Programação Orientada por Objetos

2019/2020

Trabalho de Laboratório – Curso EI

7

Objetivos:
Introdução ao uso de exceções, IO e serialização.

Programa:
Pretende-se desenvolver um sistema de suporte à gestão de eventos na comunidade académica.
Nesta primeira etapa do programa, o gestor de eventos pretende organizar os dados relativos aos vários eventos e participantes, minimizar as vulnerabilidades a falhas provocadas por erros na introdução de dados, automatizar alguns processamentos e ter a possibilidade de efetuar cópias de segurança dos dados.

Regras de implementação:

- Criar a aplicação utilizando o IDE Netbeans.
- Implementar o código necessário e testar no fim de cada nível.
- Não é necessário obter dados do utilizador. Forneça os dados ao nível do código.
- Use as convenções de codificação adotadas para a linguagem Java (ver **Notas**).
- No método `main` separe o código de cada nível imprimindo na consola uma linha do tipo:
  - `//NIVEL N **********`

Implementação:

Nível 1:

- Implemente um enumerado `ParticipantType` (que representa os tipos de participantes) com os seguintes valores possíveis: `TEACHER`, `STUDENT`, `SPEAKER`, `OTHER`;
  - Redefina o método `toString()` de forma a retornar a designação de cada tipo;
  - Crie o método `getDiscount()` que retorna o valor do desconto aplicado a cada tipo de acordo com a seguinte tabela:

Tabela 1 - Tabela de descontos aplicáveis por tipo de participante

Speaker	100%
Student	50%
Teacher	25%
Other	0%

- Implemente na classe `Participant` (que representa um participante no evento) os atributos:
  - `name` – nome do participante;
  - `type` – tipo de participante;
- Crie um construtor que recebe todos os atributos e respetivos métodos seletores.
- Crie os seguintes métodos de validação (não use ainda exceções):
  - `validateName` – o nome do participante não pode ser nulo;
  - `validateType` – o tipo de participante não pode ser nulo.
- Implemente a classe `Registration` (que representa um registo num evento) com os atributos:
  - `registrationDate` – data do registo no evento;
  - `participant` – participante a ser registado;
  - `cost` – valor pago pelo participante no ato do registo.
- Crie um construtor que recebe os atributos `registrationDate`, `participant` e `eventCost` (que contém o valor de entrada no evento).
- Crie o método de validação (não use ainda exceções) `validateDate()` que valida que a data não é nula e que é igual ou posterior à data atual.
- Crie um método `calculateCost()` que recebe o valor de entrada no evento e retorna o custo da inscrição conforme o tipo de participante, após a aplicação do desconto.
- Crie os respetivos métodos seletores.

Nível 2:

- Implemente na classe **Event** (que representa um evento) os atributos:
  - **name** – nome do evento;
  - **date** – data do evento;
  - **local** – local do evento;
  - **price** – preço do evento;
  - **registrations** – lista (ArrayList) de todos os registos de participantes para o evento.
- Crie um construtor que recebe todos os atributos exceto **registrations**, que deve ser inicializado.
- Crie os seguintes métodos de validação (não use ainda exceções):
  - **validateName** – o nome do evento não pode ser nulo;
  - **validateLocal** – o local do evento não pode ser nulo.
  - **validatePrice** – o preço do evento não pode ser negativo;
  - **validateDate** – a data não pode ser nula, nem anterior à data atual.
- Crie o método **addRegister()** que recebe e adiciona um novo registo à lista.
- Crie os respetivos métodos seletores para todos os atributos da classe.
- Redefina o método **toString()** de forma a devolver a informação relativa ao nome, local, data e número de participantes no evento.
- Crie um tipo enumerado **ErrorCode** com os dados da Tabela 2. Implemente os textos explicativos fornecidos na figura, no método **toString()** deste enumerado:

Tabela 2 - Códigos de erro e respetivos testes

PARTICIPANT_NAME_CANT_BE_NULL	O nome do participante tem de ser fornecido
PARTICIPANT_TYPE_CANT_BE_NULL	O tipo do participante tem de ser fornecido
EVENT_NAME_CANT_BE_NULL	O nome do evento tem de ser fornecido
EVENT_LOCAL_CANT_BE_NULL	O nome do evento tem de ser fornecido
INVALIDE_PRICE	O preço do ingresso tem de ser positivo
EVENT_DATE_CANT_BE_NULL	A data do evento tem de ser fornecida
EVENT_DATE_CANT_BE_BEFORE	A data do evento não pode ser anterior à atual
REGISTRATION_DATE_CANT_BE_AFTER	A data do registo não pode ser posterior à atual
REGISTRATION_DATE_CANT_BE_NULL	A data do registo tem de ser fornecida
FILE_CANT_BE_NULL_OR_EMPTY	O ficheiro para imprimir não ser vazio ou nulo

- Implemente uma classe **EventManagerIllegalArgumentException** que herda de **IllegalArgumentException** com um atributo do tipo **ErrorCode** e um método para obter o código de erro da exceção. Crie ainda o construtor que tem um único parâmetro do tipo **ErrorCode**.
- Redefina as validações utilizadas nos construtores das classes **Participant**, **Registration** e **Event** para que, no caso de os argumentos recebidos serem inválidos, seja lançada uma **EventManagerIllegalArgumentException** com o código de erro apropriado.
- Para testar as classes crie uma classe **Program** com um único método estático **void main()** onde irá colocar o código de teste. Neste método **main** tente criar planos e membros por forma a testar, pelo menos, três das possíveis exceções geradas nas classes **Participant**, **Registration** e **Event**. Capture cada uma das exceções.

Nível 3:

- Implemente na classe **EventManager** (que representa um gestor de eventos) o atributo **events**, que representa uma lista de eventos.
- Crie um construtor sem argumentos que inicializa a lista de eventos.
- Crie o método **registerEvent()** que recebe um evento e, se este não for nulo, adiciona-o à respetiva lista.

- Crie o método `ticketInfo()` que recebe um evento, um registo e retorna a informação a ser disponibilizada no bilhete de ingresso. Este deve conter informação alusiva ao nome do evento, local, data, nome do participante e o preço do bilhete.
- Crie o método `registerParticipant()` que recebe o nome do evento, um participante e a data do registo e retorna uma `String` com a informação do bilhete. Tenha em consideração o seguinte:
  - Deve procurar o evento na lista através do seu nome, se não existir, retorna *null* (utilize programação funcional);
  - Se o evento não foi encontrado, informa o utilizador. Caso contrário, cria um novo registo;
  - Se a data recebida para registo for nula, deve ser considerada a data atual;
  - Após a criação do registo, adicionar à respetiva lista;
  - Retornar a informação do bilhete.

Nível 4:

- Crie o método `getEventParticipantsList()` que recebe o nome do evento e retorna informação de todos os participantes registados no evento (reestruture os métodos `toString()` das classes que achar necessário). Deve procurar o evento na lista através do seu nome, se não existir, informa o utilizador (utilize programação funcional).
- Crie a classe abstrata `EventManagerFileHandler`.
- Implemente um método estático `printToFile` que recebe como parâmetros:
  - `fileName` - String com o nome do ficheiro;
  - `textToSave` - String com o texto a guardar no ficheiro de texto;
- O conteúdo (textToSave) não pode ser nulo nem vazio. Se for, lança a exceção do tipo `EventManagerIllegalArgumentException`, específica para o efeito.
- No método `main` teste o método para gravar o resultado do método `getEventParticipantsList()` em ficheiro (o nome do ficheiro é constituído pelo nome do evento e referência à lista de participantes, por exemplo "Seminário de POO – Participantes.txt").
- Teste, ainda, a gravação do bilhete do último registo com o nome "UltimoBilhete.txt".

Nível 5:

- Implemente na classe abstrata `EventManagerFileHandler` um método estático `saveEvents()` que recebe como parâmetros uma `String` com o nome do ficheiro e a instância de `EventManager` a guardar num ficheiro usando a serialização de dados.
- Implemente também um método estático `loadEvents` que recebe o nome do ficheiro de onde deve ler uma instância de `EventManager`.
- No método `main` guarde um gestor de eventos com 2 eventos, e respetivos participantes, no ficheiro "data.bin", leia do ficheiro e imprima na consola o gestor de eventos que leu do ficheiro, de forma a obter um output semelhante ao seguinte:

```
Eventos existentes:

Nome                Local           Data            Participantes
Seminário de POO    ESTSetúbal      2020-06-10      5
Conferência de IA   ESTSetúbal      2020-08-25      1
```

Notas:

Para os identificadores siga as convenções adotadas normalmente, em particular:

1) A notação **camelCase** para o nome das variáveis locais e identificadores de atributos e métodos.
2) A notação **PascalCase** para os nomes das classes.
3) Não utilize o símbolo '_', nem abreviaturas nos identificadores.