**Universidade Federal de Campina Grande – UFCG**

**Centro de Engenharia Elétrica e Informática – CEEI**

**Unidade Acadêmica de Sistemas e Computação – UASC**

Disciplina: Programação 2 e Laboratório de Programação 2

**Lista de Exercícios - Introdução à POO**

Crie código Java para as questões abaixo. Lembre de organizar seu código em pacotes para facilitar o reuso e restringir as visibilidades dos membros de suas classes. Essa lista foi dividida de acordo com seu nível de dificuldade. O Nível 1 envolve a criação de classes, o Nível 2 envolve a modificação de classes, enquanto que o Nível 3 contempla a integração de classes.

## Nível 1: Criação de Classes

1. Crie uma classe para representar **Carros** de uma locadora. Cada carro tem um **nome** e um **preço de diária** de aluguel. Faça com que clientes possam verificar o preço do aluguel de um carro por X dias.
2. Crie a representação de uma conta bancária do mundo real. Cada conta bancária tem um **nome de titular** (o dono da conta) e um **saldo**. É possível **depositar** e **sacar** dinheiro nessa conta. Fique atento que **não é possível sacar uma quantia maior** do que o **saldo** da conta.
3. Vamos iniciar o código de um **joguinho de RPG**. Uma das características principais do RPG são as **habilidades** dos heróis. Crie a classe Habilidade, que represente uma habilidade de um jogo de RPG. Ou seja, cada habilidade deve possuir: um **nome**, um inteiro que indica o **dano** da habilidade o **custo de energia** que o personagem precisa gastar para usar a habilidade (a maioria dos jogos chamam isso de **mana**). Cada habilidade pode ser **usada**, portanto, ao ser usada, a seguinte mensagem deve ser impressa:

*“Gastou <mana> de energia. <nome da habilidade> causa <dano> de dano.”*

Use o main abaixo para executar e verificar sua classe Habilidade:

|  |
| --- |
| public class CampoBatalha{  public static void main(String[] args){  Habilidade bolaDeFogo = new Habilidade(“Bola de Fogo”,60,5);  Habilidade raioDeGelo = new Habilidade(“Raio de Gelo”,20,3);  bolaDeFogo.usaHabilidade():  //será impresso:  //Gastou 5 de energia. Bola de Fogo vausa 60 de dano.  raioDeGelo.usaHabilidade();  //será impresso:  //Gastou 3 de energia. Raio de Gelo causa 20 de dano.  }  } |

1. Crie uma classe para representar uma Pessoa Esoterica. Uma pessoa esoterica tem um **nome**, um **mês de nascimento** (*1 = Janeiro, 2 = Fevereiro, …., 12 = Dezembro*) e um **elemento regente** que pode ser **Água**, **Ar**, **Fogo** e **Terra**. Esses elementos *são definidos a partir do mês de nascimento* da pessoa.

|  |  |  |  |
| --- | --- | --- | --- |
| **Terra** | **Ar** | **Água** | **Fogo** |
| Janeiro | Fevereiro | Março | Abril |
| Maio | Junho | Julho | Agosto |
| Setembro | Outubro | Novembro | Dezembro |

Esse elemento deve ser definido ***automaticamente*** ao criar uma PessoaEsoterica. Para facilitar sua implementação utilize **constantes** para os elementos e o operador “**MOD**” de Java (x **% y** : leia x MOD y) que retorna o **resto da divisão** de dois números. Por exemplo:

1. 1 % 3 = 1
2. 2 % 3 = 2
3. 3 % 3 = 0
4. 4 % 3 = 1
5. 5 % 3 = 2
6. 6 % 3 = 0
7. 7 % 3 = 1
8. 8 % 3 = 2
9. …
10. Agora você foi chamado para fazer um módulo de um **joguinho de luta**. Esse jogo é composto por lutadores, onde, cada **lutador** possui um **nome**, uma **quantidade de vida** (nos games chamamos isso de **health points** ou **HP**). Como o módulo da Luta ainda não está pronto, cada lutador pode apenas **receber dano**. Ao receber um dano específico, o lutador precisa **perder HP** correspondente ao **dano** realizado. Ao atingir **HP = 0**, o lutador deve **imprimir** uma mensagem dizendo que **está desmaiado**.

## Nível 2: Modificação de Classes

1. **Modifique** sua classe de Habilidade para permitir que cada habilidade tenha também um tipo de dano. Lembre de atualizar seu main e os demais trechos de código que necessitem de ajustes. Por exemplo:

Habilidade raioDeGelo = new Habilidade(“Raio de Gelo”,20,3, “Gelo”);

Habilidade bolaDeFogo = new Habilidade(“Bola de Fogo”,60,5, “Fogo”);

Habilidade chuteAtomico= new Habilidade(“Chute Atomico”,35,4, “Físico”);

1. **Modifique** sua ContaBancaria para permitir que o saque faça empréstimos ao banco. Ou seja, se ao tentar sacar dinheiro, a conta não possui suficiente, o saldo da conta fica negativo. Com isso, se ao realizar um depósito o saldo estiver negativo, a conta retira 10% do que foi depositado para *punir* o titular por pedir dinheiro emprestado.

## Nível 3: Integração de Classes

1. Você está irritado com a preguiça de seus colegas que prometeram entregar o módulo de Luta do seu joguinho de luta há duas semanas atrás. Hora de tentar evoluir seu programa sem eles. Reuse o seu código de Habilidade para criar um lutador com habilidade. Lembre que agora, cada lutador, ao ser criado, **vai possuir uma Habilidade**. Com isso o lutador vai poder atacar outro Lutador. Ao atacar outro Lutador o seguinte deve acontecer:

|  |
| --- |
| public class Lutador{  …  public void ataca(Lutador outroLutador){  //verifica se o Lutador atual (this) tem energia (mana)  //para usar a habilidade  //se não tiver, imprime: “não foi possível usar <nome\_habilidade>”.  //se tiver mana suficiente faça o seguinte:  //reduza a quantidade de mana utilizada.  //imprima “Atacou <nome outroLutador>.”  //usa a habilidade para imprimir as informações da habilidade.  //faz com que outroLutador receba o dano equivalente  // ao dano da habilidade.  }  } |

1. Faça um sistema de **dano crítico**. No mundo de games, uns lutadores tem uma chance de que o dano no oponente seja crítico, ou seja, cause o dobro do dano. Para isso, no seu método ataca faça uma verificação:

//Jogue uma moeda e se for cara (50%) uso o dano normal, mas se for coroa (o restante dos 50%) use o dano da habilidade e multiplique ele por 2.

Java permite a aleatoriedade isso utilizando o método estático **Math.random()**. Esse método retorna um número aleatório entre 0 e 0.9999999.. (nunca chega a 1). Então para simular a moeda você deve fazer o seguinte:

|  |
| --- |
| double moeda = Math.random();  if(moeda < 0.5){ //50% de chance  //o dano atual deve ser o dobro do valor do dano da habilidade.  //Imprima uma mensagem dizendo: "Dano crítico".  //Ou se vc for fã de games o famoso: "It's super effective!"  }else{  //o dano atual deve ser o dano normal da habilidade.  } |

1. **Vamos valorizar as habilidades do tipo Gelo.** Crie um atributo novo no lutador que se chama status. Ele só pode ter dois valores: normal e congelado. Ao ser atingido por uma habilidade de gelo, o outroLutador deve ser congelado. Um lutador congelado não pode atacar. Portanto, ao atacar, se o lutador estiver congelado, ele deve imprimir a mensagem "Lutador congelado" e não ataca. Porém, ele vai descongelar retornando seu status para normal, podendo então realizar ataques no próximo turno.
2. Note que, se vc se empolgou e começou a codificar tudo no método ataca, **o seu método ataca já está ficando grande.** Hora de melhorar o seu código! Crie métodos (private e public) dentro da classe Lutador para melhorar a legibilidade do seu metodo ataca. Pense no que pode ser **reusado** por clientes do Lutador.
3. Não pare por aqui! **Pense em novas funcionalidades** para o seu joguinho de luta e evolua criando até outros tipos de habilidades e métodos para os lutadores. Por exemplo:

* Que tal ser possível curar um lutador?
* E se os lutadores podem agora usar uma habilidade que recarrega o mana completo dele?
* E se cada lutador pode ter até 4 habilidades diferentes?

Poste suas ideias no Canvas, discuta com seus colegas sobre como implementar novas funcionalidades para seu joguinho de luta.