TÉCNICO LISBOA

Projecto de Programação com Objectos

8 de Outubro de 2014

Esclarecimento de dúvidas:

- Consultar sempre o corpo docente atempadamente: presencialmente ou através do endereço **po-tagus@disciplinas.ist.utl.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 atualizações do enunciado (ver informação completa na secção **[Projeto]** no Fenix):

- 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 desenvolvimento.

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

- Datas: **2014/10/24 12:00** (UML); **2014/11/14 23:59** (intercalar); **2014/12/02 23:59** (final); **2014/12/02–2014/12/12** (teste prático).
- Os diagramas UML são entregues exclusivamente em papel (impressos ou manuscritos) na portaria do Tagus. Diagramas ilegíveis serão sumariamente ignorados.
- **Apenas se consideram para avaliação os projetos submetidos no Fénix.** As classes criadas de acordo com as especificações fornecidas devem ser empacotadas num arquivo de nome `proj.jar` (apenas os ficheiros `.java`). O ficheiro `proj.jar` deve ser entregue para avaliação através da ligação apresentada no Fénix. São possíveis múltiplas entregas até à data limite, mas será avaliada apenas a última versão.
- 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 presentes no Fénix no final do prazo têm classificação 0 (zero) (não são aceites outras formas de entrega).
- A avaliação do projeto pressupõe o compromisso de honra de que o trabalho correspondente foi realizado pelos alunos correspondentes ao grupo de avaliação. **Fraudes na execução do projeto terão como resultado a exclusão dos alunos implicados do processo de avaliação em 2014/2015**.

O objectivo do projecto é criar uma aplicação que simula e gere um sistema de ficheiros. A aplicação mantém um directório de trabalho, no contexto do qual são realizadas as operações do sistema de ficheiros. A execução da aplicação considera ainda o contexto de um utilizador, que pode mudar durante a execução da aplicação. Note-se que, embora exista apenas um sistema de ficheiros de cada vez, nada impede que venham a existir outros (ao longo do tempo), ou simultâneos (em evoluções da aplicação).

Neste texto, o tipo `fixo` indica um literal; o símbolo ␣ indica um espaço; e o tipo *itálico* indica uma parte variável.

1 Estrutura de um sistema de ficheiros

Existem dois tipos de entidades, ficheiros e directórios, que têm algumas características (o nome) e funcionalidades comuns (remover e renomear). Poderão existir outras características e/ou funcionalidades comuns. Cada entidade pertence a um utilizador.

1.1 Utilizadores

Cada utilizador tem um identificador único, o *username* (cadeia de caracteres), um nome e um directório principal.

Quando é criado o sistema de ficheiros, é também criado o utilizador `root`, que é o único que pode criar ou remover outros utilizadores. O nome deste utilizador é `Super User` e o seu directório principal é `/home/root` (criado na inicialização).

Os utilizadores podem ver todos os ficheiros/directórios. No entanto, apenas podem alterar os seus directórios/ficheiros próprios ou os de outros utilizadores que estejam em modo *público*. Um utilizador pode alterar a permissão associada a cada um dos seus directórios/ficheiros: escrita por todos (*público*), ou só pelo dono (*privado*). Alterar um directório corresponde a acrescentar/remover entradas a/de esse directório. O utilizador `root` pode alterar as permissões de qualquer directório/ficheiro, independentemente do dono e do modo de acesso.

Um utilizador pode alterar o dono do seus directórios/ficheiros. O utilizador `root` pode alterar o dono de qualquer entrada do sistema de ficheiros. A alteração exige sempre que o novo dono exista.

O directório principal de cada novo utilizador é criado durante o processo de registo (ficando na posse do novo utilizador). Caso já exista, é apagado primeiro e criado de seguida. O directório principal de cada utilizador do sistema de ficheiros é `/home/`*user*, onde *user* corresponde ao *username* do utilizador.

1.2 Ficheiros

Um ficheiro contém um conjunto de caracteres (o tamanho do ficheiro é igual ao número de caracteres que contém).

É possível obter os atributos de um ficheiro (tamanho, nome do ficheiro, *username* do dono), ver o seu conteúdo, acrescentar caracteres ao conteúdo do ficheiro e remover o ficheiro do directório onde se encontra.

1.3 Directórios

Um directório pode conter outros directórios/ficheiros. O nome de uma entrada no directório é único nesse directório. O tamanho de um directório depende do número de entradas que contém. Cada entrada tem um custo de 8 bytes.

Cada directório contém sempre dois directórios especiais: `.` (o próprio directório) e `..` (o directório pai). Estes dois directórios devem contar para o cálculo do tamanho de um directório. O directório pai do directório raiz (representado por `/`) é ele próprio.

Os directórios suportam as seguintes operações:

Devolver entradas	Devolve as várias entradas do directório em causa (incluindo `.` e `..`).
Remover	Remove o directório. Não é possível remover o directório raiz do sistema de ficheiros.
Criar (sub)directório	Cria um (sub)directório, com o nome indicado, no directório em causa.
Criar ficheiro	Cria um ficheiro vazio, com o nome indicado, no directório em causa.
Obter caminho absoluto	Permite obter o nome absoluto do directório, i.e., o caminho desde o directório raiz até ao directório em causa (os nomes dos vários directórios são separados pelo caracter `/`).

2 Interação com o Utilizador

Descreve-se abaixo 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 leitura e escrita **devem** realizar-se através das classes `pt.utl.ist.po.ui.Form` e `pt.utl.ist.po.ui.Display`. As mensagens são produzidas pelos métodos das bibliotecas de suporte (**po-uilib** e **poof-support**). As mensagens não podem ser usadas no núcleo da aplicação, elas devem ser utilizadas apenas no código responsável pela interacção com o utilizador. Desta forma,será possível reutilizar o código do núcleo da aplicação com outras concretizações da interface com o utilizador. Além disso, não podem ser definidas novas mensagens. Potenciais omissões devem ser esclarecidas antes de qualquer implementação.

As excepções usadas na interacção, excepto se indicado, são subclasses de `pt.utl.ist.po.ui.InvalidOperation` e devem ser lançadas durante a execução da aplicação (podendo ser lançadas pelo núcleo da aplicação caso isso facilite o código a produzir) para representar as situações de erro indicadas nesta secção. Estas excepções são tratadas automaticamente por `pt.utl.ist.po.ui.Menu`, e portanto não devem ser tratadas pelos vários comandos a desenvolver. Estas excepções encontem-se definidas no *package* `poof.textui.exception`. Outras excepções não devem substituir as fornecidas nos casos descritos. Nas secções dos comandos (abaixo), indicam-se os números correspondentes às excepções que podem ocorrer em cada um.

	Nome	**Condição de lançamento**
1	`EntryExistsException`	Tentativa de criar uma entrada com um nome não único num directório.
2	`EntryUnknownException`	Pedido o nome de uma entrada inexistente num directório.
3	`IsNotDirectoryException`	A entrada referenciada pelo nome não é um directório.
4	`IsNotFileException`	A entrada referenciada pelo nome não é um ficheiro.
5	`AccessDeniedException`	Tentativa de realização de operação para a qual o utilizador não tem permissão. Por exemplo, tentar escrever num ficheiro que apenas pode ser escrito pelo seu dono.
6	`IllegalRemovalException`	Tentativa de remover "." ou ".."
7	`UserExistsException`	Tentativa de criar um utilizador com um *username* não único.
8	`UserUnknownException`	Referência a utilizador que não existe no sistema de ficheiros.

2.1 Menu Principal

As acções do menu, listadas em `poof.textui.main.MenuEntry`, permitem gerir a salvaguarda do estado da aplicação (§2.1.1), definir o utilizador actual (**Login**) (§2.1.2), gerir o sistema de ficheiros (**Menu Shell**) (§2.2), gerir os utilizadores (**Menu Utilizador**) (§2.3). Os métodos para as mensagens de diálogo estão definidos em `poof.textui.main.Message`.

Inicialmente, a aplicação não tem nenhum sistema de ficheiros. Nesta situação, apenas são apresentadas as opções **Criar** e **Abrir**, pois as restantes necessitam um sistema de ficheiros activo. As opções irrelevantes nesta situação devem ser omitidas (em particular, **Menu Shell** deve ser omitido até ocorrer um início de sessão de um utilizador). Para concretizar este comportamento pode utilizar a funcionalidade da biblioteca **po-uilib** que permite esconder opções de um menu.

2.1.1 Salvaguarda do Estado da Aplicação

O conteúdo do sistema de ficheiros, assim como os utilizadores registados, bem como o utilizador activo, podem ser guardados em ficheiros (não confundir com os objectos manipulados pela própria aplicação), para posterior recuperação (serialização Java: `java.io.Serializable`). Na leitura e escrita do estado da aplicação, devem ser tratadas as excepções associadas. A funcionalidade é a seguinte:

Criar – Cria um sistema de ficheiros com um único utilizador (o utilizador `root`) e correspondente directório principal. Este utilizador fica automaticamente *logged in*. A aplicação não fica associada a qualquer ficheiro de salvaguarda.

Abrir – Carrega um estado anteriormente salvaguardado a partir de um ficheiro, associado-se a esse ficheiro. Pede-se o nome do ficheiro a abrir (`openFile()`): caso não exista, é apresentada a mensagem `fileNotFound()`. O utilizador anteriormente activo fica *logged in*.

Guardar – Guarda o estado actual no ficheiro associado. Se não existir associação, pede-se o nome do ficheiro a utilizar, ficando a ele associada. Esta interacção realiza-se através do método `newSaveAs()`. Não é executada nenhuma acção se não existirem alterações desde a última salvaguarda. O estado transitório (directório de trabalho) não é salvaguardado.

Apenas existe um sistema de ficheiros na aplicação (explicitamente criado ou aberto). Quando se abandona um sistema de ficheiros com modificações não guardadas (porque se cria ou abre outro), deve perguntar-se se se quer guardar a informação actual antes de a abandonar, através da mensagem `saveBeforeExit()` (a resposta é obtida invocando `readBoolean()`). A opção "Sair" **nunca** guarda o estado da aplicação, mesmo que existam alterações.

2.1.2 Login [login de um utilizador no sistema de ficheiros] [8]

Pede-se o *username* do utilizador que quer iniciar uma sessão (`usernameRequest()`) e caso o utilizador exista, então o directório de trabalho do sistema de ficheiros passa a ser o directório principal deste utilizador (utilizador activo).

2.2 Menu Shell

Este menu permite efectuar operações sobre o sistema de ficheiros activo, tendo em conta o utilizado activo na sessão. A lista completa é a seguinte: **Listar**, **Listar Elemento**, **Remover Elemento**, **Alterar Directório de Trabalho**, **Criar Ficheiro**, **Criar Directório**, **Mostrar Caminho Actual**, **Escrever Ficheiro**, **Ver Ficheiro**, **Alterar Permissão** e **Mudar Dono**. As secções abaixo descrevem pormenorizadamente as acções associadas a estas opções.

As etiquetas das opções deste menu estão definidas na classe `poof.textui.shell.MenuEntry`. Todos os métodos correspondentes às mensagens de diálogo para este menu estão definidos na classe `poof.textui.shell.Message`.

2.2.1 Listar [listar entradas do directório de trabalho] [não lança excepções]

As várias entradas do directório de trabalho são apresentadas ordenadas pelo nome. *permissão* é `w` (entradas públicas) ou `-` (caso contrário); *dono* é o *username* do dono da entrada.

A informação a apresentar para ficheiros e directórios é, respectivamente, a seguinte:

`-` ␣ *permissão* ␣ *dono* ␣ *tamanho* ␣ *nome*
`d` ␣ *permissão* ␣ *dono* ␣ *tamanho* ␣ *nome*

2.2.2 Listar Entrada [listar entrada indicada] [2]

É pedido o nome da entrada (`nameRequest()`). A apresentação é feita como definido em §2.2.1.

2.2.3 Remover Entrada [remover entrada do directório de trabalho] [2, 5, 6]

Pede-se o nome da entrada a remover (`nameRequest()`). Só é possível executar esta operação se tanto o directório como a entrada que contém forem do utilizador que dá o comando, ou forem públicos (ou combinações destas duas hipóteses).

2.2.4 Alterar directório de trabalho [mudar o directório de trabalho actual] [2, 3]

Pede-se o nome do novo directório de trabalho (`directoryRequest()`).

Os casos especiais de `.` e `..` devem ser tratados de forma correcta. Por exemplo, se o utilizador indicar `.` como nome do novo directório, o directório de trabalho deverá permanecer o mesmo.

2.2.5 Criar Ficheiro [criar ficheiro vazio no directório de trabalho] [1, 5]

Pede-se o nome do ficheiro a criar (`fileRequest()`).

2.2.6 Criar Directório [criar sub-directório vazio no directório de trabalho] [1, 5]

Pede-se o nome do directório a criar (`directoryRequest()`). Novos directórios contêm apenas as entradas especiais.

2.2.7 Mostrar Caminho Actual [mostar caminho absoluto do directório de trabalho] [não lança excepções]

Os vários directórios, entre o directório raiz e o directório de trabalho, são separados por `/`.

2.2.8 Escrever Ficheiro [adicionar linha de texto no final de ficheiro indicado pelo utilizador] [2, 4, 5]

Pede-se o nome do ficheiro a alterar (`fileRequest()`) e a linha de texto a inserir (`textRequest()`).

2.2.9 Ver Ficheiro [ver conteúdo do ficheiro indicado pelo utilizador] [2, 4]

É pedido o nome do ficheiro a apresentar (`fileRequest()`).

2.2.10 Alterar Permissão [alterar a permissão de escrita de uma entrada do directório de trabalho] [2, 5]

Primeiro, pede-se o nome da entrada (`nameRequest()`). De seguida, pergunta-se se a permissão a atribuir à entrada é pública (resposta afirmativa) ou privada (resposta negativa) (`writeMode()`).

2.2.11 Mudar dono [alterar dono de entrada do directório de trabalho] [2, 5, 8]

Pede-se o nome da entrada (`nameRequest()`) e o *username* do novo dono (`usernameRequest()`). A acção é realizada com sucesso se o utilizador activo tiver privilégios para a realizar.

2.3 Menu de Utilizador

Este menu realiza a gestão dos utilizadores do sistema de ficheiros activo. As opções disponíveis neste menu são: **Criar** e **Listar**. As secções abaixo descrevem pormenorizadamente as acções associadas a estas opções.

As etiquetas das opções deste menu estão definidas na classe `poof.textui.user.MenuEntry`. Todos os métodos correspondentes às mensagens de diálogo para este menu estão definidos na classe `poof.textui.user.Message`.

2.3.1 Criar [criar novo utilizador] [5, 7]

Pede-se o *username* (`usernameRequest()`) e o nome do utilizador (`nameRequest()`). Note-se que apenas o utilizador `root` pode criar novos utilizadores.

2.3.2 Listar [listar utilizadores] [não lança excepções]

A lista é ordenada por *username*, segundo o formato:

username : *nome* : *caminho absoluto do directório principal*

3 Inicialização por Ficheiro de Dados Textuais

Além das opções descritas em §2.1.1, é possível iniciar a aplicação com um ficheiro de texto especificado pela propriedade Java **import** (apresentada abaixo; este exemplo está no ficheiro `test.import`). Estes dados são apenas uma forma cómoda de inicialização e **nunca** são produzidos pela aplicação (nem mesmo para salvaguardar o estado para execuções futuras). Quando se especifica a propriedade, o sistema de ficheiros é povoado com as entidades do ficheiro indicado (uma por linha).

Assume-se que não há entradas mal-formadas. Sugere-se a utilização do método `String.split`, para dividir uma cadeia de caracteres em campos. Para utilizadores, indica-se o *username*, o nome e o directório principal; para directórios, indica-se o caminho, o dono e a permissão; para ficheiros, indica-se o caminho, o dono, a permissão e o conteúdo (apenas uma linha).

```
USER|obiwan|Obi-Wan␣Kenobi|/home/obiwan
USER|yoda|Master␣Yoda|/home/yoda
USER|vader|Darth␣Vader|/home/vader
DIRECTORY|/tmp|root|public
DIRECTORY|/home/obiwan/droids|obiwan|private
DIRECTORY|/home/vader/friends|vader|public
DIRECTORY|/ifs/jedi.org|root|private
DIRECTORY|/ifs/jedi.org/force|yoda|private
DIRECTORY|/ifs/jedi.org/dark_side|vader|private
FILE|/home/obiwan/droids/r2d2|obiwan|public|picture␣of␣r2d2
FILE|/ifs/jedi.org/dark_side/calendar|vader|private|10:00␣kill␣rebels;␣12:00␣lunch␣with␣emperor
```

A definição de um utilizador implica a criação do seu directório principal (privado e por ele detido). A indicação de um caminho implica a criação de todos os directórios no caminho (privados, por omissão, e detidos pelo mesmo dono que o directório pai). Tal como indicado em §1, o utilizador `root` e o seu directório principal já existem, pelo que não aparecem no ficheiro.

Execuções subsequentes já não utilizam o ficheiro de texto (passam a utilizar a serialização do Java).

4 Considerações sobre Flexibilidade e Eficiência

Devem ser possíveis extensões ou alterações de funcionalidade com impacto mínimo no código já produzido para o sistema de ficheiros. O objectivo é aumentar a flexibilidade da aplicação relativamente ao suporte de novas funções.

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. Note-se que é necessária a definição apropriada da variável `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 (`poof.textui.Shell.main`). A propriedade `import` é acedida através de `System.getProperty("import");` (ver manual da linguagem para mais informação). As outras propriedades são tratadas automaticamente pelo código de apoio.

```
java -Dimport=test.import -Din=test.in -Dout=test.outhyp poof.textui.Shell
```

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 das saídas 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 verificando alguns aspectos da sua funcionalidade.