Exercício-Programa 2
Gerenciador de Salas para o Marcador de Reuniões

Daniel Cordeiro, Flávio Coutinho e Marcos Chaim
Escola de Artes, Ciências e Humanidades
Universidade de São Paulo

Entrega: **30 de julho de 2021**

Introdução

Este é um programa[1] que será utilizado para auxiliar na marcação de uma reunião com vários participantes. Um dos participantes, responsável pela organização da reunião, indica em qual período ele gostaria de marcar a reunião (por exemplo, entre 10 e 20 de abril); ele determina também a lista de participantes identificados pelos seus endereços eletrônicos. A seguir, cada um dos participantes indica os horários de disponibilidade dentro do período determinado pelo organizador. O organizador então visualiza a sobreposição dos horários de todos os participantes e escolhe um horário para a reunião.

O trabalho deve ser feito preferencialmente em duplas.

Organização do código

Para implementar o programa proposto, dividimos o código a ser produzido em duas partes. A primeira parte especifica as classes e métodos que devem ser implementadas para a implementação da lógica de verificação de disponibilidade de horários dos participantes. A segunda parte complementa a primeira ao implementar código para reservar uma sala para a reunião que está sendo marcada.

Parte 1: disponibilidade de horários

A implementação deve se basear em uma interface de texto somente, ou seja, a visualização dos horários dos participantes será feita em modo texto utilizando-se, por exemplo, o objeto `Console`.

A definição dos participantes da reunião será feita utilizando-se do seguinte método que deverá ser implementado na classe `MarcadorDeReuniao`:

[1] Adaptado de um EP de MAC0441 gentilmente cedido pelo prof. Fábio Kon (IME–USP).

```
public void marcarReuniaoEntre(LocalDate dataInicial,
                               LocalDate dataFinal,
                               Collection<String> listaDeParticipantes)
```

As datas devem ser do tipo `java.time.LocalDate` e a `listaDeParticipantes` deve ser uma `Collection` de Strings que identificam os participantes.

Cada participante define os seus horários disponíveis através do seguinte método:

```
public void indicaDisponibilidadeDe(String participante,
                                    LocalDateTime inicio,
                                    LocalDateTime fim)
```

onde cada participante é identificado com uma String e o início e fim da disponibilidade é indicado com dias e horários dados por instâncias de `java.time.LocalDateTime`.

Finalmente, você deve implementar o método:

```
public void mostraSobreposicao()
```

que, provavelmente, vai dar o maior trabalho e vai exigir mais criatividade de vocês para informar os dados de uma forma clara e elegante. Ele deve exibir um relatório com as escolhas realizadas e indicar em quais horários todos os participantes poderiam participar da reunião.

Parte 2: Reserva de Salas

A Parte 2 especifica uma classe cujo objetivo é reservar as salas para as nossas reuniões. A classe `GerenciadorDeSalas` deve implementar, pelo menos, os seguintes métodos:

- `public void adicionaSalaChamada(String nome, int capacidadeMaxima, String descricao)`, que deve receber o nome da sala, a capacidade máxima da sala, e uma descrição;
- `public void removeSalaChamada(String nomeDaSala)`, que deve receber o nome da sala a ser removida;
- `public List<Sala> listaDeSalas()`, que deve devolver uma instância de `List` com objetos do tipo `Sala`;
- `public void adicionaSala(Sala novaSala)`, que deve receber uma instância de `Sala`;
- `public Reserva reservaSalaChamada(String nomeDaSala, LocalDateTime dataInicial, LocalDateTime dataFinal)`, que recebe um nome de sala, um `LocalDateTime` que indica o início da reserva e um outro `LocalDateTime` para indicar o final da reserva. O método deve devolver uma instância de `Reserva`;
- `public void cancelaReserva(Reserva cancelada)`, que recebe um objeto do tipo `Reserva` e cancela esta reserva;

- `public Collection<Reserva> reservasParaSala(String nomeSala)`, que devolve uma `Collection` de objetos `Reserva` que representam as reservas da respectiva sala.

- `public void imprimeReservasDaSala(String nomeSala)`, que recebe uma `String` com o nome da sala e imprime todas as suas reservas.

  Objetos do tipo `Sala` possuem métodos de acesso para os seguintes atributos: `nome`, `local`, `capacidade` e `observacoes`; `capacidade` é um inteiro, os demais atributos são `Strings`.

  Se a reserva for efetuada com sucesso, o método `reservaSalaChamada` devolve uma instância do objeto `Reserva` (com métodos `sala`, `inicio`, e `fim`, que devolvem, respectivamente, uma instância de `sala` e dois `LocalDateTime`, do início e do fim da reserva). Se a reserva falhar por qualquer motivo (p.ex. sala inexistente ou sala já reservada) o gerenciador deve obrigatoriamente lançar uma exceção verificada (*checked exception*) e opcionalmente imprimir uma mensagem de erro. Essa exceção não está na assinatura do método acima, você deve adicioná-la.

  Não há necessidade de implementar uma interface gráfica para o programa. Os exemplos de uso das classes que devem estar presentes no relatório (veja abaixo) são suficientes.

  Você tem total liberdade para criar novas classes e adicionar novos métodos e atributos às classes mencionadas no enunciado. Você **não pode** alterar as assinaturas dos métodos especificados no enunciado.

Entrega

Um arquivo compactado no formato `.tar.xz` ou `.zip` deve ser entregue no eDisciplinas contendo o código-fonte produzido, uma versão compilada (arquivos `.class`) do programa e um relatório (nos formatos abertos ODT[2] ou PDF). Seu relatório deve descrever em detalhes a implementação proposta e deve conter, obrigatoriamente:

1. o nome e número USP de todos os integrantes do grupo;
2. instruções sobre como compilar e executar seu programa;
3. exemplos de utilização do sistema e suas respectivas saídas;
4. um diagrama de classes UML com a descrição das classes e relacionamentos implementadas no projeto.

O seu EP será recompilado e testado em um computador com o seguinte software instalado:

[2] OpenDocument Format: `https://www.libreoffice.org/discover/what-is-opendocument/`.

```
$ lsb_release -a
Distributor ID: Debian
Description: Debian GNU/Linux 11 (bullseye)
Release: 11
Codename: bullseye

$ java -version
openjdk version ”11.0.11” 2021-04-20
OpenJDK Runtime Environment (build 11.0.11+9-post-Debian-1)
OpenJDK 64-Bit Server VM (build 11.0.11+9-post-Debian-1, mixed mode, sharing)
```

Dúvidas, problemas, etc.

Quaisquer dúvidas ou problemas relacionados ao EP devem ser discutidos no fórum de discussões do eDisciplinas.