INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO RIO GRANDE DO NORTE

CAMPUS AVANÇADO LAJES

CURSO TÉCNICO INTEGRADO EM INFORMÁTICA

JOÃO VICTOR DA CUNHA DA COSTA

RELATÓRIO DA PRÁTICA PROFISSIONAL: MONITORIA DA DISCIPLINA DE PROGRAMAÇÃO ESTRUTURADA E ORIENTADA A OBJETOS

LAJES/RN

2021

JOÃO VICTOR DA CUNHA DA COSTA

RELATÓRIO DA PRÁTICA PROFISSIONAL: MONITORIA DA DISCIPLINA DE PROGRAMAÇÃO ESTRUTURADA E ORIENTADA A OBJETOS

Relatório de Prática Profissional apresentado ao Curso Técnico Integrado em Informática do Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte, Campus Avançado Lajes, em cumprimento às exigências legais como requisito parcial à obtenção do título de Técnico em Informática.

Orientador (a): Prof. Me. Fernando Helton Linhares Soares.

LAJES/RN

2021

RESUMO

O presente relatório de Monitoria da disciplina de Programação Estruturada e Orientada a Objetos, ofertada pelo curso técnico integrado de informática do IFRN campus avançado Lajes, apresenta informações acerca do período de auxílio prestado pelos monitores da referida disciplina. O objetivo geral da prática profissional consistiu em apoiar os estudantes na realização das atividades exigidas pelo professor encarregado da matéria, além de cooperar na construção de conhecimento teórico e prático dos discentes. Durante o desenvolvimento deste trabalho foi possível expor a definição e as funções dos instrumentos utilizados, da mesma forma os conteúdos abordados nos dois módulos em que se foi realizado às aulas. Para se ofertar a monitoria durante o ensino remoto, foi necessário a utilização de ferramentas de comunicação, dessa forma foi possível exercer a prática proposta. À vista disso, pode-se formular a experiência do monitor com o exercício docente, avaliando suas perspectivas e visões. Por último, é válido salientar a importância dessa prática para a geração de profissionais da área de programação, tanto desenvolvedores quanto lecionadores.

Palavras-chave: Monitoria. Programação. Prática profissional.

LISTA DE ILUSTRAÇÕES

Figura 1 - EXEMPLO DE USO DE ENTRADA/SAÍDA 11

Figura 2 - COMANDO IF 13

Figura 3 - COMANDO IF ELSE 13

Figura 4 - REPRESENTAÇÃO GRÁFICA DO COMANDO WHILE 14

Figura 5 - CÓDIGO DO COMANDO WHILE 14

Figura 6 - ESTRUTURA DO COMANDO FOR 15

Figura 7 - EXEMPLO DO USO DE CONSTRUTOR 17

Figura 8 - EXEMPLO DE APLICAÇÃO DE INTERFACE 22

Figura 9 - USO DE UM TRY CATCH 23

Quadro 1 - SÍNTESE DAS ATIVIDADES DO ALUNO NO PROJETO 08

Quadro 2 - OPERADORES ARITMÉTICOS EM JAVA 12

Quadro 3 - TIPOS DE ATRIBUTOS 17

Quadro 4 - ORDEM DE RESTRIÇÕES 19

Sumário

2. DADOS GERAIS DA MONITORIA 8
3. FUNDAMENTAÇÃO TEÓRICA 9
3.1 JAVA 9
3.1.1 Contexto Histórico 9
3.1.2 Java e a Orientação a Objetos 9
3.2 Variáveis, Entrada e Saída e Operadores Aritméticos 10
3.2.1 Variáveis 10
3.2.2 Entrada e Saída 11
3.2.3 Operadores Aritméticos 11
3.3 Estrutura Condicional 12
3.4 Repetição Condicional 13
3.5 Repetição Contada 14
3.6 Vetores Arrays 15
3.7 Orientação a Objetos 15
3.7.1 Classes 16
3.7.2 Atributos 16
3.7.3 Métodos 17
3.7.4 Objeto 17
3.8 Array de Objetos 18
3.9 Modificadores de acesso, construtores e atributos de classe 18
3.9.1 Modificadores de acesso 18
3.9.2 Construtores 19
3.9.3 Atributos de classe 20
3.10 Herança e Polimorfismo 20
3.10.1 Herança 20
3.10.2 Polimorfismo 20
3.11 Classes Abstratas e Interfaces 21
3.11.1 Classes Abstratas 21
3.11.2 Interfaces 21
3.12 Exceções 22
3.13 Collections 23
3.13.1 ArrayList 23

4. METODOLOGIA 24
4.1 Instrumentos utilizados 24
4.1.1 WhatsApp 24
4.1.2 Google Meet 24
4.1.3 Eclipse 24
5. CARACTERIZAÇÃO DAS ATIVIDADES DESENVOLVIDAS 25
5.1 Atendimento ao aluno 25
5.1.1 WhatsApp 25
5.1.2 Google Meet 25
5.1.3 Eclipse 25
5.2 Reunião 26
5.3 Resultados 26
5.3.1 Positivos 26
5.3.2 Negativos 26
6 ANÁLISE E DISCUSSÃO DOS RESULTADOS 27
6.1 Alunos 27
6.2 Impacto nos monitores 27
7 CONSIDERAÇÕES FINAIS 28
REFERÊNCIAS 29
ANEXO A – Formulário de identificação 31

1. INTRODUÇÃO

Muitas instituições de ensino técnico e/ou profissionalizantes costumam ofertar monitorias, tendo como base normas definidas pelos órgãos responsáveis pelas mesmas. Embora haja pequenas diferenças entre as aplicações e regras entre as monitorias, elas sempre pretendem auxiliar os discentes na construção do conhecimento em volta da disciplina e contribuem na formação pedagógica do aluno-monitor, possibilitando o seu aprimoramento na área e incentivando os mesmos a serem docentes capacitados futuramente (FARIA e SCHNEIDER, 2004).

A monitoria é definida como uma ação/atividade extraclasse, que tem como objetivo resgatar as dificuldades tidas fora da sala de aula e buscar auxiliar o corpo discente, assim como especular medidas capazes de amenizar essas problemáticas (SCHNEIDER, Marcia, 2006).

O presente relatório objetiva apresentar a estruturação das atividades realizadas na Monitoria da disciplina de Programação Estruturada e Orientada a Objetos (PEOO), serviço este realizado durante o primeiro e segundo módulo do ano letivo de 2021, na instituição de ensino IFRN - campus avançado Lajes.

Durante a monitoria, foram dedicadas 15 horas semanais voltadas para sancionar as diversas dúvidas dos alunos do curso de informática que estavam matriculados na disciplina de PEOO, ajudando assim os estudantes a ter uma maior facilidade na obtenção de conhecimento, e atribuindo experiência aos monitores na área de ensino-aprendizagem.

O trabalho apresentado será abordado por capítulos, tendo em sua composição como primeiro capítulo a Introdução, em segundo plano veremos os dados gerais da monitoria, por conseguinte a fundamentação teórica, posteriormente a metodologia, caracterização das atividades realizadas, análise e discussão, e por último as considerações finais.

2. DADOS GERAIS DA MONITORIA

Título do projeto: Monitoria Voluntária da disciplina de Programação Estruturada e Orientada a Objetos

Período de realização: de 31/05/2021 a 08/10/2021

Total de horas: 170 horas

Orientador: Prof. Me. Fernando Helton Linhares Soares

Função: Monitor

Quadro 1 – SÍNTESE DAS ATIVIDADES DO ALUNO NO PROJETO.

CARGA HORÁRIA	**ATIVIDADES DESENVOLVIDAS**	**RESULTADOS ALCANÇADOS**
170 horas	Tirar dúvidas dos alunos na disciplina de Programação e Estruturada Orientada a Objetos.	Dúvidas sanadas.

FONTE: AUTORIA PRÓPRIA (2021).

3. FUNDAMENTAÇÃO TEÓRICA

3.1 JAVA

Java é uma linguagem de programação orientada a objetos. Muito usada por empresas e de forma individual, ela é considerada a maior plataforma para desenvolvimento atualmente.

3.1.1 Contexto Histórico

Essa linguagem de programação começou a ser desenvolvida em 1991.

> Teve início com o Green Project, no qual os mentores foram Patrick Naughton, Mike Sheridan, e James Gosling. Este projeto não tinha intenção de criar uma linguagem de programação, mas sim de antecipar a "próxima onda" que aconteceria na área da informática e programação. Os idealizadores do projeto acreditavam que em pouco tempo os aparelhos domésticos e os computadores teriam uma ligação (INFOESCOLA, s.d).

Nesse período, a globalização da internet estava dando mais um passo à frente, e aproveitando a deixa, o grupo do Green Project avançou quando as aplicações para o Oak na internet, com o foco principal na interação. Após esses avanços, como uma versão mais completa do Oak para internet, houve o lançamento do Java em 1995.

Com o sucesso do Java, empresas de grande porte passaram a dar apoio e a produzir diferentes tipos de produtos feitos com a linguagem Java. Além disso, diversos desenvolvedores também passaram a utilizá-lo. "Estimativas apontam que a tecnologia Java foi a mais rapidamente incorporada na história da informática" (INFOESCOLA, c2021).

Podemos considerar o Java como uma das maiores revoluções tecnológicas já vista, mediante aos fatos apresentados. Até hoje a ferramenta é utilizada por muitos, e os números aumentam a cada dia. Sua linguagem de fácil entendimento e desempenho rápido, possibilita criar variados programas e aplicações.

3.1.2 Java e a Orientação a Objetos

É válido ressaltar que a linguagem java foi criada a partir do paradigma de orientação a objetos, tendo como devolutiva a capacidade de programadores

poderem usar conceitos como polimorfismo, por exemplo. Esse paradigma já existia antes mesmo do desenvolvimento do Java, entretanto, só passou a ter fama após o fortalecimento dessa tecnologia. (MENDES, Douglas, 2009, p. 18)

"Quando desenvolvemos programas orientados a objetos e estruturadas temos dois paradigmas totalmente diferentes. A forma de pensar e escrever o código são diferentes" (MENDES, Douglas, 2009, p. 18).

3.2 Variáveis, Entrada e Saída e Operadores Aritméticos

3.2.1 Variáveis

Variáveis são nomes referentes a endereços de dados, pois os endereços costumam ser apenas em números.Goldman, Kon e Silva dão um exemplo:

> Se definimos uma variável i do tipo int, ela pode ser associada, por exemplo, ao endereço de memória 3.745.908. A partir daí, quando atribuímos um valor a esta variável, este valor é armazenado nos quatro bytes (porque um int Java ocupa 4 bytes) a partir do endereço 3.745.908 da memória (GOLDMAN, Alfredo et al. p.76).

Com base neste exemplo, é possível contestar a importância desses nomes associados, pois seria inviável o uso dos endereços de memória em si, e que também não é possível salvar um tipo diferente numa variável que não seja o dela.

No caso de variáveis de tipo primitivo, como int, float, double e etc, os valores são sempre salvos nas posições de memórias correspondentes, pois são simples de se lidar e ocupam pouca memória. Mas o mesmo não é verdade para os mais complexos, como arrays e objetos criados pelo programador, nesses, a situação é um pouco diferente:

> Java não guarda nas variáveis uma cópia do objeto, mas sim o endereço de memória no qual este objeto está armazenado. Por este motivo, fala-se que em Java as variáveis associadas a objetos complexos são referências para estes objetos. No momento em que um objeto precisa ser usado, o interpretador Java o localiza a partir de uma referência para ele armazenada em uma variável que, por sua vez, está também armazenada em um endereço de memória (GOLDMAN, Alfredo et al. p.76).

Assim, podemos entender que quando uma variável primitiva copia o valor de outra, o que ocorre é realmente uma cópia dos dados do valor armazenado, que é realocado para outra variável, mas no caso das variáveis complexas, o que ocorre é

uma cópias de referência, um caminho para a variável original, e não uma cópia real de dados.

3.2.2 Entrada e Saída

Explicando de uma forma geral, dispositivos e interfaces de entrada/saída são aparelhos e programas que permitem a interação entre o homem e a máquina, com os de entrada sendo responsáveis por enviar dados para o computador (usaremos um teclado como exemplo), enquanto os de saída tem o encargo de retornar os dados da máquina para o homem (a caixa de fone retorna uma voz eletrônica que recita o que o homem escreveu).

Na programação Java, nós temos o Scanner, que pode ler o que é digitado pelo usuário no console (entrada), e o System.out.println (saída) que escreve textos no console.

Nesta imagem, é criado uma instância da classe Scanner, e logo abaixo, ela é usada para salvar um valor digitado no console em uma variável inteira x; em seguida, é usado o System.out.println para escrever no console a frase" O inteiro lido foi " com o valor digitado pelo usuário em seguida no console; esse processo se repete duas vezes, porém com tipos diferentes de variáveis. (FIGURA 1)

FIGURA 1 - EXEMPLO DE USO DE ENTRADA/SAÍDA

```
public static void main(String [] args)
{
  Scanner sc = new Scanner(System.in);
  int x = sc.nextInt();
  System.out.println(" O inteiro lido foi " + x);
  double d = sc.nextDouble();
  System.out.println(" O double lido foi " + d);
  String s = sc.next();
  System.out.println(" A string lida foi " + s);
}
}
```

FONTE: GOLDMAN, Alfredo (2004, p.91)

3.2.3 Operadores Aritméticos

Operações Aritméticas são representações das operações matemáticas básicas usadas nas linguagens de programação, como adição e subtração, pois nem todos os sinais usados na programação são iguais aos sistemas matemáticos.

QUADRO 2 - OPERADORES ARITMÉTICOS EM JAVA

Operador	Exemplo de código	Descrição
+	A1 + A2	Adição
-	A1 - A2	Subtração
*	A1 * A2	Multiplicação
/	A1 / A2	Divisão. Retorna sempre um valor inteiro(arredondado para cima)
%	A1 % A2	Resto de uma divisão

FONTE: AUTORIA PRÓPRIA (2021)

3.3 Estrutura Condicional

A estrutura de controle possibilita que um conjunto de ações sejam executadas, a partir de condições, sendo elas sentenças lógicas, ou seja, verdadeiro ou falso. Os comandos geralmente utilizados para compor essas expressões são o *if* e *else*, além do comando de *switch*, todos estes executam um bloco de códigos de acordo com uma decisão/validação.

“O uso de cada um dos comandos depende da composição da estrutura de decisão” (MENDES, Douglas, 2009, p. 43). No uso do *if* (se) ele condiciona um certo comando para uma condição eventual, sendo ele executado caso seja verdadeiro (FIGURA 2). Quanto ao uso do *else (senão)*, se o resultado da execução for falso, utiliza-o (FIGURA 3). Vale ressaltar que este comando só pode ser utilizado junto com a estrutura *if.*

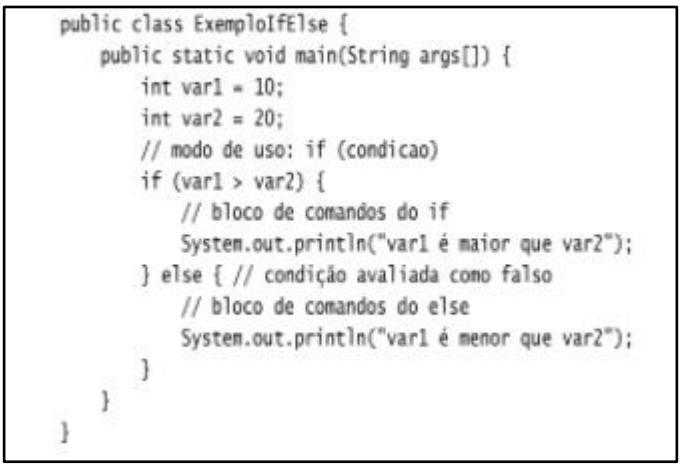
FIGURA 2 - COMANDO IF

```
public class ExemploIf {
    public static void main(String args[]) {
        int var1 = 20;
        int var2 = 10;
        // modo de uso: if (condicao)
        if (var1 > var2) {

        // bloco de comandos do if
        System.out.println("var1 é maior que var2");
    }
  }
}
```

FONTE: MENDES, Douglas (2009, p. 42-43)

FIGURA 3 - COMANDO IF ELSE



```
public class ExemploIfElse {
    public static void main(String args[]) {
        int var1 = 10;
        int var2 = 20;
        // modo de uso: if (condicao)
        if (var1 > var2) {
            // bloco de comandos do if
            System.out.println("var1 é maior que var2");
        } else { // condição avaliada como falso
            // bloco de comandos do else
            System.out.println("var1 é menor que var2");
        }
    }
}
```

FONTE: MENDES, Douglas (2009, p. 43)

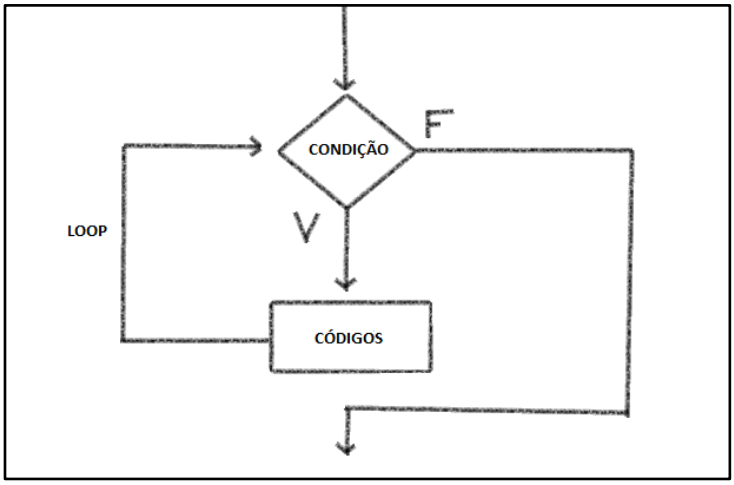
Ao usarmos o comando *switch* avaliamos se existe apenas uma variável e se esta está sendo igualada. Outras circunstâncias são necessárias e atribuídas quando utilizamos *switch*, como cita Douglas Mendes:

> Quando usamos o comando switch devemos também usar em conjunto o comando case, que com base no valor da variável do comando tch define qual opção será executada. Para que Somente um entre os vários comandos case seja executado devemos executar o comando break, logo após a execução dos comandos contidos no bloco do comando case selecionado. (MENDES, Douglas, 2009, p. 43)

3.4 Repetição Condicional

A estrutura de repetição while é usada para repetir os comandos dentro de suas chaves, ou seja, no seu bloco, enquanto a expressão definida em seu parâmetro for verdadeira.

FIGURA 4 - REPRESENTAÇÃO GRÁFICA DO COMANDO WHILE



FONTE: AUTORIA PRÓPRIA (2021)

FIGURA 5 - CÓDIGO DO COMANDO WHILE

FONTE: MENDES, Douglas (2009, p. 47)

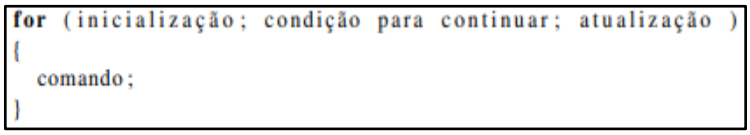
No comando while, também é possível usar o comando continue, que interrompe a sequência de código e testa novamente a condição, repetindo todo o procedimento, e o comando break, que interrompe a sequência de código, porém nesse caso saindo do bloco do while e seguindo o linha de comandos normalmente.

3.5 Repetição Contada

A estrutura de repetição *for* é usado para se executar várias vezes um conjunto de comandos. Este é considerado uma estrutura compacta, devido aos seus elementos estarem reunidos em um só corpo. “Com o for, podemos implementar exatamente o mesmo que com o while mas, no caso do for, todas as operações

relacionadas com o laço ficam na mesma linha, o que facilita a visualização e o entendimento de quem olha para o código" (GOLDMAN, Alfredo et al., p. 90).

FIGURA 6 - ESTRUTURA DO COMANDO FOR



FONTE: GOLDMAN, Alfredo et al. (2004, p. 90).

3.6 Vetores Arrays

. "A linguagem Java possui o conceito de array, que é uma estrutura de dados que permite guardar uma sequência de valores(números, caracteres ou objetos quaisquer) de uma forma única e organizada" (GOLDMAN, Alfredo et al., 2004, p.83). Imagine que você queira fazer um programa que deva registrar a umidade de um determinado mês, usando apenas variáveis deveria ser criado 30 variáveis, uma para cada dia do mês, mas usando um array, apenas uma linha de código e um elemento precisa ser criado.

A estrutura de uma array é simples, consiste do seu tipo, seguido de "[]" e o nome do array, usando o exemplo anterior, seria: Double [] umidade; após isso devemos usar um new com o tipo da array, para informar que um novo vetor deste tipo está sendo instanciado, seguido de outro "[]" com a quantidade de valores que serão salvos no vetor, novamente baseando-se no exemplo: Double [] umidade = new Double[30], e assim como uma variável, não se pode salvar um tipo diferente de valor que não corresponda ao dela.

Para acessar um índice de um vetor (ou seja, os valores salvos no vetor) é só colocar o nome do vetor, seguido do número do índice que quer acessar (lembrando que o primeiro índice de um vetor é 0).

3.7 Orientação a Objetos

A Orientação a Objetos (OO) é um paradigma da programação que tem por objetivo resolver problemáticas com base na vida cotidiana. Seu conceito também se

baseia na interação entre objetos. A programação orientada a objetos possui quatro pilares, sendo estes: Encapsulamento, Herança, abstração e polimorfismo.

3.7.1 Classes

“Antes mesmo de ser possível manipular objetos, é preciso definir uma classe, pois esta é a unidade inicial e mínima de código na 00. É a partir de classes que futuramente será possível criar objetos” (CARVALHO, Thiago, 2016).

As classes, em suma, são uma descrição de um conjunto de objetos que comparticipam algumas características, como atributos, métodos, etc.

3.7.2 Atributos

> Para um atributo ser definido em uma classe, devemos seguir o mesmo princípio de definição de variáveis - sejam globais ou locais - de uma linguagem estruturada: declarar seu tipo e, depois, seu nome. O tipo pode ser um dos listados a pouco. O nome deve seguir a mesma preocupação da classe: ser o mais representativo possível (CARVALHO, Thiago, 2016).

Um atributo é considerado uma variável pertencente ao objeto, que tem como função armazenar as informações obtidas de um objeto. Para a coleta desses dados, é necessário saber o tipo do atributo, que varia de acordo com o que vai ser definido. Em Java podemos citar alguns tipos, como:

QUADRO 3 - TIPOS DE ATRIBUTOS

TIPO	FAMÍLIA	EXEMPLO
int	inteiro	2 147 241
float	real	3.4e^+38
String	literal	“java”
long	inteiro	2^63
boolean	lógico	true
double	real	1.8e^+308

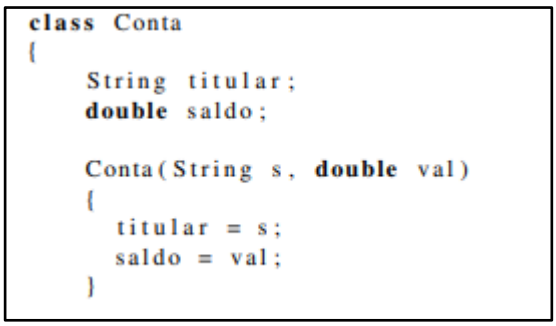
FONTE: AUTORIA PRÓPRIA (2021)

3.7.3 Métodos

Os métodos são processos que acionam ações de um objeto. As atividades realizadas pelos objetos estão diretamente ligadas com os métodos; essa relação também possibilita que haja interação entre objetos. “Além disso, independente da quantidade e da finalidade dos métodos de uma classe, existem um especial que toda classe possui: o construtor” (CARVALHO, Thiago, 2016).

O construtor tem a função de criar objetos a partir da classe em que está inserido, “é através do seu uso que será possível instanciar objetos e, a partir disto, manipular de forma efetiva seus atributos e métodos” (CARVALHO, Thiago, 2016). (FIGURA 7). O construtor também tem função de dar determinados valores iniciais.

FIGURA 7 - EXEMPLO DO USO DE CONSTRUTOR



```
class Conta
{
    String titular;
    double saldo;

    Conta(String s, double val)
    {
      titular = s;
      saldo = val;
    }
```

FONTE: GOLDMAN, Alfredo et al. (2004, p. 113)

3.7.4 Objeto

De forma resumida, um objeto é a instanciação de uma classe. Ele é um exemplo de entidade (existente na realidade) seja na forma física ou conceitual. É necessário a definição de uma classe antecedendo as instâncias dos objetos.

Para a criação de um objeto em Java, devemos definir uma classe para então instanciar um objeto a partir dela, além de uma variável, assim como também temos que transcrevemos o operador *new.*

Exemplo: Programador programador = new Programador();

No exemplo acima foi utilizado o construtor assentado na classe Programador para criar um objeto. O objeto do tipo *Programador* foi instanciado e guardado na variável *programador*.

Um objeto também possui um estado. Como já citado, ele possui atributos representados por variáveis, este tem como função armazenar dados que por sua vez variam de acordo com os momentos.

> O estado de um objeto é o conjunto dos valores dos seus atributos de um determinado momento. Estes valores podem mudar a qualquer momento e em qualquer quantidade, sob qualquer circunstância. Com isso, o objeto pode ter quantos estados necessitar enquanto ele estiver em execução (CARVALHO, Thiago, 2016).

3.8 Array de Objetos

Uma array de objetos é como uma array normal, mas com algumas diferenças, e para entendê-las, é preciso lembrar do conceito de variáveis. Como dito anteriormente, variáveis de tipos primitivos, como int, boolean ou char são simples e ocupam pouca memória, e por isso suas cópias são salvas nas posições de memória correspondentes, já para variáveis complexas, suas cópias recebem uma referência ao valor da variável original, e não uma cópia dos dados.

O mesmo ocorre com os arrays, quando copiamos os valores de uma variável para um array, ambas primitivas, o que vai acontecer é uma cópia e direcionamento dos dados, já quando ambas forem complexas, o que ocorrerá é que apenas uma referência ao valor da variável, que no caso de uma array de objetos, é um objeto.
“É comum ouvirmos "array de objetos". Porém, quando criamos uma array de alguma classe, ela tem referências. O objeto, como sempre, está na memória principal, e, na sua array, só ficam guardadas as **referências** (endereços).” (CAELUM, s.d)

3.9 Modificadores de acesso, construtores e atributos de classe

3.9.1 Modificadores de acesso

Os modificadores de acesso servem para definir a visibilidade de acesso a um atributo de uma classe, com isso pode-se definir se uma classe poderá utilizar outra, por meio da invocação de métodos, por exemplo.

“Os métodos de acesso e os modificadores garantem o encapsulamento dos atributos. Esse encapsulamento oferece uma proteção ao atributo para garantir que os valores atribuídos estejam dentro dos valores aceitáveis para o atributo” (MENDES, Douglas, 2009, p. 83).

Ao utilizarmos métodos de acesso permitimos que um atributo seja acessado por meio de um método, nos possibilitando retornar um atributo. Outras vantagens também são bem vistas quanto ao uso desse método.

A seguir temos a sequência dos tipos de modificadores em ordem de maior restrição ao mais liberal:

QUADRO 4- ORDEM DE RESTRIÇÕES

private	default	protected	public

FONTE: AUTORIA PRÓPRIA (2021)

- Private: torna as informações visíveis para a própria classe;
- Default: torna as informações visíveis para a própria classe e para classes dentro do mesmo pacote;
- Protected: torna as informações visíveis para a própria classe, para classes dentro do mesmo pacote e subclasses;
- Public: torna as informações visíveis para a própria classe e para classes dentro do mesmo pacote, além de subclasses e classes de outros pacotes.

3.9.2 Construtores

De acordo com Mendes (2009), no método construtor define-se o construtor com o mesmo nome da classe, assim como não se deixa definido o tipo de retorno, sendo esse um dos diferenciais e o diversifica dos outros métodos.

Ademais, um ponto importante para se destacar é que uma classe pode conter mais de um construtor.

> O construtor é unicamente invocado no momento da criação do objeto por meio do operador *new*. Uma classe pode conter um ou mais construtores, por isso o conceito de sobreposição (overload) se aplica aos construtores. É importante observar que quando não criamos nenhum instrutor explicitamente dentro de uma classe o compilador o fará automaticamente, ou seja, o compilador criará um construtor vazio (sem argumentos/parâmetros) (MENDES, Douglas, 2009, p. 105)

3.9.3 Atributos de classe

Os atributos de classe são domínios semelhantes aos objetos de uma certa classe. Como por exemplo, um indivíduo pode ter um nome, e além disso pode ter outros atributos, como cpf, cor, etc. Porém, quando fazemos uma instância de uma classe, cada atributo de um objeto guarda seus valores individualmente, mas um existe um atributo que é o mesmo para todas as instâncias da mesma classe “Estes atributos recebem o nome de atributos da classe ou atributos estáticos e, em C#, para criar um atributo estático basta colocar a palavra **static** na declaração do atributo”(CAELUM, s.d)

3.10 Herança e Polimorfismo

3.10.1 Herança

Segundo Carvalho (2016), herança nada mais é do que outra forma de se representar algo que já existe no mundo real, neste caso, explicitamente a relação de especialização e herança entre coisas e conceitos. Aqui ele exemplifica essa relação usando a classificação biológica, como a homo sapiens, que é uma espécie, herda características do homo, que é um gênero, e por sua vez herda características de outra divisão superior.

Já na linguagem de programação, Herança é algo importantíssimo para a Orientação a Objetos, sendo um de seus 4 pilares; consiste na possibilidade de uma classe herdar características de uma classe-mãe, como atributos e métodos, sem a necessidade de reescrita dos códigos, podendo ainda modificar métodos herdados ou criar novos que pertencem somente a classe-filha (MENDES, Douglas, 2009, p.186)

3.10.2 Polimorfismo

“O termo polimorfismo é originário do grego e significa ‘muitas formas’, com poli se significando ‘muitas’ e morphos significando ‘formas’” (MENDES, Douglas 2009). Por definição, na área de programação, polimorfismo é um princípio a partir do qual as classes derivadas de uma única classe base são capazes de invocar os métodos que, embora apresentem a mesma assinatura, comportam-se de maneira diferente para cada uma das classes derivadas.

O Polimorfismo é um mecanismo no qual podemos ter os mesmos atributos e métodos, porém com implementações diferentes, um exemplo seria uma roupa e um

jornal; ambos são da indústria têxtil, ambos possuem certa semelhança, mas ainda é necessária a diferenciação de algumas de suas características.

3.11 Classes Abstratas e Interfaces

3.11.1 Classes Abstratas

As classes abstratas servem apenas para idealizar um tipo, um ponto crucial é que ela não pode ser instanciada. Nesse tipo de classe há mais códigos genéricos, nela não existem métodos “funcionais”, eles precisam ser implementados por subclasses, e para ocorrer isso, as subclasses utilizam suas particularidades. Não poder criar objetos é um exemplo dos impedimentos de uma classe abstrata.

> Uma classe abstrata é desenvolvida para representar classes e conceitos abstratos. A classe abstrata é sempre uma superclasse que não permite que nenhum objeto seja criado a partir dela, ou seja, não se pode executar a operação de *new* usando uma classe abstrata. Uma classe abstrata é um meio termo entre uma classe concreta que contém uma implementação para todos os métodos declarados e uma interface na qual nenhum dos métodos declarados será implementado, e sim pelas classes que a implementam (MENDES, Douglas, 2009, p. 196)

3.11.2 Interfaces

> Um dos conceitos principais de orientação a objetos é o encapsulamento, através do qual, tanto os atributos quanto a implementação dos métodos de uma certa classe não são visíveis ao usuário da classe. Conhecendo-se apenas a *interface* de uma classe, isto é, os métodos disponíveis e suas respectivas assinaturas, podemos utilizar objetos desta classe sem conhecer detalhes de como ela é implementada internamente (GOLDMAN, Alfredo et al. p.117).

FIGURA 8 - EXEMPLO QUE APLICAÇÃO DE INTERFACE

```
interface Animal
{
    void nasça();
    void passeiePelaTela();
    void durma();
    double peso();
}
```

FONTE: GOLDMAN, Alfredo et al. (2004, p. 118)

Ao realizar o uso de interfaces, as operações devem ser definidas de forma abstrata, possuindo sua assinatura e com todos os métodos públicos. Pode-se definir

atributos, porém, estes devem permanecer como constantes públicas. Ademais, quando se usa interfaces, pode-se pedir para a um usuário partes escritas, mesmo que esta não conheça o programa em questão.

3.12 Exceções

Segundo Júnior (2019), um programa de computador sempre segue uma sequência lógica em seu funcionamento, transpassando de processos em processos enquanto mantém um fluxo estável. Todavia, se houver algum erro ou problema durante a execução do programa, esse fluxo poderia ser encerrado ou ter problemas de performance, para que isso não ocorra, direcionamos o fluxo do programa para uma outra sequência lógica de códigos que tenta resolver o erro que ocorreu. Isso é uma exceção. De uma forma simples, “Exceções em Java representam situações inesperadas ocorridas na de algum trecho do programa. Uma exceção deve ser vista como um retorno alternativo da execução de algum método”.

Para se usar exceções, é necessário abrir um bloco try catch, dentro do bloco do try se coloca os comandos que podem gerar uma exceção, e nos parâmetros do catch é colocado o tipo de exceção, com o devido tratamento no bloco dos catch. “Um bloco try pode estar relacionado a mais de um bloco catch, mas um bloco try sozinho apresenta erro de compilação. Não pode haver código adicional entre o bloco try e o começo do catch.” (MENDES, Douglas, 2009, p.288)

FIGURA 9 - USO DE UM TRY CATCH

```
try {
    if (nota >= 70) {
        // lançando a exceção
        throw new Exception("Aprovado");
    } else if ((nota >= 40) && (nota < 70)) {
        throw new Exception("Exame");
    } else {
        throw new Exception("Reprovado");
    }
    // capturando a exceção
} catch (Exception e) {
    System.out.println(e.getMessage());
}
```

FONTE: MENDES, Douglas (2009, p.289)

Na figura acima, vemos o uso de um try catch: os comandos que podem causar uma exceção dentro de um try, que no seu interior possui throws, comandos que

podem gerar uma exceção. Nos parâmetros do catch, temos o tipo de exceção Exception, que engloba todas as exceptions, e dentro do bloco do catch, temos o código de tratamento da exceção, nesse caso, escrever a mensagem criadas pelos throws.

3.13 Collections

O conceito de Collections vem de uma estrutura de dados que armazena objetos, sendo este também considerado um objeto. Os métodos que são realizados por coleções variam de acordo com a ocasião, exemplos de operações são adição, remoção, acesso e pesquisa. A seguir um exemplo de operação:

3.13.1 ArrayList

O ArrayList é como array de certa forma, porém, seu tamanho pode variar. Com essa implementação é mais fácil realizar o acesso a elementos, e esse é um ponto avaliado quanto a utilização do ArrayList, por causa de sua facilidade em relação ao acesso.

> A implementação mais utilizada da interface List é a ArrayList, que trabalha com uma array interna para gerar uma lista. Portanto, ela é mais rápida na pesquisa do que sua concorrente, a LinkedList, a qual é mais rápida na inserção e remoção de itens nas pontas (CAELUM, s.d).

4. METODOLOGIA

4.1 Instrumentos utilizados

Durante o período de oferta da monitoria, foram utilizadas ferramentas para dar apoio aos discentes. Essa prática foi necessária devido a pandemia, onde se tornou impossível realizar a monitoria de forma presencial. Dentre estes instrumentos estão o *WhatsApp*, *Google Meet* e o IDE *Eclipse*.

4.1.1 WhatsApp

WhatsApp é um aplicativo multi-plataformas de mensagens instantâneas, feito inicialmente como uma alternativa para o SMS. Nele, é possível criar salas de bate papo em grupo, mandar mensagens de texto, áudio e muitas outras mídias, além de permitir chamadas de voz e vídeos seguras.

Devido a sua facilidade de manuseamento e por muitos dos alunos possuírem tal aplicativo, optamos por essa escolha logo de início.

4.1.2 Google Meet

O *Meet* é uma aplicação de videoconferência desenvolvida pelo Google. Nele é possível criar reuniões e gerar um link para compartilhar com quem desejar. Além disso, um ponto positivo é que ele pode ser usado em variados dispositivos, seja computador, celular, tablet, etc. Seu funcionamento também é prático e de fácil compreensão, um dos motivos que nos levou a fazer o seu uso.

4.1.3 Eclipse

O *Eclipse* é um Ambiente de Desenvolvimento Integrado (IDE) gratuito que reúne ferramentas para a criação de softwares em várias linguagens de programação. Foi a plataforma de programação escolhida pelo ministrador da disciplina e, por isso, a escolhemos para as codificações necessárias para o sancionamento das dúvidas dos discentes.

5. CARACTERIZAÇÃO DAS ATIVIDADES DESENVOLVIDAS

5.1 Atendimento ao aluno

O horário definido para a realização da monitoria foi das 15:00h até as 17:30, com um pequeno intervalo de 30 minutos, para depois recomeçar às 18:00h, finalizando às 18:30.

O fluxo de alunos era instável, com dias com nenhum aluno, a dias com 5 alunos.

5.1.1 WhatsApp

Os números do *WhatsApp* dos monitores foram disponibilizados para os alunos da disciplina, que deveriam se comunicar com eles dentro do horário marcado caso fosse necessário.

Esse contato inicial era feito pelos chats do app, onde os alunos usavam de seus recursos, como mensagens de texto, áudios e imagens visando explicar de forma sucinta qual o problema ou dúvida que os mesmos tinham.

5.1.2 Google Meet

Quando as dúvidas eram de natureza prática, ou seja, erro ou problemas nos códigos das atividades e exercícios, o *WhatsApp* não se mostrou suficiente para o objetivo da monitoria; em casos como estes, o uso de outra plataforma se fez necessário.

Para tais momentos, foi usado o *Google Meet*, onde o aluno mostrava os códigos que havia feito apresentando a tela de forma instantânea, permitindo assim ao monitor uma melhor avaliação e uma comunicação mais rápida e dinâmica com o discente.

5.1.3 Eclipse

Em raras ocasiões, haviam erros de configuração ou compilação do Eclipse(erros de ambiente), pequenos erros nas linhas de código(erros de sintaxe) ou até mesmo o aluno não conseguia entender algum aspecto do assunto passado pelo professor. Em situações como essas, o *Eclipse* era usado para verificar se o código estava realmente errado, copiando os arquivos do discente e testando para ver se

ocorria o mesmo erro, e também para realização de códigos que serviriam como exemplo, para que o aluno pudesse entender melhor o que deveria ser feito.

5.2 Reunião

No período em que o assunto de vetor era lecionado, muitos alunos mostraram dificuldade para entender a teoria por trás dele e como usá-lo na realização das atividades, tendo isso em mente, o docente da disciplina sugeriu que os monitores se reunissem com os alunos para resolver essa questão.

Durante a reunião, que durou cerca de 1h hora, foi permitido que os alunos apresentassem seus códigos e falassem sobre suas dificuldades. Ao fim da reunião, foram apresentados dois códigos usando vetores para que os discentes pudessem entender melhor o seu funcionamento.

5.3 Resultados

5.3.1 Positivos

Como pontos positivos, foi possível reforçar a compreensão do conteúdo já visto durante a época da aprendizagem, além de aprender alguns assuntos não vistos e o ganho de experiência no lecionamento de alunos; também foi notado o aumento de interesse de alguns alunos sobre a área de programação em diálogos com os mesmos.

5.3.2 Negativos

Já quanto aos negativos, era notável que alguns alunos não entendiam o real sentido de uma monitoria, desejando e pedindo que fosse dito a eles como deveriam fazer os exercícios, em alguns casos, como se fosse uma obrigação da parte do monitor. Também era frequente a requisição da monitoria em horários diferentes dos determinados, como fins de semana, manhã ou tarde da noite.

6 ANÁLISE E DISCUSSÃO DOS RESULTADOS

No decorrer das atividades desenvolvidas na prática profissional, foram observados diversos aspectos.

6.1 Alunos

Ao falar especificamente sobre os estudantes, de maneira geral, foi observado um certo retorno negativo quanto ao ensino remoto. Alguns alunos se sentiram um tanto deslocados com esse novo formato, esse fato, querendo ou não, afetou o ensino-aprendizagem.

Outro ponto a ser destacado são as situações desconfortáveis que ocorreram algumas vezes, como os pedidos de ajuda fora do horário posto ou em dias não úteis, como também um certo aproveitamento do que estávamos ofertando.

Sobre este último ponto, alguns alunos, de certa forma, ao pedirem apoio sobre as atividades, costumavam perguntar *como* deveriam desenvolver os códigos: “Qual comando?; Em qual linha?; O que eu escrevo?; Como se faz isso aqui?; Me mostre como faz.” eram frases frequentes na semana. Tal coisa foge do sentido original de uma monitoria, que é orientar, e não fazer as atividades por eles.

Mesmo com tais momentos ruins, foi perceptível que alguns alunos passaram a gostar mais da disciplina e da área de programação, indo atrás de assuntos não ensinados pelo professor e aprimorando seus projetos que foram feitos no final da disciplina.

6.2 Impacto nos monitores

A monitoria impactou positivamente na firmação e na expansão do conhecimento de ambos os monitores acerca dos conteúdos mostrados na matéria. Também possibilitou uma experiência de vivência na área da programação, ajudando de maneira proveitosa na decisão quanto a seguir no campo da informática.

7 CONSIDERAÇÕES FINAIS

O objetivo central desta monitoria é, como já se espera, o auxílio aos alunos sobre os assuntos abordados na disciplina de Programação Estruturada e Orientada a Objetos. Tal ajuda se faz necessária devido à certa complexidade da matéria, que foi ampliada devido ao estilo de ensino aplicado durante a pandemia, junto com o fato de ser um tipo de conhecimento novo, normalmente não trabalhado nos níveis fundamentais do ensino.

O auxílio aos estudantes no decorrer da oferta da disciplina foi a atividade desenvolvida durante a monitoria. Essa atividade foi de suma importância para o enriquecimento da formação acadêmica dos monitores, devido a aproximação do exercício da docência, além da troca de conhecimentos entre professor-monitor e monitor-aluno.

No decorrer da monitoria, a procura pelo monitor era frequente, e a evolução dos alunos era lenta, porém constante; praticamente todos os alunos que pediam atendimento nos primeiros dias continuaram até o fim da disciplina.

Portanto, o resultado foi alcançado, pois com o contato contínuo com os alunos, foi possível reduzir consideravelmente a ocorrência de erros de codificação, além do bom aprendizado dos assuntos lecionados e do interesse pela área desenvolvido por alguns.

REFERÊNCIAS

ARAUJO, C. **Java Collections: Como utilizar Collections**. Disponível em: <https://www.devmedia.com.br/java-collections-como-utilizar-collections/18450>. Acesso em: 29 dez. 2021.

Caelum Escola de Tecnologia Cursos Online. Disponível em: <https://www.caelum.com.br/apostila-java-orientacao-objetos/um-pouco-de-arrays#o-problema>. Acesso em: 28 dez. 2021.

CARVALHO, T. L. E. **Orientação a Objetos: Aprenda seus conceitos e suas aplicabilidades de forma efetiva**. [s.l.] Editora Casa do Código, 2016.

FARIA, J.; SCHNEIDER, M. S. P. S. Monitoria: uma abordagem ética. (mimeo)

GOLDMAN, A. *et al.* **Introdução à ciência da computação com Java e orientação a objetos**. [s.l: s.n.].

JÚNIOR, W. **Tratamento de Exceções Concorrentes na Linguagem Java**. [s.l.] Universidade Federal de Pernambuco, 10 set. 2013.

MENDES, D. R. **Programação Java com Ênfase em Orientação a Objetos**. [s.l.] Novatec Editora, 2009.

PACIEVITCH , Y. **História do Java**. Disponível em: <https://www.infoescola.com/informatica/historia-do-java/>. Acesso em: 28 dez. 2021.

SCHNEIDER, M. **Monitoria: uma abordagem ética (Monitorship: an ethical approachMárcia Schneider - Academia.edu**, 20 maio 2019. Disponível em: <https://www.academia.edu/39197307/Monitoria_uma_abordagem_%C3%A9tica_Monitorship_an_ethical_approach>. Acesso em: 29 dez. 2021

Google Meet: o que é, como funciona, como baixar e quanto custa. Disponível

em: <https://www.linknacional.com.br/blog/google-meet-o-que-e-como-funciona-como-baixar-quanto-custa/>. Acesso em: 29 dez. 2021.

ANEXO A – Formulário de identificação

<table>
<tr><th colspan="2">Dados do Relatório Científico</th></tr>
<tr><td colspan="2">Título e subtítulo:

Relatório da prática profissional: Monitoria de Programação Estruturada e Orientada a Objetos</td></tr>
<tr><td>Tipo de relatório:

Monitoria</td><td>Data: 31/12/2021</td></tr>
<tr><td colspan="2">Título do projeto/ programa/ plano:

Monitoria Voluntária da disciplina de Programação Estruturada e Orientada a Objetos</td></tr>
<tr><td colspan="2">Autor(es): João Victor da Cunha da Costa</td></tr>
<tr><td colspan="2">Instituição e endereço completo:

Instituto Federal do Rio Grande do Norte Campus Avançado Lajes

BR-304, Km 120, s/n - Centro, Lajes - RN, 59535-000</td></tr>
<tr><td colspan="2">Resumo:

O presente relatório de Monitoria da disciplina de Programação Estruturada e Orientada a Objetos, ofertada pelo curso técnico integrado de informática do IFRN campus avançado Lajes, apresenta informações acerca do período de auxílio prestado pelos monitores da referida disciplina. O objetivo geral da prática</td></tr>
</table>

profissional consistiu em apoiar os estudantes na realização das atividades exigidas pelo professor encarregado da matéria, além de cooperar na construção de conhecimento teórico e prático dos discentes. Durante o desenvolvimento deste trabalho foi possível expor a definição e as funções dos instrumentos utilizados, da mesma forma os conteúdos abordados nos dois módulos em que se foi realizado às aulas. Para se ofertar a monitoria durante o ensino remoto, foi necessário a utilização de ferramentas de comunicação, dessa forma foi possível exercer a prática proposta. À vista disso, pode-se formular a experiência do monitor com o exercício docente, avaliando suas perspectivas e visões. Por último, é válido salientar a importância dessa prática para a geração de profissionais da área de programação, tanto desenvolvedores quanto lecionadores.

Palavras-chave/descritores:

Monitoria. Programação. Prática profissional.

Nº de páginas: 32

Jornada de trabalho: 3 horas	Horas semanais: 15 horas

Total de horas: 170 horas

Observações/notas