



Nota: 10.0



Tempo: Terminou



Sair

Modelagem e implementação de **classes** em Java que deem conta implementadas pela **Java Collections Framework**.

- Data de entrega: 07/09/2023 23:59:59

## Descrição



- Neste trabalho, você implementará as **classes** necessárias para representar para um *Sistema de Ensalamento*. Um sistema de ensalamento é responsável por alocar turmas nas salas disponíveis.
- Irá descrever **atributos** e implementar **métodos** e **construtores** com base em um **diagrama de classes** da **UML**.
- Em seguida, **objetos** serão criados a partir das **classes** implementadas.
- Nesta **abstração**, o *ensalamento* possui uma lista de *salas*, uma lista de *turmas*, e uma lista de *turma em sala*, que indica que uma determinada turma está alocada em uma sala.

## Passos Iniciais

- Inicie o Eclipse. Vá em **"File" → "New" → "New Java Project"**. Nome do projeto: **"TAP-ColecoesGenericasII"**.
  - Na parte *Module* (parte de baixo da janela), desselecione **"Create module-info.java file"**.
- Nas questões a seguir, será pedido para criar diversas classes. Para isso, vá em **"File" → "New" → "Class"**.
  - Deixe o campo **"package"** em branco.
- O Eclipse compila automaticamente as classes sempre que você salva. Para executar e testar, basta ir em **"Run" → "Run"** (Ctrl+F11).
- A saída e entrada do programa será feita através da aba **"Console"** do Eclipse.

### Questão 1: Classe Sala

C	Sala
a	bloco: <i>int</i>
a	sala: <i>int</i>
a	capacidade: <i>int</i>
a	acessivel: <i>boolean</i>
M	Sala()
M	Sala(int bloco, int sala,

 *int capacidade, boolean acessivel)*  
 *getDescricao(): String*

Crie uma classe para representar uma **Sala** de aula. Os atributos de um objeto da classe **Sala** poderão ser setados diretamente (e.g., `sala1.bloco=6`) ou pelo *método construtor*. Use a técnica de *encadeamento de construtores* para criar os construtores da classe, semelhante ao feito em sala e mostrado nos slides. No diagrama ao lado, o atributo `capacidade` indica a capacidade máxima (quantidade máxima de alunos) que uma sala comporta. Já o atributo booleano `acessivel` indica se uma sala é de fácil acesso (para um cadeirante) ou não.

Implemente o método:

- `getDescricao`: retorna uma **String** contendo a descrição da sala de acordo com o exemplo abaixo:

`Bloco 6, Sala 101 (50 lugares, acessível)`

Para testar a classe, crie uma nova classe chamada **EnsalamentoMain**. Nesta classe, crie o método `main`, que será o ponto de partida do seu programa. No método `main`, crie um ou mais objetos da classe **Sala** e, em seguida, imprima o resultado da execução do método `getDescricao` dos objetos criados.

Para essa questão, submeta apenas a classe **Sala**. Não precisa submeter a classe **EnsalamentoMain**.









O prazo de entrega do trabalho terminou. Portanto, o botão abaixo está desabilitado.

Enviar "Sala.java"

Solução correta!

## Questão 2: Classe Turma



 Turma
 nome: <i>String</i>  professor: <i>String</i>  numAlunos: <i>int</i>  acessivel: <i>boolean</i>  horarios: <i>ArrayList&lt;Integer&gt;</i>
 <i>Turma()</i>  <i>Turma(String nome, String professor, int numAlunos, boolean acessivel)</i>

- M addHorario(int horario): void
- M getHorariosString(): String
- M getDescricao(): String

De forma semelhante às questões anteriores, crie uma classe para representar uma **Turma**. Nesta classe, o atributo numAlunos indica a quantidade de alunos matriculados na disciplina. O atributo acessivel indica se algum aluno na turma requer uma sala de fácil acesso. Já o atributo horarios, que é uma lista de inteiros, indica os horários desta turma de acordo com a tabela abaixo:

Hs/Dia	Seg	Ter	Qua	Qui	Sex
8	1	8	15	22	29
10	2	9	16	23	30
12	3	10	17	24	31
14	4	11	18	25	32
16	5	12	19	26	33
18	6	13	20	27	34
20	7	14	21	28	35

ou seja, se a lista de horários de uma turma contém os números 1, 15 e 29, isso indica que esta turma tem aulas nas segundas, quartas e sextas das 8 às 10hs.

Ainda a respeito do atributo horarios, note que como as coleções genéricas em Java só aceitam classes (e não tipos primitivos), estamos usando a classe **Integer** ao invés do tipo primitivo **int**. Entretanto, as conversões entre a classe e o tipo primitivo são feitos de forma automática devido ao *autoboxing/auto-unboxing* da linguagem Java (ver slides).

Implemente os métodos:

- addHorario: adiciona um horário à lista de horários da turma.
- getHorariosString: retorna uma **String** contendo o horário da turma, de acordo com o exemplo abaixo:
- getDescricao: retorna uma **String** contendo a descrição completa da turma de acordo com o exemplo abaixo:

```
|segunda 8hs, quarta 8hs, sexta 8hs
```

```
|Turma: Algoritmos e Estrutura de Dados I
|Professor: Edleno Silva
|Número de Alunos: 60
|Horário: segunda 8hs, quarta 8hs, sexta 8hs
|Acessível: sim
```

Para testar a classe, modifique a classe **EnsalamentoMain**, criada na questão anterior, para criar um ou mais objetos da classe **Turma** e, em seguida, imprima o resultado da execução do método getDescricao dos objetos criados.

Para essa questão, submeta apenas a classe **Turma**. Não precisa submeter a classe **EnsalamentoMain**.

*O prazo de entrega do trabalho terminou. Portanto, o botão abaixo está desabilitado.*

Enviar "Turma.java"

Solução correta!

### Questão 3: Classe TurmaEmSala

C	TurmaEmSala
a	turma: <i>Turma</i>
a	sala: <i>Sala</i>
M	TurmaEmSala()
M	TurmaEmSala( <i>Turma turma, Sala sala</i> )

De forma semelhante às questões anteriores, crie uma classe para representar uma **TurmaEmSala**, que indica que uma turma está alocada em uma determinada sala.














*O prazo de entrega do trabalho terminou. Portanto, o botão abaixo está desabilitado.*

Enviar "TurmaEmSala.java"

Solução correta!

### Questão 4: Classe Ensalamento

C	Ensalamento
a	salas: <i>ArrayList&lt;Sala&gt;</i>
a	turmas: <i>ArrayList&lt;Turma&gt;</i>
a	ensalamento: <i>ArrayList&lt;TurmaEmSala&gt;</i>

 Ensalamento()  
 addSala(*Sala sala*): *void*  
 addTurma(*Turma turma*): *void*  
 getSala(*Turma turma*): *Sala*  
 salaDisponivel(*Sala sala*, *int horario*): *boolean*  
 salaDisponivel(*Sala sala*,  
*ArrayList<Integer> horarios*): *boolean*  
 alocar(*Turma turma*, *Sala sala*): *boolean*  
 alocarTodas(): *void*  
 getTotalTurmasAlocadas(): *int*  
 getTotalEspacoLivre(): *int*  
 relatorioResumoEnsalamento(): *String*  
 relatorioTurmasPorSala(): *String*  
 relatorioSalasPorTurma(): *String*

De forma semelhante às questões anteriores, crie uma classe para representar um **Ensalamento**. Esta classe será a principal parte do sistema, pois terá como atributos uma lista de salas e uma lista de turmas e, adicionalmente, terá métodos para gerar o ensalamento (alocar salas às turmas) e métodos para gerar relatórios.

O "resultado" do ensalamento será o atributo **ensalamento**, que terá uma lista de objetos da classe **TurmaEmSala**, em que cada objeto da lista "liga" uma turma a uma determinada sala. Este atributo terá todas as informações necessárias para responder perguntas e executar ações como:

- Qual a sala de uma determinada turma? -- método **Sala** getSala(**Turma** turma)
- Uma determinada sala está disponível em um determinado horário? -- método salaDisponivel(**Sala** sala, **int** horario)
- Uma determinada sala está disponível em todos os horários de uma lista? -- método **boolean** salaDisponivel(**Sala** sala, **ArrayList<Integer>** horarios)
- Aloque, caso seja possível, uma determinada turma em uma determinada sala. -- método alocar(**Turma** turma, **Sala** sala)
- Aloque todas as turmas da lista de turmas. -- método alocarTodas()
- Quantas turmas foram alocadas em uma sala com sucesso? -- método getTotalTurmasAlocadas()
- Qual o total de espaços livres nas salas? -- método getTotalEspacoLivre()

Segue um detalhamento, e dicas de implementação, para os métodos da classe:

- `addSala(Sala sala)`: adiciona um objeto da classe `Sala` na lista de salas;
- `addTurma(Turma turma)`: adiciona uma `Turma` na lista de turmas;
- `getSala(Turma turma)`: retorna a sala alocada a uma determinada turma. Se nenhuma sala for encontrada, retorna `null`.
  - Para implementar este método, será necessário percorrer a lista do atributo `ensalamento` procurando pelo objeto cujo atributo `turma` seja igual ao parâmetro `turma` do método. Como uma turma só será alocada a uma única sala, após encontrar o primeiro elemento cuja `turma` seja a procurada, pode-se retornar a turma encontrada (sem precisar continuar percorrendo a lista de `ensalamento`).
- `salaDisponivel(Sala sala, int horario)`: retorna `true` se a sala está disponível em um determinado horário. `false` caso contrário.
  - Para implementar este método, será necessário percorrer a lista do atributo `ensalamento` procurando por todos os objetos cujo atributo `sala` sejam iguais ao parâmetro `sala` do método. Para cada objeto encontrado, deve-se verificar o atributo `horarios` do atributo `turma`. Se um dos elementos do atributo `horarios` for igual ao parâmetro `horario`, isso significa que a sala encontra-se ocupada naquele horário (retorna `false`). Se o horário de nenhuma turma bater com o horário procurado, isso significa que a sala está disponível (não possui nenhuma outra turma naquele dia/horário, retorna `true`).
- `salaDisponivel(Sala sala, ArrayList<Integer> horarios)`: retorna `true` se a sala está disponível em todos os horários do parâmetro `horarios` (lista de inteiros). `false` caso contrário.
  - Para implementar este método, execute o método anterior para cada elemento do parâmetro `horarios`.
- `alocar(Turma turma, Sala sala)`: tenta alocar uma turma em uma determinada sala. Caso consiga, um objeto da classe `TurmaEmSala` com a turma e a sala deve ser adicionada na lista `ensalamento` e deve-se retornar `true`. Caso não seja possível alocar a turma na sala, retorna-se `false`. Para uma turma poder ser alocada em uma sala, os seguintes requisitos precisam ser atendidos:
  - Uma turma `acessivel` só pode ser alocada em uma sala também `acessivel`.
  - A quantidade de alunos na turma deve ser igual ou menor à capacidade da sala.
  - A sala precisa estar disponível em todos os horários da turma.
- `alocarTodas()`: aloca uma sala para todas as turmas do atributo `turmas`, caso seja possível.
  - Uma implementação simples (até demais) para este método é executar o método anterior para cada elemento da lista `turmas` e, para cada turma, ir tentando alocar em cada sala até que uma esteja disponível. Entretanto, esta solução poderá gerar um `ensalamento` menos eficiente (ver método `getTotalEspacoLivre`) e, em alguns casos, deixar algumas turmas sem sala (ver método `getTotalTurmasAlocadas`) enquanto que uma solução mais elaborada poderia ser capaz de gerar um `ensalamento` mais eficiente e alocando uma sala para todas as turmas, assunto da questão "ponto extra".
- `getTotalTurmasAlocadas()`: retorna a quantidade de turmas que tiveram uma sala alocada com sucesso. O ideal é que este número seja igual ao total de turmas. Entretanto, caso não haja salas disponíveis para todas as turmas ou caso o seu algoritmo não seja eficaz, este número será menor que a quantidade de turmas, o que significa que algumas turmas não terão salas de aula.
  - Para implementar este método, percorra o atributo `ensalamento` e conte a quantidade de objetos cujo atributo `sala` seja diferente de `null`.
- `getTotalEspacoLivre()`: retorna um número que indica a eficiência do seu algoritmo de `ensalamento`. Um algoritmo "ineficiente" seria um algoritmo que alocasse uma turma pequena em uma sala muito grande (evitando que esta sala seja usada por turmas maiores, e, possivelmente, fazendo com que as turmas grandes fiquem sem sala).
  - Para gerar este número, inicie um acumulador em zero e, para cada turma alocada, incremente o acumulador com a subtração do tamanho da turma pela capacidade da sala (`capacidade - salaAtual.capacidade - turmaAtual.numAlunos`). Quanto menor este número, mais eficiente é o

peça capacidade da sala (e.g., `total += salaAtual.Capacidade - turmaAtual.numAlunos`). Quanto menor este número, mais eficiente é o ensalamento (pois as turmas foram alocadas em salas de tamanho compatível).

- `relatorioResumoEnsalamento()`: retorna uma `String` contendo um resumo do ensalamento, conforme o exemplo a seguir:

```
Total de Salas: 4  
Total de Turmas: 4  
Turmas Alocadas: 3  
Espaços Livres: 65
```

- `relatorioTurmasPorSala()`: retorna uma `String` contendo um relatório das salas com suas turmas alocadas, conforme o exemplo a seguir:

```
Total de Salas: 4  
Total de Turmas: 4  
Turmas Alocadas: 3  
Espaços Livres: 65
```

```
--- Bloco 6, Sala 101 (50 lugares, acessível) ---
```

```
Turma: Técnicas de Programação  
Professor: Horácio Fernandes  
Número de Alunos: 50  
Horário: terça 14hs, quinta 14hs, sexta 14hs  
Acessível: não
```

```
Turma: Laboratório de Programação C  
Professor: Edson Nascimento  
Número de Alunos: 25  
Horário: segunda 8hs, quarta 8hs, sexta 8hs  
Acessível: sim
```