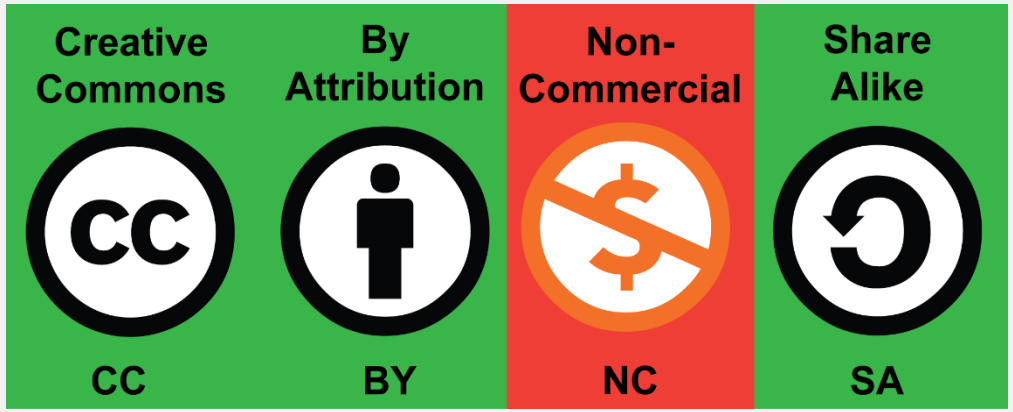
Estruturas de Dados Dinâmicas e Coleções

Prof. MSc. Jackson Antonio do Prado Lima
jacksonpradolima at gmail.com

Departamento de Sistemas de Informação – DSI

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Licença



UDESC

Histórico de Modificação

- Esta apresentação possui contribuição dos seguintes professores:
  - Alex Luiz de Souza (UDESC)
  - Anderson Fabiano Dums (UDESC)
  - Jackson Antonio do Prado Lima (UDESC)

UDESC

Agenda

- Estruturas de Dados Dinâmicas
- Coleções

ESTRUTURAS DE DADOS DINÂMICAS

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Agenda – Estrutura de Dados Dinâmicas

- Características
- Listas Encadeadas
- Estrutura da Lista Encadeada
- Listas Simplesmente Encadeadas
- Listas Duplamente Encadeadas
- Variações de Listas Encadeadas
- Vantagens e Desvantagens das Listas Encadeadas
- Classe *LinkedList*
- Métodos da Classe *LinkedList*
- Exemplos de Outras Estruturas Encadeadas

Estruturas de Dados Dinâmicas

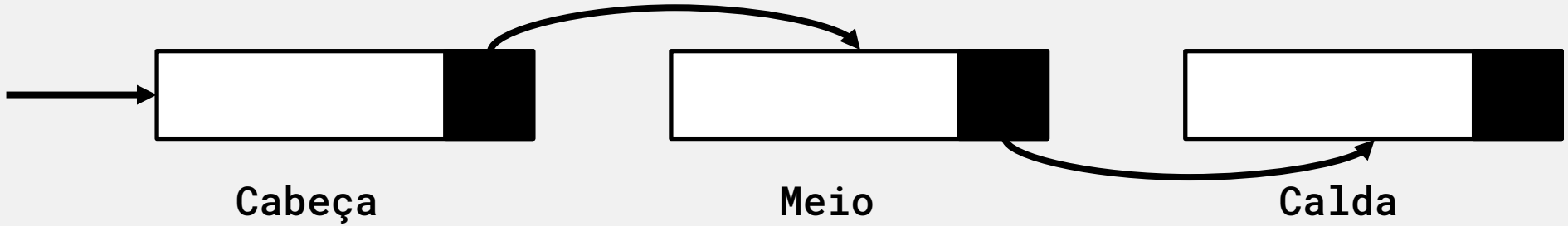
- **Uma** Estrutura de Dados **é** uma **construção utilizada** para organizar os dados **de** uma **determinada** forma
- Um array **é** um exemplo de **uma estrutura de dados** estática, as quais tem seu tamanho definido na sua declaração.
- Já as **estruturas de dados** dinâmicas **podem** aumentar **ou** diminuir **durante** a **execução** de um programa
  - Uma **estrutura de dados** encadeada **é** um exemplo de **estrutura** dinâmica

Listas Encadeadas

- **Uma** estrutura de dados encadeada **é uma** coleção de objetos, **designados por** nós (*nodes*), onde:
  - cada nó **contém** dados **e** uma referência para um **outro** nó
  - Por **exemplo**: listas encadeadas
- Portanto, **uma** lista encadeada (ou **ligada**) **é** uma coleção de objetos, **designados** por nós, onde cada nó **contém** dados **e** uma referência **para** outro nó, **de modo** a **formarem** uma lista

Estrutura da Lista Encadeada

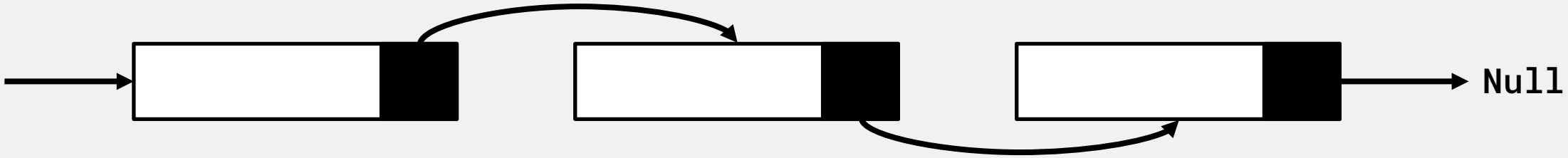
- Uma lista encadeada é **composta por**:
  - **O** primeiro nó, que é **chamado** de *head node* (**nó cabeça**)
  - Um ou mais nós intermediários, **identificados** por meio
  - **O** último nó, **chamado** de *tail node* (**nó calda**)



Listas Simplesmente Encadeadas

- **Uma** lista encadeada **diz-se** *simplesmente encadeada* (ou **ligada**) quando cada nó **contém** uma referência **para** o próximo nó

```
private class Node
{
        private Object dado;
        private Node proximo;
        ...
}
```

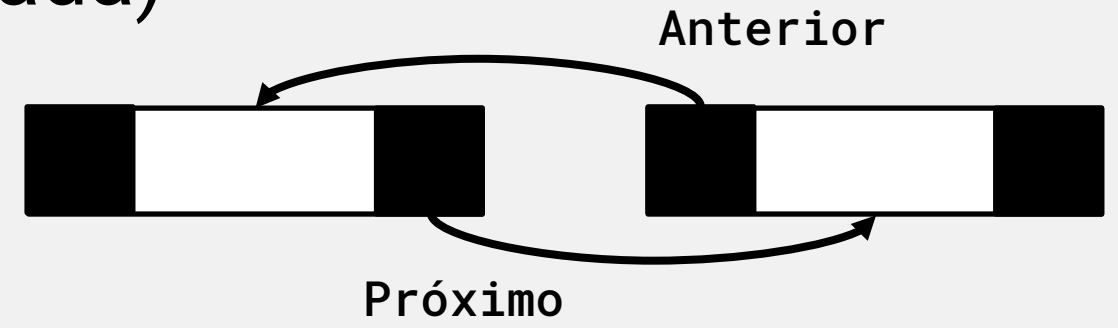


UDESC

Listas Duplamente Encadeadas

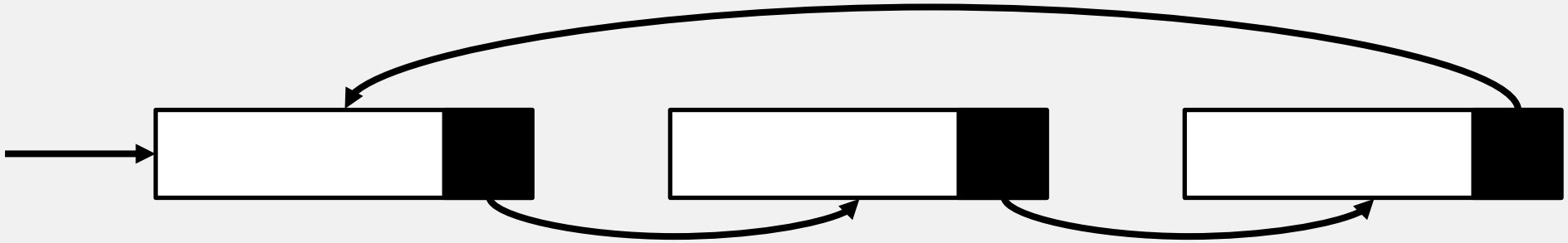
- Uma referência adicional **pode ser adicionada** para **referenciar** o nó anterior, **produzindo** uma lista duplamente encadeada (ligada)

```
private class Node
{
        private Object dado;
        private Node proximo;
        private Node anterior;
        ...
}
```



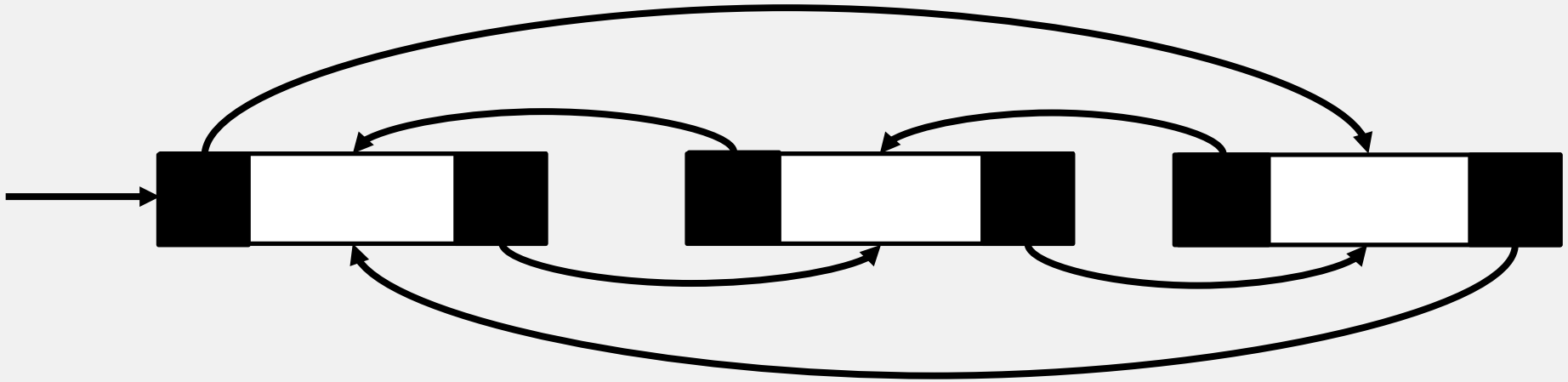
Variações de Listas Encadeadas

- O último nó numa **lista simplesmente encadeada** pode **referenciar** o primeiro nó, **produzindo** uma lista simplesmente encadeada circular



Variações de Listas Encadeadas

- O último nó numa **lista duplamente encadeada** pode **referenciar** o primeiro nó **com** a sua **referencia** “*proximo*” **e** o primeiro nó pode **referenciar** o último nó com a sua **referencia** “*anterior*”, **produzindo** uma lista duplamente encadeada circular



UDESC

Vantagens das Listas Encadeadas

- A inserção ou remoção de um **elemento na lista** não implica a mudança de lugar de **outros elementos** (diferente de um vetor, por exemplo, que precisa deslocar todos os elementos)
- **Não é necessário** definir, **no momento** da criação da lista, o número máximo de **elementos** que **ela poderá ter**. Ou seja, **é possível** alocar memória "dinamicamente", **apenas** para o número de nós **necessários**

Desvantagens das Listas Encadeadas

- A manipulação **torna-se** mais "perigosa" **uma vez que**, **se** o encadeamento (ligação) **entre** elementos da **lista for** mal feito, toda a lista **pode ser** perdida
- Para **acessar** o elemento na **posição** *n* da **lista**, **deve-se** percorrer os n - 1 **anteriores**

Classe LinkedList

- Na **linguagem** de programação Java, **existe** a **classe *LinkedList*** (**java.util.LinkedList;**) **que** implementa o conceito de listas encadeadas
- Esta classe **fornece** métodos para **obter**, **remover** e **inserir** um elemento no início e no final da lista, **além** de métodos necessários **para** operações de índices, percorrer a **lista** do início ao fim **e**, também, **outros** métodos opcionais
- Inclusive, também **fornece** métodos para manipulação de listas duplamente encadeadas

Classe LinkedList: Alguns Métodos

- add (int indice, Object elemento): **insere** um elemento em uma **posição específica** da lista
- clear(): **remove** todos os elementos da lista
- get (int indice): **retorna** o elemento na **posição especificada** na lista
- getFirst(): **retorna** o primeiro elemento da **lista**
- getLast(): **retorna** o último elemento desta **lista** remove(int indice): **remove** o elemento da **posição** especificada na lista
- size(): **retorna** o número de elementos da **lista**
- toArray(): **retorna** um array contendo todos os elementos da **lista**, na sequência do ligamento

Exemplos de Outras Estruturas

- **Exemplos** de outras estruturas de dados encadeadas:
  - pilhas (***Class Stack***):
    - *empty(), peek(), pop(), push(Obj item), search(Obj o)*;
  - filas (***Interface Queue***):
    - *element(), offer(E o), peek(), poll(), remove();*
  - árvores;
  - árvores binárias;
  - grafos;
  - grafos dirigidos.

Classes / Interfaces Java – Estruturas de Dados

- LinkedList
- Stack (Pilha)
- Queue (Fila)

COLEÇÕES

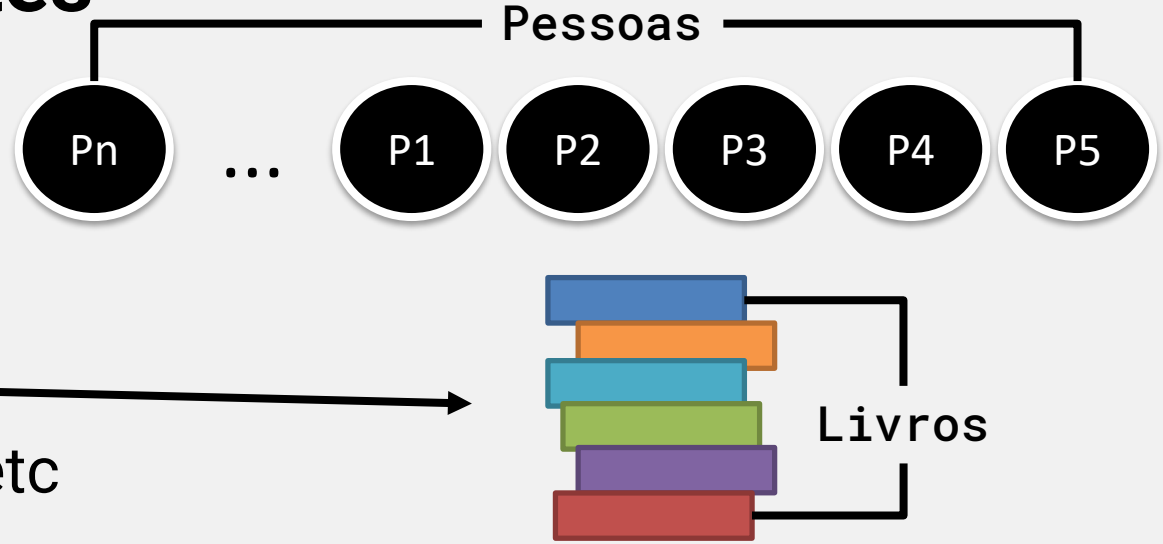
UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Agenda - Coleções

- Introdução
- *Java Collections Framework*
- Hierarquia das Interfaces
- Métodos e Classes da Interface *Collections*
- Métodos e Classes da Interface *List*
- Métodos e Classes da Interface *Set*
- Métodos e Classes da Interface *Map*

Introdução

- **Diariamente**, nos **deparamos** com várias situações **onde** as coleções **estão presentes**
  - **Por exemplo**:
    - Uma **fila de banco**
    - Uma **lista de compras**
    - Uma **pilha de livros**
    - Um **conjunto de elementos**, etc



- Em JAVA, o que **podemos chamar** de Coleção?
  - **Um** objeto onde **podemos** agrupar **vários** elementos

Introdução

- **Em** JAVA, nós **temos** toda **uma** arquitetura **para** representar e manipular coleções:
  - Interfaces: **Permitem** que as coleções **sejam manipuladas** independentes de **suas** implementações
  - Implementações: classes que i**mplementam** uma ou mais interfaces do ***framework***;
  - Algoritmos: **São** métodos que **realizam** operações (*sort*, *reverse*, etc) **sobre** as coleções

Java Collections Framework

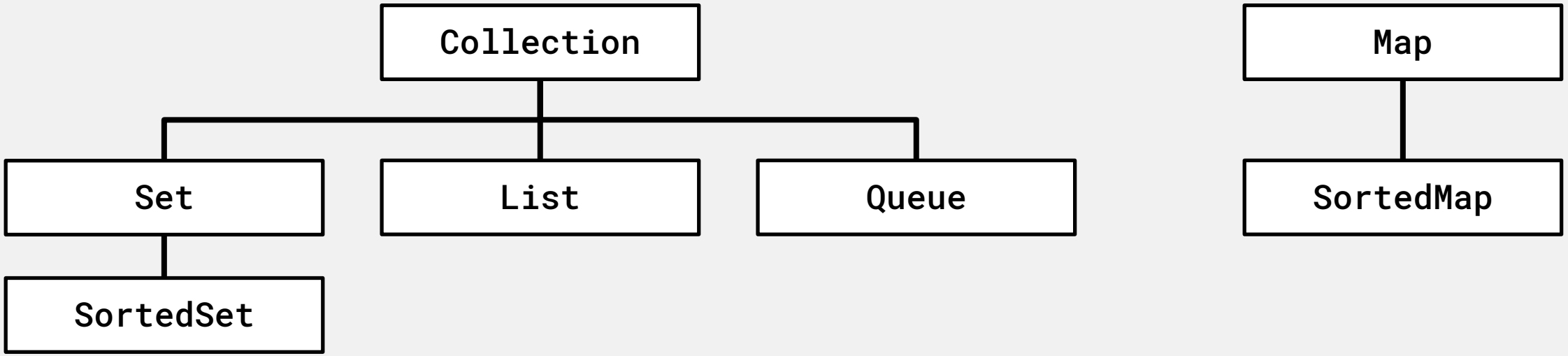
- A estrutura das coleções **é uma** arquitetura unificada para **representar** e **manipular** coleções, **permitindo** que **elas sejam** manipuladas **independentemente** dos detalhes de **sua** representação
- Isto **reduz** o esforço de programação, **aumentando** a performance, **permitindo** a interoperabilidade **entre** as APIs (*Application Programming Interface*), **independentemente**

Java Collections Framework

- **Reduz** o **esforço** em concepção e aprendizado de **novas** APIs, e **promove** a reutilização de software
- O **quadro completo** da coleção **baseia-se** em 14 interfaces, **incluindo** as implementações dessas i**nterfaces**, **e** os algoritmos para **manipulá-las**

Hierarquia das Interfaces

- **Abaixo** temos **a** hierarquia das interfaces do ***Java Collections Framework***:



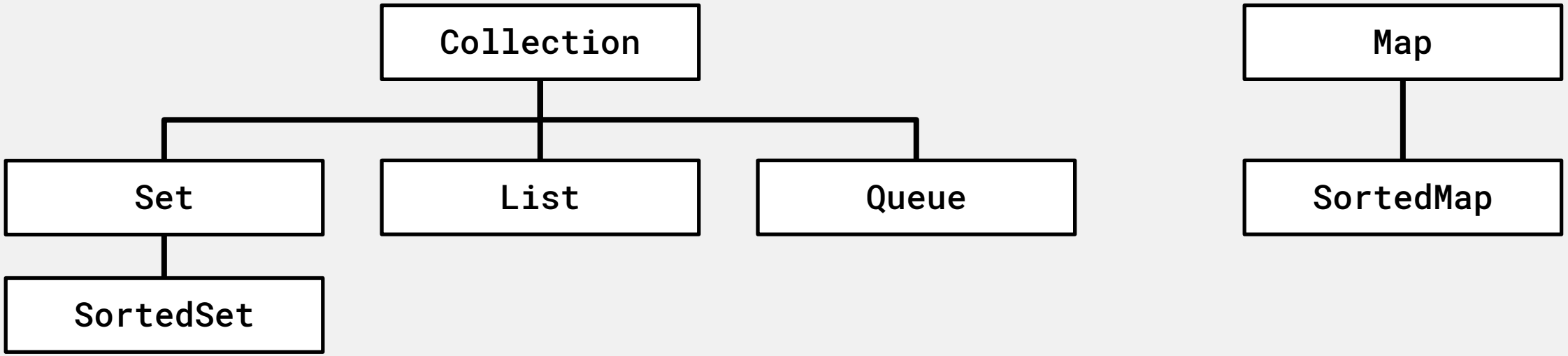
- **A seguir** vamos colocar **uma** breve descrição **sobre cada uma** das interfaces

Hierarquia das Interfaces

- **Collection**: O ***framework*** não possui implementação direta **desta** interface, porém, **ela está** no topo da hierarquia **definindo** operações (métodos) que **são comuns** a **todas** as coleções

Hierarquia das Interfaces

- **Abaixo** temos **a** hierarquia das interfaces do ***Java Collections Framework***:



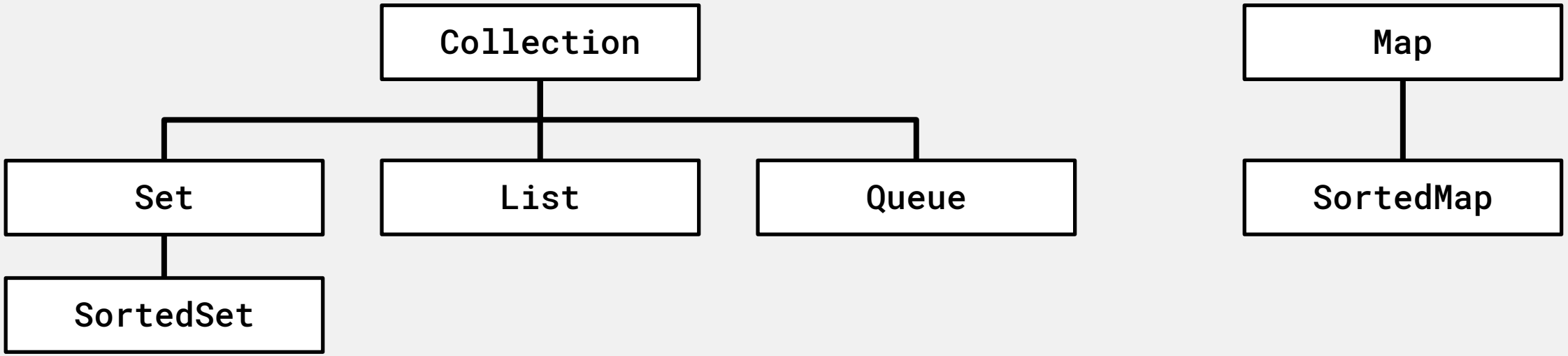
- **A seguir** vamos colocar **uma** breve descrição **sobre cada uma** das interfaces

Hierarquia das Interfaces

- **Set**: Está **diretamente** relacionada com a **ideia** de conjuntos. Assim **como um** conjunto, **as** classes que i**mplementam** esta interface não podem **conter** elementos repetidos
  - **Implementações** com *SortedSet* são **utilizadas** quando se **deseja** ordenar os elementos do **conjunto**

Hierarquia das Interfaces

- **Abaixo** temos **a** hierarquia das interfaces do ***Java Collections Framework***:



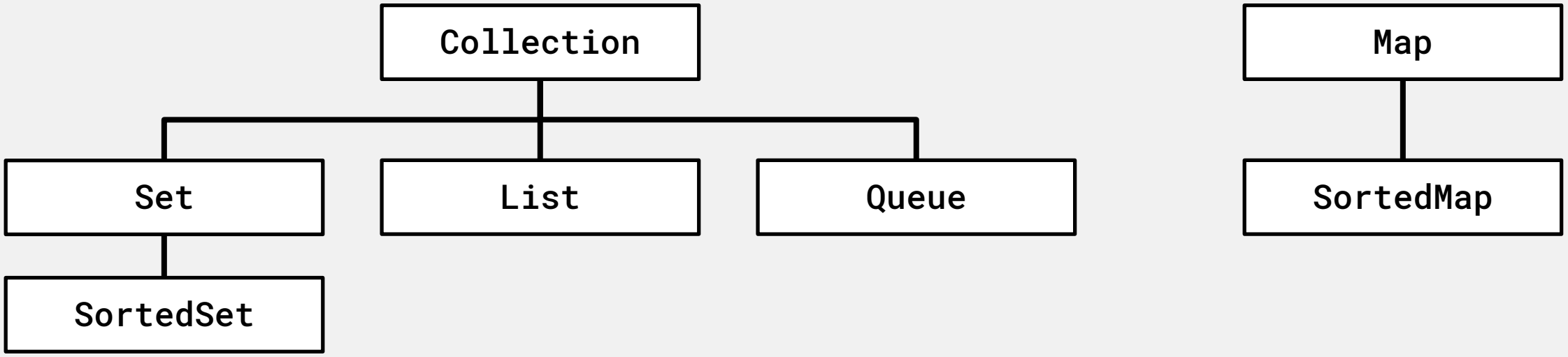
- **A seguir** vamos colocar **uma** breve descrição **sobre cada uma** das interfaces

Hierarquia das Interfaces

- **List**: É **uma** coleção ordenada, que **ao contrário** da **interface** Set, **pode conter** valores duplicados. Além disso, **permite** controle total **sobre** a posição **onde se encontra** cada elemento, **podendo acessar** cada um deles **pelo** índice

Hierarquia das Interfaces

- **Abaixo** temos **a** hierarquia das interfaces do ***Java Collections Framework***:



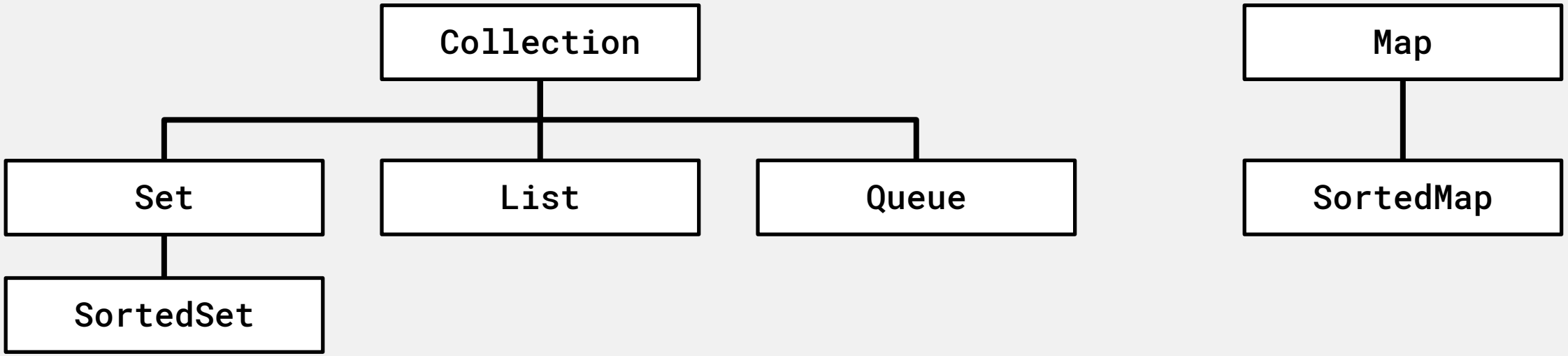
- **A seguir** vamos colocar **uma** breve descrição **sobre cada uma** das interfaces

Hierarquia das Interfaces

- **Queue**: Normalmente **utiliza-se** esta interface **quando queremos** uma coleção do **tipo** FIFO (***First-In-First-Out***), também **conhecida como** fila

Hierarquia das Interfaces

- **Abaixo** temos **a** hierarquia das interfaces do ***Java Collections Framework***:



- **A seguir** vamos colocar **uma** breve descrição **sobre cada uma** das interfaces

Hierarquia das Interfaces

- **Map**: É **utilizada quando** se **deseja** uma relação de chave-valor **entre** os elementos. **Cada** chave **pode conter** apenas um único valor **associado**.
  - **Usa-se** o *SortedMap* para **situações** onde **se deseja** ordenar os elementos

Métodos da Interface Collections

- A interface Collections **define** os métodos disponíveis **para operar** com as "coleções" de **objetos**:

- size()
- isEmpty()
- clear()
- add(Object)
- contains(Object)
- remove(Object)
- containsAll(Collection)
- removeAll(Collection)
- retainAll(Collection)
- toArray()

A Classe Collections

- **A** Classe Collection **é composta** exclusivamente **por** métodos estáticos **que operam** sobre coleções **retornando** outras novas coleções
- **Todos** os métodos **desta** classe **são especificados** para **retornar** exceções do **tipo** “*NullPointerException*” **se** as coleções **ou** objetos **manipulados** por ela **forem** nulos

A Classe Collections

- **Os** algoritmos “destrutivos” **desta** classe, ou seja, os algoritmos (**métodos**) **que modificam** a coleção original em **que operam**, são **especificados** para **retornar** exceções do **tipo** “*UnsupportedOperationException*” **se** a coleção **não suporta** a mutação do algoritmo

A Classe Collections: Principais Métodos

- reverse(List lista): **inverte** a **ordem** dos elementos de uma determinada lista
- sort(List lista): classifica a **lista especificada** em ordem crescente, de **acordo** com a ordem natural **dos** seus **elementos**
- max(Collection colecao): **retorna** o maior elemento da **coleção**, **de acordo com** a sua ordem natural
- min(Collection colecao): **retorna** o menor elemento da **coleção**, **de acordo com** a sua ordem natural

A Classe Collections: Principais Métodos

- frequency(Collection c, Object o): **retorna** o número de elementos da **coleção**, **de acordo com** o objeto especificado
- replaceAll(List lista, oldVal, newVal): **substitui todas** as ocorrências de um valor na lista, **por outro** valor especificado
- swap(List lista, int i, int j): **troca** os elementos nas posições indicadas de uma **determinada lista**
- **Lista completa** de métodos da classe Collections

Métodos da Interface List

- A interface *List* **define** os métodos para **operações** com listas:
  - get(int)
  - set(Object,int)
  - add(Object,int)
  - remove(int)
  - indexOf(Object)

Classes da Interface List

- **São** classes que **implementam** a interface List:
  - ArrayList: **implementa** uma lista de objetos num vetor **cujo tamanho pode** variar dinamicamente, mas a **capacidade é estática**
  - LinkedList: **implementa** uma lista de objetos **sob a forma** de uma lista encadeada (ligada)

Classes da Interface List: Indicações de Uso

- ArrayList é **mais adequada** em **situações** onde o acesso aleatório aos **elementos** é mais frequente. A **implementação** do **vetor** de ‘tamanho variável’ é **cara**
- LinkedList é **mais adequada** em **casos** onde o acesso aleatório não é frequente **e** o tamanho da **lista pode variar muito**

Métodos da Interface Set

- A interface Set **define** os métodos para **operações** com conjuntos de objetos (**diferente** das listas, um **conjunto** não pode **conter** elementos repetidos):
  - add(Object)
  - remove(Object)
  - contains(Object)
  - addAll(Set)
  - retainsAll(Set)
  - removeAll(Set)

Classes da Interface Set

- **Implementações** (classes) da interface Set:
  - HashSet: **baseada no conceito** de tabelas hash
  - TreeSet: **baseada** em árvore binária de **busca balanceada**

Métodos da Interface Map

- A interface *Map* **define** alguns dos seguintes métodos:
  - put(Object key, Object value)
  - get(Object key)
  - remove(Object key)
  - contains(Object key)
  - size()
  - isEmpty()
  - clear()

Classes da Interface Map

- São classes que **implementam** a interface *Map*:
  - HashMap: **baseada** em tabelas de hashing
  - TreeMap: **baseada** em árvore binária **de** busca balanceada
  - LinkedHashMap: **combinação** de tabela de hashing e lista encadeada

Leitura Complementar

- Caelum
- Devmedia

EXERCÍCIOS

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Exercícios

- Testar os exemplos:
  - ListaEncadeada.java
  - OrdenacaoColecoes.java (ordenação crescente e decrescente)
  - OperacoesColecoes.java
  - GeraVetor.java (Classe de auxílio ao trabalho da disciplina)

Exercícios

1. Desenvolva um programa em Java, utilizando o tipo de objeto ***LinkedList*** para representar uma lista encadeada. Faça as seguintes operações nesta lista:
   - Insira as letras de D até M
   - Exiba a lista
   - Insira as letras de A até C utilizando o métodos **addFirst**() de forma que a lista fique com as letras de A até M ordenadas
   - Exiba a lista
   - Remova o último elemento da ***LinkedList***
   - Exiba a lista

Exercícios

2. Ler um arquivo de texto simples e contar o número de palavras que este arquivo possui, armazenando cada palavra diferente em um ***ArrayList***. No final o programa deve listar todas as palavras armazenadas e informar o número de palavras distintas que foram encontradas no arquivo

Obrigado

jacksonpradolima.github.io
github.com/ceplan

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA