Universidade Federal do Rural do Semi-Árido
Centro Multidisciplinar de Pau dos Ferros
Departamento de Engenharias e Tecnologia
PET1707 / PEX0130 - Paradigmas de Programação / Programação Orientada a Objetos
Professor: Ítalo Assis

Lista de Exercícios

Sumário

1 Introdução ao Java **2**

2 Criação de classes - parte 1 **5**

3 Criação de classes - parte 2 **6**

4 Arrays e ArrayLists **8**

5 Reúso de Classes **10**

6 Classes abstratas, polimorfismo e interfaces **12**

7 Coleções genéricas **14**

8 Exceções **15**

9 Referências **16**

1 Introdução ao Java

1. Escreva um programa que calcule a área de um quadrado. O valor do lado de um quadrado deverá ser informado pelo usuário.

2. Escreva um programa que calcule a soma e a média aritmética de 3 números. Os 3 valores reais serão informados pelo usuário.

3. Escreva um programa que receba um ângulo em graus e o converta para sua representação em radianos. Divulgue também seno, cosseno, tangente, cossecante, secante e cotangente do ângulo.

4. Escreva um programa que receba uma temperatura em graus centígrados e a apresente em graus Fahrenheit.

5. Escreva um programa que receba um número de 3 dígitos no formato CDU (Centena, Dezena e Unidade) e, utilizando operadores aritméticos, armazene cada dígito em uma variável de tipo inteiro. Por fim, reescreva o número no formato UCD. Exemplo: 123 deve ser reescrito como 312.

6. Escreva um programa que recebe um valor inteiro representando um intervalo em minutos e imprime o equivalente a esse período expresso em valores inteiros para dias, horas e minutos. Ex.: 9257 minutos $= 6$ dias, 10 horas e 17 minutos.

7. Escreva um programa que lê três números e determina qual número é o menor.

8. Para alavancar as vendas, uma loja resolveu fazer a seguinte promoção relâmpago em dois setores:

   - No setor de Eletros, que possui código 222, todas as peças que custam mais de R$ 500 vão ter 10% de desconto.
   - No setor de Cama, mesa e banho, que possui código 111, peças com valor acima de R$ 100 vão ter 40% de desconto, peças que custam entre R$ 50 e R$ 100 vão ter 20% de desconto e peças abaixo de R$ 50 vão ter 10% de desconto.

   Escreva um programa que recebe do usuário o código do setor da loja e o valor original de um produto e imprime na tela o nome do setor e o valor do produto com desconto. Caso o usuário forneça um código diferente de 111 ou 222, exiba a mensagem *Setor Invalido*.

9. Escreva um programa que lê um número que representa o valor da carta, de um (ás) a treze (rei), e outro número correspondente ao naipe (1 = ouros, 2 = paus, 3 = copas e 4 = espadas). O programa deve imprimir o nome da carta por extenso.

10. Determine se cada uma das seguintes alternativas é verdadeira ou falsa. Se falsa, explique por quê.

    - O caso *default* é requerido na instrução de seleção *switch*.
    - A instrução *break* é requerida no último caso de uma instrução de seleção *switch*.
    - A expressão $((x > y)\&\&(a < b))$ é verdadeira se $(x > y)$ for verdadeiro ou $(a < b)$ for verdadeira.

- Uma expressão contendo o operador || é verdadeira se um ou ambos de seus operandos forem verdadeiros.
- Para testar para uma série de valores em uma instrução *switch*, pode-se utilizar um hífen (-) entre os valores inicial e final da série em um rótulo case.
- Listar casos consecutivamente sem instruções entre eles permite aos casos executar o mesmo conjunto de instruções.

11. Escreva um programa que lê dois pontos no plano cartesiano com coordenadas x e y e informa se o segundo ponto está acima, abaixo, à esquerda e/ou à direita do primeiro.

12. O mês de fevereiro de 2015 começou em um domingo. Escreva um programa que, dado o número de um dia do mês de fevereiro de 2015 (um valor entre 1 e 28), imprime o dia da semana correspondente.

    - Exemplo de entrada: 10
    - Exemplo de saída: O dia 10 será uma terça-feira.

13. Escreva um programa para imprimir uma versão aproximada de um cartão da Mega-Sena (somente com os números, respeitando o número de linhas e a distribuição dos números nas linhas).

14. Escreva um programa que apresente a série de Fibonacci até o n-ézimo termo. Assuma que $n > 0$.

15. Escreva um programa que escreve os 4 primeiros números perfeitos. Um número perfeito é aquele que é igual à soma dos seus divisores (exceto o próprio número). Exemplos: $6 = 1 + 2 + 3$, $28 = 1 + 2 + 4 + 7 + 14$.

16. Escreva um programa que determine quais são todos os números de 3 algarismos cuja soma dos cubos de seus algarismos sejam iguais ao próprio número.

    - Exemplo: $153 = 1^3 + 5^3 + 3^3 = 1 + 125 + 27$.

17. Duas cidades, A e B, têm populações de 7000 e 20000 habitantes, respectivamente. A cidade A tem um crescimento populacional de $3,5\%$ ao ano, enquanto a população da cidade B cresce 1% ao ano. Faça um programa que utilize laço(s) para calcular a quantidade de anos necessária para que a população da cidade A seja maior ou igual a população da cidade B.

18. Crie uma função recursiva *potencia(base,expoente)* que, quando chamada, retorna $base^{expoente}$.

    Por exemplo, *potencia(3,4)* deve ser implementado como $3 * 3 * 3 * 3$. (A saída é apenas 81 nesse exemplo)

    Assuma que *expoente* é um inteiro maior ou igual a 1.

19. Escreva uma função que calcule o fatorial de forma recursiva. Para cada chamada recursiva, exiba as saídas em uma linha separada e adicione um nível de recuo (tabulação). Faça o melhor que você puder para tornar a saída limpa, interessante e significativa. Seu objetivo aqui é projetar e implementar um formato de saída que facilite o entendimento da recursão.

Por exemplo, para uma entrada $n = 5$, o programa deve exibir:

$0! = 1$
$1! = 1$
$2! = 2$
$3! = 6$
$4! = 24$
$5! = 120$

20. Crie uma função recursiva mdc que retorna o máximo divisor comum de m e n. O mdc de m e n é definido recursivamente como segue:

    - se $n > m$, retorne $mdc(n, m)$;
    - se $n = 0$, retorne m;
    - senão, retorne $mdc(n, m\%n)$, onde $\%$ é o operador de resto da divisão.

21. Crie uma função recursiva que verifique se um dado número é primo.

22. Dado um polinômio

$$P_n(x) = a_0x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n \tag{1}$$

    este pode ser definido recursivamente como

$$P_n(x) = xP_{n-1}(x) + a_n \tag{2}$$

    Escreva uma função recursiva para calcular $P_n(x)$. Os parâmetros n, $a_0, \cdots, a_n$ e o valor x devem ser solicitados ao usuário na função $main$.

2 Criação de classes - parte 1

23. Escreva uma classe para representar um time de um esporte qualquer em um campeonato desse esporte. Que atributos devem ser representados nessa classe? Quais métodos ela deve conter? Escreva um aplicativo de teste que demonstre as capacidades da classe criada.

24. Crie uma classe chamada $Fatura$ para que uma loja de suprimentos de informática a utilize para representar uma fatura de um item vendido. Uma Fatura deve incluir as seguintes informações sobre o item vendido: o número de identificação, a descrição, a quantidade comprada e o preço unitário. Se a quantidade não for positiva, ela deve ser configurada como 0. Se o preço por item não for positivo, ele deve ser configurado como 0.0. Forneça também um método chamado $calculaTotal$ que calcula e retorna o valor de fatura (isto é, multiplica a quantidade pelo preço por item). Escreva um aplicativo de teste que demonstre as capacidades da classe criada.

25. Escreva uma classe cujos objetos representam alunos matriculados em uma disciplina da Ufersa. Cada objeto dessa classe deve guardar os seguintes dados do aluno: matrícula, nome, 2 notas de prova ($P1$ e $P2$) e 1 nota de trabalho (T). Escreva os seguintes métodos para esta classe:

    - $media$: calcula a média parcial do aluno (MP)
        - cada prova tem peso $2,5$ e o trabalho tem peso 2 ($MP = \frac{2{,}5 \times P1 + 2{,}5 \times P2 + 2 \times T}{7}$)
    - $provaFinal$: calcula quanto o aluno precisa para o exame final (EF)
        - retorna zero se o aluno não necessita realizar o exame final ($MP < 3$ ou $MP >= 7$)
        - média final MF = $(MP \times 6 + EF \times 4)/10$
        - é necessário que MF seja maior ou igual a 5 para que o aluno seja aprovado após realizar o exame final

    Escreva um aplicativo de teste que demonstre as capacidades da classe criada.

3 Criação de classes - parte 2

26. Crie uma classe denominada *Elevador* para armazenar as informações de um elevador dentro de um prédio. A classe deve armazenar o número do andar atual, o total de andares no prédio, a capacidade do elevador e quantas pessoas estão presentes nele. Outras classes não devem ter acesso direto aos atributos de *Elevador*. A classe deve também disponibilizar os seguintes métodos:

    - construtor : que deve receber como parâmetros a capacidade do elevador e o total de andares no prédio (um elevador sempre começa no térreo e vazio);
    - *entra* : para acrescentar uma pessoa no elevador (só deve acrescentar se ainda houver espaço);
    - *sai* : para remover uma pessoa do elevador (só deve remover se houver alguém dentro dele);
    - *sobe* : para subir um andar (não deve subir se já estiver no último andar);
    - *desce* : para descer um andar (não deve descer se já estiver no térreo).

    Escreva um aplicativo de teste que demonstre as capacidades da classe criada, ou seja, execute todos os métodos da classe.

27. Escreva a classe *Contador* que encapsule um valor usado para contagem de eventos. Esta classe deve esconder o valor encapsulado de programadores-usuários, fazendo com que o acesso ao valor seja feito através de métodos que devem zerar, incrementar e imprimir o valor do contador. Escreva um aplicativo de teste que demonstre as capacidades da classe criada.

28. Escreva uma classe *Lampada* que possui o atributo *estadoDaLampada* (ligado ou desligado) e os métodos *acende*, *apaga* e *mostraEstado* (ex.: *A lampada esta acesa*). Inclua um método *estaLigada* que retorne verdadeiro se a lâmpada estiver ligada e falso caso contrário. Adicione um campo que indique quantas vezes a lâmpada foi acesa. Para isso, utilize uma instância da classe *Contador* criada anteriormente e implemente a lógica necessária para atualizar seu valor. Escreva um aplicativo de teste que demonstre as capacidades da classe criada.

29. Escreva uma classe para representar um número complexo. Essa classe deve conter três construtores. Um construtor deverá receber os dois valores (parte real e parte imaginária) como argumentos, o outro somente o valor real, considerando o imaginário como sendo zero, e o terceiro construtor não recebe argumentos, considerando as partes real e imaginária do número complexo como sendo iguais a zero. Escreva um método *toString* que apresente o número complexo em notação apropriada (ex.: $2 + 4i$) e o use em um aplicativo de teste que demonstre as capacidades da classe criada.

30. Escreva uma classe que contenha métodos estáticos para retornar o maior de dois, três, quatro e cinco valores, considerando que os argumentos e o retorno dos métodos podem ser dos tipos int e double. Em outra classe, escreva um aplicativo de teste que, sem criar objetos, demonstre as capacidades da classe criada.

31. Escreva uma versão da classe *RegistroAcademico* (vista em aula) que tenha o campo *numeroDeMatriculas* declarado como *static*, e que incremente o valor deste campo cada vez que uma instância da classe for criada. Use o atributo *numeroDeMatriculas* para definir o atributo matricula automaticamente com um valor diferente para cada instância. Escreva também uma aplicação que crie algumas instâncias da classe para demonstrar seu funcionamento.

32. Escreva uma classe genérica com 3 atributos de um mesmo tipo genérico. Além do construtor, essa classe deve ter:

    - Um método que diz quantos dos 3 atributos são iguais;
    - Um método que imprime os 3 atributos.

    Escreva também uma classe executável para demonstrar o uso da classe criada com diferentes tipos de dados.

4 Arrays e ArrayLists

33. Crie uma classe *EntradaEmAgenda* que contenha:

    - os dados necessários para armazenar uma entrada de agenda (hora, dia, mês, ano e assunto);
    - um construtor;
    - um método *toString*;
    - um método *ehNoDia* que recebe valores de dia, mês e ano e retorna *true* se o dia, mês e ano daquela instância da classe forem iguais aos argumentos passados.

    Crie também uma classe *Agenda* que:

    - encapsule uma agenda de compromissos representada por um *ArrayList* de instâncias da classe *EntradaEmAgenda*;
      - este item é obrigatório. Caso não seja contemplado, toda a questão será desconsiderada
    - implemente um método construtor;
    - possua um método para adicionar um novo compromisso à lista de compromissos;
    - tenha um método *listaDia* que recebe valores de dia, mês e ano e lista todas as instâncias de *EntradaEmAgenda* que caem naquele dia, mês e ano.

    Por fim, escreva uma classe executável que crie uma *Agenda*, adicione a ela 5 compromissos e, usando o método *listaDia*, liste as entradas da agenda que tem a mesma da data do seu aniversário.

34. Escreva uma classe que encapsule uma matriz de tamanho 2×2 de valores do tipo *float* usando um *array* de duas dimensões. Nesta classe, além do construtor, escreva um método que calcule o determinante da matriz encapsulada e um método que permita a impressão em formato matricial dos seus valores. Escreva um aplicativo de teste que demonstre as capacidades da classe criada.

35. Crie um objeto de uma classe chamada *Cliente* com os atributos *id*, *nome*, *idade*, *telefone*. Faça um programa para solicitar os dados de vários clientes e armazenar em um *ArrayList* até que se digite um número de *id* negativo. Em seguida, exiba os dados de todos os clientes.

36. Escreva um aplicativo que calcula o produto de uma série de inteiros que são passados para um método *produto* utilizando uma lista de argumentos de comprimento variável. Escreva também uma classe executável que teste seu método com várias chamadas, cada uma com um número diferente de argumentos.

37. Escreva um aplicativo para simular o lançamento de dois dados. O aplicativo deve utilizar um objeto de classe *Random*, uma vez para lançar o primeiro dado e novamente para lançar o segundo dado. A soma dos dois valores deve então ser calculada. Cada dado pode mostrar um valor inteiro de 1 a 6, portanto a soma dos valores irá variar de 2 a

12, com 7 sendo a soma mais frequente e 2 e 12, as somas menos frequentes. Seu aplicativo deve lançar o dado 36.000.000 vezes. Utilize um *array* unidimensional para contar o número de vezes que cada possível soma aparece. Exiba os resultados.

5 Reúso de Classes

38. Crie as classes *Equipamento* e *Computador*, cada uma com dois atributos privados à sua escolha. Além disso, a classe *Computador* deverá herdar os métodos e atributos da classe *Equipamento*. Escreva métodos de acesso, *get*'s e *set*'s, para os atributos definidos em ambas as classes. Cada classe também deve ter um método *toString*. Lembre-se que o método *toString* de *Computador* também deve representar os atributos herdados. Por fim, crie uma classe executável, *TestaEquipamento*, para instanciar um objeto de cada classe, inicializar seus atributos e imprimí-los.

39. Crie uma classe para representar uma data e um horário (*DataHora*).

    - Escreva uma classe *EventoDelegacao* que seja baseada na classe *DataHora* e que contenha um campo para indicar qual o evento que ela representa (use uma *String* para isto). Use o mecanismo de delegação para criar a classe *EventoDelegacao*;
    - Escreva uma classe *EventoHeranca* que seja baseada na classe *DataHora* e que contenha um campo para indicar qual o evento que ela representa (use uma *String* para isto). Use o mecanismo de herança para criar a classe *EventoHeranca*;
    - Escreva um aplicativo de teste que demonstre o uso das classes criadas.

40. Escreva as classes *LivroLivraria* e *LivroBiblioteca* que herdam da classe *Livro*. Quais as diferenças entre as duas classes e que campos elas têm em comum? Defina os atributos de cada classe e escreva um aplicativo de teste que demonstre o uso das classes criadas.

41. Crie uma classe *Pessoa* com ao menos 2 atributos a sua escolha. Escreva a classe *Politico* que herda da classe *Pessoa* e tem um campo adicional para representar o partido do político. Escreva também as classes *Prefeito* e *Governador* que herdam da classe *Politico* e que contém campos para representar a cidade ou estado governado. Todos atributos devem ser privados. Cada classe deve ter um construtor e um método *toString*. Lembre-se que cada método *toString* deve representar todos os atributos, inclusive os herdados. Escreva também uma aplicação que demonstre o uso de instâncias destas classes.

42. Implemente a classe *Funcionario* com nome, salário e os métodos:

    - *aumentarSalario* : recebe o valor do aumento e o adiciona ao salário;
    - *ganhoAnual* : computa o valor recebido em 12 meses e o 13^o;
    - e *toString* : retorna uma representação textual de um objeto de *Funcionario*.

    (i) Crie também a classe *Assistente*, que também é um funcionário e que possui um número de matrícula (e seus métodos de acesso), além de um método *toString*.

    (ii) Escreva as classes *Tecnico* e *Administrativo*

    - Ambas as classes são filhas da classe *Assistente*
    - Ambas as classes devem ter um método *ganhoAnual*
    - Assistentes Técnicos possuem um bônus salarial

- Assistentes Administrativos possuem um turno (dia ou noite) e um adicional noturno

(iii) Lembre-se que o 13^o não possui adicional noturno, mas pode possuir bônus salarial (se aplicável).

6 Classes abstratas, polimorfismo e interfaces

43. Explique por que não podemos ter construtores declarados com a palavra-chave *abstract*.

44. Defina uma classe para conter informações sobre um funcionário de uma empresa (classe *Funcionario*). Quais são os atributos dessa classe? Inclua entre eles o salário que o funcionário deve receber por hora trabalhada. Implemente, para essa classe, pelo menos dois métodos construtores: um que receba apenas o nome do funcionário e assuma valores padrão para os demais atributos (assuma que o funcionário deve receber dois reais por hora trabalhada); o segundo construtor deve receber, além do nome, o valor que o referido trabalhador deve receber por hora trabalhada. Identifique e implemente os demais métodos que achar conveniente para um objeto da classe *Funcionario*.

45. Crie a classe *FiguraGeometrica* que possui um método abstrato *descricao()*. Crie também as classes *Circulo*, *Quadrado* e *Triangulo* que são subclasses da classe *FiguraGeometrica* e implementam o método *descricao()* apropriado para sua classe. Por fim, crie uma classe *Principal* com um método *main* que cria um objeto de cada uma das classes e chama seus respectivos métodos *descricao()*.

    - O método *descricao()* deve exibir um texto que descreva a figura.

46. Crie uma classe *Desenho* que possui dois atributos do tipo *FiguraGeometrica* (criado na questão anterior) e suas respectivas coordenadas em um plano bidimensional. Escreva um construtor para a classe *Desenho* que inicialize todos os atributos através dos parâmetros. Implemente também o método *apresenta()* que, para cada *FiguraGeometrica* em um *Desenho*, informa suas coordenadas e imprime sua descrição. Por fim, crie uma classe executável, *Principal*, que cria dois objetos do tipo *Desenho* e chama seu meu método apresenta. O primeiro *Desenho* deve ser formado por um *Circulo* e um *Quadrado* e o segundo por um *Quadrado* e um *Triangulo*.

47. Crie uma interface *ItemDeBiblioteca* que declara quais campos e métodos uma classe que representa um item para empréstimo em uma biblioteca deve implementar. Essa interface é composta por um campo *maximoDeDiasParaEmprestimo* com valor 14 e os seguintes métodos:

    - *estaEmprestado* : retorna verdadeiro se o item estiver emprestado e falso caso contrário;
    - *empresta* : modifica para verdadeiro o estado de um campo que indica se o item está emprestado ou não;
    - *devolve* : modifica para falso o estado de um campo que indica se o item está emprestado ou não;
    - *localizacao* : retorna um texto que informa o local do item na biblioteca (e.g: "corredor 2, prateleira D");
    - *descricao* : retorna texto contendo uma descrição resumida do item (e.g.: "artigo da ECOP").

Implemente também a classe *Livro* que encapsula os dados genéricos sobre um livro e métodos para processar estes dados. Essa classe é composta pelos atributos *titulo*, *autor*, *numeroDePaginas* e *anoDaEdicao*, além dos seguintes métodos:

- construtor;
- *qualTitulo*: retorna o título do livro;
- *qualAutor*: retorna o autor do livro;
- *toString*: retorna os valores dos campos desta classe em formato textual.

Em seguida, escreva a classe *LivroDeBiblioteca* que herda os campos e métodos da classe *Livro* e implementa os métodos declarados na interface *ItemDeBiblioteca*. *LivroDeBiblioteca* também deve possuir um construtor e um método *toString*. Crie os atributos que forem necessários.

Por fim, crie a classe *DemoLivroDeBiblioteca* para demonstrar o uso de uma instância da classe *LivroDeBiblioteca*, isto é, criar um objeto do tipo *LivroDeBiblioteca* e executar seus métodos.

7 Coleções genéricas

48. Escreva um programa que cria um objeto *LinkedList* de 10 caracteres e, então, cria um segundo objeto *LinkedList* contendo uma cópia da primeira lista, mas na ordem inversa. Não devem ser utilizados métodos da Java API para realizar a inversão.

49. Escreva um programa que utilize a estrutura de dados do tipo Mapa para contar o número de ocorrências de cada letra em uma *String*. Por exemplo, a string *"HELLO THERE"* contém dois *H*'s, três *E*'s, dois *L*'s, um *O* , um *T* e um *R*. Exiba os resultados em ordem alfabética.

    - Serão totalmente desconsideradas respostas que não utilizarem um Mapa como elemento principal da estratégia de solução desta questão.

50. Escreva um programa que utilize uma pilha para verificar se uma *String* de entrada formada apenas por *'('* e *')'* está balanceada.

    É proibido utilizar contador(es) na solução desta questão.

    Os parênteses estão balanceados quando, na expressão, para cada abre parênteses há um correspondente fecha parênteses e os pares de parênteses estão aninhados.

    Exemplos de *Strings* de parênteses corretamente balanceadas:

    ```
    (()()()())
    (((()())))
    (()((())()))
    ```

    Exemplos de *Strings* de parênteses não são balanceadas:

    ```
    (((((((())
    ()))
    (()()(()
    ```

51. Escreva um programa que simule cada minuto de um dia de atendimento de um consultório de um médico com as seguintes especificações:

    - O consultório mantém apenas dados número de RG e idade de seus pacientes;
        (i) Crie uma classe para representar um paciente.
    - Pacientes maiores de 60 anos são colocados na fila prioritária e os demais na fila comum;
    - Pacientes da fila prioritária são sempre atendidos primeiro;
    - O primeiro paciente chega ao consultório no momento de sua abertura e a cada 4 minutos um novo paciente chega ao consultório;
        (i) Pesquise como gerar números aleatórios e utilize essa técnica para determinar o RG e a idade de cada paciente.
    - Uma consulta demora 5 minutos e o próximo paciente da fila é chamado;
    - O consultório atende 20 pacientes por dia.

8 Exceções

52. Faça um programa para somar dois números:

    - O programa deve conter um método *int obterIntValido()* para receber um número inteiro válido
        - Enquanto o valor não for um número inteiro, deve-se solicitar um novo número e exibir uma mensagem de erro.
    - O método principal deve utilizar *obterIntValido()* para obter os dois operandos e apresentar a soma deles
    - É obrigatório o uso de tratamento de exceção para realizar esta questão. Respostas que não utilizarem essa técnica serão completamente desconsideradas.

53. Escreva um programa que:

    - armazene em um vetor os nomes dos meses do ano
    - solicite ao usuário que digite um valor inteiro
    - mostre o nome do mês correspondente ao número digitado
    - trate as exceções geradas pela digitação de valores inválidos para o índice do mês

9 Referências

SANTOS, R. Introdução à programação orientada a objetos usando JAVA. 2. ed. Rio de Janeiro: Campus, 2013. 336p.

DEITEL, Paul; DEITEL, Harvey. Java: como programar. 10. ed. São Paulo: Pearson Education do Brasil, 2017.

BATISTA, Rogério da Silva; MORAES, Rafael Araújo de. Introdução à Programação Orientada a Objetos. 2013. Disponível em: http://proedu.rnp.br/handle/123456789/611. Acesso em: 18 ago. 2021.

SANTANCHÈ, André. Classes: lista de exercícios. 2011. Disponível em: https://www.ic.unicamp.br/~santanch/teaching/oop/exercicios/poo-exercicios-02-classes-v01.pdf. Acesso em: 30 mar. 2022.

BACALÁ JÚNIOR, Sílvio. Revisão de POO em Java: lista de exercícios 1. 2022. Disponível em: http://www.facom.ufu.br/~bacala/POO/lista1.pdf. Acesso em: 30 mar. 2022.

INSTITUTO METRÓPOLE DIGITAL. Programação Orientada a Objetos. 2015. Disponível em: https://materialpublic.imd.ufrn.br/curso/disciplina/5/8. Acesso em: 21 out. 2020. 2ª Edição. ISBN: 978-85-7064-002-4.

GARCIA, Islene Calciolari. MC202 - Estruturas de Dados: Lista de Exercícios 1. Disponível em: https://www.ic.unicamp.br/~islene/mc202/lista1.pdf. Acesso em: 24 maio 2022.

BACALÁ JÚNIOR, Sílvio. Exceções. Disponível em: https://www.facom.ufu.br/~bacala/POO/08-CriandoExcecoes.pdf. Acesso em: 31 maio 2022.

FERREIRA, Nickerson. Lista de Exercícios - Herança. Disponível em: https://docente.ifrn.edu.br/nickersonferreira/disciplinas/programacao-estruturada-e-orientada-a-objetos/lista-de-exercicios-heranca/. Acesso em: 8 mar. 2023.

USP. Parênteses Balanceados. Disponível em: https://panda.ime.usp.br/panda/static/pythonds_pt/03-EDBasicos/06-ParentesesBalanceados.html. Acesso em: 23 mar. 2024.