Fundação CECIERJ - Vice Presidência de Educação Superior a Distância
Curso de Tecnologia em Sistemas de Computação
Disciplina: Programação com Interfaces Gráficas
AD1 2^o semestre de 2017.
Professores: Mario Benevides e Paulo Roma

---

1 Locadora de Vídeo

1.1 Introdução

O objetivo deste trabalho é proporcionar alguma experiência no uso de herança. Assim, será implementado um código para uma locadora de vídeo simples. Uma locadora de vídeo consiste em uma lista de itens (varios tipos de jogos e DVDs) e uma lista de clientes (pessoas que alugam itens). Tipos diferentes de itens podem ter políticas diferentes, quanto ao tempo de retenção, custo do aluguel, e como as taxas por atraso são calculadas. Tipos diferentes de clientes podem ter um comportamento diferente no que tange às taxas por atraso, ou outros aspectos relativos ao aluguel e retorno dos itens. Todos os custos no código devem ser expressos em centavos, por ex., onde estiver especificado $1.50, no código deve haver o valor inteiro 150.

1.2 Descrição

Aqui estão as descrições das classes envolvidas no sistema. Item oferece uma interface, na forma de uma classe abstrata, representando qualquer item na coleção da locadora. Esta interface é fornecida. Um item sempre possui um título, um gênero, e um identificador inteiro único, chamado de "barcode". A qualquer tempo, um item pode estar disponível ou alugado. Se estiver alugado, deve haver uma data de retorno. AbstractItem é uma implementação parcial da interface de um itemi, e também é fornecida para você. Devem ser implementadas três subclasses concretas de DVD: DVD, Game, e NewReleaseDVD, descritos com mairoes detalhes, abaixo.

A classe Customer representa um cliente da vídeo locadora. Esta classe está parcialmente implementada. Um cliente possui uma lista de itens alugados e um balanço devido à locadora. Alugar um item atualiza o objeto Item correspondente, incrementa o balanço do cliente pelo custo do aluguel e adiciona o item à lista do cliente.

Retornar um item atualiza o estado do item e o remove da lista do cliente. Além disso, quaisquer taxas por atraso são adicionadas ao balanço do cliente. Você deve terminar a implementação da classe Customer e implementar duas subclasses, ClubMember e PremierMember, descritos com mairores detalhes, abaixo. Uma VideoStore possui uma lista de items e uma lista de clientes. Esta classe é fornecida para você. Há métodos para adicionar items e para adicionar clientes. É possível procurar um cliente por nome, e procurar itens de acordo com algum critério arbitrário, implementando a interface SearchCondition. Não é necessário fazer nada com esta classe. Ela é fornecida para dar algum contexto de como as outras classes que estão sendo implementadas devem ser usadas. A classe abstrata SearchCondition representa um predicado de busca adaptável. Possui apenas um método:

```
def matches(self, item):
```

que retorna verdadeiro se o item dado satisfaz o critério de busca, e falso caso contrário. Por exemplo, uma condição de busca com casamento exato do gênero de um item, poderia ser escrita como:

```
class GenreSearch (SearchCondition):

    __init__(self,genre):
        self.__genre__ = genre

    matches(self,item):
        return self.__genre__ == item.getGenre()
```

A classe abstrata (parecido com uma interface do java) SearchCondition e a classe de exemplo GenreSearch são fornecidas. Você deve implementar uma classe adicional TitleKeywordSearch que implementa a classe abstrata SearchCondition. Há também um tipo Status Exception que estende a classe Exception. Esta classe é fornecida. Vários métodos são especificados para levantar uma StatusException sob certas condições. Para levantar uma exceção, deve-se construir um novo objeto de exceção. Há um argumento opcional do tipo string que é a mensagem que deve aparecer como parte do

stack trace, se a exceção ocorrer; isto é útil para debugging e estas mensagens podem ser configuradas a gosto. Por exemplo, o método setRented de um Item incluirá normalmente um teste, como descrito abaixo:

```
if (self.isRented()):
        raise StatusException("Item is already rented")
```

Neste trabalho, não é necessário capturar as exceções. Tudo o que é preciso é construí-las e levantá-las (ou simplesmente deixá-las propagarem-se, como no exemplo do método rentItem, que já está implementado na classe Customer).

1.3 Resumo do que é necessário fazer

Ao todo, devem ser implementadas sete classes:

- ClubMember
- Customer (implementada parcialmente)
- DVD
- Game
- NewReleaseDVD
- PremierMember
- TitleKeywordSearch

As sete classes abaixo são fornecidas e não devem ser modificadas, a menos da AbstractItem.

- AbstractItem
- GenreSearch
- Item
- SearchCondition
- SimpleDate

- StatusException
- VideoStore

Para AbstractItem, pode-se adicionar qualquer código adicional julgado necessário, desde que não seja adicionado nenhum método público ou construtores, e que não sejam modificados os métodos __eq__() ou __str__(). Da mesma forma, na classe Customer, não deve ser adicionado nenhum método público ou constructores.

1.4 Detalhes dos tipos para Item a serem implementados

Devem ser implementados três subclasses concretas de AbstractItem, com os seguintes comportamentos especializados:

- DVD – aluguel por 5 dias a $3.50. A taxa por atraso é $1.00 por dia. Deve haver um único construtor:

  def __init__(self, title, genre, barcode):

- NewReleaseDVD – aluguel por 2 dias a $4.00. Se for devolvido antecipadamente, o cliente ganha $1.00 de crédito no seu balanço (i.e., calculateLateFee retorna -100). A taxa por atraso é $4.00 para o primeiro dia, mais $1.50 por dia adicional (por exemplo, se for retornado 2 dias atrasado, a taxa é de $5.50.) Há um único construtor:

  def __init__(self, title, genre, barcode):

- Game – aluguel de $5.00 por uma semana. Taxa por atraso de $5.00 para cada semana ou parte de uma semana (por exemplo, se retornado com 17 dias de atraso, a taxa será de $15.00). Há um único construtor:

  def __init__(self, title, barcode):

  O método getGenre() da classe Game deve retornar sempre a string "GAME". Por isso, não é especificado gênero algum no construtor.

A classe AbstractItem deve ser uma super classe para todos os três tipos de classes, e além disso, a hierarquia de classes pode ser organizada da maneira que se quiser (e.g., NewReleaseDVD deve ser uma subclasse de DVD, ou deve ser estendida diretamente de AbstractItem?)

Lembre-se que o objetivo é usar herança para minimizar a quantidade de código duplicado entre essas classes. É possível adicionar código extra a AbstractItem, se necessário, incluindo implementatações para qualquer ou todos os métodos abstratos de um Item, ou métodos privados adicionais, ou atributos.

Não se deve adicionar qualquer método público ou construtores e não se deve modificar as implementações dos métodos mágicos __eq__ ou __str__. Lembre-se também, de não usar valores numéricos diretamente no código, como as taxas por atraso; use constantes ou variáveis definidas apropriadamente.

1.5 Detalhes dos tipos de Customer a serem implementados

A implementação da classe Customer deve ser concluída e duas subclasses concretas com o seguinte comportamento especializado. Cada subclasse possui um único construtor, com um único argumento do tipo string, com o nome do cliente.

- Customer – clientes regulares não podem alugar novos itens se possuírem algum item atrasado. As taxas por atraso devem ser sempre aplicadas (e coletados os bonus por retorno antecipado referentes a lançamentos novos).

- ClubMember – podem alugar novos items mesmo estando com algum item atrasado. Eles gozam de um dia de bonificação para retornos atrasados (para dois ou mais dias, eles pagam as mesmas taxas dos clientes regulares).

- PremierMember – podem alugar novos items mesmo possuindo items atasados, e nunca pagam taxa por atraso (embora eles ainda façam jus ao bonus por devolução antecipada dos novos lançamentos). Membros Premier acumulam 3 pontos de bonificação por cada item devolvido dentro do prazo. Dez pontos de bonus valem $1.50. Cada vez que acumularem 10 pontos de bonificação, estes devem ser convertidos automaticamente em $1.50 de crédito no balanço do cliente.

Mais uma vez, lembre-se de que o objetivo da herança é minimizar a quantidade de código duplicado entre as classes.

1.6 Detalhes para TitleKeywordSearch

Um TitleKeywordSearch possui um único construtor:

    def __init__(self, keyword, allowSubstrings):

O método matches deve retornar verdadeiro se o título do item contiver a palavra fornecida. Se o parâmetro allowSubstrings for falso, então o título é considerado casado somente se a palavra casar completamente (strings separadas por whitespace) no título, senão, matches retorna verdadeiro se a palavra aparecer em algum lugar no título, mesmo como uma substring.

Notas:

1. Todo o código deve estar dentro de um único diretório, AD1.

2. Nunca é necessário testar explicitamente o tipo de um item. Baseie-se no polimorfismo para obter um comportamento diferente para cada subclasse. Deve ser possível adicionar novos tipos de item ao sistema e o seu código deve continuar funcionando (é possível fazer isso na fase de testes).

3. Variáveis de instância (objeto) e métodos auxiliares podem ser protegidos se for necessário que as subclasses os acessem. (Tome cuidado para não redeclarar variáveis de objeto em uma subclasse, porque, em geral, irá produzir resultados inadequados).

4. Lembre-se de que a primeira coisa que uma subclasse deve fazer é chamar o construtor da sua super classe (usando a palavra reservada super). É possível chamar métodos da super classe explicitamente, usando a palavra reservada super, se necessário.

5. Não há interface com o usuário ou I/O na AD1, e nenhum código deve gerar algum tipo de saída. É necessário pensar em como testar o código e é fortemente recomendado que você pense em casos de testes desde o início. O código de teste pode ser entregue ao tutor.

6. Quando se sobre-escreve um método que já possui uma documentação Doxygen em uma super classe, em geral, não é necessário reescrever a documentação, a menos que o comportamente do método mude. Se um método for sobre-escrito e não for fornecido um comentário para o Doxygen, ele será copiado da documentação da super

classe. Se for necessário substituir a descrição de um método, apenas proveja um novo comentário no Doxygen.

1.7 Estilo e Documentação

Mais ou menos 15 a 20% dos pontos será pela documentação e estilo de programação. Parte dos pontos estará baseado em quão bem for empregada herança, para reduzir a quantidade de código duplicado.

- Cada classe, método, e variáveis de instância (objeto), tanto públicas como privadas, devem ter um comentário que faça sentido. A documentação do projeto deve incluir o tag @author, e métodos devem incluir os tags @param e @return, apropriadamente.
- Todos os nomes de variáveis devem ter algum significado (i.e., de acordo com o valor que armazenam).
- Use variáveis de instância somente para os estados permanentes do objeto. Use variáveis locais para cálculos temporários dentro dos métodos. Todas as variáveis de instância devem ser privadas.
- Use um estilo consistente para identação e formatação.

1.8 O que entregar

Por favor, submeta um arquivo .tar.gz ou zip que inclua todos os arquivos utilizados. No arquivo comprimido, todos os arquivos devem estar dentro de um diretório chamado AD1. Se houver arquivos extra, como classes de teste, estes podem ser incluídos também. Normalmente, a coisa mais simples a fazer é clicar com o botão direito do mouse no diretório fonte do seu projeto e selecionar "Send To → Compressed/zipped file". Após criar o aquivo zip, verifique-o cuidadosamente. Extraia os arquivos em um diretório temporário vazio, e olhe-os antes de submetê-los. Eles são arquivos .py? Todos os arquivos necessários estão presentes no arquivo comprimido? Estão nos diretórios corretos? Fazem parte da última versão que funciona do seu código?