```
class VendingMachine

values
     public Capacity: real = 15;

types
     public String = seq of char;
     public Coins = nat
            inv c == c in set {100, 50, 20, 10, 5};
     public Boxes = nat
        inv b == b in set {1,...,20};
     public Products :: name: String
                          quantity : nat1
                          price : nat1;
     public ProductsInBoxes = map Boxes to [Products]
        inv pb == forall b in set dom pb & pb(b) = nil or
                     pb(b)<>nil => pb(b).quantity <= Capacity;

     public State = <init> | <configuration> | <idle>;

instance variables
  public stockProd: ProductsInBoxes;
  public stockCoins: map Coins to nat;
  public stateMachine: State := <init>;
  public prodSelected: [Boxes] := nil;
  public insertedCoins: seq of Coins := [];
  public coinsTroco: seq of Coins := [];

operations
  public VendingMachine : () ==> VendingMachine
  VendingMachine () ==
  (
    stateMachine := <configuration>;
    stockProd := { |-> };
    stockCoins := { |-> };
  )
  pre self.stateMachine = <init>;
```

Abastece máquina com um produto numa certa posição. Substitui o que se encontrava antes na mesma posição. Pressupõe a máquina em configuração.

```
  public SetStockProducts(num:Boxes, name:String,
                          price:nat1, quant:nat1) ==
    stockProd := stockProd ++ {num |-> mk_Products(name, quant, price)}
  pre self.stateMachine = <configuration> and
      quant <= Capacity and price mod 5 = 0;
```

Altera o *stock* de moedas. Pressupõe a máquina em configuração.

```
  public SetStockCoins(novoStockCoins: map Coins to nat) ==
    stockCoins := stockCoins ++ novoStockCoins
  pre self.stateMachine = <configuration>;
```

Consulta a quantidade em *stock* de um produto pelo número.

```
  public GetStockProducts(number: Boxes) res : nat1 ==
    return stockProd(number).quantity
  pre self.stateMachine = <configuration> and
      number in set dom self.stockProd;
```

Consulta o preço de um produto pelo número.

```
  public GetPriceProduct(number: Boxes) res : nat1 ==
    return stockProd(number).price
  pre self.stateMachine = <configuration> and
      number in set dom self.stockProd;
```

Fim de configuração

```
  public EndConfiguration() ==
    stateMachine := <idle>
  pre self.stateMachine = <configuration>;
```

Seleciona um produto pelo número. Depois de selecionar um produto pode inserir moedas.

```
  public SelectProduct(number : Boxes) ==
    prodSelected := number
  pre self.stateMachine = <idle> and
      number in set dom self.stockProd and
      self.stockProd(number) <> nil;
```

Para saber se ainda não inseriu o valor suficiente de moedas.

```
  public InsertedCoinsValue() res : nat ==
   (
     dcl sum:nat := 0;
     for all e in set inds insertedCoins do sum:=sum+insertedCoins(e);
     return sum;
   )
  pre self.prodSelected <> nil;
```

Insere uma moeda de um determinado valor. Deve estar um produto selecionado, e não devem ter sido inseridas moedas de valor suficiente.

```
  public InsertCoin(c: Coins) ==
    insertedCoins := insertedCoins ^ [c]
  pre self.prodSelected <> nil and
      self.InsertedCoinsValue()<
        self.stockProd(self.prodSelected).price;
```

Pergunta/observa o nome do produto selecionado.

```
  public GetNomeProdSel() res : String ==  return
    stockProd(prodSelected).name
  pre self.prodSelected <> nil;
```

Calcula o valor das moedas que fazem parte do troco

```
  public Sum(troco: seq of Coins) res : nat ==
  (   dcl sum:nat := 0;
      for all e in set inds troco do sum := sum + troco(e);
      return sum;
  );
```

Cancela a compra em curso

```
  public Cancelar() ==
  ( prodSelected := nil;  insertedCoins := []; coinsTroco := []  )
  pre self.prodSelected <> nil;
```

Simula utilizador a recolher o produto selecionado

```
  public RecolheProduto() == (
    coinsTroco := GiveChange();
```

```
    if (self.InsertedCoinsValue() >
      self.stockProd(self.prodSelected).price and coinsTroco = [])
    then insertedCoins := []
    else
    (
      if (stockProd(prodSelected).quantity<>1)
      then
        stockProd(prodSelected).quantity :=
        stockProd(prodSelected).quantity – 1
      else
        stockProd := {prodSelected} <-: stockProd ;
        coinsTroco := [];
     )
  )
  pre self.prodSelected in set dom self.stockProd;
```

Qual o valor total das moedas em stock?

```
  public SumMap(x:map Coins to nat) res:nat ==
   (dcl sum: nat := 0;
    for all e in set dom x do sum:=sum+e*x(e);
    return sum;
   );

  public GiveChange() res: seq of Coins ==
  (
    dcl sortedCoins: seq of Coins := [100,50,20,10,5];
    dcl troco : nat := InsertedCoinsValue() –
                       stockProd(prodSelected).price;
    coinsTroco := [];
    while (sortedCoins <> [])  do
    (
      if (hd sortedCoins in set dom stockCoins and
          stockCoins(hd sortedCoins) > 0 and
          hd sortedCoins <= troco)
      then
      (   coinsTroco := coinsTroco ^ [hd sortedCoins];
          if (stockCoins(hd sortedCoins)=1) then
            stockCoins := {hd sortedCoins} <-: stockCoins
          else
            stockCoins(hd sortedCoins) := stockCoins(hd sortedCoins)-1;
          troco := troco – hd sortedCoins;
      )
      else sortedCoins := tl sortedCoins
    );
    if (troco <> 0) then
    (
      for all e in set elems coinsTroco do
        stockCoins(e) := stockCoins(e)+1;
      coinsTroco := [];
    );
    return coinsTroco;
  )
  pre self.prodSelected <> nil and self.InsertedCoinsValue() >
      self.stockProd(self.prodSelected).price;

public Configuration () ==
```

```
  stateMachine := <configuration>
pre self.stateMachine = <idle>;

end VendingMachine
```