

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

ENGENHARIA ELETRÓNICA E TELECOMUNICAÇÕES E DE COMPUTADORES

LEETC

Programação III

Trabalho 3 – Simulador de chamadas

Realizado por:

Miguel Carvalho, A45331

Pedro Aquino, A45908

20 de fevereiro de 2021



Resumo

Foi-nos proposto para o terceiro trabalho da unidade curricular de Programação III, desenvolver um simulador de chamadas onde é guardada as informações sobre a chamada dos dois telefones intervenientes.

E aqui também descrito e explicadas das estruturas de dados utilizadas tanto neste trabalho como no segundo.

Índice

Resumo	2
Índice de Ilustrações	5
1. Introdução	7
2. Implementação do gestor de agenda de contactos	8
1.1. <i>Back-end</i>	8
1.1.1. Classe <i>Contact</i>	8
1.1.2. Classe <i>Utils</i>	8
1.1.3. Classe <i>NoteBook</i>	9
1.2. <i>Front-End</i>	10
1.2.1. Classe <i>DocumentNumber</i>	10
1.1.1. Classe <i>TextPanel</i>	10
1.1.2. Classe <i>AbstractDialog</i>	10
1.1.3. Classe <i>ContactDialog</i>	10
1.1.4. Classe <i>DatePanel</i>	11
1.1.5. Classe <i>TelephonesPanel</i>	11
1.1.6. Classe <i>NoteBookFrame</i>	12
2. Implementação do simulador de telefones	13
2.1. Diagrama estático de classes UML	13
2.2. <i>Back-End</i>	13
2.2.1. Classe <i>DateTime</i>	13
2.2.2. Classe <i>Call</i>	13
2.2.3. Classe <i>Record</i>	14
2.2.4. Classe <i>Telephone</i>	14
2.3. <i>Front-End</i>	15
2.3.1. Classe <i>ContactPanel</i>	15
2.3.2. Classe <i>CallTable</i>	15
2.3.3. Classe <i>RecordPanel</i>	16

2.3.4.	Classe <i>FilterPanel</i>	18
2.3.5.	Classe <i>CallFrame</i>	22
2.3.6.	Classe <i>RingFrame</i>	23
2.3.7.	Classe <i>InitFrame</i>	23
2.3.8.	Classe <i>PhoneFrame</i>	25
Conclusão.....		28

Índice de Ilustrações

Figura 1 - Apresentação da caixa de diálogo para a criar ou editar um contacto.....	11
Figura 2 - Apresentação do painel para inserir uma data	11
Figura 3 - Apresentação do painel para inserir números de telefone	11
Figura 4 - Apresentação dos contactos da agenda.....	12
Figura 5 - Painel de seleção de um número da agenda de contactos	15
Figura 6 - Apresentação de uma CallTable	15
Figura 7 - Modo “List” do RecordPanel	16
Figura 8 - Modo “Group” do RecordPanel, com o segundo grupo selecionado	17
Figura 9 - Painel de opções apresentado ao selecionar um grupo de chamadas.....	17
Figura 10 - Apresentação da tabela após adicionar contacto	18
Figura 11 - Apresentação no modo “List”	18
Figura 12 - Apresentação no modo “Group”	19
Figura 13 - Apresentação de chamadas respondidas.....	19
Figura 14 - Apresentação de chamadas perdidas.....	19
Figura 15 - Apresentação de chamadas rejeitadas.....	20
Figura 16 - Apresentação de chamadas enviadas	20
Figura 17 - Apresentação de chamadas recebidas.....	21
Figura 18 - Apresentação de chamadas cuja origem é o contacto selecionado	21
Figura 19 - Apresentação de chamadas cujo destino é o contacto selecionado	22
Figura 20 - Apresentação da janela de chamada a decorrer	22
Figura 21 - Apresentação da janela quando o telefone “toca”	23
Figura 22 - Apresentação da janela inicial.....	24
Figura 23 - Mensagem de erro ao inicializar um telefone com um número no formato errado.....	24
Figura 24 - Mensagem que informa que o número introduzido já tem uma sessão aberta	24
Figura 25 - Apresentação da caixa de diálogo para introduzir o número a chamar	25
Figura 26 - Apresentação de mensagem de erro ao chamar um telefone indisponível	26
Figura 27 - Apresentação da caixa de diálogo para selecionar o contacto a chamar	26
Figura 28 - Apresentação da agenda de contactos a partir da janela do telefone	27

1. Introdução

Este relatório contém as etapas e a implementação de uma aplicação de interface gráfica que simula um telefone. A implementação desta aplicação gráfica consistiu em duas etapas distintas. São elas:

- A implementação de uma aplicação gráfica que manipula uma agenda de contactos. As funcionalidades desta aplicação incluem:
 - Adicionar, editar e remover contactos;
 - Listar contactos que obedeçam a determinados critérios;
 - Abrir e guardar agendas de contactos.
- A implementação de uma aplicação gráfica que simule um telefone. As funcionalidades desta aplicação incluem:
 - Realizar chamadas para outros telefones “abertos” introduzindo um número um contacto da agenda;
 - Agrupar as chamadas enviadas/recebidas para/de a mesma origem/destino, mostrando a soma das durações das chamadas de cada grupo;
 - Listar chamadas do registo que obedeçam a determinados critérios;
 - Adicionar ou editar um contacto através da chamada seleccionada;
 - Guardar a informação do telefone (agenda de contactos e registo de chamadas) para que o telefone, ao “ligar”, recupere as informações da sessão anterior.

2. Implementação do gestor de agenda de contactos

1.1. *Back-end*

1.1.1. Classe *Contact*

A classe *Contact* com um nome, uma data de nascimento e um conjunto de números de telefone, representados pelos campos *name*, *birthDate* e *telephones*, respetivamente.

A estrutura de dados utilizada para armazenar os telefones de um contacto foi a *LinkedHashSet* porque não é interessante que números repetidos estejam no conjunto e a obtenção dos números deve ser realizada pela ordem em que foram inseridos.

Esta classe está devidamente encapsulada, visto que os seus campos são imutáveis e inacessíveis fora da classe, podendo haver acesso de leitura aos campos da classe através dos métodos *getters*: *getName()*, *getBirthDate()* e *getTelephones()*.

É importante salientar esta classe explora o mecanismo de polimorfismo que o Java disponibiliza, sobrepondo os métodos *toString()*, *equals()* e *hashCode()* da classe *Object* e o método *compareTo()* da interface *Comparable*.

O método *toString()* permite converter um contacto em *String*.

O método *equals()* verifica se duas instâncias de contacto são iguais.

O método *hashCode()* obtém o código de distribuição de um contacto. Este método permite a implementação de tabelas de distribuição que tenham instâncias de contactos como chaves.

O método *compareTo()* permite comparar duas instâncias de contacto, possibilitando a ordenação natural de uma coleção de contactos.

1.1.2. Classe *Utils*

A classe *Utils* implementa métodos de utilidade geral. Os métodos implementados são *actualize()*, *foreachV()* e *greater()* e servem essencialmente para manipular os contentores associativos que armazenam as informações da agenda de contactos.

O método *actualize()* verifica se existe algum valor associado a uma chave “passada por parâmetro”. Se existir um valor, aplica uma ação sobre a chave. Caso não exista um valor, adiciona uma entrada com a chave e o valor “passados por parâmetro”. É importante salientar a utilização de interfaces funcionais do Java na implementação deste método. Tanto o valor como a chave não são passados por parâmetro diretamente. Pelo contrário, são passadas por parâmetro duas instâncias de *Supplier* que fornecem, respetivamente, a chave e o valor. A ação aplicada sobre a chave é uma instância de *Function*.

O método *foreachV()* realiza uma ação sobre os valores das coleções associadas às chaves de um contentor associativo passado por parâmetro. A ação aplicada é uma instância de *Consumer*.

O método *greater()* devolve a coleção de chaves cujos valores associados são os maiores dado um comparador.

1.1.3. Classe *NoteBook*

A classe *NoteBook* representa uma agenda de contactos.

Os contactos são armazenados no campo *contacts*, um contentor associativo onde a chave é o nome do contacto. A estrutura de dados escolhida foi *TreeMap* para manter os contactos ordenados por ordem alfabética (ordem natural dos nomes).

O campo *telephones* é um contentor associativo onde a chave é o número de telemóvel e o valor são os contactos que têm esse número associado. Para este campo, foi escolhida a estrutura de dados *HashMap*, visto que não há necessidade de ordenação e a inserção (*put()*) e obtenção (*get()*) são mais eficientes.

O campo *birthdays* é um contentor associativo onde a chave é a data e o valor é o conjunto ordenado de contactos que partilham a mesma data de nascimento. Para este campo, foi escolhida a estrutura de dados *TreeMap*, uma vez que as entradas do contentor associativo têm de estar ordenadas pela data, da mais recente para a mais antiga, tendo em consideração apenas o dia e o mês.

Para manipular o conteúdo dos contentores associativos, foram implementados os métodos *add()*, *remove()* e *clear()*.

O método *add()* é sobrecarregado. A primeira sintaxe recebe um contacto a adicionar na agenda e a adição é realizada através de três chamadas do método *actualize()* da classe *Utils*, onde em cada chamada são passados cada um dos contentores associativos e as expressões lambda que implementam as instâncias de *Supplier* e *Function* do método *actualize()* de forma que a adição de um novo contacto respeite as condições exigidas.

O método *remove()* remove um contacto da agenda e o método *clear()* remove todos os contactos da agenda.

Foram implementados os métodos *getters* e métodos de escrita e leitura para guardar a agenda de contactos num ficheiro ou carregar uma agenda de contactos de um ficheiro.

Foram implementados os métodos *greaterContacts()*, *greaterTelephones()*, *greaterDates()* para obter, respetivamente, os telefones com maior número de contactos, os contactos com maior número de telefones e as datas que têm o maior número de contactos que partilham o mesmo aniversário. Estes métodos foram implementados através da chamada do método *greater* da classe *Utils*, passando

como parâmetros o contentor associativo que melhor permite obter a informação desejada e o respetivo comparador implementado através de uma expressão lambda.

1.2. *Front-End*

As classes implementadas para a interface gráfica da aplicação já estavam implementadas quase na totalidade pelo docente, sendo que a parte que foi implementada por nós apenas replicou a lógica daquilo que já estava implementado, utilizando os recursos implementados em *back-end*.

1.2.1. Classe *DocumentNumber*

A classe *DocumentNumber* estende a classe *PlainDocument* e permite validar a introdução de números de telefone em campos de texto.

1.1.1. Classe *TextPanel*

A classe *TextPanel* estende a classe *JPanel* e nada mais é do que um campo de texto com um título passado por argumento no construtor.

1.1.2. Classe *AbstractDialog*

A classe abstrata *AbstractDialog* estende a classe *JDialog*, sendo uma caixa de diálogo abstrata, tendo um painel personalizado, sendo possível adicionar quaisquer componentes e sendo possível atribuir uma ação de submissão personalizada, representado pelo *Consumer* passado por parâmetro no construtor.

1.1.3. Classe *ContactDialog*

A classe *ContactDialog* estende a classe *AbstractDialog*, sendo uma caixa de diálogo para instanciar ou editar um contacto.

Figura 1 - Apresentação da caixa de diálogo para a criar ou editar um contacto

1.1.4. Classe *DatePanel*

A classe *DatePanel* permite a leitura de uma data através do conteúdo inserido em três campos de texto correspondentes ao dia, mês e ano.

Figura 2 - Apresentação do painel para inserir uma data

1.1.5. Classe *TelephonesPanel*

A classe *TelephonesPanel* estende a classe *JPanel* e permite inserir num campo de texto um número a adicionar ao conjunto de números apresentado numa área de texto.

Figura 3 - Apresentação do painel para inserir números de telefone

1.1.6. Classe NotebookFrame

A classe *NoteBookFrame* estende a classe *JFrame*, sendo a janela principal da aplicação da agenda de contactos. Esta classe tem uma barra de menus no topo, uma área de texto no centro e dois botões a sul.

Na área de texto são apresentados os registos dos contactos da agenda.

A partir dos botões a sul, é possível adicionar um contacto ou adicionar números de telefone a contactos existentes.

O menu “*File*” permite carregar a agenda a partir de um ficheiro, gravar a agenda num ficheiro e sair da aplicação.

O menu “*Edit*” permite realizar as ações dos botões a sul e eliminar um contacto.

O menu “*List*” permite realizar filtragem ao contactos a apresentar.

O menu “*With More*” permite apresentar, o retorno das funções *greater()* implementadas na classe *NoteBook*.

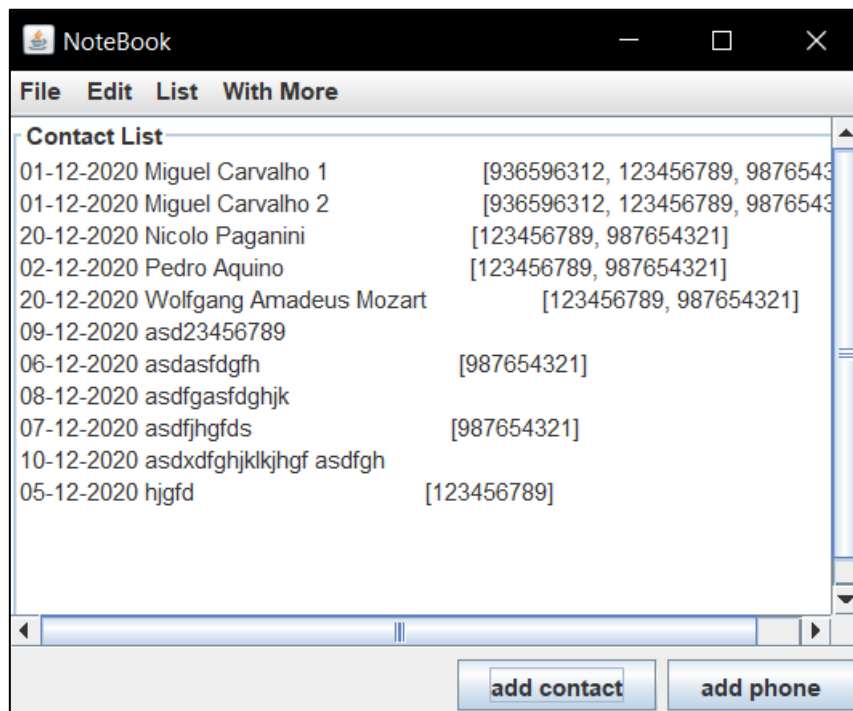


Figura 4 - Apresentação dos contactos da agenda

2. Implementação do simulador de telefones

2.1. Diagrama estático de classes UML

Segue-se o diagrama estático de classes definidas na implementação da aplicação gráfica que manipula a agenda de contactos.

2.2. *Back-End*

2.2.1. Classe *DateTime*

A classe *DateTime* representa uma data e um tempo. A data é uma instância da classe *Date*, realizada na etapa anterior, enquanto que o tempo é uma instância da classe *LocalTime*, disponível nas bibliotecas do Java.

Os construtores implementados permitem a criação de instâncias desta classe com uma determinada data e tempo, com a data e o tempo atual e com a data e o tempo representados por uma *String*. Este último construtor irá permitir instanciar um objeto desta classe através de registos nos ficheiros.

Também foram implementados métodos *getters* e foram sobrepostos os métodos *toString()*, *equals()*, *compareTo()* e *hashCode()*. Este último permite utilizar instâncias desta classe como chave de contentores associativos.

2.2.2. Classe *Call*

A classe *Call* representa uma chamada com um tipo (respondida, rejeitada ou perdida), um número de origem, um número de destino, uma data de realização e um tempo de duração. Todos os campos são instâncias do tipo *String*, à exceção da data e da duração que são, respetivamente, instâncias da classe *DateTime* (descrita anteriormente) e da classe *LocalTime* (disponível nas bibliotecas do Java).

Esta classe tem um construtor privado que recebe, por argumento, todos os campos à exceção da duração. Este construtor é chamado para instanciar chamadas nos métodos estáticos *answeredCall()*, *missedCall()* e *declinedCall()*, de forma a impedir que sejam instanciadas chamadas que não correspondam a estes três tipos. Os construtores públicos, por sua vez, permitem instanciar uma chamada através de uma *String* que tenha sido gerada pelo método *toString()* desta mesma classe, ou criar uma cópia de uma chamada passada por parâmetro.

Foram implementados os métodos *getters* e um único método setter, *setDuration()*, visto que a duração de uma chamada só poderá ser determinada no término da mesma, e não quando for instanciada.

2.2.3. Classe *Record*

A classe *Record* representa um registo de chamadas.

O campo *number* é o número de telefone associado ao registo.

As chamadas são armazenadas no campo *calls*, um contentor associativo onde a chave é a data de realização da mesma. A estrutura de dados escolhida foi *TreeMap*, visto que é interessante manter as chamadas por ordem de data, da mais recente para a mais antiga (comparador inverso ao da ordem natural).

O campo *groups* é um contentor associativo onde para cada última chamada de/para um determinado número, existe um contentor associativo de chamadas equivalente o campo *calls*, com as chamadas realizadas de/para esse mesmo número.

Os métodos implementados permitem adicionar uma chamada ao registo, obter os campos do registo, guardar/carregar o registo de chamadas para/de um ficheiro.

2.2.4. Classe *Telephone*

A classe *Telephone* representa um telefone com um número, uma agenda de contactos e um registo de chamadas, representados pelos campos *number*, *noteBook* e *record*, respetivamente. O campo *pathname* é diretoria onde serão armazenadas as informações relativas à agenda de contactos e ao registo de chamadas do telefone instanciado.

O construtor recebe por parâmetro o número de telefone e a diretoria onde verifica se o telefone já foi instanciado noutra sessão, carregando os dados da agenda de contactos e do registo de chamadas.

Foram implementados os métodos *getters* para obtenção dos campos da classe, bem como o método *save()* para guardar as informações da atual sessão (agenda de contactos e do registo de chamadas) na diretoria passada por parâmetro no construtor.

O método *resolveNumber()* recebe um número de telefone por parâmetro, devolvendo o respetivo nome de contacto associado ao telefone caso apenas exista um contacto associado ao número. Se o número for o próprio, devolve “Me”.

O método *resolveName()* realiza a operação inversa ao método *resolveNumber()*, ou seja, para uma dada *String number*, a expressão

```
number.equals( telephone.resolveName( telephone.resolveNumber( number ) ) )
```

é sempre verdadeira.

2.3. Front-End

2.3.1. Classe *ContactPanel*

Esta classe estende a classe *JPanel*, representando um painel para a seleção de um número a partir da agenda de contactos.

O painel contém duas caixas de seleção do tipo *JComboBox*. A primeira caixa de seleção tem como itens os nomes dos contactos disponíveis na agenda aquando da inicialização do painel no construtor. A segunda caixa de seleção tem os números do contacto cujo nome é o item selecionado na primeira caixa de seleção. Sempre que é selecionado um item na primeira caixa, a segunda é atualizada com os números do respetivo contacto, através de programação *event driven*.

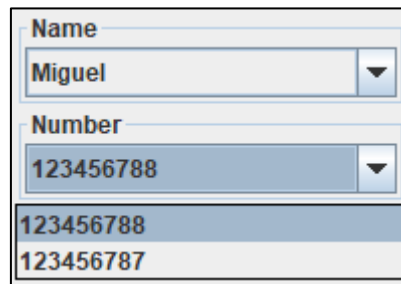


Figura 5 - Painel de seleção de um número da agenda de contactos

A classe disponibiliza o método *getNumber()* para obter o número selecionado e o método *refresh()* para realizar uma nova leitura da agenda de contactos, de forma a atualizar os nomes de contacto a selecionar na primeira caixa de seleção.

2.3.2. Classe *CallTable*

A classe *CallTable* é uma tabela do tipo *JTable*. Esta tabela serve para apresentar uma coleção de chamadas, tendo como colunas os campos da classe *Call*. Tem um telefone representado por uma instância da classe *Telephone*, de forma a resolver os números através do método *resolveNumber()* antes de apresentar a coleção de tabelas.

Type	Origin	Destination	Date	Duration
answered	Me	Miguel	18-02-2021 2...	00:00:03
declined	Me	Miguel	18-02-2021 2...	00:00:00
missed	Me	Miguel	18-02-2021 2...	00:00:00
missed	Miguel	Me	18-02-2021 2...	00:00:00
declined	Miguel	Me	18-02-2021 2...	00:00:00
answered	Miguel	Me	18-02-2021 2...	00:00:01
answered	Me	Miguel	17-02-2021 2...	00:00:04
declined	Me	Miguel	17-02-2021 0...	00:00:00
missed	Me	Miguel	17-02-2021 0...	00:00:00
answered	Miguel	Me	17-02-2021 0...	00:00:05
missed	Miguel	Me	17-02-2021 0...	00:00:00
missed	Miguel	Me	17-02-2021 0...	00:00:00

Figura 6 - Apresentação de uma *CallTable*

O conteúdo é atualizado através da função *refresh()* que recebe como parâmetros um *Supplier* de uma coleção de chamadas a apresentar e um *Predicate* para filtrar as chamadas que devem ser apresentadas. Este método é sobrecarregado, sendo possível atualizar a tabela de acordo com o *Supplier* e o *Predicate* utilizados na última chamada.

2.3.3. Classe *RecordPanel*

A classe *RecordPanel* é um painel do tipo *JPanel*. Este painel é responsável por apresentar as chamadas de dois modos diferentes:

- O modo “*List*” apresenta uma única tabela contendo a listagem de todas as chamadas por ordem de data. Este modo é ativado por defeito na inicialização e aquando da chamada do método *setListDisplay()*;

List				
Type	Origin	Destination	Date	Duration
answered	asdfg	Me	18-02-2021 23:...	00:00:02
answered	Me	Miguel	18-02-2021 23:...	00:00:03
declined	Me	Miguel	18-02-2021 23:...	00:00:00
missed	Me	Miguel	18-02-2021 23:...	00:00:00
missed	Miguel	Me	18-02-2021 23:...	00:00:00
declined	Miguel	Me	18-02-2021 23:...	00:00:00
answered	Miguel	Me	18-02-2021 23:...	00:00:01
answered	Me	Miguel	17-02-2021 20:...	00:00:04
declined	Me	Miguel	17-02-2021 04:...	00:00:00
missed	Me	Miguel	17-02-2021 04:...	00:00:00
answered	Miguel	Me	17-02-2021 04:...	00:00:05
missed	Miguel	Me	17-02-2021 04:...	00:00:00
missed	Miguel	Me	17-02-2021 04:...	00:00:00

Figura 7 - Modo “*List*” do *RecordPanel*

- O modo “*Group*” apresenta um painel dividido em dois painéis (*JSplitPanel*). O painel principal apresenta a tabela de chamadas agrupadas de acordo com o agrupamento realizado no contentor associativo *groups* da classe *Record*. A duração apresentada é a soma da duração de todas as chamadas do grupo. O outro painel expande o grupo selecionado, apresentando todas as chamadas do grupo selecionado na outra tabela. Este modo é ativado aquando da chamada do método *setGroupDisplay()*;

Group				
Type	Origin	Destination	Date	Duration
answered	asdfg	Me	18-02-2021 23:55	00:00:02
missed	Me	Miguel	17-02-2021 04:25	00:00:07
missed	Miguel	Me	17-02-2021 04:00	00:00:06
Expanded calls to Miguel				
Type	Origin	Destination	Date	Duration
answered	Me	Miguel	18-02-2021 23:43	00:00:03
declined	Me	Miguel	18-02-2021 23:43	00:00:00
missed	Me	Miguel	18-02-2021 23:43	00:00:00
answered	Me	Miguel	17-02-2021 20:24	00:00:04
declined	Me	Miguel	17-02-2021 04:25	00:00:00
missed	Me	Miguel	17-02-2021 04:25	00:00:00

Figura 8 - Modo "Group" do RecordPanel, com o segundo grupo selecionado

Sempre que é selecionada um grupo de chamadas na tabela principal do modo "Group", é apresentado um painel com opções. Até à data de término deste trabalho, apenas foi possível implementar a opção de adicionar ou editar um contacto cujo número corresponde ao grupo de chamadas selecionado.

Options	Group				
	Type	Origin	Destination	Date	Duration
Add/Edit contact	answered	123456780	Me	18-02-2021 23:55	00:00:02
	missed	Me	Miguel	17-02-2021 04:25	00:00:07
	missed	Miguel	Me	17-02-2021 04:00	00:00:06
Expanded calls from 123456780					
	Type	Origin	Destination	Date	Duration
	answered	123456780	Me	18-02-2021 23:55	00:00:02

Figura 9 - Painel de opções apresentado ao selecionar um grupo de chamadas

Ao carregar no botão, é apresentado um ContactDialog, utilizado na etapa anterior, para adicionar um novo contacto. Após adicionar o contacto, a próxima atualização da tabela terá em consideração as alterações feitas na agenda de contactos.

Options	Group				
	Add/Edit contact				
	Type	Origin	Destination	Date	Duration
	answered	asdfgh	Me	18-02-2021 23:55	00:00:02
	missed	Me	Miguel	17-02-2021 04:25	00:00:07
	missed	Miguel	Me	17-02-2021 04:00	00:00:06
	Expanded calls from asdfgh				
	Type	Origin	Destination	Date	Duration
	answered	asdfgh	Me	18-02-2021 23:55	00:00:02

Figura 10 - Apresentação da tabela após adicionar contacto

À semelhança da classe *CallTable*, esta classe disponibiliza o método *refresh()* para atualizar a(s) tabela(s) de acordo com um predicado. Este método, por sua vez, chama o método *refresh()* da classe *CallTable*, passando o mesmo *Predicate* e um *Supplier* que varia de acordo com o modo de apresentação selecionado. A sobrecarga deste método permite a atualização da(s) tabela(s) de acordo com o *Predicate* utilizado na última chamada.

2.3.4. Classe *FilterPanel*

Esta classe estende a classe *JPanel*, sendo o painel responsável pela filtragem do conteúdo apresentado numa *RecordPanel* a este associado. Este painel, disponibiliza os seguintes filtros:

- “Display”, permitindo selecionar o modo de apresentação do *RecordPanel*;

Filter					
Display	Type	Direction	Origin	Destination	
	<input checked="" type="radio"/> All	<input checked="" type="radio"/> All	<input checked="" type="radio"/> All	<input checked="" type="radio"/> All	
	<input type="radio"/> Answered	<input type="radio"/> Sent	<input type="radio"/> Number	<input type="radio"/> Number	
	<input type="radio"/> Missed	<input type="radio"/> Received	<input type="radio"/> Contact	<input type="radio"/> Contact	
<input checked="" type="radio"/> List	<input type="radio"/> Group				
List					
Type	Origin	Destination	Date	Duration	
answered	asdfgh	Me	18-02-2021 23:55	00:00:02	
answered	Me	Miguel	18-02-2021 23:43	00:00:03	
declined	Me	Miguel	18-02-2021 23:43	00:00:00	
missed	Me	Miguel	18-02-2021 23:43	00:00:00	
missed	Miguel	Me	18-02-2021 23:42	00:00:00	
declined	Miguel	Me	18-02-2021 23:42	00:00:00	
answered	Miguel	Me	18-02-2021 23:42	00:00:01	
answered	Me	Miguel	17-02-2021 20:24	00:00:04	
declined	Me	Miguel	17-02-2021 04:25	00:00:00	
missed	Me	Miguel	17-02-2021 04:25	00:00:00	
answered	Miguel	Me	17-02-2021 04:01	00:00:05	
missed	Miguel	Me	17-02-2021 04:00	00:00:00	
missed	Miguel	Me	17-02-2021 04:00	00:00:00	

Figura 11 - Apresentação no modo “List”

Filter

Display: ☐ List ☒ **Group**

Type: ☒ All ☐ Answered ☐ Missed ☐ Declined

Direction: ☒ All ☐ Sent ☐ Received

Origin: ☒ All ☐ Number ☐ Contact

Destination: ☒ All ☐ Number ☐ Contact

Group

Type	Origin	Destination	Date	Duration
answered	asdfgh	Me	18-02-2021 23:55	00:00:02
missed	Me	Miguel	17-02-2021 04:25	00:00:07
missed	Miguel	Me	17-02-2021 04:00	00:00:06

Expanded calls from asdfgh

Type	Origin	Destination	Date	Duration
answered	asdfgh	Me	18-02-2021 23:55	00:00:02

Figura 12 - Apresentação no modo "Group"

- "Type", apresentando as chamadas de um determinado tipo (respondidas, rejeitadas ou perdidas);

Filter

Display: ☒ **List** ☐ Group

Type: ☐ All ☒ **Answered** ☐ Missed ☐ Declined

Direction: ☒ All ☐ Sent ☐ Received

Origin: ☒ All ☐ Number ☐ Contact

Destination: ☒ All ☐ Number ☐ Contact

List

Type	Origin	Destination	Date	Duration
answered	asdfgh	Me	18-02-2021 23:55	00:00:02
answered	Me	Miguel	18-02-2021 23:43	00:00:03
answered	Miguel	Me	18-02-2021 23:42	00:00:01
answered	Me	Miguel	17-02-2021 20:24	00:00:04
answered	Miguel	Me	17-02-2021 04:01	00:00:05

Figura 13 - Apresentação de chamadas respondidas

Filter

Display: ☒ **List** ☐ Group

Type: ☐ All ☐ Answered ☒ **Missed** ☐ Declined

Direction: ☒ All ☐ Sent ☐ Received

Origin: ☒ All ☐ Number ☐ Contact

Destination: ☒ All ☐ Number ☐ Contact

List

Type	Origin	Destination	Date	Duration
missed	Me	Miguel	18-02-2021 23:43	00:00:00
missed	Miguel	Me	18-02-2021 23:42	00:00:00
missed	Me	Miguel	17-02-2021 04:25	00:00:00
missed	Miguel	Me	17-02-2021 04:00	00:00:00
missed	Miguel	Me	17-02-2021 04:00	00:00:00

Figura 14 - Apresentação de chamadas perdidas

Filter

Display
☒ List
☐ Group

Type
☐ All
☐ Answered
☐ Missed
☒ Declined

Direction
☒ All
☐ Sent
☐ Received

Origin
☒ All
☐ Number
☐ Contact

Destination
☒ All
☐ Number
☐ Contact

List

Type	Origin	Destination	Date	Duration
declined	Me	Miguel	18-02-2021 23:43	00:00:00
declined	Miguel	Me	18-02-2021 23:42	00:00:00
declined	Me	Miguel	17-02-2021 04:25	00:00:00

Figura 15 - Apresentação de chamadas rejeitadas

- “Direction”, apresentando as chamadas enviadas ou recebidas;

Filter

Display
☒ List
☐ Group

Type
☒ All
☐ Answered
☐ Missed
☐ Declined

Direction
☐ All
☒ Sent
☐ Received

Origin
☒ All
☐ Number
☐ Contact

Destination
☒ All
☐ Number
☐ Contact

List

Type	Origin	Destination	Date	Duration
answered	Me	Miguel	18-02-2021 23:43	00:00:03
declined	Me	Miguel	18-02-2021 23:43	00:00:00
missed	Me	Miguel	18-02-2021 23:43	00:00:00
answered	Me	Miguel	17-02-2021 20:24	00:00:04
declined	Me	Miguel	17-02-2021 04:25	00:00:00
missed	Me	Miguel	17-02-2021 04:25	00:00:00

Figura 16 - Apresentação de chamadas enviadas

Filter

Display: ☒ List ☐ Group

Type: ☒ All ☐ Answered ☐ Missed ☐ Declined

Direction: ☐ All ☐ Sent ☒ Received

Origin: ☒ All ☐ Number ☐ Contact

Destination: ☒ All ☐ Number ☐ Contact

List

Type	Origin	Destination	Date	Duration
answered	asdfgh	Me	18-02-2021 23:55	00:00:02
missed	Miguel	Me	18-02-2021 23:42	00:00:00
declined	Miguel	Me	18-02-2021 23:42	00:00:00
answered	Miguel	Me	18-02-2021 23:42	00:00:01
answered	Miguel	Me	17-02-2021 04:01	00:00:05
missed	Miguel	Me	17-02-2021 04:00	00:00:00
missed	Miguel	Me	17-02-2021 04:00	00:00:00

Figura 17 - Apresentação de chamadas recebidas

- “Origin”, apresentando as chamadas cuja origem é um determinado número ou um determinado contacto;

Filter

Display: ☒ List ☐ Group

Type: ☒ All ☐ Answered ☐ Missed ☐ Declined

Direction: ☐ All ☐ Sent ☐ Received

Origin: ☐ All ☐ Number ☒ Contact

Name: Miguel

Number: 123456788

Destination: ☒ All ☐ Number ☐ Contact

List

Type	Origin	Destination	Date	Duration
missed	Miguel	Me	18-02-2021 23:42	00:00:00
declined	Miguel	Me	18-02-2021 23:42	00:00:00
answered	Miguel	Me	18-02-2021 23:42	00:00:01
answered	Miguel	Me	17-02-2021 04:01	00:00:05
missed	Miguel	Me	17-02-2021 04:00	00:00:00
missed	Miguel	Me	17-02-2021 04:00	00:00:00

Figura 18 - Apresentação de chamadas cuja origem é o contacto selecionado

- “Destination”, apresentando as chamadas cujo destino é um determinado número ou um determinado contacto;

Filter

Display

☒ List
☐ Group

Type

☒ All
☐ Answered
☐ Missed
☐ Declined

Direction

☒ All
☐ Sent
☐ Received

Origin

☒ All
☐ Number
☐ Contact

Destination

☐ All
☒ Number
☐ Contact

Number

123456788

List

Type	Origin	Destination	Date	Duration
answered	Me	Miguel	18-02-2021 23:43	00:00:03
declined	Me	Miguel	18-02-2021 23:43	00:00:00
missed	Me	Miguel	18-02-2021 23:43	00:00:00
answered	Me	Miguel	17-02-2021 20:24	00:00:04
declined	Me	Miguel	17-02-2021 04:25	00:00:00
missed	Me	Miguel	17-02-2021 04:25	00:00:00

Figura 19 - Apresentação de chamadas cujo destino é o contacto selecionado

Sempre que um filtro é alterado, é lançado um evento que resulta na verificação de todos os filtros, produzindo um *Predicate*. O *Predicate* produzido é enviado como parâmetro da função *refresh()* do *RecordPanel* e a apresentação do registo de chamadas é apresentada de acordo com a filtragem selecionada.

2.3.5. Classe CallFrame

A classe *CallFrame* estende a classe *JFrame*, sendo a janela que aparece quando uma chamada é atendida, permanecendo aberta até ao término da chamada. Quando a chamada é terminada por um telefone, ficará registada em ambos os telefones. O tempo de duração é contado em segundos através de um temporizador.

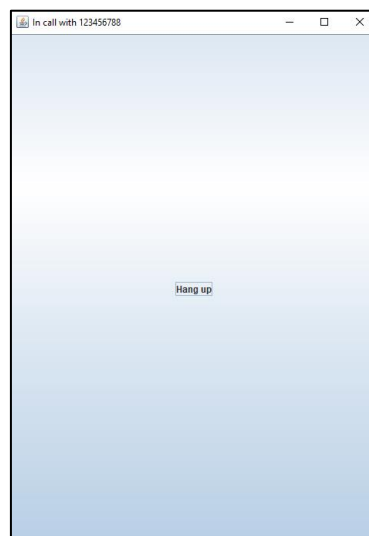


Figura 20 - Apresentação da janela de chamada a decorrer

2.3.6. Classe *RingFrame*

A classe *RingFrame* estende a classe *JFrame*, sendo a janela que aparece quando um telemóvel está a receber uma chamada. Após a apresentação desta janela, podem ser desencadeadas três ações:

- O utilizador aceita a chamada, sendo apresentada, de seguida, uma janela de chamada (*CallFrame*);
- O utilizador rejeita a chamada, sendo adicionada uma chamada rejeitada ao registo de chamadas;
- O tempo de espera acaba e a chamada é registada como não atendida.

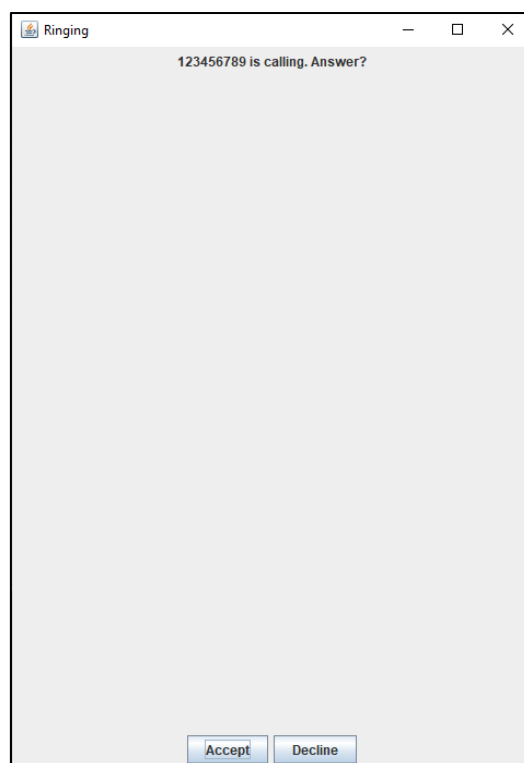


Figura 21 - Apresentação da janela quando o telefone “toca”

2.3.7. Classe *InitFrame*

A classe *InitFrame* estende a classe *JFrame*, sendo a janela de abertura de sessão de um telefone, bem como o intermediário entre as comunicações dos diversos telefones com sessão iniciada, permanecendo aberta durante a execução de toda a aplicação. Ao ser fechada, termina a sessão de todos os telemóveis e guarda as respetivas informações para serem recuperadas automaticamente na sessão seguinte.

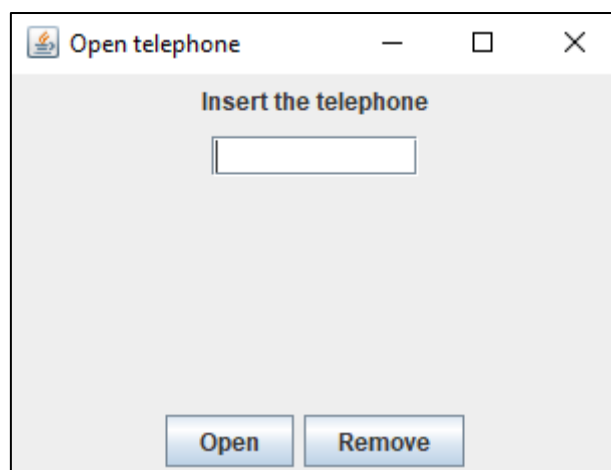


Figura 22 - Apresentação da janela inicial

Esta classe tem um contentor associativo onde a chave é um número com sessão aberta e o valor é a respetiva janela. Sempre que é submetido um número, é realizada uma validação de formato e verificada a existência de uma sessão já aberta associada ao número introduzindo, produzindo uma caixa de diálogo com a respetiva mensagem de erro.

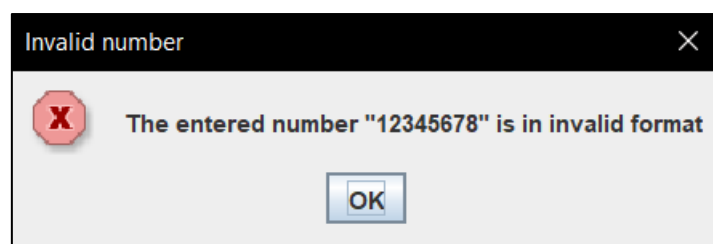


Figura 23 - Mensagem de erro ao inicializar um telefone com um número no formato errado



Figura 24 - Mensagem que informa que o número introduzido já tem uma sessão aberta

2.3.8. Classe *PhoneFrame*

A classe *PhoneFrame* representa a janela de um telefone com a sessão aberta. Ao ser fechada, todos os dados do respetivo telefone são guardados para serem automaticamente carregados em sessões futuras.

Nesta janela são apresentados o painel de filtragem (*FilterPanel*), o painel de registos (*RecordPanel*) e um painel de três botões a sul da janela.

O botão “*Call Number*” permite realizar uma chamada com destino ao número introduzido na caixa de diálogo.

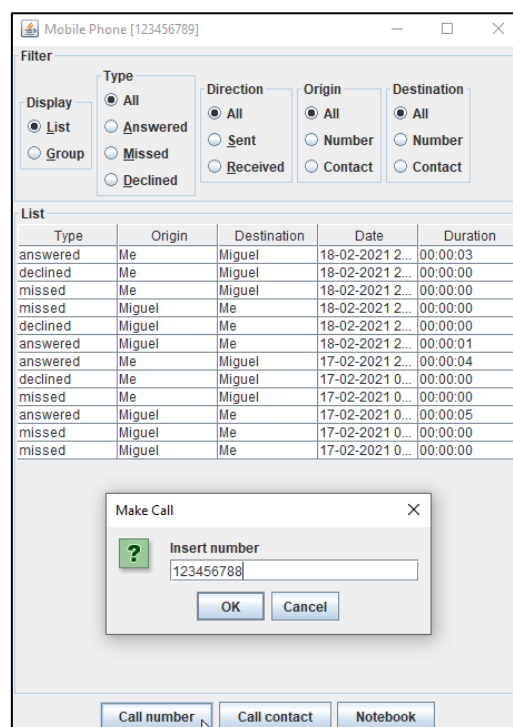


Figura 25 - Apresentação da caixa de diálogo para introduzir o número a chamar

Todas as chamadas realizadas começam por notificar a janela inicial (*InitFrame*) de que está a ser efetuada uma chamada. Após ser verificada a disponibilidade do telefone de destino, é apresentada uma *RingFrame* e a aplicação toma o rumo descrito anteriormente, consoante a ação desencadeada. Se o telefone não tiver sessão iniciada, é apresentada uma mensagem de erro e a chamada fica gravada no telefone de origem como não atendida.

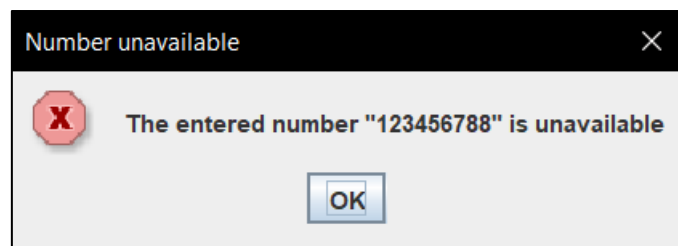


Figura 26 - Apresentação de mensagem de erro ao chamar um telefone indisponível

Também é possível realizar chamadas a partir de um número da agenda de contactos. Ao carregar no botão “Call contact”, é apresentada uma caixa de diálogo para seleccionar o contacto.

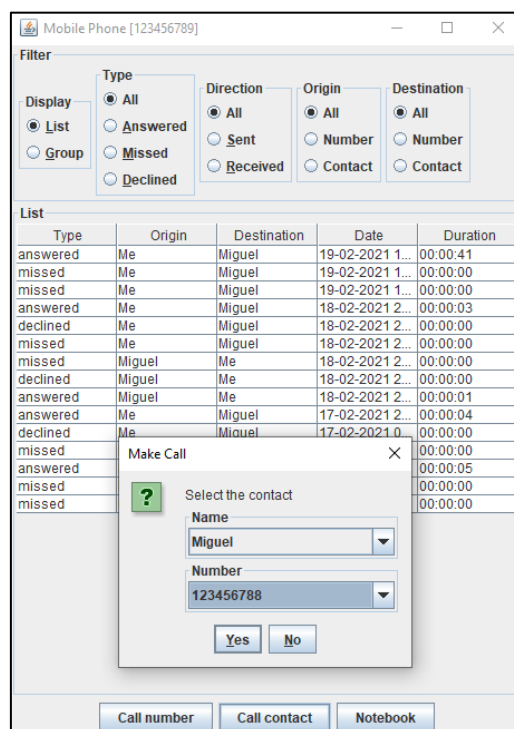


Figura 27 - Apresentação da caixa de diálogo para seleccionar o contacto a chamar

O acesso à agenda de contactos realizada na etapa anterior pode ser feito carregando no botão “Notebook”. Será apresentada a janela da aplicação da etapa anterior, não sendo necessário gravar qualquer tipo de alterações realizadas à agenda antes de fechar.

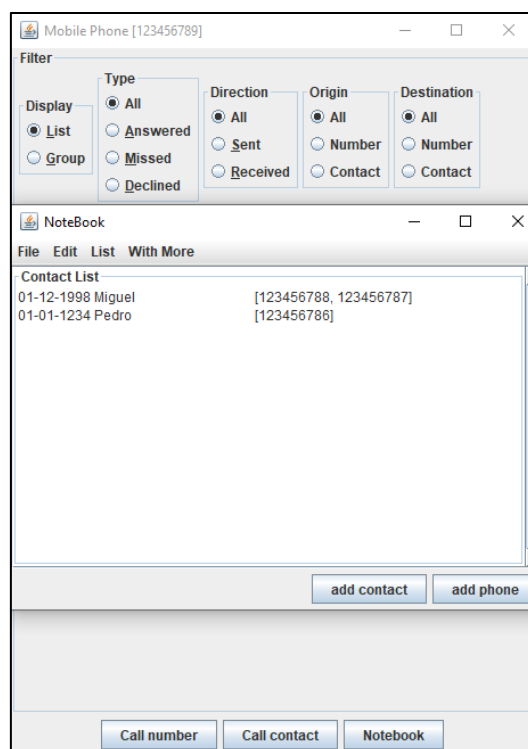


Figura 28 - Apresentação da agenda de contactos a partir da janela do telefone

Conclusão

Todos pontos enumerados na Introdução foram cumpridos através da aplicação dos conhecimentos adquiridos ao longo do semestre. Durante a realização deste trabalho, destacam-se algumas das características da linguagem Java (e das linguagens de programação orientada a objetos de forma geral) que permitiram a realização deste trabalho:

- A herança foi principalmente aplicada no desenvolvimento da interface gráfica, uma vez que permitiu a criação de janelas, painéis e tabelas específicas para serem aplicadas neste projeto, mantendo as características (métodos e campos) gerais de uma janela, painel ou tabela;
- A sobrecarga de métodos permitiu às classes *RecordPanel* e *CallTable* atualizar os conteúdos a apresentar utilizando o *Supplier* ou o *Predicate* utilizados na chamada anterior. Isto permite que métodos que, do ponto de vista do programador, representam a mesma ação, não necessitem de ter nomes diferentes apesar de terem uma sintaxe diferente;
- A variedade de estruturas de dados permitiu que a agenda de contactos e o registo de chamadas guardassem a informação de forma a cumprir com os requisitos, como por exemplo a ordenação e a não repetição de elementos, mas também a otimizar a leitura da informação consoante o tipo de acesso;
- As interfaces funcionais auxiliaram no processo de atualização da informação apresentada consoante a aplicação de filtros;
- A programação *event driven* permitiu que ações se desencadeassem em determinadas condições sem a necessidade de o processador estar constantemente a verificar, por exemplo, o estado de uma variável. Este tipo de programação é fundamental para a implementação de interfaces gráficas.