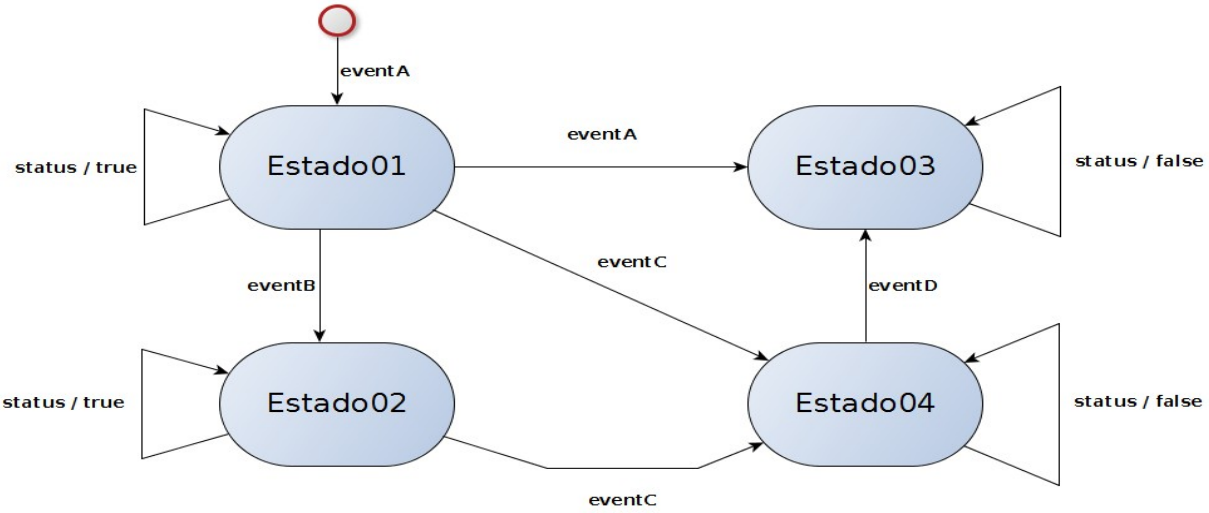
Lista de Exercícios: Teste Baseado em Modelo

Prof. André Takeshi Endo

(Exercício 1) Considere a máquina de estados a seguir.



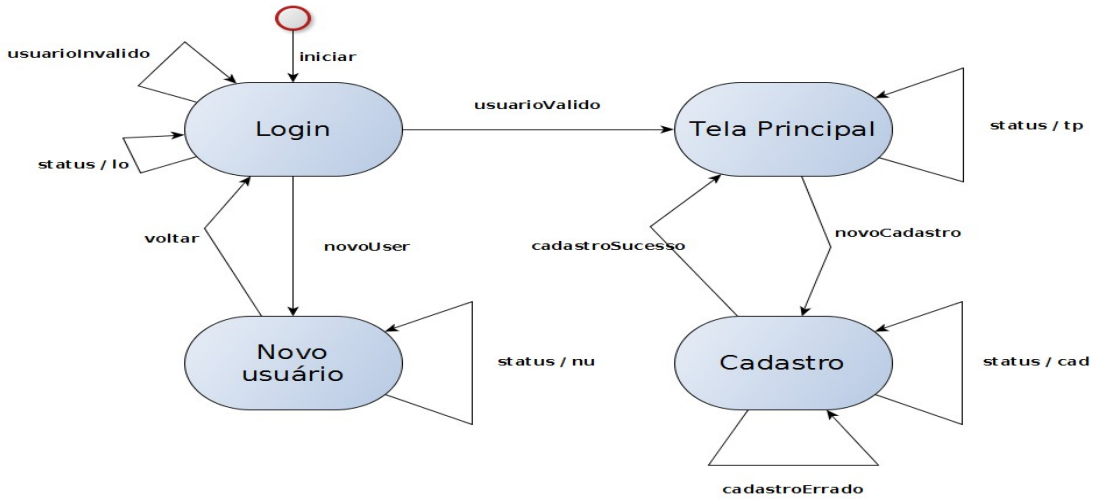
Gere os seguintes casos de teste a partir da máquina de estados apresentada:

- um caso de teste que aplica o evento status nos quatro estados
- um caso de teste que execute as transições que não foram cobertas pelo caso de teste anterior.

Concretize os casos de teste usando JUnit assumindo que exista uma classe que simula interação do usuário a seguir.

```
public class SimuladorDeEventos {

    //os metodos a seguir retornam true se conseguiram fazer a operação e false, caso contrário
    public boolean status() { return true;  }

    public boolean eventA() { return true;  }

    public boolean eventB() { return true;  }

    //TODO implemente os métodos para os outros eventos.
}
```

(Exercício 2) Considere a máquina de estados a seguir.



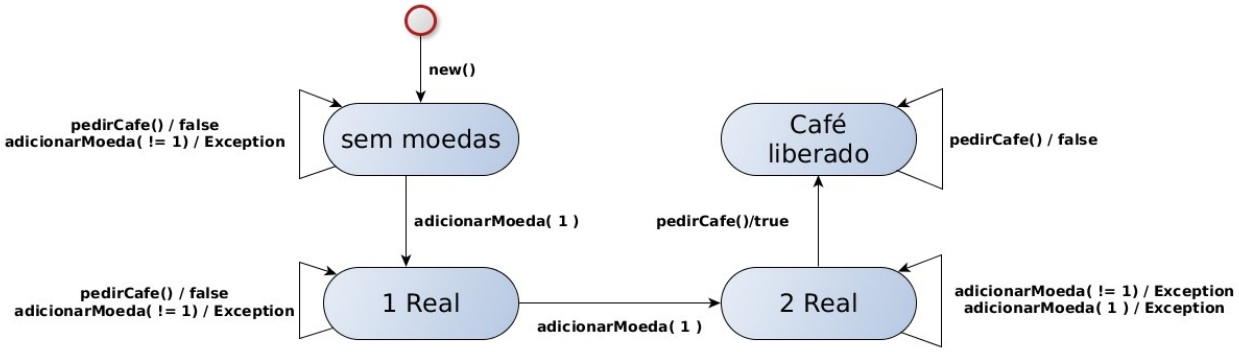
Gere os seguintes casos de teste a partir da máquina de estados apresentada:

- um caso de teste que aplica o evento status nos quatro estados
- um caso de teste que execute as transições que não foram cobertas pelo caso de teste anterior.

Concretize os casos de teste usando JUnit assumindo que exista uma classe que simula os eventos e saídas da máquina.

```
public class SimuladorDeEventos {

    //os metodos a seguir retornam true se conseguiram fazer a operação e false, caso contrário
    public String status() { return "...";  }

    public boolean iniciar() { return true;  }

    public boolean voltar() { return true;  }

    //TODO implemente os métodos para os outros eventos.
}
```

(Exercício 3) Considere a máquina de estados e a classe seguir:



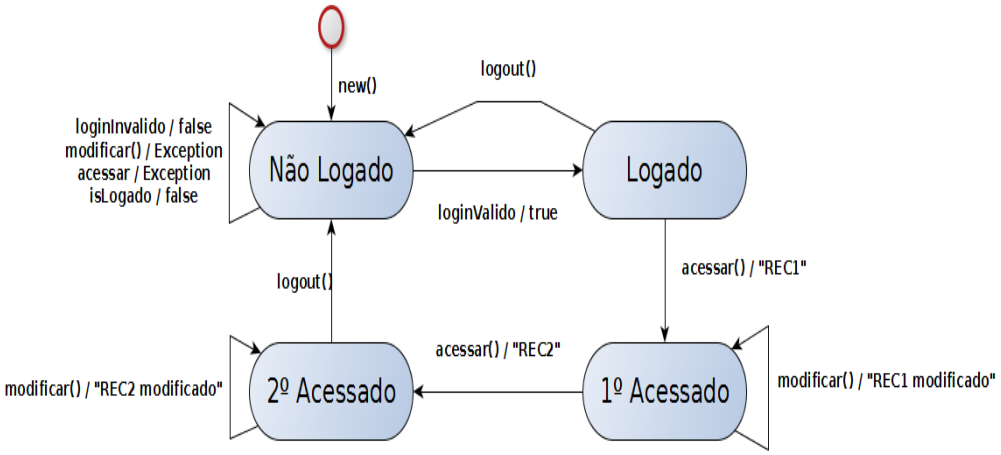

```
public class MaquinaCafe {
    private int saldo = 0;

    public boolean pedirCafe() {
        if(saldo == 2) {
            saldo = 0;        //cafe eh liberado
            return true;
        }
        return false;
    }

    public void adicionarMoeda(int valor) throws Exception {
        if(valor != 1)
            throw new Exception("maquina so aceita um real.");
        if(saldo == 2)
            throw new Exception("maquina nao aceita mais moedas.");

        saldo = saldo + valor;
    }
}
```

Gere os seguintes casos de teste a partir da máquina de estados apresentada:

- um caso de teste que aplica o evento “pedirCafe()” nos quatro estados
- um caso de teste que execute as transições que não foram cobertas pelo caso de teste anterior.

Concretize os casos de teste usando o JUnit.

(Exercício 4) Considere a máquina de estados a seguir.



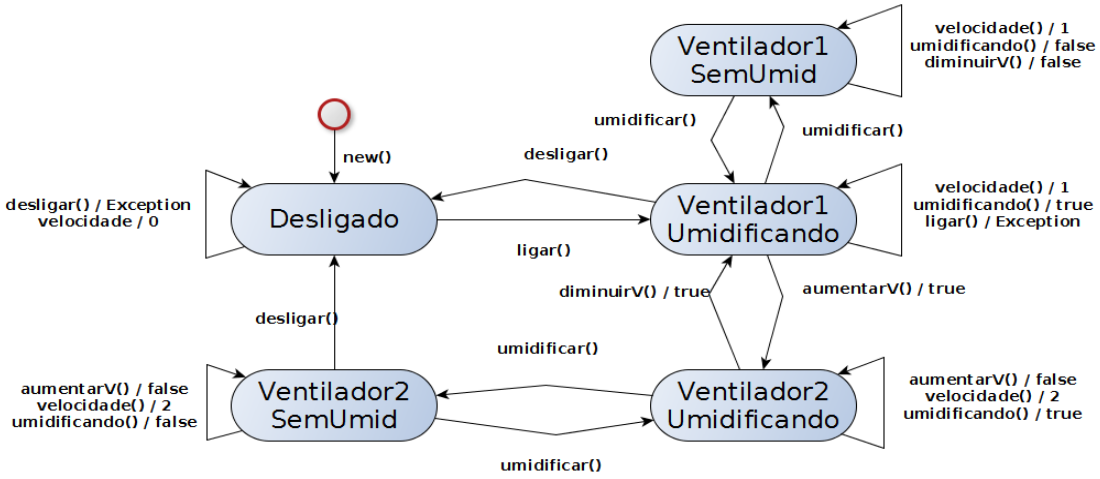
Gere os seguintes casos de teste a partir da máquina de estados apresentada:

- um caso de teste que testa se o logout() nos estados "logado" e "2º acesso" foi realizado com sucesso; use o método isLogado() para verificar.
- um caso de teste que testa o modificar() nos estados "não logado", "1º acesso" e "2º acesso".
- um caso de teste que execute as transições que não foram cobertas pelos casos de teste anterior.

Concretize os casos de teste usando JUnit assumindo que exista uma classe que simula os eventos e saídas da máquina.

```
public class SistemaDeRecurso {
   private boolean logado = false;
   private int rec = 0;

   public boolean login(String usuario, String senha) {
      if(usuario.equals("adm") && senha.equals("123"))
         logado = true;
      return logado;
   }

   public void logout() {
      logado = false;
      rec = 0;
   }

   public boolean isLogado() {  return logado;  }

   public String acessar() {
      if(! logado)
         throw new RuntimeException("nao pode acessar");

      return "REC" + (++rec);
   }

   public String modificar() {
      if(! logado)
         throw new RuntimeException("nao pode modificar");

      return "REC" + rec + " modificado";
   }
}
```

(Exercício 5) Considere a máquina de estados a seguir.



Gere os seguintes casos de teste a partir da máquina de estados apresentada:

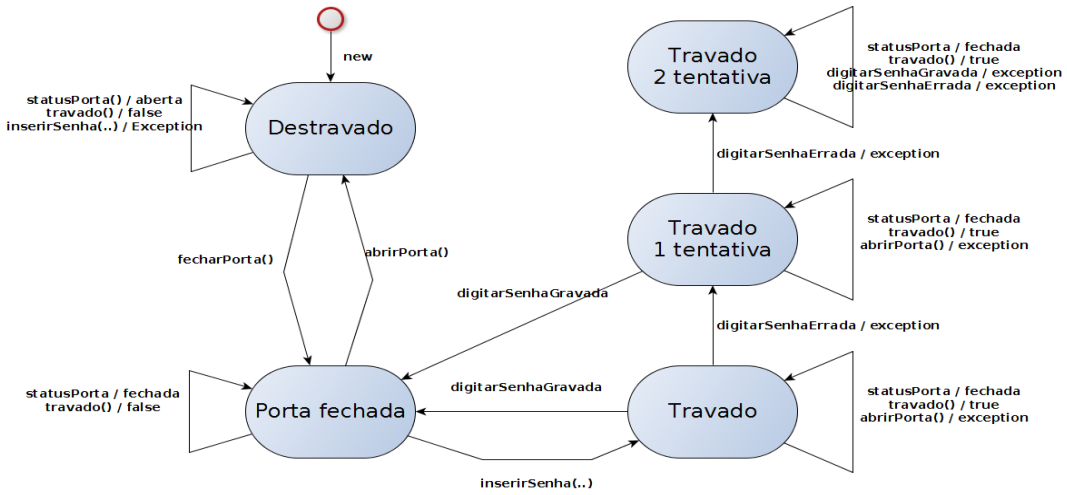
- um caso de teste que testa se o método *umidificar()* em todos os estados que ele acontece foi realizado com sucesso; use o método *umidificando()* para verificar.
- um caso de teste que testa o *desligar()* em todos os estados no quais ele acontece.
- um caso de teste que execute as transições que não foram cobertas pelos casos de teste anterior.

Concretize os casos de teste usando JUnit.

```
public class Climatizador {
   private boolean ligado = false, umid = false;
   private int velAtual = 0;

   public int velocidade() { return velAtual; }

   public boolean umidificando() { return umid; }

   public void umidificar() { umid = ! umid; }

   public void ligar() {
      if(ligado) throw new RuntimeException();

      ligado = umid = true;
      velAtual = 1;
   }

    //continua ao lado
```

```
   //continuacao

   public void desligar() {
      if(! ligado) throw new RuntimeException();

      ligado = umid = false;
      velAtual = 0;
   }

   public boolean aumentarV() {
      if(velAtual == 2) return false;

      velAtual++;
      return true;
   }

   public boolean diminuirV() {
      if(velAtual <= 1) return false;

      velAtual--;
      return true;
   }
}
```

(Exercício 6) Considere a máquina de estados de um cofre a seguir.



Gere os seguintes casos de teste a partir da máquina de estados apresentada:

- um caso de teste que percorre até o estado “Travado 2 tentativa” e executa suas 4 transições.
- um caso de teste que trava o cofre, retorna até o estado “Destravado” e tenta inserirSenha.
- um caso de teste que execute as transições que não foram cobertas pelos casos de teste anterior.

Concretize os casos de teste usando JUnit para a classe a seguir.

```
public class CofreEletronico {
        private boolean portaAberta = true;
        private boolean cofreTravado = false;
        private String senhaSalva = "";
        private int tentativas = 0;

        public String statusPorta() { return portaAberta ? "aberta" : "fechada"; }

        public boolean travado() { return cofreTravado; }

        public void abrirPorta() {
                if (cofreTravado) throw new RuntimeException("Nao eh possivel abrir a porta;
                                                        cofre travado com senha");

                portaAberta = true;
        }

        public void fecharPorta() { portaAberta = false; }

        public void inserirSenha(String senha) {
                if (portaAberta) throw new RuntimeException("Porta aberta");

                if (cofreTravado) throw new RuntimeException("Cofre ja possui senha definida");

                senhaSalva = senha;
                cofreTravado = true;
        }

        public void digitarSenha(String senha) {
                if (!senhaSalva.equals(senha) || tentativas >= 2) {
                        tentativas++;
                        var excMsg = tentativas <= 2 ? "Senha errada. Tente novamente" :
                                                "Cofre bloqueado por tentativas; reiniciar";
                        throw new RuntimeException(excMsg);
                }
                tentativas = 0;
                cofreTravado = false;
                senhaSalva = "";
        }
}
```