LISTA DE REVISÃO DE JAVA

Instruções

Todos os aplicativos devem funcionar na linha de comando. Quando algum valor de entrada for solicitado, este valor deve ser passado como um parâmetro do aplicativo. Quando uma ou mais respostas forem esperadas, esta(s) deve(m) ser impressa(s) na linha de comando.

Por exemplo, se a questão pedir um aplicativo que recebe dois números e retorna a soma entre eles, este aplicativo deve funcionar sendo chamado na linha de comando da seguinte forma:

```
$ java SomaDoisNumeros 1 2
> 3
```

Você pode usar qualquer editor de texto ou IDE da sua preferência para o desenvolvimento. Seus aplicativos serão executados na JVM 11.

Você deverá entregar seu projeto através de um repositório privado no github. O professor deverá ter acesso a esse repositório.

Exercícios

1) [PopulacaoMundial] Faça uma busca para descobrir a população mundial atual e a taxa de crescimento demográfico mundial anual (dica: cheque o site – em inglês - http://www.worldometers.info/world-population/). Escreva um aplicativo da linha de comando que receba como parâmetro um inteiro que representa uma quantidade de anos e utilize esse valore para retornar a população mundial estimada depois que essa quantidade de anos informada passar. No exemplo abaixo, após **3** anos (a partir de agora), a população mundial seria de **7.444.333.222**. Você não precisa obter esses valores online. A população mundial corrente e a taxa de crescimento anual podem ser fixadas em seu programa.

```
$ java PopulacaoMundial 3
> 7.444.333.222 pessoas
```

2) [Calculadora] Escreva um aplicativo de linha de comando que receba por dois inteiros, e imprima sua soma, produto, diferença e divisão. Exiba o maior dos dois números, seguido pelas palavras "é maior". Se os números forem iguais, imprima a mensagem "Esses números são iguais".

```
$ java Calculadora 1 2
> Soma: 3
> Produto: 2
> Diferença: -1
> Divisão: 0.5
> 2 é maior
```

3) [CalculaPI] Escreva um aplicativo que calcule o valor de pi (π) , usando a fórmula abaixo. Utilize o maior denominador (o último deles) como a condição de término. Você vai passar por parâmetro o maior denominador. Utilize o valor *built-in* de pi do java, para comparar os valores e obter a diferença em %.

$$\pi = 4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} - \frac{1}{15} + \cdots\right)$$

```
$ java CalculaPI 10000
> O valor de pi é: 3,141393
> Erro: 0,006366%
```

4) [Fatura] Crie uma classe chamada Fatura para uma loja de eletrônicos e utilize-a para representar uma fatura de um item ali vendido. Uma fatura deve incluir quatro informações como variáveis de instância (privados):

- o código da fatura (tipo String)
- a descrição (tipo String)
- a quantidade comprada de um item (tipo Integer)
- e o preço por item (tipo Double)

Sua classe deve ter um **construtor** que inicializa as quatro variáveis de instância. Além disso, deve fornecer um método chamado **totalFaturado** que calcula o valor da fatura (isto é, multiplica a quantidade pelo preço por item) e depois retorna esse valor. Escreva um aplicativo de linha de comando chamado FaturaTeste que utiliza a classe Fatura e recebe os parâmetros e imprime no terminal.

```
$ java FaturaTeste 12345 Mouse 5 4.5
> Código: 12345
> Descrição: Mouse
> Quantidade: 5
> Preço Unitário: R$ 4.50
> Total: R$ 22.5
```

5) [Retangulo] Crie uma classe Retangulo com os atributos privados comprimento e largura. Cada um deles assumirá como padrão o valor 1. Forneça os métodos que calculam o **perimetro** e a **area** do retângulo. A classe deve ter métodos set e get para o comprimento e a largura. Os métodos set devem verificar se os atributos comprimento e largura são, cada um, números de ponto flutuante maiores que 0,0. Se não forem maiores, deve levantar uma exceção. Escreva um programa para testar a classe Retangulo chamado RetanguloTeste e que trate as exceções levantadas.

Com erro:

```
$ java Retangulo 0 1
> Erro: Um dos lados do retângulo é igual ou menor que zero.
```

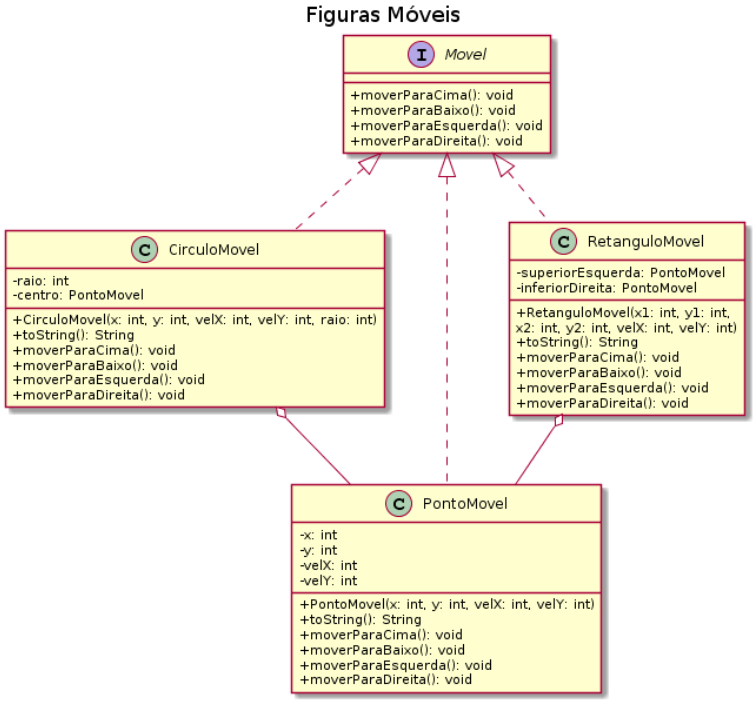
Sem erro:

```
$ java Retangulo 2.5 3
> Área: 7.5
> Perímetro: 11
```

6) [Geometria] Considere o esquema abaixo, relativo a um sistema de desenho. Esse esquema modela uma hierarquia de formas geométricas. Você deve implementar as classes conforme especificado. Note que a classe Forma é abstrata. Além dela, seus métodos getArea e getPerimetro também são abstratos. Implemente uma classe GeometriaTeste. Que recebe, respectivamente, um valor para o raio do círculo, base e altura do retângulo e lado do quadrado. Instancie cada um nesta classe, inicialize-os com os valores passados por parâmetro e imprima na tela o perímetro e área de cada um deles. Você pode considerar a cor padrão como 'Vermelho' e preenchido como 'true'.

```
$ java GeometriaTeste 2 3 4 5
> Testes realizados
>
> Área Círculo: 12.5663706144
> Perímetro Círculo: 12.5663706144
> Área Retângulo: 12
> Perímetro Retângulo: 14
> Área Quadrado: 25
> Perímetro Quadrado:  20
```

7) [FigurasMoveis] Implemente as classes do arquivo UML abaixo. Observe que as classes implementam a interface **Movel**. Esses objetos possuem um comportamento em comum. Eles podem se mover para cima, baixo, esquerda e direita. O comportamento exato depende dos próprios objetos. Modelamos esse problema utilizando a interface **Movel**. As classes que a implementarão devem definor como elas se moverão. Implemente as classes do diagrama UML abaixo. Implemente uma classe FigurasMoveisTeste que receba por parâmetro um caracter (c ou r). Caso receba **c**, leia cinco inteiros: o raio e os valores x e y do centro do círculo, a velocidadeX e a velocidadeY de movimentação. Caso receba **r** leia seis inteiros: os valores x e y dos pontos superiorEsquerda e inferiorDireita e a velocidadeX e a velocidadeY de movimentação. Movimente-os em todas as direções e imprima seus valores após cada operação. Observe o exemplo a seguir para configurar os valores de toString.




```
$ java FigurasMoveisTeste c 3 5 2 1 1
> CírculoMovel[raio=3, PontoMovel[x=5, y=2, velX=1,
velY=1]
> Movendo para cima.
> CirculoMovel[raio=3, PontoMovel[x=5, y=3, velX=1,
velY=1]
> Movendo para direita.
> CírculoMovel[raio=3, PontoMovel[x=6, y=3, velX=1,
velY=1]
> Movendo para baixo.
> CírculoMovel[raio=3, PontoMovel[x=6, y=2, velX=1,
velY=1]
> Movendo para esquerda.
> CírculoMovel[raio=3, PontoMovel[x=5, y=2, velX=1,
velY=1]
```

8) [Cartas] Implemente um aplicativo para jogar cartas.

Implemente a classe **Carta** contendo dois atributos de instância do tipo String **naipe** e **valor** que serão utilizadas para armazenar o nome do valor e o nome do naipe de uma carta específica. O construtor da classe deve receber duas Strings que ele vai utiizar para inicializar **naipe** e **valor**. Implemente um método chamado **toString** que retorna uma String com formato "Rainha de Copas".

Implemente a classe **Baralho** para gerenciar um baralho de cartas. Ela deve conter um atributo de instância chamado **monte** que use um tipo que implementa a interface List. A classe **Baralho** também deve declarar um atributo de instância do tipo Integer chamado **cartaAtual** que representa um número sequencial (0 a 51) indicando a próxima **Carta** a ser a distribuída a partir do **monte**, e uma constante com nome **TOTAL_DE_CARTAS** para indicar o número de Cartas total no baralho (52). Deve também implementar uma da constante chamada **VALORES** que contém as Strings de "Ás" a "Rei" e uma da constante **NAIPES** que contém as Strings “Paus”, “Ouros”, “Copas” e “Espadas”.

Implemente o **construtor** da classe **Baralho** onde deve ser instanciado o **monte** com tamanho **TOTAL_DE_CARTAS**. O **construtor** deve usar um *loop* para preencher o **monte** com **Carta**s. Cada **Carta** é instanciada e inicializada com duas Strings, uma da constante VALORES (que contém as Strings de "Ás" a "Rei") e uma da constante NAIPES (que contém as Strings “Paus”, “Ouros”, “Copas” e “Espadas”. Use como referência as informações da wikipedia para fazer essa implementação:
https://pt.wikipedia.org/wiki/Baralho#Baralho_francês_de_52_cartas

Implemente na classe **Baralho** o método **embaralhar** para embaralhar as Cartas do **monte**.

Implemente o método **distribuir** para distribuir uma **Carta** do **monte**. Lembre-se de que **cartaAtual** indica o índice da próxima Carta a ser distribuída, isto é, a Carta na parte superior do baralho. Se monte não estiver vazio este método retorna a Carta na parte superior e incrementa **cartaAtual** para preparar-se para a próxima chamada do método distribuir. Caso contrário, deve-se **levantar uma exceção** dizendo que não existem mais cartas no baralho.

Implemente um aplicativo de linha de comando chamado **CartasTeste** que demonstre o funcionamento da classe Baralho, embaralhando e distribuindo todas as cartas. Imprima mensagens que indiquem quando o baralho está sendo embaralhado e uma linha para cada carta distribuída.

```
$ java CartasTeste
> Embaralhando o monte!
>
> Distribuindo Valete de Paus
> Distribuindo Ás de Ouros
…
> Nenhuma carta no baralho! Fim de jogo!
```

Nesta questão você pode implementar seu próprio método de embaralhamento, ou então consultar a API do Java. Observe algumas sugestões:

```
Arrays.asList(list)
Collections.shuffle(list)
Collections.reverse(list)
Collections.copy(list, list)
Collections.fill(list, 's')
Collections.max(list)
Collections.min(list)
Collections.sort(list)
Collections.binarySearch(list, key)
```

9) [JSON] Antes de começar, leia sobre o formato JSON:
https://www.json.org
https://pt.wikipedia.org/wiki/JSON

Implemente um aplicativo de linha de comando chamado JsonTeste que recebe como parâmetro o nome de um arquivo no formato JSON e carregue este arquivo. Utilize a classe **Fatura** implementada no exercício anterior. Você vai carregar os dados do arquivo JSON como objetos do tipo **Fatura**. Retorne no terminal o valor total das faturas que estão no arquivo JSON.

Crie um aplicativo de linha de comando chamado FaturaTeste que utiliza a classe Fatura, recebe os parâmetros e imprime o total das faturas no terminal.

```
$ java JsonTeste nome_do_arquivo.json
> Total das faturas: R$ 1000.00 (20 faturas)
```

Considere como exemplo de conteúdo para o arquivo JSON a estrutura abaixo:

```
{
     "faturas": [
          {
               "codigo": "A12C",
               "descricao": "Computador",
               "quantidade", 1,
               "preco": 2021.42
          },
          {
               "codigo": "B132",
               "descricao": "Impressora",
               "quantidade", 1,
               "preco": 821.42
          },
          {
               "codigo": "D34F",
               "descricao": "Mouse",
               "quantidade", 8,
               "preco": 42.42
          }
     ]
}
```