Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

1 Interface com o *Control*

Implementou-se o módulo *Control* em *software*, recorrendo a linguagem *Kotlin* seguindo a arquitetura lógica apresentada na Figura .

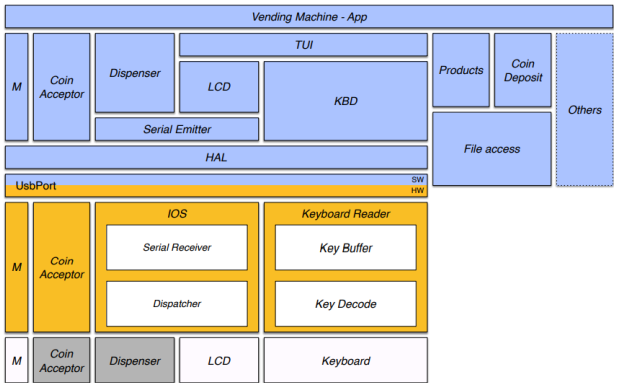

Figura 1 – Diagrama lógico da *vending machine* (máquina de vendas)

Os módulos de software *TUI*, *FILE ACESS*, *PRODUCTS*, *COINDEPOSIT*, *COINACCEPTOR*, *M*, *VENDINGMACHINE-APP* desenvolvidos são descritos nas secções **Error! Reference source not found.**., **Error! Reference source not found.**, 1.3, 1.4, 1.5, 1.6 e 1.7 e o código fonte desenvolvido nos Anexos D e E, respetivamente.

1.1 *TUI*

Na classe *TUI* esta apresenta as funções genéricas que serão utilizadas em diferentes momentos. O *TUI* apresenta quatro funções apresenta seis funções. (Podemos observar a classe no A dos anexos.)

1.1.1 Função *init*

Esta função tem como objetivo iniciar a classe na qual para os elementos em *keyArray* irá escrever os *costum Char* no local definido.

1.1.2 Função *writeCostumChar*

Esta função tem como objetivo escrever os *Chars costum* desenhados (o EURO e a ARROW) esta irá utilizar o *private val specialCharMap* de modo a escrever os caracteres no *LCD*.

1.1.3 Função *writeAligned*

Esta função tem como objetivo escrever a função alinhada de acordo com o desejado podendo colocar o texto desejado à esquerda, à direita ou ao centro.

1.1.4 Função *writeAlignedSupport*

Esta função tem como objetivo auxiliar a função *writeAligned* na qual utilizando as funções da classe *LCD* escrever o texto no local indicado.

1.1.5 Função *getKeyboardInput*

Esta função tem como objetivo receber as *keys* vindas do teclado matricial. Esta função irá entregar o Char recebido ou irá entregar *null* caso passe o tempo de *timeout* definido.

1.1.6 Função *clearLineLCD*

Esta função tem como objetivo fazer a limpeza de uma determinada linha do *LCD* deixando-a sem nada escrito de modo a ser possível utilizá-la novamente.

1.1.7 Função *clearLCD*

Esta função tem como objetivo fazer a limpeza de todo o LCD deixando este pronto para receber nova informação.

1.1.8 Função *getDate*

Esta função tem como objetivo receber a data e horas e apresentar, no caso da data, DD/MM/AA e no caso das horas HH/mm.

1.1.9 Função *writeProductBuyScreen*

Esta função tem como objetivo escrever o ecrã de compra de um produto fazendo aparecer neste o produto em questão escrevendo no meio e na linha de cima o nome do mesmo e por baixo o seu preço.

1.5.10 Função *writeMaintenceSomethingScreen*

Esta função tem como objetivo escrever o ecrã com a informação de escolha de um produto escrevendo "*something"* na linha superior e "*choose product"* na linha debaixo.

1.5.11 Função privada *writeYesNoScreen*

Esta função tem como objetivo escrever o ecrã

1.5.12. Função *writeCollectScreen*

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

Esta função tem como objetivo escrever a mensagem para recolher os produtos.

1.5.13. Função *writeThankYouScreen*

Esta função tem como objetivo escrever a mensagem para agradecer a compra dos produtos.

1.5.14. Função *writeCanceledOperationScreen*

Esta função tem como objetivo escrever a mensagem de cancelamento do produto.

1.5.15. Função *writeProductBuyScreen*

Esta função tem como objetivo escrever no *LCD* as informações da compra do produto que foi selecionado apresentando o nome do produto e o seu valor em euros.

1.5.16. Função *writeProductSelectScreen*

Esta função tem como objetivo escrever o produto o seu preço e a sua quantidade, caso o produto não exista este apresenta a mensagem a dizer que não está disponível

1.5.17. Função *writeDefaultScreen*

Esta função tem como objetivo escrever o Ecrã inicial o qual é consituido pela data, a hora e o texto *Vending-Machine* no centro

1.5.18 Função *writeDateTime*

Esta função tem como objetivo escrever a data utilizando as funções do TUI de modo a apresentar a data na forma desejada.

1.2 *FILEACESS*

Na classe *FileAcess* esta apresenta as funções que permitem o acesso aos ficheiros de texto, sendo estes o *PRODUCTS.txt* e o *CoinDeposit.txt*, de modo a possibilitar a escrita e a leitura de informações dos mesmos mediante a necessidade apresentada. O *FileAcess* apresenta duas funções principais e mais duas funções privadas que de modo a auxiliar estas funções. (Podemos observar a classe no B dos anexos.)

1.2.1 Função privada *creatReader*

Esta função permite a interação de leitura entre a função *readFile* e o ficheiro que se pretende ler

1.2.2 Função privada *createWriter*

Esta função permite a interação de escrita entre a função *writeFile* e o ficheiro em que se pretende escrever

1.2.3 Função *readFile*

Esta função tem como objetivo ler informação do ficheiro de modo a utilizar a mesma para qualquer finalidade desejada.

1.2.4 Função *writeFile*

Esta função tem como objetivo escrever informação no ficheiro de modo a alterar, apagar ou adicionar a mesma de acordo com o desejado.

1.3 PRODUCTS

Na classe *Products* esta apresenta as funções que realizam a interação dos produtos registados em *PRODUCTS.txt* de modo a permitir selecionar o produto e fazer alterações sobre a sua informação como por exemplo a sua quantidade. Esta classe tem 3 funções principais e uma função privada (Podemos observar a classe no C dos anexos.)

1.3.1 Função privada *Product.toStrings*

Esta função tem como objetivo devolver a informação dos parâmetros da *data class Product* para uma String.

1.3.2 Função *fetchProduct*

Esta função tem como objetivo retornar o Id do produto selecionado a partir das *keys* selecionadas ou então retorna null caso não seja selcionada nenhuma tecla escolhida.

1.3.3 Função *readFileProduct*

Esta função tem como objetivo ler a informação do ficheiro *PRODUCTS.txt* utilizando as funções da classe fileAcess.

1.3.4 Função *writeFileProducts*

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

Esta função tem como objetivo, utilizando as funções da classe *fileAcess* de modo a permitir escrever no ficheiro de texto *PRODUCTS.txt* de modo a alterar informações sobre o produto desejado.

1.4 *COINDEPOSIT*

Na classe *CoinDeposit* esta apresenta as funções que permitem depositar novas moedas e ler quantas já se encontram depositadas utilizando a informação escrita em *CoinDeposit.txt*. Esta classe apresenta só uma função. (Podemos observar a classe no D dos anexos.)

1.4.1 Função *depositCoinsOnFile*

Esta função tem como objetivo, utilizando a função *writeFile* da classe *FileAcess*, realizar o registo do número de moedas depositadas escrevendo estas no ficheiro *COINDEPOSIT.txt* de modo a guardar o seu valor total

1.5 COINACCEPTOR

Na classe *CoinAcceptor* esta apresenta as funções que permitem a interação entre o utilizador e o moedeiro e a *vending-machine* de modo a este conseguir inserir moedas, retirar as moedas entre outras ações. (Podemos observar a classe no E dos anexos.)

1.5.1 Função *init*

Esta função tem como objetivo iniciar a classe utilizando as funções da classe *HAL* para limpar a informação anterior do moedeiro.

1.5.2 Função *hasCoin*

Esta função tem como objetivo verificar se foi introduzida uma moeda no moedeiro e caso esta tenho recebido uma moeda retorna true.

1.5.3 Função *acceptCoin*

Esta função tem como objetivo informar o moedeiro que a moeda recebida foi contabilizada e guardando essa informação na variável CoinInInserter.

1.5.4 Função *ejectCoins*

Esta função é responsável por devolver as moedas que foram colocadas no moedeiro ao utilizador fazendo assim com que o coinInserter fique a 0.

1.5.5 Função *collectCoins*

Esta função é responsável por recolher as moedas que foram colocadas no moedeiro pelo utilizador de modo a comprar o roduto selecionado.

1.6 *M*

Na classe M existe uma única função que tem como objetivo ativar o modo de manutenção. Esta função é a função *isM()*, esta função faz com que se o modo de manutenção for ativo irá retornar *true* fazendo com que o modo fique ativo.(Pode se observar a classe M na documentação abaixo no ponto F)

1.7 *VENDINGMACHINE-APP*

Na classe V*endingMachine-App* esta apresenta as funções que permitem a interação com a máquina, ou seja, que permitem utilizar os menus e que apresentam um específico output dependendo a ação que o cliente tome podendo ser esta remover um produto, desligar a máquina, comprar um item dos disponíveis e alterar as informações de um produto. Esta função é constituída por cinco funções e por vinte e três funções privadas. (Pode se observar a classe *VendingMachine-App* na documentação abaixo no ponto G)

1.5.1 Função *main*

Esta função tem como objetivo colocar a *app* a funcionar a partir de uma única função de modo a iniciar a *vending-machine*

1.5.2 Função *init*

Esta função tem como objetivo iniciar a classe utilizando as outras classes já mencionadas antes de modo a permitir a sua utilização por outras funções.

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

1.5.3 Função *app*

Esta função tem como objetivo iniciar a aplicação fazendo a verificação se o modo manutenção está ativo, escrevendo a tela inicial e esperando para receber as *keys* que serão selecionadas.

1.5.4 Função privada *productDispense*

Esta função tem como objetivo realizar a dispensa de um produto começando então por limpar o ecrã do LCD de modo a escrever o produto indicado, a partir das teclas, no ecrã e esperando pela confirmação da seleção dos mesmo a partir da variável *asteriskFlag* dentro do tempo necessário e o pagamento do mesmo irá dispensar o produto.

1.5.5 Função privada *buyProduct*

Esta função tem como objetivo verificar a compra do produto indo descontando o valor das moedas colcocadas até ao preço do produto ser 0 de modo a permiter a função privada *productDispense* realize a dispnesa do produto

1.5.6 Função privada *cancelBuy*

Esta função tem como objetivo cancelar a compra do produto caso a compra não seja concluida.Irá tambéd ejetar as moedas que o utilizador tenha inserido.

1.5.7 Função *privada productBought*

Esta função tem como objetivo concluir a venda de um produto selecionado o que fará com que apareça a mensagem para se recolher o produto, irá retirar um à quantidade de produtos que se tem em "*stock*", irá dispensar o produto, irá escrever um agradecimento na tela voltando assim a escrever o menu inicial e a guardar as moedas

1.5.8 Função *privada updateKeys*

Esta função tem como objetivo atualizar as keys tendo em conta se o modo utilizado se for o modo das arrows usa a função *updateKeysForWalk caso* seja indicando o número do produto utilizando o teclado.

1.5.9 Função *privada updateKeysForNum*

Esta função tem como objetivo receber as keys quando o modo teclado matricial está ativo e entrega-as na função *updateKeys.*

1.5.10 Função *privada updateKeysForWalk*

Esta função tem como objetivo atualizar as *Keys* quando o teclado se encontra a ser usado no modo das setas de modo a saber se a tecla pode subir na lista de produtos ou se pode descer e entrega-a na função *updateKeys*

1.5.11 Função *privada getKeysReturnProduct*

Esta função tem como objetivo receber as *keys* de modo a apresentar o produto que for selecionado quer seja usando as setas quer seja usando os números. Os produtos irão continuar a ser mostrados até ser premida a tecla de confirmação ou o sistema fazer o *reset* para o ecrã inicial

1.5.12. Função privada *updateLastInput*

Esta função irá registar a última vez que um key foi pressionada de modo a permitir que o tempo de timeout seja garantido.

1.5.13. Função privada *resetToDefault*

Esta função tem como objetivo reesecrever o a *default Screen* sempre que necessário e colocar as flags e as keys como estavam inicialmente.

1.5.14. Função privada *updateDateTime*

Esta função tem como objetivo atualizar as horas e os minutos de acordo com a hora local.

1.5.15. Função *maintence*

Esta função tem como objetivo realizar todas as funções que o modo *maintence* permite, para tal é necessário o modo ser ativado e desta forma será apresentado no *LCD* indo mostrando as opções que o modo manutenção apresenta sendo estas desligar a máquina, adicionar a quantidade de um produto, apagar um produto e testar o funcionamento da máquina.

1.5.16. Função *privada maintenceDispenseTest*

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

Esta função tem como objetivo testar se tem algum erro na *vending-machine*

1.5.17. Função *privada maintenceUpdateProduct*

Esta função tem como objetivo atualizar o número de produtos existentes na máquina. Começando por selecionar esta opção a partir do menu esta irá escrever o texto designado para dar *update* no produto e pedirá o *id* do produto a confirmação do prudot usando a tecla # e de seguida pedirá a quantidade do produto e de seguida a confirmação com a tecla #.

1.5.18. Função *privada maintenceRemoveProduct*

Esta função tem como objetivo remover um produto. Começando por selecionar esta opção a partir do menu esta irá escrever o texto designado para a remoção do produto e pedirá o *id* do produto e a confirmação da ação fazendo com que este seja removido

1.5.19. Função *privada maintenceShutdown*

Esta função tem como objetivo desligar a máquina. Começando por selecionar esta opção a partir do menu esta irá escrever o texto designado para desligar a máquina e pedirá a confirmação na qual clicando no 5 aparecerá o texto a dizer *"bye bye" e* desligará a máquina e clicando no 8 irá abortar o comando fazendo com que volte ao menu anterior.

2 Conclusões

Concluindo o módulo do *dispenser* podemos temos a indicar que este apresenta-se concluído na sua totalidade sendo assim possível efetuar a compra de um produto e a manutenção da máquina de acordo com os parâmetros que nos foram entregues.

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

A. Código *Kotlin – TUI*

```kotlin
import isel.leic.utils.Time
import java.text.SimpleDateFormat
import java.util.*

object TUI {

    enum class AlignSide { LEFT, RIGHT, CENTER }

    enum class SpecialChar { EURO, ARROW }

    private val SpecialCharMap = mapOf(SpecialChar.EURO to 0, SpecialChar.ARROW to 1)
    private val EURO = (arrayOf(0x07, 0x08, 0x1E, 0x10, 0x1E, 0x08, 0x07, 0x00))
    private val ARROW = (arrayOf(0x04, 0x0E, 0x1F, 0x04, 0x1F, 0x0E, 0x04, 0x00))
    private val keyArray = arrayOf(EURO, ARROW)

    private const val dateHeight = 1

    fun init() {
        keyArray.forEach { LCD.writeCostumCharToCGRAM(it, keyArray.indexOf(it)) }
    }

    private fun writeCustomChar(specialChar: SpecialChar) {
        SpecialCharMap[specialChar]?.let { LCD.write(it) }
    }

    //Add condition for what line
    //adicionar para quando excede o numero de letras dar slide
    fun writeAligned(
        text: String,
        alignHeight: Int = 0,
        alignSide: AlignSide = AlignSide.LEFT,
        specialChar: SpecialChar? = null
    ) {
        when (alignSide) {
            AlignSide.RIGHT -> {
                writeAlignedSupport(
                    text,
                    alignHeight,
                    specialChar,
                    LCD.COLS - text.length - if (specialChar != null) 1 else 0
                )
            }
            AlignSide.LEFT -> {
                writeAlignedSupport(text, alignHeight, specialChar, 0)
            }
            AlignSide.CENTER -> {
                writeAlignedSupport(
                    text,
                    alignHeight,
                    specialChar,
                    (LCD.COLS - text.length - if (specialChar != null) 1 else 0) / 2
                )
            }
        }
    }
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

```
private fun writeAlignedSupport(text: String, alignHeight: Int = 0, specialChar: SpecialChar? = null, column: Int) {
    LCD.cursor(if (alignHeight in 0..LCD.LINES) alignHeight else LCD.LINES, column)
    LCD.write(text)
    if (specialChar != null) writeCustomChar(specialChar)
}

fun getKeyboardInput(timeout: Long): Char? {
    val value = KBD.waitKey(timeout)
    return if (value == KBD.DEFAULT_KEY) null else value
}

fun clearLineLCD(line: Int) {
    LCD.cursor(line, 0)
    repeat(LCD.COLS) { LCD.write(' ') }
    LCD.cursor(line, 0)
}

fun clearLCD() = LCD.clear()

private fun getDate(milliSeconds: Long, dateFormat: String): String? {
    val formatter = SimpleDateFormat(dateFormat)
    val calendar = Calendar.getInstance()
    calendar.timeInMillis = milliSeconds
    return formatter.format(calendar.time)
}

fun writeDoubleCentered(line1:String, line2:String){
    clearLCD()
    writeAligned(line1, 0, AlignSide.CENTER)
    writeAligned(line2, 1,AlignSide.CENTER)
}
fun writeYesNoScreen(action: String) {
    clearLCD()
    writeAligned(action, 0, AlignSide.RIGHT)
    writeAligned("Yes-5", 1)
    writeAligned("No-8", 1, AlignSide.RIGHT)
}

fun writeCollectScreen() {
    clearLineLCD(1)
    writeAligned("Collect Products", 1, AlignSide.CENTER)
}

fun writeProductBuyScreen(product: Products.Product) {
    clearLineLCD(1)
    writeAligned("${product.cost}", 1, AlignSide.CENTER)
    writeCustomChar(SpecialChar.EURO)
}
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

```
    fun writeProductSelectScreen(keysValue: Int, arrowMove: Boolean, product: Products.Product?) {
        clearLCD()
        if (product == null || product.amount == 0) {
            writeAligned(Products.PRODUCT_NA, 0, AlignSide.CENTER)
            writeAligned(
                keysValue.toString(),
                1,
                AlignSide.CENTER,
                if (arrowMove) SpecialChar.ARROW else null
            )
        } else {
            writeAligned(product.name, 0, AlignSide.CENTER)//
            writeAligned("${product.cost}", 1, AlignSide.RIGHT, SpecialChar.EURO)
            writeAligned(
                keysValue.toString().padStart(2, '0'),
                1,
                AlignSide.LEFT,
                if (arrowMove) SpecialChar.ARROW else null
            )
            writeAligned("#${product.amount.toString().padStart(2, '0')}", 1, AlignSide.CENTER)
        }
    }

    fun writeDefaultScreen(): Long {
        LCD.clear()
        writeAligned("Vending Machine", 0, AlignSide.CENTER)
        writeDateTime()
        return Time.getTimeInMillis() / 60000
    }

    fun writeDateTime() {
        writeAligned(getDate(Time.getTimeInMillis(), "dd/MM/yyyy hh:mm:ss.SSS")!!, dateHeight)
    }
}
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

B. Código *Kotlin* - *FILEACESS*

```kotlin
import java.io.BufferedReader
import java.io.FileReader
import java.io.PrintWriter

object FileAccess {
    private fun createReader(fileName: String): BufferedReader {
        return BufferedReader(FileReader(fileName))
    }

    private fun createWriter(fileName: String): PrintWriter {
        return PrintWriter(fileName)
    }

    fun readFile(fileName: String, size: Int): Array<String?>{
        val array: Array<String?> = arrayOfNulls(size)
        val productFile = createReader(fileName)
        var line: String?
        var count = 0
        line = productFile.readLine()
        while (line != null) {
            if(count < 16) {
                array[count] = line
                count++
            }
            line = productFile.readLine()
        }
        return array
    }

    fun writeFile(fileName: String, array: Array<String>) {
        val write = createWriter(fileName)
        array.forEach {
            if (it != "/")
                write.println(it)
        }
        write.close()
    }
}
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

C. Código *Kotlin - PRODUCTS*

```kotlin
object Products{

    data class Product(val id: Int, val name: String, var amount: Int, var cost: Double)

    const val PRODUCT_NA = "Not Available"

    fun fetchProduct(keys: Int, arrProducts: Array<Product?>,makePointer:Boolean): Product? {
        return if(keys in 0..15 &&  arrProducts[keys] != null){
            if(makePointer) arrProducts[keys]!! else arrProducts[keys]!!.copy()
        }
        else {
            null
        }
    }

    fun readFileProducts(): Array<Product?>{
        val array = FileAccess.readFile("PRODUCTS.txt", 16)
        val newArray: Array<Product?> = arrayOfNulls(16)
        array.forEach {
            if(it != null){
                val li = it.split(";")
                newArray[li[0].toInt()] = Product(li[0].toInt(), li[1], li[2].toInt(), li[3].toDouble())
            }
        }
        return newArray
    }

    private fun Product.toStrings(): String = "$id;$name;$amount;$cost"

    fun writeFileProducts(arrProducts: Array<Product?>){
        println(arrProducts.asList())
        val array: Array<String> = Array(arrProducts.size) { "/" }
        var count = 0
        arrProducts.forEach {
            if (it != null)
                array[count] = it.toStrings()
            count++
        }
        println(array.asList())
        FileAccess.writeFile("PRODUCTS.txt", array)
    }
}
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

D. Código *Kotlin - COINDEPOSIT*

```
object CoinDeposit {
    fun depositCoinsOnFile(amount : Int){
        FileAccess.writeFile("CoinDeposit.txt", arrayOf("$amount"))
    }
}
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

E. Código *Kotlin – COINACCEPTOR*

```
object CoinAcceptor { // Implementa a interface com o moedeiro.
    private const val COIN_INPUT = 0x20 // I5
    private const val COIN_ACCEPT = 0x10 // O4
    private const val COIN_COLLECT = 0x20 // O5
    private const val COIN_EJECT = 0x40 // O6
    private var coinsInAcceptor = 0
    var COIN_AMOUNT = 0

    // Inicia a classe
    fun init() {
        HAL.clrBits(COIN_ACCEPT)
        HAL.clrBits(COIN_COLLECT)
        HAL.clrBits(COIN_EJECT)
    }
    // Retorna true se foi introduzida uma nova moeda.
    fun hasCoin(): Boolean {
        return HAL.isBit(COIN_INPUT)
    }
    // Informa o moedeiro que a moeda foi contabilizada.
    fun acceptCoin() {
        HAL.setBits(COIN_ACCEPT)
        HAL.clrBits(COIN_ACCEPT)
        coinsInAcceptor++
    }
    // Devolve as moedas que estão no moedeiro.
    fun ejectCoins() {
        HAL.setBits(COIN_EJECT)
        HAL.clrBits(COIN_EJECT)
        coinsInAcceptor = 0
    }
    // Recolhe as moedas que estão no moedeiro.
    fun collectCoins() {
        HAL.setBits(COIN_COLLECT)
        HAL.clrBits(COIN_COLLECT)
        COIN_AMOUNT += coinsInAcceptor
        coinsInAcceptor = 0
    }
}
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

F. Código *Kotlin – M*

```
object M {
    private const val M = 0x10 //I4

    fun isM(): Boolean{
        return HAL.isBit(M)
    }
}
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

G. Código *Kotlin – VENDINGMACHINE-APP*

```kotlin
import isel.leic.utils.Time
import kotlin.math.pow
import kotlin.system.exitProcess

//full interlock na situacao da moeda entrar, para pagar
//Mudar a forma como as keys funcionam para a situacao do v=v*10+key (modulo de max)

fun main(args: Array<String>){
    App.app()
}

object App {
    //Faltam mambos do coin accept
    private const val ONLY_COIN_IN_THE_WORLD = 0.50
    private const val NUMBER_OF_DIGITS = 2
    private const val SELECT_TIMEOUT = 7000
    private const val MAXIMUM_NUM_OF_PRODUCTS = 15 //starts at 0 to


    private var keysValue: Int = 0
    private var lastKeyTime: Long = Time.getTimeInMillis()
    private var lastTD: Long = Time.getTimeInMillis() / 60000
    private var productArray: Array<Products.Product?> = Products.readFileProducts()
    private var arrowMove = false
    private var currentKey: Char? = null


    //very intressante

    private fun init() {
        HAL.init()
        KBD.init()
        SerialEmitter.init()
        LCD.init()
        Dispenser.init()
        CoinAcceptor.init()
        TUI.init()

    }
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

```
fun app() {
    init()
    TUI.writeDefaultScreen()
    while (true) {
        if (M.isM()) {
            maintenace()
            TUI.writeDefaultScreen()
        }
        updateKeys(50, true)
        if (currentKey == '#') {
            productDispense()
        } else {
            updateDateTime()
        }
    }
}

private fun productDispense() {
    currentKey = null
    val currentProduct = getKeysReturnProduct()
    if (currentKey == '#' && (currentProduct != null) && (currentProduct.amount >0)) {
        buyProduct(currentProduct)
        if (currentKey == '#') {
            cancelBuy()
        }
    }
    if ((currentProduct != null) && (currentProduct.cost == 0.00)) productBought(currentProduct)
    resetToDefault(true, true)
}

private fun buyProduct(currentProduct: Products.Product) {
    TUI.writeProductBuyScreen(currentProduct)
    currentKey = null
    while (currentKey != '#') {
        if (CoinAcceptor.hasCoin()) {
            CoinAcceptor.acceptCoin()
            currentProduct.cost -= ONLY_COIN_IN_THE_WORLD
            if (currentProduct.cost == 0.00) {
                break
            }
            TUI.writeProductBuyScreen(currentProduct)
        }
        updateKeys(50)
    }
}
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

```
private fun cancelBuy() {
    TUI.writeDoubleCentered("Operation","Canceled")
    Time.sleep(1000)
    resetToDefault(true, true)
    CoinAcceptor.ejectCoins()
}

private fun productBought(currentProduct: Products.Product) {
    TUI.writeCollectScreen()
    productArray[currentProduct.id]!!.amount -= 1
    Dispenser.dispense(currentProduct.id)
    TUI.writeDoubleCentered("Thank you","see you again")
    Time.sleep(1000)
    resetToDefault(true, true)
    CoinAcceptor.collectCoins()
}

private fun updateKeys(timeout: Long, justCurrentKey: Boolean = false) {
    currentKey = TUI.getKeyboardInput(timeout)
    if (!justCurrentKey) {
        if (arrowMove) updateKeysForWalk() else updateKeysForNum()
    }
    if (currentKey != null) updateLastInput()
}

private fun updateKeysForNum() {
    if (currentKey != null) {
        val keyInt = Character.getNumericValue(currentKey!!)
        if (keyInt in 0..10) {
            keysValue = (((keysValue * 10 + keyInt) % (10.0.pow(NUMBER_OF_DIGITS.toDouble()))).toInt())
        }
    }
}
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

```
private fun updateKeysForWalk() {
    if (currentKey == '2') {
        if (keysValue < MAXIMUM_NUM_OF_PRODUCTS) {
            do {
                keysValue += 1
            } while (productArray[keysValue] == null)
        } else {
            keysValue =
                MAXIMUM_NUM_OF_PRODUCTS
        }
        updateLastInput()
    }
    if (currentKey == '8') {
        if (keysValue > 0) {
            do {
                keysValue -= 1
            } while (productArray[keysValue] == null)
        } else {
            keysValue = 0
        }
        updateLastInput()
    }
}

private fun getKeysReturnProduct(): Products.Product? {
    var currentProduct = Products.fetchProduct(keysValue, productArray, false)
    TUI.writeProductSelectScreen(keysValue, arrowMove,currentProduct)
    var oldKeys: Int
    while (currentKey != '#' && !resetToDefault(false)) {
        oldKeys = keysValue
        updateKeys(50)
        currentProduct = Products.fetchProduct(keysValue, productArray, false)
        if (currentKey == '*') {
            arrowMove = !arrowMove
            if(keysValue> MAXIMUM_NUM_OF_PRODUCTS) keysValue= MAXIMUM_NUM_OF_PRODUCTS
        }
        if (!(oldKeys == keysValue && currentKey != '*')) {
            TUI.writeProductSelectScreen(keysValue, arrowMove,currentProduct)
        }
    }
    return currentProduct
}
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

```
private fun resetToDefault(justReset: Boolean, resetScreen: Boolean = false): Boolean {
    if (lastKeyTime + SELECT_TIMEOUT <= Time.getTimeInMillis() || justReset) {
        keysValue = 0
        currentKey = null
        updateLastInput()
        if (resetScreen) TUI.writeDefaultScreen()
        return true
    }
    return false
}

private fun updateLastInput() {
    lastKeyTime = Time.getTimeInMillis()
}

private fun updateDateTime() {
    if (Time.getTimeInMillis() / 60000 != lastTD) {
        lastTD = Time.getTimeInMillis() / 60000
        TUI.writeDateTime()
    }
}
```

```
private fun maintenace() {
    while (M.isM()) {
        var key: Char?
        while (true) {
            TUI.clearLCD()
            TUI.writeAligned("Maintenance Mode")
            TUI.writeAligned("1-Dispense Test", 1)
            key = TUI.getKeyboardInput(1000)
            if (key != null || !M.isM()) break
            TUI.clearLineLCD(1)
            TUI.writeAligned("2-Update Prod.", 1)
            key = TUI.getKeyboardInput(1000)
            if (key != null || !M.isM()) break
            TUI.clearLineLCD(1)
            TUI.writeAligned("3-Remove Prod.", 1)
            key = TUI.getKeyboardInput(1000)
            if (key != null || !M.isM()) break
            TUI.clearLineLCD(1)
            TUI.writeAligned("4-Shutdown", 1)
            key = TUI.getKeyboardInput(1000)
            if (key != null || !M.isM()) break
        }
        when (key) {
            '1' -> maintenanceDispenseTest()
            '2' -> maintenanceUpdateProd()
            '3' -> maintenanceRemoveProd()
            '4' -> maintenanceShutdown()
        }
    }
}

private fun maintenanceDispenseTest() {
    TUI.writeDoubleCentered("Test","Choose Product")
    val currentProduct = getKeysReturnProduct()
    if (currentKey == '#' && currentProduct != null && currentProduct.amount!=0) {
        productBought(currentProduct)
    }
    resetToDefault(true)
}
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

```
    private fun maintenanceUpdateProd() {
        TUI.writeDoubleCentered("Update","Choose Product")
        val currentProduct: Products.Product? = getKeysReturnProduct()
        if (currentKey == '#' && currentProduct != null) {
            TUI.clearLCD()
            resetToDefault(true, false)
            while (true) {
                TUI.writeDoubleCentered(currentProduct.name,keysValue.toString().padStart(2, '0'))
                updateKeys(5000)
                if (currentKey == '#') {
                    productArray[currentProduct.id]!!.amount = currentProduct.amount + keysValue
                    break
                }
            }
        }
        resetToDefault(true)
    }


    private fun maintenanceRemoveProd() {
        TUI.writeDoubleCentered("Remove","Choose Product")
        val currentProduct: Products.Product? = getKeysReturnProduct()
        if (currentKey == '#' && currentProduct != null) {
            TUI.writeYesNoScreen("Remove product")
            while (currentKey == '#') {
                when (TUI.getKeyboardInput(1000)) {
                    '5' -> {
                        productArray[currentProduct.id] = null
                        break
                    }
                    '8' -> break
                }
            }
        }
        resetToDefault(true)
    }

      private fun maintenanceShutdown() {
          TUI.writeYesNoScreen("Shutdown")
          while (true) {
              when (TUI.getKeyboardInput(5000)) {
                  '5' -> {
                      LCD.clear()
                      LCD.write("Bye Bye")
                      Time.sleep(500)
                      Products.writeFileProducts(productArray)
                      CoinDeposit.depositCoinsOnFile(CoinAcceptor.COIN_AMOUNT)
                      exitProcess(0)
                  }
                  else -> break
              }
          }
      }
}
```