P.PORTO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

Relatório

Grupo 20

Realizado por:

Hugo Silva – 8180378
Nicolas Sousa - 8180658
Rómulo Leite – 8200593

Índice

Índice de ilustrações

1.Manual de compilação

O projeto foi desenvolvido recorrendo ao JDK 13, logo deve existir uma versão no computador que permita executar essa versão do JDK.

Para correr o programa na linha de comandos é necessário:

1- Abrir a linha de comandos na raiz do projeto, onde este tem as seguintes pastas: *.idea, Documentação, imgs, out e src*.

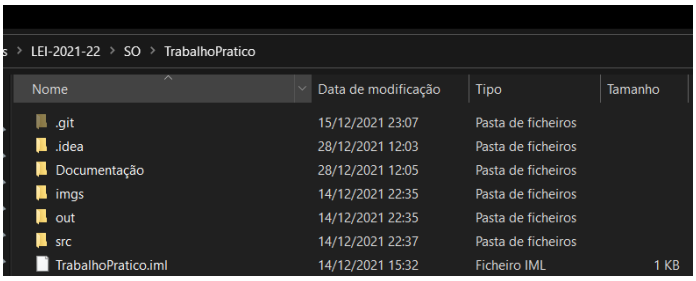

FIGURA 1 - LOCALIZAÇÃO DAS PASTAS DO PROJETO

2- Executar comando "cd src";

3 *Windows*- Executar comando "*dir /s /B *.java > compiles.txt*"**;**

```
C:\Users\hugod\Documents\LEI-2021-22\SO\TrabalhoPratico>cd src

C:\Users\hugod\Documents\LEI-2021-22\SO\TrabalhoPratico\src>dir /s /B *.java > compiles.txt

C:\Users\hugod\Documents\LEI-2021-22\SO\TrabalhoPratico\src>javac @compiles.txt
```

FIGURA 2 - COMANDO DE INICIAÇÃO DO PROJETO

3 *Linux*- Executar comando "find -name "*.java" > *compiles*.txt**";**

4- Executar comando *"javac @compiles.txt"**;**

5- Executar comando “java LavagemdeCarros/AppMain.java”;

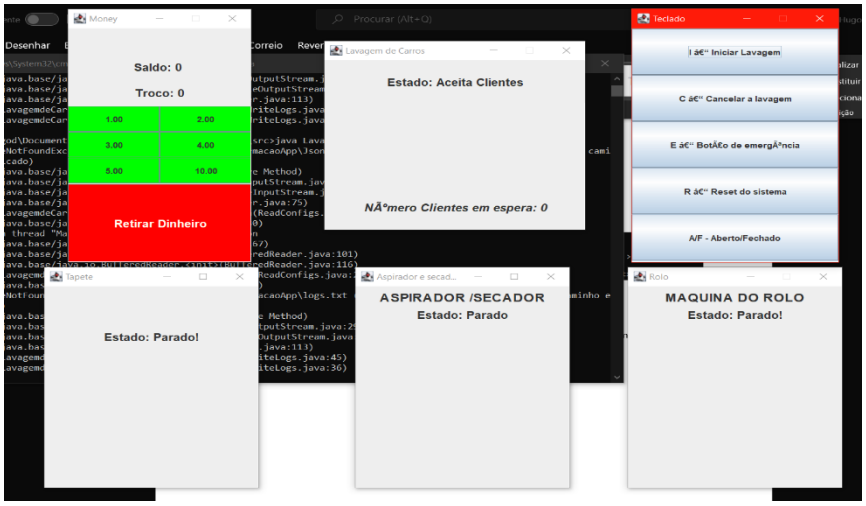

FIGURA 3 - PROGRAMA INCIALIZADO

Nota: No **passo 5** pode, dependendo do computador, ser executado o comando “java LavagemdeCarros/AppMain”.

1.1 Configuração

A configuração do projeto está relacionada com o ficheiro *JsonFile*. Nele podem ser alterados os valores dos tempos de execução do tapete/rolo etc.

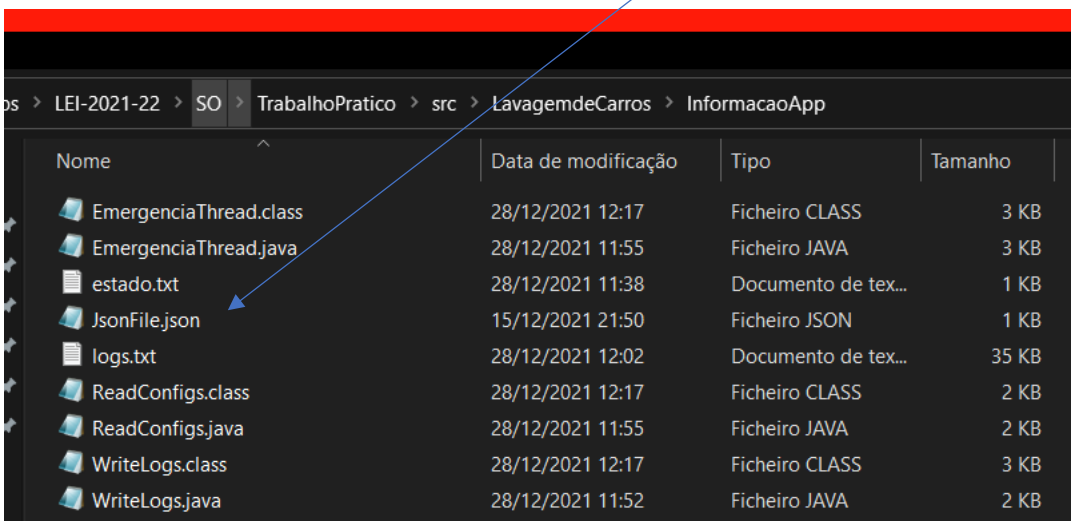

FIGURA 4 - LOCALIZAÇÃO DO FICHEIRO JSON

2.Descrição das funcionalidades implementadas

2.1 Logs

Foi criada uma classe logs para acompanhar/registar todas as atividades ao qual a lavagem de carros exercerá, para além disso apresenta a data e anuncia o fim da tarefa ao qual estava em execução, nas respetivas classes é colocado uma mensagem que fornece o registo no ficheiro logs sempre que essa classe é utilizada. Foi criada uma thread para este processo (*WriteLogs*) que é gerida por um semáforo.

```
@Override
public void run() {
    super.run();

    while (Main.running) {
        try {
            semDisplay.acquire();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        try {
            log(mensagem.getMensagem());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

FIGURA 5 – THREAD DE LOGS APRESENTADO NO FICHEIRO LOGS

2.2 Importação de configurações de um ficheiro

Para esta funcionalidade foi criado um ficheiro json, esse ficheiro json vai fornecer todos os dados necessários para execução do programa. Antes de ser iniciada a lavagem é feita a leitura de dados e só depois iniciada a thread da *main*.

Existe um recurso partilhado entre as threads que é um objeto com os dados lidos do ficheiro chamado classe *ValueConfigs*.

2.3 Botão Reset

Esta funcionalidade permite reiniciar a lavagem. É excutado na *Main*_auxiliar e quando clicado interrompe todas as *threads* que estão em excução e faz *reset* ao número de clientes em espera.

Para interromper é chamado o método *interrup*() e libertado os semáforos das threads para atualizar os estados.

2.4 Botão Emergência

Esta funcionalidade tem como intuito parar o funcionamento da lavagem. Quando o utilizador clica no botão de emergência é guardado o estado atual do sistema em ficheiro. Assim quando o utilizador clica novamente no botão de emergência o programa volta para o ponto de partida onde estava anteriormente.

Todo esse processo é feito a partir da *Main*_auxiliar, e existem um objeto partilhado entre as *threads* que guarda a informação do sistema (classe Emergencia).

```
public void writeEstado(Emergencia emergencia) throws IOException {
    File file = new File( pathname: "src/LavagemdeCarros/InformacaoApp/estado.txt");
    FileOutputStream f = new FileOutputStream(file);
    FileWriter fr = new FileWriter(file, append: true);
    ObjectOutputStream o = new ObjectOutputStream(f);
    // Write objects to file
    o.writeObject(emergencia);
    o.close();
    fr.close();
}

public Emergencia readEstado() throws IOException {
    File file = new File( pathname: "src/LavagemdeCarros/InformacaoApp/estado.txt");
    FileInputStream fi = new FileInputStream(file);
    ObjectInputStream oi = new ObjectInputStream(fi);

    // Read objects
    Emergencia pr1 = null;
    try {
        pr1 = (Emergencia) oi.readObject();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
```

FIGURA 6 - CLASSE BOTÃO DE EMERGÊNCIA

2.5 Número de clientes em espera

Esta classe permite criar gerir o número de clientes em espera na lavagem.

O número de clientes em espera(classe ClientesEmEspera) é um objeto partilhado entre a *Main* e a *Main* auxiliar e permite incrementar e decrementar o número de clientes de forma sincronizada. Como é um recurso partilhado entre threads o método possui o synchronized. Assim é garantido um acesso correto ao objeto.

```
public class ClientesEmEspera {
    private int count;

    public ClientesEmEspera() { this.count = 0; }

    public synchronized void addClient() { this.count += 1; }

    public synchronized void as() { this.count = count - 1; }

    public synchronized int getNumberClients() { return count; }

    public synchronized void setCount(int count) { this.count = count; }

    public synchronized void reset() { this.count=0; }
```

FIGURA 7 - CLASSE DOS CLIENTES EM ESPERA

2.6 Como é feito o controlo do valor do moedeiro

O moedeiro possui um objeto partilhado entre a *Main* e a *Main* auxiliar (classe ValoresMoedeiro). A *Main* auxiliar é responsável por fazer o controlo dos valores sendo o cliente apenas inicia a lavagem se tiver um valor suficiente.

2.7 Como é controlado os clientes em espera?

De forma a existir melhor aproveitamento dos recursos e a possibilidade de um programa eficaz, este recurso é utilizado pela thread main e a *Main_*auxiliar, ao qual a a *Main_* auxiliar faz a atribuição de um novo cliente em fila de espera e a Main a decrementação do número de clientes.

Compete ao *Main_*auxiliar verificar se no moedeiro está inserido o valor mínimo de lavagem, em caso de valo suficiente o cliente é adicionado, caso contrário aparece uma notificação de falta de saldo.

A Main é responsável também pelo estado da lavagem, ou seja, verificar se não existem clientes em espera e colocar a lavagem como disponível.

2.8 Estado do aspirador, Rolos de Lavagem e Secador

Os 3 processos de lavagem passam por dois estados ativo ou parado, esses processos vão sendo alternados consoante a necessidade uso, ou seja, o tapete envia a informação para a main do seu movimento e consoante a posição do “veiculo” ele inicia o processo necessário.

No caso de ser ativado o botão de emergência as *thread* são todas colocadas em modo parado até ordem contrária.

Se for o botão de reset eles voltam todos ao estado parado.

Cada *thread* possui um semáforo e os estados de cada uma são controlados pelo *Main* que dá ordem de início de execução e é informada do fim.

2.9 Main Auxiliar

Esta Main foi criada com o principal intuito de evitar congestionamento do Main Principal, com isso possibilita que os recursos sejam partilhados com melhor eficácia, como tal isso impossibilita que possa existir algum processo ao qual não seja realizado:

Exemplo: Supondo que entra um novo cliente e a *main* principal está a aguardar a thread rolo termine a sua execução. O cliente só era adicionado quando o processo do rolo terminasse. Dado isto foi criada a *Main_*auxiliar que serve de "apoio" à Main principal nas tarefas de gestão de moedeiro, gestão de clientes, reset e botão de emergência.

Main: O módulo main foi implementado, e é capaz de manter as estruturas de dados. Por exemplo, quantos clientes em espera existem, valida o valor monetário introduzido para a lavagem do carro, paragem de emergência, contabilização. Este módulo, é responsável por lançar a execução dos outros módulos e é notificado seu término.

Interface e moedeiro: O teclado permite a interação com o sistema, sendo que é utilizado um semáforo entre a UI e a thread.

```
private ActionListener my20 = new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent actionEvent) {
        moedeiro.releaseSemaforo( mod: 2);
        updateUI();
    }
};
```

FIGURA 8 – SEMÁFORO MOEDEIRO

3.Mecanismos de sincronização e comunicação entre módulos

3.1 Semáforos

Para a sincronização entre os vários componentes da aplicação são utilizados semáforos.

Foram utilizados semáforos entre o Main e o Moedeiro, Main e Rolo, Main e Aspirador etc… como é possível ver na figura 8, estes são os diversos semáforos utilizados por parte da main.

Cada thread tem um semáforo, a main controla os semáforos dos componentes da lavagem, esses semáforos são controlados de forma a verificar a atividade do processo , assim que é libertado pode passar para a seguinte tarefa , ou até ser interrompido como no caso do de emergência.

```
//Semaforos
private Semaphore semaforoMain, semMoedeiro, semTeclado,
        semTapete, semDisplay, semRolo, semLogs, semAspiradorSecador,
        semMainMoedeiro, semEmergencia;
```

FIGURA 9- TODOS OS SEMÁFOROS DO PROGRAMA UTILIZADOS NA MAIN

3.2 Sinalização entre Threads

De forma a demonstrar os diversos conhecimentos obtidos utilizamos também o mecanismo de sinalização entre as threads do modulo aspirador/secador e o main como é apresentado na figura 10 e 11.

```
        case ACTION_ASPIRADOR:
            mensagemLog.setMensagem("Carro em aspiração");
            semLogs.release();
            aspiradorSecador.releaseSemorofo( estado: 0);
            emLavagem.espera();
            action = ACTION_ROLO;
```

FIGURA 10 - SINALIZADOR DA LAVAGEM PARA O ASPIRADOR

```
case ACTION_ROLO:
    mensagemLog.setMensagem("Carro no rolo");
    semLogs.release();
    rolo.setAction(1);
    semRolo.release();
    emLavagem.espera();//Aguarda e dá autorização pra atualiza a label do rolo
    rolo.setAction(0);//termina
    semRolo.release();
    action = ACTION_SECADOR;
```

FIGURA 11 - SINALIZADOR DA LAVAGEM PARA O ROLO

4.Enumeração das funcionalidades pedidas e não implementadas

- ✓ **Módulo Main**: implementado;
- ✓ **Módulo Interface e moedeiro**: implementado;
- ✓ **Módulo Tapete**: implementado;
- ✓ **Módulo Rolos de Lavagem**: implementado;
- ✓ **Módulo Aspersores e secador**: implementado;
- ✓ **Botão de emergência**: implementado;
- ✓ **Botão de reset do sistema**: implementado;
- ✓ **Botão de aberto/fechado**: implementado;
- ✓ **Ficheiro de configuração**: implementado;
- ✓ **Logs**: implementado;