UNIVERSIDADE AUTÓNOMA DE LISBOA

Departamento de Ciências e Tecnologias

Licenciatura em Engenharia Informática

Paradigmas de Programação

Docentes:
Adrian Dediu
Valéria Pequeno

Realizado por:
30007214 Nuno Cartaxo
30007679 André Santos
30007252 André Martins

UNIVERSIDADE AUTÓNOMA DE LISBOA

Índice

Conteúdo

1. Ficheiro de configuração

Este projeto tem um ficheiro de configuração da simulação o qual nós optamos por ser preenchido da seguinte maneira:

- O ficheiro é lido linha a linha e executará um dos dois comandos seguintes:
  - "Cidade" + "Número de autocarros a começar na mesma"
  - Bus + "Quantidade" + "Tipo de autocarro"
  - Passenger + "Número de passageiros"

- Os comandos **"Cidade"** diz quantos autocarros deverão começar na mesma.
  - Os comandos "Cidade" necessitam de ser os primeiros no ficheiro de configuração.

- Os comandos **Bus** adiciona a quantidade especificada do tipo escolhido de autocarro que começa numa cidade aleatória.

- É necessário para correr uma simulação um autocarro de cada tipo e de 4 até 10 autocarros.

- O comando **Passenger** cria e adiciona o número desejado de passageiros à simulação.

- É necessário para correr a simulação que o número de passageiros seja igual ao superior à capacidade dos autocarros.

- Todos os comandos devem ser separados por espaço.
  - Comando válido: Bus 10 Convencional
  - Comando inválido: Bus10Convencional

Existem 4 tipos de autocarro:

- convencional: Capacidade(51) , Velocidade media(80km/h)
- miniBus: Capacidade(24) , Velocidade media(80km/h)
- expresso: Capacidade(69) , Velocidade media(80km/h)
  - Este só para em Lisboa, Porto e Braga.
- longDrive: Capacidade(59) , Velocidade media(60km/h)

Cidades na simulação:
Cascais <-30km-> Lisboa <-204km-> Coimbra <-122km-> Porto <-55km-> Braga

Exemplo de configuração para (4 autocarros, 1 de cada tipo disponível e 1000 passageiros):

Cascais 1
Lisboa 1
Coimbra 1
Porto 1
Braga 0
Bus 1 miniBus
Bus 1 convencional
Bus 1 expresso
Bus 1 longDrive
Passenger 1000

2. Decisões de implementação

Ao começarmos o projeto, decidimos ao ler o enunciado que a simulação deveria ser feita com algumas threads obrigatórias. Estas seriam:

- Uma thread por cada autocarro
- Uma thread para gerar anomalias nestes autocarros
- Uma thread para verificar a condição de término da simulação

O nosso ficheiro de configuração é lido antes da simulação começar, linha a linha, e executa os comandos nele presentes. Estes no final de serem inseridos serão verificados para erros de configuração: passageiros insuficientes, número de autocarros inválido, a falta de pelo menos um tipo de autocarro, e por fim, um autocarro expresso não pode começar em Cascais ou Coimbra.

Para as nossas localizações criamos uma classe que tem propriedades parecidas a doubly linked list, pois dentro da classe contem duas localizações a seguinte e a anterior, para poder navegar entre elas facilmente.

Esta simulação está a correr 1000 vezes mais rápida que em tempo real, e as distâncias entre as cidades dentro da simulação são aproximadas ao valor real tirado do Google Maps.

Temos também uma Graphical User Interface (GUI) que usa java.swing, onde o funcionário (utilizador) tem algumas opções de controlo sobre os autocarros. Usamos uma thread para atualizar o texto referente à simulação na nossa GUI a cada meio-segundo.

3. Problemas de implementação

Ao longo da realização deste projeto deparamo-nos com vários problemas, a que recorremos a diferentes soluções para estes mesmos.

Problema de remoção de passageiro (da localização) – Ao remover passageiros da localização para o autocarro, caso o autocarro fosse do tipo expresso, alguns passageiros não poderiam entrar, porque o seu destino era diferente, isto causou problemas de passageiros presos infinitamente num autocarro que não iria para o seu destino. Resolvemos este problema com uma simples condição.

Problema de remoção de passageiro (do autocarro) – Ao remover passageiros do autocarro, tínhamos um problema que só metade dos passageiros cuja paragem era a atual eram removidos, este erro era causado pela remoção do passageiro que faria a lista de passageiros ser menor, que significa que as posições podem ser diferentes, exemplo lista = 0, 1, 2 se fizermos lista.get(1) dará o valor 1, mas após remover o valor 1 a mesma posição dará o valor 2.

Problema de sincronização de threads – Se vários autocarros chegarem à mesma localização ao mesmo tempo estes podem duplicar passageiros na entrada dos passageiros no autocarro. Para resolver este problema, nós utilizamos sincronização, para que se um autocarro acedesse à variável que contem os passageiros bloquea-se esta mesma (dar lock) para que nenhum outro autocarro aceda ao mesmo tempo.

Problema de parar uma thread a partir de outra – Em várias situações, como por exemplo as anomalias, queremos pausar uma thread a partir de outra thread. Nós não conseguimos isto com o método sleep nem com o método wait, por isso tivemos de recorrer a dois métodos “deprecated” que são thread.suspend() e thread.resume(), estes métodos são propícios a problemas de deadlock.

Problema de finalizar o programa – Este problema é parecido ao anterior, mas surge da thread que verifica se a simulação acabou e tenta terminar todas as threads. Neste caso, utilizamos o método .stop() que é “deprecated”, pois não encontramos outra solução.

Problema com o GUI – Na criação da GUI tivemos vários problemas a demonstrar texto referente ao estado da simulação, e após tentar usar “JLabel”, “JTextField” e “JTextArea”, sem conseguir demonstrar texto e o elemento desaparecia do painel principal, trocamos para “JTextPane” que funcionou como nós pretendíamos.

Problema de FileNotFoundException (ao iniciar o programa no IDE InteliJ ou na CMD) – O caminho para o ficheiro de configuração dava erro ao iniciar pela CMD ou pelo InteliJ, para resolvermos isto criamos uma variável que tem o caminho absoluto para o projeto, mesmo que este seja iniciado dentro do diretório src.

4. Divisão de tarefas

O grupo realizou o código em conjunto, apenas o desenvolvimento do GUI foi realizado na sua totalidade pelo Nuno Cartaxo e as verificações do ficheiro de configuração foram realizadas pelo André Santos.

O colega André Martins juntou-se ao projeto numa data já tardia, ao qual ficou responsável pela tarefa exigida no enunciado.

5. Erro devido à interferência de threads

No desenvolvimento deste projeto foi encontrado este erro de sincronização. No método “load_passengers()” foi necessário a utilização de sincronização. Uma vez que a variável “this.location” não estava sincronizada, os autocarros acediam à “location” ao mesmo tempo, resultando numa duplicação de passageiros nesta localização. Não só apanhavam o mesmo passageiro, como também removiam exponencialmente os passageiros, seguindo o método “load_passengers()” como está implementado, fazendo com que o loop implementado no bloco de código excedesse o limite retornando um erro.

```
Porto : entrou o passageiro 13 no bus 2_miniBus
Porto : entrou o passageiro 13 no bus 1_miniBus
Porto : entrou o passageiro 21 no bus 1_miniBus
Porto : entrou o passageiro 16 no bus 2_miniBus
Porto : entrou o passageiro 23 no bus 1_miniBus
Porto : entrou o passageiro 26 no bus 2_miniBus
```

Exemplo de 2 minibus na mesma paragem sem sincronização. O passageiro 13 entrou nos dois minibus.

Neste caso temos um exemplo do método “load_passengers() “ com sincronização, em que após um minibus correr um método este dá “lock” na variável “this.location” impedindo que os outros acedessem à mesma até este dar release ao “lock”, ou seja, acabar o método.

```
Porto : entrou o passageiro 107 no bus 2_miniBus
Porto : entrou o passageiro 117 no bus 2_miniBus
Porto : entrou o passageiro 128 no bus 2_miniBus
Porto : entrou o passageiro 131 no bus 2_miniBus
Porto : entrou o passageiro 146 no bus 2_miniBus
Porto : entrou o passageiro 149 no bus 1_miniBus
Porto : entrou o passageiro 155 no bus 1_miniBus
Porto : entrou o passageiro 159 no bus 1_miniBus
Porto : entrou o passageiro 160 no bus 1_miniBus
Porto : entrou o passageiro 165 no bus 1_miniBus
Porto : entrou o passageiro 171 no bus 1_miniBus
```

Com sincronização, a lista dos passageiros recebe um lock e só o minibus que deu lock pode utilizar a variável até esse mesmo dar release.