UAlg FCT
UNIVERSIDADE DO ALGARVE
FACULDADE DE CIÊNCIAS E TECNOLOGIA

Algoritmos e Estruturas de Dados

2025/26

Projecto 1 – Tipos Abstratos de Informação

Choose Your Own Adventure

(v1.0)

Introdução

Este projeto tem como objetivo que os alunos - mais experientes na linguagem C - se familiarizem com a linguagem de programação Java, com o ambiente de desenvolvimento, e que aprendam a criar e utilizar Tipos Abstratos de Informação em Java. Este projeto é bastante fácil, e foi baseado no Trabalho Dirigido[1] n.º 3 da cadeira de Laboratório de Programação do Semestre passado. Assim os alunos já estarão familiarizados com o tipo de dados as funcionalidades pretendidas.

Imagem gerada com IA (Image Creator do Microsoft Designer)

[1] Gostaria de agradecer ao prof. Simão Melo de Sousa por ter fornecido o TD3.

Tal como em Laboratório de Programação iremos desenvolver, de forma incremental, um jogo do tipo “Choose Your Own Adventure” na linguagem Java, mas desta vez usando classes em vez de estruturas com ponteiros.

Um jogo desta natureza tem como elemento central um conjunto de cenas interligadas, em que cada cena é um capítulo na aventura do jogo. Quando impressa no ecrã, uma cena tem o seguinte formato:

```
[5]
A sua aventura — tal como a sua sanidade — chegou a um cruzamento.
À direita, a floresta das Sombras Eternas sussurra segredos que não
deviam ser ouvidos. Em frente, o Cume das Vozes Perdidas aguarda,
com ecos estridentes que talvez sejam seus. À esquerda, o Deserto
dos Ossos Quebrados estala a sua esperança de qualquer salvação.
Inspire o medo. Expire a esperança. Faça a sua escolha.
+ 7. Seguir para a floresta das Sombras Eternas.
+ 9. seguir para o cume das Vozes Perdidas.
+ 3. seguir para o deserto dos Ossos Quebrados.
```

Neste exemplo, representamos uma cena comum numa história a decorrer. Esta cena tem um identificador (é a cena nº5), uma descrição textual, o número de opções que a cena oferece (aqui 3). A descrição de cada uma das 3 opções possíveis acompanhada da identificação da cena para qual a aventura segue em cada opção. A título de ilustração, se neste ponto do jogo escolher seguir para a floresta das Sombras Eternas, terá de introduzir o valor 7 na entrada standard para escolher esta opção e a aventura seguirá para a cena nº 7.

Existem ainda dois tipos de cenas de fim de história, onde o jogador ganha ou perde o jogo. Como por exemplo:

```
[4]
Num último espasmo de coragem — ou loucura — lança-se para a escuridão
gotejante da gruta, fugindo no pânico cego da criatura cujo bafo pútrido já
aquece a sua nuca como um presságio do fim. Mas é inútil. Ela conhece todos os
seus recantos como se fosse a luz negra da sua alma. A gruta é dela. Sempre foi.
E agora, também é sua. Garras surgem das trevas e, num único movimento
bruto, rasgam-no em dois, num terror estupefacto, sem um único grito. Só
vísceras.
Morreu.
<FAILED>
```

Problema A: Implementação do TAI Option (3 valores)

Implemente o Tipo Abstracto de Informação ***Option***, que representa a informação de uma única opção de uma cena, tendo em conta a seguinte especificação:

TAI *Option* – Representa a informação sobre uma opção de uma cena
`Option(int id, String description)`
Construtor que cria uma opção com a informação recebida: id refere-se ao identificador da opção e description é uma cadeia de caracteres que representa a descrição textual da opção Caso algum dos argumentos recebidos não seja válido (id não pode ser negativo, e description não pode ser nula ou vazia) deve ser lançada uma exceção do tipo IllegalArgumentException.
`Int getId()`
Retorna o identificador da opcao.
`String getDescription()`
Retorna uma cadeia de caracteres com a descrição da opção
`boolean equals(Object o)`
Retorna *true* se o objeto recebido for igual à opção sobre a qual o método é invocado. Duas opções são iguais se tiverem o mesmo identificador e se as descrições forem iguais.
`String toString()`
Retorna uma string com a informação completa de uma opção, usando o seguinte formato + <id>. <description> Por exemplo + 7. Seguir para a floresta das Sombras Eternas.
`Static Option parseOption(String s)`
Método estático que dada uma string representando uma opção, extrai um objeto do tipo Option a partir dessa descrição. O formato a usar para a leitura da descrição é igual ao formato da sua escrita (ver método toString()). Caso não seja possível extrair o id da opção (exemplo não é um número válido) deve ser lançada uma NumberFormatException.

Implemente o tipo *Option* no ficheiro *Option.java*. O ficheiro não pode declarar nenhuma package[2]. Submeta **apenas o ficheiro Option.java** no Problema A.

Problema B: Implementação do Tipo Scene (5 Valores)

Nesta tarefa iremos implementar o conceito de cena, usando para isso a classe Scene. Recomendamos a utilização da estrutura de dados ArrayList<Option> para guardar a lista de opções para a cena. O ArrayList em Java é um vetor redimensionável (ou seja, tal como o *dynvec* implementado por vocês na cadeira de LP, aumenta o seu tamanho de forma automática quando está demasiado cheio, e precisamos de inserir um novo elemento).

Implemente as seguintes funções:

TAI *Scene* – Classe que guarda informação sobre uma cena.
`Scene(int id, String description, Type type)`
Construtor que cria uma cena, onde id corresponde ao identificador da cena, description a uma cadeia de caracteres que contem a descrição da cena, type é um tipo enumerado em Java com os valores (NORMAL,WON,FAILED).

[2] Uma classe definida num ficheiro sem declaração de package diz-se pertencer à package *default*, também conhecida como *unnamed*, e é normalmente colocada na pasta *src* do projeto Java.

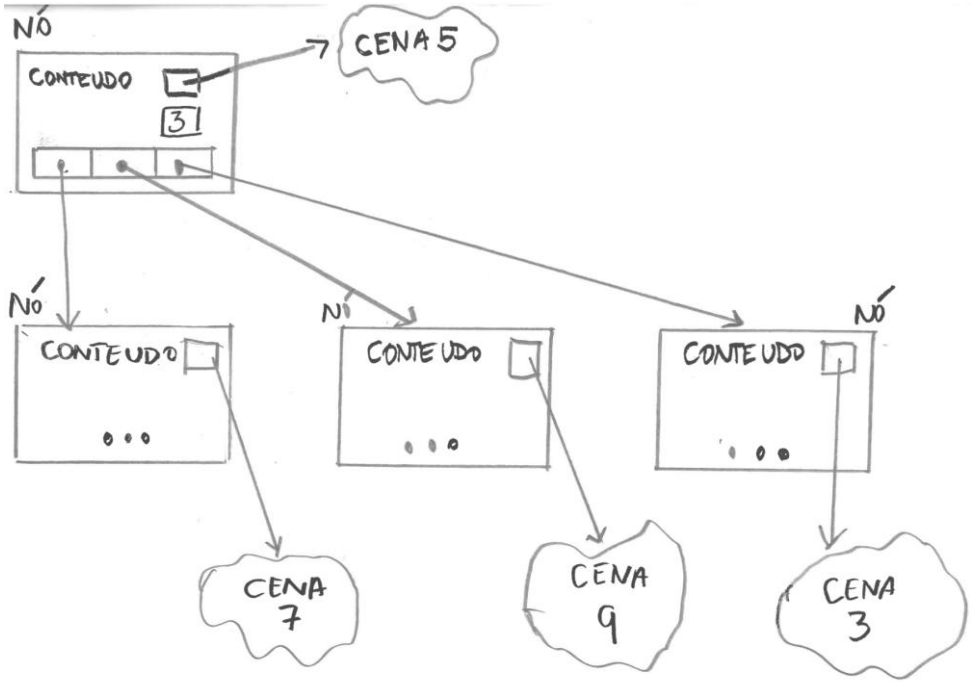
boolean addOption(Option o)
Adiciona uma nova opção à cena, colocando-a no fim da lista. No entanto não podem existir duas opções com o mesmo identificador para a mesma cena. Caso seja adicionada uma opção que entre em conflito com alguma das opções já existentes, a opção não deve ser adicionada. O método retorna *true* se a opção tiver sido adicionada com sucesso, e *false* caso contrário.
int getId()
Este método retorna o identificador da cena.
Type getType()
Retorna o enumerado correspondente ao tipo da cena.
String getDescription()
Retorna uma cadeia de caracteres coma descrição textual da cena.
Iterable<Option> getOptions()
Retorna uma coleção iterável de todas as opções da cena.
String toString()
Retorna uma cadeia de caracteres com a representação textual do conteúdo da cena, que deverá ter o seguinte formato, no caso de uma cena do tipo normal: [<id>] <descrição> <opção 1> … <opção n> No caso de uma cena do tipo WON, ou FAILED, a lista de opções deverá ser substituída por <WON> ou <FAILED> respetivamente.
static Scene parseScene(BufferedReader br)
Método estático que dado um bufferedReader, já aberto, irá tentar ler e criar uma cena a partir do leitor recebido. O formato para leitura de uma cena do tipo NORMAL é a seguinte. [<id>] <<< <descrição> >>> *** <opção 1> … <opção n> *** A 1ª linha corresponde ao id da cena (entre "[" e "]"). A 2.ª linha corresponde à sequência "<<<" que identifica o início da descrição. As linhas seguintes correspondem à descrição da cena (que pode ter múltiplas linhas). Uma linha com a sequência ">>>" marca o fim da descrição de uma cena normal Uma linha com a sequência "***" marca o inicio das opções De seguida podemos ter qualquer número de linhas, com uma opção por linha Uma linha com a sequência "***" marca o fim das opções No caso de uma cena do tipo WON, a descrição termina com uma linha com a sequência ">>>WON". Não existem opções. No caso de uma cena do tipo FAILED, a descrição termina com uma linha com a sequência ">>>FAILED". Não existem opções. O método não pode fechar o BufferedReader recebido. Este método lança (e não necessita de as apanhar) exceções do tipo IOException e NumberFormatException.

Caso exista algum problema a ser lido um número inteiro (por exemplo na leitura do id da cena) deve ser lançada uma NumberFormatException, e caso haja algum problema com a leitura do buffered reader deve ser lançada uma IOException.

Submeta o ficheiro ***Scene.java*** no Problema B. O ficheiro não pode declarar nenhuma package. Não é necessária a submissão de mais nenhum ficheiro.

Problema c: Implementação Tipo Node (12 Valores)

Nesta tarefa iremos implementar a classe que representa a informação completa de uma história, e que é também responsável pela funcionalidade de jogar a aventura. Tal como no TD3 de LP, aqui representar a história na forma de um grafo, em que o conteúdo dos nós do grafo correspondem a Objetos do tipo Cena, como se pode ver no diagrama seguinte.



O conceito genérico de Nó que iremos usar está implementado na classe genérica Node<T> fornecida, e não precisa de ser implementado (é uma boa ideia consultarmos o ficheiro para percebermos como funciona). Mas temos de usar obrigatoriamente a classe Node<T> para implementar a classe Story.

Iremos implementar os seguintes métodos (recomendamos esta ordem de implementação):

TAI *Story* – Classe que guarda informação sobre uma história na forma de um grafo, e que implementa um sistema interativo para o jogador jogar a história.
`Story(String title, Node<Scene>[] nodes)`
Este construtor constroi uma nova história, com o título recebido, a partir do vetor de nós recebido. Podemos assumir que os nós recebidos já vão ter todo o grafo criado pela história. O nó inicial é considerado o nó no índice 0 do vetor. Casos algum dos argumentos seja inválido, deverá ser lançada uma IllegalArgumentException. O título recebido não pode ser nulo ou vazio, e o vetor de nós não pode ter tamanho 0.
`String getTitle()`
Retorna o título da história.

Node<Scene> *getCurrent()*
Retorna o nó do grafo onde a história se encontra atualmente. Por exemplo, no inicio da história retornamos o nó correspondente ao nó inicial, mas se transitarmos para o nó/cena n.º 7, o nó a ser retornado deverá ser o correspondente ao da cena n.º 7.
void *reset()*
Faz com que a história volte a ter como estado atual o estado inicial da história.
static Story *parseStory(String fileName)*
Método estático responsável por ler e criar uma história a partir de um ficheiro. A 1ª linha do ficheiro contem o título da história. A 2ª linha do ficheiro contem um inteiro n que indica quantas cenas a história tem. De seguida o ficheiro contém n cenas, com o formato especificado no método parseScene do problema B. Quando se está a fazer leitura de objetos a partir de texto, muita coisa pode correr mal, e iria dar algum trabalho (aborrecido) capturar todos os diferentes tipos de erros que podem ocorrer. Aqui iremos apenas detetar 2 tipos de erros. Caso exista um erro ao abrir o ficheiro, ou a ler do ficheiro (IOException) deverá ser escrita a mensagem FILE_ERROR[3] para o standard output. Caso exista algum problema na leitura de um número, deverá ser escrita a mensagem NUMBER_FORMAT_ERROR. Nestes casos deve ser retornado null. Dica: este método corresponde aos passos 1) e 2) do exercício 4 do TD3. Sugestão de implementação: 1) Abrir o ficheiro com um buffered reader 2) Ler o título da história, e o número de cenas 3) Ler todas as cenas, e guardá-las num vetor de Node<Scene>. Estes nós ainda não têm os vizinhos definidos. 4) Percorrer todos os nós guardados no vetor, e adicionar os vizinhos correspondentes a cada nó (com base nas opções da cena). 5) Criar uma Story com base no vetor de nós.
void *nonInteractivePlay()*
Método que irá simular a execução de uma história, mas sem se preocupar com a impressão das cenas no ecrã, e sem se preocupar muito com escolhas inválidas. Apenas escreve o resultado final. Corresponde ao passo 3) do exercício 4 do TD3. A partir do standard input lemos uma sequência de escolhas, em que cada linha tem uma única escolha. Desenrolamos a história a partir do seu estado inicial, usando as escolhas lidas do standard input. Cada execução tem 3 desfechos possíveis: a) O jogo termina com a vitória. Deverá escrever numa única linha “WON”. b) O jogo termina com uma derrota. Deverá escrever numa única linha “FAILED”. c) Não foram dadas opções suficientes para terminar o jogo, ou foi usada uma opção inválida no decorrer do jogo (por exemplo uma opção inexistente). Deverá escrever numa única linha “WAITING”. Não se esqueça de libertar toda a memória alocada no fim do jogo. Estou a brincar, a JVM liberta automaticamente toda a memória quando já não for necessária, por isso não nos temos de preocupar com a libertação de memória.

[3] As mensagens de erro concretas estão já definidas como constantes no ficheiro Story.java

boolean saveStoryState(String fileName)
Este método recebe um nome de um ficheiro e guarda o estado da história num ficheiro. Para isto basta guardar na 1.ª linha o título da história, e na 2ª linha o identificador da cena/nó atual. Caso haja um erro de output, deve ser impressa no ecrã a mensagem de FILE_ERROR e retornado false. Caso o processo decorra normalmente deve ser impressa no ecrã a mensagem de SAVE_SUCCESS, e deverá ser retornado true.
boolean loadStoryState(String fileName)
Dado um nome de um ficheiro, este método tenta carregar o ficheiro, e mudar a cena atual para a cena indicada no ficheiro. Retorna um booleano que indica se o processo correu bem ou não. A 1ª linha do ficheiro deverá indicar o título da história, e a 2.ª linha deverá ter um inteiro que representa o id da cena a carregar. Devem ser feitas algumas verificações. Caso exista um erro de leitura (IOException) deve ser impressa a mensagem de FILE_ERROR e retornado false. Caso exista um erro no processo de leitura do número da cena deve ser impressa a mensagem de NUMBER_FORMAT_ERROR e retornado false. Caso a leitura tenha corrido bem, mas o título da história no ficheiro não corresponde à nossa história, não podemos carregar. Deve ser impressa a mensagem INVALID_LOAD_STORY_NAME e retornado false. Caso o título da história esteja ok, mas o id da cena guardada no ficheiro não seja válido (por exemplo é de uma cena que não existe na história atual), também não podemos carregar e devemos imprimir a mensagem INVALID_LOAD_SCENE e retornar false. Caso contrário, está tudo ok, atualizamos a cena atual para a cena indicada, imprimimos a mensagem LOAD_SUCCESS e retornamos true.
void interactivePlay()
Método que irá correr uma versão mais completa e interativa do método nonInteractivePlay, em que vamos imprimindo a cena atual, e vamos pedindo ao utilizador (através do standard input) para escolher a opção a usar. Começamos sempre por imprimir o título da história. Imprime a cena atual, **e caso seja uma cena normal**, escreve a prompt "Select your option: " no ecrã, e lê a escolha do utilizador a partir do standard input. Se o utilizador escrever algum dos seguintes comandos: RESET PRINT SAVE nome_ficheiro LOAD nome_ficheiro Devemos executar a operação apropriada. O comando RESET volta a colocar como a cena atual a cena inicial da história. O comando PRINT volta a imprimir a cena atual. O comando SAVE grava a cena atual para um ficheiro. O comando LOAD lê a cena do ficheiro, e muda a cena atual para a cena indicada (caso seja possível). Se o utilizador escrever algo que não seja, nem um dos comandos acima, nem um número, deve ser escrita a mensagem INVALID_NUMBER, e voltamos a escrever a prompt e a ler novo número. Se o utilizador escrever um número, mas o número não corresponder a uma opção válida da cena atual, deve ser escrita a mensagem INVALID_OPTION, e voltamos a escrever a prompt e a ler novo número. Caso contrário, o número escolhido é válido e transitamos para o nó da cena correspondente. Se chegamos a uma cena final (WON ou FAILED), apenas imprimimos a cena e saímos. Deixamos aqui um pequeno exemplo de interação deste método:

```
Floresta e Deserto
[0]
A sua aventura — tal como a sua sanidade — chegou a um cruzamento.
À direita, a floresta das Sombras Eternas sussurra segredos que não
deviam ser ouvidos. À esquerda, o Deserto dos Ossos Quebrados
estala a sua esperança de qualquer salvação. Inspire o medo. Expire
a esperança. Faça a sua escolha.
+ 1. Seguir para a floresta das Sombras Eternas.
+ 2. Seguir para o deserto dos Ossos Quebrados.
Select your option: nao me apetece
Invalid option. Please enter a number of an option, or one of the commands
(PRINT,RESET,LOAD,SAVE).
Select your option: 24
Invalid option. Such option does not exist.
Select your option: SAVE
Error: invalid file or error opening file!
Select your option: 1
[1]
Atravessa o limiar da floresta, onde a luz hesita e os sussurros
não têm dono. As árvores parecem mover-se, como se respirassem com
a noite. Algo observa. Algo sempre observou. Cada passo é engolido
pelo musgo e pela memória de quem nunca regressou. Mas agora é
tarde para recuar — ou será que não?
+ 0. Voltar para trás.
+ 2. Seguir para o deserto dos Ossos Quebrados.
+ 3. Seguir em frente.
Select your option: SAVE save1
Story state saved successfully.
Select your option: 0
[0]
À direita, a floresta das Sombras Eternas sussurra segredos que não
A sua aventura — tal como a sua sanidade — chegou a um cruzamento.
deviam ser ouvidos. À esquerda, o Deserto dos Ossos Quebrados
estala a sua esperança de qualquer salvação. Inspire o medo. Expire
a esperança. Faça a sua escolha.
+ 1. Seguir para a floresta das Sombras Eternas.
+ 2. Seguir para o deserto dos Ossos Quebrados.
Select your option: LOAD save1
Story state loaded successfully.
[1]
não têm dono. As árvores parecem mover-se, como se respirassem com
Atravessa o limiar da floresta, onde a luz hesita e os sussurros
a noite. Algo observa. Algo sempre observou. Cada passo é engolido
pelo musgo e pela memória de quem nunca regressou. Mas agora é
tarde para recuar — ou será que não?
+ 0. Voltar para trás.
+ 2. Seguir para o deserto dos Ossos Quebrados.
+ 3. Seguir em frente.
Select your option: 3
[3]
grito que já não é de medo, mas de fúria ancestral, desce o golpe
A lâmina treme nas suas mãos — ou será o mundo que treme? Com um
final. A criatura contorce-se, guincha, parte-se em fumo e
silêncio. A escuridão recua. O ar volta aos pulmões.
Sobreviveu. Venceu. Mas terá paz?
<WON>
```

Pode adicionar outros métodos à classe (de preferência privados).

Submeta o ficheiro ***Story.java*** no Problema C. O ficheiro não pode declarar nenhuma package. Não é necessária a submissão de mais nenhum ficheiro.

Dicas e sugestões

Deverá implementar primeiro o Problema A, e só depois o Problema B, e só depois o Problema C.

Juntamente com o enunciado do projeto foram disponibilizados uma série de ficheiros, incluindo os ficheiros Option.java, Scene.java, Node.java, e Story.java. Deverá implementar o seu código nos ficheiros fornecidos. Copie todos os ficheiros fornecidos para o seu projeto, pois serão necessários para conseguir executar o projeto.

Foi também disponibilizado um pequeno ficheiro de exemplo.

Neste 1.º projeto não serão avaliadas as complexidades temporal e espacial dos métodos desenvolvidos, pelo que não será necessário preocuparem-se demasiado com a eficiência dos mesmos. Nos próximos projetos iremos olhar em mais detalhe para estas questões.

Condições de realização

O 1.º projeto vale 15 % da nota final na componente de frequência da unidade curricular. O projeto deve ser **realizado individualmente**. Projetos iguais, ou muito semelhantes, originarão a reprovação na disciplina. O corpo docente da disciplina será o único juiz do que se considera ou não copiar num projeto.

O código do projecto deverá ser entregue obrigatoriamente por via eletrónica, através do sistema Mooshak, **até às 23:59 do dia 26 de setembro.** As validações terão lugar na semana seguinte, de **29 setembro a 3 de Outubro**. Os alunos terão de validar o código juntamente com o docente **durante** o horário de laboratório correspondente ao turno em que estão inscritos. **A avaliação e correspondente nota do projeto só terá efeito após a validação do código pelo docente.**

A avaliação da execução do código é feita automaticamente através do sistema *Mooshak*, usando vários testes configurados no sistema. O tempo de execução de cada teste está limitado, bem como a memória utilizada. Em breve serão disponibilizadas mais informações sobre o registo e acesso ao sistema *Mooshak*.