Universidade Federal de Campina Grande – UFCG
Centro de Engenharia Elétrica e Informática – CEEI
Departamento de Sistemas e Computação – DSC
Disciplina: Laboratório de Programação 2

Laboratório 05 - 25/05/2015 - 10/06/2015

Neste laboratório iremos praticar polimorfismo e **tipos polimórficos por meio de Herança**. Além disso, iremos utilizar Herança para criar uma **Hierarquia de Exception** e também introduzir o uso de **Enumerations**. O principal objetivo do laboratório é entender os **benefícios e limitações** no uso de Herança e Classes Abstratas. Para praticar a **documentação alto-nível** do seu código vamos praticar também o uso de **Javadoc**, e a criação de **diagramas de classes**.

Implementação:

Você foi contratada(o) para implementar o sistema ***P2-CG: Programação 2 - Central de Games***. O P2-CG *armazena e gerencia uma coleção de jogos de um usuário*. Semelhante à plataforma **Steam**, os usuários do sistema podem comprar diferentes jogos da loja, e com isso acumular diversos pontos que fornecem benefícios e reconhecimento em meio à comunidade de jogadores da plataforma. **Forneça, testes, especificação e trate os erros por meio de Exception. Use Herança para fazer um tratamento sofisticado de Exceptions.**

Passo 1: O Jogo

Crie uma entidade Jogo que possui um **nome**, um valor real para o **preço**, e diversas informações sobre o uso do Jogo. Dentre elas: **o maior score** (pontuação) atingida pelo usuário dentre as vezes que jogou o jogo, a **quantidade de vezes** que o usuário **jogou** aquele jogo, e a quantidade de vezes que o usuário conseguiu **concluir** (**zerar**) o jogo. O jogo deve possuir o método `joga`, que recebe um inteiro referente ao *score* atingido pelo usuário, e um *boolean* indicando se o usuário conseguiu zerar o jogo. A pontuação máxima, e as quantidades de vezes que o jogo foi jogado/zerado começam com um valor zero (0). Note que o atributo de máximo *score* só é atualizado se a pontuação do usuário, ao jogar, *for maior do que a pontuação atual*. Além disso, existem **três tipos de Jogos**, o RPG (*Role Playing Game*), Luta e Plataforma.

Passo 2: Estilo de Jogo (Jogabilidade)

Cada jogo possui também um agrupamento que descreve a jogabilidade de cada Jogo. Os estilos são: *Online, Offline, Multiplayer, Cooperativo* e *Competitivo*. Qualquer jogo pode possuir **nenhum ou vários desses estilos**, porém eles não podem se repetir. A função do estilo é apenas caracterizar para o usuário a jogabilidade do jogo.

Passo 3: Usuário

Os usuários do P2-CG possuem um **nome**, um nome para realizar **login** (que funciona como um identificador), a sua **lista de jogos comprados**, e a quantidade de dinheiro que ela(e) possui para comprar jogos. Portanto, o Usuário pode: comprar jogos e adicionar mais dinheiro ao seu perfil.

Existem **dois tipos distintos** de usuários: **Noob** e **Veterano**. O Noob é o tipo de usuário **iniciante**, enquanto que o Veterano representam os usuários com mais experiência em jogos. Os tipos de usuários são utilizados para **fornecer benefícios** na compra dos jogos. Portanto, um Noob, ao comprar jogos, possui **10% de desconto** no respectivo preço do jogo. O Veterano, por sua vez, possui **20% de desconto** no preço dos jogos que compra.

Passo 4: A Loja

A loja deve possuir uma **lista de usuários** e um método que **cria jogos**. Para facilitar a implementação, cada usuário terá sua própria cópia do jogo (ou seja, um objeto). *Isso evita que dados compartilhados sejam corrompidos pelo uso simultâneo de um mesmo conjunto de dados por dois clientes distintos.* A loja também gerencia a compra de jogos, e é onde deve ocorrer toda a captura de Exceptions, e impressão de dados no console. No nosso projeto, chamamos a Loja de Fachada, ou de ***Façade***. As responsabilidades da loja são, por enquanto:

- Criar Usuários.
- Criar Jogos e adicioná-los a um usuário, se ela(e) possuir dinheiro suficiente para comprar o jogo (**Dica:** pense numa forma **desacoplada e coesa** de gerenciar essa funcionalidade… pense em ter uma **classe cuja única responsabilidade é criar os diferentes tipos de jogos.** Chamamos essa classe de **Factory** - fábrica - e ela será um atributo de Loja. Ela possuirá apenas os métodos criarJogoRPG, criarJogoLuta, criarJogoPlataforma, retornando a respectiva instância criada de Jogo.)
- Para criar use Strings na Loja (ou no Factory) para definir o tipo de jogo a ser criado. Exemplo: `loja.criaJogo("Super Mario", 40.00, "Plataforma", listaEstilo);`
- Adicionar dinheiro na conta de um usuário.

- Imprimir as Informações de todos os Usuários e seus respectivos jogos. Na impressão de usuários não precisa levar em consideração o desconto dos jogos. Porém, para a impressão da loja, é preciso ter um atributo com o total arrecadado em vendas. Esse atributo por sua vez é afetado pelo desconto resultado dos diferentes tipos de usuários. Use a seguinte formatação:

```
=== Central P2-CG ===

francisco.neto
Francisco Oliveira Neto - Jogador Noob
Lista de Jogos:
+ Magicka - RPG:
==> Jogou 5 vez(es)
==> Zerou 0 vez(es)
==> Maior score: 65478

Total de preço dos jogos: R$ 25,00

-----------------------------------------

Total arrecadado com vendas de jogos: R$ 22,50
```

Passo 5: Pontuação de Experiência de Jogadores

Em complemento com a funcionalidade de gerenciar os jogos de um usuário, o P2-CG deseja fornecer uma comunidade para os seus diversos usuários *gamers*. Portanto, o P2-CG decidiu incluir o conceito de uma **pontuação baseada em privilégios** para alguns jogadores(as) da comunidade. Essa pontuação é chamada de **x2p**, ou *eXperiente Player Priviledge*. Cada Usuário possui seus próprios pontos, que são representados por meio de uma **quantidade inteira**. Porém, existem diversas formas de **obter** e/ou **perder** esses pontos. Diversos fatores influenciam na obtenção/remoção de x2p. Dentre eles:

- Preços dos jogos que o usuário comprou.
- Tipo do jogo e seu respectivo uso.
- Desempenho do jogador de acordo com seu tipo (*Noob*, *Veterano*).
- A jogabilidade do respectivo Jogo (*Offline, Online, Competitivo, etc.*)

Todo usuário começa com zero (0) x2p. A cada jogo que compra, um usuário (independente de ser Noob ou Veterano) ganha x2p. O cálculo de x2p baseado na compra de jogos é feito da seguinte forma:

- **Cada 1 real do preço do jogo, o usuário ganha 10 pontos.** Em outras palavras, a compra irá fornecer `x2p = 10 * precoJogo` para o respectivo Usuário. Note que o precoJogo é um reflexo do preço SEM o desconto referente ao tipo de usuário.

Outra estratégia é recompensar jogadores pelo seu **frequente uso e desempenho** nos diversos tipos de jogos (RPG, Luta e Plataforma). Diante disso, crie o método '`jogar(nomeDoJogo, score,zerou)`' em **Usuario** que irá jogar um jogo (i.e., chamar o método `jogar(score, zerou)` do respectivo **Jogo**) e fornecer x2p para o Usuario. Para cada jogo, implemente as seguintes estratégias de cálculo de pontuação extra de forma que toda vez que o método 'jogar' **do respectivo Jogo** é chamado, ele retorna uma quantidade de x2p correspondente aos seguintes casos:

- **RPG:** Para cada vez que o **usuário jogou**, adicione 10 pontos extras.
- **Luta:** O *score* do usuário em um jogo de luta varia de 0 a 100.000 (**máximo**). Então ele(a) ganha **1 ponto a cada mil (1000) pontos do seu score máximo atual.** Ao atualizar o score máximo, não precisa verificar a pontuação passada, apenas some com o correspondente atual. Por exemplo: Eu ganhei 20 pontos quando atingi o score de 20.000, e agora eu atingi um novo score máximo de 40.000 pontos, me fornecendo portanto 40 pontos adicionais.
- **Plataforma:** Para cada vez que o **usuário zerou o jogo** de plataforma, ela(e) ganha **vinte (20) pontos**.

Por exemplo, considere os exemplos abaixo:

	Tipo	**Preço**	**Num. Vezes Jogou**	**Max Score**	**Num. Vezes Zerou**	**X2P Compra**	**X2P Extra**	**Total**
Super Mario World	Plataforma	30,00	5	5000	3	*300*	*60*	**360**
Guilty Gears	Luta	80,00	3	80000	2	*800*	*80*	**880**
Paper Mario	RPG	75,00	15	48000	0	*750*	*150*	**900**

OBS: Ao atualizar qualquer um dos atributos (*score máximo*, *num. de vezes que jogou* e o *num.*

de vezes que zerou), não precisa verificar a quantidade de x2p anterior, apenas some com o correspondente atual. **Por exemplo:** Eu ganhei 20 pontos quando atingi o score de 20.000, e agora eu atingi um novo score máximo de 40.000 pontos, me fornecendo portanto 40 pontos adicionais. No total o usuário terá como x2p extra: 20 + 40 = 60. Semelhantemente, se eu joguei **Paper Mario (RPG)** pela primeira vez, recebo 10 pontos. Ao jogar pela segunda vez recebo 20 pontos (totalizando em 20 + 10 = 30 x2p). **Note que o x2p (total) será armazenado no Usuario.**

Importante: Realize as modificações necessárias para **imprimir** também a quantidade de **x2p** de cada usuário, ao imprimir as informações dos usuários da loja. Um exemplo de impressão atualizada com o x2p é exibida abaixo:

```
=== Central P2-CG ===

francisco.neto
Francisco Oliveira Neto
Jogador Noob: 300 x2p
Lista de Jogos:
+ Magicka - RPG:
==> Jogou 5 vez(es)
==> Zerou 0 vez(es)
==> Maior score: 65478

Total de preço dos jogos: R$ 25,00

-------------------------------------------

Total arrecadado com vendas de jogos: R$ 22,50
```

Passo 6: Pontos Extras pela Jogabilidade

Depois de várias **reclamações** de usuários jogando com Noobs e dando *'ragequit'* nas partidas de alguns jogos, você foi chamada para **melhorar o sistema de pontuação do P2-CG**. Agora o tipo de usuário vai **ganhar ou perder** pontuação extra de acordo com o tipo do usuário jogando e a respectiva jogabilidade do jogo. Substitua o método jogar(...) em Usuário pelos seguintes métodos:

- `recompensar(String nomeJogo,int scoreObtido,boolean zerou)`: Indica que um usuário teve um bom desempenho no jogo especificado. O objetivo é **recompensar** o usuário com x2p.

- `punir(String nomeJogo, int scoreObtido, boolean zerou):` Semelhante ao método anterior porém o comportamento desse método irá **diminuir** os x2p do usuário.

O comportamento de punir/recompensar muda de acordo com o tipo do usuário. Portanto, fique atenta(o) para o polimorfismo nas subclasses e na superclasse.

Para o usuário **Noob** considere as seguintes estratégias:

- **Punir:**
  - -10 pontos se o jogo for **Online**.
  - -20 pontos se o jogo for **Competitivo**.
  - -50 pontos se o jogo for **Cooperativo** (ninguém merece Noob '*feedando*').
- **Recompensar:**
  - +30 pontos se o jogo for **Offline** (é melhor praticar offline primeiro).
  - +10 pontos se o jogo for **Multiplayer**.

Para o usuário **Veterano** considere as seguintes estratégias:

- **Punir:**
  - -20 pontos se o jogo for **Competitivo**.
  - -20 pontos se o jogo for **Cooperativo**.
- **Recompensar:**
  - +10 pontos se o jogo for **Online**.
  - +20 pontos se o jogo for **Cooperativo**.

Note que um jogo pode possuir mais de um desses estilos. Se for o caso, os pontos devem ser somados/subtraídos de forma **cumulativa**. Então, se um Usuário for punido em um Jogo Online e Cooperativo, então o total de pontos perdido é 60 (-10 - 50). Da mesma forma, se um Veterano for recompensado em um Jogo Online e Multiplayer ele ganha apenas os 10 pontos do Online.

Note também que esses dois métodos (punir/recompensar) estão ligados com o método `jogar(...)` da classe **Jogo** que também fornece pontos extras. Fique atenta(o) para **manter a integridade** no cálculo de seus x2p e de armazená-los no local correto.

Passo 7: Upgrade/Downgrade de tipo de Usuário

Chegou o momento de **recompensar aqueles usuários Noob (upgrade)** que acumularam pontos e mostraram **excelência** durante sua experiência de jogos e, semelhantemente, **punir aqueles Veteranos (downgrade)** que não estão à altura de seu título. Cada usuário poderá, a partir de

agora mudar de tipo de acordo com a quantidade de pontos atingida. **O limiar de pontos para upgrade e downgrade é: 1000 x2p**. Então se um usuário Noob atingir a meta de 1000 x2p (ou seja, seus pontos são **maiores ou iguais** a 1000 x2p), ela(e) será promovida(o) para um usuário do tipo Veterano. Semelhantemente, se um usuário Veterano reduzir sua quantidade de pontos para abaixo de (ou seja, **menor que**) 1000 x2p, então ela(e) será considerada(o) um usuário Noob. Note que essa transformação deve ser **dinâmica**, aonde os tipos de usuários devem refletir os seus respectivos pontos de experiência. Porém ela **não precisa ser automática**, ou seja, não precisa ocorrer no momento exato que atingir o limiar de x2p.

Para facilitar a implementação, crie um método na **Loja** que tenta fazer `upgrade` ou `downgrade` de usuários. Por exemplo, `loja.upgrade("francisco.neto")` tentará fazer o upgrade do usuário com login "francisco.neto", porém se o Usuário já for veterano, ou não possui a quantidade suficiente de pontos, deve ser lançada uma **Exception**. De forma semelhante faça a implementação e verificações necessárias para o método `loja.downgrade("francisco.neto");`

Para isso, você deve usar o tipo polimórfico de Herança para **gerar uma nova instância** do usuário mantendo as suas informações. **Essa etapa não pode ser feita com uma tipagem por meio de variáveis String. Isso implica na anulação do passo. Deve ser usado o tipo polimórfico de Herança.**

Passo 8: Diagrama UML de classes

Está na hora de começar a **documentar seu projeto OO por meio de um diagrama.** Nessa disciplina usaremos o diagrama de classes UML. É uma linguagem utilizada para representar **classes**, seus respectivos **componentes** e os **diversos relacionamentos** (tipos polimórficos, dependências, acoplamentos, etc.) A princípio vamos aprender o básico da representação como alguns relacionamentos (Herança), a definição dos membros de classes e suas respectivas visibilidades.

Junto com o projeto você entregará um diagrama. Existem várias ferramentas para a criação de Diagramas. Uma delas é a **Astah** que é simples de usar.

Considerações importantes para o seu projeto:

- Escreva o **Javadoc** para os seus métodos. Procure ser objetivo e comunicativo. Pratique suas habilidades de comunicação mencionando as funcionalidades do método e da classe de acordo com seus parâmetros e atributos. **Não seja óbvio, nem verborrágico**… seja **assertivo**. :)
- **Cuidado ao tratar as Exceptions.** Realize o lançamento e captura de forma adequada para não quebrar o funcionamento de seu código devido ao mau gerenciamento de

Exceções. Agrupe funções semelhantes usando uma Heirarquia de Exceptions por meio de Herança. Isso facilita a legibilidade do código e o processo de captura de Exception por `try/catch`.

- **Escreva os testes em JUnit para o seu código.** Para desenvolver o projeto em TDD, comece pelos testes da Loja. Ela é o que chamamos de Fachada (Façade) do seu projeto e é o **ponto de entrada** para a lógica de negócios (as demais classes de seu projeto).
- Faça a implementação do Lab em um **projeto no Eclipse.** Nomeie o seu projeto da seguinte forma: **Matricula_PrimeiroNome_Lab05**. Por exemplo:

`114210216_Gerson_Lab05`

A nomeação de pacotes e classes fica a seu critério. Porém, use nomes intuitivos e curtos, isso é o primeiro passo para evitar um código 'seboso'. Legibilidade é um dos critérios básicos para a avaliação.

Boa sorte e boa implementação!