Sprint 1

Aluno: Gustavo Zacarias Souza

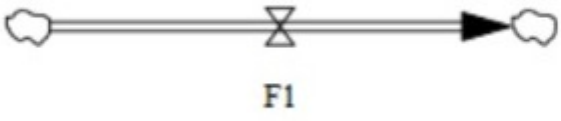
Identifique os “casos de uso” aos quais sua API deverá satisfazer. Então, identifique os “critérios de aceitação” do cliente.

Os casos de uso identificados foram os seguintes:

1. **Apenas um sistema**
   Neste caso de caso de uso, definimos a criação de apenas um sistema no modelo.

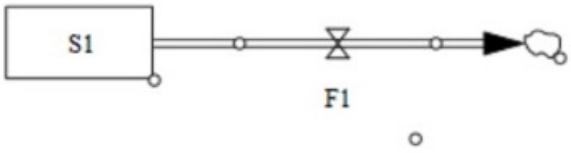

```
System system1("sistema 1", 1);
Model modelo1("uso 1", 1);
modelo1.add(1, system1);
```

2. **Apenas um fluxo sem origem nem destino**
   Neste caso de uso, definimos apenas um fluxo sem origem nem destino




```
MyFlow flow1("fluxo 1", 1);
Model modelo1("uso 2", 1);
modelo1.add(flow1);
```

3. **Um fluxo com um sistema de origem, mas sem destino**
   Neste caso, temos um sistema e um fluxo conectado a este como origem, porém com um destino indefinido.

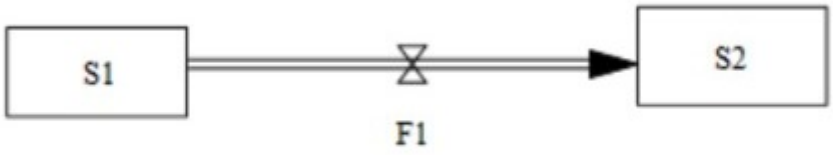



```
MyFlow flow1("fluxo 1", 1);
System system1("sistema 1", 1);
Model modelo1("uso 3", 1);
modelo1.add(system1);
modelo1.add(flow1);
flow1.set_source(system1);
```

4. **Um fluxo sem origem, mas com um sistema de destino**
   Neste caso, temos um sistema e um fluxo conectado a este como destino, porém com uma origem indefinida.


```
MyFlow flow1("fluxo 1", 1);
System system1("sistema 1", 1);
Model modelo1("uso 4", 1);
modelo1.add(system1);
modelo1.add(flow1);
flow1.set_target(system1);
```

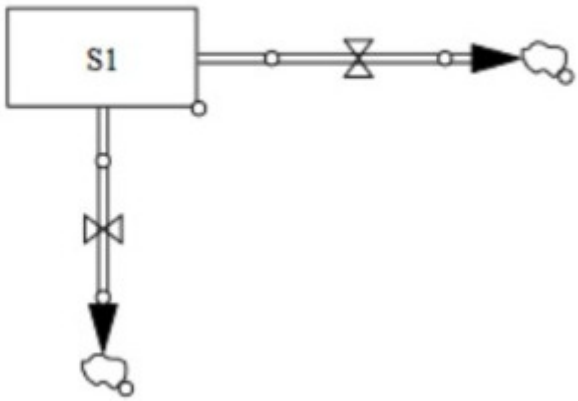
5. **Dois sistemas conectados por um fluxo**
   Neste caso, temos dois sistemas conectados por um fluxo




```
MyFlow flow1("fluxo 1", 1);
System system1("sistema 1", 1);
Model modelo1("uso 5", 1);
System system2("sistema 2", 2);
modelo1.add(system1);
modelo1.add(flow1);
modelo1.add(system2);
flow1.set_source(system1);
flow1.set_target(syestem2);
```

6. **Dois fluxos com o mesmo sistema de origem**

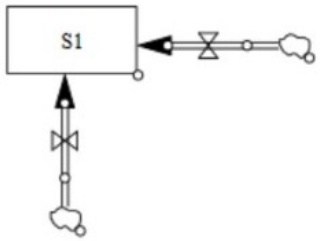
   Neste caso, temos um sistema do qual sai dois fluxos distintos




```
Model modelo1("uso 6", 1);
System system1("sistema 1", 1);
MyFlow flow1("fluxo 1", 1);
MyFlow flow2("fluxo 2", 2);
modelo1.add(system1);
modelo1.add(flow1);
```

```
modelo1.add(flow2);
flow1.set_source(system1);
flow2.set_source(system1);
```

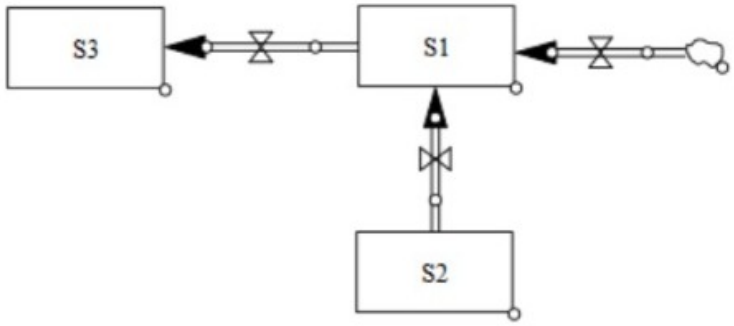
7. **Dois fluxos com o mesmo sistema de destino**
   Neste caso, temos um sistema no qual chegam dois fluxos distintos




```
Model modelo1("uso 7", 1);
System system1("sistema 1", 1);
MyFlow fluxo1("fluxo 1", 1);
MyFlow fluxo2("fluxo 2", 2);
modelo1.add(system1);
modelo1.add(fluxo1);
modelo1.add(fluxo2);
flow1.set_target(system1);
flow2.set_target(system1);
```

8. **Um modelo com vários sistemas conectados e um fluxo de origem.**
   Neste caso, temos um modelo que possui um fluxo sem origem (F1), com um sistema S1 de destino. S1 é origem do fluxo que conecta ao S3 e S1 também é destino do

fluxo que possui S2 como origem.

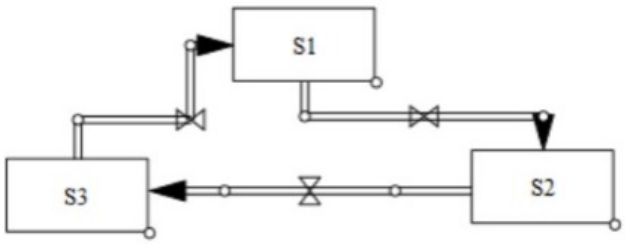



```
Model modelo1("uso 8", 1);
System system1("sistema 1", 1), system2("sistema2", 2), system3("sistema 3", 3);
MyFlow flow1("fluxo 1", 1), flow2("fluxo 2", 2), flow3("fluxo 3", 3);

modelo1.add(system1);
modelo1.add(system2);
modelo1.add(system3);
modelo1.add(flow1);
modelo1.add(flow2);
modelo1.add(flow3);

flow1.set_target(system1);

flow2.set_source(system2);
flow2.set_target(system1);

flow3.set_source(system1);
flow3.set_target(system3);
```

9. **Um modelo cíclico**

   Neste caso, temos 3 sistemas, todos conectados entre si, com S1 conectado a S2,

S2 conectado a S3 e S3 conectado a S1.




```
Model modelo1("uso 9", 1);
System system1("sistema 1", 1), system2("sistema 2", 2), system3("sistema 3", 3);
MyFlow flow1("fluxo 1", 1), flow2("fluxo 2", 2), flow3("fluxo 3", 3);

modelo1.add(system1);
modelo1.add(system2);
modelo1.add(system3);
modelo1.add(flow1);
modelo1.add(flow2);
modelo1.add(flow3);

flow1.set_source(system1);
flow1.set_target(system2);

flow2.set_source(system2);
flow2.set_target(system3);

flow3.set_source(system3);
flow3.set_target(system1);
```

10. **Um modelo com 2 fluxos e 1 sistema**

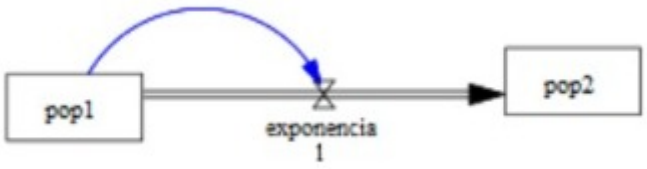
    Neste caso, temos um sistema que é origem de um fluxo sem destino e destino de

um fluxo sem origem.


```
Model modelo1("uso 10", 1); System system1("sistema 1", 1);
MyFlow flow1("fluxo 1", 1), flow2("fluxo 2", 2);


modelo1.add(system1);
modelo1.add(flow1);
modelo1.add(flow2);

flow1.set_target(system1);
flow2.set_source(system1);
```

Critérios de aceitação

1. **Dois sistemas conectados por um fluxo com uma função exponencial utilizando o sistema de origem como entrada para a equação:**



Para este critério de aceitação, devemos definir uma maneira de inserir a equação a ser utilizada pelo fluxo. Algumas ideias surgiram, e, dentre elas as melhores foram:

- Criar uma função evaluate, assim, haveria uma função `set_equation(String equation)`, porém, deste modo, seria difícil verificar os erros de uma equação mal escrita, e a implementação de uma função de evaluate não é simples de ser feita;

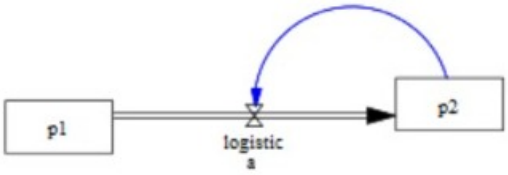
- Utilizar da sobrecarga para substituir o método `execute` da classe `Flow`, porém, desta forma, a classe não seria tão genérica e o código poderá quebrar, por não permitir que hajam múltiplas equações;

Fazer a classe `Flow` ser abstrata, e permitir a herança da mesma. Assim, haveriam várias classes filhas possuindo suas próprias equações. O problema desta ideia é a criação de muitas classes.

No momento, a melhor ideia foi a de abstrair a classe `Flow`. Seguindo com essa ideia, o caso de uso do critério de aceitação ficou da seguinte forma:

```
   class FlowExponential : public Flow
   { public:
           FlowExponential() : Flow() {}

        double execute() {return this.get_source().get_value() * 0.01; }
};

   int main() {
           Model modelo;
           System system1, system2;
           FlowExponential f1;

           modelo.add(system1);
           modelo.add(system2);
           modelo.add(f1);

           f1.set_source(system1);
           f1.set_target(system2);


           model.execute(0, 100, 1);

   }
```

2. Dois sistemas conectados por um fluxo, utilizando o sistema de destino como entrada para a equação

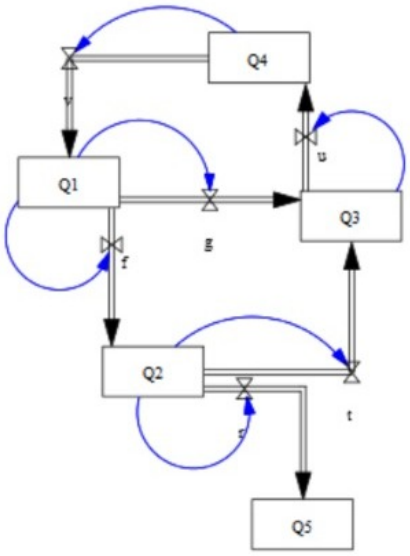


Neste exemplo, temos de criar uma equação para satisfazer a logística do modelo.

Para isso, vamos utilizar do `p2` como entrada para o fluxo. Como já temos a forma de criar o fluxo definida, vamos para o caso de uso:

```
class FlowLogistico : public Flow
{ public:
        FlowLogistico() : Flow()
        {}
        double execute() {
                double value = this.get_source().get_value();
                return 0.01 * value * (1 - value / 70);
        }
};

int main() {
        Model model;
        System system1, system2;
        FlowLogistico f1;

        model.add(system1);
        model.add(system2);
        model.add(f1);

        f1.set_source(system1);
        f1.set_target(system2);

         model.execute(0, 100, 1);

 }
```

3. Vários sistemas conectados por vários fluxos, com várias equações exponenciais



Neste caso de uso temos 5 sistemas, com `Q1` sendo origem de `Q3` e `Q2`, `Q2` origem de `Q5` e `Q3`, `Q3` origem de `Q4` e `Q4` sendo origem apenas de `Q1`, todos estes tendo fluxos com equações exponenciais. Para representar este caso de uso, temos o seguinte pseudo-código:

```
Model model;
System q1, q2, q3, q4, q5;
ExponentialFlow f, g, r, t, u, v;


model.add(q1);
model.add(q2);
model.add(q3);
model.add(q4);
model.add(q5);
```

```
        model.add(f);
        model.add(g);
        model.add(r);
        model.add(t);
        model.add(u);
        model.add(v);

        f.set_source(q1);
        f.set_target(q2);

        g.set_source(q1);
        g.set_target(q3);

        r.set_source(q2);
        r.set_target(q5);

        t.set_source(q2);
        t.set_target(q3);

        u.set_source(q3);
        u.set_target(q4);

        v.set_source(q4);
        v.set_target(q1);

        // Run the model with a time step of 1 until time 100 starting
from 0.
        model.execute(0, 100, 1);
```

Diagrama de classes UML