

Ao longo do semestre, implementaremos um sistema para **gestão de listas de sugestões de presentes** que as pessoas gostariam de receber. A ideia é que as pessoas possam criar listas com produtos que gostariam de ganhar de seus parentes e amigos. Cada usuário, poderá ter mais de uma lista e poderá consultar as listas de outras pessoas (mediante um identificador que deverá ser passado pelo criador da lista). Um usuário poderá criar mais de uma lista (ex.: aniversário, casamento, natal etc.) e cada lista poderá ter vários produtos. Os produtos deverão ser identificados pelo [GTIN-13](#). (antigo EAN-13). Para conhecer mais sobre esse identificador, visite a [GS1](#).

Neste primeiro trabalho prático, criaremos as listas de presentes, mas deixaremos a inclusão dos produtos nelas para o segundo trabalho prático. No entanto, como as listas são pertencentes a usuários específicos, precisaremos também criar os próprios usuários.

O ACESSO AO SISTEMA:

Para acessar o sistema, um usuário deverá estar cadastrado. O acesso será feito mediante email e senha. Caso o usuário não esteja cadastrado, deverá haver uma opção que permita o seu próprio cadastro. Assim, uma primeira tela poderia ser algo assim:

```
PresenteFácil 1.0
-----
(1) Login
(2) Novo usuário

(S) Sair

Opção: __
```

A ENTIDADE USUÁRIO:

Nossa entidade usuário precisará contar com pelo menos os seguintes atributos:

- Nome
- E-mail
- HashSenha
- PerguntaSecreta
- RespostaSecreta

Nós não armazenamos a senha de um usuário, mas o código hash retornado por ela. Como temos uma limitação de recursos a serem usados em uma interface textual, os atributos PerguntaSecreta e RespostaSecreta deverão ser usados para recuperação da senha. Lembrem-se de que toda entidade precisa de um identificador exclusivo (ID) que, como vimos nas aulas, será um número inteiro positivo sequencial (). Notem que um usuário se identificará por meio do seu email e não do seu ID. A diferença é que o email pode ser alterado, o ID não.

A ENTIDADE LISTA:

Também precisaremos cadastrar as listas de presentes. Uma lista será composta por vários produtos (que só cadastraremos no trabalho prático 2). Uma lista pertencerá a apenas um usuário, mas um usuário poderá ter várias listas. É por isso que dizemos que o relacionamento é de 1 para N (ou 1:N):

- 1 usuário tem N listas
- 1 lista pertence a 1 único usuário

Cada lista deve ter, pelo menos, os seguintes atributos (além do ID):

- Nome
- Descrição detalhada
- Data criação
- Data limite (opcional)
- Código compartilhável

Aqui, o atributo código compartilhável é algo que merece uma atenção especial. Usaremos um código alfanumérico de **10 caracteres** seguindo o padrão da biblioteca [NanoID](#), e que será gerado automaticamente pelo sistema. Quando alguém quiser mostrar uma lista a seus amigos, deverá usar esse código. Observem que ele não se confunde com o ID da lista. Finalmente, vocês também precisarão de uma chave estrangeira nessa entidade (o ID do usuário).

Minha sugestão é de que vocês façam uma organização das operações desta forma:

```
PresenteFácil 1.0
-----
> Inicio > Minhas Listas

LISTAS
(1) Aniversário 2025 – 30/07/2025
(2) Decoração da casa – 15/03/2025
(3) Dia dos pais – 20/07/2025
(4) Natal 2024 – 01/12/2024

(N) Nova lista
(R) Retornar ao menu anterior

Opção: __
```

Notem que, abaixo do título, estamos usando um *breadcrumb* para facilitar a percepção do usuário de onde ele está no sistema. Ao entrar no menu Minhas Listas, o usuário deveria ver todas as listas que ele já criou.

Para visualizar os dados (ou editar) uma lista, o usuário deve selecionar a lista desejada por meio do número no menu. Esse menu é construído a partir das listas cadastradas pelo usuário ativo. Observem que o número da lista no menu não corresponde ao ID da entidade, mas é um número sequencial a partir da **ordem alfabética das listas**. Para isso, vocês precisaram de um índice indireto baseado no nome da lista (usem uma árvore B+) que retorne todas as listas do usuário ativo. A opção N deve permitir o

cadastro de uma nova lista.

Após a seleção da lista, vocês podem apresentar uma interface como esta:

PresenteFácil 1.0

> Inicio > Minhas Listas > Aniversário 2025

CÓDIGO: tdfd9as8bp

NOME: Aniversário 2025

DESCRIÇÃO: Sugestões de presentes para o meu
aniversário, que será comemorado na cervejaria.

DATA DE CRIAÇÃO: 30/07/2025

DATA LIMITE: 31/10/2025

(1) Gerenciar produtos da lista

(2) Alterar dados da lista

(3) Excluir lista

(R) Retornar ao menu anterior

Opção: __

Neste TP1, ainda não implementaremos a opção 1 desse menu. Isso ficará para o TP2.

PROGRAMA PRINCIPAL:

O programa principal agora já deve oferecer uma interface para o usuário, por meio da qual ele possa fazer inclusões, alterações, buscas e exclusões, para todas as entidades. A sugestão é que vocês ofereçam uma interface inicial semelhante a esta:

PresenteFácil 1.0

> Inicio

(1) Meus dados

(2) Minhas listas

(3) Produtos

(4) Buscar lista

(S) Sair

Opção: __

Lembrem-se de que não colocamos código de interface com o usuário (visão) na mesma classe que o acesso aos dados (modelo). Tentaremos seguir o [padrão MVC](#). Assim, vocês deveriam criar uma classe **VisaoLista** que conteria todas as operações de entrada e de saída de dados relacionadas a listas (não inclui o menu acima). Por exemplo, vocês poderiam ter pelo menos uma função **leLista()** e outra **mostraLista()**. Finalmente, teriam uma outra classe responsável pela lógica da operação que poderia se chamar **ControleLista**. Essa última classe seria responsável pelo menu e pela lógica das operações de inclusão, alteração e exclusão, entre outras. Ela acessaria os arquivos necessários, bem como chamaria as funções da visão.

Neste TP1, a opção 3 desse menu não deve ser implementada. Isso acontecerá no TP2.

CÓDIGO QUE JÁ ESTÁ PRONTO:

Nesse projeto, vocês devem necessariamente usar o [CRUD genérico](#) que desenvolvemos em sala como base. Nosso CRUD cria registros com a seguinte estrutura:

- Lápide - Byte que indica se o registro é válido ou se é um registro excluído;
- Indicador de tamanho do registro - Número inteiro (short) que indica o tamanho do vetor de bytes;
- Vetor de bytes - Bytes que descrevem a entidade (obtido por meio do método `toByteArray()` do próprio objeto da entidade).

Além disso, vocês precisarão usar as classes [TabelaHashExtensível](#) e [Árvore B+](#) que disponibilizei para criar os índices. Não vale inventar uma nova estrutura de dados para os índices nesse projeto, ok?

O QUE DEVE SER FEITO?

- Implementar o CRUD de Usuários.
- Implementar o CRUD de Listas, assegurando que cada lista pertença a um usuário específico.
- Implementar o relacionamento 1:N com o par (idUsuario; idLista) usando a Árvore B+.
- Criar a visão e o controle de usuários. Assegurar que um usuário não possa ser excluído se alguma lista estiver vinculada a ela.
- Criar a visão e o controle de listas. Uma nova lista deverá ser automaticamente vinculada ao usuário ativo no sistema.
- Implementar a busca de listas de outras pessoas por meio do código (NanoID) que o usuário já deverá ter em mãos.

Observe que para tudo funcionar, vocês precisarão acessar os arquivos e as visões de usuários e de listas em todas as classes de controle.

FORMA DE ENTREGA

- **Código:** Vocês devem postar o seu trabalho no GitHub e enviar apenas o URL do seu projeto. Criem um repositório específico para este projeto (ao invés de mandar o repositório pessoal de algum de vocês em que estejam todos os seus códigos). Acrescentem um arquivo **readme.md** ao projeto que será o relatório do trabalho de vocês (explicado abaixo).
- **Relatório:** O relatório deve começar com a **lista dos participantes** do trabalho prático e, em seguida, ter uma descrição completa do que o sistema faz. Capturem algumas telas e citem os nomes das classes que foram criadas. Expliquem todas as operações especiais que foram implementadas. O objetivo é que vocês facilitem ao máximo a minha correção, de tal forma que eu possa entender com facilidade tudo aquilo que fizeram e dar uma nota justa. No relatório, vocês devem, necessariamente, responder ao seguinte checklist (copie as perguntas abaixo para o seu relatório e responda sim/não em frente a elas, justificando a resposta quando necessário):
 - Há um CRUD de usuários (que estende a classe `ArquivoIndexado`, acrescentando Tabelas

Hash Extensíveis e Árvores B+ como índices diretos e indiretos conforme necessidade) que funciona corretamente?

- Há um CRUD de listas (que estende a classe ArquivoIndexado, acrescentando Tabelas Hash Extensíveis e Árvores B+ como índices diretos e indiretos conforme necessidade) que funciona corretamente?
 - As listas de presentes estão vinculadas aos usuários usando o idUsuario como chave estrangeira?
 - Há uma árvore B+ que registre o relacionamento 1:N entre usuários e listas?
 - Há um CRUD de usuários (que estende a classe ArquivoIndexado, acrescentando Tabelas Hash Extensíveis e Árvores B+ como índices diretos e indiretos conforme necessidade)?
 - Há uma visualização das listas de outras pessoas por meio de um código NanoID?
 - O trabalho compila corretamente?
 - O trabalho está completo e funcionando sem erros de execução?
 - O trabalho é original e não a cópia de um trabalho de outro grupo?
- **Vídeo de demonstração:** Grave um vídeo de **até 3 minutos** (captura de tela com narração em áudio) mostrando as principais operações do seu sistema. Se o vídeo ficar grande demais para o GitHub, vocês podem publicá-lo no YouTube e compartilhar o link ou usar a própria ferramenta do Canvas para captura de vídeo.

Lembre-se de que, para essa atividade, eu avaliarei tanto o esforço quanto o resultado. Portanto, escrevam o relatório e gravem o vídeo de forma que me ajude a observar o resultado.

Atenção: As respostas incorretas ao *checklist* prejudicaram consideravelmente a nota do grupo. Se vocês disserem que fizeram algo que não foi implementado, a nota final será reduzida em 50% por resposta incorreta (duas respostas incorretas significam a nota zero). Além disso, se vocês disserem que algo está funcionando corretamente, mas a operação não funcionar direito, a nota final será reduzida em 25% por resposta incorreta. Dessa forma, quando necessário, justifiquem as respostas ao *checklist*. A falta do relatório no repositório implicará em perda de 50% dos pontos obtidos na atividade. A falta do vídeo implicará, da mesma forma, em perda de 50% dos pontos. Se os dois faltarem, a nota será, automaticamente, zero.

DISTRIBUIÇÃO DE PONTOS

Essa atividade vale 5 pontos. A rubrica de avaliação estabelece os critérios que serão usados na correção.

Atenção: o TP é específico por grupo. TPs iguais receberão a nota zero (independentemente de quem realmente fez o trabalho).

Se tiverem dúvidas sobre o trabalho a fazer, me avisem. Não deixem de observar que o URL com o código no GitHub deve ser entregue até o dia especificado na atividade.