



Exercícios

Herança

Segunda-feira, 8 de maio de 2023.

1. Resolva os exercícios abaixo do Capítulo 9 - Programação orientada a objetos: herança - do livro Java Como Programar. 6ª ed. DEITEL, H. M., DEITEL, P. J. São Paulo: Pearson Prentice Hall, 2006.

Exercícios 9.4 a 9.8.



Nota: Esses exercícios estão disponíveis também nas edições 8 e 10 do livro Java Como Programar, publicados, respectivamente, em 2010 e 2017.

2. Explique por que fazer a coerção (*cast*) de uma referência de superclasse para uma referência de subclasse é potencialmente perigoso.
3. Explique a diferença entre sobrecarregar (*overload*) e sobrescrever (*override*) um método.
4. É possível criar uma subclasse da classe `java.lang.String`? Explique sua resposta.
5. Como é possível evitar que um método seja sobrescrito em uma subclasse?
6. Utilize o projeto Java Heranca para incluir as modificações abaixo:
 - a) acrescentar as classes Paralelogramo, Losango e Trapezio;
 - b) acrescentar o método `area` para calcular a área de cada uma dessas formas geométricas usando as fórmulas abaixo:

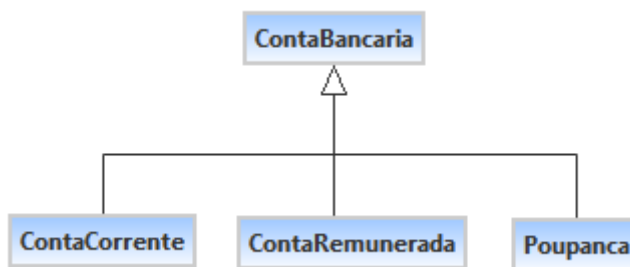
Paralelogramo: $A = \text{base} * \text{altura}$

Losango: $A = \frac{D \cdot d}{2}$, onde D = diagonal maior e d = diagonal menor

Trapézio: $A = \frac{B+b}{2} \cdot h$, onde B = base maior e b = base menor

- c) alterar a classe `FormasGeometricas` para realizar o cálculo da área dessas novas formas geométricas via a referência da superclasse `Forma`.

7. Desenvolva as classes do diagrama abaixo para permitir a movimentação de contas bancárias de três tipos: conta corrente, conta remunerada e poupança. As classes devem possuir as propriedades abaixo, além de construtores, métodos de acesso e toString.



- ContaBancaria: possui o número da agência, o número, a senha, o valor e a data de abertura da conta bancária.
- ContaCorrente: possui o número total de todas as contas correntes criadas.
- ContaRemunerada: possui rentabilidade diária somente em dias úteis.
- Poupanca: possui rentabilidade mensal a ser aplicado sobre o saldo da conta todo dia do mês correspondente a data de depósito. Por exemplo, se o depósito foi realizado no dia 20 de abril, todo o dia 20 dos meses subsequentes o saldo deve ser atualizado.

As operações bancárias de saldo, depósito, transferência e saque devem ser registradas com a data e hora de sua realização. Portanto, crie uma classe chamada OperacaoBancaria que registre a data, a hora, o tipo e o valor da operação realizada. Todas as operações efetuadas em uma conta bancária devem ser registradas para se ter todo o histórico de movimentação da conta.

As contas bancárias acima estão em um estilo em que o nome do titular da conta não é conhecido. Portanto, crie uma classe Cliente com os atributos nome e CPF e as operações de acesso. Para ajustar as contas bancárias ao estilo brasileiro use a classe Cliente para criar um relacionamento de agregação com a classe ContaBancaria, assim cada conta bancária criada conterá o nome do seu titular.

As classes ContaCorrente, ContaRemunerada e Poupanca não podem ser estendidas.

8. Usando a hierarquia de classes construída no Exercício 7 crie um aplicativo Java GUI, chamado BancoVirtual, que possui as funções abaixo.

a. Abrir conta

Lê os dados do cliente (CPF e nome), senha da conta, número da agência e cria uma conta bancária segundo o tipo escolhido pelo usuário: conta corrente, conta remunerada ou poupança. A conta corrente pode ser aberta com saldo inicial zerado, já a conta remunerada deve ter saldo inicial de R\$ 100 e a poupança de R\$ 1. O número da conta bancária deve ser gerado automaticamente pelo programa. Este número deve ser único. O número da conta deve

ter o formato: **T-DDDD**. Onde **T** é igual a 1 (Conta Corrente), 2 (Poupança) ou 3 (Conta Remunerada). **DDDD** é um número inteiro sequencial começando com 1.

A data de abertura deve ser gerada pelo programa obtendo a data do sistema operacional. Como a data do sistema pode não corresponder a data do dia de abertura da conta o usuário deve confirmar a data gerada pelo programa.

b. Depósito

Permite acrescentar um valor na conta bancária do cliente escolhida pelo usuário.

c. Saldo

Mostra o valor atualizado da conta bancária escolhida pelo usuário. O saldo deve ser atualizado para a conta remunerada e poupança de acordo com a data atual obtida do sistema operacional, a data da operação bancária e os percentuais a seguir:

Poupança = 0,5% ao mês
Conta Remunerada = 0,02% ao dia

d. Transferência

Permite transferir um valor em reais da conta corrente para a poupança ou conta remunerada.

e. Saque

Permite retirar um valor em reais da conta bancária do cliente.

f. Relatório

Exibe um relatório usando os componentes Swing JTextArea e JScrollPane para mostrar todos os dados de cada tipo de conta bancária criada para cada cliente. O saldo das contas poupança e remunerada também deve ser exibido atualizado usando as mesmas instruções do item c.

O conteúdo do relatório deve ter o leiaute abaixo. A data e a hora do cabeçalho devem ser obtidas do sistema operacional.

Data: 24/04/2015

Hora: 10:45

1-0001. Aline Silva	111.111.111-11	R\$ 850,50	12/04/2013
2-0002. João Carlos	222.222.222-22	R\$ 1.600,50	02/10/2014
3-0003. Sandra Fagundes	333.333.333-33	R\$ 3.950,50	28/09/2012
....			



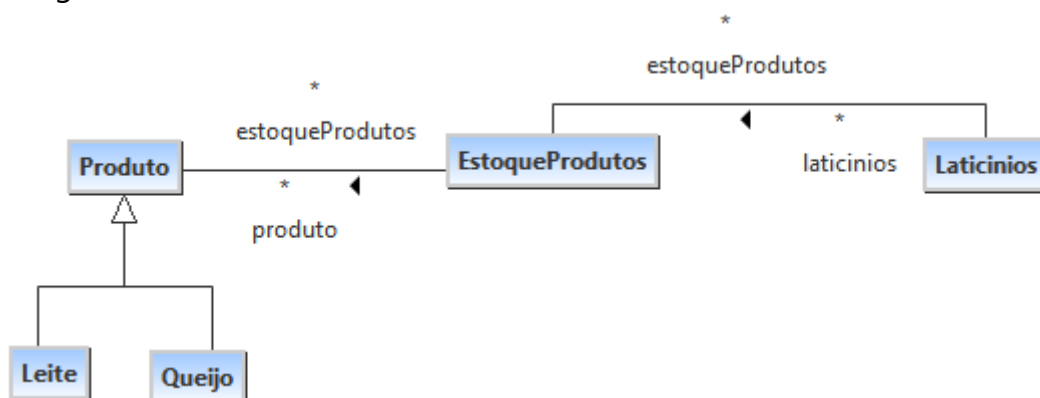
Atenção: 1. Escreva a classe ContaBancariaList para possibilitar que as operações sejam executadas em uma lista (java.util.List) de objetos do tipo ContaBancaria.

2. O programa deve criar uma única instância de `ContaBancariaList` e essa deve ser uma variável de instância criada por um método não estático da classe `BancoVirtual` chamado `criarContasBancarias`. Use a declaração abaixo.

```
public ContaBancariaList criarContasBancarias();
```

3. As operações de saldo, saque e transferência só podem ser realizadas mediante a identificação do titular com o número da agência, número e senha da conta bancária. Na operação de depósito o programa deve solicitar apenas os números da agência e da conta bancária.

9. Considere o diagrama de classes abaixo.



Desenvolva um aplicativo Java GUI chamado `Laticinios` que forneça os seguintes serviços:

a. Cadastrar Leite

Cadastra os seguintes dados do leite:

- a) tipo de leite (fresco ou gelado);
- b) volume de leite recebido (litros);
- c) data de recebimento;
- d) preço do litro de leite.

Os dados do leite recebido em um determinado dia devem ser cadastrados no estoque de produtos como um objeto da subclasse `Leite`.

b. Cadastrar Queijo

Realiza o cadastro dos seguintes dados do queijo:

- a) tipo de queijo;
- b) quantidade, em quilos (Kg), de queijos produzidos;
- c) data de fabricação;
- d) preço unitário, ou seja, preço de 1 Kg de queijo.

Considere que são necessários 10 litros de leite para produzir 1 Kg de queijo e que a produção sempre utiliza o leite recebido no dia, ou seja, a produção diária de queijos utiliza somente o

volume de leite recebido no dia. Por exemplo, o volume total de leite recebido na segunda é consumido completamente na produção de queijos desse dia.

Os tipos de queijo fabricados neste laticínio são:

- Muçarela;
- Minas Padrão;
- Parmesão;
- Prato;
- Frescal.

Validar se a quantidade do produto informada pelo usuário é possível, ou seja, se o volume de leite disponível no estoque é suficiente para atender a fabricação desse produto.

Os dados de cada tipo de queijo devem ser cadastrados no estoque de produtos como um objeto da subclasse Queijo.

c. Vendas

Permite realizar a venda de um queijo. Os dados das vendas são:

- a) código da venda;
- b) tipo de queijo vendido;
- c) quantidade, em quilos (Kg), de cada tipo de queijo;
- d) data e hora da venda;
- e) nome do cliente.

d. Relatórios

Gera os seguintes tipos de relatório:

d.1 - Relatório Geral de Produção

Este relatório deve exibir os seguintes dados para cada tipo de queijo que está armazenado no estoque de produtos:

- quantidade produzida (em Kg);
- volume total de leite (em litros) usado na produção;
- custo total (em reais) de produção.

No cálculo do custo total de produção considerar o preço do litro de leite e a quantidade de queijos produzidos.


d.2 - Relatório Diário de Produção

Este relatório deve exibir os mesmos dados do Relatório Geral de Produção (item d.1) para cada tipo de queijo que está armazenado no estoque de produtos que possui a data de fabricação igual a data informada pelo usuário.

d.3 - Relatório de Vendas

Este relatório deve exibir os seguintes dados para cada venda:

- código, data e hora da venda;
- nome do cliente;
- relação de queijos vendidos, incluindo para cada queijo: o tipo, a quantidade vendida, o preço unitário e o preço total;
- valor total da venda.

 **Atenção:** 1. Os relatórios devem usar os componentes Swing JTable e JScrollPane.

2. A classe `EstoqueProdutos` deve usar uma lista (`java.util.List`) de objetos da superclasse `Produto` para permitir a inserção e consulta de leite (`Leite`) e queijo (`Queijo`), além de obter o número desses produtos na lista. Portanto, observe que essa classe deve fornecer apenas os três métodos abaixo.

```
// Insere um produto no estoque.  
public void inserir(Produto produto);  
  
/* Obtém um produto do estoque de acordo com o seu tipo. Se tipo for queijo  
   retorna um objeto Queijo, se for leite fresco ou gelado retorna um objeto Leite.  
   Este método deve permitir que seja obtido todos os produtos do estoque que  
   possuem o mesmo tipo, p. ex., todos os queijos do tipo parmesão.  
*/  
public Optional<Produto> obter(String tipoProduto);  
  
// Informa o tamanho (número de produtos) do estoque.  
public int tamanho();
```

A classe `java.util.Optional` representa um objeto que pode ter um valor opcional, ou seja, um valor não nulo que pode ou não está presente no objeto.

3. O programa deve criar uma única instância da classe `EstoqueProdutos` e essa deve ser uma variável de instância criada por um método não estático da classe `Laticinios` chamado `criarEstoqueProduto`. Use a declaração abaixo.

```
public EstoqueProdutos criarEstoqueProduto();
```

10. Faça a refatoração do programa `CalculoInvestimentos` desenvolvido no Exercício 15 da lista sobre Classes e Objetos incluindo as subclasses `RendaFixa` e `RendaVariavel` como especializações da superclasse `Investimento`. Inclua nessa classe a implementação do método `toString` conforme está explicado no comentário a seguir.

/ Retorna uma string que representa os dados do investimento. Usa o formato abaixo:*

<nome> | <rating> | <estratégia> | FGC: <sim | não>

Veja nos exemplos abaixo como os valores devem substituir as strings no formato acima representadas

por <>. O caractere pipe (|) entre os parênteses angulares < > significa ou, por exemplo: <sim | não> resulta em sim ou não.

Exemplos:

CDB Original Jan/2020 | BBB+ | Prefixado | FGC: sim
Ações da Vale | AA+ | Renda Variável | FGC: não

*/

```
public String toString();
```

Inclua na subclasse RendaFixa a implementação dos métodos abaixo.

/* Retorna uma string que representa os dados do investimento. Usa o formato abaixo:

<nome> | <rating> | <estratégia> | FGC: <sim | não> | Prazo: <X> anos | <Y> meses

Veja nos exemplos abaixo como os valores devem substituir as strings no formato acima representadas por <>. O caractere pipe (|) entre os parênteses angulares < > significa ou, por exemplo: <sim | não> resulta em sim ou não.

Na renda fixa o prazo deve ser exibido em meses se o período de investimento for inferior a um ano (até 12 meses), caso contrário, em anos.

Exemplos:

CDB PAN Nov/2025 | BBB+ | Pós-fixado | FGC: sim | Prazo: 4 anos
CRI WS | AA+ | Prefixado | FGC: não | Prazo: 11 meses

*/

```
public String toString();
```

/* Retorna o prazo em meses se o período de investimento for inferior a um ano (12 meses), caso contrário, em anos.

*/

```
public int getPrazo();
```

Inclua na subclasse RendaVariavel a implementação dos métodos abaixo.

/* Retorna uma string que representa os dados do investimento. Usa o formato abaixo:

<nome> | <rating> | <estratégia> | FGC: <sim | não> | Prazo: <X> anos | <Y> dias

Veja nos exemplos abaixo como os valores devem substituir as strings no formato acima representadas por <>. O caractere pipe (|) entre os parênteses angulares < > significa ou, por exemplo: <sim | não> resulta em sim ou não.

Na renda variável o prazo deve ser exibido em dias se o período de investimento for inferior a um ano (até 252 dias úteis), caso contrário, em anos.

Exemplos:

CS Saúde Digital | AA+ | Alternativos | FGC: não | Prazo: 5 anos
Trend Bolsa Americana | A+ | Internacional | FGC: não | Prazo: 180 dias

*/

```
public String toString();
```

/* Retorna o prazo em dias se o período de investimento for inferior a um ano (252 dias úteis), caso contrário, em anos.

*/

public int getPrazo();

Prof. Márton Oliveira da Silva
marlon.silva@ifsudestemg.edu.br