Exercícios[1]

Classes e Objetos

Quinta-feira, 27 de abril de 2023.

---

1. Crie uma classe denominada Signo para representar o signo do zodíaco. A tabela abaixo mostra o último dia de cada mês e o signo correspondente.

Mês	Último dia	Signo
1	20	Capricórnio
2	19	Aquário
3	20	Peixes
4	20	Áries
5	20	Touro
6	20	Gêmeos
7	21	Câncer
8	22	Leão
9	22	Virgem
10	22	Libra
11	21	Escorpião
12	21	Sagitário

2. Faça um programa Java GUI, chamado Zodiaco, usando a classe Signo, criada no Exercício 1, para exibir o signo de uma pessoa após ler sua data de nascimento. O programa deve possuir uma função que permita o usuário consultar os doze signos do zodíaco em uma caixa de diálogo via componente javax.swing.JTable.

3. Crie uma enumeração denominada Signo para representar os 12 signos do zodíaco listados na tabela do Exercício 1. Em seguida, modifique o programa Zodiaco, criado no Exercício 2, para exibir o signo de uma pessoa após ler a sua data de nascimento. A alteração de Signo de classe para enumeração gerou muitas alterações no programa Zodiaco ao usar a enumeração Signo? É possível reduzir essas alterações a zero? Se sim, a refatoração[2] necessária deve ser implementada na enumeração ou na classe Signo?

4. Crie uma classe chamada PlanoCartesiano usando uma composição de objetos. Os objetos compostos são da classe Ponto. A classe Ponto deve ter os seguintes métodos: 2 construtores, 2 métodos set, 2 métodos get e toString. A classe PlanoCartesiano deve possuir dois eixos, x e y, onde cada um pode armazenar valores positivos, negativos e zero. A classe PlanoCartesiano deve

[1] Atualizado em 16/05/2023.
[2] Refatoração é uma técnica para melhorar a estrutura interna do código de um sistema de *software* sem alterar seu comportamento externo.

implementar os métodos necessários para a sua funcionalidade, incluindo construtores, métodos de definição (*set*), de obtenção (*get*) dos valores dos eixos x e y, e outros serviços que permitem ao usuário da classe manipular os pontos do plano cartesiano.

5. Desenvolva um aplicativo para testar a classe PlanoCartesiano criada no Exercício 4.

6. Crie uma classe de nome Discente com dados privados para armazenar a matrícula, o nome e o curso do discente. Inclua na classe:

a) um membro estático privado para contar o número de objetos criados;
b) um construtor *default* que contabilize o número de discentes criados;
c) um construtor sobrecarregado que contabilize o número de discentes criados e inicialize os atributos da classe;
d) um método estático que retorna o número de objetos criados;
e) métodos de acesso aos dados encapsulados (*get* e *set*);
f) um método para retornar uma representação *string* do objeto.

7. Acrescente na classe Discente, criada no Exercício 6, um vetor de objetos capaz de armazenar um número de disciplinas definido pelo usuário. Os atributos das disciplinas (nome e nota) devem ser organizados em uma classe Disciplina. A classe Discente deve ter um relacionamento de composição com a classe Disciplina.

8. Use a classe Discente, ampliada no Exercício 7, para criar um aplicativo Java GUI chamado ControleNota com os seguintes recursos.

a. Cadastrar

Permite ao usuário fornecer os dados do discente (nome, matrícula, curso) e de suas disciplinas (nome, nota) e armazena-os como um objeto na lista (java.util.List) de objetos da classe Discente. Use a declaração abaixo:

```
public void cadastrarDiscente(List<Discente> discenteList);
```

O cadastro deve ser cancelado quando o usuário clicar no botão Cancelar da caixa de diálogo, no entanto, se os dados do discente (nome, matrícula, curso) já tiverem sido lidos pelo programa eles devem ser gravados na lista. Quando a opção cadastrar for ativada novamente para um discente já cadastrado, o programa deve ler os demais dados dele, ou seja, os nomes e as notas de suas disciplinas. Isso significa que sempre que a matrícula for fornecida o programa deve verificar se esse discente já está cadastrado, se tiver, somente os dados de suas disciplinas serão lidos, caso todas elas não tenham sido cadastradas ainda. Logo, não é permitido cadastrar os dados de um discente mais de uma vez e nem de suas disciplinas, porque sua matrícula deve ser usada como código de identificação único e os dados das disciplinas só podem ser cadastrados se o nome dessa não estiver vazio.

b. Pesquisar Discente

Solicita ao usuário o nome do discente, pesquisa na lista de objetos e exibe todos os dados do discente, incluindo de suas disciplinas (nome e nota). Se não for encontrado exibe a mensagem Discente não cadastrado. Use a declaração abaixo:

```
public void pesquisarDiscente(List<Discente> discenteList);
```

O método pesquisarDiscente deve usar sua versão sobrecarregada para pesquisar o discente e, em caso de sucesso, retornar o objeto Discente, se não retornar null. Use a declaração abaixo:

```
private Discente pesquisarDiscente(List<Discente> discenteList, String nome);
```

c. Pesquisar Disciplina

Solicita ao usuário a matrícula ou o nome do discente e o nome de uma disciplina que o discente está matriculado. Observe que o usuário poderá digitar a matrícula ou o nome, ambos do mesmo tipo de dado.

Pesquisa pelo discente na lista de objetos e, em caso de sucesso, exibe os dados da disciplina (nome e nota) e o nome do discente. Se a matrícula ou o nome não estiverem cadastrados exibe a mensagem Discente não cadastrado. Se a disciplina não estiver cadastrada exibe a mensagem O discente XXX não está matriculado nesta disciplina. Onde XXX deve ser substituído pelo nome ou número da matrícula do discente. Use a declaração abaixo:

```
public void pesquisarDisciplina(List<Discente> discenteList);
```

O método pesquisarDisciplina deve usar os métodos pesquisarMatricula e pesquisarDisciplina para, respectivamente, pesquisar o discente usando a sua matrícula e pesquisar o nome da disciplina. Eles devem retornar, respectivamente, o objeto Discente e um vetor de Object em caso de sucesso, caso contrário retornam null. O vetor de Object deve armazenar objetos com o nome e a nota da disciplina. Use as declarações abaixo:

```
private Discente pesquisarMatricula(List<Discente> discenteList, String matricula);
private Object[] pesquisarDisciplina(Discente discente, String nome);
```

d. Alterar

Solicita ao usuário o nome do discente, pesquisa na lista de objetos e permite ao usuário fornecer novos dados do discente, incluindo os nomes e as notas das disciplinas. O único dado que não pode ser alterado é a matrícula do discente. Se não for encontrado exibe a mensagem Discente não cadastrado. Use a declaração abaixo:

```
public void alterarDiscente(List<Discente> discenteList);
```

O método alterarDiscente deve usar o método pesquisarDiscente.

e. Excluir

Solicita ao usuário o nome do discente, pesquisa na lista de objetos e exibe todos os dados do discente, incluindo suas disciplinas (nome e nota). Se não for encontrado exibe a mensagem Discente não cadastrado. Uma mensagem de confirmação de exclusão deve ser exibida ao usuário. Use a declaração abaixo:

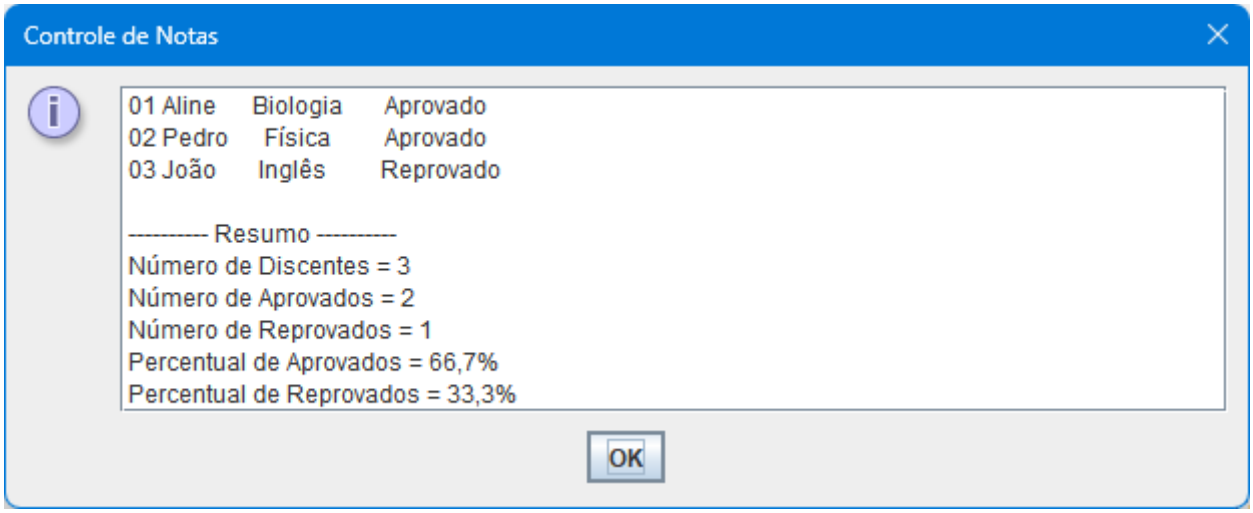
```
public void excluirDiscente(List<Discente> discenteList);
```

O método excluirDiscente deve usar o método pesquisarDiscente.

f. Relatório

Exibe um relatório usando o leiaute da figura abaixo com as seguintes informações:

1. matrícula;
2. nome do discente;
3. nome do curso;
4. situação final (aprovado ou reprovado);
5. número de discentes;
6. número de aprovados;
7. número de reprovados;
8. percentual de aprovados;
9. percentual de reprovados.



Para exibir o relatório use os componentes JScrollPane e JTextArea do pacote javax.swing. Use a declaração abaixo:

```
public void relatorio(List<Discente> discenteList);
```

☞ **Nota:** 1. O programa deve criar uma única instância da lista de objetos da classe Discente e essa deve ser uma variável local criada por um método não estático da classe ControleNota chamado criarTurma. Use a declaração abaixo.

```
private List<Discente> criarTurma();
```

2. Considere como nota válida os valores de 0 a 10, inclusive.

3. O discente será considerado aprovado se as suas notas em todas as disciplinas forem igual ou superior a 6.

4. Validar os dados de entrada fornecidos pelo usuário. Valor do tipo *string* não pode ser vazio e a matrícula deve ser um valor alfanumérico no formato SSSAAAA-DDD, onde:

   SSS = representa a sigla do curso do discente (máximo de 3 caracteres);
   AAAA = o ano de matrícula do discente (usar 4 dígitos);
   DDD = valor numérico inteiro, positivo e sequencial (máximo de 3 dígitos).

   Exemplos de matrícula: TSI2023-015, TI2023-025.

9. Por que o método pesquisarDisciplina no Exercício 8 retorna um vetor de objetos do tipo Object em vez de objetos da classe Disciplina?

10. Modifique o programa do Exercício 8 criando a classe Turma e coloque nessa classe os métodos cadastrarDiscente, pesquisarDiscente, pesquisarDisciplina, alterarDiscente e excluirDiscente. Altere a assinatura desses métodos de modo que não recebam nenhum parâmetro. Modifique também o tipo de retorno do método criarTurma para retornar a referência de um objeto Turma.

11. Acrescente na classe Turma, criada no Exercício 10, os seguintes métodos:

```
/* Obtém uma cópia superficial da relação de disciplinas do discente.

   O tipo de cópia usado neste método é chamado de cópia superficial, porque os valores das
   variáveis de instância do objeto são copiados em outro objeto do mesmo tipo. Os valores dos
   atributos dos objetos de tipo primitivo ou tipo por referência são sempre copiados. A lista
   retornada possui as mesmas referências de objetos da lista interna da classe Turma.
*/
public List<Discente> clonarTurma();

/* Obtém uma cópia em profundidade da relação de disciplinas do discente.

   O tipo de cópia usado neste método é chamado de cópia em profundidade, porque os valores das
   variáveis de instância do objeto são copiados em um novo objeto para cada variável de instância
   de tipo primitivo ou tipo por referência. A lista retornada possui as referências dos novos objetos
   criados para receber os valores dos objetos da lista interna da classe Turma.
*/
public List<Discente> copiarTurma();
```

12. É possível modificar os atributos dos objetos da classe Discente armazenados na lista interna da classe Turma após usar os métodos clonarTurma e copiarTurma? Se sim, isso representa uma quebra de encapsulamento da classe Turma em relação a sua lista interna de objetos?

13. A saída abaixo mostra que um programa pode produzir resultados imprecisos usando os tipos primitivos float ou double. A classe BigDecimal fornece métodos para realizar cálculos exatos e sem nenhum arredondamento. No entanto, se a precisão dos cálculos matemáticos não é uma exigência de uma aplicação, mas o desempenho sim, é melhor usar os tipos primitivos float ou double por apresentarem um desempenho superior ao tipo por referência BigDecimal, conforme se pode observar na imagem abaixo. Escreva um programa para verificar essas questões na prática (precisão de cálculo x desempenho de execução) e gerar uma saída idêntica ao exemplo abaixo.

```
Soma 1 bilhão de R$ 0,01 com double = R$ 9.999.999,83
Tempo inicial: 22:59:52.3762239
Tempo final..: 22:59:53.3710854
     Tempo de execução = 0,99 segundos

Soma 1 bilhão de R$ 0,01 com BigDecimal = R$ 10.000.000,00
Tempo inicial: 22:59:53.3767283
Tempo final..: 22:59:57.9392221
     Tempo de execução = 4,56 segundos

O tempo de execução com double foi 4,6 vezes mais veloz do que com BigDecimal.
```

14. Um profissional de educação física que trabalha como *personal trainer* necessita de um aplicativo que o permita acompanhar a evolução das atividades físicas de seus clientes. Baseado nesse acompanhamento ele faz sua orientação personalizada para cada pessoa. O programa que o *personal trainer* necessita deve importar os dados de um *app* instalado no *smartphone* de seus clientes. Esse *app* permite exportar um arquivo texto com os dados do Arquivo 1 (ver figura abaixo) quando a atividade física realizada pelo indivíduo for corrida ou caminhada.

Quando atividade física for diferente de corrida ou caminhada, por exemplo, se o indivíduo jogar basquete e usar o *app* para monitorar o seu desempenho, ao exportar o arquivo os dados serão iguais do Arquivo 2 (ver figura abaixo).

Pra cada tipo de atividade física que o usuário do *app* deseja compartilhar com o *personal trainer* ele deve gerar um arquivo texto. É um arquivo por atividade física realizada. Arquivos gerados para uma determinada atividade física que possuem mais dados de avaliação do exercício, por exemplo corrida, possuem a estrutura idêntica do Arquivo 1. Já as atividades físicas que possuem menos dados de avaliação do exercício, por exemplo dança, possuem a estrutura idêntica do Arquivo 2.

Enfim, independente do tipo de atividade física praticada, o arquivo texto gerado pelo *app* será sempre igual ao formato do Arquivo 1 ou Arquivo 2.

Desenvolva um programa Java GUI chamado PersonalTraining com os seguintes recursos:

1. Importação dos arquivos de texto com estrutura igual ao dos arquivos 1 e 2 para armazenamento desses dados em uma lista de objetos mantendo os campos nome e data do exercício como campos de identificação única. O usuário deve digitar em uma caixa de diálogo o nome do arquivo que ele deseja importar para o programa. Considere esses

arquivos armazenados no diretório corrente do projeto Java. Use a classe TextFile para ler os arquivos a serem importados. A classe TextFile e sua documentação estão disponíveis no arquivo anexo TextFile.jar. Use os arquivos caminhada1.txt, caminhada2.txt e basquete.txt fornecidos como anexo para testar a importação do programa. Esses arquivos estão codificados com UTF-8.

2. Lê e pesquisa o nome do cliente do *personal trainer* para exibir os seus dados e os detalhes de seus exercícios, incluindo o ritmo em cada quilômetro se a atividade física for corrida ou caminhada. Se o cliente não estiver cadastrado exibe a mensagem Cliente não cadastrado.

3. Exibe um relatório usando os componentes JTextArea e JScrollPane para mostrar para cada cliente do *personal trainer* os valores abaixo e a data em que eles ocorreram.

    a) A maior duração de um exercício.
    b) A maior distância percorrida.
    c) O maior número de calorias perdidas.
    d) O maior número de passos dados.
    e) A velocidade máxima mais rápida.

Arquivo 1

Exercício: Caminhada

------ Usuário ------
Nome: Bruno Silva
Sexo: masculino
Altura: 1,70 m
Peso: 76,0 Kg
Data de nascimento: 22/04/1980

------ Detalhes do exercício ------
Data: 20/05/2022
Tempo: 10:30 - 11:42
Duração: 60m12s
Distância: 6,75 Km
Calorias perdidas: 375 Kcal
Passos: 8.298
Velocidade média: 6,5 Km/h
Velocidade máxima: 9,0 Km/h
Ritmo médio: 09’08” /Km
Ritmo máximo: 06’40” /Km
Menor elevação: 1.044 m
Maior elevação: 1.154 m

------ Ritmo ------
1 Km: 09’15”
2 Km: 09’00”
3 Km: 08’54”
4 Km: 08’56”
5 Km: 09’06”
6 Km: 09’09”
6,75 Km: 09’45”

Arquivo 2

Exercício: Basquete

------ Usuário ------
Nome: Bruno Silva
Sexo: masculino
Altura: 1,70 m
Peso: 76,0 Kg
Data de nascimento: 22/04/1980

------ Detalhes do exercício ------
Data: 20/05/2022
Tempo: 15:00 - 15:30
Duração: 30m0s
Distância: 4,54 Km
Calorias perdidas: 580 Kcal
Passos: 4.152

Para desenvolver esse programa deve-se criar no mínimo as classes abaixo.

a) Pessoa: representa as propriedades (atributos e métodos) dos clientes do *personal trainer*.

b) Exercicio: representa as propriedades dos exercícios praticados pelos clientes do *personal trainer*.

c) ExercicioList: representa uma lista de objetos da classe Exercicio com métodos para gerenciar os dados dos exercícios importados de cada cliente do *personal trainer*.

d) PersonalTraining: inicia o programa e fornece os serviços de importação, pesquisa e relatório do aplicativo.

15. Desenvolva um programa Java GUI chamado CalculoInvestimentos com as seguintes funcionalidades abaixo.

1. Importar investimentos

Lê o arquivo texto fornecido pelo usuário com o nome Investimentos.txt e que possui os seguintes dados: tipo, estratégia e nome do investimento, *rating*, proteção do FGC, valor investido, taxa do investimento ao ano (a.a.), data em que o dinheiro foi investido e a data em que ele pode ser resgatado do investimento. O arquivo possui um formato em que cada linha, a partir da segunda, possui os dados de um investimento separados por ponto-e-vírgula (;). A primeira linha do arquivo sempre possui uma descrição dos dados, como se pode ver no exemplo abaixo.

```
Tipo;Estratégia;Nome;Rating;FGC;Valor Investido;Taxa;Data do Investimento;Data de Resgate
RF;Pós-fixado;CDB Original Jan/2026;AAA;sim;2000;10%;14/01/2021;14/01/2026
RV; Multimercado;Western Asset FIM;A;não;4000;58%;15/01/2021;15/12/2021
```

Os dados de cada investimento devem ser extraídos do arquivo e armazenados em um objeto da classe Investimento. O tipo do objeto a ser criado deve ser definido de acordo com o tipo de investimento obtido em cada linha do texto (RF = Renda Fixa ou RV = Renda Variável).

Esses objetos devem ser armazenados em uma lista de objetos da classe Investimento. Considere o arquivo Investimentos.txt armazenado no diretório corrente do projeto Java. Use a classe TextFile para ler o conteúdo desse arquivo. A classe TextFile e sua documentação estão disponíveis no arquivo anexo TextFile.jar. Use o arquivo Investimentos.txt fornecido como anexo para testar a importação do programa. Esse arquivo está codificado com UTF-8.

2. Relatório de Investimentos

Gera um relatório para cada tipo de investimento obtido da lista de objetos. Os dados a serem exibidos devem ser agrupados pelo tipo de investimento, exibindo primeiro os investimentos de renda fixa e depois os de renda variável. Os valores a serem exibidos no relatório para cada investimento são:

1. nome;
2. *rating*;
3. estratégia de investimento;
4. proteção FGC;
5. prazo de acordo com o tipo de investimento;
6. valor investido;
7. taxa do investimento ao ano (a.a.);
8. taxa do investimento ao mês (a.m.);
9. data do investimento;
10. data de resgate;
11. valor bruto acumulado do investimento (sem desconto de IR);
12. valor líquido acumulado do investimento (com desconto de IR);
13. alíquota de IR (Imposto de Renda);
14. valor do IR em reais a ser pago sobre o rendimento;
15. rendimento bruto (sem desconto de IR);
16. rendimento líquido (com desconto de IR).
17. rentabilidade bruta percentual do investimento.

A rentabilidade expressa a valorização (ou desvalorização) de um determinado investimento em termos percentuais. Desta forma, um investimento de R$ 10 que, após um mês vale R$ 11, registrou uma rentabilidade de 10%. A fórmula de cálculo da rentabilidade é a seguinte:

Rentabilidade = ((Valor Bruto Acumulado / Valor Investido) - 1) * 100

O rendimento em reais do investimento corresponde aos juros recebidos de acordo com a taxa de remuneração do investimento. Para calcular o valor bruto acumulado de um investimento use a fórmula abaixo.

$$\mathbf{VF = VP \times (1 + t)^{p}}$$

Onde:

VF = valor futuro (valor bruto acumulado)
VP = valor presente (valor investido)
t = é a taxa de remuneração do investimento
p = é o prazo (período) do investimento

☞ **Atenção:** 1. A taxa e o prazo devem usar a mesma unidade, ou seja, ambos em dias, meses ou anos.

2. O cálculo do valor bruto acumulado considera apenas os dias úteis, portanto assuma que um ano possui 252 dias úteis e um mês 21 dias úteis.

3. Para se calcular o número de dias úteis entre duas datas desconsidere qualquer feriado nos dias úteis da semana (segunda a sexta).

4. 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. 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.

5. Investimentos de renda fixa possuem quatro alíquotas de IR segundo o prazo do investimento, conforme a Tabela 1 abaixo. Investimentos de renda variável são tributados com uma única alíquota de IR de 15%.

Alíquotas de Imposto de Renda	
Taxa de IR	**Tempo em dias corridos**
22,5%	Até 180 dias
20,0%	De 181 a 360 dias
17,5%	De 361 a 720 dias
15,0%	Acima de 720 dias

Tabela 1 - Alíquota de Imposto de Renda

Para obter uma taxa equivalente, por exemplo, 12% ao ano correspondem a que taxa mensal? Use a fórmula abaixo para obter a taxa mensal**.**

$$taxaequivalente = (1 + taxaconhecida)^{\frac{prazoquequero}{prazoquetenho}} - 1$$

Usando a fórmula acima, ((1 + 12%)$^{(1 / 12)}$ - 1), temos a taxa equivalente de 0,94% ao mês, que corresponde aos 12% ao ano. Veja que o prazoquequero corresponde a um mês (1) e o prazoquetenho a um ano (12 meses).

Use o leiaute abaixo para exibir o relatório usando os componentes JTextArea e JScrollPane.

Relatório de Investimentos

1. Renda Fixa

- CDB PAN Jan/2026 | AAA | Pós-fixado | FGC: sim | Prazo: 3 anos

Valor investido:	R$ 4 .500,00
Taxa ao ano:	11,00% a.a.
Taxa ao mês:	0,87% a.m.
Data do investimento:	14/04/2018
Data de resgate:	14/04/2021
Valor bruto:	R$ 6.154,34
Valor líquido:	R$ 5.906,19
Alíquota de IR:	15,00%
Valor do IR:	R$ 248,15
Rendimento bruto:	R$ 1.654,34
Rendimento líquido:	R$ 1.406,19
Rentabilidade:	36,763%

2. Renda Variável

- Western Asset FIM | A | Multimercado | FGC: não | Prazo: 231 dias

Valor investido:	R$ 4.000,00
Taxa ao ano:	58,00% a.a.
Taxa ao mês:	3,89% a.m.
Data do investimento:	15/01/2017
Data de resgate:	15/12/2017
Valor bruto:	6.083,62
Valor líquido:	5.771,08
Alíquota de IR:	15,00%
Valor do IR:	R$ 312,54
Rendimento bruto:	R$ 2.083,62
Rendimento líquido:	R$ 1.771,08
Rentabilidade:	52,09%

Para desenvolver esse programa deve-se criar no mínimo os tipos de dados (classe ou enumeração) abaixo.

a) Imposto: enumeração para definir as alíquotas de Imposto de Renda (IR) (ver a Tabela 1) a serem aplicadas sobre o rendimento bruto do investimento.

b) EstrategiaInvestimento: enumeração para definir as estratégias dos diferentes tipos de investimento. Essas estratégias são: Inflação, Prefixado, Pós-fixado, Renda Variável, Internacional, Multimercado, Alternativo.

c) Investimento: essa classe define as propriedades (atributos e métodos) dos investimentos. Ela deve ter, dentre outras, duas variáveis de instância para representar: 1. o tipo de estratégia usada no investimento; 2. a taxa de imposto de renda a ser calculado sobre o rendimento bruto de um investimento de acordo com o seu prazo em dias.

d) CalculoInvestimentos: inicia o programa e fornece os serviços de importação e relatório. Essa classe deve ter uma lista de objetos da classe Investimento.

Prof**. M**árlon **O**liveira **d**a **S**ilva
marlon.silva@ifsudestemg.edu.br