NOVA SCHOOL OF SCIENCE & TECHNOLOGY

Departamento de Engenharia Electrotécnica e de Computadores

Secção de Robótica e Manufactura Integrada

Description of Lab Work nº 2	
Course	**Real-Time systems / *Sistemas de Tempo Real***
Year	2024/2025
Aim	Development of real-time applications using high level languages
Classes	4 classes x 3 hours + 12 extra hours (outside)
Delivery Date	**27/11/2024**

Concrete Objectives:

1. Java language
    a. Characteristics of this programming language
    b. The “Java Virtual Machine”(JVM)
    c. Object-oriented programming
    d. Java training examples

2. Real-time features of Java
    a. The Thread class and the “Runnable” interface
    b. Synchronization and communication between Threads.
    c. Interaction with physical systems; The “Java Native Interface” (JNI)

3. Develop a practical application of Java real-time as described in Annex 1

Annex 1 – Description of Work

This work is aimed at providing the concepts of the Java concurrency language features. For such, the development a of real-time solution, for the problem described below, is proposed. The scenario of this problem is related with a real-time control of a manufacturing assembly line that must split boxes containing furniture for different vendors (Iquêá, ConforRama, LasKaças, Homae, etc) as presented in Figure 1.

Figure 1– Example of Furniture

(From: https://www.deviantart.com/coolarts223/art/Furniture-with-eyes-Weird-scene-991655658)

The adequate solution for this problem requires the utilization of a set of developing technologies based on Java SDK, Java Native Interface (JNI), and Dynamic-link library (DLL) development, with more emphasis on the Java concurrency features.

For the work development, in the context of classes, it will be used an educational kit simulating a manufacturing distribution conveyor as shown in Figure 2. The connection to the kit is done through the NI USB 6509 Data Acquisition board (DAQ), already used in Labwork 1.

A simulator of the distribution conveyor kit (Figure 2) is also available, which allows continuing the work outside of the lab. This simulator allows testing before running the implementation in the real kit, which helps eliminate most of the mistakes that might occur during development.

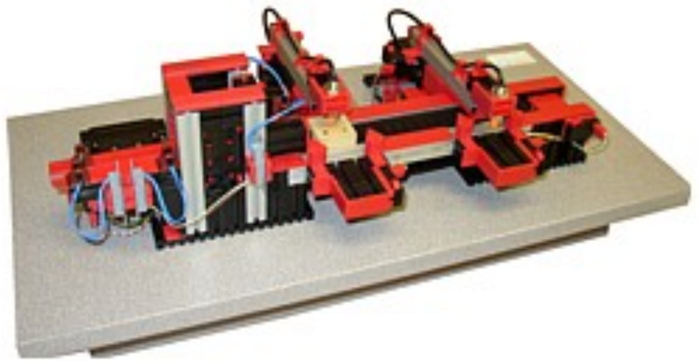

Figure 2– Kit which models the manufacturing assembly line

The mapping of the sensors and actuators of the distribution conveyor into the DAQ is detailed in Figure 3.

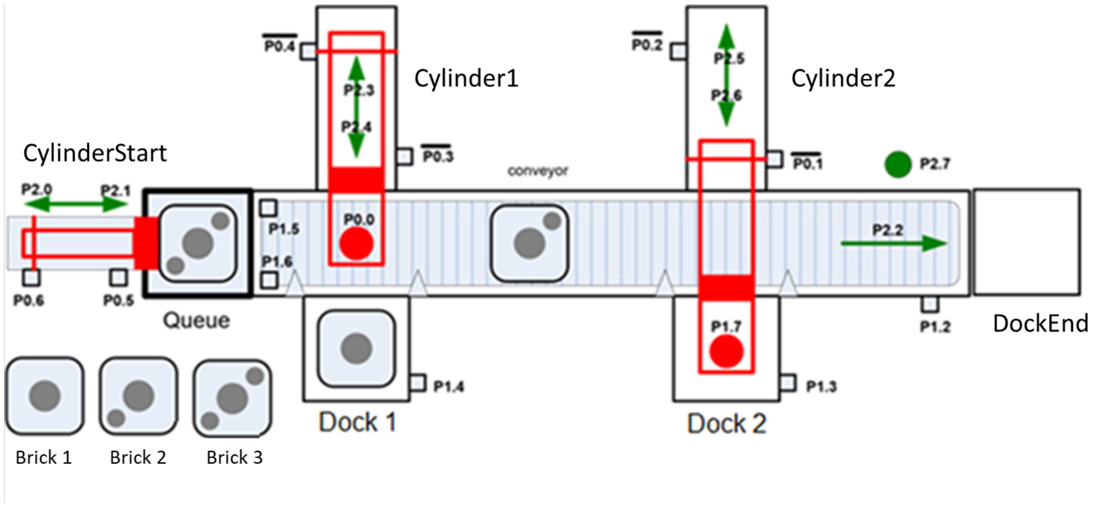

Figure 3 – Sensors and actuators addresses of the distribution line

The intralogistics line has got three distributions docks: Dock 1, Dock 2 and Dock End, which should work in the following way:

- By pressing **<u>“S”</u>** the system **<u>Starts</u>** its working cycle, and by pressing **<u>“F”</u>**, the system **<u>Finishes</u>** its work. (The interaction of the system can be done via console application or through a GUI where the integration of buttons “Start” and “Finish” can be used for this purpose. Please note that the development of a GUI is completely optional and is mostly the responsibility of the student)
- **<u>Dock 1</u>** receives boxes from Iquêá vendor: “Box1”. After delivering a box in this Dock, the led at P2.7 must flash (On/off) for 1 second.
- **<u>Dock 2</u>** receives boxes from ConforRama vendor: “Box2”. After delivering a box in this Dock, the led at P2.7 must flash (On/off) for 2 seconds.
- **<u>Dock End</u>** receives all the boxes that are from other vendors “Box3” and all the rejected by dock 1 and dock 2. Whenever this happens (rejected by dock1 and dock2), the mentioned led must frantically flash many times for 3 seconds.

- The entry of a new box in the conveyor is signaled by pressing character "P" on the keyboard (or a corresponding button in a Java UI).
- If a human operator at **dock 1** presses the corresponding button P1.4, the dock is closed and the boxes from Iquêá must be rejected and sent to DockEnd. When the operator presses again the button, the station becomes open again.
- If a human operator at **dock 2** presses the corresponding button P1.3, the dock is closed and the boxes from ConforRama must be rejected and sent to DockEnd. When the operator presses again the button, the station becomes open again.
- If a human operator at **dock End** presses the corresponding button P1.2, then it means that dock 1 and dock 2 are closed, **for 10 seconds**, and the corresponding boxes that should follow be delivered to that docks will proceed to dock end (Dock1 and Dock2 are closed so all boxes must go to DockEnd). In this case, the LED should flash for at least 3000 milliseconds to ensure that the operators are aware of the situation. After the 10 seconds, the system must automatically open both docks (Dock1 and Dock2).
- It is also necessary to provide an emergency stop and resume operation.
- During the emergency stop state, the LED should frantically flash with, e.g. times Ton =400ms and Toff=400ms (but these values can be adjusted).
- The system must remember the total number of boxes of each type that got into each dock, as well as the number of boxes rejected in Dock 1 and Dock 2.
- Information regarding the state of the system, including statistics, must be shown on the console screen whenever requested (console mode, JAVA Swing, or JavaFX, remember that the user interface is mostly the responsibility of the student).

In terms of operationality, each box is taken from the entry queue, using cylinder start, and placed into the conveyor. There are two sensors at the beginning of the conveyor, which can identify the package type. For instance, a box of type 3 contains two magnetic marks at their corners, in a diagonal way (see figure 3). Whenever these marks pass under the mentioned sensors, the corresponding box can be identified.

However, there is a technological constraint on identifying each box, because when cylinder start completes its onward movement to deliver a box, the incoming box is already half inside the conveyor. Therefore, using a non-concurrent approach, in which the box is sequentially delivered and then identified, will fail because one of the marks already passed under the sensors, leading to misidentifying the box. As such, it is recommended that the mentioned identification routine finishes identifying only after each box arrives to the sensor wired to P0.0 (in Cylinder 1 at dock 1), in which the central mark is identified. Furthermore, a box of type a only contains the center mark and cannot be spotted by the identification sensors.

Finally, keep in mind that the system has active high and active low sensors. Active low sensors provide logical value false/0 when active, and true/1 otherwise. An example of active low sensor is in P0.4, shown in figure 3. These are usually identified with a top bar. An example of an active high sensor is P0.6 (no top bar).

To develop a solution for adequately coordinating the operation of the distribution conveyor, it is necessary to fulfill the following functional requirements, which are divided into low-level and high-level requirements:

Table 1 – Needed requirements

Req.	Description
	Low-level requirements (DLL development)
FR_L1	Move and stop each **cylinder**
FR_L2	Move and stop moving the **conveyor**

FR_L3	Read **Box identification** and position sensors
FR_L4	Detect the **button/switches** states
FR_L5	Turn **LED** on and off
FR_L6	Keyboard or UI interaction
High-level requirements (Java development)	
FR_H1	Using the keyboard, develop robust (semi-automatic) **calibration**. Using keys, the user defines the direction of the movements. During a movement, the actuator must stop when it reaches a position sensor (Cylinders)
FR_H2	Develop the **getBox**() functionality, which gets a box from the entry queue and puts it in the conveyor
FR_H3	By pressing "S" the system Starts its working cycle. Pressing "F", the system Finishes its work. In a GUI, the Buttons "Start" and "Finish" can be used for this purpose
FR_H4	Develop **CylinderX.gotoPosition**(int position): in which X is related to the corresponding cylinder; position 0 means resting position; 1 means working position (pushing a package)
FR_H5	Perform **identifyBoxType**, that identifies each box with the sensors at the beginning of the conveyor (sensors P1.5 e P1.6); identification ends when box reaches sensor P0.0 (in cyilinder1 at dock 1)
FR_H6	Delivering **Box1** for **dock 1**
FR_H7	Delivering **Box2** for **dock 2**
FR_H8	**Delivery Box3 and all the rejected boxes (from dock1 and dock2)** to D**ockEnd**
FR_H9	Detection of buttons P1.4 and P1.3 pressed by human operator, at dock 1 and dock 2 to close and open the Docks
FR_H10	Perform the **temporarily close** of dock 1 and dock 2 according to the detection of button P1.2 pressed with corresponding LED signalization
FR_H11	Handle Emergency **STOP/RESUME** operation with corresponding LED signalization
Information retrieval requirements	
FR_H12	**Collect statistics** of the total number of boxes that were delivered at each dock, and

	the rejections. Show these statistics on the console screen whenever requested. If a GUI is used, then these numbers must be updated in the GUI in **real-time**.
Non-Functional requirements:	
NF1	While operating, the system **can accumulate requests.**
NF2	While operating, the system can **provide information** (e.g., number and type of boxes delivered at dock 1)
NF3	The system **works appropriately** (e.g. (i) when placing a new box on the conveyor, it must be moving; (ii) delivers corresponding boxes to docks according to sequence)
NF4	The system always moves within **limits** (cylinders)

<u>Classes plan for the Lab Work 2</u>

Class 1:
SW Installation & Interaction with "physical" system. Low-level functionalities implementation.

Class 2:
JNI development for interacting with the physical system from Java. Elaboration of both low-level and high-level routines for LabWork2.

Class 3:
Introduction to Threads and synchronization and communication mechanisms. Development of the necessary threads for modeling LabWork2 concurrent behavior.

Class 4:
Fulfillment of each functional requirement towards the solution for LabWork2.

Professors:

– Ana Inês Oliveira, aio@fct.unl.pt	(laboratory classes)
– André Rocha, ad.rocha@fct.unl.pt	(theoretical and laboratory classes)
– Filipa Ferrada, fam@fct.unl.pt	(theoretical classes)
– Ricardo Peres, ra.peres@fct.unl.pt	(laboratory classes)
– João Rosas, jrosas@uninova.pt	(laboratory classes)

Lab Work 2 Delivery:

Students must submit via Moodle **(and only via Moodle)**:

- ➔ Upload a zip/rar archive with your Visual Studio Project + IntelliJ IDEA Project.
- ➔ Upload a demo video (aprox. 5/8 minutes) presenting all implemented requirements/functionalities. The video **must be** accompanied by a narration of what is being presented.

Evaluation criteria

It is provided (as a guiding object) a table with the project evaluation criteria. Each item will be quoted according to the approach and quality of the generated code.

Items		**Quotation/Penalty**
Functionality	FR_L1 – Cylinders	0.25
	FR_L2 – Conveyor	0.25
	FR_L3 – Boxes Identification	0.25
	FR_L4 – Switches	0.25
	FR_L5 – LED	0.25
	FR_L6 – UI interaction	0.50
	FR_H1 –calibration	0.25
	FR_H2 –getBox	0.50
	FR_H3 –Start / finish system	0.50
	FR_H4 – cylinder gotoPosition	0.50
	FR_H5 –identifyBoxType	1.00
	FR_H6 –delivery of box1 for dock 1	1.00

	FR_H7 – delivery of box2 for dock 2	1.00
	FR_H8 – delivery of box3 for dock end	1.00
	FR_H9 –delivery of rejected boxes from dock 1 and dock 2	1.50
	FR_H10 –temporarily close dock 1 and dock 2	2.00
	FR_H11 –STOP/RESUME	1.50
	FR_H12 –collect statistics	1.50
	NF1 – While operating, the system can accumulate requests	1.50
	NF2 – While operating, the system can provide information	1.50
	NF3 – The system works appropriately	0.50
	NF4 – The system always moves within limits	0.50
Video		2.00
	TOTAL	**20.00**
No delivery of the VS project or Netbeans project or video (* all MANDATORY)		**-20**