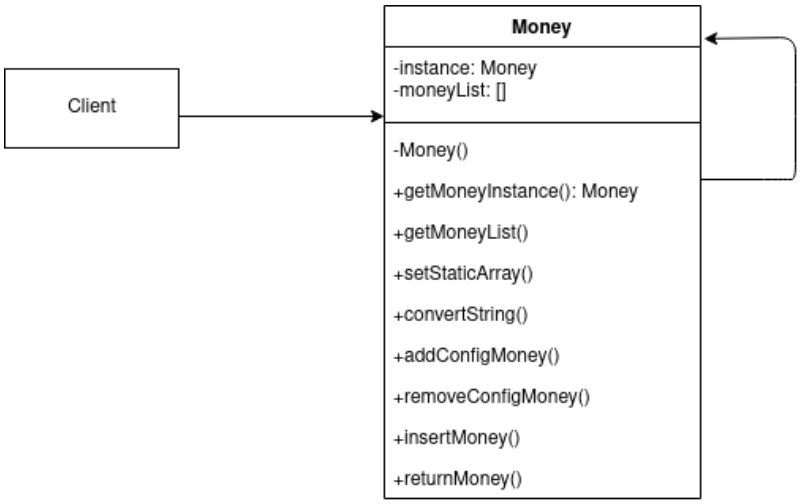
Design Patterns used to solve the Vending Machine Problem:

There Are 3 Core modules in this system.

1. Money Management Module
2. Product Management Module
3. Vending Machine Core Module.

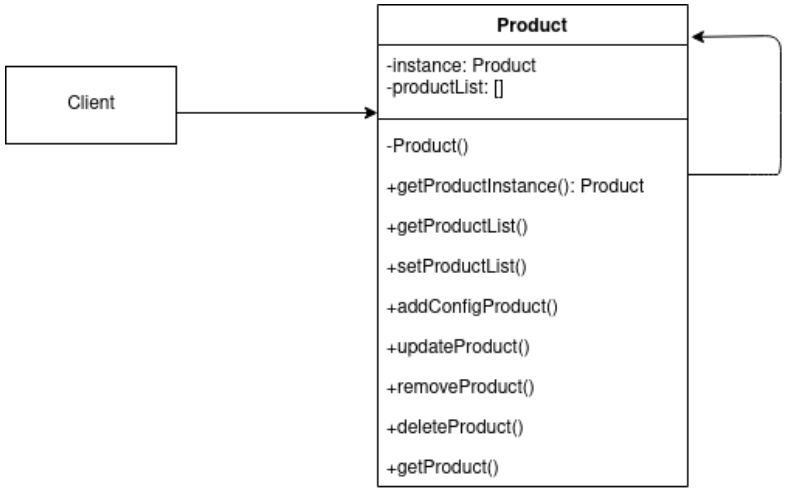
For the Money module and product module, we need to store some data at runtime and Those modules should be accessed from any other modules. As money and products have significant structures and no need for any other instances and they need to provide a global access point, **Singleton Design Pattern** is used to solve the problem.

For **money**, There is a Money class and has a static array named **moneyList** for holding all money indexes. This static array can be modified by any other operation happening by this class.



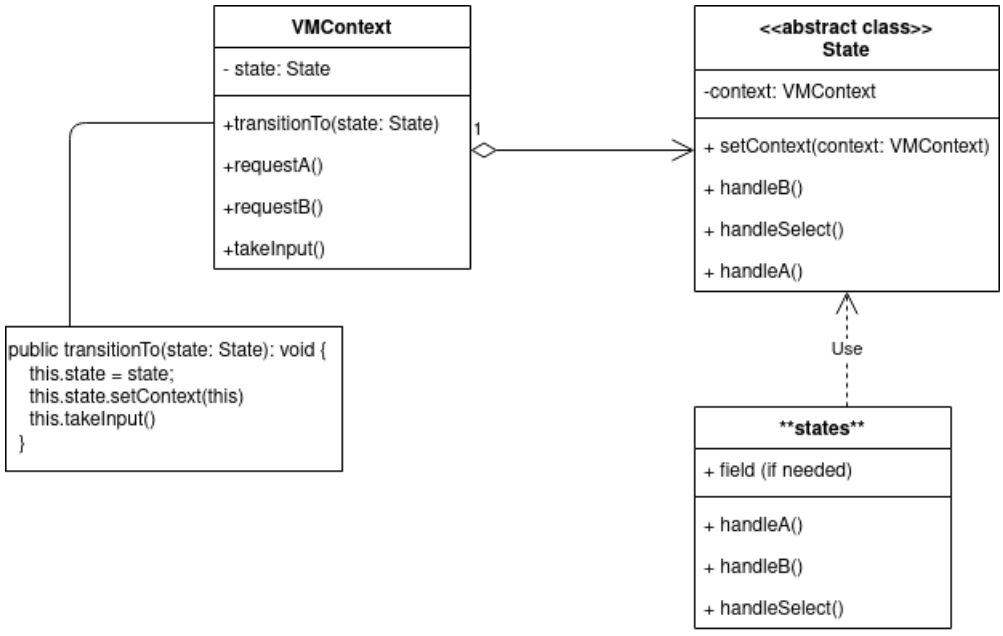
Singleton For Money Class

For **products**, There is a Product class and has a static array named **productList** for holding all money indexes. This static array can be modified by any other operation happening by this class.



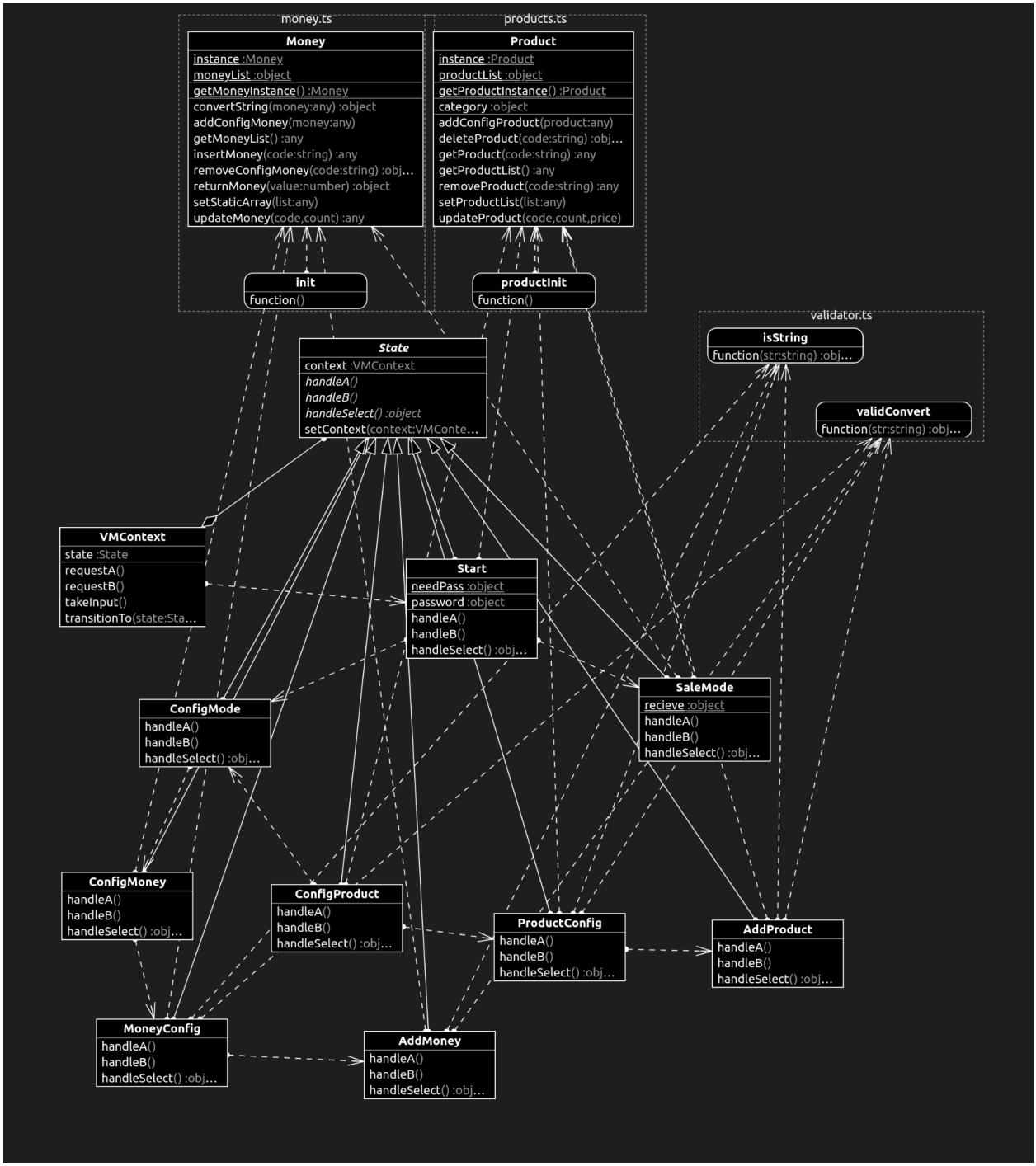
Singleton For Product class

For the core operation of **Vending Machine**, there are many states which depend on user decision by taking inputs from users. So there should be many states and they should be switched to another by user choice for serving different purposes. For solving this problem, **State Design Pattern** is used. The UML diagram of the classes is attached here:



State Design Pattern Structure

The Whole System’s Diagram is attached to the next page::


money.ts
Money
instance :Money
moneyList :object
getMoneyInstance() :Money
convertString(money:any) :object
addConfigMoney(money:any)
getMoneyList() :any
insertMoney(code:string) :any
removeConfigMoney(code:string) :obj…
returnMoney(value:number) :object
setStaticArray(list:any)
updateMoney(code,count) :any
products.ts
Product
instance :Product
productList :object
getProductInstance() :Product
category :object
addConfigProduct(product:any)
deleteProduct(code:string) :obj…
getProduct(code:string) :any
getProductList() :any
removeProduct(code:string) :any
setProductList(list:any)
updateProduct(code,count,price)
init
function()
productInit
function()
validator.ts
isString
function(str:string) :obj…
validConvert
function(str:string) :obj…
State
context :VMContext
handleA()
handleB()
handleSelect() :object
setContext(context:VMConte…
VMContext
state :State
requestA()
requestB()
takeInput()
transitionTo(state:Sta…
Start
needPass :object
password :object
handleA()
handleB()
handleSelect() :obj…
SaleMode
recieve :object
handleA()
handleB()
handleSelect() :obj…
ConfigMode
handleA()
handleB()
handleSelect() :obj…
ConfigMoney
handleA()
handleB()
handleSelect() :obj…
ConfigProduct
handleA()
handleB()
handleSelect() :obj…
ProductConfig
handleA()
handleB()
handleSelect() :obj…
AddProduct
handleA()
handleB()
handleSelect() :obj…
MoneyConfig
handleA()
handleB()
handleSelect() :obj…
AddMoney
handleA()
handleB()
handleSelect() :obj…