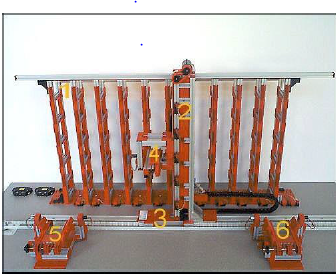
**logoFCT.png**



**Supervisão Inteligente**

**Projeto e Implementação do módulo de Despacho de um Supervisor Inteligente**

**1ª Parte**



Realizado por:

Francisco Costa nº 32189

Gonçalo Oliveira nº 32522

João Ralo nº32146

Índice

[Introdução 3](#_Toc400310639)

[Classes 4](#_Toc400310640)

1. [Hardware.java 4](#_Toc400310641)
2. [Action.java 5](#_Toc400310642)
3. [Sensores.java 8](#_Toc400310643)
4. [Interface.java 9](#_Toc400310644)

[Conclusão 10](#_Toc400310645)

# Introdução

Este trabalho é composto apenas pelas funções mais básicas do armazém inteligente, e.g. movimento manual e ler sensores. Implementámos as funções a partir de uma Dll que possibilita o acesso aos portos da máquina. Definimos os movimentos possíveis pela máquina (que serão demonstrados mais a frente), testamos no simulador e constatamos que funcionam como é pedido.

# Classes

## Hardware.java

package hardware;

public class Hardware {

static {

System.load("C:\\Hardware\\x64\\Debug\\Hardware.dll");

}

native public void create\_di(int port);

native public void create\_do(int port);

native public void write\_port(int port, int value);

native public int read\_port(int port);

}

Esta classe serve para definir as funções de escrita e leitura do hardware a partir da Dll criada anteriormente.

## Action.java

package hardware;

public class Action{

Hardware h = new Hardware();

public void move(String bit){

switch (bit) {

case "up":

h.write\_port(4, setBit(Sensores.p4,5, 1) );

break;

case "down":

h.write\_port(4, setBit(Sensores.p4,6, 1) );

break;

case "right":

h.write\_port(4, setBit(Sensores.p4,0, 1) );

break;

case "left":

h.write\_port(4, setBit(Sensores.p4,1, 1) );

break;

case "in":

h.write\_port(4, setBit(Sensores.p4,4, 1) );

break;

case "out":

h.write\_port(4, setBit(Sensores.p4,3, 1) );

break;

default:System.out.println("don’t MOVE");

break;

}

}

public void stop(String bit){

switch (bit) {

case "up":

h.write\_port(4, setBit(Sensores.p4,5, 0) );

break;

case "down":

h.write\_port(4, setBit(Sensores.p4,6, 0) );

break;

case "right":

h.write\_port(4, setBit(Sensores.p4,0, 0) );

break;

case "left":

h.write\_port(4, setBit(Sensores.p4,1, 0) );

break;

case "in":

h.write\_port(4, setBit(Sensores.p4,4, 0) );

break;

case "out":

h.write\_port(4, setBit(Sensores.p4,3, 0) );

break;

case "e": //paragem de emergencia

h.write\_port(4, 0);

h.write\_port(5, 0);

break;

default:System.out.println("don’t STOP");

break;

}

}

public void station(String stat){

switch (stat){

case "left in":

h.write\_port(4, setBit(h.read\_port(4),7, 1) );

break;

case "left out":

h.write\_port(5, setBit(h.read\_port(5),0, 1) );

break;

case "right in":

h.write\_port(5, setBit(h.read\_port(5),2, 1) );

break;

case "right out":

h.write\_port(5, setBit(h.read\_port(5),1, 1) );

break;

default: System.out.println("don’t I/O");

}

}

//AuxFunctions----------------

boolean getBit(int var, int nBit) {

int mask = 1 << nBit;

int result = var & mask;

if (result == 0) {

return (true);

} else {

return (false);

}

}

public int setBit(int var, int nBit, int value) {

int mask\_0 = value << nBit;

if (value == 0){

var = var & mask\_1;

} else {

var = var | mask\_0;

}

return var;

}

}

Esta classe define as funções básicas de movimento do elevador de carga do armazém e das estações que interagem diretamente com o hardware. Assim como uma função de setBit, escreve em bits específicos de um dado porto, e uma função getBit, lê um bit específico de um porto e retorna true ou false consoante o valor do bit.

## Sensores.java

package hardware;

public class Sensores implements Runnable{

Hardware h = new Hardware();

public static int p0;

public static int p1;

public static int p2;

public static int p3;

public static int p4;

public static int p5;

public void run(){

while(true){

try{

p0 = h.read\_port(0);

p1 = h.read\_port(1);

p2 = h.read\_port(2);

p3 = h.read\_port(3);

p4 = h.read\_port(4);

p5 = h.read\_port(5);

Thread.sleep(50);

}catch(Exception e){}

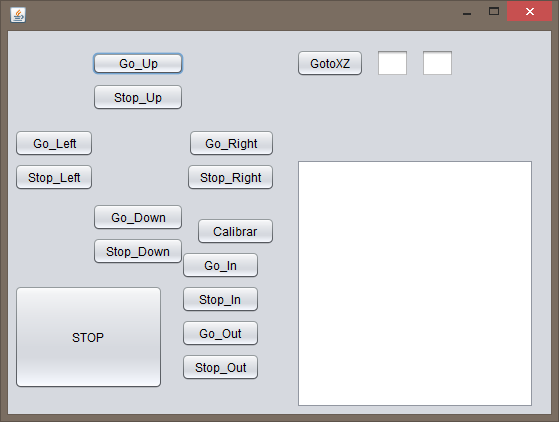
}

}

}

Esta classe lê todos os portos da máquina guarda-os em variáveis para poderem ser chamados por outras classes, criando assim um buffer de memória no sistema para que não existam várias classes a aceder ao hardware a todo o momento.

## Interface.java



private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

a.move("up");

jTextArea1.setText("going up");

}

Para facilitar a utilização do simulador foi elaborada a interface disposta em cima. Nesta fase o programa só executa comandos manuais. O código que está aqui descrito é um exemplo de o código de um dos botões da interface. Os restantes são semelhantes.

# Conclusão

Neste trabalho só a class Action e Sensores são as versões finais. As outras classes (algumas não estão neste relatório pois não são importantes para esta fase do trabalho) serão sujeitas a alterações ao longo do trabalho.´

Foi criada uma boa base para construir as novas classes que serão precisas para a parte de supervisão inteligente propriamente dita nas semanas vindouras.