



Agregação versus Composição

Richarlyson A. D'Emery

site: <https://sites.google.com/site/profricodemery/mpoo>

grupo: http://groups.google.com/group/mpoo_uast

email grupo: mpoo_uast@googlegroups.com

contato: rico_demery@yahoo.com.br

Sumário



Agregação



Composição

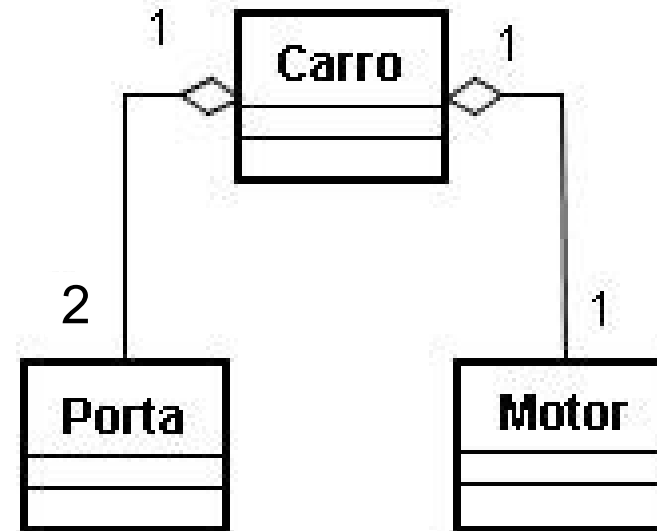
Agregação



- Demonstra que as informações de uma classe precisam ser complementadas por uma outra classe.
- A parte pode existir sem o todo
- A agregação é definida pelo relacionamento “*tem um*” .
 - Herança
 - Já discutimos que os relacionamentos “*é um*” são implementados por herança

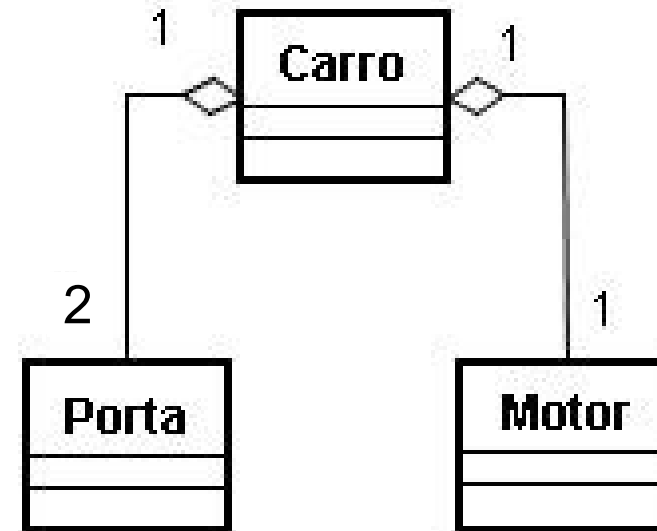
Agregação - Representação

- O losango colocado nas linhas de associação da classe Carro indica que a classe Carro tem um relacionamento de agregação com as classes Porta e Motor.
- Como citado, a agregação implica em um relacionamento **todo/parte**.



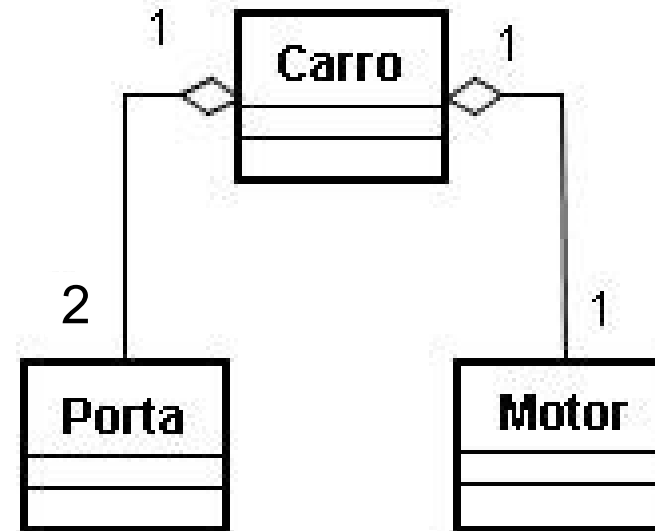
Agregação - Representação

- A classe que tem o símbolo de agregação (o losango vazio) em sua extremidade de uma linha de associação é o **todo** (nesse caso, Carro) e a classe na outra extremidade da linha da associação é a **parte** (neste caso, as classes Porta e Motor)



Agregação - Exemplo

- O carro “*tem um*” motor e duas portas.
- Se analisarmos a frase “*a parte pode existir sem o todo*”
- Podemos concluir que as classes Porta e Motor podem existir sem a necessidade de Carro existir



Agregação - Implementação



- Para ilustrar o exemplo de agregação, vejamos uma aplicação Java que fará uso dos arquivos
 - **Motor.java**
 - **Porta.java**
 - **CarroAgregacao.java.**

Agregação - Implementação



- **Motor.java**

```
//Arquivo Motor.java
public class Motor {
    double potencia;
    public Motor(double pot) {
        this.potencia=pot;
    }
}
```


Agregação - Implementação



- **Porta.java**

```
//Arquivo Porta.java
public class Porta {
    String cor;
    public Porta(String c) {
        this.cor=c;
    }
}
```

Agregação - Implementação



```
/*Exemplo de Agregação em Java - Arquivo: CarroAgregacao.java */
public class CarroAgregacao {

    private String modeloCarro;
    private static Porta pErq = new Porta("azul");
    private static Porta pDir = new Porta("azul");
    private static Motor motor = new Motor(1.6);

    public CarroAgregacao(String m){ this.modeloCarro=m; }

    public static void main(String [] args){
        CarroAgregacao carro = new CarroAgregacao("Passeio");
        System.out.println("Descrição do carro:\n" + "Modelo: " +
        carro.modeloCarro + "Cor da porta esquerda" + carro.pEsq.cor +
        "Potência do motor" + carro.motor.potencia);
    }
}
```

Agregação - Implementação

- Observe que se o objeto carro não existir

```
CarroAgregacao carro = new CarroAgregacao("Passeio");
```

- Ainda assim os objetos porta e motor existirão

```
private static Porta pEsq = new Porta("azul");
```

```
private static Porta pDir = new Porta("azul");
```

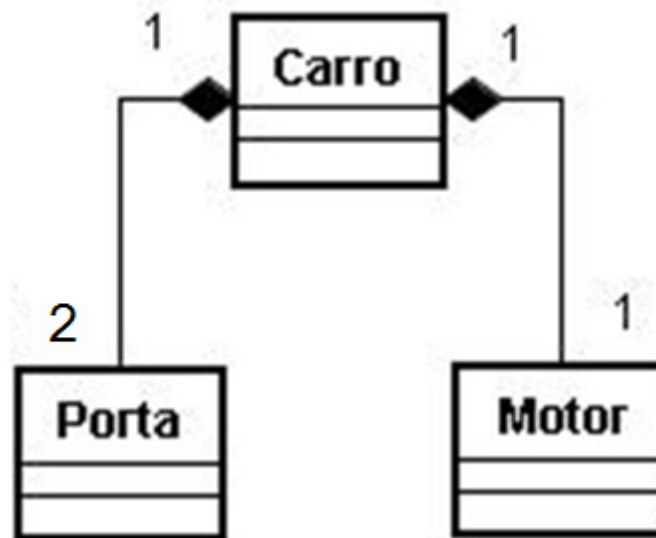
```
private static Motor motor = new Motor(1.6);
```

- Conclusão:
 - instanciados independentemente!

Composição

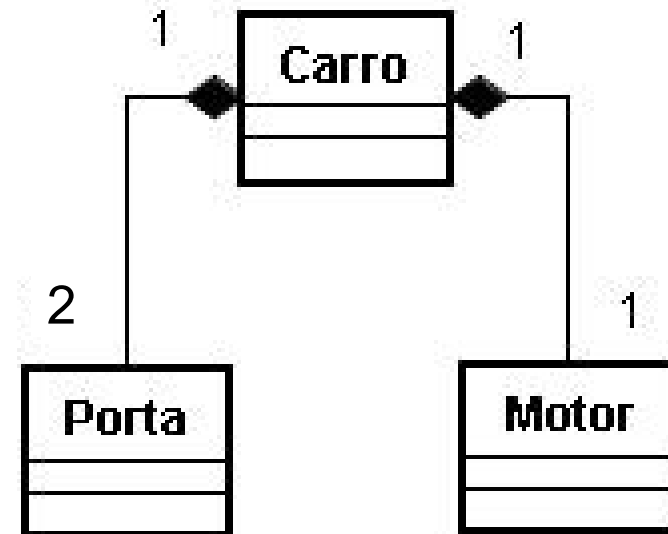


- É um tipo de agregação, da mesma forma que na agregação, a composição também é dada pelo relacionamento “**tem um**”.
- A figura abaixo ilustra um exemplo de composição:



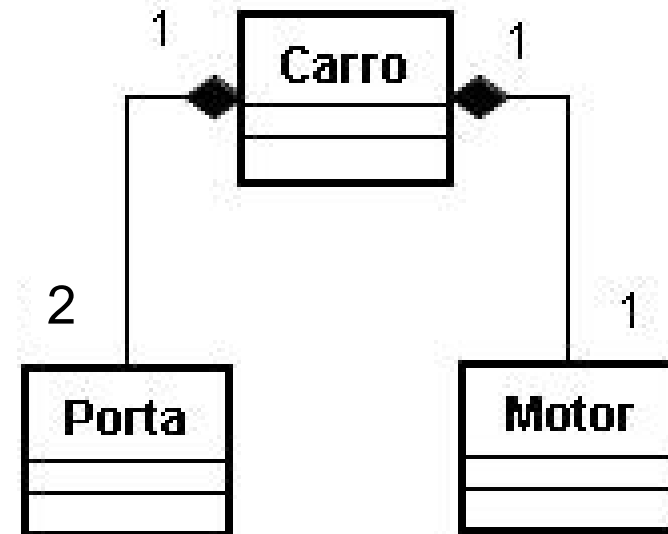
Composição - Representação

- Assim como a agregação a composição é representado por um losango, porém preenchido, colocado nas linhas de associação da classe Carro indica que a classe Carro tem um relacionamento de composição com as classes Porta e Motor.
- Quando o todo desaparece, todas as partes também desaparecem.



Composição - Representação

- Se analisarmos a frase “*Quando o todo desaparece, todas as partes também desaparecem*” podemos concluir que se uma instância (objeto) de Carro não existir, as instâncias de variáveis (atributos) das classes Porta e Motor também não existirão.



Composição - Implementação



- Para ilustrar o exemplo de composição, vejamos uma aplicação Java que fará uso dos arquivos
 - **Motor.java** – Mesmo arquivo da agregação
 - **Porta.java** – Mesmo arquivo da agregação
 - **CarroComposicao.java.**

Composição - Implementação



```
/* Exemplo de Composição em Java - Arquivo: CarroComposicao.java */

public class CarroComposicao {

    private String modeloCarro;
    private static Porta pEsq, pDir;
    private static Motor motor;

    public CarroComposicao (String m, String c, double p){
        this.modeloCarro = m;
        pEsq = new Porta(c);
        pDir = new Porta(c);
        motor = new Motor(p);
    }

    public static void main(String [] args){
        CarroComposicao carro = new CarroComposicao("Passeio", "azul", 1.6);
        System.out.println("Descrição do carro:\n" + "Modelo: " + carro.modeloCarro +
            "Cor da porta esquerda:" + carro.pEsq.cor + "Potência do motor: " +
            carro.motor.potencia);
    }
}
```


Composição - Implementação

- Observe que se o objeto carro não existir

```
CarroComposicao carro = new CarroComposicao("Passeio ",  
                                             "azul", 1.6);
```

- Conclusão:
 - os objetos porta e motor TAMBÉM NÃO existirão,
pois só passam existir quando carro é
instanciado!

Discussão: Agregação versus Composição



No relacionamento por agregação ou por composição podemos destacar que a diferença está basicamente nas regras de criação dos objetos.

Discussão:

Agregação versus Composição



- Na **Agregação** podemos observar que ainda que a instância da classe Carro (objeto carro) não existisse os objetos porta e motor ainda existiriam, pois foram instanciados independente da criação do objeto carro, ou seja:

“A parte pode existir sem o todo.”

Discussão:

Agregação versus Composição



- Logo se substituirmos as linhas de comando:

```
System.out.println("Número de portas" + carro.porta.numPortas);  
System.out.println("Potência do motor" + carro.motor.potencia);
```

- por:

```
System.out.println("Número de portas" + porta.numPortas);  
System.out.println("Potência do motor" + motor.potencia);
```

- os atributos **cor** e **potencia** ainda serão exibidos no console. Isso porque, como explicado, os objetos porta e motor continuam existindo independente do objeto carro existir.

Discussão:

Agregação versus Composição



- Na **Composição** podemos observar que a classe `CarroComposicao` contém as variáveis de instância (atributos) `modeloCarro`, `porta` e `motor`. O atributo `porta` é uma referência para a classe `Porta`, assim como `motor` é uma referência para `Motor`, que por sua vez contém as variáveis de instância `cor` e `potencia`, respectivamente.
- Isso mostra que uma classe pode conter referências para objetos de outras classes.
- O método construtor `CarroComposicao` recebe três parâmetros – `m`, `c` e `p`, no qual o parâmetro `c` é passado para o método construtor `Porta`, com o objetivo de inicializar o objeto `porta`, da mesma forma que o parâmetro `p` é passado para `Motor` com o objetivo de inicializar o objeto `motor`.

Discussão:

Agregação versus Composição



- E dessa forma não será possível acessar os atributos de Porta e Motor sem que o objeto carro tenha sido instanciado (diferentemente da agregação), ou seja:

“Quando o todo desaparece, todas as partes também desaparecem.”

Com o auxílio do IDE Eclipse podemos perceber que ao compilar a aplicação **CarroComposicao.java** não é possível acessar os objetos porta e motor:

Discussão:

Agregação versus Composição



```
public class CarroComposicao {  
    private String modeloCarro;  
    private static Porta porta;  
    private static Motor motor;  
  
    public CarroComposicao(String m, int q, double p){  
        this.modeloCarro = m;  
        porta = new Porta(q);  
        motor = new Motor(p);  
    }  
  
    public static void main(String [] args){  
        //CarroComposicao carro = new CarroComposicao("Passeio", 3, 1.6);  
        System.out.println("Descrição do carro:\n");  
        //System.out.println("Modelo: " + carro.modeloCarro);  
        System.out.println("Número de portas: " + porta.numPortas);  
        System.out.println("Potência do motor: " + motor.potencia);  
    }  
}
```

1 → //CarroComposicao carro = new CarroComposicao("Passeio", 3, 1.6);

2 → System.out.println("Número de portas: " + porta.numPortas);

3 → System.out.println("Potência do motor: " + motor.potencia);

4 → Exception in thread "main" java.lang.NullPointerException
at CarroComposicao.main(CarroComposicao.java:20)

Discussão:

Agregação versus Composição



- Vejamos:
 - 1 – o objeto carro não foi instanciado;
 - 2 – o objeto porta não pode ser acessado;
 - 3 – o objeto motor não pode ser acessado;
 - 4 – Exceção gerada pelo fato dos objetos porta e motor não terem sido instanciados.

Analizando...



- Observe ainda que na composição:
 - Porta e Motor ainda podem ser instanciados, sem utilizar obrigatoriamente Carro.

```
public class Aplicacao {
    Motor motor = new Motor(1.0);
    Porta porta = new Porta("azul");

    public static void main(String[] args) {
        Aplicacao app = new Aplicacao();
        System.out.println(app.motor.potencia);
        System.out.println(app.porta.getCorPorta());
    }
}
```

app utiliza porta e motor sem a existência de carro

Analizando...



- Então:

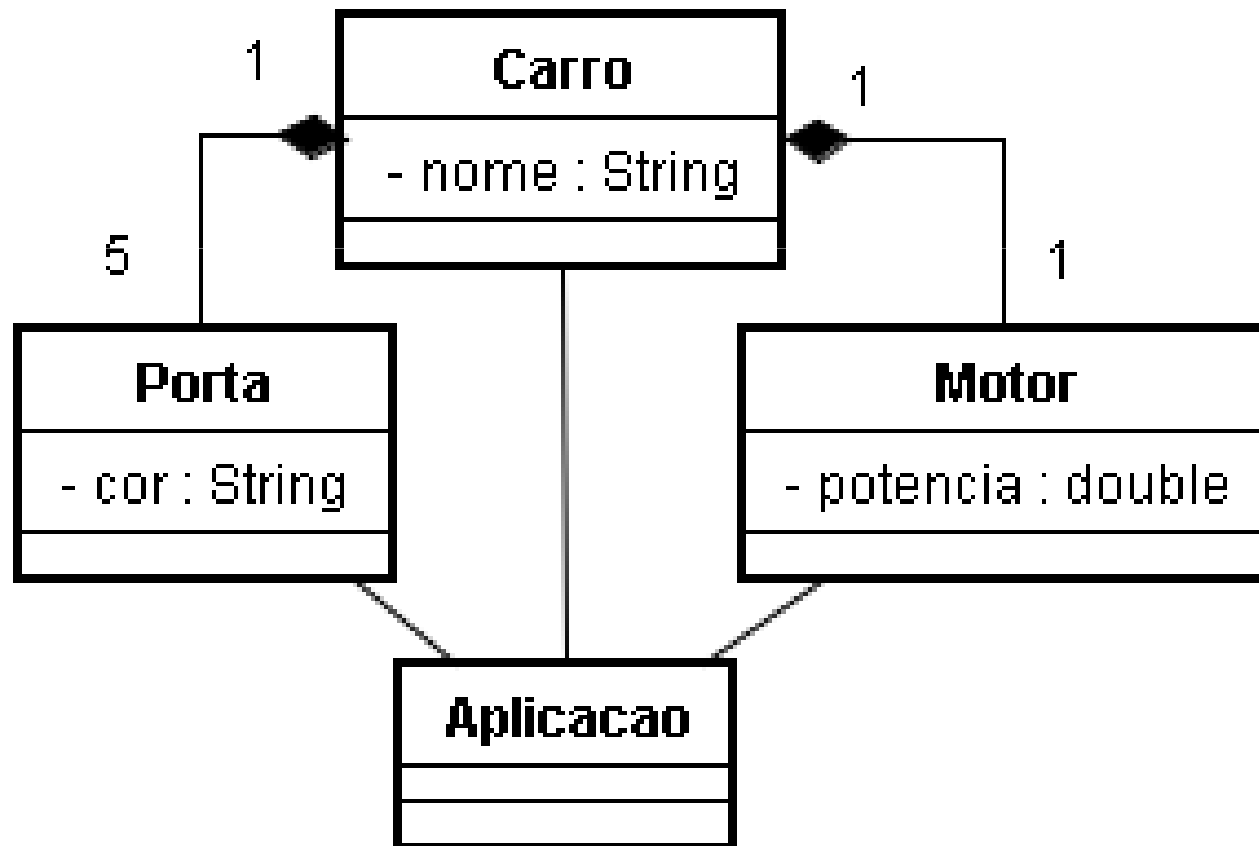
“Como tornar o acesso da parte dependente a existência do todo?”

- Solução
 - 1: Utilizar classe interna
 - 2: Forçar o uso do todo no construtor da parte.

Composição



- Analise a situação:



Composição



- Solução 1: Utilizando classe interna:

```
public class Carro {  
    String nome;  
    Motor motor;  
    Porta porta [] = new Porta[5];
```

```
    public Carro(String nome,  
double potenciaMotor, String corPortas){  
        this.nome=nome;  
        this.motor = new Motor(potenciaMotor);  
        for (int i=0; i<porta.length;i++)  
            this.porta[i]= new Porta(corPortas);  
    }
```

```
    private class Motor {  
        double potencia;  
        public Motor(double pot){  
            this.potencia=pot;  
        }  
    }
```

```
    private class Porta {  
        String cor;  
        public Porta(String c){  
            this.cor=c;  
        }  
    }
```

← garante a cardinalidade

Composição



- Analisando a Solução 1:

```
Carro.java  Aplicacao.java X
public class Aplicacao {
    public static void main(String[] args) {
        Carro carro = new Carro("sentra", 2.0, "cinza");
        System.out.println(carro.nome + "\n" +
            carro.motor.potencia + "\n" +
            carro.porta[0].cor);
    }
}
```

Uma vez que Motor e Porta são *private*, então Aplicação não poderá utilizar diretamente tais classes, a não ser por Carro

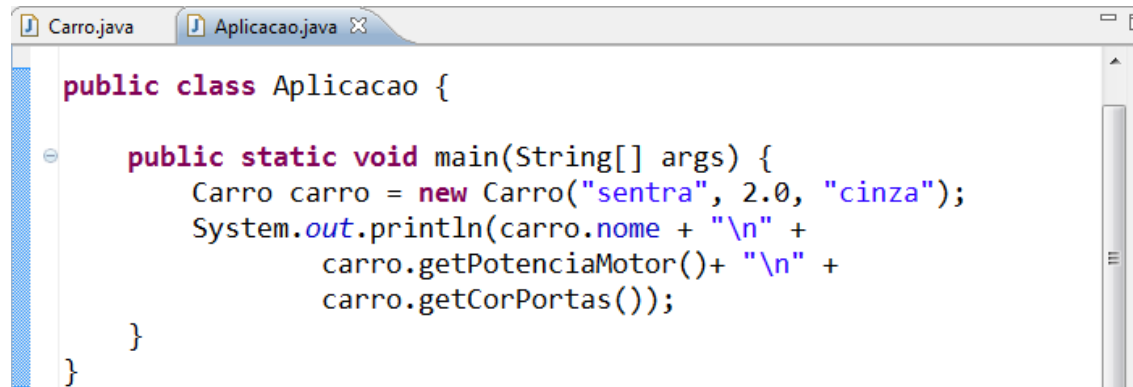
- Solução: *Métodos de acesso ao conteúdo de Porta e Motor*

Composição

- Solução para acesso a Porta e Motor:

```
public class Carro {  
    ...  
    public double getPotenciaMotor() {  
        return this.motor.potencia;  
    }  
    public ArrayList<String> getCorPortas() {  
        ArrayList<String> cor = new ArrayList<String>();  
        for (int i=0; i<this.porta.length;i++)  
            cor.add(this.porta[i].cor);  
        return cor;  
    }  
}
```

- Logo:



```
public class Aplicacao {  
    public static void main(String[] args) {  
        Carro carro = new Carro("sentra", 2.0, "cinza");  
        System.out.println(carro.nome + "\n" +  
            carro.getPotenciaMotor() + "\n" +  
            carro.getCorPortas());  
    }  
}
```

Composição



- Solução 2: Forçar o uso do todo no construtor da parte.

```
//Carro.java
public class Carro {
    String nome;
    Motor motor;
    Porta porta [] = new Porta[5];

    public Carro(String nome){
        this.nome=nome;
        this.motor = new Motor(this);
        for (int i = 0; i < porta.length; i++)
            this.porta[i] = new Porta(this);
    }
}
```

```
//Motor.java
public class Motor {
    double potencia;
    public Motor(Carro carro){}
    public Motor(double pot, Carro carro){
        carro.motor.potencia=pot;
    }
}
```

```
//Porta.java
public class Porta {
    String cor;

    public Porta(Carro carro){}
    public Porta(String cor, int
indicePorta, Carro carro){
        carro.porta[indicePorta].cor=cor;
    }
}
```

Composição



- Cont. Solução 2

```
//Aplicacao.java
public class Aplicacao {

    public static void main(String[] args) {
        Carro carro = new Carro("sentra");
        Motor motor = new Motor(2.0, carro);
        Porta porta [] = new Porta[5];
        String corPorta [] = {"cinza", "cinza", "cinza", "cinza", "cinza"};

        for (int i=0; i<porta.length;i++)
            porta[i]= new Porta(corPorta[i], i, carro);

        //Para visualizar o resultado
        ArrayList<String> saidaCorPorta = new ArrayList<String>();
        for (int i=0; i<carro.porta.length;i++)
            saidaCorPorta.add(carro.porta[i].cor);

        System.out.println(carro.nome + "\n" +
            carro.motor.potencia + "\n" +
            saidaCorPorta);
    }
}
```


Composição



- Analisando a Solução 2:
 - Esta é a solução mais utilizável na Prática,
 - Permite que sistemas tenha visão as classe (respeitando eventuais relacionamentos, como, por exemplo, em Aplicacao)

Exercício

- Implementa em Java um programa que representa as informações de um Corpo Humano
 - Corpo, possui
 - Cabeça, Braço e Perna
 - Cabeça, possui:
 - Boca, Olhos, Cabelo, Orelhas
 - Boca, possui
 - Dentes e Língua
 - Braço
 - Mão
 - Perna
 - Pé
 - Mão e Pé
 - Dedos



FIM

Prof. Richarlyson D'Emery

site: <https://sites.google.com/site/profricodemery/mpoo>

grupo: http://groups.google.com/group/mpoo_uast

email grupo: mpoo_uast@googlegroups.com

contato: rico_demery@yahoo.com.br