INTEGRANTES

Matheus Pecoraro – 11917271
Sungwon Yoon – 9822261

COMO COMPILAR

No diretório /src, crie um método main e digite comando
$ javac *.java && java <arquivoContendoMain>

DIAGRAMA DE CLASSES UML

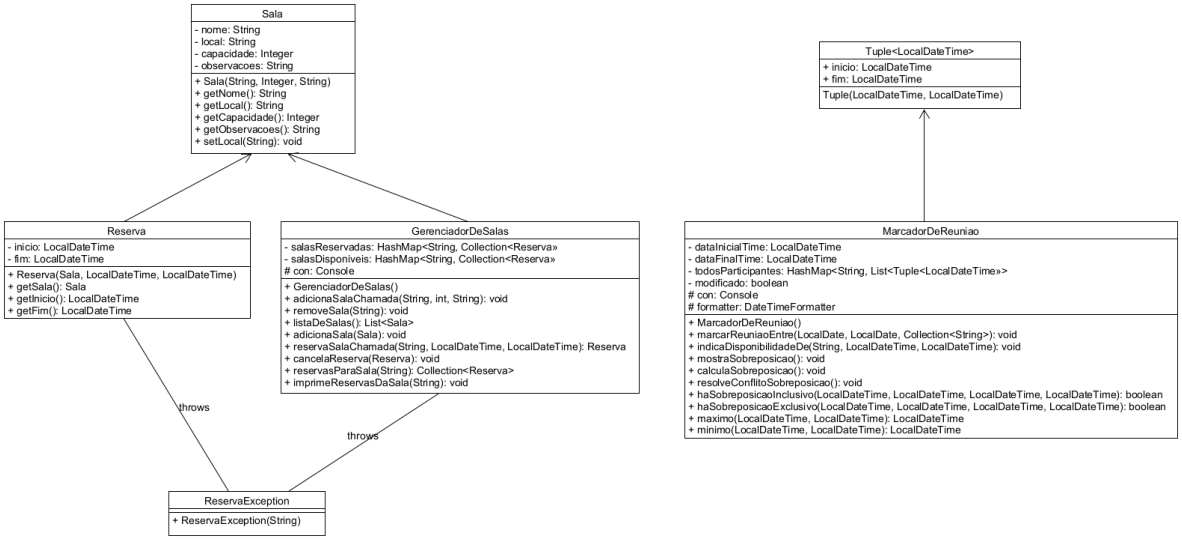

imagem maior : https://1.bp.blogspot.com/-M_FlqT6BRDQ/YQQ4naNEL4I/AAAAAAAAAUY/4z7YKX8-4JAiYnSMBIV6qk4TPyekJYqeACNcBGAsYHQ/s1610/diagrama.png

IMPLEMENTAÇÃO - Marcador de Reunião

Em um método main, crie um objeto de classe MarcadorDeReuniao para definir um horário de reunião.

Para definir a lista de participantes e um intervalo de reunião, deve-se invocar o método **marcarReuniaoEntre**, que recebe como parâmetros inicio e fim de LocalDate e participantes de string.

Uma vez invocado o método com argumentos corretos, a lista de participantes é guardada e as datas são guardadas como tipo LocalDateTime para facilitar operações posteriores. E imprime o início e fim conforme formato “dia-mês-ano”

```
==========================================
Marcar reuniao entre 05-07-2021 e 10-07-2021
==========================================
```

Caso a data de início for maior que data de término, a reserva não é realizada e é impressa a mensagem:

```
Data inicial nao pode ser maior que data final
Digite novamente a data inicial e final
```

Para definir a disponibilidade de cada um dos participantes, deve ser invocado o método **indicaDisponibilidadeDe**, que recebe um string participante, e LocalDateTime inicio e fim como parâmetros.

Caso o participante não estiver na lista de participantes definida no método marcarReuniaoEntre, apenas imprime a mensagem:

```
nao adicionado coo@usp.br : nao encontrado na lista de participantes
```

Caso o participante estiver na lista de participantes mas o horário inicial for maior que o horário final, apenas imprime a mensagem:

```
nao adicionado coo@usp.br : data inicial nao pode ser maior que data final
```

Por último, se os horários estiverem fora do intervalo definido no método marcarReuniaoEntre, imprime a mensagem:

```
nao adicionado coo@usp.br : disponibilidade fora do intervalo de reuniao
definido
```

Se todas as condições forem satisfeitas, o intervalo de horário é guardado no objeto do tipo **Tuple<LocalDateTime>**, que tem variáveis internas LocalTimeDate inicio e fim, e inserido no HashMap com key participante. E imprime a mensagem confome formato “dia-mês-ano hora:minuto:segundo”.

```
adicionado coo@usp.br : 06-07-2021 00:00:00 ~ 08-07-2021 00:00:00
```

Para visualizar a sobreposição de horários, deve-se invocar o método **mostraSobreposicao**, que não possui parâmetros.

Primeiramente, com método **calculaSobreposicao** é calculada a sobreposição com

base nos intervalos de disponibilidades indicados anteriormente. Finalmente, se houver mais de dois horários possíveis de reunião, é executado **resolveConflitoSobreposicao** para unir horários contínuos.

Por exemplo, se a entrada for

```
pessoa a : 05-07-2021 00:00 ~ 06-07-2021 23:59
pessoa a : 06-07-2021 23:59 ~ 07-07-2021 23:59

pessoa b : 05-07-2021 00:00 ~ 06-07-2021 23:59
pessoa b : 06-07-2021 23:59 ~ 07-07-2021 23:59
```

a execução do método calculaSobreposição resultará em

```
05-07-2021 00:00:00 ~ 06-07-2021 23:59:59
06-07-2021 23:59:59 ~ 07-07-2021 23:59:59
```

devido a dois intervalos oferecidos separadamente. Assim, é preciso unir esses horários em um único intervalo contínuo. Para isso, é executado o método, resultado em

```
05-07-2021 00:00:00 ~ 07-07-2021 23:59:59
```

Se houver pelo menos um participante sem disponibilidade ou não for encontrado uma sobreposição, imprime a mensagem:

```
==========================================
possiveis horarios de reuniao :

Nao existe um horario comum para todos os participantes
==========================================
```

Caso contrário, imprime horários sobrepostos:

```
==========================================
possiveis horarios de reuniao :

06-07-2021 00:00 ~ 10-07-2021 23:59
==========================================
```

E em caso de mais de um horário possível:

```
==========================================
possiveis horarios de reuniao :

06-07-2021 00:00 ~ 08-07-2021 00:00
09-07-2021 00:00 ~ 10-07-2021 23:59
==========================================
```

EXEMPLOS – Marcador de Reunião

Exemplo – sem erro de inserção

```
MarcadorDeReuniao marcador = new MarcadorDeReuniao();

List<String> participantes = new ArrayList<>(List.of("a", "b"));

marcador.marcarReuniaoEntre(LocalDate.of(2021, 7, 5), LocalDate.of(2021, 7, 10), participantes);

marcador.indicaDisponibilidadeDe("a", LocalDateTime.of(2021, 7, 6, 0, 0, 0), LocalDateTime.of(2021, 7, 7, 23, 59, 59));
marcador.indicaDisponibilidadeDe("a", LocalDateTime.of(2021, 7, 7, 23, 59, 59), LocalDateTime.of(2021, 7, 8, 23, 59, 59));
marcador.indicaDisponibilidadeDe("b", LocalDateTime.of(2021, 7, 6, 0, 0, 0), LocalDateTime.of(2021, 7, 7, 23, 59, 59));
marcador.indicaDisponibilidadeDe("b", LocalDateTime.of(2021, 7, 7, 23, 59, 59), LocalDateTime.of(2021, 7, 8, 23, 59, 59));

marcador.mostraSobreposicao();
```

Output

```
==========================================
Marcar reuniao entre 05-07-2021 e 10-07-2021
==========================================

adicionado a : 06-07-2021 00:00 ~ 07-07-2021 23:59
adicionado a : 07-07-2021 23:59 ~ 08-07-2021 23:59
adicionado b : 06-07-2021 00:00 ~ 07-07-2021 23:59
adicionado b : 07-07-2021 23:59 ~ 08-07-2021 23:59

==========================================
possiveis horarios de reuniao :

06-07-2021 00:00 ~ 08-07-2021 23:59
==========================================
```

Exemplo – um participante com um intervalo e seu subintervalo

```
MarcadorDeReuniao marcador = new MarcadorDeReuniao();

List<String> participantes = new ArrayList<>(List.of("a", "b", "c", "d"));

marcador.marcarReuniaoEntre(LocalDate.of(2021, 7, 5), LocalDate.of(2021, 7, 10), participantes);

marcador.indicaDisponibilidadeDe("a", LocalDateTime.of(2020, 7, 6, 11, 59, 56), LocalDateTime.of(2021, 7, 11, 23, 21, 59));
marcador.indicaDisponibilidadeDe("b", LocalDateTime.of(2021, 7, 5, 10, 56, 0), LocalDateTime.of(2021, 7, 7, 11, 0, 0));
marcador.indicaDisponibilidadeDe("c", LocalDateTime.of(2021, 7, 6, 0, 0, 0), LocalDateTime.of(2021, 7, 10, 23, 59, 59));
marcador.indicaDisponibilidadeDe("c", LocalDateTime.of(2021, 7, 7, 0, 0, 59), LocalDateTime.of(2021, 7, 8, 23, 59, 59));
marcador.indicaDisponibilidadeDe("d", LocalDateTime.of(2021, 7, 6, 0, 45, 0), LocalDateTime.of(2021, 7, 9, 12, 22, 7));

marcador.mostraSobreposicao();
```

Output

```
==========================================
Marcar reuniao entre 05-07-2021 e 10-07-2021
==========================================

adicionado a : 05-07-2021 00:00 ~ 10-07-2021 23:59
adicionado b : 05-07-2021 10:56 ~ 07-07-2021 11:00
adicionado c : 06-07-2021 00:00 ~ 10-07-2021 23:59
adicionado c : 07-07-2021 00:00 ~ 08-07-2021 23:59
adicionado d : 06-07-2021 00:45 ~ 09-07-2021 12:22

==========================================
possiveis horarios de reuniao :

06-07-2021 00:45 ~ 07-07-2021 11:00
==========================================
```

Exemplo – inserção de um participante 'c' não pertencente à lista

```java
MarcadorDeReuniao marcador = new MarcadorDeReuniao();

List<String> participantes = new ArrayList<>(List.of("a", "b"));

marcador.marcarReuniaoEntre(LocalDate.of(2021, 7, 5), LocalDate.of(2021, 7, 10), participantes);

marcador.indicaDisponibilidadeDe("a", LocalDateTime.of(2020, 7, 6, 11, 59, 56), LocalDateTime.of(2021, 7, 11, 23, 21, 59));
marcador.indicaDisponibilidadeDe("b", LocalDateTime.of(2021, 7, 5, 10, 56, 0), LocalDateTime.of(2021, 7, 7, 11, 0, 0));
marcador.indicaDisponibilidadeDe("c", LocalDateTime.of(2021, 7, 6, 0, 0, 0), LocalDateTime.of(2021, 7, 10, 23, 59, 59));

marcador.mostraSobreposicao();
```

Output

```
==========================================
Marcar reuniao entre 05-07-2021 e 10-07-2021
==========================================

adicionado a : 05-07-2021 00:00 ~ 10-07-2021 23:59
adicionado b : 05-07-2021 10:56 ~ 07-07-2021 11:00
nao adicionado c : nao encontrado na lista de participantes

==========================================
possiveis horarios de reuniao :

05-07-2021 10:56 ~ 07-07-2021 11:00
==========================================
```

Exemplo – um participante ‘c’ sem disponibilidade indicada

```
MarcadorDeReuniao marcador = new MarcadorDeReuniao();

List<String> participantes = new ArrayList<>(List.of("a", "b", "c"));

marcador.marcarReuniaoEntre(LocalDate.of(2021, 7, 5), LocalDate.of(2021, 7, 10), participantes);

marcador.indicaDisponibilidadeDe("a", LocalDateTime.of(2020, 7, 6, 11, 59, 56), LocalDateTime.of(2021, 7, 11, 23, 21, 59));
marcador.indicaDisponibilidadeDe("b", LocalDateTime.of(2021, 7, 5, 10, 56, 0), LocalDateTime.of(2021, 7, 7, 11, 0, 0));

marcador.mostraSobreposicao();
```

Output

```
==========================================
Marcar reuniao entre 05-07-2021 e 10-07-2021
==========================================

adicionado a : 05-07-2021 00:00 ~ 10-07-2021 23:59
adicionado b : 05-07-2021 10:56 ~ 07-07-2021 11:00

==========================================
possiveis horarios de reuniao :

Nao existe um horario comum para todos os participantes
==========================================
```

IMPLEMENTAÇÃO - Gerenciador de Salas

Em um método main, crie um objeto de classe GerenciadorDeSalas para gerenciar salas de reunião.

Para criar uma sala de reunião pode-se usar o método **adicionaSalaChamada**, que recebe nome, capacidadeMaxima e descricao como parâmetros, sendo todos strings menos capacidadeMaxima que é um Integer, o método irá criar um objeto Sala com esses parâmetros e adicioná-lo no HashMap salasDisponiveis, utilizando o próprio nome da sala como chave. Alternativamente o objeto Sala pode ser criado diretamente no método main e adicionado na lista de salas do objeto de GerenciadorDeSalas atráves do método **adicionaSala** que recebe um objeto de Sala como parâmetro e o adiciona em salasDisponiveis, também utilizando o próprio nome da sala como chave.

Para remover uma sala da lista de salas disponiveis deve-se utilizar o método **removeSalaChamada**, que recebe uma string nomeDaSala como parâmetro, o método irá buscar a chave com o valor de nomeDaSala no HashMap salasDisponiveis e remove-lo.

O método **listaDeSalas** retorna uma ArrayList contendo todos os objetos de salas existentes no HashMap salasDisponiveis no objeto de GerenciadorDeSalas utilizado.

Para reservar uma sala deve-se utilizar o método **reservaSalaChamada**, que recebe uma string nomeDaSala, dataInicial e dataFinal (ambos LocalDateTime) como parâmetros, o método irá usar os parâmetros para criar um objeto Reserva que será retornado no final da função e também guardado numa coleção de objetos Reserva no HashMap salasReservadas utilizando o nome da sala a qual a reserva está relacionada como chave. Caso a data passada pelos parâmetros tenha colisão em horário com alguma data já salva para aquela sala ou uma sala com o nome igual a nomeDaSala não exista, o método lança uma exceção do tipo ReservaException.

Para cancelar uma reserva deve-se utilizar o método **cancelaReserva**, que recebe um objeto Reserva de parâmetro, o método irá remover o objeto Reserva recebido da coleção de reservas da sala a qual ele pertence no HashMap salasReservadas.

O método **reservasParaSala**, que recebe uma string nomeSala como parâmetro, devolve a coleção de reservas relacionadas a chave com o valor de nomeSala guardada no HashMap salasReservadas.

O método **imprimeReservasDaSala** recebe uma string nomeSala como parâmetro e imprime todas as reservas para a sala especificada no seguinte formato

```
A sala <nomeSala> esta reservada para os seguintes horarios:
Das <dataInicioR1> ate as <dataInicioR2>
...
```

obs: as datas são escritas no formato dd-MM-yyyy HH:mm:ss

A classe **Sala** possui, 4 variáveis internas: nome, local, capacidade e observacoes,

um construtor que recebe de parâmetro nome e observacoes como strings e capacidade como integer e, ao ser executado, assimila os parametros as variáveis de Sala, uma função **setLocal** que recebe uma string local de parâmetro e serve para modificar o valor da varíavel local, e 4 funções que, cada uma, retornam os valores das variáveis de Sala.

A classe **Reserva** possui, 3 variáveis internas: sala (objeto da classe Sala), inicio e fim (ambas LocalDateTime), um construtor que recebe de parâmetro sala, inicio e fim e, ao ser executado, assimila os parametros as variáveis de Reserva, e 3 funções que, cada uma, retornam os valores das variáveis de Reserva.

A classe **ReservaException** extende a classe **Exception** e tem apenas um construtor que recebe de parâmetro uma string msgErro que é apenas passada como parâmetro para o construtor da classe Exception.

EXEMPLOS - Gerenciador de Salas

Adicionando e removendo salas:

```
GerenciadorDeSalas g = new GerenciadorDeSalas();
Sala sala = new Sala("Sala 1", 153, "Sala de teste1");
sala.setLocal("Sao Paulo");
g.adicionaSala(sala);
g.adicionaSalaChamada("Sala 2", 130, "Sala de teste2");
g.adicionaSalaChamada("Sala 3", 200, "Sala de teste3");
g.adicionaSalaChamada("Sala 4", 400, "Sala de teste4");
g.adicionaSalaChamada("Sala 5", 3, "Sala de teste5");

List<Sala> salasLista;
salasLista = g.listaDeSalas();
for(Sala salaLoop : salasLista){
    System.out.println(salaLoop.getNome()+", "+salaLoop.getObservacoes()+", "+salaLoop.getCapacidade()+", "+salaLoop.getLocal());
}

System.out.println();

g.removeSalaChamada("Sala 3");
g.removeSalaChamada("Sala 1");

salasLista = g.listaDeSalas();
for(Sala salaLoop : salasLista){
    System.out.println(salaLoop.getNome()+", "+salaLoop.getObservacoes()+", "+salaLoop.getCapacidade()+", "+salaLoop.getLocal());
}
```

Output:

```
Sala 1, Sala de teste1, 153, Sao Paulo
Sala 2, Sala de teste2, 130, Nao especificado
Sala 3, Sala de teste3, 200, Nao especificado
Sala 4, Sala de teste4, 400, Nao especificado
Sala 5, Sala de teste5, 3, Nao especificado

Sala 2, Sala de teste2, 130, Nao especificado
Sala 4, Sala de teste4, 400, Nao especificado
Sala 5, Sala de teste5, 3, Nao especificado
```

Adicionando e removendo reservas + exceção:

```java
LocalDateTime dataInicio1 = LocalDateTime.of(2001, 9, 8, 21, 47, 5);
LocalDateTime dataFim1 = LocalDateTime.of(2002, 9, 8, 21, 47, 5);
LocalDateTime dataInicio2 = LocalDateTime.of(2003, 9, 8, 21, 47, 5);
LocalDateTime dataFim2 = LocalDateTime.of(2004, 9, 8, 21, 47, 5);
LocalDateTime dataInicio3 = LocalDateTime.of(2005, 9, 8, 21, 47, 5);
LocalDateTime dataFim3 = LocalDateTime.of(2006, 9, 8, 21, 47, 5);

try {
    Reserva r = g.reservaSalaChamada("Sala 4", dataInicio1, dataFim1);
    g.reservaSalaChamada("Sala 4", dataInicio3, dataFim3);
    g.imprimeReservasDaSala("Sala 4"); System.out.println();

    g.reservaSalaChamada("Sala 2", dataInicio2, dataFim2);
    g.imprimeReservasDaSala("Sala 2"); System.out.println();

    g.cancelaReserva(r);
    g.imprimeReservasDaSala("Sala 4"); System.out.println();

    g.reservaSalaChamada("Sala 2", dataInicio1, dataFim3);
}
catch(ReservaException e) {
    System.out.println(e.getMessage());
}
```

Output:

```
A sala Sala 4 esta reservada para os seguintes horarios:
Das 08-09-2001 21:47:05 ate as 08-09-2002 21:47:05
Das 08-09-2005 21:47:05 ate as 08-09-2006 21:47:05

A sala Sala 2 esta reservada para os seguintes horarios:
Das 08-09-2003 21:47:05 ate as 08-09-2004 21:47:05

A sala Sala 4 esta reservada para os seguintes horarios:
Das 08-09-2005 21:47:05 ate as 08-09-2006 21:47:05

A sala ja esta reservado dentro do horario especificado
```