**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 06 - 11/08/2016 - 21/08/2016

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. Usaremos **Javadoc** para a elaboração de **documentação alto-nível** do seu código e vamos praticar **JUnit** para criação de **testes de unidade**.

Usaremos o Git para manter seu tutor sempre atualizado sobre o que for sendo feito. **Crie um repositório remoto chamado lab06_seuNomeSeuSobrenome** (ex. lab06_RaquelLopes). **Compartilhe** esse repositório com o seu **tutor**. Você deve fazer um **push** do que você for implementando para o repositório. Faça um **push obrigatoriamente após cada aula de LP2**. Assim, seu tutor pode acompanhar o que está sendo feito.

**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, documentação e trate os erros por meio de exceções. 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 `registraJogada`, 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 sua jogabilidade e essa jogabilidade está associada aos seguintes estilos: *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 único), 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 representa 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: 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** esses pontos. Por enquanto, dois fatores influenciam a obtenção de x2p:

- preços dos jogos que o usuário comprou
- tipo do jogo e seu respectivo uso.

**O usário Noob começa com zero (0) x2p e o usuário Veterano começa com hum mil (1.000) x2p (essa estratégia é para atrair as estrelas do eSports :)).** 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 se for Noob ou 15 pontos se for veterano.** Em outras palavras, a compra irá fornecer a um usuário Noob `x2p = 10 * precoJogo` para o respectivo Usuário Noob. Se fosse um usuário veterano o cálculo seria `x2p = 15 * precoJogo` para o respectivo Usuário Veterano. 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 `registraJogada(nomeDoJogo, score,zerou)` em **Usuario** que irá registrar uma jogada de um jogo (i.e., chamar o método `registraJogada(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 `registraJogada` **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:** Cada usuário mantém o seu *score* máximo alcançado em jogos de luta. Os scores em todos os jogos de luta variam de 0 a 100.000 (**máximo**). Ao jogar um jogo de luta qualquer, o score alcançado no jogo deve ser analisado para identificar se é um novo score máximo. Caso um novo score máximo tenha sido alcançado, o/a jogador(a) ganha **1 ponto para cada mil (1000) pontos do seu score máximo alcançado.** Ao atualizar o score máximo, some à quantidade de x2p anterior a quantidade de pontos adicionais alcançados. Por exemplo: Eu acumulei 20 pontos no x2p quando atingi o score máximo de 20.000, e agora eu atingi um novo score máximo de 40.000 pontos, me fornecendo portanto 40 pontos adicionais (x2p = x2p+40).
- **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 | X2P Total |
|---|---|---|---|---|---|---|---|---|
| **Super** | Plataforma | 30,00 | 5 | 5000 | 3 | *300* | *60* | **360** |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Mario World** | | | | | | | | |
| **Guilty Gears** | Luta | 80,00 | 1 | 80000 | 1 | *800* | *80* | **880** |
| **Paper Mario** | RPG | 75,00 | 15 | 48000 | 0 | *750* | *150* | **900** |

**Reforçando:** Ao atualizar qualquer um dos atributos (*score máximo*, *num. de vezes que jogou* e o *num. de vezes que zerou*), some à quantidade de x2p anterior a quantidade de pontos adicionais alcançados. **Por exemplo:** Eu ganhei 20 pontos quando atingi o score máximo de 20.000 em um jogo de luta, 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: 20 + 40 = 60. Semelhantemente, se eu joguei **Paper Mario (RPG)** pela primeira vez, recebo 10 pontos. Ao jogar pela segunda vez recebo mais 10 pontos (totalizando em 10 + 10 = 20 x2p). **Note que o x2p total será armazenado no Usuario.**

## Passo 5: A Loja

A loja deve possuir uma **lista de usuários** e um método que **vende jogos aos usuários**. Na loja 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:

- Adicionar Usuários recebendo os usuários a serem adicionados.
- Adicionar dinheiro na conta de um usuário. Use o login do usuário para pesquisar o usuário na lista de usuários.
- Vender jogos a um usuário se ela(e) possuir dinheiro suficiente para comprar o jogo. Use o login do usuário para pesquisar o usuário na lista de usuários. 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.*
- Imprimir as Informações de todos os Usuários e seus respectivos jogos. Na impressão de usuários não precisa levar em consideração o desconto dos jogos. 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

-----------------------------------------
```

## Passo 6: Upgrade de tipo de Usuário

Chegou o momento de **recompensar aqueles usuários Noob (upgrade)** que acumularam pontos e mostraram **excelência** durante sua experiência de jogos. Cada usuário poderá, a partir de agora, mudar de tipo de acordo com a quantidade de pontos atingida. **O limiar de pontos para upgrade 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. 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 faz `upgrade` 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**.

Para isso, você deve usar o tipo polimórfico de Herança para **gerar uma nova instância** do usuário mantendo as suas informações. Você pode usar métodos acessores (gets) para obter as informações necessárias para a criação do novo usuário específico. Lembre-se que o usuário antigo deve ser removido da lista de usuários e o novo deve ser adicionado. **Nessa etapa devem existir sub-classes de Usuario, o tipo de usuário deve ser determinado pela classe do objeto Usuario (se Noob ou Veterano). A determinação do tipo de usuário não pode ser feita por um atributo String ou uma enumeração que indica o tipo de usuário. Isso implica na anulação do passo. Deve ser usado o tipo polimórfico de Herança.**

## Considerações importantes para o seu projeto:

- Todas as classes devem ter os métodos toString, equals e hashCode (lembre de usar no equals os mesmos atributos que você selecionara para usar no hashCode);
- Escreva métodos get e set necessários;
- **Escreva os testes em JUnit 4 para o seu código.**
- Escreva o **Javadoc (veja mais sobre esse assunto aqui)** 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 Hierarquia de Exceptions por meio de Herança. Isso facilita a legibilidade do código e o processo de captura de Exception por `try/catch.`
- Faça a implementação do Lab em um **projeto no Eclipse.** Nomeie o seu projeto da seguinte forma: **PrimeiroNome_Sobrenome_Matricula_Lab06**. Por exemplo:

  `Lucas_Arcoverde_MATRÍCULA_Lab06`

  **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.

## Como o seu lab será avaliado:

Serão considerados os seguintes critérios com suas respectivas notas:

- (6.5) Funcionalidade e os tipos de dados associados: observar se foram implementadas as funcionalidades pedidas, associadas aos seus respectivos tipos de dados.
  - (1.0) Hierarquia de Jogo
  - (1.0) Hierarquia de Usuário
  - (2.0) Loja
  - (1.5) X2P
  - (1.0) Upgrade de usuários
- (1.0) Testes: observar a cobertura e qualidade dos testes de unidade (tem testes para todas as classes e esses testes cobrem condições normais e limites de uso dos objetos).
- (0.5) Enumerações
- (0.5) Javadoc
- (0.5) Uso especializado de exceções: procure usar suas próprias exceções
- (1.0) Legibilidade: inclui organização do código, estilo e lógica clara

***Boa sorte e boa jornada!***