LP II - 2014.1 / 1ª Lista de Exercícios

Em todos os exercícios a aluno deve atender aos requisitos enunciados. Métodos e variáveis auxiliares podem ser criadas e usadas, desde que pertinentes. O aluno deve necessariamente empregar e explorar as características de orientação a objetos do Java:

- Encapsulamento (incluindo modificadores de acesso),
- Herança (de classe e interface) e polimorfismo; Classes e métodos abstratos.
- Sobrecarga de métodos;
- Tratamento e geração de Exceções;
- Uso das classes básicas (Object, por exemplo);
- Classes / pacotes

Exercício 1:

Fazer um programa que receba as coordenadas de 3 pontos como argumento de linha de comando, valores com tipo *real*, da seguinte forma (Pontos: A, B, C): **`Ax Ay Bx By Cx Cy`**

Observe que 3 pontos formam um triângulo, então para o triângulo formado, calcule: Perímetro e Área. Classifique se ele é: Eqüilátero, Isósceles ou Escaleno

Utilize tratamento de exceções para capturar problemas de entrada (número insuficiente de argumentos, argumentos que não formam um triângulo e erros de conversão [argumento(s) inválido(s))

Crie dois métodos de classe para:

i) calcular o perímetro, recebendo como parâmetros os 3 lados
ii) calcular a área, recebendo como parâmetros os 3 lados

Chame estes métodos para fazer os cálculos (tornando o programa mais modular)

Exemplos:

```
>java L1exc01 0.0 0.0 0.0 -3.0 4.0 -3.0
O Perímetro do triangulo eh: 12.0 unidades.
A area do triangulo eh: 6.0 unidades de area.
O Triangulo eh escaleno

>java L1exc01 0 0 0.0 -4.3 4.3 -4.3
O Perímetro do triangulo eh: 14.681118318204309 unidades.
A área do triangulo é: 9.245000000000003 unidades de área.
O Triangulo é isósceles.
```

Para esta 1ª experiência, o programa pode ser codificado todo no método *main* (sabendo que isso não é incentivado), como na Figura 1.

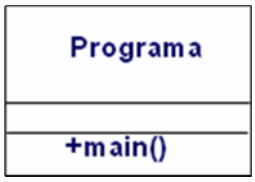

Figura 1. Diagrama de classe - Ex1

Exercício 2:

Dado um "baralho" de cartas rotuladas com uma letra, a seguinte operação se repete até que reste apenas uma carta:

- 1 A carta do topo é descartada.
- 2 A carta que agora está no topo vai para o fim do baralho.

O programa deve ler o número de cartas que vai compor o baralho, depois ler o rótulo de cada carta (ler uma letra, mas mantê-la como objeto String, para facilitar). E por fim imprimir na tela as cartas descartadas, na ordem em que foram descartadas, e a carta restante.

Modele o processo usando fila. Use a classe *java.util.LinkedList*.

Será que temos duas soluções (uma com laço interativo e outro com recursão?

Exemplo:

```
Digite o numero de cartas: 7
Digite a letra da carta #1: a
Digite a letra da carta #2: e
Digite a letra da carta #3: f
Digite a letra da carta #4: j
Digite a letra da carta #5: k
Digite a letra da carta #6: b
Digite a letra da carta #7: v

Cartas descartadas: A, F, K, V, J, E
Carta restante: B
```

A Figura 2 representa as classes da aplicação

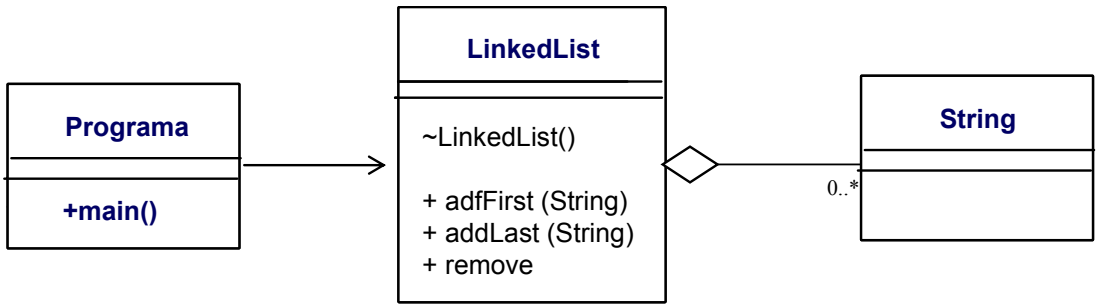

Figura 2. Diagrama de classe – Ex2

Exercício 3:

Faça um programa que leia um número em **decimal** e o converta para **binário** e vice-versa.

Uma classe chamada *Convert* será desenvolvida, contendo métodos estáticos para conversão:

- *toDec()*. O método *toDec()* deverá aceitar como argumento um objeto da classe String ou um *char* e retornar um String contendo a representação decimal do argumento passado.
- *toBin ()*.O método *toBin ()* deverá aceitar como argumento um objeto da classe String, um *char* ou um *int*, e retornar um *String* contendo a representação em binário – base 2 do argumento passado.

O programa principal fará uso da classe *Convert.*

Vamos deixar para o aluno a decisão de como informar para os métodos *toDec() e toBin()* qual a base do argumento. Pode ser até que usar outros nomes de métodos fosse mais interessantes, por exemplo, *decToBin*() ou *binToOct()*

Essa decisão vai afetar a possibilidade de reuso da classe *Convert* (já pensando no próximo exercício). Na dúvida, fale como o professor ou com a Monitora.

A entrada dedados poderá ser feita na linha de comando ou via *stream* (fluxo) – usando a receita dos slides. Faça para os dois.

Observe que os parâmetros passados como argumentos dos métodos de conversão devem ser criticados e/ou as exceções devem ser tratadas. Também, existe a necessidade do usuário passar o tipo de conversão a ser feita (fica difícil saber qual a base de entrada pela análise do conteúdo da entrada).

Exemplo:

```
Digite o tipo/base de origem (1) binário (2) decimal: 1
Digite o tipo/base de destino (1) binário (2) decimal: 2
Digite o número de origem:  1001
O valor em decimal é: 9
```

Exercício 4:

Estenda, através de herança, a classe *Convert* para fazer conversão de/para base octal. (ou seja, de Binário / Decimal / Octal – todos para todos).

Remodele o programa anterior para perguntar de que base para que base é necessária a conversão. **A idéia é usar, o máximo possível, o mecanismo da herança para reusar código.**

Dica: use uma base como "base-intermediária" para fazer as outras conversões – isso vale para os exercícios 3 e 4.

Exercício 5:

Crie a classe *Pessoa* com os campos protegidos (encapsulados), *nome* e *dataNascimento*, objetos da classe *String*, que vão representar o nome e data de nascimento. A classe *Pessoa* deve conter:

- Um construtor que recebe como parâmetros duas *strings* e inicializa os campos *nome* e *dataNascimento*.
- O método *toString*, que não recebe parâmetros e retorna um objeto da classe String na seguinte forma:

```
Nome: <nome da pessoa>
Data de Nascimento: <data de nascimento da pessoa>
```

Crie a classe abstrata *PessoaIMC* que herde da classe *Pessoa* e contenha tenha os campos protegidos *peso* e *altura*, ambos do tipo *double*. O construtor desta classe deve receber como parâmetros duas *strings* e dois valores do tipo *double* e inicializar os campos *nome*, *dataNascimento*, *peso* e *altura*. A classe *PessoaIMC* deve conter os seguintes métodos:

- *public double getPeso()* que retorna o peso;
- *public double getAltura()* que retorna a altura;
- *calculaIMC()* que recebe como parâmetros dois valores do tipo *double* que são a altura e o peso e retorna um valor do tipo *double* correspondente ao IMC (Índice de Massa Corporal = peso / altura ao quadrado) calculado.
- o método abstrato *resultIMC()* que não recebe parâmetros e retorna uma instância da classe String. (o método não é implementado nesta classe - **abstrato**)
- O método *toString()* desta classe deve retornar uma string da seguinte forma:

```
Nome: <nome da pessoa>
Data de Nascimento: <sua data de nascimento>
Peso: <seu peso>
Altura: <sua altura>
```

Crie as classes *Homem* e *Mulher*, herdeiras de *PessoaIMC*. Cada uma deve implementar o método abstrato *resultIMC()* para realiza o calculo do IMC e exibe uma mensagem de resultado acordo com o valor obtido.

Para Homem:	Para Mulher:
IMC < 20.7 : Abaixo do peso ideal 20.7 < IMC < 26.4: Peso ideal IMC > 26.4 : Acima do peso ideal	IMC < 19 : Abaixo do peso ideal 19 < IMC < 25.8: Peso ideal IMC > 25.8 : Acima do peso ideal

Crie uma classe para o programa principal, com o método *main()*, que crie instâncias das classes *Homem* e *Mulher* e armazene essas instâncias em um objeto array do tipo *PessoaIMC*. O programa deve perguntar ao usuário o tamanho do ***array***, que tipo de objeto (Homem ou Mulher) deseja criar e os dados referentes a cada objeto. A leitura de dados deve ser feita através de fluxo de entrada. Após o armazenamento de todos os objetos, o programa deve ler cada posição do *array*, imprimindo os dados do objeto ali contido e calculando seu IMC.

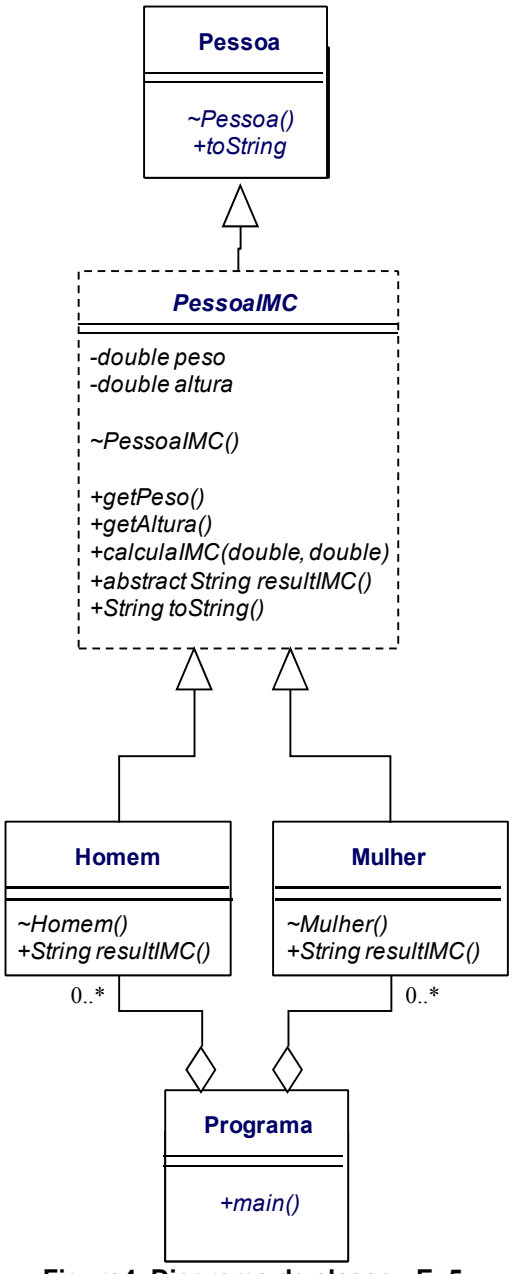

Figura4. Diagrama de classe - Ex5

Exemplo:

```
java PesoM
Digite o numero de pessoas:
2
Inserir homem (h) ou mulher(m)?
j
--- Opcao Invalida!!!
Inserir homem (h) ou mulher(m)?
h
Digite o nome:
Zezinho
Digite a data de nascimento:
01/01/1901
Digite o peso:
64.8
Digite a altura (em metros):
um m
--- A altura deve ser um numero real!!!
Digite a altura (em metros):
1.80
Inserir homem (h) ou mulher(m)?
m
Digite o nome:
Mariazinha
Digite a data de nascimento:
02/02/02/1902
Digite o peso:
64.8
Digite a altura (em metros):
1.8
--------
Nome: Zezinho
Data de Nascimento: 01/01/1901
Peso: 64.8
Altura: 1.8
IMC: 19.999999999999996  Abaixo do peso
--------
--------
Nome: Mariazinha
Data de Nascimento: 02/02/02/1902
Peso: 64.8
Altura: 1.8
IMC: 19.999999999999996  Peso ideal
--------
```

Exercício 6:

Utilize tratamento de exceções para capturar problemas de entrada (argumentos fora da pré-condição e erros de conversão [argumento(s) inválido(s))

a) Crie a interface *JogoInterface*

```
public interface JogoInterface {
   public void resetar ();
   public Placar jogar ();
   public void abortar ();
}
```

b) Crie uma classe chamada *Placar*. Esta classe será usada representar o final do jogo, o resultado ou alguma outra informação que você precise passar para a estrutura de gerenciamento de jogos.

- o construtor deve receber um objeto String, a ser atribuído para a uma variável ou campo de instância
- um campo com que terá atribuído uma constante que indica se o jogo foi ganho ou não, com métodos *set* e *get*
- outros campos e métodos que você achar necessários

c) Crie uma classe chamada *MenorMaior*, que implemente *JogoInterface*, que sorteie um número aleatório **>= 1** e **<= 100**. O usuário deve tentar adivinhar qual foi o número sorteado.

O construtor deve sortear o número aleatório e guardar numa variável de instância.
O método *resetar()* pode gerar novo número aleatório
Pense no método *abortar()*
O método *jogar()* deve entrar na rotina do jogo propriamente:

Pergunta um número para o usuário, espera a sua entrada. A entrada deve ser criticada.
Ao receber o número o programa informa o número entrado tela e em seguida:

- se o usuário não acertou o número, informa se o mesmo é MAIOR ou MENOR que o número sorteado e pede uma nova entrada/tentativa de número;
- se o usuário acertar o número sorteado, o método termina e um objeto Placar deve ser criado com a informação do resultado preenchida nos seus campos. Por exemplo, uma mensagem dizendo que o usuário venceu, o numero sorteado e o número de tentativas.
- o usuário pode desistir do jogo digitando o valor 0 (zero), então o mesmo procedimento do fim de jogo deve ser usado, criando-se um objeto Placar , com uma mensagem dizendo que o usuário perdeu, o número sorteado, e o numero de tentativas, etc. e esse objeto é o retorno do método *jogar()*.

Reforçando, o método *jogar()* retorna um objeto da classe Placar, com o resultado do jogo.

Exemplo:

```
O jogo Maior Menos comecou...

Digite um número: 50
O numero fornecido eh MAIOR que o sorteado.
Digite um número: 35
O numero fornecido eh MAIOR que o sorteado.
Digite um número: 25
O numero fornecido eh MAIOR que o sorteado.
Digite um número: 15
```

```
O numero fornecido eh MAIOR que o sorteado.
Digite um número: 5
O numero fornecido eh MENOR que o sorteado.
Digite um número: 9
O numero fornecido eh MENOR que o sorteado.
Digite um número: 12
O numero fornecido eh MAIOR que o sorteado.
Digite um número: 11
```

d) Crie uma classe chamada *CaraCoroa* que implemente a interface *JogoInterface*.

O jogo você conhece. Use os requisitos da classe *MaiorMenor* como base e desenvolva classe. Adapte o que for necessário. Mas, o uso da *JogoInterface* amarra propositalmente algumas coisas. Isso serve para fazer com que todas as classes de "jogos" possam ser usados da mesma forma por uma classe "gerenciadora de jogos".

e) Crie uma classe *Jogo*, classe principal do programa. A idéia é que o usuário vai executar esta classe e vai poder escolher o jogo, jogar ou abortar, jogar novamente o mesmo jogo, jogar outro jogo e terminar o programa.

Também, queremos mostrar que o uso de modularidade, interface e herança pode tormar a produção do programa mais ágil. Imagine por exemplo a criação de uma fábrica de jogos.

O método main deve:
- criar uma instância de objeto da classe Jogo (sim!, não vamos mais usar a execução de tudo no m;etodo main, estático – vamos criar um objeto mesmo)
- chamar o método *iniciar ()*
- chamar o método *finalizar()*

O método *iniciar()* deve:
- perguntar ao usuário qual jogo ele deseja jogar.
- criar o objeto da classe do jogo correspondente
- chamar o método *jogar()*, passando o objeto do jogo
- fazer os laços necessários para o usuário jogar novamente
- aqui, se você for "esperto" vai preparar a estrutura de "menus" para que novos jogos sejam incluídos.

O método *jogar()* do objeto da classe *Jogo* deve:
- receber como parâmetro o objeto do jogo (ou seja, (JogoInterface j))
- chamar o método *jogar()* do objeto do jogo passado como parâmetro.
- com o retorno do método anterior, deve chamar o método *exibirPlacar()*, que exibe o resultado do jogo

O método *exibirPlacar(Placar p)* recebe o objeto Placar e exibe o resultado dessa "rodada"

O método *finalizar()* fecha a aplicação. Neste método, seria possível fazer uma contabilidade de quanto tempo o usuário usou o jogo, poderia totalizar alguns resultados e até pedir uma contribuição se ele gostou de usar a aplicação.