UNIVERSIDADE FEDERAL DO CARIRI
TECNÓLOGO EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

3ª ATIVIDADE AVALIATIVA

Professor	Ricardo Ferreira Vilela
Disciplina:	Programação Orientada a Objetos
Semestre Letivo:	2023-1

Nesta série de exercícios, você terá a chance de aprimorar suas habilidades em Programação Orientada a Objetos, com foco nos conceitos essenciais de Encapsulamento e Abstração. Os exercícios são construídos em torno de um conjunto de classes e interfaces projetadas para simular uma versão de um sistema de Biblioteca (*Library*).

Instruções Gerais:

- Certifique-se de que os nomes das classes, interfaces, atributos e funções correspondam >>>**EXATAMENTE<<<** ao texto, pois serão avaliados por meio de testes automatizados;
- Cada classe e interface solicitada deve ser criada em um arquivo separado;

Lembre-se de que o objetivo da atividade é, além da avaliação, uma prática na construção de código orientado a objetos, portanto, evite consultar soluções prontas.

EXERCÍCIOS

1) No sistema de biblioteca você deve criar uma classe chamada *Person* que não deve permitir instâncias, pois não compreende nenhum objeto concreto do sistema. A classe deve ser composta pelos seguintes atributos, os quais só devem ser acessíveis às subclasses e classes no mesmo pacote:
   - *firstname* (String)
   - *lastName* (String)
   - *dateOfBirth* (Date)

   Na classe *Person*, crie um construtor que aceita os três atributos mencionados acima e inicializa os campos correspondentes. Em seguida, forneça métodos públicos de acesso (*getters*) e métodos públicos de modificação (*setters*) para todos os atributos da classe abstrata *Person*.

UNIVERSIDADE FEDERAL DO CARIRI
TECNÓLOGO EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Dica: Os métodos de acesso e modificação seguem a seguinte estrutura:

- Método de acesso: *getNomeDoAtributo*
- Método de modificação: *setNomeDoAtributo*

2) Crie uma subclasse chamada *Author* que herda da classe *Person*. Essa classe representará os autores das obras contidas na biblioteca.

   Na classe *Author, adicione os seguintes atributos privados:*

   - *nationality* (String) para representar a nacionalidade
   - *biography* (String) para representar a biografia
   - *area* (String) para representar a área de atuação

   Após a definição dos atributos, crie um construtor que aceita todos os atributos herdados da classe *Person* e os atributos específicos da classe. A ordem dos parâmetros deve seguir a mesma ordem em que foram apresentados anteriormente, sendo primeiro os parâmetros para a superclasse e posteriormente os parâmetros da subclasse. Em seguida, forneça os métodos públicos de acesso (*getters*) e métodos públicos de modificação (*setters*) para todos os atributos da classe *Author*.

3) Como você já possui a classe abstrata *Person* com os atributos e métodos adequados, você deve criar uma subclasse chamada *Manager*. Essa classe representará o gerente da biblioteca.

   Na classe *Manager*, adicione os seguintes atributos privados:

   - *hireDate* (Date) para representar a data de contratação
   - *salary* (double) para representar o salário

   Após isso, crie um construtor para a classe *Manager* que aceita todos os atributos herdados da classe *Person* e os atributos específicos da classe. A ordem dos parâmetros deve seguir a mesma ordem em que foram apresentados anteriormente. Em seguida, forneça métodos públicos de getters e setters para os atributos *hireDate* e *salary*.

4) Conforme realizado nas classes *Manager* e *Author*, você deverá criar uma classe que representa os usuários da Biblioteca. A classe deve se chamar *User* e também deve herdar da classe *Person*. Esta classe será composta pelos seguintes atributos privados:

   - *userId* (int) para representar o identificador de usuário.
   - *address* (String) para representar o endereço do usuário.

   O construtor da classe deve inicializar todos os campos correspondentes na ordem em que são apresentados, inclusive os campos da superclasse. Além disso, também devem ser definidos os métodos públicos de *getters* e *setters* dos atributos da classe *User*.

Até este ponto, concluímos a definição das 'Pessoas' na aplicação. Utilizamos o conceito de herança para reutilizar as características comuns entre os diferentes tipos de pessoas e desenvolvemos cada tipo de acordo com suas particularidades. A seguir, avançaremos para a definição dos artefatos da biblioteca, que incluem livros, periódicos e mapas.

5) Com o intuito de definir um conjunto de regras para todos os tipos de artefatos que possam ser cadastrados na biblioteca, você deverá criar uma interface chamada *ILibraryArtifact*. Essa interface deve conter os seguintes métodos abstratos e públicos:
    - *getTitle*(): Retorna uma String representando o título do item de biblioteca.
    - *getAuthor*(): Retorna um objeto da classe *Author* representando o autor do item de biblioteca.
    - *getPublicationYear*(): Retorna um inteiro representando o ano de publicação do item de biblioteca.
    - *isAvailableForLoan*(): Retorna um valor booleano (true ou false) indicando se o item de biblioteca está disponível para empréstimo.

6) Após definidas as regras que todo artefato deve possuir, você deve partir para criação de uma classe chamada *Book* que implementa a interface *ILibraryArtifact*. A classe *Book* representa um livro em uma biblioteca, e você deve garantir que ela cumpra os requisitos da interface. Na classe *Book*, adicione os seguintes atributos privados:
    - *title* (String) para representar o título do livro.
    - *author* (Author) para representar o autor do livro.
    - *publicationYear* (int) para representar o ano de publicação do livro.
    - *available* (int) para representar a quantidade disponível de cópias do livro.
    - *genre* (String) para representar o gênero do livro.

    Em seguida, crie um construtor para a classe *Book* que inicializa todos os campos correspondentes na ordem em que são apresentados. Além disso, forneça métodos públicos de *getters* e *setters* para todos os atributos da classe *Book*.

    O método *isAvailableForLoan()*, obrigatório para classes que implementam a interface *ILibraryArtifact*, deve retornar verdadeiro (true) se houver pelo menos uma cópia disponível do livro (ou seja, se *available* for maior ou igual a 1) e falso (false) caso contrário.

7) O sistema de biblioteca também deverá armazenar mapas em sua estrutura. Para isso, crie uma classe chamada *Map* que implementa a interface *ILibraryArtifact* e apresente os seguintes atributos privados:

- *title* (String) para representar o título do mapa.
- *author* (Author) para representar o autor do mapa.
- *publicationYear* (int) para representar o ano de publicação do livro.
- *available* (int) para representar a quantidade disponível de cópias do livro.
- *scale* (int) para representar a escala do mapa.

Em seguida, crie um construtor para a classe *Map* que inicializa todos os campos correspondentes na ordem em que são apresentados. Além disso, forneça métodos públicos de *getters* e *setters* para todos os atributos da classe *Map*.

O método *isAvailableForLoan()*, obrigatório para classes que implementam a interface *ILibraryArtifact*, deve retornar verdadeiro (true) se houver pelo menos uma cópia disponível do mapa (ou seja, se *available* for maior ou igual a 1) e falso (false) caso contrário.

8) Além dos livros e mapas, o sistema deve contemplar periódicos. Para isso, crie uma classe chamada *Periodical* que também implementa a interface *ILibraryArtifact* contendo os seguintes atributos privados:

- *title* (String) para representar o título do mapa.
- *author* (Author) para representar o autor do mapa.
- *publicationYear* (int) para representar o ano de publicação do livro.
- *available* (int) para representar a quantidade disponível de cópias do livro.
- *journal* (String) para representar a escala do mapa.

Em seguida, crie um construtor para a classe *Periodical* que inicializa todos os campos correspondentes na ordem em que são apresentados. Além disso, forneça métodos públicos de *getters* e *setters* para todos os atributos da classe *Periodical*.

O método *isAvailableForLoan()*, obrigatório para classes que implementam a interface *ILibraryArtifact*, deve retornar verdadeiro (true) se houver pelo menos uma cópia disponível do periódico (ou seja, se *available* for maior ou igual a 1) e falso (false) caso contrário.

Até agora, nós definimos os 'Artefatos' na nossa aplicação. Começamos criando uma interface que estabelece os métodos que todos os artefatos devem ter. Em seguida, construímos classes para cada tipo de artefato, como livros, mapas e periódicos. Agora, vamos avançar para a definição de uma classe central para a Biblioteca. Essa classe utilizará as estruturas que definimos anteriormente para fornecer seus métodos e funcionalidades.

---

9) Nesta etapa, você irá criar a classe *Library* que representa a parte central da biblioteca. A classe *Library* tem como objetivo gerenciar livros, mapas, periódicos e os usuários da biblioteca. Esta classe deve ser responsável por registrar esses itens e fornecer informações sobre eles.

   Para isso, crie a classe chamada *Library* contemplando os seguintes atributos privados:

   - *name* (String) para representar o nome da biblioteca.
   - *manager* (Manager) para representar o gerente da biblioteca.
   - *books* (List<Book>) para armazenar a lista de livros na biblioteca.
     - ***Dica**: A interface List é implementada por várias estruturas de dados e o <> representa o tipo de dado a ser armazenado na lista. No caso atual, você deve definir a variável books como do tipo List, onde 'Book' é o tipo esperado para armazenar na lista.*
   - *maps* (List<Map>) para armazenar a lista de mapas na biblioteca.
   - *periodicals* (List<Periodical>) para armazenar a lista de periódicos na biblioteca.
   - *users* (List<User>) para armazenar a lista de usuários da biblioteca.

   Após a definição dos atributos, crie um construtor para a classe *Library* que aceite os seguintes parâmetros:

   - *name* (String)
   - *manager* (Manager)

   O construtor deve inicializar os campos correspondentes. Além disso, as listas definidas para *books*, *maps*, *periodicals* e *users* devem ser inicializadas com uma implementação da interface *List* chamada *ArrayList*. Da mesma forma que é fornecido para a interface *List*, o tipo de informação a ser adicionada deve ser especificado ao criar o *ArrayList*.

Dica: Considere um cenário em que você deseja criar uma lista para armazenar nomes de animais usando a interface *List* e a implementação *ArrayList*. Essa lista será chamada de 'animais', e os nomes de animais que esperamos armazenar têm o tipo *String*. O código resultante é o seguinte:

```
private List<String> animais;
animais = new ArrayList<String>();
```

Nesse código, estamos declarando uma lista chamada 'animais' do tipo *List* < *String* >, que pode armazenar nomes de animais. Em seguida, inicializamos essa lista como uma instância de *ArrayList* vazia."

Com o construtor da classe *Library* à sua disposição, é hora de prosseguir com a implementação dos métodos que serão disponibilizados pela biblioteca. Crie os seguintes métodos conforme as especificações a seguir:

- *registerBook*(*Book book*): Este método deve permitir o registro de um livro, recebido por parâmetro, na biblioteca. O livro será adicionado à lista de livros da biblioteca.
    - ***Dica:*** Você pode usar o método *add*() em uma lista para adicionar itens. O *ArrayList* é uma implementação da interface *List*, o que significa que você pode usá-lo para armazenar objetos, como livros. Para adicionar um livro à lista, basta chamar o método *add*() e passar, neste caso, o objeto do tipo Book como argumento.
- *registerMap*(*Map map*): Este método deve permitir o registro de um mapa, recebido por parâmetro, na biblioteca. O mapa será adicionado à lista de mapas da biblioteca.
- *registerPeriodical*(*Periodical periodical*): Este método deve permitir o registro de um periódico, recebido por parâmetro, na biblioteca. O periódico será adicionado à lista de periódicos da biblioteca.
- *registerUser*(*User user*): Este método deve permitir o registro de um usuário, recebido por parâmetro, na biblioteca. O usuário será adicionado à lista de usuários da biblioteca.
- *getUsers*(): Este método deve retornar a lista de usuários da biblioteca.
- *getBooks*(): Este método deve retornar a lista de livros da biblioteca.
- *getMaps*(): Este método deve retornar a lista de mapas da biblioteca.
- *getPeriodicals*(): Este método deve retornar a lista de periódicos da biblioteca.

- *getName*(): Este método deve retornar o nome da biblioteca.
- *getManager*(): Este método deve retornar o gerente da biblioteca.
- *setName*(String *name*): Este método deve permitir a alteração do nome da biblioteca.
- *setManager*(Manager manager): Este método deve permitir a alteração do gerente da biblioteca.