# Gerenciador de Lojas de um Shopping Center

Um dos locais mais visitados nos finais de semana, especialmente em época de festas, são os Shoppings. Além de climatizados e comumente seguros, são locais onde é possível encontrar uma grande variedade de lojas e atividades para todas as idades.

Neste contexto, o desafio de Laboratório I será o desenvolvimento de um gerenciador de lojas em um Shopping Center.

Um Shopping pode ter diversas lojas de diferentes segmentos (vestuário, alimentação, lazer, cinema, serviços etc.). A imagem abaixo ilustra a estrutura de um Shopping para a finalidade do sistema a ser desenvolvido.


Cada "quadrado" representa um determinado espaço no Shopping, que poderá (ou não) ser alugado por alguma loja. No exemplo acima, temos um Shopping com 20 espaços para locação, nomeados de E0 a E19.

Desta forma, você deverá implementar um sistema que gerencie as lojas presentes no shopping, bem como alterações nas lojas, emissões de relatórios e diversos outros aspectos referentes ao assunto.

**ATENÇÃO**

- crie as classes, métodos, atributos e demais componentes solicitados exatamente como solicitado
- você pode criar métodos auxiliares, desde que não fuja da lógica solicitada para o problema
- o tipo das variáveis que não estiverem definidos no enunciado deve ser definido por você, levando em conta a lógica e a necessidade de armazenamento de cada variável
- você não pode criar mais atributos do que os solicitados nas classes.

## Etapa 1. Criação de classes

Inicialmente, seu sistema deve possuir as seguintes classes:

- `Loja`: uma loja possui os atributos `nome`, `quantidadeFuncionarios` e `salarioBaseFuncionario` (o nome dos atributos intuitivamente indica o que cada um deles significa). Esta classe possui os seguintes métodos:
  - **Métodos Construtores:** crie 2 construtores para a classe, sendo que um recebe parâmetros para inicializar todos os atributos e outro recebe apenas valores para inicializar o nome e a quantidade de funcionários, colocando -1 no salário base dos funcionários.
  - **Métodos de acesso:** crie os métodos de acesso (`getters` e `setters`) para todos os atributos da classe.
  - **Método `toString`:** se necessário, pesquise sobre o método `toString` e implemente-o nesta classe, retornando uma `String` formatada da forma

que você desejar, desde que contenha as informações de todos os atributos da classe.

- **Método `gastosComSalario`:** este método não recebe parâmetros e retorna quanto a loja gasta com o salário de todos os seus funcionários.

  Atente para o fato de que não é possível realizar este cálculo caso o valor do salário base seja -1. Neste caso, não realize o cálculo e retorne -1.
- **Método `tamanhoDaLoja`:** este método não recebe parâmetros e retorna um dos seguintes caracteres: 'P', caso a loja possua menos de 10 funcionários; 'M', caso a loja possua entre 10 (inclusive) e 30 (inclusive) funcionários; ou 'G', caso a loja possua mais do que 31 (inclusive) funcionários.

- `Produto`: um produto possui os atributos `nome` e `preco` (sem cedilha). Esta classe possui os seguintes métodos:
  - **Método Construtor:** crie 1 construtor que um recebe parâmetros para inicializar todos os atributos.
  - **Métodos de acesso:** crie os métodos de acesso (`getters` e `setters`) para todos os atributos da classe.
  - **Método `toString`:** se necessário, pesquise sobre o método `toString` e implemente-o nesta classe, retornando uma `String` formatada da forma que você desejar, desde que contenha as informações de todos os atributos da classe.

- `Endereco`: esta classe possui os atributos `nomeDaRua`, `cidade`, `estado`, `pais` (sem acento), `cep` (do tipo `String`), `numero` (sem acento e também do tipo `String`) e `complemento` (`String`). Esta classe possui os seguintes métodos:
  - **Método Construtor:** crie 1 construtor que um recebe parâmetros para inicializar todos os atributos.

- **Métodos de acesso:** crie os métodos de acesso (`getters` e `setters`) para todos os atributos da classe.
- **Método `toString`:** se necessário, pesquise sobre o método `toString` e implemente-o nesta classe, retornando uma `String` formatada da forma que você desejar, desde que contenha as informações de todos os atributos da classe.

- `Data`: esta classe possui os atributos `dia`, `mes` (sem acento) e `ano`, todos do tipo inteiro. Esta classe possui os seguintes métodos:
  - **Método Construtor:** crie 1 construtor que um recebe parâmetros para inicializar todos os atributos. Neste construtor, você deve validar a data informada nos parâmetros. Ou seja, o método construtor deve verificar se o dia é condizente com o mês, levando em conta, também, o fato de o ano poder ser bissexto. Por exemplo, o dia 29 para o mês 2 só pode ser atribuído em anos bissextos. Caso a data seja inválida, o método construtor deve imprimir uma mensagem de erro e alterar a data para a seguinte data padrão: 1/1/2000.
  - **Métodos de acesso:** crie os métodos de acesso (`getters` e `setters`) para todos os atributos da classe.
  - **Método `toString`:** se necessário, pesquise sobre o método `toString` e implemente-o nesta classe, retornando uma `String` que representa a data no formato dia/mês/ano.
  - **Método `verificaAnoBissexto`**: este método não recebe parâmetros e retorna verdadeiro caso o ano seja bissexto e falso caso contrário.

## Etapa 2. Associação entre classes

Atualize as classes criadas anteriormente para contemplar o que é solicitado abaixo:

- Uma `Loja` possui, além dos atributos já criados, um endereço (do tipo `Endereco`) e uma data de fundação (do tipo `Data`). Crie os métodos de acesso destes atributos. Além disso, atualize os construtores para receberem o endereço e a data de fundação da loja.

- Um `Produto` possui, além dos atributos já criados, uma data de validade (do tipo `Data`). Crie os métodos de acesso deste atributo. Além disso, atualize o construtor para receber a data de validade do produto.

- Na classe `Produto`, crie um método chamado `estaVencido`, que recebe uma data por parâmetro (objeto do tipo `Data`) e retorna verdadeiro caso o produto esteja vencido em relação a esta data ou falso caso contrário.

- Crie uma classe chamada `Principal` e, nela, coloque o método `main`. Neste método, crie um menu para ser exibido para o usuário, desta forma:

```
(1) criar uma loja
(2) criar um produto
(3) sair
```

O usuário, então, deve digitar 1 para criar um objeto do tipo `Loja`, 2 para criar um objeto do tipo `Produto` ou 3 para sair. O menu deve ser apresentado para o usuário até que ele informe o valor 3 (sair). Caso ele informe um valor inválido, imprima a mensagem "`Opção inválida`" e mostre o menu novamente, solicitando uma nova opção. Todas as informações para criação da loja e do produto devem ser solicitadas ao usuário pelo Teclado.

Depois de criados os 2 objetos corretamente, seu programa deve:

- imprimir a mensagem "`PRODUTO VENCIDO`" ou "`PRODUTO NÃO VENCIDO`" caso o produto criado esteja vencido na data de

20/10/2023 (utilizando o método criado anteriormente para isso) o imprimir as informações da loja criada.

## Etapa 3. Herança e Polimorfismo:

- Na classe `Loja`, atualize o método `toString` incluindo a informação acrescentada na etapa 2, para retornar também o endereço (do tipo `Endereco`) e a data de fundação (do tipo `Data`).

- Na classe `Produto`, atualize o método `toString` incluindo a informação acrescentada na etapa 2, para retornar também a data de validade (do tipo `Data`).

- Crie uma classe chamada `Cosmetico`, que é um tipo de loja, representando uma loja de cosméticos. As lojas de cosméticos possuem o atributo `taxaComercializacao`, do tipo `double` (além de tudo que uma `Loja` tem). Esta taxa é aplicada para as lojas que comercializam produtos de beleza. Crie um construtor para a classe, que receba informações para inicializar todos os atributos. Além disso, crie os métodos de acesso dos atributos (os métodos que ainda não existem) e sobrescreva o método `toString`, incluindo a informação específica da classe.

- Crie uma classe chamada `Vestuario`, que é um tipo de loja, representando uma loja de vestuário. As lojas de vestuários possuem como atributo (além de tudo que uma `Loja` tem) um valor booleano chamado `produtosImportados`. Este atributo indica se a loja de vestuário vende roupas importadas ou não. Crie um construtor para a classe, que receba informações para inicializar todos os atributos. Além disso, crie os métodos de acesso dos atributos (os métodos que ainda não existem) e sobrescreva o método `toString`, incluindo a informação específica da classe.

- Crie uma classe chamada `Bijuteria`, que é um tipo de loja, representando uma loja de bijuteria. As lojas de bijuteria possuem como atributo (além de tudo que uma `Loja` tem) um `double` chamado `metaVendas`, que representa a meta de vendas mensais desta loja. Crie um construtor para a classe, que receba informações para inicializar todos os atributos. Além disso, crie os métodos de

acesso dos atributos (os métodos que ainda não existem) e sobrescreva o método `toString`, incluindo a informação específica da classe.

- Crie uma classe chamada `Alimentacao`, que é um tipo de loja, representando uma loja de alimentação. As lojas de alimentação possuem como atributo (além de tudo que uma `Loja` tem) um valor do tipo `Data` chamado `dataAlvara`, que indica a data que a loja de alimentação recebeu o alvará de funcionamento. Crie um construtor para a classe, que receba informações para inicializar todos os atributos. Além disso, crie os métodos de acesso dos atributos (os métodos que ainda não existem) e sobrescreva o método `toString`, incluindo a informação específica da classe.

- Crie uma classe chamada `Informatica`, que é um tipo de loja, representando uma loja de informática. As lojas de informática possuem como atributos (além de tudo que uma `Loja` tem) um valor do tipo `double` chamado `seguroEletronicos`, que representa um seguro que lojas que vendem aparelhos eletrônicos devem pagar mensalmente. Crie um construtor para a classe, que receba informações para inicializar todos os atributos. Além disso, crie os métodos de acesso dos atributos (os métodos que ainda não existem) e sobrescreva o método `toString`, incluindo a informação específica da classe.

## Etapa 4. Arrays:

- Na classe `Loja`, crie um novo atributo, que representa os produtos que a loja possui. Este atributo deve ser chamado de `estoqueProdutos`, e é um array de `Produto`. Sempre que uma loja for criada (ou seja, nos métodos construtores), deve ser informada a quantidade máxima de produtos por parâmetro. Assim sendo, o array `estoqueProdutos` será instanciado nos construtores com o tamanho recebido por parâmetro. Naturalmente, quando uma loja for criada, o estoque de produtos não terá produtos ainda, apenas espaço necessário para armazená-los. Crie os métodos de acesso para este atributo. Atualize o método `toString`.

- Na classe `Loja`, crie os seguintes métodos:
  - `imprimeProdutos`: este método não recebe parâmetros e imprime a informação de todos os produtos da loja.

  - `insereProduto`: este método recebe um `Produto` por parâmetro e insere-o na primeira posição livre do array de estoque desta loja (ou seja, na primeira posição nula do array). O método deve retornar verdadeiro

caso o produto seja inserido no estoque ou falso caso não seja (no caso de não haver espaço).

- `removeProduto`: este método recebe uma `String` que representa o nome de um produto e remove o produto correspondente do estoque. O método retorna verdadeiro caso o produto tenha sido removido e falso caso contrário (caso não haja o produto solicitado no estoque).

- Crie uma classe chamada `Shopping`. Um `Shopping` possui os atributos `nome` (do tipo `String`), `endereco` (do tipo `Endereco`) e `lojas` (que deve ser um array de `Loja`). No construtor de `Shopping`, receba informações para inicializar o nome, o endereço e, também, receba a quantidade máxima de lojas deste shopping (do tipo inteiro). No construtor de shopping, então, instancie o array `lojas`, informando a capacidade dele de acordo com o valor recebido por parâmetro. Naturalmente, quando um `Shopping` for criado, ele não terá lojas ainda, apenas o espaço em memória necessário para armazená-las. Crie os métodos de acesso dos atributos. Implemente o método `toString` nesta classe, retornando uma `String` formatada da forma que você desejar, desde que contenha as informações de todos os atributos da classe.

- Na classe `Shopping`, crie os seguintes métodos:
  - `insereLoja`: este método recebe um objeto do tipo `Loja` por parâmetro e insere esta loja no array `lojas`, na primeira posição livre do array (ou seja, a primeira posição nula). O método retorna verdadeiro caso a loja seja inserida corretamente e falso caso contrário (ou seja, caso não haja lugar no array).

  - `removeLoja`: este método recebe uma `String` que representa o nome de uma loja e remove a loja com este nome do array `lojas`. O método retorna verdadeiro caso a loja seja removida e falso caso contrário (caso não haja a loja com o nome solicitado no array).

  - `quantidadeLojasPorTipo`: este método recebe como parâmetro uma String que indica o tipo de loja que deve ser buscado (Cosmético, Vestuário, Bijuteria, Alimentação ou Informática). Deve-se então retornar a quantidade de lojas desse tipo que existem no shopping. Caso seja recebida uma String que não corresponde a nenhuma das opções anteriores, o método retorna –1.

  - `lojaSeguroMaisCaro`: este método não recebe parâmetros e retorna a loja do tipo `Informatica` que paga o maior valor de seguro de

eletrônicos do shopping. Caso não haja lojas deste tipo, o método retorna `null`.

## O que eu devo entregar?

Ao final as quatro etapas você deve enviar no link de entrega da tarefa:

- Um arquivo compactado com a pasta do seu projeto.
  - Certifique-se de que todos os arquivos necessários para compilação do seu projeto sejam entregues.
  - Antes de enviar sua resposta, execute os validadores disponíveis para encontrar eventuais problemas.
  - O nome do arquivo compactado deve seguir o padrão abaixo. Substitua o código de exemplo pelo código da sua disciplina (verifique no título da comunidade no LXP).
    - **GR96002-00000-Nome_completo_aluno.zip**

- Um link para um vídeo de NO MÁXIMO 4 (QUATRO) MINUTOS apresentando o que você desenvolveu. Você deve, no vídeo, apresentar brevemente o código que foi feito e executando-o. Certifique-se que a imagem e sua voz estejam nítidas. Este link deve ser enviado no campo texto na entrega da tarefa.

**Orientações para a o vídeo:**

- A gravação do vídeo deve ser feita no formato screencast (https://pt.wikipedia.org/wiki/Screencast).

- Para gravar, você pode utilizar qualquer tipo de ferramenta. Alguns exemplos:
  - Teams
    - Crie uma reunião com você apenas, compartilhe sua tela e faça a gravação (assim como fazemos nas aulas).
    - Mais em: Gravar uma reunião no Teams

  - Screencast-O-Matic
    - Possui a ferramenta de screen recorder gratuita.
    - https://screencast-o-matic.com

  - Kazam Screencaster
    - Ferramenta open source, para screencast.
    - https://launchpad.net/kazam

  - Loom
    - Ferramenta web e desktop, com uma versão free, para screencast.

        - https://www.loom.com

- Disponibilize o vídeo no seu OneDrive (conta da Unisinos) e crie um link compartilhável. Certifique-se de que este link seja acessível por qualquer pessoa. Para testar, após gerar o link, abra-o em uma aba anônima do navegador. Ele deve funcionar.
    - Mais em: Criar um link compartilhável

- Antes de iniciar a gravação, crie um "roteiro" para seguir e não esquecer de falar sobre algum ponto. Comece se apresentando. 😊

- Prefira explicar com mais detalhes os métodos que possuem regras de negócios relevantes. Não perca tempo com *getters* e *setters*, por exemplo.

**Observações gerais sobre a entrega:**

- Verifique todos os arquivos que serão entregues. Não serão aceitos arquivos "complementares" após o prazo final da entrega.

- O desafio é composto pela apresentação e código. Desta forma, não serão avaliadas tarefas enviadas sem o vídeo.

- Observe o tempo limite para o vídeo (4 minutos).