TÉCNICO LISBOA

Projecto de Programação com Objectos

22 de Setembro de 2022

Esclarecimento de dúvidas:

- Consultar sempre o corpo docente atempadamente: presencialmente ou através do endereço **po-tagus@disciplinas.tecnico.ulisboa.pt**.
- Não utilizar fontes de informação não oficialmente associadas ao corpo docente (podem colocar em causa a aprovação à disciplina).
- Não são aceites justificações para violações destes conselhos: quaisquer consequências nefastas são da responsabilidade do aluno.

Requisitos para desenvolvimento, material de apoio e actualizações do enunciado (ver informação completa na secção **[Projecto]** no Fénix):

- O material de apoio é de uso obrigatório e não pode ser alterado.
- Verificar atempadamente (mínimo de 48 horas antes do final de cada prazo) os requisitos exigidos pelo processo de avaliação.

Processo de avaliação (ver informação completa nas secções **[Projecto]** e **[Método de Avaliação]** no Fénix):

- Datas: **Ver secção Projecto da página da disciplina para as 3 entregas do projecto**. Teste prático:**2022/11/09-2022/11/10**
- Não serão consideradas quaisquer alterações aos ficheiros de apoio disponibilizados: eventuais entregas dessas alterações serão automaticamente substituídas durante a avaliação da funcionalidade do código entregue.
- Trabalhos não entregues no Fénix até final do prazo têm classificação 0 (zero) (não são aceites outras formas de entrega).
- A avaliação do projecto pressupõe o compromisso de honra de que o trabalho foi realizado pelos alunos correspondentes ao grupo de avaliação.
- **Fraudes na execução do projecto terão como resultado a exclusão dos alunos implicados do processo de avaliação**.

O objectivo do projecto é desenvolver uma aplicação que irá funcionar como o gestor de uma rede de terminais de comunicação, denominado por *prr*. Genericamente, o programa deverá permitir a gestão e consulta de clientes, terminais e comunicações. Especificamente, o sistema disponibiliza vários serviços aos seus utilizadores, entre outros: (i) registar dados sobre os clientes; (ii) registar dados sobre os terminais; (iii) registar dados sobre comunicações efectuadas; (iv) fazer pesquisas sobre comunicações efectuadas; e (v) contabilizar o saldo associado a terminais.

Este documento está organizado da seguine forma. A secção 1 apresenta as entidades do domínio da aplicação a desenvolver. As funcionalidades da aplicação a desenvolver são descritas nas secções 3 e 4. A secção 2 descreve os requisitos de desenho que a aplicação desenvolvida deve oferecer.

Neste texto, o tipo **negrito** indica um literal (i.e., é exactamente como apresentado); o símbolo ␣ indica um espaço; e o tipo *itálico* indica uma parte variável (i.e., uma descrição).

1 Entidades do Domínio

Nesta secção descrevem-se as várias entidades que vão ser manipuladas no contexto da aplicação a desenvolver. Existem vários conceitos importantes neste contexto: rede de terminais, cliente, comunicação e terminal.

Os clientes, terminais e comunicações possuem chaves únicas, cadeias de caracteres para os clientes e para os terminais e inteiros para as comunicações.

Cada rede de terminais, cliente e terminal tem um determinado *saldo*. A noção de saldo na aplicação é definida como a diferença entre os valores dos pagamentos efectuados e das dívidas por pagar tendo em conta o contexto da entidade. Assim, o saldo de uma rede de terminais é calculado como o somatório dos saldos dos seus clientes. O saldo de um cliente considera os saldos dos seus vários terminais e, finalmente, o saldo de um terminal é a diferença entre o custo das comunicações do terminal que já foram pagas e as que estão por pagar.

Na concretização desta aplicação poderá ser necessário considerar outros conceitos além dos que já foram identificados explicitamente nesta secção.

1.1 Cliente

Cada cliente, para além da chave única, tem ainda o nome (cadeia de caracteres) e o número de identificação fiscal (inteiro). A cada cliente podem estar associados vários terminais. O cliente mantém informação sobre os pagamentos efectuados (sobre comunicações passadas) e valores em dívida (comunicações cujo valor ainda não foi pago).

Existem três tipos de clientes: Normal (situação inicial, após o registo – no entanto, ver a situação da leitura de dados textuais), Gold e Platinum. O tipo de cliente influencia o custo das comunicações que efectua (ver planos tarifários). O tipo do cliente evolui nas condições indicadas na tabela 1

Antes	Depois	Condição
Normal	Gold	O saldo do cliente (após realizar um pagamento) é superior a 500 créditos.
Normal	Platinum	Não é possível
Gold	Normal	O saldo do cliente (após realizar uma comunicação) é negativo.
Gold	Platinum	O cliente realizou 5 comunicações de vídeo consecutivas (a contabilização da 5ª comunicação ainda considera que o cliente é do tipo Gold) e não tem saldo negativo.
Platinum	Normal	O saldo do cliente (após realizar uma comunicação) é negativo.
Platinum	Gold	O cliente realizou 2 comunicações de texto consecutivas (a contabilização da 2ª comunicação ainda considera que o cliente é do tipo Platinum) e não tem saldo negativo.

Tabela 1: Evolução do tipo de cliente.

1.2 Terminal

Cada terminal é identificado por uma cadeia de caracteres numérica (exactamente 6 dígitos) e está associado a um único cliente.

Os terminais realizam comunicações. Existem, pelo menos, dois tipos de terminal: básicos e sofisticados. Os terminais básicos só conseguem realizar comunicações de texto e de voz, não podendo nem iniciar nem receber comunicações de vídeo. Os terminais sofisticados podem realizar todos os tipos de comunicação: texto, voz e vídeo.

As comunicações realizadas pelo terminal são contabilizadas de acordo com o tarifário associado ao cliente. O terminal tem contabilidade própria, sendo sempre possível saber os valores dos pagamentos efectuados e dos valores devidos. Um terminal pode ter um ou mais terminais *amigos*. As comunicações entre terminais amigos são mais baratas do que entre terminais não amigos. Um terminal não pode ser incluído no seu conjunto de amigos, esta situação não deve ter qualquer efeito no estado da aplicação.

Um terminal recém-criado fica no estado de espera (*idle*); tem os valores de pagamentos e dívidas ambos a zero; e tem uma lista de amigos vazia. De seguida descreve-se o comportamento dos vários estados em que um terminal pode estar.

1.2.1 Estados dos terminais

Um terminal pode estar desligado ou ligado. Um terminal desligado não pode iniciar nem receber qualquer tipo de comunicação. Um terminal ligado tem que distinguir três situações: *Ocupado*, *Espera* e *Silêncio*. Um terminal ligado está ocupado quando está a realizar uma comunicação (de video ou de voz, as de texto são consideradas instantâneas) com outro terminal. Quando um terminal está ocupado não pode iniciar outras comunicações mas pode receber mensagens de texto. Um terminal ligado e que não está a realizar nenhuma comunicação de voz ou de vídeo está em *Silêncio* ou em *Espera*. Se estiver em silêncio, então o terminal pode iniciar qualquer tipo de comunicação suportada pelo terminal, mas só podem ser recebidas comunicações de texto. Finalmente, se o terminal estiver em espera, então pode iniciar e receber qualquer tipo de comunicação suportada pelo terminal. Considere ainda que um terminal que esteja em espera ou em silêncio não pode realizar uma comunicação interactiva com ele próprio.

Um terminal pode chegar aos vários estados nas seguintes condições (outras transições não são possíveis):

Espera – de desligado (ir para espera); de silêncio (ir para espera); de ocupado (final de comunicação);

Silêncio – de espera (colocar em silêncio); de ocupado (final de comunicação);

Ocupado – de espera ou de silêncio (início de comunicação);

Desligado – de espera ou de silêncio (ao desligar).

1.2.2 Notificações

A aplicação deve ter a capacidade de poder avisar os clientes quando um terminal com o qual tentarem comunicar mas que não tiveram sucesso passou a estar disponível. Apenas são passíveis de notificação os clientes que tentaram comunicação com um terminal e a comunicação não foi possível nessa altura. Quando uma comunicação não se efectua, regista-se a tentativa de contacto, para que, assim que seja possível a realização do contacto pretendido, se enviarem notificações aos terminais de origem. O registo da tentativa de contactos só tem lugar quando o cliente do terminal de origem tem activa a recepção de contactos falhados no instante em que se tentou efectuar a comunicação. Em qualquer momento, um cliente pode activar ou desactivar a recepção de contactos falhados.

São geradas notificações e é possível avisar um cliente nas seguintes circunstâncias:

- Um terminal desligado é colocado em silêncio (*off-to-silent*): notifica-se disponibilidade para receber comunicações de texto.
- Um terminal desligado ou em silêncio é colocado em espera (*off-to-idle* ou *silent-to-idle*): notifica-se disponibilidade para receber comunicações (qualquer suportada).
- Um terminal deixa de estar ocupado (*busy-to-idle*): notifica-se disponibilidade para receber comunicações (qualquer suportada).

As notificações contêm informação acerca da sua natureza e do terminal a que dizem respeito. Um dado evento apenas produz uma notificação por cliente e o conjunto de clientes a notificar por um terminal é limpo após o envio da notificação.

O mecanismo de entrega de notificações a um cliente deve ser flexível por forma a permitir vários meios de entrega, e.g., SMS, email, mensagens de texto. Inicialmente, cada cliente fica associado ao mecanismo de entrega por omissão mas deverá ser possível alterar o meio de entrega de um cliente. O meio de entrega por omissão corresponde a registar a notificação na aplicação e depois em apresentar as notificações (pela ordem em que elas foram recebidas) quando se visualiza a informação de um cliente..

1.3 Comunicações

Cada comunicação tem um identificador único (número inteiro, no contexto de todos os clientes). A primeira comunicação realizada com sucesso tem como identificador "1", sendo os identificadores subsequentes obtidos por incremento unitário do identificador mais recente utilizado. A comunicação contém ainda informação sobre os terminais de origem e de destino e o estado da comunicação: em curso ou terminada. As comunicações de texto têm ainda a mensagem enviada. As comunicações interactivas (vídeo e voz) possuem informação sobre a duração da comunicação. O custo de uma comunicação de texto depende do comprimento da mensagem. No caso das comunicações interactivas, o seu custo depende da sua duração. O custo depende ainda do plano tarifário associado a cada cliente (calculado no final da comunicação). Todos os cálculos envolvendo os custos das comunicações devem ser realizados sem arredondamentos.

1.4 Plano Tarifários

Cada plano tarifário define os custos para cada tipo de comunicação, baseado no nível do cliente, no tipo de comunicação, entre outras características. Os planos tarifários têm um nome único no contexto da rede de terminais a que estão associados. A rede de terminais pode oferecer vários planos tarifários mas em cada momento um cliente apenas tem um plano tarifário. A rede de terminais oferece pelo menos o plano tarifário designado como *base*. Este plano tarifário é o plano atribuído inicialmente a todos os clientes. O custos das comunicações é medido em créditos.

O custo de uma comunicação deve ser calculado quando a comunicação termina e guardado, por forma a garantir que o custo não é afectado por mudanças futuras dos planos tarifários. O custo (medido em créditos) de uma comunicação de texto com *N* caracteres no plano tarifário base está representado na tabela 2.

	Normal	**Gold**	**Platinum**
N <50	10	10	0
50 <= N <100	16	10	4
N >= 100	2 x N	2 x N	4

Tabela 2: Custo das comunicações de texto.

	Normal	**Gold**	**Platinum**
Comunicação de voz	20	10	10
Comunicação de vídeo	30	20	10

Tabela 3: Custo das comunicações de vídeo e voz para o plano tarifário base O custo está indicado em créditos por minuto.

Quando é efectuada uma comunicação de voz ou de vídeo, o seu custo no é proporcional ao tempo de conversação (medido em minutos). Se a comunicação envolver um terminal amigo, então é aplicado um desconto de 50%. O custo das comunicações de video e voz para terminais não amigos no plano tarifário base está indicado na tabela 3 .

2 Requisitos de Desenho

Devem ser possíveis extensões ou alterações de funcionalidade com impacto mínimo no código já produzido para a aplicação. O objectivo é aumentar a flexibilidade da aplicação relativamente ao suporte de novas funções. Assim, deve ser possível: definir novos tipos de clientes; definir novos tipos de comunicação; definir novos planos tarifários; definir novas formas de pesquisa; permitir a gestão de várias redes de terminais.

Embora na especificação actual não seja possível remover algumas entidades, a inclusão desta funcionalidade deve ser prevista, por forma a minimizar o impacto da sua futura inclusão.

2.1 Funcionalidade da Aplicação

A aplicação permite gerir a informação sobre as entidades do modelo, Possui ainda a capacidade de preservar o seu estado (não é possível manter várias versões do estado da aplicação em simultâneo). Deve ser possível efectuar pesquisas sujeitas a vários critérios e sobre as diferentes entidades geridas pela aplicação. Uma base de dados textual com conceitos pré-definidos pode ser carregada no início da aplicação. Inicialmente, a aplicação apenas tem informação sobre as entidades que foram carregados no arranque da aplicação.

Note-se que não é necessário concretizar a aplicação de raiz. Já é fornecido algum código, nomeadamente, o esqueleto das classes necessárias para concretizar os menus e respectivos comandos a utilizar na camada de serviços da aplicação a desenvolver, a classe `prr.app.App`, que representa o ponto de entrada da aplicação a desenvolver, algumas classes do domínio da aplicação e um conjunto de excepções a utilizar durante o desenvolvimento da aplicação. Alguns dos comandos já estão completamente finalizados, outros apenas (a grande maioria) apenas estão parcialmente concretizados e é necessário terminar a concretização dos vários métodos fornecidos.

Já é fornecido um esqueleto da aplicação a desenvolver, não sendo por isso necessário concretizar a aplicação de raiz. Este esqueleto está disponível no arquivo `prr-skeleton.jar` presentena secção *Projecto* da página da cadeira. O esqueleto inclui a classe `prr.app.App`, que representa o ponto de entrada da aplicação a desenvolver e os vários comandos e menus da aplicação a desenvolver (descritos na secção 3. Alguns dos comandos já estão completamente finalizados, outros apenas (a grande maioria) apenas estão parcialmente concretizados e é necessário terminar a concretização dos vários métodos fornecidos. Estas classes estão concretizads no package `prr.app` e nos seus sub-packages (por exemplo, `prr.app.main`).

O esqueleto inclui ainda algumas classes do domínio da aplicação e um conjunto de excepções a utilizar durante o desenvolvimento da aplicação. Será ainda necessário concretizar novas entidades no core e finalizar as que já estão parcialmente fornecidas por formaa ter um conjunto de entidades que oferecem a funcionalidade do domínio da aplicação a desenvolver. As excepções a criar na camada de serviços (ou seja nos comandos) já estão todas definidas no package `prr.app.exception`. O package `prr.core.exception` contém algumas excepções a utilizar na camada do domínio. Caso seja necessário podem ser definidas novas excepções neste package para representar situações anómalas que possam ocorrer nas entidades do domínio. Finalmente, será ainda necessário concretizar de raiz algumas classes do domínio da aplicação necessárias para suportar a operação da aplicação.

2.2 Serialização

É possível guardar e recuperar o estado actual da aplicação, preservando toda a informação relevante do domínio da aplicação e que foi descrita na secção 1.

3 Interação com o utilizador

Esta secção descreve a **funcionalidade máxima** da interface com o utilizador. Em geral, os comandos pedem toda a informação antes de proceder à sua validação (excepto onde indicado). Todos os menus têm automaticamente a opção **Sair** (fecha o menu).

As operações de pedido e apresentação de informação ao utilizador **devem** realizar-se através dos objectos *form* e *display*, respectivamente, presentes em cada comando. No caso de um comando utilizar mais do que um formulário para interagir com o utilizador então será necessário criar pelo menos mais uma instância de `Form` durante a execução do comando em causa. As mensagens são produzidas pelos métodos das bibliotecas de suporte (**po-uilib** e de classes presentes no esqueleto da aplicação. **Não** podem ser definidas novas mensagens. Potenciais omissões devem ser esclarecidas com o corpo docente antes de qualquer concretização.

A apresentação de valores monetários é sempre feita com arredondamento ao inteiro mais próximo, mas a representação interna não deve ser arredondada. A apresentação de listas de entidades do domínio (clientes, etc.) faz-se por ordem crescente da respectiva chave: dependendo dos casos, a ordem pode ser numérica ou lexicográfica (UTF-8), **não havendo** distinção entre maiúsculas e minúsculas.

Por forma a garantir uma independência entre as várias camadas da aplicação não deve haver código de interacção com o utilizador no núcleo da aplicação (*prr.core*). Desta forma, será possível reutilizar o código do núcleo da aplicação (onde é concretizado o domínio da aplicação) com outras concretizações da interface com o utilizador. Para garantir um melhor desenho da aplicação toda a lógica de negócio deve estar concretizada no núcleo da aplicação (camada *core*) e não na camada de interação com o utilizador. Assim potencia-se a reutilização de código e, além disso, o código fica concretizado nas entidades a que diz respeito, o que torna a aplicação mais legível e mais fácil de manter.

As excepções usadas no código de interacção com o utilizador para descrever situações de erro, excepto se indicado, são subclasses de `pt.tecnico.uilib.menus.CommandException` e devem ser lançadas pelos comandos (sendo depois tratadas automaticamente pela classe já existente `pt.tecnico.po.ui.Menu`). Estas excepções já estão definidas no package (fornecido) `prr.app.exception`. Outras excepções não devem substituir as fornecidas nos casos descritos. É da responsabilidade de cada grupo definir as excepções que achar necessárias no contexto das entidades do domínio da aplicação (ou reutilizar excepções

já existentes na biblioteca do Java caso façam sentido serem utilizadas). As excepções disponíveis em `prr.app.exception`. apenas devem ser utilizadas no código de interacção com o utilizador.

Sempre que existe uma interacção com o utilizador, seja para pedir dados seja para apresentar dados, é indicado (caso seja necessário) qual é o método da classe `Message` do package em causa que é responsável por devolver a mensagem a apresentar ao utilizador.

3.1 Menu Principal

As acções deste menu permitem gerir a salvaguarda do estado da aplicação, abrir submenus e aceder a alguma informação global. A lista completa é a seguinte: **Abrir**, **Guardar**, **Gestão de clientes**, **Gestão de terminais**, **Consultas**, e **Mostrar informação global sobre pagamentos e dívidas**. As várias mensagens de diálogo utilizadas nos vários comandos deste menu estão definidos nos vários métodos da classe `prr.app.main.Message`.

Alguns dos comandos que vão concretizar as funcionalidades deste menu já estão parcialmente concretizados nas várias classes do package `prr.app.main`: `DoOpen`, `DoSave`, e `DoShowGlobalPaymentsAndDebts`. Os restantes comandos (que correspondem a apresentar um submenu) já estão completamente concretizados.

3.1.1 Salvaguarda do estado actual

Inicialmente, a aplicação está vazia ou tem apenas informação sobre as entidades que foram carregados no arranque via ficheiro textual (ver 4).

O conteúdo da aplicação (que representa o estado relevante actualmente em memória da aplicação) pode ser guardado para posterior recuperação (via serialização Java: `java.io.Serializable`). Na leitura e escrita do estado da aplicação, devem ser tratadas as excepções associadas. A funcionalidade é a seguinte:

Abrir – Carrega os dados de uma sessão anterior a partir de um ficheiro previamente guardado (ficando este ficheiro associado à aplicação, para futuras operações de salvaguarda). Pede-se o nome do ficheiro a abrir (utilizando a cadeia de caracteres devolvida por `openFile()`). Caso ocorra um problema na abertura ou processamento do ficheiro, deve ser lançada a excepção `FileOpenFailedException`. A execução bem sucedida desta opção substitui toda a informação da aplicação.

Guardar – Guarda o estado actual da aplicação no ficheiro associado. Se não existir associação, pede-se o nome do ficheiro a utilizar ao utilizador (utilizando a cadeia de caracteres devolvida por `newSaveAs()`), ficando a aplicação associada a este ficheiro.

Note-se que a opção `Abrir` não suporta a leitura de ficheiros de texto (estes apenas são utilizados na inicialização da aplicação). A opção `Sair` **nunca** guarda o estado da aplicação, mesmo que existam alterações.

3.1.2 Gestão e consulta de dados da aplicação

A gestão e consulta de dados da aplicação são realizadas através das seguintes opções:

Menu de Gestão de Clientes – Abre o menu de gestão de clientes.

Menu de Gestão de Terminais – Abre o menu de gestão de terminais.

Menu de Consultas – Abre o menu de consultas (pesquisas).

3.1.3 Mostrar informação global sobre pagamentos e dívidas

Esta opção apresenta os valores globais correspondentes a pagamentos e dívidas (soma dos valores parciais para todos os clientes registados). A apresentação destes valores deve ser feita utilizando a mensagem devolvida por `Message.globalPaymentsAndDebts()`. Note-se que, embora internamente os pagamentos e as dívidas serem números reais, a apresentação dos valores deve ser arredondada ao inteiro mais próximo.

3.2 Menu de Gestão de clientes

Este menu permite efectuar operações sobre a base de dados de clientes. A lista completa é a seguinte: **Visualizar cliente**, **Visualizar todos os clientes**, **Registar cliente**, **Activar recepção de contactos falhados**, **Desactivar recepção de contactos falhados** e **Mostrar informação sobre pagamentos e dívidas de cliente.**. Os comandos deste menu já estão parcialmente concretizados nas classes do package `prr.app.client`: `DoShowClient`, `DoShowAllClients`, `DoRegisterClient`, `DoEnableClientNotifications`. `DoDisableClientNotifications` e `DoShowClientPaymentsAndDebts`. Todos os métodos correspondentes às mensagens de diálogo para este menu estão definidos na classe `Message` deste package.

Sempre que no contexto de uma operação deste menu for pedido o identificador de um cliente (utilizando a mensagem devolvida por `Message.key()`) e não existir nenhum cliente com o identificador indicado (excepto na operação de registro), a operação deve lançar a excepção `UnknownClientKeyException` Na ocorrência de excepções (estas ou outras), as operações não devem ter qualquer efeito.

3.2.1 Visualizar cliente

É pedido o identificador do cliente e apresentada a sua informação. O formato de apresentação de cada cliente é como se indica de seguida:

```
CLIENT|key|name|taxId|type|notifications|terminals|payments|debts
```

Os valores para *type* são NORMAL, GOLD, ou PLATINUM. Os valores para *notifications* são YES ou NO (notificações activas/inactivas). O campo *terminals* representa o número de terminais activos associados ao cliente, *payments* é o valor dos pagamentos do cliente e *debts* é o valor das dívidas do cliente. Se o cliente não tiver terminais, os valores destes três campos são 0 (zero).

Após esta linha, são apresentadas as notificações do cliente (caso tenha o modo de entrega por omissão), pela ordem em que foram enviadas pela aplicação utilizando o seguinte formato:

```
tipo-de-notificação|idTerminal
```

O tipo de notificação é um de **O2S** (off-to-silent), **O2I** (off-to-idle), **B2I** (busy-to-idle) ou **S2I** (silent-to-idle), tal como descritos anteriormente. Após esta visualização, considera-se que o cliente fica sem notificações registadas.

3.2.2 Visualizar todos os clientes

Apresenta todos os clientes da aplicação. O formato de apresentação de cada cliente é o descrito em 3.2.1, mas não se apresentam as notificações dos clientes nem se limpam as notificações dos clientes.

3.2.3 Registar cliente

O sistema pede o identificador que ficará associado ao cliente (identificador único). De seguida, pede o nome do cliente (`Message.name()`) e o número de identificação fiscal `Message.(taxId())`. Após o registo, o cliente fica no estado *Normal* e o registo de contactos falhados fica activo. Caso o identificador indicado já exista, esta operação deve lançar a excepção `DuplicateClientKeyException`, não se realizando o registo.

3.2.4 Activar recepção de contactos falhados

É pedido o identificador do cliente. Se o registo de contactos falhados já estava activo, o cliente não é alterado e é apresentada a mensagem devolvida por `Message.clientNotificationsAlreadyEnabled()`.

3.2.5 Desactivar recepção de contactos falhados

É pedido o identificador do cliente. Se o registo de contactos falhados já estava inactivo, o cliente não é alterado e é apresentada a mensagem devolvida por `Message.clientNotificationsAlreadyDisabled()`.

3.2.6 Mostrar informação sobre pagamentos e dívidas de cliente

O sistema pede o identificador do cliente, apresentando os valores dos seus pagamentos e dívidas utilizando a mensagem devolvida por `Message.clientPaymentsAndDebts())`.

É pedido o identificador do produto e são apresentados os lotes conhecidos para esse produto. A ordenação é como indicada para a listagem de todos os lotes.

3.3 Menu de Gestão de Terminais

Este menu permite efectuar operações sobre a base de dados de terminais. A lista completa é a seguinte: **Mostrar todos os terminais**, **Registar terminal** e **Menu da consola de um terminal**. Os comandos deste menu já estão parcialmente concretizados nas classes do package `prr.app.terminals`: `DoShowAllTerminals`, `DoRegisterTerminal`, e `DoOpenMenuTerminalConsole`. Todos os métodos correspondentes às mensagens de diálogo para este menu estão definidos na classe `prr.app.terminals.Message`.

Sempre que for pedido o identificador de um terminal (utilizando a mensagem devolvida por `Message.terminalKey()`) e o terminal não existir, deve ser lançada a excepção `UnknownTerminalKeyException` (excepto no processo de registo). Sempre que é pedido o identificador de um cliente (`Message.clientKey()`) e o identificador não existir, deve ser lançada a excepção `UnknownClientKeyException`. Na ocorrência de excepções (estas ou outras), as operações não têm efeito.

3.3.1 Mostrar todos os terminais

Mostra todos os terminais registados na rede de terminais. O formato de apresentação de cada terminal é um dos seguintes:

```
terminalType|terminalId|clientId|terminalStatus|balance-paid|balance-debts|friend1,...,friend
terminalType|terminalId|clientId|terminalStatus|balance-paid|balance-debts
```

dependendo se o terminal tem terminais amigos ou não.

Os valores para o campo *terminalType* são `BASIC` (terminais básicos) ou `FANCY` (terminais sofisticados). Os valores para o campo *terminalStatus* são `IDLE, OFF, SILENCE, BUSY`. Os valores `friend1, ..., friendN` são os identificadores dos terminais amigos e são apresentados por ordem crescente desses identificadores. Note-se que os valores dos pagamentos efectuados e os dos valores em dívida são apresentados separadamente.

3.3.2 Registar terminal

É pedido o número identificador do terminal. De seguida, o sistema pede o tipo de terminal (Message.terminalType()). A resposta deve ser *BASIC* (terminal básico) ou *FANCY* (terminal sofisticado). Se a resposta não corresponder a nenhum dos dois valores, a pergunta é repetida até se obter uma resposta válida. Finalmente, é pedido o identificador do cliente que ficará associado ao terminal. Caso o identificador do terminal seja inválido ou já esteja a ser utilizado, então esta operação deve lançar a excepção `InvalidTerminalKeyException` ou `DuplicateTerminalKeyException`, respectivamente.

3.3.3 Menu da consola de um terminal

É pedido o número identificador de um terminal, sendo aberto o correspondente menu da sua consola.

3.4 Menu de Consultas

Este menu apresenta as operações relacionadas com consultas. A lista completa é a seguinte: **Mostrar todas as comunicações**, **Mostrar comunicações feitas por um cliente**, **Mostrar comunicações recebidas por um cliente**, **Mostrar clientes sem dívidas**, e **Mostrar clientes com dívidas**, **Mostrar terminais sem actividade**, **Mostrar terminais com saldo positivo**. Estes comandos já estão parcialmente concretizaos nas classes do package `prr.app.lookup`: **DoShowAllCommunications**, **DoShowCommunicationsFromClient**, **DoShowCommunicationsToClient**, **DoShowClientsWithoutDebts**, **DoShowClientsWithDebts**, **DoShowUnusedTerminals**, **DoShowTerminalsWithPositiveBalance**.

No contexto de cada comando deste menu deve ser tido em conta o seguinte. Sempre que é pedido o identificador do cliente (`Message.clientKey()`), a operação em causa deve lançar a excepção `UnknownClientKeyException` se o cliente indicado não existir. Sempre que é pedido o identificador do terminal (`Message.terminalKey()`), a operação em causa lança a excepção `UnknownTerminalKeyException` se o terminal indicado não existir. A apresentação de resultados é como se indica nos casos já descritos de apresentação das várias entidades. Sempre que for feita uma consulta e nenhuma entidade satisfizer as condições associadas ao pedido, nada deve ser apresentado.

3.4.1 Mostrar todas as comunicações

Apresenta todas as comunicações da rede de terminais, uma comunicação por linha. O formato de apresentação é o seguinte:

```
type|idCommunication|idSender|idReceiver|units|price|status
```

Os possíveis valores para o campo *type* são `VOICE`, `TEXT` ou `VIDEO`. Os possíveis valores para o campo *status* são `ONGOING` (comunicação em curso) ou `FINISHED` (comunicação terminada). O valor de *units* corresponde às unidades de contabilização (caracteres ou minutos). Caso a comunicação esteja em curso, os valores de *units* e de *price* são ambos zero.

3.4.2 Mostrar comunicações feitas por um cliente)

É pedido o identificador do cliente, sendo apresentadas as comunicações iniciadas pelos seus terminais. O formato de cada comunicação é o indicado anteriormente em 3.4.1

3.4.3 Mostrar comunicações recebidas por um cliente)

É pedido o identificador do cliente, sendo apresentadas as comunicações recebidas pelos seus terminais. O formato de cada comunicação é o indicado anteriormente em 3.4.1

3.4.4 Mostrar clientes sem dívidas

São apresentados os clientes sem dívidas. Utiliza-se o formato de apresentação de um cliente descrito em 3.2.1.

3.4.5 Mostrar clientes com dívidas)

São apresentados os clientes por ordem decrescente do valor das respectivas dívidas (valores superiores a zero). Se as dívidas tiverem o mesmo valor, apresentam-se os clientes por ordem crescente do seu identificador. Utiliza-se o formato de apresentação de um cliente descrito em 3.2.1.

3.4.6 Mostrar terminais sem actividade

São apresentados os terminais que ainda não efectuaram nem receberam qualquer comunicação. Utiliza-se o formato de apresentação de um terminal descrito em 3.3.1.

3.4.7 Mostrar terminais com saldo positivo

São apresentados os terminais que que têm um valor de pagamentos estritamente superior ao valor das dívidas. Utiliza-se o formato de apresentação de um terminal descrito em 3.3.1.

3.5 Menu da consola de um terminal

Este menu apresenta as operações relacionadas com a consola de um terminal, i.e., o seu uso e administração. A lista completa é a seguinte:**Ligar terminal**, **Desligar terminal**, **Silenciar terminal**, **Adicionar amigo**, **Retirar amigo**, **Efectuar pagamento**, **Mostrar informação sobre pagamentos e dívidas**, **Enviar comunicação de texto**, **Iniciar comunicação interactiva**, **Terminar comunicação interactiva**, **Mostrar comunicações em curso**. Todos os métodos correspondentes às mensagens de diálogo para este menu estão definidos na classe `prr.app.terminal.Message`. Estas operações já estão concretizadas nas classes do package `prr.app.terminal`: **DoTurnOnTerminal**, **DoOffTurnTerminal**, **DoSilenceTerminal**, **DoAddFriend**, **DoRemoveFriend**, **DoPerformPayment**, **DoShowTerminalPaymentsAndDebts**, **DoSendTextCommunication**, **DoStartInteractiveCommunication**, **DoEndInteractiveCommunication**, **DoShowOngoingCommunications**.

No contexto de cada comando deste menu deve ser tido em conta o seguinte. Sempre que é pedido o identificador do cliente (`Message.clientKey()`), a operação em causa deve lançar a excepção `UnknownClientKeyException`, se o cliente indicado não existir. Sempre que é pedido o identificador do terminal (`Message.terminalKey()`), é lançada a excepção `UnknownTerminalKeyException`, se o terminal indicado não existir.

3.5.1 Colocar terminal em espera

Coloca em espera o terminal selecionado. Se este já estiver em espera, é apresentada a mensagem `Message.alreadyIdle()`. Se não for possível colocar o terminal em espera, o comando não tem efeito.

3.5.2 Desligar terminal

Desliga o terminal selecionado. Se este já estiver desligado, é apresentada a mensagem devolvida por `Message.alreadyOff()`. Se não for possível desligar o terminal, o comando não tem efeito.

3.5.3 Silenciar terminal

Silencia o terminal selecionado. Se este já estiver silenciado, é apresentada a mensagem devolvida por `Message.alreadySilent()`. Se não for possível colocar o terminal em silêncio, o comando não tem efeito.

3.5.4 Adicionar amigo

Adiciona um terminal amigo ao terminal selecionado. É pedido o identificador do terminal a adicionar à lista de amigos. Se o terminal indicado já fizer parte da lista de amigos, a operação termina sem alterações.

3.5.5 Retirar amigo

É pedido o identificador do terminal a retirar da lista de amigos. Se o terminal indicado não fizer parte da lista de amigos, a operação termina sem alterações.

3.5.6 Efectuar pagamento

Efectua o pagamento de uma dada comunicação do terminal selecionado. É pedido o identificador da comunicação a pagar (`Message.commKey()`). A comunicação tem de pertencer ao terminal selecionado. Caso contrário, é apresentada a mensagem devolvida por `Message.foreignCommunication()`.

3.5.7 Mostrar informação sobre pagamentos e dívidas

São apresentados os valores dos pagamentos e das dívidas do terminal selecionado utilizando a mensagem devolvida por `Message.terminalPaymentsAndDebts()`.

3.5.8 Enviar comunicação de texto

Esta operação está disponível para um terminal que não esteja desligado ou em comunicação. Permite enviar uma comunicação de texto para outro terminal. É pedido o número do terminal de destino e o corpo da mensagem (`Message.textMessage()`).

3.5.9 Iniciar comunicação interactiva

Esta operação está disponível caso o terminal selecionado não esteja desligado ou em comunicação. Permite estabelecer uma comunicação interactiva entre o terminal selecionado com outro terminal. É pedido o número do terminal de destino e o tipo de comunicação (`Message.commType()`). A resposta deve ser uma das seguintes opções (cadeia de caracteres): `VIDEO` ou `VOICE`. Se a resposta não corresponder a nenhum destes valores, a pergunta é repetida até se obter uma resposta válida. Quando o terminal de destino está desligado, é apresentada a mensagem devolvida por `Message.destinationIsOff()`. Quando o terminal de destino está ocupado, é apresentada a mensagem `Message.destinationIsBusy()`. Quando o terminal de destino está em silêncio, é apresentada a mensagem `Message.destinationIsSilent()`.

Se se tenta iniciar uma comunicação não suportada pelo terminal selecionado ou pelo terminal de destino, deve ser apresentada a mensagem devolvida por `Message.unsupportedAtOrigin()` ou `unsupportedAtDestination()`, respectivamente. Em qualquer destes casos, a operação termina sem qualquer acção.

3.5.10 Terminar comunicação interactiva

Termina a comunicação em curso iniciada pelo terminal selecionado. Esta operação está disponível se o terminal selecionado iniciou uma comunicação interactiva e ainda não a finalizou. Note-se que o terminal de destino da comunicação em curso não pode interromper a comunicação, pelo que esta operação não está disponível para este terminal. Esta operação terminar a comunicação e registar a sua duração. É pedida a duração da comunicação (`Message.duration()`, em segundos). Após o término da comunicação, é apresentado o seu custo, através da mensagem devolvida por `Message,communicationCost()`.

3.5.11 Mostrar comunicação em curso

É apresentada a comunicação em curso do terminal selecionado (de acordo com o formato indicado em 3.4.1). Se não houver nenhuma comunicação em curso, é apresentada a mensagem devolvida por `Message.noOngoingCommunication()`.

4 Leitura de Dados a Partir de Ficheiros Textuais

Além das opções de manipulação de ficheiros descritas na secção §3.1.1, é possível iniciar a aplicação com um ficheiro de texto especificado pela propriedade Java com o nome `import`. Este ficheiro contém a descrição dos clientes e terminais a carregar no estado inicial da aplicação. Cada linha deste ficheiro textual descreve uma dada entidade a carregar no estado inicial do sistema.

Pode assumir que não existem entradas mal-formadas nestes ficheiros. A codificação dos ficheiros a ler é garantidamente `UTF-8`. Sugere-se a utilização do método `String.split` para o processamento preliminar destas linhas.

As várias entidades têm os formatos descritos abaixo. Assume-se que os campos de cada entidade que correspondem a uma cadeia de caracteres não podem conter o carácter separador |. Cada linha tem uma descrição distinta mas segue um dos seguintes formatos:

```
CLIENT|id|nome|taxId
terminal-type|idTerminal|idClient|state
FRIENDS|idTerminal|idTerminal1,...,idTerminalN
```

As definições de clientes precedem sempre as dos terminais. As ligações entre amigos estão sempre após a definição das restantes entidades. Para os terminais, *terminal-type* é `BASIC` ou `FANCY`. De seguida apresenta-se um exemplo de conteúdo do ficheiro inicial:

```
CLIENT|cli001|Manuel␣Pinheiro|103443
CLIENT|cli002|Pedro␣Pinheiro|103447
CLIENT|Cli201|Ludgero␣Oliveira|103440
CLIENT|cli␣Es|Maria␣Eucalipto|103441
CLIENT|01|Oliveira␣Preto|103547
CLIENT|cli003|Pedro␣Oliveira|103449
BASIC|969001|cli001|ON
BASIC|969003|cli002|ON
FANCY|969002|cli002|SILENCE
FANCY|969007|cli␣Es|ON
BASIC|969008|cli003|OFF
BASIC|969009|cli003|OFF
BASIC|969010|cli003|ON
FANCY|969006|cli003|ON
FANCY|969005|cli003|ON
BASIC|969004|cli003|ON
FRIENDS|969001|969008,969009,969004
FRIENDS|969004|969001
FRIENDS|969003|969008
```

Note-se que o programa **nunca** produz ficheiros com este formato, apenas lê ficheiros com este formato.

5 Execução dos Programas e Testes Automáticos

Usando os ficheiros `test.import`, `test.in` e `test.out`, é possível verificar automaticamente o resultado correcto do programa. O primeiro ficheiro representa o ficheiro de texto com a descrição do estado inicial das entidades do sistema, o segundo ficheiro simula um utilizador, contendo os caracteres a serem fornecidos ao programa durante a sua execução e o terceiro ficheiro irá conter todos os caracteres escritos pelo programa (via biblioteca *po-uilib*) durante a sua execução. Note-se que pode ser necessária a definição apropriada da variável de ambiente `CLASSPATH` (ou da opção equivalente `-cp` do comando `java`), para localizar as classes do programa, incluindo a que contém o método correspondente ao ponto de entrada da aplicação (`prr.app.App.main`). As propriedades são tratadas automaticamente pelo código de apoio.

```
java -cp -Dimport=test.import -Din=test.in -Dout=test.outhyp prr.app.App
```

Assumindo que aqueles ficheiros estão no directório onde é dado o comando de execução, o programa produz o ficheiro de saída `test.outhyp`. Em caso de sucesso, os ficheiros da saída esperada (`test.out`) e obtida (`test.outhyp`) devem ser iguais. A comparação pode ser feita com o comando:

```
diff -b test.out test.outhyp
```

Este comando não deve produzir qualquer resultado quando os ficheiros são iguais. Note-se, contudo, que este teste não garante o correcto funcionamento do código desenvolvido, apenas verifica alguns aspectos da sua funcionalidade.

6 Notas de Concretização

Tal como indicado neste documento, algumas classes fornecidas como material de apoio, são de uso obrigatório e **não** podem ser alteradas: as várias classes `Message`, `Label` e de excepção presentes em diferentes subpackages de `prr.app`. Outras dessas classes são de uso obrigatório e têm de ser alteradas: os diferentes comandos presentes em subpackages de `prr.app` e algumas classes do domínio da aplicação já parcialmente concretizadas em `prr.core`.

A serialização Java usa as classes da package `java.io`, em particular, a interface`java.io.Serializable` e as classes de leitura `java.io.ObjectInputStream` e escrita `java.io.ObjectOutputStream` (entre outras).