INSTITUTO FEDERAL DE MINAS GERAIS
CAMPUS SÃO JOÃO EVANGELISTA
SISTEMAS DE INFORMAÇÃO

VALDIR DE SOUZA CARVALHO NETO

Linguagem de Programação I - SI 241

Aula 8 - Exercícios de Aprendizagem

São João Evangelista
2025

SUMÁRIO

LISTA DE FIGURAS

1. **Exercício 1**

Na aula anterior, você implementou uma classe Stack em Java, com operações básicas como empilhar, desempilhar, verificar se está vazia e tratamento de exceções para situações como tentativa de desempilhar de uma pilha vazia. Nesta atividade, adicione um construtor de cópia à classe Stack, de forma que seja possível criar uma nova pilha a partir de outra já existente, copiando todos os elementos na mesma ordem. Após implementar o construtor, crie uma classe de teste que demonstre seu funcionamento. No método main, crie uma pilha original, adicione alguns elementos e, em seguida, utilize o construtor de cópia para criar uma nova pilha baseada na primeira. Mostre, por meio de saídas no console, que ambas as pilhas contêm os mesmos valores, mas são objetos independentes (ou seja, alterações em uma não afetam a outra).

Código:

Arquivo Main:

```
public class Ex01Main {
    public static void main(String[] args) {
        Ex01Stack Original = new Ex01Stack();
        for (int i = 100; i >= 10; i -= 10) {
            Original.push(i);
        }
        Ex01Stack Copia = new Ex01Stack(Original);

        try {
            System.out.println("Pilha original: ");
            System.out.println(Original.imprime());
            System.out.println("\nPilha cópia: ");
            System.out.println(Copia.imprime());
        } catch (Exception e) {
            System.out.println("Erro: " + e.getMessage());
        }

        try {
            Original.pop();
            Original.pop();
            Original.pop();
            Copia.push(0);

System.out.println("\n-----------------------------------------------
\n");
            System.out.println("Pilha original após exclusão de três
elementos: ");
```

```
            System.out.println(Original.imprime());
            System.out.println("\nPilha cópia após adição de um
elemento: ");
            System.out.println(Copia.imprime());
        } catch (Exception e) {
            System.out.println("Erro: " + e.getMessage());
        }
    }
}
```

Arquivo da Classe:

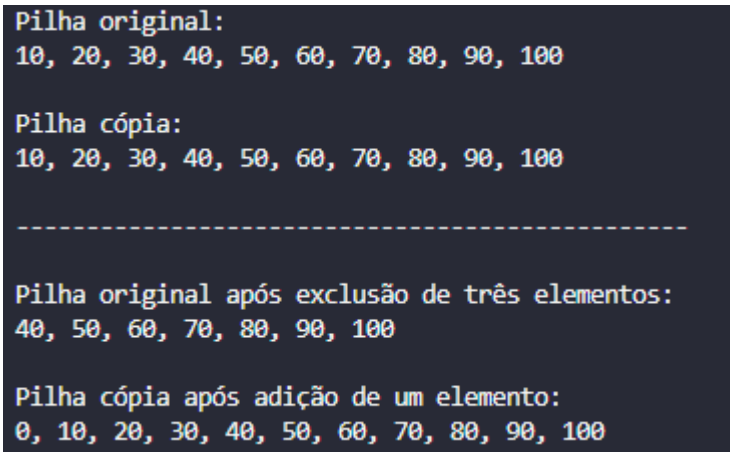
```
public class Ex01Stack {

    private Node top;
    private int size;

    public Ex01Stack() { // Criando uma pilha vazia
        this.top = null;
        this.size = 0;
    }

    public Ex01Stack(Ex01Stack copia) {
        Node t = copia.top;
        Ex01Stack S = new Ex01Stack();
        while (t != null) {
            S.push(t.item);
            t = t.next;
        }
        this.size = 0;
        this.top = null;
        try {
            while (!S.isEmpty()) {
                this.push(S.pop());
            }
        } catch (Exception e) {
        }
    }

    public void push(int item) { // Empilhar
        Node t = new Node(item);
        t.next = this.top;
```

```
        this.top = t;
        this.size++;
    }

    public boolean isEmpty() {
        return this.top == null;
    }

    public int size() {
        return this.size;
    }

    public int pop() throws Exception {
        if (this.isEmpty()) {
            throw new Exception("A pilha está vazia");
        }
        int item = this.top.item;
        this.top = this.top.next;
        this.size--;
        return item;
    }

    public int top() {
        if (this.isEmpty()) {
            throw new IllegalStateException("A pilha está vazia");
        }
        return this.top.item;
    }

    public String imprime() throws Exception {
        if (this.isEmpty()) {
            throw new Exception("A pilha está vazia");
        }
        String result = "";
        Node t = this.top;
        while (t != null) {
            Integer aux = t.item;
            result += aux.toString();
            if (t.next != null) {
                result += ", ";
            }
            t = t.next;
```

```
        }
        return result;
    }
}

class Node {
    public int item;
    public Node next;

    public Node(int item) {
        this.item = item;
        this.next = null;
    }
}
```

Imagens:

Figura 1. Captura de tela Atividade 01



Fonte: Captura de tela retirada pelo autor em 02 de novembro de 2025.

2. Exercício 2

Amplie a classe Stack criada anteriormente, implementando um construtor que receba um número inteiro n como parâmetro. Esse construtor deve inicializar a pilha vazia e, em seguida, empilhar automaticamente os números inteiros de 1 até n, nessa ordem (ou seja, o número 1 é empilhado primeiro e o número n ficará no topo). Após implementar o construtor, crie um programa de teste no método main que solicite ao usuário um valor n, crie uma nova pilha utilizando o construtor implementado e exiba os elementos empilhados. Em seguida, desempilhe todos os elementos mostrando a sequência de remoção, para confirmar que a pilha está funcionando corretamente e que os elementos foram empilhados na ordem esperada.

Código:

Arquivo Main:

```
import java.util.Scanner;

public class Ex02Main {
    public static void main(String[] args) {
        Scanner cin = new Scanner(System.in);
        int num = 0;
        System.out.print("Digite um número inteiro: ");
        num = cin.nextInt();
        Ex02Stack A = new Ex02Stack(num);

        try {
            System.out.println("\nPilha A: ");
            System.out.println(A.imprime());
        } catch (Exception e) {
            System.out.println("Erro: " + e.getMessage());
        }
        try {
            System.out.println("\nSequência de remoção dos elementos:
");
            while (!A.isEmpty()) {
                System.out.print(A.pop());
                if (!A.isEmpty()) {
                    System.out.print(", ");
                }
            }
            System.out.println("\n\nPilha A após remoção: ");
            System.out.println(A.imprime());
        } catch (Exception e) {
            System.out.println("Erro: " + e.getMessage());
        }
        cin.close();
    }
}
```

Arquivo da Classe:

```
public class Ex02Stack {

    private Node top;
```

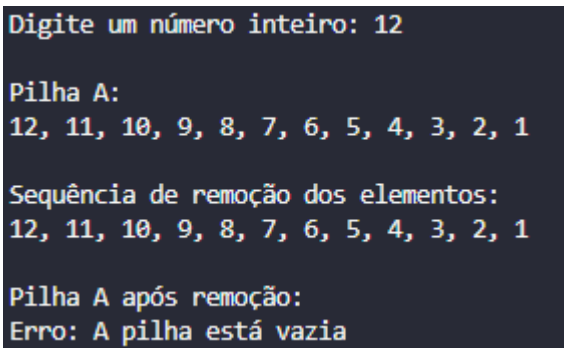
```
    private int size;

    public Ex02Stack() { // Criando uma pilha vazia
        this.top = null;
        this.size = 0;
    }

    public Ex02Stack(Ex02Stack copia) {
        Node t = copia.top;
        Ex02Stack S = new Ex02Stack();
        while (t != null) {
            S.push(t.item);
            t = t.next;
        }
        this.size = 0;
        this.top = null;
        try {
            while (!S.isEmpty()) {
                this.push(S.pop());
            }
        } catch (Exception e) {
        }
    }

    public Ex02Stack(int n) {
        this.top = null;
        this.size = 0;
        if (n > 0) {
            for (int i = 1; i <= n; i++) {
                this.push(i);
            }
        } else {
            for (int i = 1; i >= n; i--) {
                this.push(i);
            }
        }
    }

    public void push(int item) {
        Node t = new Node(item);
        t.next = this.top;
        this.top = t;
```

```
        this.size++;
    }

    public boolean isEmpty() {
        return this.top == null;
    }

    public int size() {
        return this.size;
    }

    public int pop() throws Exception {
        if (this.isEmpty()) {
            throw new Exception("A pilha está vazia");
        }
        int item = this.top.item;
        this.top = this.top.next;
        this.size--;
        return item;
    }

    public int top() {
        if (this.isEmpty()) {
            throw new IllegalStateException("A pilha está vazia");
        }
        return this.top.item;
    }

    public String imprime() throws Exception {
        if (this.isEmpty()) {
            throw new Exception("A pilha está vazia");
        }
        String result = "";
        Node t = this.top;
        while (t != null) {
            Integer aux = t.item;
            result += aux.toString();
            if (t.next != null) {
                result += ", ";
            }
            t = t.next;
        }
```

```
        return result;
    }
}

class Node {
    public int item;
    public Node next;

    public Node(int item) {
        this.item = item;
        this.next = null;
    }
}
```

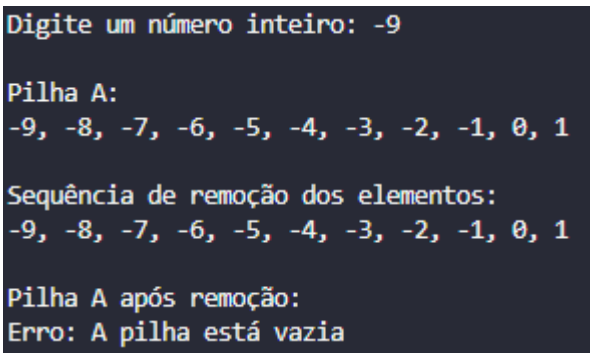
Imagens:

Figura 2. Captura de tela Atividade 02



Fonte: Captura de tela retirada pelo autor em 02 de novembro de 2025.

Figura 3. Captura de tela Atividade 02



Fonte: Captura de tela retirada pelo autor em 02 de novembro de 2025.

3. Exercício 3

Agora, modifique a classe Stack já desenvolvida, incluindo um método estático chamado equals que receba como parâmetros dois objetos do tipo Stack e retorne um valor lógico (true ou false) indicando se as pilhas são iguais ou diferentes. Duas pilhas devem ser consideradas iguais se possuírem o mesmo número de elementos e se todos os elementos corresponderem exatamente na mesma ordem, do topo até a base. O método não deve alterar o conteúdo das pilhas comparadas; caso seja necessário percorrer os elementos, utilize pilhas auxiliares para preservar o estado original. No método main, crie duas instâncias da classe Stack, empilhando alguns valores em cada uma delas (você pode utilizar o construtor que empilha automaticamente os números de 1 até n). Em seguida, invoque o método equals para comparar as duas pilhas e exiba uma mensagem informando se elas são iguais ou diferentes.

Código:

Arquivo Main:

```
public class Ex03Main {
    public static void main(String[] args) {
        Ex03Stack A = new Ex03Stack(10);
        Ex03Stack B = new Ex03Stack(12);

        try {
            System.out.println("\nPilha A: ");
            System.out.println(A.imprime());

            System.out.println("\nPilha B: ");
            System.out.println(B.imprime());

            if (Stack3.equals(A, B)) {
                System.out.println("As pilhas são iguais!");
            } else {
                System.out.println("As pilhas não são iguais!");
            }
        } catch (Exception e) {
            System.out.println("Erro: " + e.getMessage());
        }

    }
}
```

Arquivo da Classe:

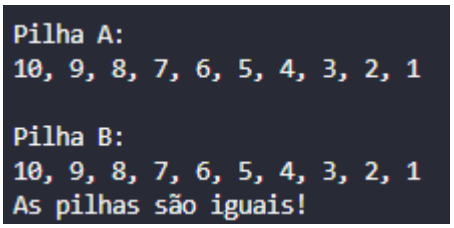
```java
public class Ex03Stack {

    private Node top;
    private int size;

    public Ex03Stack() { // Criando uma pilha vazia
        this.top = null;
        this.size = 0;
    }

    public Ex03Stack(Ex03tack copia) {
        Node t = copia.top;
        Ex03Stack S = new Ex03Stack();
        while (t != null) {
            S.push(t.item);
            t = t.next;
        }
        this.size = 0;
        this.top = null;
        try {
            while (!S.isEmpty()) {
                this.push(S.pop());
            }
        } catch (Exception e) {
        }
    }

    public Ex03Stack(int n) {
        this.top = null;
        this.size = 0;
        if (n > 0) {
            for (int i = 1; i <= n; i++) {
                this.push(i);
            }
        } else {
            for (int i = 1; i >= n; i--) {
                this.push(i);
            }
        }
    }

    public void push(int item) {
```

```
        Node t = new Node(item);
        t.next = this.top;
        this.top = t;
        this.size++;
    }

    public boolean isEmpty() {
        return this.top == null;
    }

    public int size() {
        return this.size;
    }

    public int pop() throws Exception {
        if (this.isEmpty()) {
            throw new Exception("A pilha está vazia");
        }
        int item = this.top.item;
        this.top = this.top.next;
        this.size--;
        return item;
    }

    public int top() {
        if (this.isEmpty()) {
            throw new IllegalStateException("A pilha está vazia");
        }
        return this.top.item;
    }

    public String imprime() throws Exception {
        if (this.isEmpty()) {
            throw new Exception("A pilha está vazia");
        }
        String result = "";
        Node t = this.top;
        while (t != null) {
            Integer aux = t.item;
            result += aux.toString();
            if (t.next != null) {
                result += ", ";
```

```
            }
            t = t.next;
        }
        return result;
    }

    public static boolean equals(Stack3 A, Stack3 B) {
        int cont = 1;
        Ex03Stack A1 = new Ex03Stack(A);
        Ex03Stack B1 = new Ex03Stack(B);
        if (A1.size == B1.size) {
            while (!A1.isEmpty()) {
                try {
                    if (A1.pop() == B1.pop()) {
                        cont = 0;
                    }
                } catch (Exception e) {
                }
            }
        }
        if (cont == 0) {
            return true;
        } else {
            return false;
        }
    }
}

class Node {
    public int item;
    public Node next;

    public Node(int item) {
        this.item = item;
        this.next = null;
    }
}
```

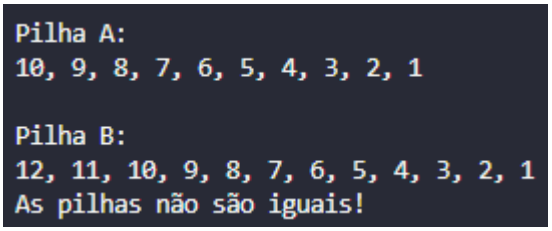
Imagens:

Figura 4. Captura de tela Atividade 03



Fonte: Captura de tela retirada pelo autor em 02 de novembro de 2025.

Figura 5. Captura de tela Atividade 03



Fonte: Captura de tela retirada pelo autor em 02 de novembro de 2025.

4. Exercício 4

Crie um programa em Java com classes de um sistema simples de gerenciamento de alunos de uma universidade. Cada aluno deve ser representado por uma classe contendo atributos como nome, matrícula, curso e idade. Utilize o conceito de encapsulamento para garantir que os atributos só possam ser acessados por meio de métodos públicos (getters e setters). Em seguida, crie um método para exibir as informações do aluno de forma formatada. No método main, instancie pelo menos três objetos da classe Aluno e exiba seus dados na tela.

Código:

Arquivo Main:

```
public class Ex04Main {
        public static void main(String[] args) {
                Ex04Aluno Aluno1 = new Ex04Aluno();
                Aluno1.setNome("Valdir de Souza Carvalho Neto");
                Aluno1.setMatricula(1);
                Aluno1.setCurso("Licenciatura em Matemática");
                Aluno1.setIdade(21);

                Ex04Aluno Aluno2 = new Ex04Aluno();
                Aluno2.setNome("Patrícia da Silva Costa");
                Aluno2.setMatricula(2);
                Aluno2.setCurso("Bacharelado em Sistemas de
```

```
Informação");
                Aluno2.setIdade(20);

                Ex04Aluno Aluno3 = new Ex04Aluno();
                Aluno3.setNome("Juliana Gabriel Lopes");
                Aluno3.setMatricula(3);
                Aluno3.setCurso("Licenciatura em Ciências
Biológicas");
                Aluno3.setIdade(42);

                System.out.print(Aluno1.exibeInformacoes());
                System.out.print(Aluno2.exibeInformacoes());
                System.out.print(Aluno3.exibeInformacoes());
        }
}
```

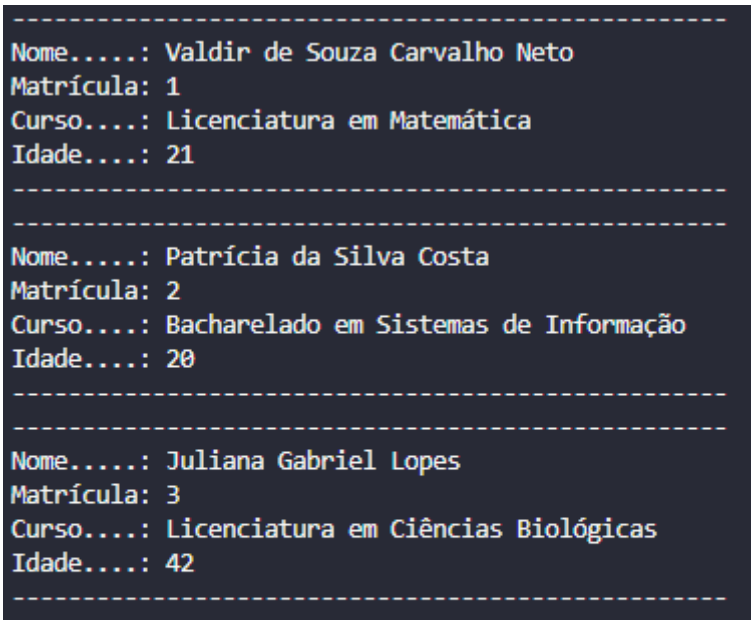
Arquivo da Classe:

```
public class Ex04Aluno {
    private String nome;
    private int matricula;
    private String curso;
    private int idade;

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getNome() {
        return nome;
    }

    public void setMatricula(int matricula) {
        this.matricula = matricula;
    }

    public int getMatricula() {
        return matricula;
    }

    public void setCurso(String curso) {
        this.curso = curso;
```

```
    }

    public String getCurso() {
        return curso;
    }

    public void setIdade(int idade) {
        this.idade = idade;
    }

    public int getIdade() {
        return idade;
    }

    public String exibeInformacoes() {
        String result = "";
        result +=
"-------------------------------------------------------\n";
        result += "Nome.....: " + getNome() + "\n";
        result += "Matrícula: " + getMatricula() + "\n";
        result += "Curso....: " + getCurso() + "\n";
        result += "Idade....: " + getIdade() + "\n";
        result +=
"-------------------------------------------------------\n";
        return result;
    }
}
```

Imagens:

Figura 6. Captura de tela Atividade 04



Fonte: Captura de tela retirada pelo autor em 02 de novembro de 2025.

5. Exercício 5

Amplie o exercício anterior incluindo construtores sobrecarregados na classe Aluno: um construtor sem parâmetros, um que receba apenas nome e matrícula, e outro que receba todos os atributos. No programa principal, crie objetos utilizando cada um dos construtores e verifique como a sobrecarga facilita a criação de instâncias com diferentes quantidades de informações.

Código:

Arquivo Main:

```
public class Ex05Main {
    public static void main(String[] args) {
        Ex04Aluno Aluno1 = new Ex04Aluno();
        Aluno1.setNome("Valdir de Souza Carvalho Neto");
        Aluno1.setMatricula(1);
        Aluno1.setCurso("Licenciatura em Matemática");
        Aluno1.setIdade(21);
        Ex05Aluno Aluno2 = new Ex05Aluno("Patrícia da Silva Costa",
2);
        Aluno2.setCurso("Bacharelado em Sistemas de Informação");
        Aluno2.setIdade(20);
        Ex05Aluno Aluno3 = new Ex05Aluno("Juliana Gabriel Lopes", 3,
"Licenciatura em Ciências Biológicas", 42);
```

```
        System.out.print(Aluno1.exibeInformacoes());
        System.out.print(Aluno2.exibeInformacoes());
        System.out.print(Aluno3.exibeInformacoes());
    }
}
```

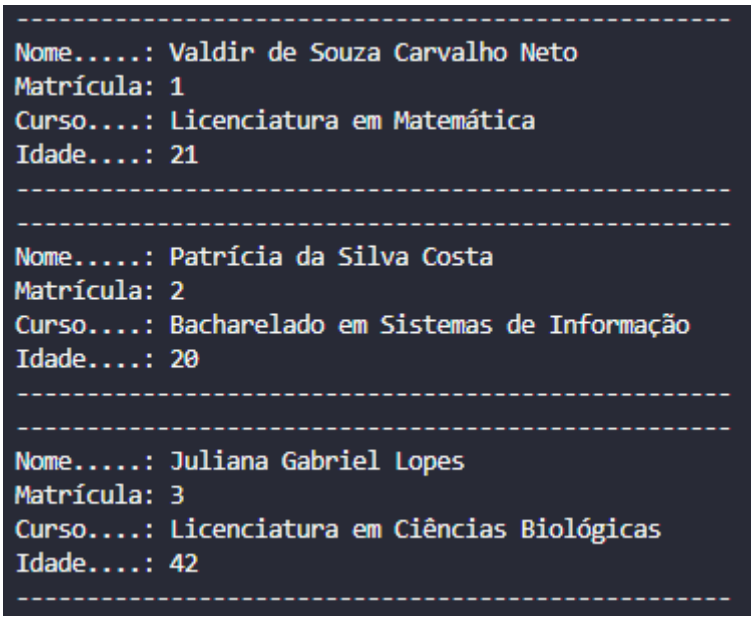
Arquivo da Classe:

```
public class Ex05Aluno {
    private String nome;
    private int matricula;
    private String curso;
    private int idade;

    public Ex05Aluno() {
    }

    public Ex05Aluno(String nome, int matricula) {
        this.setNome(nome);
        this.setMatricula(matricula);
    }

    public Ex05Aluno(String nome, int matricula, String curso, int
idade) {
        this.setNome(nome);
        this.setMatricula(matricula);
        this.setCurso(curso);
        this.setIdade(idade);
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getNome() {
        return nome;
    }

    public void setMatricula(int matricula) {
        this.matricula = matricula;
    }
```

```
    public int getMatricula() {
        return matricula;
    }

    public void setCurso(String curso) {
        this.curso = curso;
    }

    public String getCurso() {
        return curso;
    }

    public void setIdade(int idade) {
        this.idade = idade;
    }

    public int getIdade() {
        return idade;
    }

    public String exibeInformacoes() {
        String result = "";
        result +=
"-------------------------------------------------\n";
        result += "Nome.....: " + getNome() + "\n";
        result += "Matrícula: " + getMatricula() + "\n";
        result += "Curso....: " + getCurso() + "\n";
        result += "Idade....: " + getIdade() + "\n";
        result +=
"-------------------------------------------------\n";
        return result;
    }
}
```

Imagens:

Figura 7. Captura de tela Atividade 05



Fonte: Captura de tela retirada pelo autor em 02 de novembro de 2025.

6. Exercício 6

Crie uma classe chamada Universidade com um atributo estático que armazene o nome da instituição. Faça com que cada objeto da classe Aluno utilize esse atributo para exibir, junto de suas informações, o nome da universidade. Além disso, crie um método estático na classe Universidade que permita alterar o nome da instituição e verifique como essa alteração afeta todos os objetos já criados.

Código:

Arquivo Main:

```
public class Ex06Main {
    public static void main(String[] args) {

        Ex06Classes Aluno1 = new Ex06Classes("Valdir de Souza Carvalho
Neto", 1, "Licenciatura em Matemática", 21);
        Ex06Classes Aluno2 = new Ex06Classes("Patrícia da Silva
Costa", 2, "Bacharelado em Sistemas de informação", 20);
        Ex06Classes Aluno3 = new Ex06Classes("Juliana Gabriel Lopes",
3, "Licenciatura em Ciências Biológicas", 42);

        System.out.print(Aluno1.exibeInformacoes());
        System.out.print(Aluno2.exibeInformacoes());
        System.out.print(Aluno3.exibeInformacoes());
```

```
        Universidade.setNomeInstituicao("IFMG - Campus Bambuí");

        System.out.print(Aluno1.exibeInformacoes());
        System.out.print(Aluno2.exibeInformacoes());
        System.out.print(Aluno3.exibeInformacoes());
    }
}
```

Arquivo da Classe:

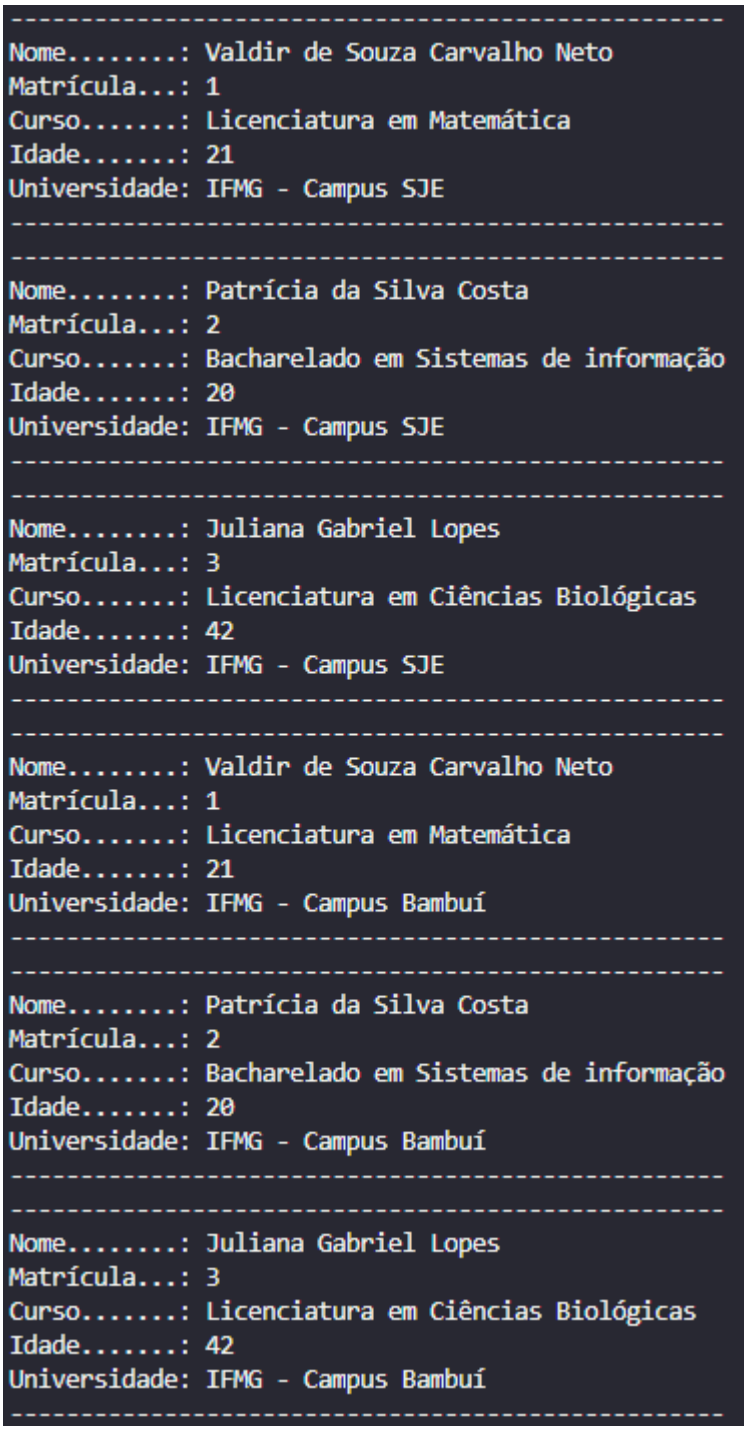
```
public class Ex06Classes {
    private String nome;
    private int matricula;
    private String curso;
    private int idade;

    public Ex06Classes() {
    }

    public Ex06Classes(String nome, int matricula) {
        this.setNome(nome);
        this.setMatricula(matricula);
    }

    public Ex06Classes(String nome, int matricula, String curso, int
idade) {
        this.setNome(nome);
        this.setMatricula(matricula);
        this.setCurso(curso);
        this.setIdade(idade);
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getNome() {
        return nome;
    }

    public void setMatricula(int matricula) {
        this.matricula = matricula;
```

```
    }

    public int getMatricula() {
        return matricula;
    }

    public void setCurso(String curso) {
        this.curso = curso;
    }

    public String getCurso() {
        return curso;
    }

    public void setIdade(int idade) {
        this.idade = idade;
    }

    public int getIdade() {
        return idade;
    }

    public String exibeInformacoes() {
        String result = "";
        result +=
"-----------------------------------------------------\n";
        result += "Nome.....: " + getNome() + "\n";
        result += "Matrícula...: " + getMatricula() + "\n";
        result += "Curso.....: " + getCurso() + "\n";
        result += "Idade.....: " + getIdade() + "\n";
        result += "Universidade: " + Universidade.nomeInstituicao +
"\n";
        result +=
"-----------------------------------------------------\n";
        return result;
    }
}

class Universidade {
    static String nomeInstituicao = "IFMG - Campus SJE";

    public static void setNomeInstituicao(String nomeInstituicao) {
```

```
        Universidade.nomeInstituicao = nomeInstituicao;
    }
}
```

Imagens:

Figura 8. Captura de tela Atividade 06



Fonte: Captura de tela retirada pelo autor em 02 de novembro de 2025.

7. **Exercício 7**

Explique, com suas próprias palavras, os conceitos de checked exceptions e unchecked exceptions na linguagem Java, destacando as diferenças entre esses dois tipos de exceção e apresentando exemplos que ilustrem cada caso. Em seguida, discuta de que forma a distinção entre exceções verificadas e não verificadas pode contribuir para a adoção de boas práticas de programação, como a escrita de códigos mais seguros, organizados e fáceis de manter. Seu texto deve evidenciar a importância do tratamento adequado de erros para garantir maior confiabilidade e estabilidade na execução de aplicações Java.

Resposta: Em Java, exceções são eventos anormais que interrompem o fluxo do programa e dividem-se em verificadas (checked) e não verificadas (unchecked).

- **Checked Exceptions (Verificadas):** São exceções que o compilador Java obriga o programador a tratar. Elas geralmente representam erros previsíveis e externos, como falhas de I/O. O tratamento deve ser feito usando *try/catch* ou declarando *throws* na assinatura do método para delegar a responsabilidade.
    - **Exemplo:** *IOException*. Ao tentar ler um arquivo (ex: com *FileReader*), o método deve tratar essa exceção, caso contrário, o programa *nem compila*.
- **Unchecked Exceptions (Não Verificadas):** São exceções que o compilador não exige tratamento. Elas derivam de *RuntimeException* e, na maioria das vezes, representam erros de lógica interna do programa.
    - **Exemplo:** *ArithmeticException* (divisão por zero) ou *NullPointerException*. Se não forem tratadas, causam um erro em tempo de execução que finaliza o programa.

Importância para Boas Práticas: Essa distinção ajuda a criar códigos mais seguros e organizados. As **checked exceptions** forçam o desenvolvedor a lidar com erros recuperáveis (como um arquivo não encontrado), garantindo que o programa saiba como reagir sem falhar abruptamente. As **unchecked exceptions** sinalizam erros de lógica (bugs) que devem ser corrigidos no código, e não apenas capturados. Esse tratamento adequado de erros é vital para a confiabilidade e estabilidade da aplicação.

8. **Exercício 8**

Explique, com suas próprias palavras, a diferença entre o uso das palavras-chave throw e throws no contexto do tratamento de exceções em Java.

Resposta: A diferença principal é que *throw* executa o lançamento de uma exceção, enquanto *throws* declara que um método pode lançar uma exceção.

- ***throw*:** É uma instrução usada *dentro* de um bloco de código para lançar manualmente uma exceção. É a ação de disparar o erro.
    - **Exemplo:** *if (idade < 18) { throw new IllegalArgumentException("Menor de idade!"); }*.
- ***throws*:** É uma palavra-chave usada na assinatura (declaração) de um método. Ela *avisa* que aquele método pode lançar uma exceção e delega a responsabilidade de tratamento (com *try/catch*) para o método que o chamou.
    - **Exemplo:** *public static void lerArquivo() throws IOException*.