

# Amigo Oculto 4 - Sugestões



[Amigo Oculto](#) > 4. Sugestões

## Sugestões de presentes



Cada usuário pode sugerir quais presentes gostaria de receber. Essas sugestões serão mantidas em um segundo arquivo que também precisará do seu CRUD. Nesse arquivo, é necessário manter o vínculo da sugestão ao usuário que a fez. Esse relacionamento é do tipo 1:N (1 usuário para N sugestões).

### SUGESTÕES

1	1	2	3	4	5	6	7	8	9	10	11
---	---	---	---	---	---	---	---	---	---	----	----

## Arquivos genéricos

Agora, precisamos criar um segundo CRUD. Mais tarde, criaremos um terceiro CRUD, um quarto CRUD e, talvez, ainda outros mais. Para evitar criar várias classes semelhantes, que fazem praticamente a mesma coisa, você precisa transformar o seu CRUD em um CRUD de tipos genéricos.

Isso significa que os métodos *create*, *read*, *update* e *delete* poderão receber e manipular objetos de quaisquer classes e não apenas de usuários. Em todos os pontos do CRUD em que você tiver usado uma referência da classe Usuário, agora precisará trocar por uma referência à classe genérica T. Se ainda não souber como fazer isso, consulte a nossa página sobre tipos genéricos. Apesar dessa página explicar o processo em Java, não é difícil fazer algo semelhante nas outras linguagens.

[Tipos genéricos em .Java](#)

Como mostra essa página, você precisará criar uma classe CRUD de tipos genéricos e uma interface que será implementada pelas entidades (Usuário, Sugestão, etc.). No nosso projeto, chamarei essa interface de `Registro`.

## A entidade Sugestão

Uma das coisas mais difíceis na brincadeira de Amigo Oculto é saber o que a outra pessoa gostaria de ganhar. Assim, seria bem legal ter algum tipo de dica ou sugestão do que essa pessoa gostaria que facilitasse a nossa compra. Vamos fazer isso aqui.

Em primeiro lugar, precisamos definir a estrutura da nossa entidade Sugestão. Usaremos os seguintes atributos para ela:

- `int idSugestão` - Este atributo identificará cada sugestão de forma exclusiva. Da mesma forma que no caso dos usuários, ele será gerado automaticamente e será na forma de um número inteiro sequencial. Os valores sequenciais, no entanto, não serão considerados por usuário, mas para todo o arquivo. Assim, a sugestão de ID 1 pode pertencer a um usuário, a de ID 2 pode pertencer a outro usuário e a sugestão de ID 3 pode ser de um terceiro usuário.
- `int idUsuário` - O ID do usuário será usado para associar esta sugestão ao seu responsável. É por meio desse atributo que criaremos o nosso relacionamento 1:N, isto é, 1 usuário poderá ter N sugestões, mas 1 sugestão só pode pertencer a 1 único usuário. Para mantermos a integridade dos nossos dados, será sempre importante só permitir a inclusão de IDs de usuários se eles de fato existirem no arquivo de usuários. Também é importante assegurar que, se um usuário for excluído, todas as suas sugestões também deverão ser excluídas.
- `String produto` - Neste atributo será armazenado o nome ou a descrição do produto que o usuário gostaria de receber de seus amigos, isto é, da sua sugestão.
- `String loja` - Esta string deve conter o nome da loja ou do local onde o produto pode ser encontrado.
- `float valor` - O preço aproximado do produto deve ser armazenado neste atributo.
- `String observações` - Este atributo será uma string para qualquer tipo de observação adicional. Por exemplo, pode ser necessário especificar tamanhos ou cores dos produtos. Eventualmente, pode ser usado também para conter o endereço da página do produto em uma loja virtual.

Você deve criar uma classe Sugestão com esses atributos e com os métodos `getID()`, `setID()`, `chaveSecundaria()`, `toByteArray()` e `fromByteArray()` da mesma forma que na classe Usuário.

No gerenciamento das sugestões, não precisaremos da busca pela chave secundária, mas como esse método deve estar previsto na interface `Registro`, não dá para deixar de implementá-lo. Podemos usar, assim, o nome do produto como chave secundária. No entanto, temos um problema: o índice indireto não aceitará repetições. Você dificilmente


cadastraria duas repetições de um mesmo produto para você mesmo, mas tem uma grande chance de outro usuário cadastrar um produto de mesmo nome. Assim, podemos evitar a repetição, usando como chave secundária uma *string* que combine os quatro *bytes* do ID do usuário com os *bytes* do nome do produto. Você pode fazer algo tão simples quanto o código abaixo, em que mantemos o caráter | apenas como separador dos valores.

```
public String chaveSecundaria() {  
    return this.idUsuario + "|" + this.produto;  
}
```

## Menu principal

Antes de implementarmos as operações com sugestões, precisamos começar a criar o nosso menu principal. Você deverá criar, para a sua aplicação, um laço de mais alto nível para conter esse menu. Ele oferecerá acesso a cada uma das outras operações, inclusive às de sugestões. O seu menu principal, por enquanto, deverá ter uma tela como esta:

```
AMIGO OCULTO 1.0  
=====
```



```
INÍCIO
```

```
1) Sugestões de presentes  
2) Grupos  
3) Novos convites: 0  
  
0) Sair  
  
Opção: _
```

Por enquanto, você só precisa implementar o acesso à primeira opção, isto é, ao menu de sugestões, e a última, de saída do sistema.

## Operações com sugestões

Precisaremos implementar quatro operações relacionadas às nossas sugestões: inclusão, alteração, exclusão e listagem. Neste ponto, você já deverá ter a classe CRUD lidando com tipos genéricos, porque precisaremos dos seus métodos.

Cada usuário só poderá realizar essas operações para si mesmo. A forma de consulta das sugestões de presentes para outros usuários será tratada mais tarde, dentro dos grupos de amigo oculto.



**DICA:** Crie uma variável global para armazenar o ID do usuário que estiver usando o sistema. Assim, você agilizará as operações que dependem dessa informação.

Todas as operações deverão ser realizadas a partir do menu de sugestões, que pode ter essa aparência:

```
AMIGO OCULTO 1.0
=====

INÍCIO > SUGESTÕES

1) Listar
2) Incluir
3) Alterar
4) Excluir

0) Retornar ao menu anterior

Opção: _
```

Cada opção levará a uma operação diferente que será descrita abaixo.

## Listagem

Na maioria dos sistemas que oferecem relatórios baseados em algum tipo de filtro, é comum encontrarmos arquivos auxiliares para a geração desses relatórios. Caso contrário, seria necessário percorrer todo o arquivo principal e analisar cada um dos registros para ver qual atende aos critérios do filtro.

No nosso caso, o filtro é o atributo `idUsuário`, isto é, só listaremos as sugestões que tiverem o valor do atributo `idUsuário` igual ao ID do usuário que estiver usando o sistema.

Como estrutura de dados para esse arquivo auxiliar, usaremos uma Árvore B+. As árvores B+ são estruturas ordenadas que aceleram consideravelmente processos como as listagens.

A sua Árvore B+ deverá armazenar apenas dois valores para cada sugestão cadastrada no arquivo: o par `idUsuário` e `idSugestão`. Para que tudo funcione adequadamente, é importante que a árvore permita repetições de `idUsuário`, pois um usuário pode ter mais de uma sugestão cadastrada. No entanto, não deve ser permitida a repetição do par de chaves `[idUsuário, idSugestão]`.

Sua árvore já deve oferecer um método para retornar todos os valores de `idSugestão` cuja primeira chave seja a de um `idUsuário` especificado.

A partir dessa lista, você deve consultar cada `idSugestão` no arquivo de sugestões, usando o índice direto desse arquivo (método `read(ID)`) e apresentá-la na tela.

Os passos dessa operação, portanto, são:

1. Obter a lista de IDs de sugestões na Árvore B+ usando o ID do usuário;
2. Para cada ID nessa lista,
  1. Obter os dados da sugestão usando o método `read(ID)` do CRUD;

## 2. Apresentar os dados da sugestão na tela.

**Atenção:** as sugestões na tela devem ser apresentadas numeradas sequencialmente. O ID da sugestão e o ID do usuário não devem ser apresentados, pois esses IDs são dados para uso interno pelo sistema. Sua tela deve ficar parecida com esta:

Todas as operações deverão ser realizadas a partir do menu de sugestões, que pode ter essa aparência:

### MINHAS SUGESTÕES

1. Camisa polo branca  
Hering  
R\$ 35,00  
Tamanho P
2. Bermuda  
Taco  
R\$ 49,90  
Cor azul com bolsos laterais
3. Porta retratos  
Imaginarium  
R\$ 60,00  
Tamanho 10x15 e acabamento em vidro
4. Kit de 3 ou 4 cervejas artesanais  
Qualquer supermercado  
R\$ 69,00  
Marcas que gosto: Walls, Loba, Verace e semelhantes

Pressione qualquer tecla para continuar...

## Inclusão

Para incluir uma sugestão no arquivo, o usuário precisará informar quatro dados: nome do produto, loja em que pode ser encontrado, valor aproximado e outras observações. Depois, basta incluir esses valores e o ID do próprio usuário no arquivo usando o método *create()* CRUD.

Mas aqui há algo bem importante a ser feito. O par de chaves [`idUsuário`, `idSugestão`] deve ser incluído na árvore B+ que mantém o relacionamento entre esses valores. Caso contrário, a listagem acima não poderá ser realizada.

Assim, a sequência de passos da inclusão é:

- Solicitar o nome do produto;
  - Se o nome estiver em branco, retornar ao menu de sugestões;
- Solicitar o nome da loja;
- Solicitar o valor aproximado do produto;
- Solicitar as observações do produto;
- Solicitar a confirmação da inclusão da nova sugestão;

- Se o usuário não confirmar a inclusão, voltar ao menu de sugestões;
- Incluir a sugestão no arquivo, por meio do método *create()*, usando os dados digitados e o ID do usuário;
  - O método retornará o ID da nova sugestão;
- Incluir o par ID do usuário e ID da sugestão na árvore B+ do relacionamento.

## Alteração

Para alterarmos os dados de uma sugestão, precisamos do ID dessa sugestão. Em um sistema com interface gráfica, apresentaríamos uma lista das sugestões do usuário na tela e ele apenas clicaria na sugestão desejada. Aqui, como estamos usando uma interface textual, teremos que simular algo semelhante.

A primeira ação é listar todas as sugestões da mesma forma que foi mostrada na tela acima para a listagem. Só que, em seguida, será importante saber qual dessas sugestões o usuário deseja alterar. Assim, você deve manter uma associação (um simples vetor é suficiente) entre o número sequencial apresentado na tela e o real ID da sugestão. O usuário informará o número sequencial apresentado na tela, mas nos passos abaixo você precisará usar o ID da sugestão.

Então vamos aos passos desta operação:

1. Obter a lista de IDs de sugestões na Árvore B+ usando o ID do usuário;
2. Para cada ID nessa lista,
  1. Obter os dados da sugestão usando o método *read(ID)* do CRUD;
  2. Apresentar os dados da sugestão na tela.
3. Solicitar do usuário o número da sugestão que deseja alterar;
  1. Se o usuário digitar 0, retornar ao menu de sugestões;
4. Usando o ID da sugestão escolhida, recuperar os dados da sugestão usando o método *read(ID)* do CRUD;
5. Apresentar os dados da sugestão na tela;
6. Solicitar o novo nome do produto;
  1. Se o usuário deixar esse campo em branco, usar o valor atual;
7. Solicitar o novo nome da loja;
  1. Se o usuário deixar esse campo em branco, usar o valor atual;
8. Solicitar o novo valor do produto;
  1. Se o usuário deixar esse campo em branco, usar o valor atual;
9. Solicitar as novas observações;
  1. Se o usuário deixar esse campo em branco, usar o valor atual;
10. Se pelo menos um dos campos acima tiver sido alterado, solicitar a confirmação de alteração ao usuário;
  1. Se o usuário não confirmar a alteração, voltar ao menu de sugestões;
11. Alterar os dados da sugestão por meio do método *update()* do CRUD;
12. Apresentar mensagem de confirmação da alteração;
13. Voltar ao menu de sugestões.

Como você deve ter observado, não demos a opção de alteração do ID de sugestão ou do ID de usuário. IDs nunca são alterados. Assim, podemos alterar todos os outros dados, mas não o seu ID. Quanto ao ID do usuário, não demos a opção de sua alteração, porque cada usuário só pode gerenciar as suas próprias sugestões.

## Exclusão

A exclusão de sugestões será parecida com a alteração. Apenas não precisaremos solicitar novos dados para a sugestão. Assim, a sequência de passos será:

1. Obter a lista de IDs de sugestões na Árvore B+ usando o ID do usuário;
2. Para cada ID nessa lista,
  1. Obter os dados da sugestão usando o método *read(ID)* do CRUD;
  2. Apresentar os dados da sugestão na tela.
3. Solicitar do usuário o número da sugestão que deseja excluir;
  1. Se o usuário digitar 0, retornar ao menu de sugestões;
4. Usando o ID da sugestão escolhida, recuperar os dados da sugestão usando o método *read(ID)* do CRUD;
5. Apresentar os dados da sugestão na tela;
6. Solicitar a confirmação de exclusão ao usuário;
  1. Se o usuário não confirmar a exclusão, voltar ao menu de sugestões;
8. Excluir a sugestão por meio do método *delete()* do CRUD;
9. Excluir o par ID do usuário e ID da sugestão na árvore B+ do relacionamento;
10. Apresentar mensagem de confirmação da exclusão;
11. Voltar ao menu de sugestões.

## Exclusão de usuários

No nosso projeto, não estamos prevendo a exclusão de usuários. Caso você decida implementar essa possibilidade, a ser executada exclusivamente pelo administrador do sistema, deverá excluir também todas as sugestões vinculadas a esse usuário. Para isso, poderá usar a árvore B+ do relacionamento para obter os IDs das sugestões desse usuário específico e excluir uma a uma por meio do método *delete()* do CRUD de sugestões.