ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO
Departamento de Engenharia de Computação e Sistemas Digitais

PCS 3111 - Laboratório de Programação Orientada a Objetos para Engenharia Elétrica
2022

Aula 08 – Classes Abstratas, Herança Múltipla e Métodos e Atributos Estáticos

Atenção

- Código inicial a ser usado na resolução dos exercícios encontra-se disponível no e-Disciplinas.
- Os nomes, os atributos, os métodos, e as respectivas assinaturas das classes dadas **devem seguir o especificado** em cada exercício para fins de correção automática.
- Submeta um arquivo comprimido (faça um ".zip" – **não pode ser ".rar"**) colocando apenas os arquivos ".cpp" e ".h". Não crie pastas no "zip".
- Comente a função *main* ao submeter.

AVISO: para evitar problemas de compilação no Judge, envie os exercícios à medida que implementá-los!

Exercício 1

Implemente a classe abstrata **Item**, e faça com que **Filme** seja uma classe concreta derivada de **Item**. Passe todos os métodos públicos da classe **Filme** fornecida para a classe **Item**, exceto o método `imprimir`, que deve ser abstrato na classe **Item**. Ajuste também a localização e a visibilidade dos atributos. Note que a única funcionalidade que **Filme** deve implementar é o método `imprimir()` (não altere a implementação fornecida).

De forma análoga ao que foi feito com **Filme**, implemente a classe concreta **Episodio**, que também herda de **Item** e apenas redefine o método `imprimir()`. Esse método deve ter exatamente a mesma implementação que em **Filme**.

```
Episodio(string nome, int duracao);
virtual ~Episodio();

virtual void imprimir();
```

Por fim, implemente a classe concreta **Serie**, que também herda de **Item** e deve ser capaz de armazenar diversos episódios em um vetor. Para isso, seu construtor recebe uma quantidade máxima, que deve ser usada para alocar o vetor de ponteiros para episódios. Os métodos públicos que devem ser implementados para essa classe estão listados abaixo, mas é permitido criar métodos e atributos privados caso necessário.

```
Serie(string nome, int quantidadeMaxima);
virtual ~Serie();

bool adicionar(Episodio* ep);
Episodio** getEpisodios();
int getQuantidadeEpisodios();

void imprimir();
```

Note que o construtor não recebe o valor da duração, pois ainda não há episódios adicionados. Porém, lembre-se de que a classe possui um atributo duração (herdado de **Item**) que deve ser inicializado apropriadamente e atualizado de forma a conter a duração total de todos os episódios adicionados.

No destrutor, delete todos os atributos alocados dinamicamente e todos os episódios adicionados.

O método `adicionar()` recebe um ponteiro para um **Episodio** e deve adicioná-lo ao vetor se houver espaço. Caso tenha conseguido adicionar, retorne **true**, caso contrário, retorne **false**.

O método `getEpisodios()` deve retornar o vetor de ponteiros para **Episodio** e o método `getQuantidadeEpisodios()` deve retornar a quantidade de episódios adicionados.

O método `imprimir()` deve mostrar o nome, a duração total e o número de episódios da Serie e, em seguida, imprimir cada um dos episódios adicionados:

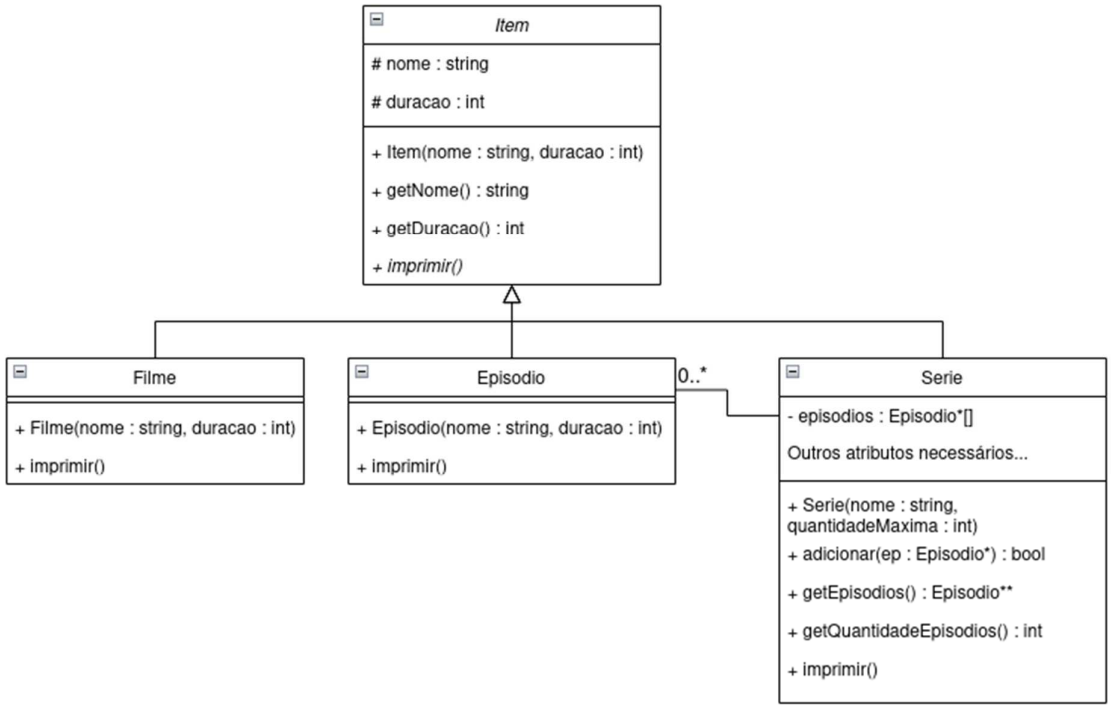
```
<nome> - <duracao> minutos - <quantidadeEpisodios> episodios
      \t1: <nome> - <duracao> minutos
      \t2: <nome> - <duracao> minutos
      …
      \t<quantidadeEpisodios>: <nome> - <duracao> minutos
```

Nota: \t representa um tab.

Por exemplo:
Silicon Valley - 90 minutos - 3 episodios
1: Minimum Viable Product - 30 minutos
2: The Cap Table - 30 minutos
3: Articles of Incorporation - 30 minutos

Dica: use a função de impressão desenvolvida para cada episódio.

Ao final, você deve ter uma estrutura de classes parecida com a mostrada no diagrama abaixo (um detalhe de notação: a linha entre **Serie** e **Episodio** é uma associação, que permite indicar que uma **Serie** possui 0 ou mais **Episodios**):



Importante: note que **Filme** e **Episodio** possuem implementações iguais, porém, ao criar um tipo específico para cada um, podemos impedir determinados comportamentos indesejáveis. Por exemplo, é impossível adicionar um Filme, uma outra Serie ou um Item qualquer à Serie, apenas objetos que foram declarados como Episodio podem ser adicionados, o que deixa o código mais seguro contra esse tipo de erro. Vale a pena testar essa parte você mesmo!

Além disso, implemente a função `teste1()` seguindo os passos descritos a seguir.

1. Crie um Filme chamado "*Luca"* de duração 95 minutos;
2. Imprima o Filme;
3. Crie uma Serie chamada "*Modern Family"* com no máximo 5 episódios;
4. Crie 5 Episodios, todos com duração de 30 minutos e com nome *Episodio <i>*, onde i vai de 1 até 5, e adicione-os à serie criada;
5. Imprima a Serie;
6. Delete todos os objetos alocados dinamicamente.

Exercício 2

Modifique a classe **Item** adicionando um *id* inteiro e sequencial a todos os itens. Para isso, crie um novo atributo id e seu getter, aqui chamado de `getId()`. O controle dos *ids* atribuídos a cada novo objeto será feito por meio de um atributo estático e de seu getter, o `getProximoId()`. Dessa forma, os novos métodos públicos que devem ser adicionados a classe **Item** são os seguintes:

```
int getId();

static int getProximoId();
```

Observação: o id do primeiro **Item** criado deve ser 0.

Implemente a função `teste2()` considerando os seguintes passos:

1. Crie o Filme *"Minari"*, de 155 minutos;
2. Crie a Serie *"His Dark Materials"*, com no máximo 5 episódios;
3. De forma análoga ao que foi feito em `teste1()`, adicione 5 episódios à Serie, mantendo o padrão de nomes ("Episodio 1", "Episodio 2", etc.), porém todos os episódios com 50 minutos;
4. Imprima o id do Filme "*Minari"* (somente o valor, pulando uma linha após);
5. Imprima o id da Serie *"His Dark Materials"* (somente o valor, pulando uma linha após);
6. Imprima o retorno de `getProximoId()` (somente o valor, pulando uma linha após);
7. Delete os objetos alocados dinamicamente.

Considere, por exemplo, o teste descrito acima. Inicialmente, o atributo estático que guarda o próximo id vale 0, logo, ao criar o filme *"Minari"*, esse novo objeto deve possuir id também igual a 0. Note que o id do filme criado é um atributo do objeto, enquanto a variável que armazena o próximo id disponível é um atributo da classe (portanto, estático).

Em seguida, a série *"His Dark Materials",* cujo id deve valer 1, é criada. Perceba que os ids são incrementados sequencialmente a medida que objetos que herdam de **Item** são criados e que dois objetos diferentes nunca podem ter o mesmo id, caso contrário, seu propósito de ser um identificador único para cada **Item** não faria mais sentido.

Testes do Judge
Exercício 1

- Item é classe abstrata;
- Filme, Episodio e Serie herdam de Item;
- Getters herdados de Item;
- Método adicionar de Serie;
- Imprimir de Filme, Episodio e Serie;
- Função `teste1`.

Exercício 2

- `getProximoId` retorna zero antes de criar qualquer objeto;
- `getProximoId` é incrementado após criar um objeto;

- `getProximoId` é incrementado após criar diversos objetos de tipos diferentes;
- `getId` ao criar um objeto;
- `getId` ao criar diversos objetos de tipos diferentes;
- Função `teste2`