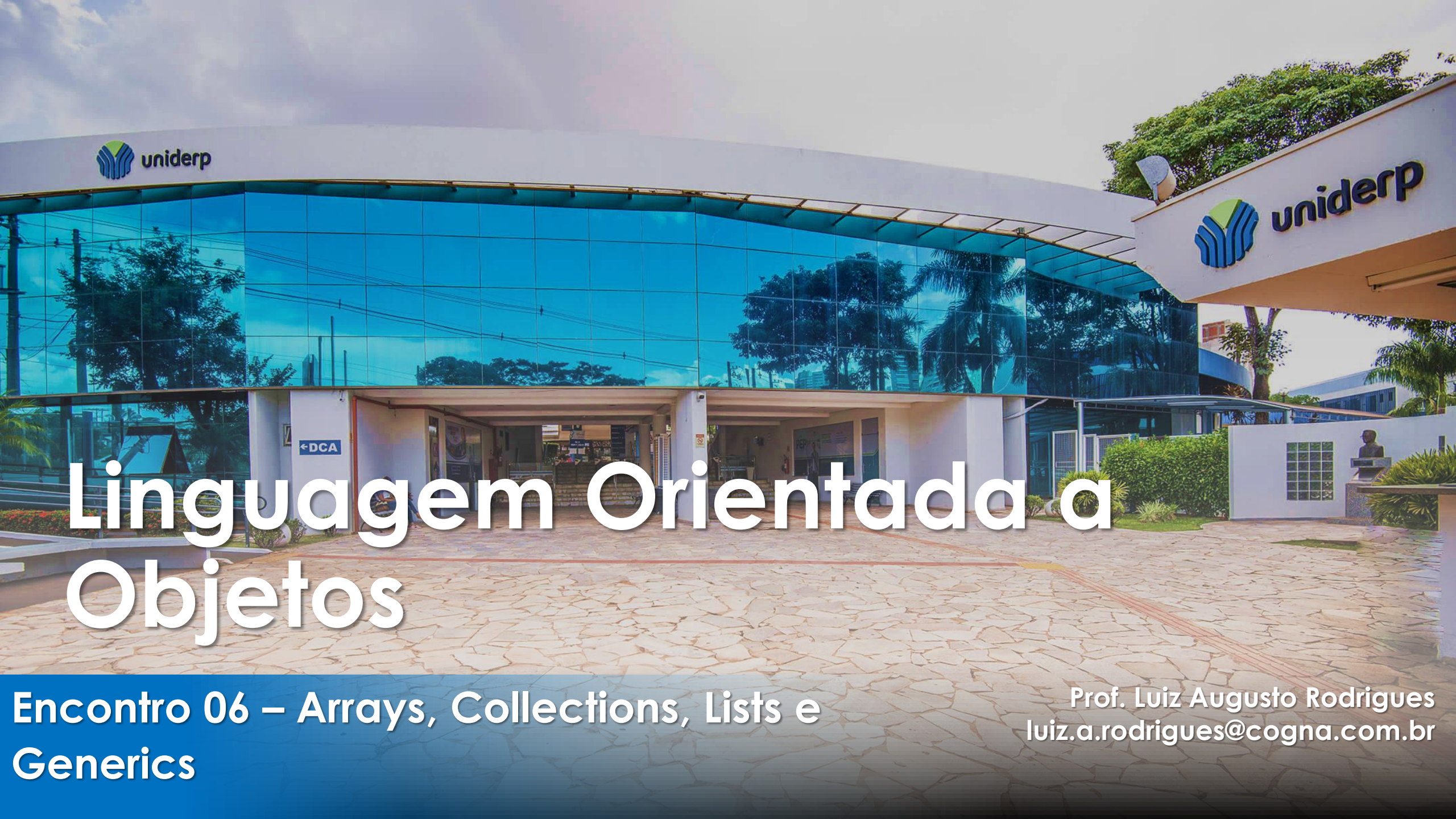


Linguagem Orientada a Objetos

Encontro 06 – Arrays, Collections, Lists e Generics

Prof. Luiz Augusto Rodrigues
luiz.a.rodrigues@cogna.com.br

Avisos Importantes

Nenhum material será enviado via e-mail. Os materiais serão disponibilizados no **AVA** e no google drive:
https://bit.ly/47qzfvl

Dúvidas, questionamentos, entre outros deverão ser realizados pelo **e-mail** e pelo **Whatsapp** da disciplina**.**

Para ingressar no grupo do **Whatsapp** da disciplina acesse o link a seguir e selecione sua disciplina.
linklist.bio/profluizao_2023-2

A Disciplina de Programação Orientada a Objetos, a partir deste slide, será referenciada pela sigla **LOO**.

Assuntos Abordados

- Revisão – Pilares
- Arrays
- Collections
- Lists
- Generics

LOO

Encontro 06

Instruções Importantíssimas!!!

Instruções Importantíssimas!!!

Para este semestre, na disciplina de POO, utilizaremos OBRIGATORIAMENTE as seguintes ferramentas:

- Sistema Operacional Windows 10, Linux ou MacOS.
- Linguagem de Programação Java.
- JDK 17 ou superior
  - https://download.oracle.com/java/17/archive/jdk-17.0.6_windows-x64_bin.exe
- VS Code.

Instruções Importantíssimas!!!

- Git 2.39.2 ou superior
  - https://github.com/git-for-windows/git/releases/download/v2.39.2.windows.1/Git-2.39.2-64-bit.exe
- Github
  - https://github.com/
  - https://desktop.github.com/
- Tutorial em Git e Github
  - Tutorial Git e Github 2022 – Introdução prática para iniciantes
  - Canal DevSuperior
  - https://youtu.be/_hZf1teRFNg

Quem for utilizar Linux ou MacOSx, fale comigo após a aula, ou pelo grupo do WhatsApp, para instruções de instalação.

Instruções Importantíssimas!!!

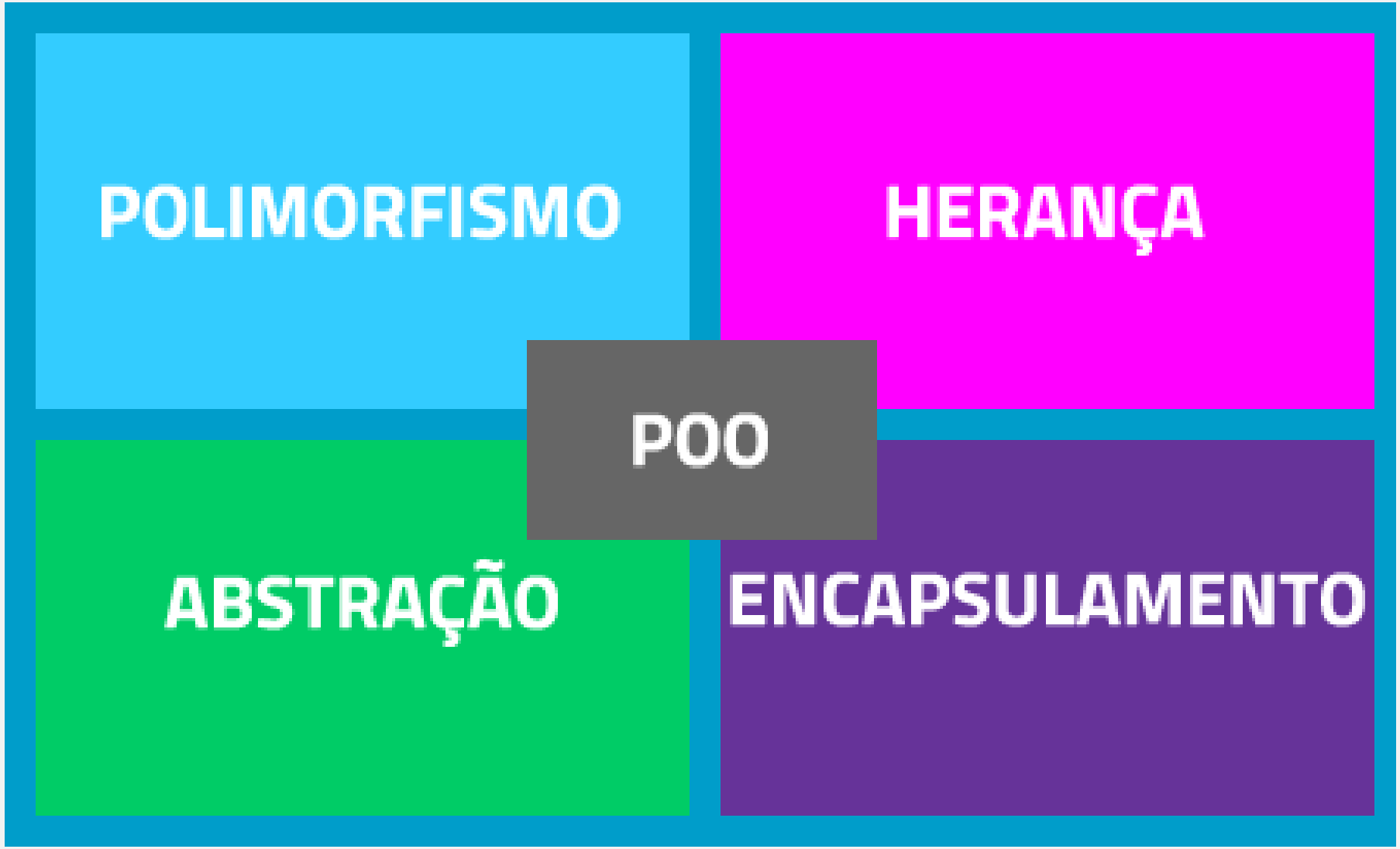
- Os relatórios não serão utilizados nessa disciplina, por se tratar de uma disciplina 90% prática.
- No lugar, teremos atividades de pós-aula, que deverão ser respondidas e justificadas, para fixação do conteúdo discutido na aula.
- As atividades de aula estarão disponíveis apenas no Github da turma.
- O tutorial do Git e Github é **obrigatório**.

LOO

Encontro 06

Revisão – Pilares

Princípios de POO



LOO

Encontro 06

Arrays

Arrays

Quando precisamos armazenar mais de um valor em uma variável, fazemos isso utilizando um array. Essa palavra pode ser traduzida de forma livre como um "conjunto de variedades" — como qualquer conjunto, implica na existência de alguns elementos agrupados, portanto.

Nos arrays, cada elemento desse grupo tem a sua posição. Dessa forma, para armazenar esse conjunto de valores em uma única variável, sem precisar declarar cada uma para um valor, usamos um array.

Sintaxe de arrays em Java

Em Java, quando utilizamos um array, escrevemos sua representação com o uso de chaves — "[elemento1, elemento2 ...]". Podemos definir uma array sem número especificado, apenas indicando que aquela variável é um array.

A sintaxe é a seguinte:

```
tipo do objeto, como String, Integer, etc. [] nome de nosso array
```

Para facilitar o entendimento, veja um exemplo aplicado:

```
String[] nomes;
```

Sintaxe de arrays em Java

No exemplo, apenas definimos um array — sem determinar elementos ou número de posições para serem ocupadas. Poderemos preencher esses espaços no array apenas os designando durante o desenvolvimento do código.

Assim:

```
String[] nomes = {"Darci", "Cris", "Alex", "Juraci"}
```

Sintaxe de arrays em Java

Quando precisarmos acessar alguns desses elementos pertencentes às mesmas variáveis, buscamos diretamente a variável do array "nome". Vamos a um exemplo prático, bem simples, para ajudar no entendimento:

```
public class Main {
	public static void main(String[] args) {
		String[] carros = {"Fusca", "Brasilia", "Gurgel"};
		System.out.println(carrs[0]);
	}
}
```

Sintaxe de arrays em Java

O primeiro elemento de um array começa sempre a ser contado em 0.

Nesse caso, quando pedimos a posição zero (“[0]”) do array “carros”, a saída no console desse código será:

Fusca

Tipos de Arrays em Java

Durante o desenvolvimento de aplicações, muitas vezes, não temos a possibilidade de alocar todos os elementos de que precisamos para o funcionamento do sistema durante a programação.

Nesses casos, podemos fazer um array vazio, mas com “espaços” para alocar cada elemento.

Tipos de Arrays em Java

Sendo um objeto, um array em Java, não pode apenas ser declarado - para definirmos o tamanho de nosso array sem a necessidade de criar os elementos, precisamos definir o número de posições, dessa forma:

```
String[] nomes = [4]
```

Já para criar um novo array, por sua vez, usamos a expressão “new”, um comum operador do Java:

```
int[] novo_array = new int[100];
```

Tipos de Arrays em Java

```
String[] nomes = [4]
```

Aqui, temos um array simples, o qual tem 100 posições possíveis de serem ocupadas. Isso é chamado “Index” - o índice do array existe, basicamente, para que seja possível enumerar e classificar cada um dos valores contidos nele.

Mas antes de criarmos nosso array, para que possamos manipulá-lo, precisamos saber alguns detalhes importantes. Por exemplo, um array pode conter apenas uma linha de elemento, enquanto outros serão preenchidos da mesma forma que matrizes. Essa é a diferença entre arrays unidimensionais e multidimensionais.

Tipos de Arrays em Java

Arrays unidimensionais

Arrays de uma única dimensão são vetores que têm um único índice como identificador para cada elemento contido em um array. No caso do exemplo de que falamos anteriormente, a forma que usamos para declarar um array sem determinar seus elementos, mas apenas as posições que serão ocupadas, será da seguinte maneira:

```
public class MyClass {
	public static void main(String args[]) {
		int[] array_de_uma_dimensão = new int[5];
	}
}
```

Tipos de Arrays em Java

Arrays unidimensionais

```
public class MyClass {
	public static void main(String args[]) {
		int[] array_de_uma_dimensão = new int[5];
	}
}
```

Repare que, nesse exemplo, criamos um array com o nome de "array_de_uma_dimensão", e percebemos que ele terá 5 posições possíveis de serem ocupadas.

Tipos de Arrays em Java

Arrays multidimensionais

Saiba que também podemos operar com arrays de mais de uma dimensão. É possível compará-los a tabelas. Isso porque você pode enumerar múltiplos índices em um mesmo array, formando, assim, diversas linhas e colunas, com cada espaço sendo pertencente a um elemento. Para declararmos um array multidimensional, fazemos dessa forma:

```
public class MyClass {
	public static void main(String args[]) {
		int[][] array_multimensão = new int[5][5];
	}
}
```

Tipos de Arrays em Java

Arrays multidimensionais

```
public class MyClass {
	public static void main(String args[]) {
		int[][] array_multimensão = new int[5][5];
	}
}
```

Aqui, temos a aplicação de um novo array, mas com duas dimensões - ou dois índices. Eles acabam formando uma tabela 5 X 5, com 25 posições possíveis, nesse caso.

Características dos arrays em Java

Para um bom funcionamento e maior dinâmica durante as operações realizadas por meio de nossos sistemas em Java, temos quase uma inumerável série de características que permeiam os recursos para esse devido aproveitamento.

Para facilitar o seu entendimento dessas questões, trazemos, em nosso material, algumas das principais delas, as quais mais interferem ao desenvolvermos serviços com essas estruturas de dados.

Características dos arrays em Java

Elementos do array e tipos de dados

É cada unidade do tipo de dado inserido, podendo ser números (integer), caracteres (string), decimais (double) ou verdadeiro/falso (boolean). Cada elemento ocupa uma posição determinada no índice e pode ser acessado a partir da manipulação do array.

Características dos arrays em Java

Índices do array

O índice do array pode ser explicado como uma lista, ordenada de alguma maneira ou não, dos elementos contidos em um dado array.

Os arrays podem ter apenas um ou vários índices.

Características dos arrays em Java

Dimensão do array

A dimensão de um array é relacionada à sua quantidade de índices contidos.

Se um array tem um único índice, ele é chamado unidimensional, e caso tenha mais de um, multidimensional.

Maiores detalhes

- https://blog.betrybe.com/java/java-array/

LOO

Encontro 06

Collections

Collections

Já vimos em Java que podemos armazenar um conjunto de valores escalares ou objetos utilizando arranjos, ou seja, os vetores. Estas estruturas podem ser criadas dinamicamente com o auxílio do operador new responsável por buscar mais recursos de memória junto ao sistema.

Vimos também que poderíamos criar estruturas de dados mais complexas com disciplinas de acesso através da implementação de tipos abstratos de dados como listas, pilhas e filas.

Collections

Na versão atual de Java, uma coleção é um objeto que pode agrupar outros objetos. Uma coleção tem por objetivo facilitar a inserção, busca e recuperação destes objetos agrupados sistematicamente.

Para ser mais completo, o que temos por baixo de uma coleção de objetos é uma collection framework, ou seja, um arcabouço lógico para este tipo abstrato de dados, que se compõe de:

- Interfaces que representam e manipulam a coleção independentemente dos detalhes de implementação;
- Implementações que são as materializações, a codificação das interfaces; e
- Algoritmos que são os métodos usados para realizar as tarefas propostas pelas interfaces, como ordenação, inserção e recuperação dos objetos armazenados.

Collections

Os benefícios da utilização de coleções são os já aclamados: redução do esforço de programação, redução dos custos de implementação, aumento da eficiência da geração de códigos, interoperabilidade de codificação, redução do esforço de aprendizagem, entre outros.

Tipos de Coleções

As operações que podem ser feitas em coleções variam mas normalmente incluem:

- Adição de elementos;
- Remoção de elementos;
- Acesso aos elementos;
- Pesquisa de elementos;

Tipos de Coleções

Dependendo da forma de fazer as 4 operações básicas (adição, remoção, acesso e pesquisa), teremos vários tipos de coleções. Os três grandes tipos de coleções são:

- Lista (também chamado de “sequência“);
- Conjunto;
- Mapa(também chamado de “dicionário“).

Tipos de Coleções

As vantagens na utilização de coleções vem justamente da principal desvantagem dos Arrays: manipular arrays é bastante trabalhoso. Essa dificuldade aparece em diversos momentos:

- Não podemos redimensionar um array em Java;
- É impossível buscar diretamente por um determinado elemento cujo índice não se sabe;
- Não conseguimos saber quantas posições do array já foram "populadas".

Tipos de Coleções

O núcleo da interface de coleção diferencia estes tipos distintos de agrupamentos de objetos, cada um com sua interface específica:

- Set Um conjunto que não pode ter elementos duplicados;
- List coleção de objetos ordenados sequencialmente sob controle do usuário da lista;
- Queue para a fila existem disciplinas específicas de inserção, acesso e remoção, e não só estas;
- Map é uma facilidade de acesso a objetos através de chaves, cada chave corresponde a um objeto;
- Collection que é a raiz de todas as interfaces acima, a que representa uma coleção de objetos.

Tipos de Coleções

O denominador comum de todas as implementações e variações de tipos de coleções de objetos.

Desta interface derivam-se as interfaces mais utilizadas em Java, que são, List e Set.

Todas as classes e interfaces que formam este arcabouço lógico, esta estrutura de coleções, está no pacote java.util.

Maiores detalhes

- https://dcm.ffclrp.usp.br/~evandro/ibm1030/colecao/colec.html
- http://www.dsc.ufcg.edu.br/~jacques/cursos/p2/html/ed/colecoes.htm
- https://www.alura.com.br/conteudo/java-collections

LOO

Encontro 06

Lists

Lists

Uma lista é uma coleção de elementos arrumados numa ordem linear, isto é, onde cada elemento tem um antecessor (exceto o primeiro) e um sucessor (exceto o último).

Esta interface é uma Collection ordenada que pode conter elementos duplicados. Quase como os arranjos, as listas têm o elemento inicial no índice zero e, além dos métodos herdados da interface Collection. Esta interface fornece métodos específicos para manipular elementos usando os índices.

Ao usar Collection devemos prestar atenção para não instanciarmos interfaces e sim classes concretas.

Lists

A interface List é implementada, entre outras, pelas classes concretas ArrayList, LinkedList e Vector. Objetos da classe ArrayList são geralmente mais rápidos que objetos instanciados de Vector pois não se fartam das vantagens e seguros da implementação sobre threads.

ArrayList oferece acesso constante a todas as posições do objeto e para usá-la o usuário não precisa alocar cada nó que precisa para cada elemento da List, como seria esperado para uma interface deste tipo.

Se o programador sentir que o seu programa acessa constantemente o primeiro e o último elemento da lista ele poderá usar a classe LinkedList que também é uma classe concreta que implementa a interface List.

Características das Listas

- Normalmente implementada como "Array" ou "Lista Encadeada";
- A Lista pode ser mantida ordenada ou não;

As operações mais importantes de uma coleção do tipo Lista são:

- Adição de elementos
  - Adicionar um objeto em qualquer lugar da lista, fornecendo o índice desejado
- Remoção de elementos
  - Remover um objeto presente em qualquer lugar da lista, fornecendo o índice desejado

Características das Listas

- Acesso aos elementos
  - Obter o elemento de qualquer posição da lista, fornecendo o índice desejado
  - Iterar sobre os elementos
- Pesquisa de elementos
  - Descobrir se um certo elemento está na lista
  - Descobrir o índice de um certo elemento na lista (onde está)
- Indagar sobre atributos
  - Obter o número de elementos da coleção

Exemplo

```
import java.util.*;
public class TestArrayList {
    public static void main(String args[]) {
        ArrayList<String> lista;
        lista = new ArrayList<String>();
        System.out.println("tamanho: "
            + lista.size());
        lista.add("Primeiro");
        lista.add("Segundo");
        lista.add("Terceiro");
        System.out.println("tamanho: "
            + lista.size());
        String ret = lista.get(2);
        System.out.println("elem[3]: " + ret);
        lista.add(2, "novo");
        System.out.println(lista);
    }
}
```

Veja o resultado da execução:

```
tamanho: 0
tamanho: 3
elem[3]: Terceiro
[Primeiro, Segundo, novo, Terceiro]
```

Maiores detalhes

- https://www.alura.com.br/conteudo/java-collections

LOO

Encontro 06

Generics

Generics

Seria bom se pudéssemos escrever um único método de classificação que pudesse classificar os elementos em um array Integer, um array String ou um array de qualquer tipo que suporte ordenação.

Pensando nisso, a programação genérica, ou simplesmente ***Generics***, foi adicionado à linguagem de programação Java em 2004 na versão J2SE 5.0. Eles foram projetados para estender o sistema de tipos Java para permitir que "um tipo ou método opere em objetos de vários tipos, ao mesmo tempo que fornece segurança de tipos em tempo de compilação".

Generics

Os métodos genéricos e as classes genéricas em Java permitem que os programadores especifiquem, com uma única declaração de método, um conjunto de métodos relacionados, ou com uma única declaração de classe, um conjunto de tipos relacionados, respectivamente.

Os genéricos também fornecem segurança de tipos em tempo de compilação que permite aos programadores capturar tipos inválidos em tempo de compilação.

Generics

Usando o conceito de Generics em Java, podemos escrever um método genérico para classificar um array de objetos e, em seguida, invocar o método genérico com arrays inteiros, arrays duplos, arrays String e assim por diante, para classificar os elementos do array.

Generics

Assim como C++, usamos o operador diamante <> para especificar tipos de parâmetros na criação de classes genéricas. Para criar objetos de uma classe genérica, usamos a seguinte sintaxe.

```
BaseType<Type> obj = new BaseType<Type>()
```

Onde Type (ou simplesmente T) pode ser virtualmente qualquer tipo de objeto.

Observação: no tipo do parâmetro não podemos usar primitivas como int, char ou double. Apenas tipos encapsulados, como Integer, String, Double. Esta limitação se refere apenas ao Java.

Exemplo

```
//Utilizando código comum.

List list = new ArrayList();


list.add("abc");

list.add(new Integer(5)); //OK


for(Object obj : list){

//Conversão de tipos gera exceção do
//tipo ClassCastException em execução.

    String str=(String) obj;

}
```

```
//Utilizando generics.

List<String> list = new
ArrayList<String>();

list.add("abc");

//list1.add(new Integer(5));

//Erro de compilação.

for(String str : list1){

//nenhuma conversão é necessária,
//assim se evita exceções do tipo
//ClassCastException.

}
```

Exemplo

Podemos também utilizar tipos customizados para criação de listas de objetos próprios, desenvolvidos pelo programador.

Observe o exemplo a seguir.

```
public class Aluno{
      ...
}
```

```
//No código principal, teríamos:
List alunos = new ArrayList<Aluno>();
for(Aluno al : alunos){
//Acessando as propriedades da
//instância de Aluno.
}
```

Classes Genéricas

Utilizando os conceitos e metodologia de Generics, podemos criar nossas próprias classes genéricas, permitindo extender muito mais o reaproveitamento de código.

Observe o exemplo a seguir.

Classes Genéricas

```
public class Box<T> {
    private T t;
    public void add(T t) {
        this.t = t;
    }
    public T get() {
        return t;
    }
}
```

Classes Genéricas

```
public class App {
    public static void main(String[] args) {
        Box<Integer> integerBox = new Box<Integer>();
        Box<String> stringBox = new Box<String>();
        integerBox.add(10);
        stringBox.add("Hello World");
        System.out.printf("Integer Value :%d\n\n", integerBox.get());
        System.out.printf("String Value :%s\n", stringBox.get());
    }
}
```

LOO

Encontro 06

Prática

Prática

Utilizando nosso Projeto Escola, vamos:

- Criar o pacote FakeDB, que vai servir como um banco de dados falso;
- Criar listas de Pessoas que atuam em nossa empresa:

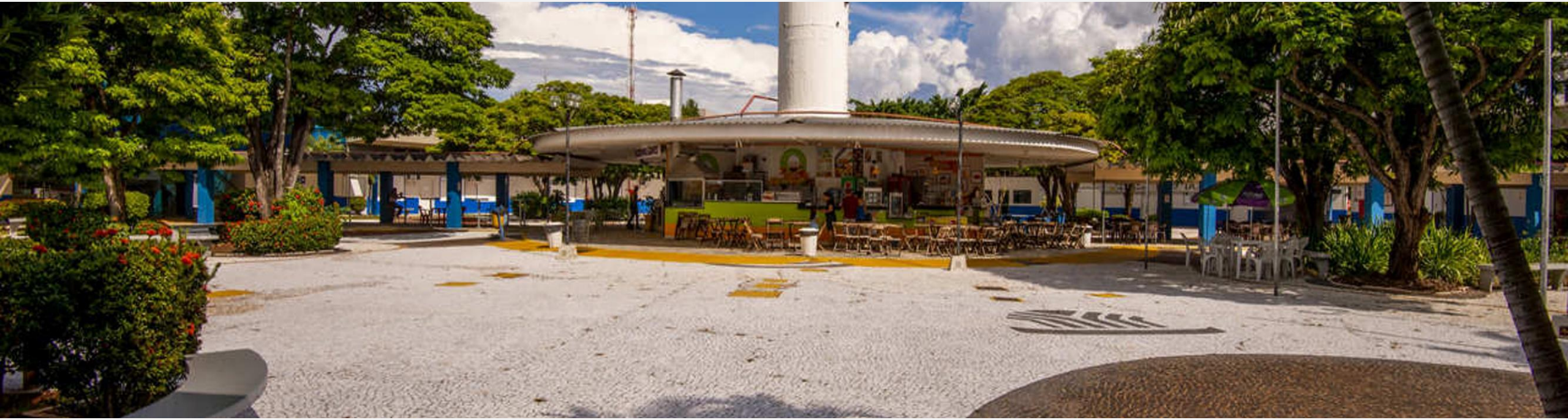
List<Aluno> alunos = ArrayList();

List<Professor> professores = ArrayList();

List<Tecnico> tecnicos = ArrayList();

List<Fornecedor> fornecedores = ArrayList();

Encerramento



Dúvidas e sugestões, entre em contato pelo whatsapp da disciplina, ou mande um e-mail para luiz.a.rodrigues@cogna.com.br