TÉCNICO LISBOA

Projecto de Programação com Objectos

17 de Setembro de 2024

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 de gestão de um hotel veterinário. Um hotel veterinário tem que gerir diferentes tipos de entidade, incluindo tratadores, veterinários e habitats com árvores. 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 4 e 5. 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 (por exemplo, hotel veterinário, espécie, animal, funcionário).

Um hotel veterinário tem vários habitats onde os animais colocados à guarda do hotel são depois colocados. Cada animal pertence a uma espécie. O hotel tem dois tipos de funcionários: tratadores e veterinários. Os tratadores são responsáveis por distribuir a comida pelos animais e pela limpeza dos habitats. Os veterinários, têm a responsabilidade de zelar pela saúde dos animais. Quando um animal recebe uma vacina não apropriada à sua espécie, por causa de um erro veterinário, por exemplo, fica com a saúde afectada até ao fim da sua vida. Cada animal tem que ser colocado num dos habitats do hotel. O habitat pode ainda ter árvores. Deve ser possível calcular o grau de satisfação dos animais e dos funcionários.

Em todos os campos *identificador* ou *nome* das várias entidades do domínio, excepto se indicado o contrário, as diferenças entre maiúsculas e minúsculas são irrelevantes.

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

1.1 Espécies

Cada animal pertence a uma dada espécie. As espécies são identificadas por uma chave única (número inteiro sequencial incrementado automaticamente e com início em 0). Cada espécie tem ainda um nome (cadeia de caracteres única, i.e., não existem duas espécies distintas com o mesmo nome) e conhece todos os animais que pertencem à espécie.

1.2 Animais

Os animais são identificados por uma chave única (cadeia de caracteres arbitrária definida na altura da criação). Cada animal tem um nome (cadeia de caracteres não única) e sabe a sua espécie. O animal mantém ainda informação sobre o seu estado de saúde.

É possível calcular o nível de satisfação de um animal. Este valor depende de vários factores. A satisfação dos animais é afectada positivamente quando existem mais animais da mesma espécie no mesmo habitat e afectada negativamente quando existem

animais de outras espécies. O espaço médio por animal do habitat e o próprio habitat também influenciam a satisfação de todos os animais que lá vivem. Assim, a satisfação de um animal a presente no habitat h é calculada pela seguinte fórmula:

$$\textit{satisfação}(a) = 20 + 3 * \textit{espécieIgual}(a, h) - 2 * \textit{espécieDiferente}(a, h) + \frac{\textit{área}(h)}{\textit{população}(h)} + \textit{adequação}(a, h) \quad (1)$$

onde `espécieIgual(a, h)` representa o número de animais da mesma espécie de a presentes em h (sem contar com a), `espécieDiferente(a, h)` é o número de animais presentes em h de espécies diferentes da de a, `população(h)` representa o número de animais no habitat h e `adequação(a, h)` representa a influência do habitat no animal tendo em conta a espécie do animal. A adequação é 20 se a influência do habitat for positiva, -20 se for negativa e 0 se a influência for neutra (valor por omissão).

1.3 Habitats

Os habitats são identificados por uma chave única (cadeia de caracteres definida na altura do registo). Cada habitat tem um nome (cadeia de caracteres não única) e uma área (número inteiro). Os habitats conhecem ainda as árvores e os animais presentes no habitat.

Os habitats podem ser mais (ou menos) adequados para determinadas espécies. A adequação tem impacto no grau de satisfação dos animais. Por omissão, um habitat tem um impacto neutro para cada espécie. Mas um habitat pode ser mais adequado a um determinado conjunto de espécies e/ou menos adequado a outro subconjunto de espécies.

1.4 Árvores

As árvores são identificadas por uma chave única (cadeia de caracteres arbitrária definida na altura da criação). Cada árvore tem ainda nome (cadeia de caracteres não única) e idade em anos (número inteiro). As árvores podem ser de folha caduca ou perene e são caracterizadas pela dificuldade base de limpeza (número inteiro) que induzem no habitat onde estão implantadas. Quando uma árvore é criada, fica na estação do ano em que a aplicação estiver no momento, i.e., todas as árvores estão sincronizadas nesse aspecto. A estação inicial da aplicação é a Primavera.

A vida das árvores segue o ciclo definido pelas estações do ano. Assim, por exemplo, as árvores de folha caduca perdem as folhas principalmente durante o Outono, ficando sem folhas no Inverno. As árvores de folha perene perdem algumas folhas durante todas as estações, mas mais durante o Inverno. O esforço de limpeza de uma árvore é determinado pelo produto de três factores. O primeiro corresponde à dificuldade base de limpeza da árvore (designado como *dificuldade de limpeza*). O segundo, designado como *esforço sazonal*, depende da estação do ano e do tipo de árvore, tal como indicado na tabela seguinte:

Estação / Árvore	Inverno	Primavera	Verão	Outono
Folha caduca	0	1	2	5
Folha perene	2	1	1	1

O terceiro corresponde a um factor que cresce logaritmicamente com a idade da árvore: $log(idade + 1)$ (logaritmo natural). Combinando os vários factores, o esforço de limpeza de uma árvore a é calculado pela seguinte fórmula:

$$\textit{esforço\_limpeza}(a) = \textit{dificuldade\_limpeza}(a) * \textit{esforço\_sazonal}(a) * log(\textit{idade}(a) + 1) \quad (2)$$

As árvores envelhecem com as estações, i.e., a idade aumenta em 1 unidade (1 ano) a cada 4 estações (note-se que o incremento não é necessariamente simultâneo em todas as árvores).

1.5 Funcionários

Os funcionários são identificados por uma chave única (cadeia de caracteres arbitrária definida na altura da criação). Os funcionários têm um nome (cadeia de caracteres não única). Existem dois tipos de funcionários: tratadores e veterinários. Cada tratador está atribuído a um conjunto de habitats (zero ou mais). Cada veterinário tem informação relativa às espécies que sabe vacinar(zero ou mais). Cada funcionário tem ainda um nível de satisfação. O cálculo da satisfação depende do tipo de funcionário.

1.5.1 Satisfação dos Veterinários

A satisfação de um veterinário v é afectada positivamente pela existência de outros veterinários que sabem vacinar as várias espécies que v sabe vacinar e negativamente pelo número de animais que estão sob a sua responsabilidade (ou seja, que v sabe vacinar). Assim, a satisfação de um veterinário v é calculada pela seguinte fórmula:

$$\textit{satisfação}(v) = 20 - \sum_{e \in \textit{espécies}(v)} \frac{\textit{população}(e)}{\textit{n\_veterinários}(e)} \tag{3}$$

onde `espécies(v)` representa o conjunto de espécies que v sabe tratar, `população(e)` representa o número de animais da espécie e e `n_veterinários(e)` representa o número de veterinários com a responsabilidade de vacinar a espécie v.

1.5.2 Satisfação dos Tratadores

A satisfação dos tratadores é afectada negativamente pelo trabalho que lhes é atribuído. O trabalho depende dos vários habitats atribuídos ao tratador. A satisfação para um tratador t é calculada pela fórmulas:

$$\textit{satisfação}(t) = 300 - \sum_{h \in habitats(t)} \frac{\textit{trabalho\_no\_habitat}(h)}{\textit{número\_de\_tratadores\_do\_habitat}(h)} \tag{4}$$

$$\textit{trabalho\_no\_habitat}(h) = \textit{área}(h) + 3 * \textit{população}(h) + \sum_{a \in \textit{árvores}(h)} \textit{esforço\_limpeza}(a) \tag{5}$$

onde `habitats(t)` representa o conjunto de habitats sob responsabilidade de t e `árvores(h)` representa o conjunto de árvores presente no habitat h.

1.6 Vacinas

As vacinas são identificadas por uma chave única (cadeia de caracteres arbitrária definida na altura da criação). Cada vacina tem um nome (cadeia de caracteres; não única) e sabe as espécies a que pode ser aplicada. Cada vacina mantém ainda um registo de todas as vezes que foi administrada. O registo deve ainda identificar o veterinário que aplicou a vacina e o animal ao qual a vacina foi aplicada. O registo deve preservar a ordem de vacinação.

1.6.1 Problemas associados à vacinação

O estado de saúde de um animal é o histórico de eventos (inicialmente vazio) relacionados com a vacinação do animal, entre os quais estão os danos causados por más vacinações. O valor do dano de uma vacinação correcta (a espécie do animal está incluída nas espécies para as quais a vacina é apropriada) é igual a 0. O valor do dano causado pela má administração de uma vacina v a um animal a depende do número de letras distintas entre o nome da espécie de a (`espécie(a)`) e o nome das espécies a que se destinava v (`espécies(v)`):

$$\textit{dano}(v, a) = \textit{MAX}_{e \in \textit{espécies}(v)}(tamanho\_nomes(\textit{espécie}(a), e) - \textit{caracteres\_comuns}(\textit{espécie}(a), r)) \tag{6}$$

em que

$$tamanho\_nomes(\textit{espécie1}, \textit{espécie2}) = max(tamanho(\textit{espécie1}), tamanho(\textit{espécie2})) \tag{7}$$

Considera-se que os nomes `aab` e `ba` têm apenas 1 carácter distinto, sendo a ordem irrelevante. Por exemplo, quando uma *ave* recebe uma vacina destinada a um *mamífero*, o dano resultante da má aplicação da vacina é igual a 6, sendo calculado por:

$max(tamanho(\textit{ave}), tamanho(\textit{mamífero})) - caracteres\_comuns(ave, \textit{mamífero}) = 8 - 2 = 6$

A seguinte tabela descreve os resultados associados ao dano causado pelas vacinações:

Dano	Termo a adicionar na apresentação do estado de saúde
0 (mesma espécie)	NORMAL
0 (espécies diferentes)	CONFUSÃO
1 a 4	ACIDENTE
5 ou mais	ERRO

Por exemplo, um animal que tenha recebido 2 vacinas com um dano entre 1 e 4 (ACIDENTE), três vacinas com um dano igual ou superior a 5 (ERRO) e uma vacina correctamente aplicada, apresenta o seguinte historial de saúde (usa-se a vírgula como separador): ACIDENTE,ACIDENTE,ERRO,ERRO,ERRO,NORMAL (i.e., concatenação ordenada por ordem de ocorrência).

1.7 Serialização

É possível guardar e recuperar o estado actual da aplicação, preservando toda a informação relevante do domínio da aplicação.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).

2 Requisitos de Desenho

O sistema a desenvolver deve seguir o princípio de desenho *aberto-fechado* por forma a aumentar a extensibilidade do seu código. Por exemplo, deve ser possível adicionar novos tipos de funcionários com um impacto mínimo no código já existente do domínio da aplicação.

Aplicação de um padrão de desenho para melhorar a legibilidade do código relacionado com o conceito *Árvore* e permitir ao mesmo tempo que se possam adicionar novos eventos relacionados com as estações do ano sem que isso implique alterar o código relacionado com *Árvore*. O padrão deve ainda poder suportar novas funcionalidades dependentes da estação actual (por exemplo, qual a cor das folhas de uma árvore, que vai depender do tipo de árvore e da estação actual) sem comprometer a legibilidade da solução.

Aplicação de um padrão de desenho que permita novas políticas de cálculo da satisfação dos funcionários sem impacto no código existente. A solução concretizada deve permitir alterar o cálculo da satisfação de um funcionário em tempo de execução. Quando é criado um funcionário, o cálculo da satisfação segue a fórmula descrita neste enunciado.

Apesar de haver na especificação actual apenas um hotel, o código não deve assumir que não possam vir a existir mais instâncias em simultâneo.

Do ponto de vista do desenho do domínio da aplicação a realizar para a primeira entrega do projecto não é necessário ter em conta estes dois requisitos. Estes dois requisitos apenas precisam de ser considerados na entrega final do projecto.

3 Arquitectura da Aplicação

A aplicação a desenvolver deve seguir uma arquitectura em camadas: as entidades relacionadas com interacção com o utilizador estão concretizadas na camada de apresentação (package `hva.app`) e as entidades relacionadas com as entidades do domínio estão concretizadas na camada de domínio (package `hva.core`). 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 (concretizada na camada de domínio). Desta forma, será possível reutilizar o código do domínio da aplicação com outras concretizações da interface com o utilizador sem que isso implique alterar o código da camada de domínio. 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 apresentação. 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.

A interacção com o utilizador deve ser realizada utilizando a *framework* de interacção com o utilizador **po-uilib** (disponível na secção **Projecto** da página da disciplina. A interacção com o utilizador suportada por esta framework é realizada através de menus, em que cada menu tem um determinado conjunto de opções. Cada menu é concretizado por uma subclasse da classe `pt.tecnico.po.ui.Menu`) (fornecida pela framework), onde se deve indicar qual o conjunto de opções do menu. Cada opção de um menu (designado como *comando*) é concretizada por uma subclasse de `pt.tecnico.uilib.menus.Command` (também fornecida pela framework) que tem a responsabilidade de suportar a funcionalidade correspondente à opção em causa. Todos os menus suportam 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 a apresentar ao utilizador são produzidas pelos métodos de interfaces já definidas na camada de apresentação do esqueleto da aplicação fornecido (ver secção 3.1).

Podem acontecer situações de erro durante a execução de um comando. Quando isto ocorre, o comando não deve ter qualquer efeito no estado da aplicação (excepto se indicado em contrário na operação em causa) e o comando deve instanciar uma excepção (pertencente a uma subclasse de `pt.tecnico.uilib.menus.CommandException` correspondente ao erro que aconteceu) e lançá-la de seguida. Estas excepções serão depois tratadas automaticamente pela classe `Menu`.

3.1 Esqueleto da Aplicação

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 `hva-skeleton.jar` presente na secção *Projecto* da página da cadeira. O esqueleto inclui

a classe `hva.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 4). Alguns dos comandos já estão completamente finalizados, outros (a grande maioria) estão parcialmente concretizados e têm de ser alterados por forma a suportarem a funcionalidade descrita neste enunciado. Cada menu e respectivos comandos estão definidos num subpackage de `hva.app`. Por exemplo, o menu principal e respectivos comandos estão definidos em `hva.app.main`. Cada menu está completamente concretizado e a grande maioria dos comandos apenas tem o esqueleto da classe e é necessário acrescentar código por forma a que o comando tenha a funcionalidade necessária.

Cada subpackage relacionado com um menu contém ainda as interfaces `Message`, `Prompt` (definem as mensagens a utilizar na interacção com o utilizador) e `Label` (define as etiquetas do menu e dos vários comandos). **Não** podem ser definidas novas mensagens. Potenciais omissões devem ser esclarecidas com o corpo docente antes de qualquer concretização.

O esqueleto inclui ainda algumas classes do domínio da aplicação (presentes no package `hva.core`) parcialmente concretizadas. 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.

Finalmente, as excepções a lançar nos vários comandos já estão definidas no package `hva.app.exception` do esqueleto da aplicação. Estas excepções **não** podem ser alteradas e apenas devem ser utilizadas no código de interacção com o utilizador pela razão indicada anteriormente. O package `hva.core.exception` contém algumas excepções a utilizar na camada do domínio, mas podem (e devem) ser acrescentadas novas excepções para representar outras situações anómalas que possam acontecer no contexto das entidades do domínio.

4 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).

A aplicação permite gerir a informação sobre as várias 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, a aplicação está vazia, mas pode ser carregada uma base de dados textual com conceitos pré-definidos. Neste caso, a aplicação começará com um estado referente às entidades que foram carregadas no arranque da aplicação. Deve ser possível registar, visualizar, e operar sobre as várias entidades do domínio. Deve ser possível efectuar pesquisas sujeitas a vários critérios e sobre as diferentes entidades geridas pela aplicação.

A apresentação de listas (e.g., animais, funcionários, etc.) faz-se por ordem crescente da respectiva chave, excepto em caso de indicação em contrário. A ordem das chaves alfanuméricas deve ser lexicográfica (UTF-8), **não havendo** distinção entre maiúsculas e minúsculas.

A descrição da funcionalidade de cada comando indica quais as excepções que podem ser lançadas pelo comando. Para cada interacção com o utilizador que exista no contexto de um comando em que é necessário utilizar uma mensagem específica é indicado qual é o método da interface `Prompt` ou `Message` que é responsável por devolver a mensagem em questão.

No contexto de um comando pode ser pedido ao utilizador um ou mais identificadores de objectos do domínio necessários para a realização da funcionalidade do comando em causa. A mensagem a utilizar para pedir o identificador e a excepção a lançar caso o identificador não corresponda a um objecto conhecido (excepto no processo de registo ou caso indicação em contrário) depende do tipo de entidade em questão:

Entidade	Pedido a apresentar	Excepção a lançar se desconhecido
Animal	`hva.app.animal.Prompt.animalKey()`	`UnknownAnimalKeyException`
Espécie	`hva.app.animal.Prompt.speciesKey()`	`UnknownSpeciesKeyException`
Funcionário	`hva.app.employee.Prompt.employeeKey()`	`UnknownEmployeeKeyException`
Habitat	`hva.app.habitat.Prompt.habitatKey()`	`UnknownHabitatKeyException`
Árvore	`hva.app.habitat.Prompt.treeKey()`	`UnknownTreeKeyException`
Vacina	`hva.app.vaccine.Prompt.vaccineKey()`	`UnknownVaccineKeyException`

Note-se que estas excepções não devem ser utilizadas no código do domínio da aplicação como explicado em 3. Alguns casos particulares podem usar pedidos específicos não apresentados nesta tabela.

4.1 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**, **Avançar ciclo biológico das árvores**, **Ver Estado de Satisfação**, **Gestão de Animais**, **Gestão de Funcionários**, **Gestão de Vacinas**, **Gestão de Habitats** e **Consultas**. Todos os métodos correspondentes às mensagens de diálogo para este menu estão definidos em `hva.app.main.Prompt` e `hva.app.main.Message`.

Estes comandos estão concretizadas (nalguns casos parcialmente) nas seguintes classes do package `hva.app.main`: `DoNewFile`, `DoOpenFile`, `DoSaveFile`, `DoAdvanceSeason`, `DoShowGlobalSatisfaction`, `DoOpenAnimalsMenu`, `DoOpenEmployeesMenu`, `DoOpenHabitatsMenu`, `DoOpenVaccinesMenu` e `DoOpenLookupsMenu`.

4.1.1 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 5). 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 aplicação vazia: a aplicação 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 `Prompt.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 `Prompt.newSaveAs()`), ficando a aplicação associada a este ficheiro. Não é executada nenhuma acção se não existirem alterações desde a última salvaguarda.

Apenas existe um hotel na aplicação em cada momento. Quando se abandona um hotel com modificações não guardadas (porque se cria ou abre outro), deve perguntar-se se se quer guardar a informação actual antes de o abandonar, através de `Prompt.saveBeforeExit()` (a resposta é obtida invocando o método `readBoolean()` ou `Form.confirm()` 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.

4.1.2 Avançar Estação do Ano

Esta opção permite avançar a estação do ano da aplicação, afectando todas as árvores em todos os habitats. O comando deve apresentar na saída o código para a estação actual: 0 para Primavera, 1 para Verão, 2 para Outono e 3 para Inverno

4.1.3 Ver Estado de Satisfação Global

O sistema apresenta, de imediato, o somatório do nível de satisfação de todas as entidades registadas no hotel: animais e funcionários. Note-se que o valor a apresentar é o que resulta do arredondamento ao inteiro mais próximo (via `Math.round`).

4.1.4 Gestão e consulta de dados da aplicação

Existem ainda no menu principal várias opções para abrir um segundo menu com as operações disponíveis sobre determinadas entidades do sistema (animais, funcionários, habitats e vacinas). Finalmente, é também disponibilizado uma opção para abrir o menu de consultas (pesquisas).

4.2 Menu de Gestão de Animais

Este menu permite visualizar e alterar informação sobre animais. A lista completa de opções é a seguinte: **Visualizar todos os animais**, **Registar um novo animal**, **Transferir um animal para um habitat** e **Calcular Satisfação** de um animal. Estas opções já estão parcialmente concretizados nas classes do package `hva.app.animal`: `DoShowAllAnimals`, `DoRegisterAnimal`, `DoTransferToHabitat`, e `DoShowSatisfactionOfAnimal`.

4.2.1 Visualizar todos os animais

Esta opção permite visualizar todos os animais. O formato de apresentação de cada animal é o seguinte:

```
ANIMAL|idAnimal|nomeAnimal|idEspécie|historialDeSaúde|idHabitat
```

O historial de saúde é apresentado como descrito na secção 1.6.1, i.e., uma sequência de eventos separados por vírgulas. Se o animal nunca foi vacinado (com ou sem sucesso), o historial de saúde deve ser apresentado como *VOID*:

```
ANIMAL|idAnimal|nomeAnimal|idEspécie|VOID|idHabitat
```

4.2.2 Registar um novo animal

Esta opção registar um novo animal. O sistema pede o identificador que ficará associado ao animal. De seguida pede o nome do animal (Prompt.animalName()), o identificador da espécie e o identificador do habitat. Quando o identificador da espécie não existe, deve pedir-se o nome da espécie (`Prompt.speciesName()`) (cadeia de caracteres) e registá-la com o identificador indicado. Se o identificador do animal já existir, lança a excepção `DuplicateAnimalKeyException`, não se processando o registo.

4.2.3 Transferir um animal para um habitat

Esta opção transfere um animal de um habitat para outro. O sistema pede o identificador do animal e o habitat de destino. Em caso de erro, o habitat não é alterado.

4.2.4 Calcular Satisfação

Esta opção apresentar a satisfação de um animal. O sistema pede o identificador do animal, sendo apresentada o valor da sua satisfação ((arredondado ao inteiro mais próximo via `Math.round`).

4.3 Menu de Gestão de Funcionários

visualizar todos os funcionários; (ii) registar um novo funcionário; (iii) atribuir uma nova responsabilidade a um funcionário; (iv) retirar uma responsabilidade a um funcionário; (v) calcular a satisfação de um funcionário.

Este menu permite visualizar e alterar informação sobre funcionários. A lista completa de opções é a seguinte: **Visualizar todos os funcionários**, **Registar um novo funcionário**, **Atribuir uma nova responsabilidade a um funcionário**, **Retirar uma responsabilidade a um funcionário**, **Calcular a satisfação de um funcionário**. Estas opções já estão parcialmente concretizados nas classes do package `hva.app. employee`: `DoShowAllEmployees`, `DoRegisterEmployee`, `DoAddResponsibility`, `DoRemoveResponsibility` e `DoShowSatisfactionOfEmployee`.

4.3.1 Visualizar todos os funcionários

Esta opção permite visualizar todos os funcionários. O formato de apresentação (quando o funcionário tem pelo menos uma responsabilidade) é o seguinte:

```
tipo|id|nome|idResponsabilidades
```

Os valores para o campo *tipo* são `VET` ou `TRT`. Os valores para o campo *idResponsabilidades* são os identificadores, separados por vírgulas, dos habitats que um tratador pode limpar ou das espécies de animais que um veterinário pode tratar. Se o funcionário não tiver responsabilidades atribuídas, o formato de apresentação é o seguinte:

```
tipo|id|nome
```

4.3.2 Registar um novo funcionário

Esta opção permite registar um novo funcionário. O sistema pede o identificador do novo funcionário, o seu nome (`Prompt.employeeName`
(cadeia de caracteres) e o tipo do funcionário (`Prompt.employeeType()`). A resposta a este campo deve ser VET (é registado um veterinário) ou TRT (é registado um tratador). Se a resposta não corresponder a nenhum destes dois valores, a pergunta referente ao tipo é repetida até se obter uma resposta válida. Quando um funcionário é registado fica sem qualquer responsabilidade. Se já existir um funcionário com o mesmo identificador, deve ser lançada a excepção `DuplicateEmployeeKeyException`, não se realizando qualquer acção.

4.3.3 Atribuir uma nova responsabilidade a um funcionário

Esta opção atribui uma nova responsabilidade a um funcionário. O sistema pede o identificador do funcionário e o identificador da nova responsabilidade (`Prompt.responsibilityKey()`), que deverá corresponder ao identificador de uma espécie, se for um veterinário, ou ao identificador de um habitat, se for um tratador. Se o funcionário já tinha essa responsabilidade, não é executada nenhuma acção. Se a responsabilidade não existir, então deve ser lançada a excepção `NoResponsibilityException`.

4.3.4 Retirar uma responsabilidade a um funcionário

Esta opção retira uma responsabilidade a um funcionário. O sistema pede o identificador do funcionário e o identificador da responsabilidade a retirar (`Prompt.responsibilityKey()`), a qual deverá corresponder ao identificador de uma espécie, se for um veterinário, ou ao identificador de um habitat, se for um tratador. Se a responsabilidade não estava atribuída ao funcionário ou não existe é lançada a excepção `NoResponsabilityException`.

4.3.5 Calcular satisfação de um funcionário

Esta opção apresenta a satisfação de um funcionário. O sistema pede o identificador do funcionário, sendo apresentado o valor da sua satisfação (arredondado ao inteiro mais próximo via `Math.round`).

4.4 Menu de Gestão de Habitats

Este menu permite visualizar e alterar informação sobre funcionários. A lista completa de opções é a seguinte: **Visualizar todos os habitats**, **Registar um novo habitat**, **Alterar área**, **Alterar influência de um habitat sobre uma espécie**, **Plantar uma nova árvore num habitat** e **Visualizar todas as árvores de um habitat**. Estas opções já estão parcialmente concretizados nas classes do package `hva.app.habitat`: `DoShowAllHabitats`, `DoRegisterHabitat`, `DoChangeHabitatArea`, `DoChangeHabitatInfluence`, `DoAddTreeToHabitat` e `DoShowAllTreesInHabitat`.

4.4.1 Visualizar todos os habitats

Esta opção permite visualizar todos os habitats. O formato de apresentação para cada habitat é o seguinte:

```
HABITAT|idHabitat|nome|área|númeroÁrvores
```

Esta linha inicial pode ser seguida de uma ou mais linhas, cada uma com a descrição de cada árvore pertencente ao habitat usando o seguinte formato:

```
ÁRVORE|idÁrvore|nomeÁrvore|idadeÁrvore|dificuldadeBaseLimpeza|tipoÁrvore|cicloBiológico
```

O valor relativo ao campo *dificuldadeBaseLimpeza* corresponde ao valor da dificuldade base de limpeza da árvore. Os valores possíveis para o campo *tipoÁrvore* são PERENE e CADUCA. No caso do campo *cicloBiológico*, os valores possíveis são os seguintes:

Estação / Tipo de Árvore	Inverno	Primavera	Verão	Outono
Folha caduca	SEMFOLHAS	GERARFOLHAS	COMFOLHAS	LARGARFOLHAS
Folha perene	LARGARFOLHAS	GERARFOLHAS	COMFOLHAS	COMFOLHAS

4.4.2 Registar um novo habitat

Esta opção regista um novo habitat. O sistema pede o identificador, o nome (`Prompt.habitatName()`) (cadeia de caracteres) e a área do habitat (`Prompt.habitatArea()`) (número inteiro), sendo registado o novo habitat. Caso já exista um habitat com o mesmo identificador, então deve ser lançada a excepção `DuplicateHabitatKeyException` e o comando não deve ter qualquer acção.

4.4.3 Alterar a área de um habitat

Esta opção altera a área de um habitat. O sistema pede o identificador do habitat assim como a nova área desse habitat (Prompt.habitatArea()) (número inteiro).

4.4.4 Alterar influência de um habitat sobre uma espécie

Esta opção altera a influência de um habitat sobre uma espécie. O sistema pede o identificador do habitat, o identificador de uma espécie e se a influência do habitat (`Prompt.habitatInfluence()`) sobre a espécie indicada é positiva (resposta: POS), negativa (resposta: NEG), ou neutra (resposta: NEU). Se a resposta a este campo não corresponder a nenhum dos três valores, esta última pergunta é repetida até se obter uma resposta válida.

4.4.5 Plantar uma nova árvore num habitat

Esta opção introduz uma nova árvore num habitat. O sistema pede o identificador do habitat, o identificador da nova árvore a plantar, o nome (`Prompt.treeName()`) (cadeia de caracteres), a idade da árvore (`Prompt.treeAge()`), a dificuldade base de limpeza associada à árvore (`Prompt.treeDifficulty()`) (número inteiro) e o tipo de árvore (`Prompt.treeType()`, PER para árvore de folha perene; CAD para árvores de folha caduca), registado-se a nova árvore. Se a resposta relativa ao tipo não corresponder a nenhum dos dois valores, esta última pergunta é repetida até se obter uma resposta válida. Se já existir uma árvore com o mesmo identificador, é lançada a excepção `DuplicateTreeKeyException`, não se realizando qualquer acção.

Caso seja criada uma árvore, então esta opção deve apresentá-la utilizando o formato descrito em 4.4.1.

4.4.6 Visualizar todas as árvores de um habitat

Esta opção permite visualizar todas as árvores de um determinado habitat. O sistema pede o identificador do habitat e apresenta as árvores que nele existem.

Esta opção apresenta todas as árvores de todos os habitats. O formato de apresentação de cada árvore é como descrito em 4.4.1.

4.5 Menu de Gestão de Vacinas

Este menu permite visualizar e alterar informação sobre vacinas e vacinações. A lista completa de opções é a seguinte: **Visualizar todas as vacinas**, **Registar uma nova vacina**, **Vacinar um animal** e **Listar o histórico de vacinações**. Estas opções já estão parcialmente concretizados nas classes do package `hva.app.vaccine`: `DoShowVaccine`, `DoRegisterVaccine`, `DoVaccinateAnimal` e `DoShowVaccinations`.

4.5.1 Visualizar todas as vacinas

Esta opção apresenta todas as vacinas. O formato de apresentação de cada vacina, uma por linha, é o seguinte:

```
VACINA|idVacina|nomeVacina|númeroAplicações|espécies
```

O campo *espécies* contém os identificadores, separados por vírgulas, das espécies de animais que podem receber a vacina.

4.5.2 Registar uma nova vacina

Esta opção regista uma nova vacina. O sistema pede o identificador da vacina, o nome (`Prompt.vaccineName()`) (cadeia de caracteres) da vacina e os identificadores das espécies que podem receber esta vacina (`Prompt.listOfSpeciesKeys()`) (lista de identificadores separados por vírgulas. Os espaços brancos são irrelevantes nesta resposta). De seguida regista a nova vacina. Deve ser lançada a excepção `DuplicateVaccineKeyException`, se existir uma vacina com o mesmo identificador, não se realizando qualquer acção. Se algum dos identificadores de espécies indicados for desconhecido, deve ser lançada a excepção `UnknownSpeciesKeyException`, não sendo registada a nova vacina.

4.5.3 Vacinar um animal

Esta opção regista a vacinação de um animal. O sistema pede o identificador da vacina, o identificador do veterinário (`Prompt.veterinari`
e o identificador do animal a vacinar. Se o identificador não for de um veterinário, deve ser lançada a excepção `UnknownVeterinarianKey`
Se o veterinário não tiver permissão para ministrar a vacina deve ser lançada a excepção `VeterinarianNotAuthorizedException`
e o comando não tem qualquer efeito. Se a vacina não for adequada ao animal, deve ser apresentada uma mensagem de aviso (`Message.wrongVaccine()`). Note-se que, neste caso, a vacinação aconteceu e os danos para o animal devem ter sido registados.

4.5.4 Listar o histórico de vacinações

Esta opção lista o histórico de vacinações ordenado pela ordem em que as vacinas foram aplicadas. O sistema lista todas as vacinas aplicadas, usando o seguinte formato:

```
REGISTO-VACINA|idVacina|idVeterinário|idEspécie
```

4.6 Menu de Consultas

O menu de consultas permite efectuar pesquisas sobre as entidades do domínio e suas relações. A lista completa de opções é a seguinte: **Consultar animais de um habitat**, **Consultar vacinas de um animal**, **Consultar actos médicos de um veterinário** e **Consultar vacinas dadas com incúria**. Estas opções já estão parcialmente concretizados nas classes do package `hva.app.search`: `DoShowAnimalsInHabitat`, `DoShowMedicalActsOnAnimal`, `DoShowMedicalActsByVeterinarian` e `DoShowWrongVaccinations`.

4.6.1 Consultar animais de um habitat

Esta opção apresenta a informação relativa aos animais de um habitat. O sistema pede o identificador do habitat e de seguida apresenta os animais que residem nesse habitat. O formato de apresentação é como em 4.2.1.

4.6.2 Consultar vacinas de um animal

Esta opção apresenta a informação relativa às vacinações de um animal. O sistema pede o identificador do animal pretendido e de seguida apresenta as vacinações do animal (por ordem de aplicação). O formato de apresentação é como indicado em 4.5.4.

4.6.3 Consultar actos médicos de um veterinário

Esta opção apresenta a informação relativa a vacinações feitas por um veterinário. O sistema pede o identificador de um funcionário. Se o identificador não for de um veterinário, deve ser lançada a excepção `UnknownVeterinarianKeyException`. Caso contrário, apresenta todas as vacinas por ele realizadas (por ordem de aplicação). O formato de apresentação é como indicado em 4.5.4.

4.6.4 Consultar vacinas dadas com incúria

Esta opção apresenta a informação relativa a vacinações erróneas. O sistema apresenta todas as ocorrências que provocaram problemas de saúde aos animais (por ordem de aplicação). O formato de apresentação é como indicado em 4.5.4.

5 Leitura de Dados a Partir de Ficheiros Textuais

Por omissão, quando a aplicação começa, não contém nenhuma informação e está na estação do ano *Primavera*. No entanto, além das opções de manipulação de ficheiros descritas no menu principal, é possível iniciar exteriormente a aplicação com um ficheiro de texto especificado pela propriedade Java com o nome `import`. Quando se especifica esta propriedade, a aplicação é povoada com os objectos correspondentes ao conteúdo do ficheiro indicado. No processamento destes dados, assume-se que não existem entradas mal-formadas e assume-se que os identificadores referidos numa entrada já foram previamente descritos. Sugere-se a utilização do método `String.split()` para dividir uma cadeia de caracteres em campos. Cada entrada do ficheiro é composto por um conjunto de campos separados pelo caracter `'|'`. O número de campos dependo do tipo da entrada (indicado no primeiro campo). De seguida apresenta-se o formato para cada tipo de entrada:

```
ESPÉCIE|id|nome
ÁRVORE|id|nome|idade|dificuldade|tipo
HABITAT|id|nome|área|idÁrvore1,...,idÁrvoreN
ANIMAL|id|nome|idEspécie|idHabitat
TRATADOR|id|nome|idHabitat1,...,idHabitatN
VETERINÁRIO|id|nome|idEspécie1,...,idEspécieN
VACINA|id|nome|idEspécie1,...,idEspécieN
```

Todos os campos referentes a uma árvore, animal ou espécie são obrigatórios. O último campo referente a um habitat (corresponde às árvores implantadas no habitat), tratador (corresponde aos habitats sob responsabilidade do tratador), vacina e veterinário (corresponde em ambos os casos a uma lista de espécies) é opcional e só é apresentado caso a lista indicada não seja vazia. Se a lista for vazia, então não se apresenta o último campo, nem o separador.

Para os outros tipos, o último campo é opcional: para um habitat, o último campo corresponde às árvores implantadas no habitat, parar a um tratador, o último campo corresponde aos habitats sob responsabilidade do tratador e para uma vacina ou veterinário, o último campo corresponde à lista de espécies associada à vacina ou ao veterinário. De seguida apresenta-se um exemplo de conteúdo do ficheiro de texto:

```
ESPÉCIE|C10|ave
ESPÉCIE|C8|mamífero
ESPÉCIE|C1|peixe
ÁRVORE|T1|Pinheiro␣1|5|20|PERENE
ÁRVORE|T4|Pinheiro␣4|60|20|PERENE
ÁRVORE|T2|Oliveira|1200|10|PERENE
ÁRVORE|T6|Figueira|5|10|CADUCA
ÁRVORE|T3|Plátano|300|20|CADUCA
ÁRVORE|P1|Plátano|100|20|CADUCA
ÁRVORE|F30|Figueira|50|20|CADUCA
HABITAT|AR1|Aldeia␣dos␣Macacos|20|T3,P1,F30,T2
HABITAT|AR2|Floresta␣tropical|30|T1,T4,T6
HABITAT|AR3|Deserto|3
ANIMAL|A1|Alex|C10|AR1
ANIMAL|A2|Nemo|C1|AR2
ANIMAL|AA|Bobi|C8|AR1
ANIMAL|MA|Chita|C8|AR2
TRATADOR|W2SO4|John␣Figueiredo|AR2,AR3
TRATADOR|W2O|Rohit␣Figueiredo|AR1
TRATADOR|W24|Rohit␣Figueiredo
VETERINÁRIO|V2|Jorge␣Figueiredo|C10,C8,C1
VETERINÁRIO|V4|Filomena␣Figueiredo
VETERINÁRIO|VR|Abdul␣Figueiredo|C8
VACINA|V3|Tétano|C8
VACINA|V200|Parasitas␣intestinais|C10,C8,C1
VACINA|V42172|Tétano|C10
VACINA|Vexperimental|Gripe␣A1|C8
```

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

6 Execução dos Programas e Testes Automáticos

É possível verificar automaticamente o resultado correcto do programa. 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 (`hva.app.App.main`). As propriedades são tratadas automaticamente pelo código de apoio.

```
java -cp -Dimport=test.import -Din=test.in -Dout=test.outhyp hva.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.