Licenciatura em Engenharia Informática e de Computadores

Máquina de venda
(*Vending Machine*)

Docentes:
Pedro Miguens Matutino (pedro.miguens@isel.pt)
Nuno Sebastião (nuno.sebastiao@isel.pt)
Sérgio André (sergio.andre@isel.pt)

Autores:
Bernardo Serra (47539@alunos.isel.pt)
Pedro Raposo (48316@alunos.isel.pt)
Rafael Costa (48315@alunos.isel.pt)

Projeto
de
Laboratório de Informática e Computadores
2021 / 2022 inverno

04 de outubro de 2021

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

1 Introdução

Neste projeto implementa-se um sistema de controlo de uma máquina de venda (*Vending Machine*), onde são armazenados diferentes tipos de produtos. A máquina permite armazenar até 16 tipos de produtos e no máximo 20 produtos de cada tipo. A seleção do produto é realizada digitando o identificador do produto ou através das teclas ↑ e ↓, sendo exibido num ecrã o identificador do produto, o nome, a quantidade existente e o preço. A ordem de dispensa é dada através da pressão da tecla de confirmação, sendo dispensada uma unidade do produto exibido no ecrã.

Para além do modo de Dispensa, o sistema tem mais um modo de funcionamento designado por Manutenção, que é ativado por uma chave de manutenção. Este modo permite a gestão dos produtos disponíveis na máquina, selecionando-se através do teclado, o tipo de produto que se pretende visualizar, carregar ou anular.

O sistema de controlo da máquina de venda é constituído por um teclado de 12 teclas, um moedeiro (designado por *Coin Acceptor*), um ecrã *Liquid Cristal Display* (LCD) de duas linhas de 16 caracteres, um mecanismo de dispensa de produtos (designado por *Dispenser*) e uma chave de manutenção (designada por *M*) que define se o dispensador está em modo de Manutenção, conforme o diagrama de blocos apresentado na Figura 1.

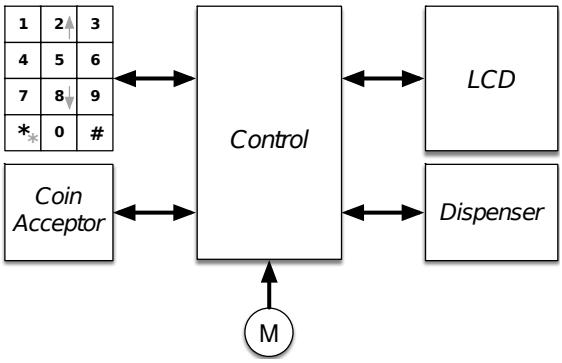


Figura 1 – Diagrama de blocos do sistema de controlo da máquina de venda (*Vending Machine*)

Sobre o sistema podem-se realizar as seguintes ações em modo Dispensa:

- **Consulta e dispensa** – A consulta de um produto, é realizada digitando o identificador do produto ou listando os produtos através das teclas ↑ e ↓, a dispensa deste é realizada após confirmação dada pela tecla '#' e a inserção do respetivo valor monetário. Durante a dispensa ficam afixados no LCD as informações referentes ao produto até que o mecanismo de dispensa confirme que a dispensa já foi efetuada. O modo de seleção ↑ e ↓ alterna com a seleção numérica por pressão da tecla '*'.

Sobre o sistema podem realizar-se as seguintes ações no modo Manutenção:

- **Carregamento** – Para cada produto é possível estabelecer qual a quantidade existente. O carregamento de cada produto é iniciado, pela seleção da opção no menu, seguido da seleção do identificador do produto e da quantidade. Após inserir o identificador prime-se a tecla de confirmação '#', para de seguida inserir a quantidade seguida da tecla '#' para concretizar o carregamento do produto.
- **Anulação** – Para anular um tipo de produto seleciona-se a operação de anulação no menu, seguindo-se a seleção do identificador de produto finalizando com a tecla de confirmação '#'.
- **Desligar** – O sistema desliga-se ao selecionar-se esta opção no menu, ou seja, o software de gestão termina armazenando as estruturas de dados de forma persistente em ficheiros de texto. O ficheiro contendo a informação dos produtos, deve estar organizado com uma linha por cada produto, em que os campos de dados são separados por ";", com o formato "*SLOT;NAME;STOCK;PRICE*" que é carregado no início do programa e reescrito no final do programa.

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

Nota: A inserção de informação através do teclado tem o seguinte critério: *i*) se não for premida nenhuma tecla num intervalo de cinco segundos o comando em curso é abortado; *ii*) quando o dado a introduzir é composto por mais que um dígito, são considerados apenas os últimos dígitos, a inserção realiza-se do dígito de maior peso para o de menor peso.

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

2 Arquitetura do sistema

O sistema é implementado numa solução híbrida de hardware e software, como apresentado no diagrama de blocos da Figura 2. A arquitetura proposta é constituída por três módulos principais: *i*) um leitor de teclado, designado por *Keyboard Reader*; *ii*) um módulo de interface com o LCD e com o mecanismo de dispensa, designado por *Integrated Output System* (*IOS*); e *iii*) um módulo de controlo, designado por *Control*. Os módulos *i*) e *ii*) deverão ser implementados em *hardware*, enquanto o módulo de controlo deverá ser implementado em *software* usando linguagem *Kotlin* executado num PC.

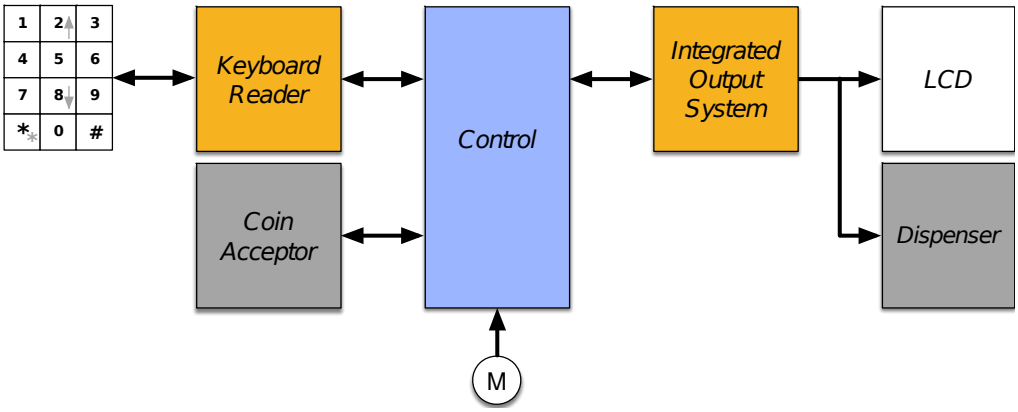
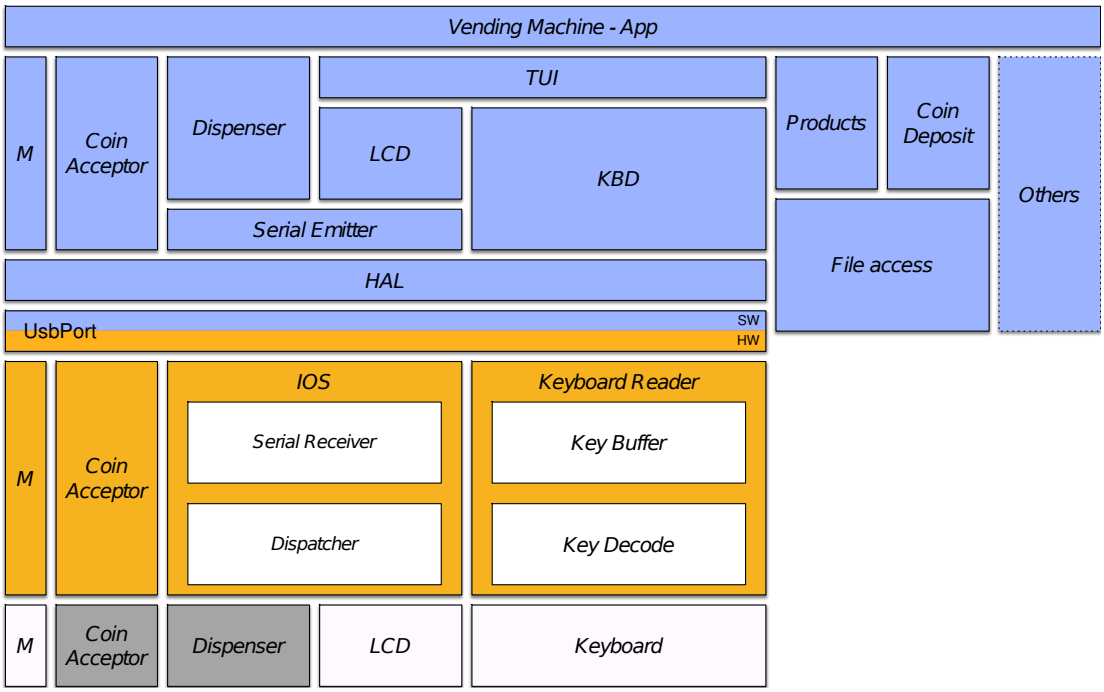


Figura 2 – Arquitetura do sistema que implementa a máquina de venda (*Working Time Recorder*)

O módulo *Keyboard Reader* é responsável pela descodificação do teclado matricial de 12 teclas, determinando qual a tecla pressionada e disponibilizando o seu código, com quatro bits, ao módulo *Control*. Caso este não esteja disponível para o receber imediatamente, o código da tecla é armazenado até ao limite de dois códigos. O módulo *Control* processa os dados e envia a informação a apresentar no *LCD* através do módulo *IOS*. O mecanismo de dispensa, designado por *Dispenser*, é atuado pelo módulo *Control*, através do módulo *IOS*. Por razões de ordem física, e por forma a minimizar o número de fios de interligação, a comunicação entre o módulo *Control* e o módulo *IOS* é realizada recorrendo a um protocolo série.



Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

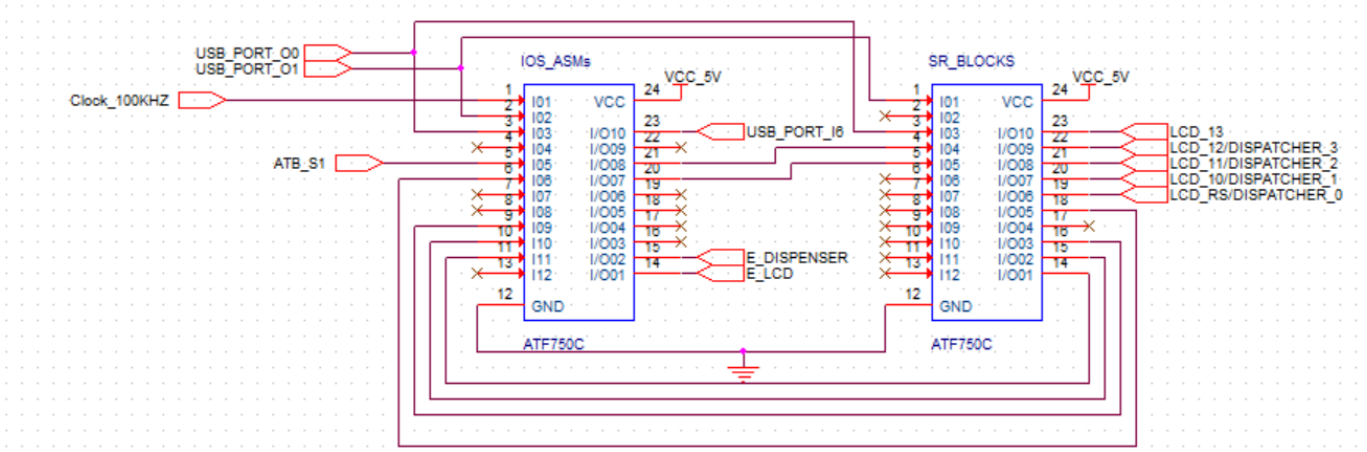
Figura 3 – Diagrama lógico do sistema de controlo da máquina de venda (*Vending Machine*)

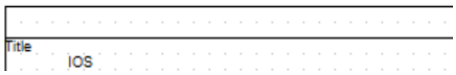
A. Interligações entre o HW e SW

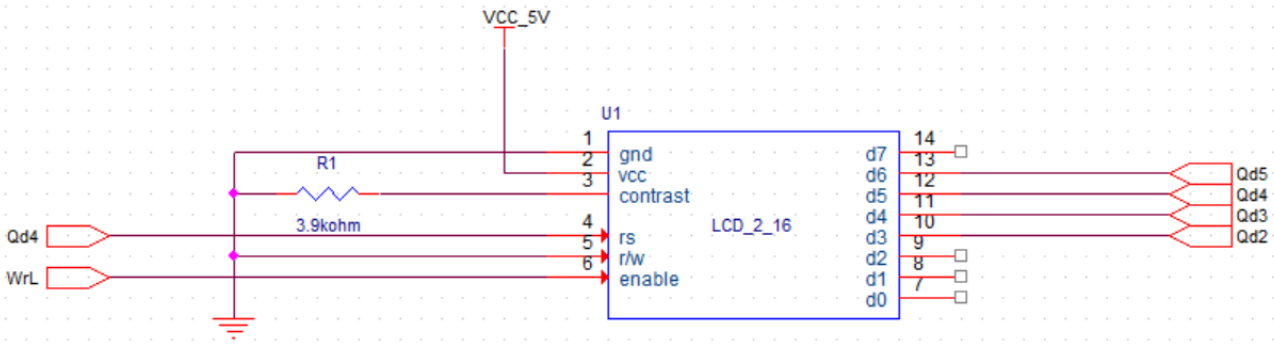
Neste módulo podemos observar através dos esquemas elétricos quais as ligações que são feitas entre o *Usb-Port* e a *ATB* com as *PAL's ATF750C* usadas para implementar o *Hardware.* Para além destas ligações o *Usb-Port* também se encontra ligado a um computador com o *software* em *Kotlin.* Podemos também observar as ligações entre o *Hardware* e o *software* a partir dos esquemas elétricos e do *hardware.simul*

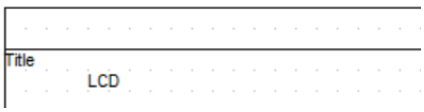
```
## --- Additional packages with Hardware to Simulator
package vendingmachine.simul
usbport = UsbPort

# Generic modules to activate from Simulator
kbd  = Keyboard("123456789*0#",4,3,0)
lcd  = LCD
m    = Switch ; "manut" ; setLabel("M")

# Costume modules from Vending Machine package
kb  = KeyBuffer
coin = CoinAcceptor
sr = SerialReceiverParity(5,6)
dp  = Dispatcher
dproduct = Dispenser

## --- Project Links ---
# -------------------------------------
# KeyBoard Reader
# -------------------------------------
# Key Decode
1 -> kbd.oe
kbd.K[0-3]      -> kb.D[0-3]
kb.K[0-3]       -> usbport.I[0-3]
kbd.val         -> kb.DAV

# Key Buffer
kb.Dval         -> usbport.I7
usbport.O7      -> kb.ACK
kb.DAC          -> kbd.ack

# -------------------------------------
# IOS (Integrated Output System)
# -------------------------------------


# SERIAL
usbport.O0 -> sr.SDX
usbport.O1-> sr.SCLK
dp.done -> sr.accept
sr.busy -> usbport.I6

# Dispatcher
sr.DXval -> dp.Dval
sr.D[1-5] -> dp.I[0-4]
sr.D0 -> dp.LnD

# -------------------------------------
# LCD
# -------------------------------------
dp.D[1-4] -> lcd.D[4-7]
dp.D0 -> lcd.rs
dp.WrL -> lcd.e

# -------------------------------------
# Dispenser
# -------------------------------------
dp.WrD -> dproduct.Ej
dp.D[0-3] -> dproduct.PID[0-3]
dp.finish -> dproduct.Fn

# -------------------------------------
# Manut
# -------------------------------------
m.out           ->  usbport.I4

# -------------------------------------
# Coin Acceptor
# -------------------------------------
coin.coin       ->  usbport.I5
coin.accept     ->  usbport.O4
coin.collect    ->  usbport.O5
coin.eject      ->  usbport.O6
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315









Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

B. Código *Kotlin* da classe *HAL*

A classe *HAL* faz a ligação entre o software e o hardware, mais propriamente entre o *Usb-Port (hardware)* e o código escrito em *kotlin (software)*. As funções que se encontram nesta classe permitem ler os valores que se encontram na entrada do *Usb-Port*, ou escrever na saída do mesmo a partir de uma máscara especifica.

```kotlin
import isel.leic.UsbPort

object HAL {   // Virtualiza o acesso ao sistema UsbPort
    // Inicia a classe
    private var usbPortOut = 0x00
    fun init() {
        usbPortOut = 0x00
        doOutput()
    }

    // Retorna true se o bit tiver o valor lógico '1', tem teste
    fun isBit(mask: Int): Boolean = readBits(mask) == mask

    // Retorna os valores dos bits representados por mask presentes no UsbPort, tem teste
    fun readBits(mask: Int): Int = (getInput() and mask)

    // Escreve nos bits representados por mask o valor de value, tem teste
    fun writeBits(mask: Int, value: Int) {
        usbPortOut = (usbPortOut and mask.inv()) or (value and mask)
        doOutput()
    }

    // Coloca os bits representados por mask no valor lógico '1', tem teste
    fun setBits(mask: Int) {
        usbPortOut = usbPortOut or mask
        doOutput()
    }

    // Coloca os bits representados por mask no valor lógico '0', tem teste
    fun clrBits(mask: Int) {
        usbPortOut = usbPortOut and mask.inv()
        doOutput()
    }

    private fun doOutput() = UsbPort.out(usbPortOut.inv())

    private fun getInput() : Int {
        UsbPort.`in`()
        return UsbPort.`in`().inv()
    }
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

C. Código *Kotlin* da classe *KBD*

A classe *KBD* faz a interface de ligação entre o teclado matricial(4x3) e o software. Na função *waittKey* esta irá retornar a tecla premida no teclado ou *NONE* se o *timeout* milisegundos for atingido. Esta função utilizará a função *getKey* que irá devolver instantaneamente a tecla que for premida no teclado ou NONE se não tiver a tecla premida.

```
1   import isel.leic.UsbPort
2   import isel.leic.utils.Time
3
4   object KBD {  // Ler teclas. Métodos retornam '0'..'9','#','*' ou NONE.
5       private const val NONE = 0
6       const val DEFAULT_KEY = NONE.toChar()
7       private const val KEY = 0x0F
8       private const val VAL = 0x80
9       private const val ACK = 0x80
10      private val keyArray = arrayOf('1','2','3','4','5','6','7','8','9','*','0','#')
11      // Inicia a classe
12      fun init() {
13          HAL.clrBits(ACK)
14      }
15
16      // Retorna de imediato a tecla premida ou NONE se não há tecla premida.
17      fun getKey(): Char {
18          var key = DEFAULT_KEY
19          if (HAL.isBit(VAL)) {
20              key = keyArray[HAL.readBits(KEY)]
21              HAL.setBits(ACK)
22              while (HAL.isBit(VAL)) {}
23              HAL.clrBits(ACK)
24          }
25          return key
26      }
27
28      // Retorna quando a tecla for premida ou NONE após decorrido 'timeout' milisegundos.
29      fun waitKey(timeout: Long): Char {
30          var key = DEFAULT_KEY
31          val timeToStop = Time.getTimeInMillis()+timeout
32          while(Time.getTimeInMillis()<timeToStop){
33              key = getKey()
34              if (key!= DEFAULT_KEY) break
35          }
36          return key
37      }
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

D. Código *Kotlin* da classe *SerialEmitter*

A classe *SerialEmitter* irá interagir com os diferentes módulos do *Serial Receiver*, utilizando as funções desenvolvidas na classe *HAL*. Partindo da função *send* esta irá enviar para o *SerialReceiver,* de acordo com o destino especificado em *addr*, uma trama com os dados presentes em *data*.

```
import isel.leic.utils.Time

object SerialEmitter {  // Envia tramas para os diferentes módulos Serial Receiver.

    private const val BUSY = 0x40
    private const val SDX = 0x01
    private const val SCLK = 0x02
    private var BITS = 4

    enum class Destination { DISPENSER, LCD }

    // Inicia a classe
    fun init() {
        HAL.setBits(SDX)
        HAL.clrBits(SCLK)
    }

    fun send(addr: Destination, data: Int) {

        while(isBusy());
        var p = 0
        HAL.clrBits(SDX)
        if(Destination.LCD == addr) {
            HAL.setBits(SDX)
            p++
            BITS = 5
        }
        if(Destination.DISPENSER == addr) {
            BITS = 4
        }
        for(i in 0 until BITS){
            val bit = (data shr i) and 0x1
            p += bit
            clockSCLK(bit)
        }
        val parityCheck = p.inv() and 0x1
        clockSCLK(parityCheck)
        clockSCLK(1)
    }

    //muda o valor do bit na posicao SDX
    private fun changeSDXBit (int: Int){
        return when(int){
            1->{
                HAL.setBits(SDX)
            }
            else->{
                HAL.clrBits(SDX)
            }
        }
    }
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

```
    private fun changeSDXBit (int: Int){
        return when(int){
            1->{
                HAL.setBits(SDX)
            }
            else->{
                HAL.clrBits(SDX)
            }
        }
    }

    // Retorna true se o canal série estiver ocupado
    private fun isBusy(): Boolean {
        return HAL.isBit(BUSY)
    }

    private fun clockSCLK(bit:Int) {
        HAL.setBits(SCLK)
        changeSDXBit(bit)
        HAL.clrBits(SCLK)
    }
}
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

E. Código *Kotlin* da classe *LCD*

A classe *LCD* permite fazer a interface do *LCD*(*software*) utilizando as funções criadas para o mesmo permitindo assim iniciar o *LCD* de forma correta com a função *init*, escrever no *LCD* com as funções *write (*quer seja um *Char,* uma *String*, ou *Data)*na posição atual, posicionar o cursor numa posição desejada com a função *cursor* e por fim a limpeza do *LCD* com a função *clear.*

```
66      // Faz o efeito do writeNibble, mas em paralelo
67      private fun writeNibbleParallel(rs: Boolean, data: Int){
68          //e,rs,outro
69          val rsBit = if(rs) RS_MASK else 0
70          HAL.writeBits(RS_MASK,rsBit)
71          Time.sleep(1)
72          HAL.writeBits(RS_MASK+E_MASK,rsBit + E_MASK)
73          Time.sleep(1)
74          HAL.writeBits(RS_MASK+E_MASK+DATA_BITS_MASK,rsBit+E_MASK+data)
75          Time.sleep(1)
76          HAL.clrBits(E_MASK)
77      }
78
79      // Escreve um byte de comando/dados no LCDprivate
80      private fun writeByte(rs: Boolean, data: Int){
81          writeNibble(rs,data shr HALF_BYTE)
82          writeNibble(rs,(data and DATA_BITS_MASK))
83      }
84
85      // Escreve um comando no LCD
86      private fun writeCMD(data: Int) = writeByte(false,data)
87
88      // Escreve um dado no LCD
89      fun writeDATA(data: Int) = writeByte(true,data)
90
91      // Escreve um caráter na posição corrente. Pode rebentar, .toInt() -> .code
92      fun write(c: Char) = writeDATA(c.code)
93
94      // Escreve uma string na posição corrente.
95      fun write(text: String){
96          for(element in text) { write(element) }
97      }
98
99      // Envia comando para posicionar cursor (‘line’:0..LINES-1 , ‘column’:0..COLS-1)
100     fun cursor(line: Int,column: Int) = writeCMD(SET_DDRAM_ADDRESS+column+(if(line!=0)line*DDRAM_ADDRESS_NEXT_LINE else 0))
101
102     // Envia comando para limpar o ecrã e posicionar o cursor em (0,0)
103     fun clear() = writeCMD(CLEAR_DISPLAY)
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

F. Código *Kotlin* da classe *Dispenser*

Na classe *Dispenser* esta permite enviar a informação necessária para o *Dispenser* saber qual o produto a dispensar. Este processo é feito pela função *dispense* utilizando a classe do *Serial Emitter* de modo a este ir para o *Dispenser* e entregando o *productID* de maneira a identificar o produto.

```
1   object Dispenser {
2       fun init(){}
3       //Envia informacao á maquina para dispensar o produto desejado por ID
4       fun dispense(productId: Int){
5           SerialEmitter.send(SerialEmitter.Destination.DISPENSER,productId)
6       }
7
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

G. Código *Kotlin* da classe *TUI*

A classe *TUI (Text User Interface)*, é uma classe com funções genéricas que serão usadas em contextos diferentes. A classe *TUI* utiliza funções das classes *LCD*, *HAL* e *KBD* de modo a realizar as suas próprias funções fazendo assim com que a *App* só utilize o *TUI* para interagir com o utilizador. A funções do *TUI* permitem escrever um Char específico com a função *writeCostumChar*, escrever o texto alinhado com a função *writeAligned*, receber as teclas do teclado matricial(4x3) com a função *getKeyboardInput*, apagar uma linha do *LCD* com a função *clearLineLCD*, limpar todo o *LCD* com a função *clearLCD* e para apresentar a data com a função *getDate*.

```
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) }
    }
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

```
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
            )
        }
    }
}

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)
}
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

```
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

H. Código *Kotlin* da classe *FileAccess*

A classe *fileAcess* é utilizada para qualquer necessidade de aceder ao ficheiro de texto (neste caso ao *PRODUCTS.txt* ou *CoinDeposit.txt)*. Com as funções desta classe é possível ler informação dos ficheiros usando a função *readFile* e escrever no ficheiro utilizando o *writeFile*.

```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?>{
        var 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

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

I. Código *Kotlin* da classe *Products*

Na classe *Products* esta irá fazer a interação com a lista de produtos que e as suas respetivas informações que estão guardados em *Products.txt*. Utilizando a classe *fileAcess* irá ler os produtos em *PRODUCTS.txt* utilizando a função *readFileProducts* e irá escrever a informação atualizada utilizando a função *writeFileProducts*.

```kotlin
object Products{

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

    fun fetchProduct(keys: Array<Char?>, arrProducts: Array<Product?>,makePointer:Boolean): Product? {
        val tempArr = keys.map { it ?: 0 }
        val keyProduct = tempArr.reversed().joinToString (separator = "").toInt()
        return if(keyProduct<15 &&  arrProducts[keyProduct] != null){
            if(makePointer) arrProducts[keyProduct]!! else arrProducts[keyProduct]!!.copy()
        }
        else {
            Product(-1, "Not Available", 0, 0.0)
        }
    }

    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())
        var 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

J. Código *Kotlin* da classe *CoinDeposit*

Na classe CoinDeposit esta irá fazer a interação com o ficheiro de texto de modo a escrever o número de moedas no CoinDeposit.txt é usada a função *depositCoinsOnFile.*

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

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

K. Código *Kotlin* da classe *CoinAcceptor*

A classe *CoinAcceptor* é responsável pela interface do moedeiro apresentada ao utilizador. Esta utiliza funções da classe *HAL* nas suas funções de modo a realizar as suas próprias funções. Esta classe apresenta funções que recebem as moedas e guarda-as no ficheiro com as funções *hasCoin* e *acceptCoin*, devolvem as moedas que forma previamente inseridas com a função *ejectCoins* e recolhe as moedas que foram inseridas na compra de um produto com a função *collectCoins.*

```
1   object CoinAcceptor { // Implementa a interface com o moed
2       private const val COIN_INPUT = 0x20
3       private const val COIN_ACCEPT = 0x10
4       // Inicia a classe
5       fun init() {
6           throw Exception("Not Implemented")
7       }
8       // Retorna true se foi introduzida uma nova moeda.
9       fun hasCoin(): Boolean {
10          return HAL.isBit(COIN_INPUT)
11      }
12      // Informa o moedeiro que a moeda foi contabilizada.
13      fun acceptCoin() {
14          HAL.setBits(COIN_ACCEPT)
15          HAL.clrBits(COIN_ACCEPT)
16      }
17      // Devolve as moedas que estão no moedeiro.
18      fun ejectCoins() {
19          throw Exception("Not Implemented")
20      }
21      // Recolhe as moedas que estão no moedeiro.
22      fun collectCoins() {
23          throw Exception("Not Implemented")
24      }
25  }
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

L. Código *Kotlin* da classe *M*

A classe *M* tem como função iniciar a classe de manutenção. Esta utiliza as funções da classe *HAL* de modo a implementar as suas próprias funções. Na classe *M* esta apresenta uma única função de modo a ativar a manutenção da *vending-machine*.

```
1   object M {
2       private const val M = 0x10
3
4       fun isM(): Boolean{
5          return HAL.isBit(M)
6       }
7   }
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

M. Código *Kotlin* da classe *VendingMachine - App*

A classe *VendingMachine-App* é responsável pela apresentação do menu e das interações com o utilizador desde a venda de um produto até à utilização do modo de manutenção da máquina. Esta classe utiliza funções de outras classes sendo estas a classe *M, TUI, CoinAcceptor, Dispenser, Products, HAL e KBD* de modo a criar as suas próprias funções. Estas classe apresenta as funções *init* que inicia as classes necessárias para utilizar esta classe, a função *App* que apresenta o ecrã inicial e inicia o funcionamento da *App* e a função *maintence* que serve para apresentar o modo manutenção e permitir o seu funcionamento.

```
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)
}
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

```
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)
    }
}

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()
}
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

```
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())
        }
    }
}

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()
    }
}
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

```
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
}

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()
    }
}
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

```
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)
}
```

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)
}
```

Autores:
Bernardo Serra 47539
Pedro Raposo 48316
Rafael Costa 48315

```
    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
            }
        }
    }
}
```