TÉCNICO LISBOA

Projecto de Programação com Objectos

19 de Outubro de 2023

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:**Consultar a mesma secção**.
- 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 folha de cálculo. 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.

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: folha de cálculo, células, conteúdo, gamas e utilizador.

A aplicação a desenvolver é capaz de manipular números inteiros, cadeias de caracteres, referências entre células e funções sobre células. Além disso, permite também operações de corte, cópia e inserção (*cut/copy/paste*). A aplicação apenas gere uma folha de cada vez. O número de linhas e de colunas de uma folha é fixo, sendo definidos na altura da criação.

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

1.1 Células e Gama

Os endereços de uma célula (posições na folha: linha e coluna) têm início em 1 (um). Uma célula é definida com base na sua posição na folha: `CÉLULA ::= LINHA;COLUNA`.

Um intervalo de endereçamento (são sempre fechados) é definido entre duas células da mesma linha ou da mesma coluna: `INTERVALO ::= CÉLULA:CÉLULA`. Utiliza-se o termo *gama* para especificar indiscriminadamente uma célula única ou um intervalo de células. Exemplo:

Célula: `1;2` (linha 1, coluna 2); ou `23;4` (linha 23, coluna 4)

Intervalo: `1;2:1;20` (linha 1, entre as colunas 2 e 20); ou `23;4:57;4` (coluna 4, entre as linhas 23 e 57)

1.2 Conteúdo: Literais, Referências, Funções

Por omissão, as células estão vazias (sem conteúdo). Os conteúdos admissíveis são: literais (números inteiros e cadeias de caracteres), referências para células e funções. As referências indicam-se com o símbolo "=" seguido do endereço da célula referenciada. As funções indicam-se com o símbolo = ("igual"), o nome da função e a lista (possivelmente vazia) de argumentos entre parênteses (separados por vírgulas). Cada função tem um dado valor que corresponde à avalição da expressão indicada. O tipo do valor retornado depende da função. Estão pré-definidas as seguintes funções:

- Funções binárias sobre valores inteiros, cujos argumentos podem ser referências a células ou valores literais: `ADD` (adição), `SUB` (subtração), `MUL` (multiplicação), e `DIV` (divisão inteira). Todos os argumentos destas funções têm de ser passíveis de produzir valores inteiros.
- Funções aplicáveis a um intervalo de células, ou seja, com um único argumento que é uma gama de células:
  - `AVERAGE` (média de todos os valores do intervalo; a divisão é inteira) e `PRODUCT` (produto de todos os valores do intervalo). Todos os valores do intervalo têm de ser inteiros (não podem conter outros valores).
  - `CONCAT` que devolve uma cadeia de caracteres que representa a concatenação dos valores das várias células da gama indicada cujo valor é uma cadeia de caracteres. Células do intervalo que tenham valores que não são cadeias de caracteres são ignoradas pela função. Se o intervalo tiver apenas uma célula cujo valor é uma cadeia de caracteres, então esta função deve devolver esta cadeia de caracteres. Se não existir no intervalo nenhuma cadeia de caracteres, então deve produzir a cadeia vazia.
  - `COALESCE` (aceita um intervalo de células que podem ou não conter cadeias de caracteres e retorna o primeiro valor encontrado na gama indicada que seja uma cadeia de caracteres). Se o intervalo não contiver nenhuma cadeia de caracteres, produz a cadeia vazia.

Considera-se que não existem dependências circulares, nem directas, nem indirectas, entre uma função e as células referidas pelos seus argumentos. Funções com argumentos inválidos (e.g., uma gama com células vazias no caso da função `AVERAGE`), têm um valor inválido (apresentado como `#VALUE`).

Uma célula pode ter conteúdos de diferentes tipos:

- Inteiros (números com um sinal opcional no início): `-25`, `48`
- Cadeias de caracteres (começam sempre com plica): 'string
- Cadeias de caracteres (vazia): '(apenas uma plica)
- Referências: `=1;2`
- Funções: `=ADD(2;3,1)`, `=SUB(6;2,22;1)`, `=MUL(1,2)`, `=DIV(1,5;2)`, `=AVERAGE(1;2:1;19)`, `=PRODUCT(2;33:5;33)`, `=CONCAT(1;12:1;17)`, `=COALESCE(1;12:1;17)`

1.3 Cut Buffer

Cada folha de cálculo pode ter um *cut buffer*. O *cut buffer* é exclusivo da folha de cálculo a que está associado. Esta entidade permite as operações habituais (copiar, cortar, colar) entre células de uma folha. As operações têm a seguinte semântica:

Copiar O conteúdo da origem é copiado para o *cut buffer* e torna-se independente da fonte, i.e., alterações aos objectos originais não são propagadas para os que estão no *cut buffer*. O conteúdo anterior do *cut buffer* é destruído. O endereço da primeira célula do *cut buffer* começa sempre na 1ª linha e 1ª coluna, independentemente do endereço da primeira célula a copiar para o *cut buffer*. No entanto, o *cut buffer* deve preservar a orientação da gama a copiar, se é uma linha, as diferentes células do *cut buffer* devem estar todas na linha 1; se for uma coluna, as diferentes células do *cut buffer* devem estar todas na coluna 1.

Cortar Esta operação funciona como a operação de cópia, mas o conteúdo original é destruído.

Colar Esta operação insere o conteúdo do *cut buffer* numa gama da folha de destino. Se o *cut buffer* estiver vazio, não é realizada qualquer operação. Se a gama for uma única célula, todo o *cut buffer* deve ser inserido a partir da célula especificada, até ser atingido o limite da folha de cálculo. Caso contrário, se a dimensão do *cut buffer* for diferente da da gama de destino, não insere nenhum valor. O conteúdo do cut buffer não é alterado pela operação. Os objectos inseridos no destino são independentes dos que estão no *cut buffer*.

1.4 Utilizador

A aplicação deve suportar o conceito de utilizador. Cada utilizador tem um identificador único na aplicação (o seu nome) e detém um conjunto de folhas de cálculo. Devido a necessidades futuras ainda não completamente especificadas, deve existir uma relação bidireccional de muitos para muitos entre utilizadores e folhas de cálculo. A aplicação tem sempre um utilizador activo. Cada vez que se cria uma folha de cálculo na aplicação, ela deve ser associada ao utilizador activo. O utilizador com o nome **root** deve existir sempre. Por omissão, o utilizador activo é o utilizador com o nome **root**.

2 Requisitos de Desenho

Devem ser possíveis extensões ou alterações de funcionalidade com impacto mínimo no código produzido: em particular, deve ser simples definir novas funções e novas pesquisas sobre células. O objectivo é aumentar a flexibilidade da aplicação relativamente ao suporte de novas funções.

A estrutura de armazenamento dos conteúdos da folha deve ser flexível, por forma a permitir a representação eficiente de folhas de diferentes dimensões. Assim, deve ser possível usar diferentes estratégias de representação do conteúdo, sem impacto no resto da aplicação. O objectivo deste requisito é optimizar o espaço ocupado em memória para guardar o conteúdo de uma folha de cálculo. Não é necessário concretizar para todas as situações, bastando ser suficientemente flexível para permitir novas implementações.

A determinação do valor de uma função que envolve uma gama pode ter um custo relativamente alto, caso envolva um número grande células. Caso nenhuma das células envolvidas tenha sido alterada desde a última vez que o valor da função foi determinado, então calcular o valor de novo é um desperdício. Pretende-se obter uma solução que optimize o número de vezes que uma função que envolva uma gama necessita de calcular o seu valor. Este requisito deve ser considerado apenas depois de tudo estar implementado e apenas vai ser tido em conta na entrega final do projecto.

A solução encontrada para optimizar o número de vezes que uma função tem que ser calculada deve ser extensível por forma a permitir que seja possível especificar outros tipos de funcionalidade que devem ocorrer quando o valor de uma célula muda de valor. Por exemplo, deveria ser possível concretizar o seguinte requisito sem que isso implicasse modificar as classes existentes (não é necessário implementar este exemplo): enviar um email a um dado utilizador cada vez que a célula em causa muda de valor. A sua solução deve ser flexível por forma a poder suportar diferentes tipos de entidades interessadas em modificações de uma célula sem que isso implique alterar o código já realizado.

A aplicação já deve estar preparada para suportar vários utilizadores, devendo já ter o código (ao nível da camada de domínio apenas) para a inserção de novos utilizadores.

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). No início da aplicação pode ser carregada uma base de dados textual com conceitos pré-definidos. Inicialmente, a aplicação apenas tem informação sobre as entidades que foram carregadas no arranque da aplicação.

Deve ser possível inserir, apagar e mostrar conteúdos de células da folha activa e realizar as operações de copiar, colar e cortar. Deve ser possível pesquisar o conteúdo das células sob diferentes aspectos: (i) valores resultantes de avaliação; (ii) nomes de funções.

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 `xxl-skeleton.jar` presente na secção *Projecto* da página da cadeira. O esqueleto inclui a classe `xxl.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 concretizadas no package `xxl.app` e nos seus sub-packages (por exemplo, `xxl.app.main`).

O esqueleto inclui ainda algumas classes do domínio da aplicação (presentes no package `xxl.core`) e um conjunto de excepções a utilizar durante o desenvolvimento da aplicação. Será ainda necessário concretizar novas entidades do domínio da aplicação e finalizar as que já estão parcialmente fornecidas por forma a 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 `xxl.app.exception`. O package `xxl.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.

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.

Por forma a garantir uma independência entre as várias camadas da aplicação não deve haver código de interacção com o utilizador no código respeitante ao domínio da aplicação (*xxl.core*). Desta forma, será possível reutilizar o código do 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 código respeitante ao domínio 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.

De um modo geral, sempre que no contexto de uma operação com o utilizador aconteça alguma excepção, então a operação não deve ter qualquer efeito no estado da aplicação, excepto se indicado em contrário na operação em causa.

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) `xxl.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 `xxl.app.exception`. apenas devem ser utilizadas no código de interacção com o utilizador pela razão indicada anteriormente.

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.

Para o pedido de uma gama, utiliza-se a mensagem `Message.address()`. A excepção xxl.app.exception.InvalidCellRangeException é lançada se forem especificados endereços inválidos (excedem o limite da folha de cálculo ou representam gamas de células que não são intervalos verticais ou horizontais).

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: **Criar**, **Abrir**, **Guardar**, **Menu de Edição**, e **Menu de Consultas**. As várias mensagens de diálogo utilizadas nos vários comandos deste menu estão definidas nos vários métodos da classe `xxl.app.main.Message`.

Alguns dos comandos que vão concretizar as funcionalidades deste menu já estão parcialmente concretizados nas várias classes do package `xxl.app.main`: `DoNew`, `DoOpen`, e `DoSave`. Os restantes comandos deste menu, `DoOpenMenuEdit` e `DoOpenMenuSearch` já estão completamente concretizados.

3.1.1 Salvaguarda do estado actual

Inicialmente, a aplicação está vazia (não tem nenhuma folha) ou tem apenas informação sobre as entidades que foram carregados no arranque via ficheiro textual (ver 4). Na situação em que a aplicação começa sem nenhuma folha, apenas são apresentadas as opções **Criar** e **Abrir**, pois as restantes necessitam da existência de uma folha. As opções irrelevantes nesta situação devem ser omitidas. 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:

Criar – Cria uma nova folha vazia: pedem-se as dimensões da nova folha, devendo ser utilizadas as mensagens `Message.lines()` e `Message.columns()`. Esta folha não fica associada a nenhum ficheiro (é anónima).

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 relevante 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. Não é executada nenhuma acção se não existirem alterações desde a última salvaguarda.

Apenas existe uma folha activa na aplicação. Quando se abandona uma folha com modificações não guardadas (porque se cria ou abre outra), deve perguntar-se se se quer guardar a informação actual antes de a abandonar, através de `Message.saveBeforeExit()` (a resposta é obtida invocando o método `readBoolean()` da *framework* de interacção com o utilizador).

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

Além das operações de manipulação de ficheiros, existem ainda no menu principal as seguintes opções:

Menu de Edição – Abre o menu de edição.

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

Estas opções só estão disponíveis quando existir uma folha activa válida.

3.2 Menu de Edição

O menu de edição permite visualizar e alterar o conteúdo das células da folha activa. A lista completa é a seguinte: **Visualizar**, **Inserir**, **Copiar**, **Apagar**, **Cortar**, **Colar** e **Visualizar cut buffer**. Os comandos deste menu já estão parcialmente concretizados nas classes do package `xxl.app.edit`: `DoShow`, `DoInsert`, `DoCopy`, `DoDelete`. `DoCut`, `DoPaste`, e `DoShowCutBuffer`. Todos os métodos correspondentes às mensagens de diálogo para este menu estão definidos na classe `Message` deste package.

3.2.1 Visualizar

Permite visualizar a gama especificada, de acordo com os formatos abaixo. Se o conteúdo não for um literal, deve ser apresentado o seu valor e a sua expressão. Não deve ser apresentado qualquer conteúdo para células vazias. A apresentação de uma célula segue o seguinte formato:

```
Célula vazia – linha;coluna|
Célula com um valor literal – linha;coluna|valor
Célula com uma referência ou função – linha;coluna|valor=expressão
```

onde linha e coluna representam o endereço da célula a apresentar e o valor no último caso representa o valor da final da célula depois de efectuadas todas as avaliações envolvidas. Na visualização de uma gama, deve-se apresentar cada uma das células da gama utilizando o formato anterior. No caso de um intervalo vertical, deve-se seguir o seguinte formato:

```
linha1;coluna|conteúdo
linha2;coluna|conteúdo
linha3;coluna|conteúdo
```

No caso de um intervalo horizontal devem-se apresentar as células por ordem ascendente de coluna:

```
linha;coluna1|conteúdo
linha;coluna2|conteúdo
linha;coluna3|conteúdo
```

Em ambos os casos, o conteúdo da célula deve ser apresentado utilizando o formato descrito para o caso de uma célula.

3.2.2 Inserir

Pede-se a gama e o conteúdo a inserir, através da mensagem `Message.contents()`, ao utilizador. O conteúdo indicado deve ser inserido em todas as células da gama. Este comando lança a a excepção `xxl.app.exception.UnknownFunctionException` caso se insira um nome de função inexistente. O tratamento de outros erros que possam acontecer durante o processamento do conteúdo está especificado pelo que qualquer solução adotada é válida.

3.2.3 Copiar

Pede-se a gama ao utilizador e depois copia o conteúdo da gama indicada para o *cut buffer*.

3.2.4 Apagar

Pede-se a gama ao utilizador e depois apaga o conteúdo da gama indicada.

3.2.5 Cortar

Pede-se a gama ao utilizador e depois *corta* o conteúdo da gama indicada.

3.2.6 Colar

Pede-se a gama ao utilizador e depois insere o conteúdo do cut buffer na gama especificada.

3.2.7 Visualizar cut buffer

Permite visualizar o conteúdo do cut buffer. O formato de apresentação é o descrito em 3.2.1.

3.3 Menu de Consultas

O menu de consultas permite efectuar pesquisas sobre as células da folha activa, produzindo listas de células. Sempre que for feita uma consulta e nenhuma entidade satisfizer as condições associadas ao pedido, nada deve ser apresentado. A lista completa das consultas é a seguinte: **Procurar Valor** e **Procurar Função**. Estes comandos já estão parcialmente concretizaos nas classes do package `xxl.app.lookup`: **DoShowValues** e **DoShowFunctions**.

3.3.1 Procurar Valor

Pede-se o valor a procurar (`Message.searchValue()`) e apresentam-se as células cujo valor (avaliado no caso de o conteúdo da célula ser uma referência ou uma função) é igual ao termo de pesquisa. As células que obedecem ao critério de procura devem ser apresentadas por ordem ascendente de linha e coluna e utilizando o formato especificado em 3.2.1. Assim, considere-se o seguinte exemplo que descreve o conteúdo de uma folha com três linhas e três colunas:

```
1;1|4
1;2|1
1;3|5
2;1|5=ADD(1;1,1;2)
2;2|
2;3|
```

Neste exemplo, uma pesquisa pelo valor 5 apresentaria o seguinte resultado:

```
1;3|5
2;1|5=ADD(1;1,1;2)
```

3.3.2 Procurar Função

Pede-se o segmento de texto a procurar num nome de função (`Message.searchFunction()`) e apresentam-se todas as células cujo conteúdo é uma função cujo nome contém o segmento de texto indicado. As células que verificam o critério de pesquisa são apresentadas de forma ascendente, por nome de função, linha e coluna (por esta ordem), utilizando o formato descrito em 3.2.1.

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 da folha a carregar no estado inicial da aplicação. Quando se especifica esta propriedade, é criada uma folha activa anónima que representa o conteúdo do ficheiro indicado.

As duas primeiras linhas deste ficheiro de texto definem o número de linhas e de colunas da folha de cálculo. As restantes linhas contêm sempre o formato `linha;coluna|conteúdo` (o campo conteúdo está descrito na secção 1.2). No processamento destes dados, assume-se que não existem entradas mal-formadas. Assume-se que células não explicitamente referidas estão vazias (que também podem ser explicitamente definidas). A codificação dos ficheiros a ler é garantidamente `UTF-8`. Sugere-se a utilização do método `String.split` para o processamento preliminar destas linhas. De seguida apresenta-se um exemplo de conteúdo do ficheiro de texto:

```
linhas=4
colunas=3
3;3|=ADD(3;1,3;2)
4;1|
1;1|5
1;2|49
2;1|25
2;2|43
2;3|=ADD(2;2,5)
3;1|10
3;2|=1;1
1;3|=ADD(2,5)
4;3|=AVERAGE(1;3:3;3)
4;2|
```

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

5 Execução dos Programas e Testes Automáticos

Cada teste é constituído por dois ou três ficheiros de texto:

- um com extensão `.in` e que representa o utilizador, ou seja, vai conter os dados de entrada a serem processados pelo programa;
- um com extensão `.out` e que representa o resultado esperado da execução do programa para o teste em causa;
- um com extensão `.import` e que representa o estado inicial do sistema. Este ficheiro é opcional e pode não fazer parte de um teste.

Dado um teste constituído pelos ficheiros `test.import`, `test.in` e `test.out`, é possível verificar automaticamente o resultado correcto do programa. 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 (`xxl.app.App.main`). As propriedades são tratadas automaticamente pelo código de apoio.

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

Caso o teste não tenha o ficheiro de *.import*, então a forma de executar o teste é semelhante ao anterior, não especificando a propriedade *import*. 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 `xxl.app`. Outras dessas classes são de uso obrigatório e têm de ser alteradas: os diferentes comandos presentes em subpackages de `xxl.app` e algumas classes do domínio da aplicação já parcialmente concretizadas em `xxl.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).