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.

Próximas etapas:

- *Etapa 2. Associação entre classes*
- *Etapa 3. Herança e Polimorfismo*
- *Etapa 4. Arrays*