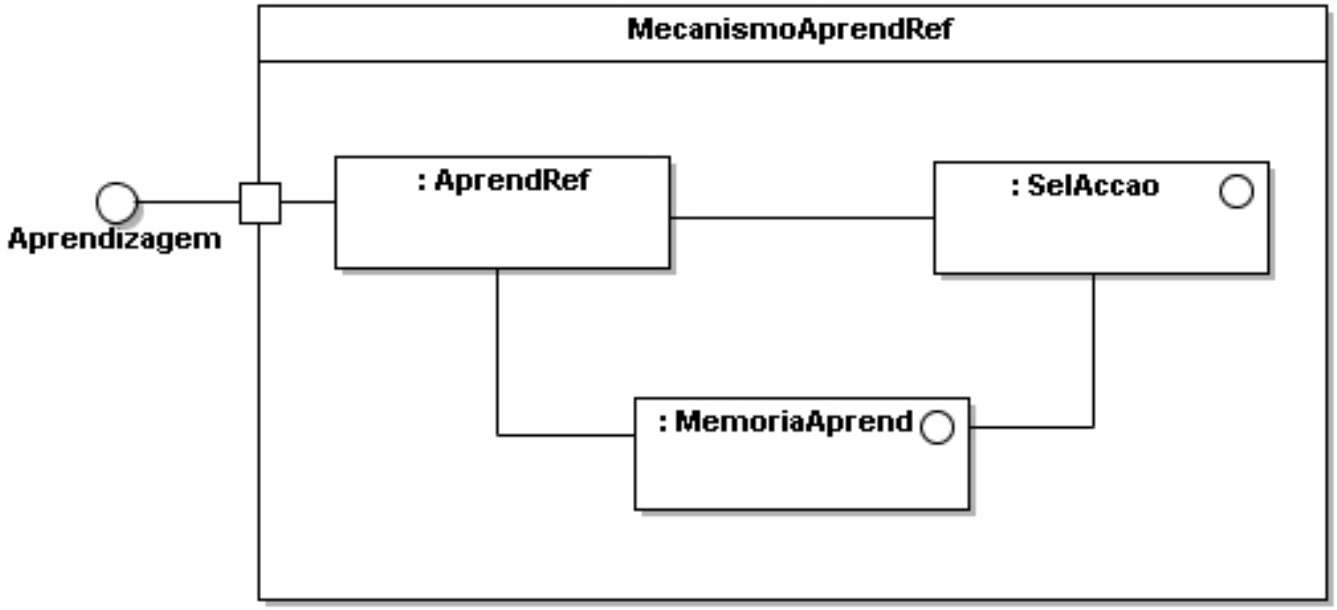
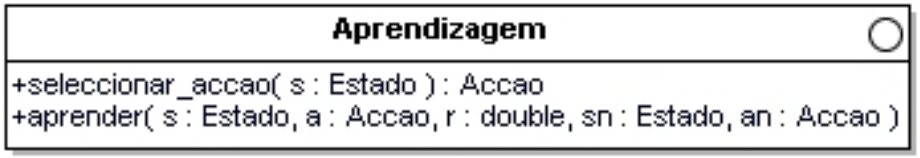
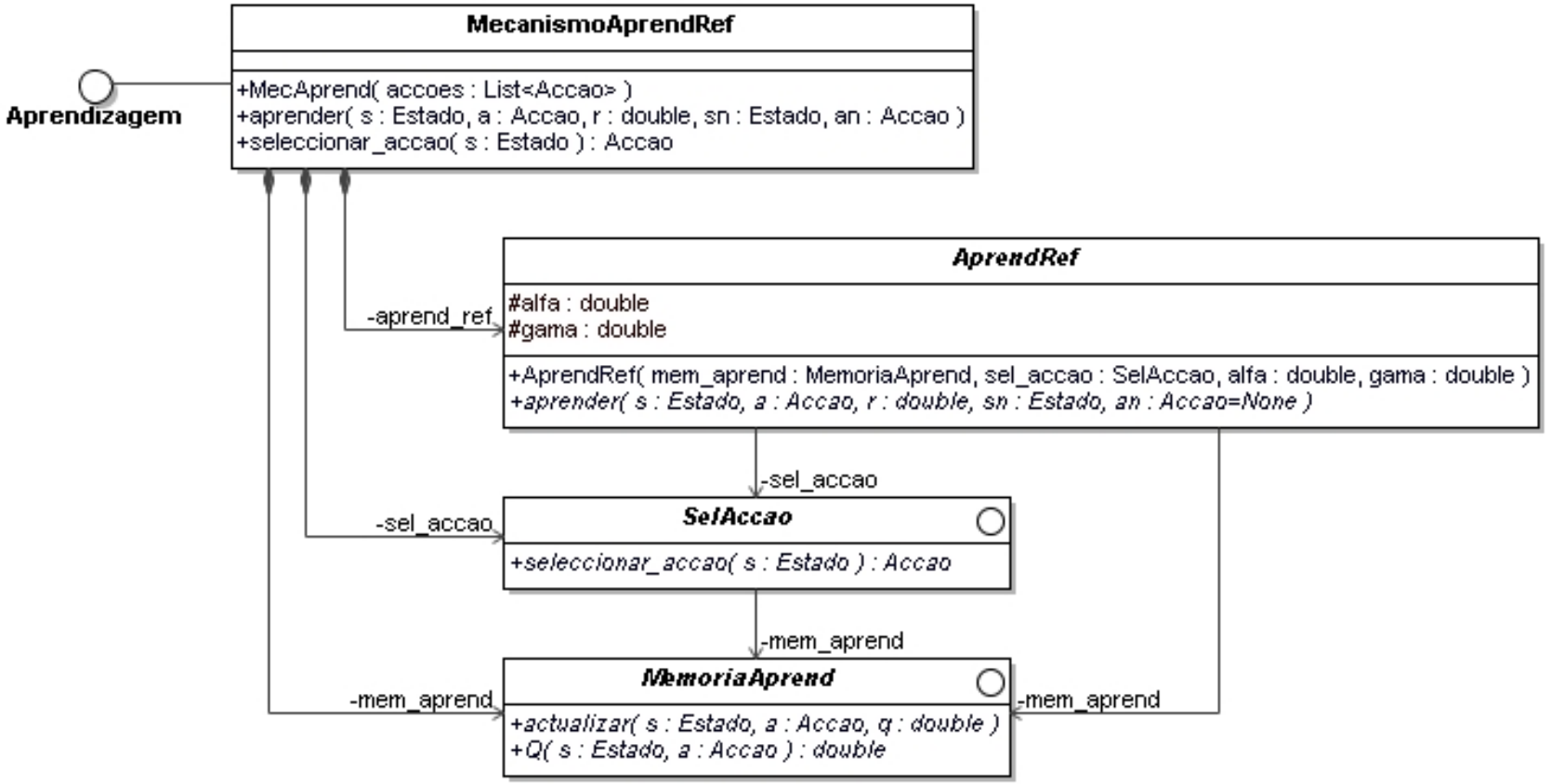
MECANISMO DE APRENDIZAGEM POR REFORÇO

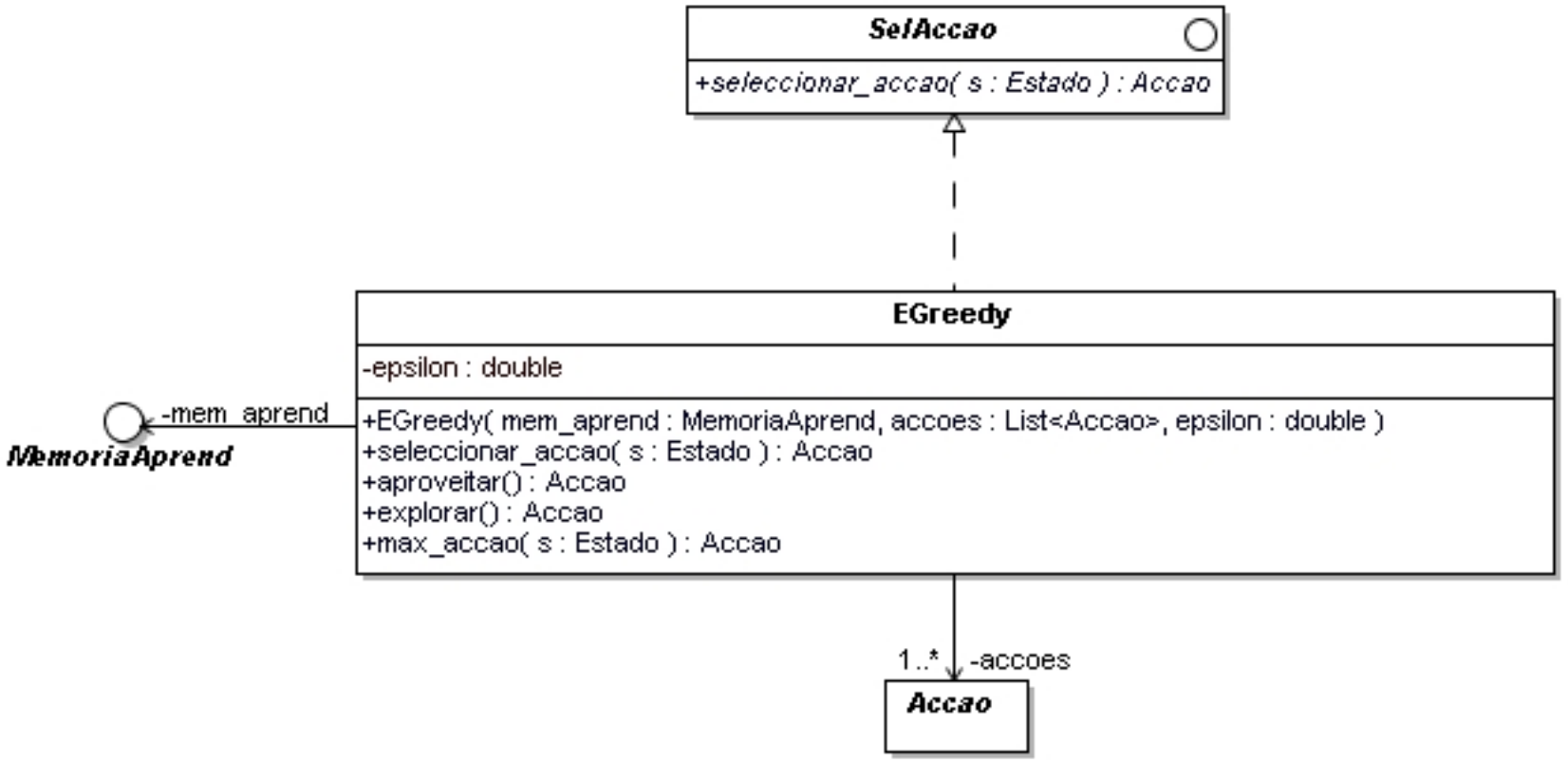




MECANISMO DE APRENDIZAGEM POR REFORÇO



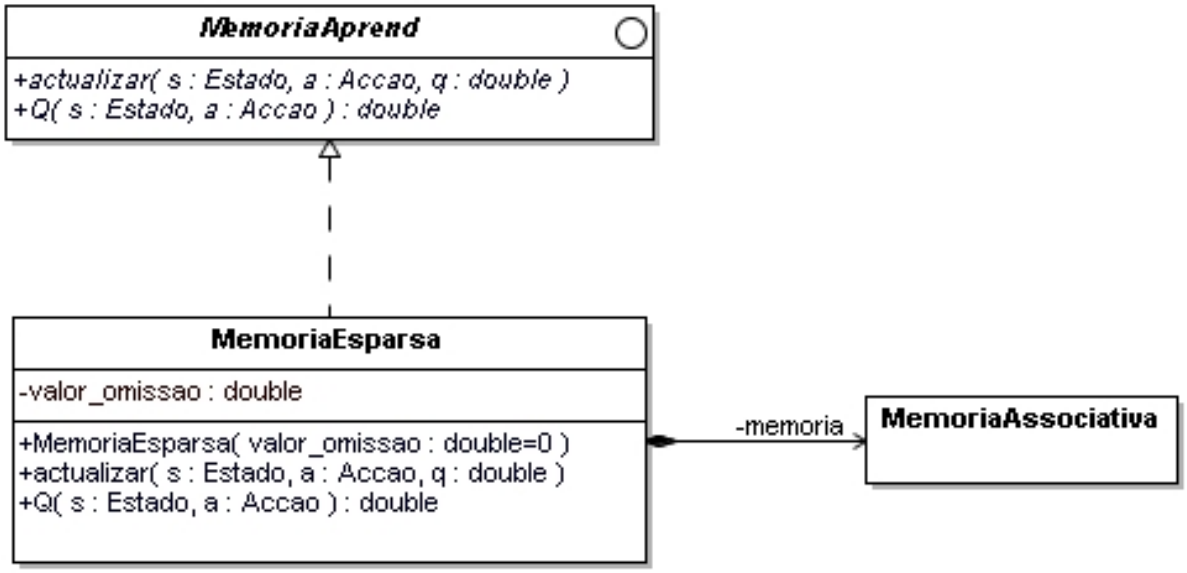
SELECÇÃO DE ACÇÃO ε-*greedy*



SELECÇÃO DE ACÇÃO ε-*greedy*

```
class EGreedy extends SelAccao:
    init(mem_aprend, accoes, epsilon):
        this.mem_aprend = mem_aprend
        this.accoes = accoes
        this.epsilon = epsilon

    max_accao(s):
        shuffle(accoes)
        return argmax(accoes, lambda(a): mem_aprend.Q(s, a))

    aproveitar(s):
        return max_accao(s)

    explorar():
        return choice(accoes)

    seleccionar_accao(s):
        if random() > epsilon:
            accao = aproveitar(s)
        else:
            accao = explorar()
        return accao
```

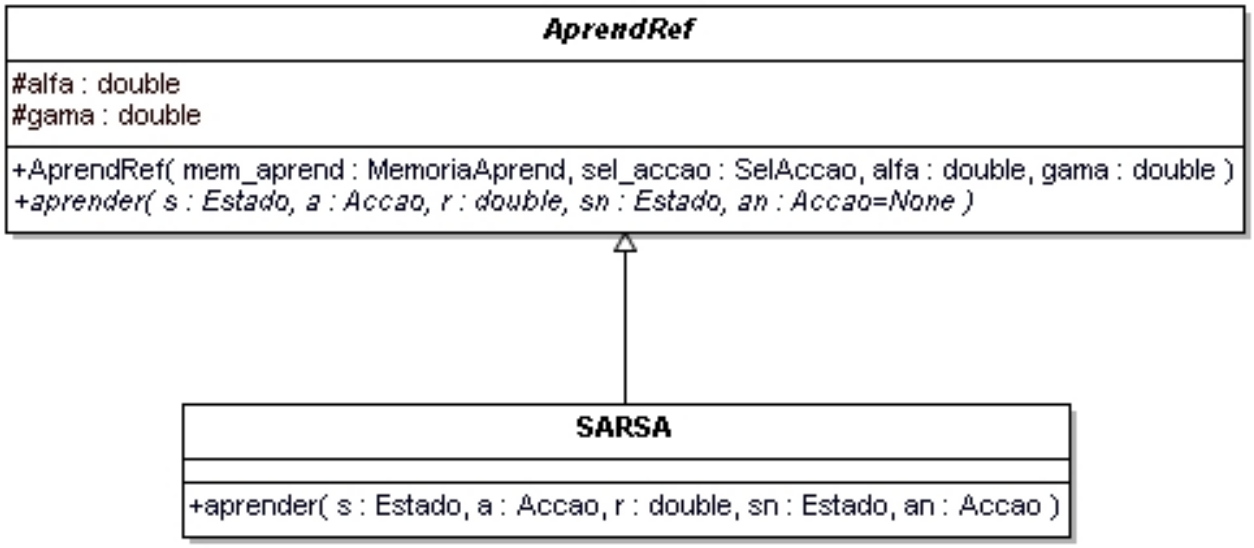
MEMÓRIA DE APRENDIZAGEM ESPARSA



MEMÓRIA DE APRENDIZAGEM ESPARSA

```
class MemoriaEsparsa extends MemoriaAprend:
    init(valor_omissao = 0.0):
        this.valor_omissao = valor_omissao
        this.memoria = {}

    Q(s, a):
        return memoria.get((s, a), valor_omissao)

    actualizar(s, a, q):
        memoria[(s, a)] = q
```

APRENDIZAGEM SARSA

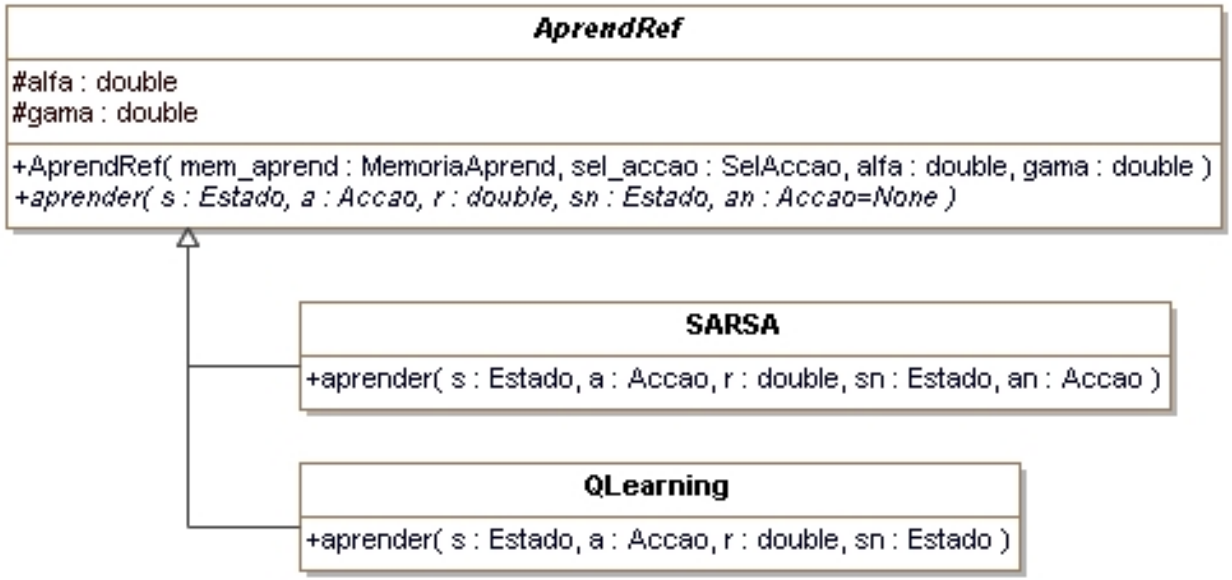


APRENDIZAGEM SARSA

```
class AprendRef:
    init(mem_aprend, sel_accao, alfa, gama):
        this.mem_aprend = mem_aprend
        this.sel_accao = sel_accao
        this.alfa = alfa
        this.gama = gama

    abstract aprender(s, a, r, sn, an = None)
```

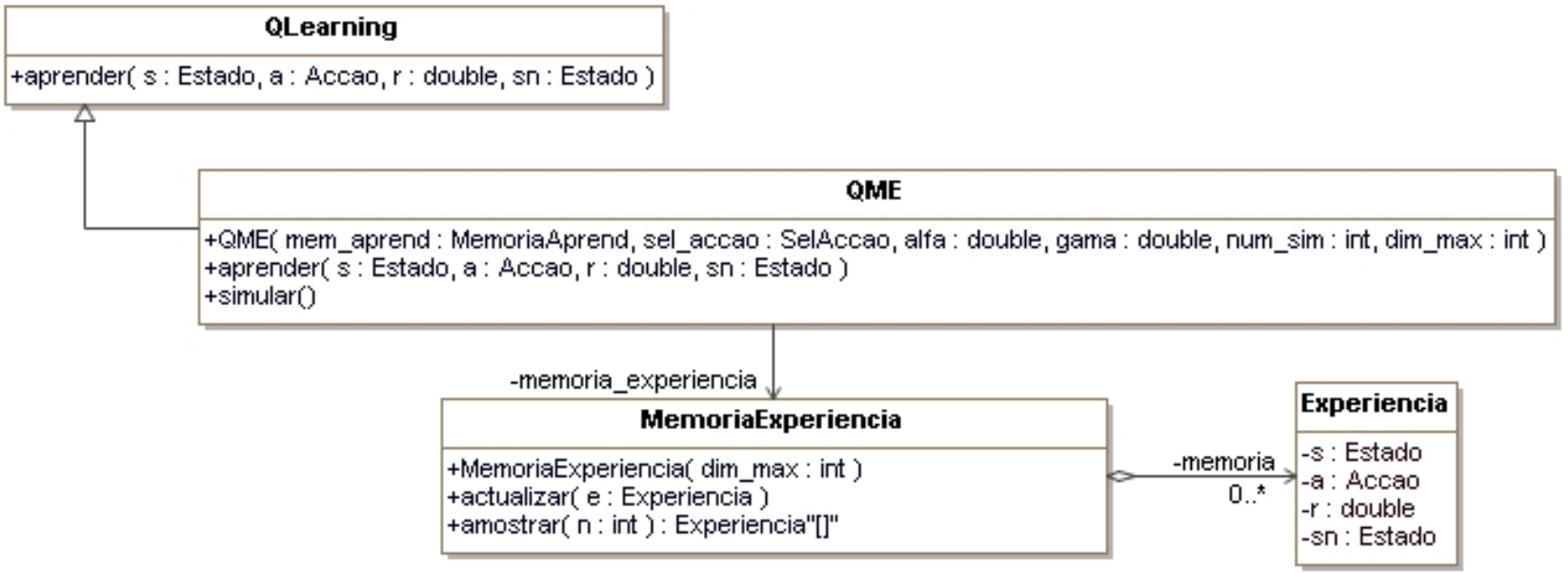
```
class SARSA extends AprendRef:
    aprender(s, a, r, sn, an):
        qsa = mem_aprend.Q(s, a)
        qsnan = mem_aprend.Q(sn, an)
        q = qsa + alfa * (r + gama * qsnan - qsa)
        mem_aprend.actualizar(s, a, q)
```

APRENDIZAGEM *Q-LEARNING*




```
class QLearning extends AprendRef:
    aprender(s, a, r, sn):
        an = sel_accao.max_accao(sn)
        qsa = mem_aprend.Q(s, a)
        qsnan = mem_aprend.Q(sn, an)
        q = qsa + alfa * (r + gama * qsnan - qsa)
        mem_aprend.actualizar(s, a, q)
```

APRENDIZAGEM *Q-LEARNING* COM MEMÓRIA DE EXPERIÊNCIA (QME)



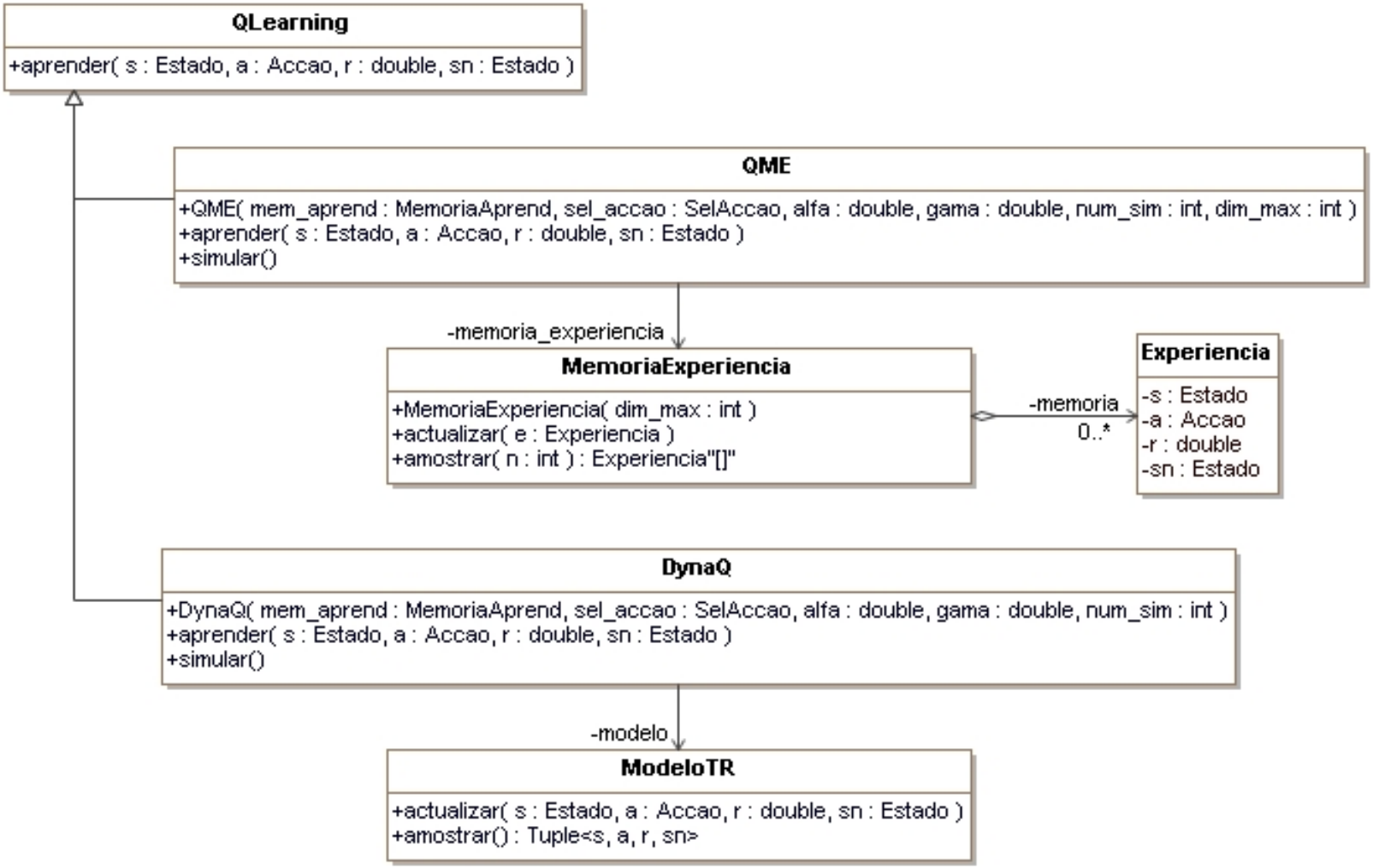
APRENDIZAGEM *Q-LEARNING* COM MEMÓRIA DE EXPERIÊNCIA (QME)

```
class QME extends QLearning:
    init(mem_aprend, sel_accao, alfa, gama, num_sim, dim_max):
        super.init(mem_aprend, sel_accao, alfa, gama)
        this.num_sim = num_sim
        this.memoria_experiencia = MemoriaExperiencia(dim_max)

    aprender(s, a, r, sn):
        super.aprender(s, a, r, sn)
        e = (s, a, r, sn)
        memoria_experiencia.actualizar(e)
        simular()

    simular():
        amostras = memoria_experiencia.amostrar(num_sim)
        for (s, a, r, sn) in amostras:
            super.aprender(s, a, r, sn)
```

APRENDIZAGEM *Q-LEARNING* COM MEMÓRIA DE EXPERIÊNCIA (QME)

```
class MemoriaExperiencia:
    init(dim_max):
        this.dim_max = dim_max
        this.memoria = []

    actualizar(e):
        if memoria.dim() == dim_max:
            memoria.remove(0)
        memoria.append(e)

    amostrar(n):
        n_amostras = min(n, memoria.dim())
        return sample(memoria, n_amostras)
```

APRENDIZAGEM *DYNA-Q*



APRENDIZAGEM *DYNA-Q*

```
class DynaQ extends QLearning:
    init(mem_aprend, sel_accao, alfa, gama, num_sim):
        super.init(mem_aprend, sel_accao, alfa, gama)
        this.num_sim = num_sim
        this.modelo = ModeloTR()

    aprender(s, a, r, sn):
        super.aprender(s, a, r, sn)
        modelo.actualizar(s, a, r, sn)
        simular()

    simular():
        repeat num_sim:
            s, a, r, sn = modelo.amostrar()
            super.aprender(s, a, r, sn)
```

APRENDIZAGEM *DYNA-Q*

MODELO DO MUNDO

```
class ModeloTR:
    init():
        this.T = {}
        this.R = {}

    actualizar(s, a, r, sn):
        T[(s, a)] = sn  # Modelo determinista
        R[(s, a)] = r

    amostrar():
        s, a = choice(T.keys())
        sn = T[(s, a)]
        r = R[(s, a)]
        return s, a, r, sn
```