# Descrição de classes

A organização de classes neste trabalho tem a seguinte estrutura base:

Software

Hardware

Kit

OutputManager

InputManager

NotUSBPort

Emitter

LCD

UserInterfaceeeee

Dispenser

Stock

Keyboard

Main

As setas no esquema anterior não representam herança de classes, mas sim interações realizadas entre as classes, em termos de obter informação e expor informação.

As setas intervaladas representam interações menores com classes, que existem mas não são de objectivo principal da classe, normalmente existem para garantir protocolos com o hardware.

Nesta descrição de classes começaremos por “baixo”, a classe NotUSBPort.

## NotUSBPort

Esta classe existe com o objectivo de “esconder” a particularidade do Kit de negar toda a informação que passa por ele, input ou output. Isto é feito utilizando dois métodos:

*public static int in();*

*public static void out(int argumento);*

esta classe usa a biblioteca fornecida no inicio do semester, que contém também métodos com esses nomes, esta classe envolve os métodos dessa biblioteca com estes que tratam da negação do input e do output, de forma a ficar com os dados recebidos, e enviar os dados desejados.

## InputManager e OutputManager

Durante as fases iniciais da realização do projecto deparamo-nos com várias acções que tinhamos de realizar sobre o output e o input do Kit, acções como escrever um nibble no output numa dada máscara ou let apenas um bit de um byte inteiro.

Para isso criamos estas duas classes, InputManager e OutputManager.

O OutputManager consiste de 4 métodos principais:

*public static void toggleMask(int mask);*

*public static void clearMask(int mask);*

*public static void setMask(int mask);*

*public static void toggleMask(int mask);*

*public static void setBits(int mask, int val, int pos);*

os métodos de clear, set e toggle, tal como os nomes indicam dada uma mascara de 8 bits fazem clear,set ou toggle dos bits a 1 dessa mascara, no outputport. O método setBits, recebe um valor que vai ocupar os bits a 1 dados pela mascara.

O InputManager tem dois métodos principais:

*public static int getBit(int mask);*

*public static int getBits(int keyInput, int keyInputPos);*

estes métodos oferecem a possibilidade de obter ou um bit do input dada a mascara, ou obter vários bits .

## Emitter

Esta classe foi criada para como suporte a classe LCD, a classe LCD faz interação com o hardware de forma a enviar comandos para um Display, o Emitter serve para enviar os dados garantindo os protocolos predefinidos. Desta forma a classe LCD não tem nada mas lógica de envio de comandos e inicialização do display.

A classe emitter contém o seguinte método principal:

*public static boolean sendTrama(int bits, int size);*

este método envia uma trama para o hardware seguindo todos os protocolos definidos no enunciado, com o numero de bits indicados em size.

Esta classe contém uma série de métodos privados, criados com o objectivo de simplificar a leitura do método sendTrama.

## LCD

Esta classe tem como objectivo interagir com um display hardware, existem duas possiveis acções enviar um caracter ou enviar um comando para o display.

Para garantir essas duas funcionalidades existem os seguintes métodos publicos:

*public static void writeString(String s);*

*public static void writeChar(char c);*

*public static void writeCommand(int cmd);*

*public static void init();*

o método *init* emite uma série de comandos de inicialização, este método é chamado inicialmente antes de qualquer um dos outros.

De forma a comunicar com o device hardware o método privado mais importante é o seguinte:

*private static void writeNibble(int nibble, boolean rs);*

este método tem duas formas de funcionamento, comunicação em paralelo ou em série, na fase final do projecto a comunicação é sempre feita em série, isto é feito utilizando a classe Emitter. Basicamente, writeNibble com comunicação em série cria a trama de bits a ser enviada e manda para o emitter que faz o protocolo de comunicação e envia os dados criados no writeNibble.

## Keyboard

Esta classe fornece a interacção com o device em hardware de um teclado 4x4. Desta forma o método mais importante é o método:

*public static char getKey(int timeout);*

este método publico serve para expor a funcionalidade de obtenção de uma tecla que foi carregada, dado um timeout, que se não for possivel obter uma tecla dentro desse tempo o método retorna sem tecla.

A lógica de comunicação com o teclado é feita dentro de um método privado de mesmo nome, *getKey*, que também, como o LCD *writeNibble*, tem comunicação em série ou em paralelo mas na fase final é em série. Este método realiza o protocolo de comunicação com o teclado definido no enunciado, e após obter o valor inteiro indentificador duma tecla vai associá-lo à tecla correspondente e retornar essa tecla.

Nota: Esta classe também contém um método de inicialização de forma a colocar os “tuneis” de comunicação num estado conhecido.

## Stock

Esta classe existe como modelo do stock existente na máquina, este stock é carregado no inicio da aplicação através de um ficheiro de texto, e exportado para um ficheiro de texto quando a aplicação termina. Nesta classe também existem métodos, e estruturas de dados que suportam a realização das operações indicadas no enunciado de dispensa de produtos, carregamento de produtos e anulação de produtos. Para responder a todas as necessidades existem os seguintes métodos:

*public static boolean importFrom(String filePath);*

*public static boolean exportTo(String filePath);*

*public static void restock(int prodCod,int quant);*

*public static void remove(int prodCod);*

*public static void cancelProduct(int prodCod);*

*public static Product getProduct(int prodCod);*

## Product

Esta classe representa um produto existente na máquina, com o código, nome e quantidade. Instâncias desta classe são criadas quando é importado o stock de um ficheiro de texto, e são obtidas pela aplicação principal através do método

*public static Product getProduct(int prodCod);*

em Stock.

## UserInterface & Main

A classe UserInterface serve como ponto de interacção entre as classes faladas anteriormente, e a aplicação principal. As principais funcionalidades desta classe são obtenção de um código de produto, obtenção de uma quantidade digitada por um utilizador e obtenção de uma tecla função digitada pelo utilizador.

Estas 3 funcionalidades existem para responder aos requisitos postos pelo enunciado, em que :

* Em funcionamento normal de venda, a máquina a qualquer altura recebe um pedido de um produto que começa por ser digitado o numero do codigo do produto, seguido da tecla ok. Além de numeros também podém ser usadas as teclas de setas para cima ou baixo para percorrer a lista de produtos na máquina. Para isto usamos o método:  *public static int tryGetProductCode(boolean mode);*

Este método, recebe as teclas do teclado que são pressionadas pelo utilizador e até receber a tecla OK, vai criar o codigo de produto que o utilizador está a inserir, isto significa que se o utilizador carregar 1-2 o código é 12, se seguidamente carregar numa seta para baixo o numero seria 11. Este método contém um timeout, no caso de existir inactividade por parte de um utilizador durante 5 segundos o método retorna -1;

* Em funcionamento de manutenção o que é realizado são funções, é obtida a tecla de função usando o método: *public static char tryGetFunctionKey();*

este método, também com timeout a avaliar a inactividade, obtém a tecla de função. As funções são as seguintes:

* + Carregamento, em que é ao obter a tecla de função C e o produto que quer carregar irá pedir a quantidade através do método:

*public static int tryGetQuantity();*

Este método recebe, tal como no codigo de produto, um numero digitado pelo utilizador a diferença é que as teclas de setas não tem qualquer funcionalidade;

* + Anulação, em que após obter a tecla A e o código de produto irá anular um produto em stock;
  + Desligar, em que após receber a tecla D irá desligar a aplicação.

Na aplicação principal tendo já suporte em diversas classes e métodos, foi realizada apenas uma separação dependendo do estado da aplicação, visto que existe modo de manutenção e modo de dispensa. Em modo de manutenção, é obtida a função a realizar e dependendo dessa função obter a informação necessária a concluir a operação.

Além de suporte para realização de operações sobre o stock e o dispensador de produtos através da UserInterface, também é realizado um *import* no inicio da aplicação para fazer *load* dos produtos existentes num ficheiro de texto, e no fim da aplicação é realizado um *export* para esse ficheiro de texto do estado currente do dispensador na altura em que foi desligado.