

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO CEARÁ**

**IFCE *CAMPUS* *CANINDÉ***

**SUBSEQUENTE EM INFORMÁTICA**

**THIAGO LEITÃO LOPES**

**USABILIDADE**

**CANINDÉ/CE**

**2022**THIAGO LEITÃO LOPES

USABILIDADE

Trabalho apresentado ao curso Subsequente em Informática do Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE) – *Campus Canindé*, como requisito parcial para obtenção da nota N2 da Disciplina Análise e Projeto de Sistemas II.

Professor: Alex Lacerda Ramos.

Coordenador: Prof. Allyson Bonetti Franca.

CANINDÉ/CE

2022

**LISTA DE FIGURAS**

|  |  |  |  |
| --- | --- | --- | --- |
| Figura 1 | — | Classe de coleção e hierarquia de interface........................................................ | 09 |
| Figura 2 | — | Classe de mapa e hierarquia de interface........................................................... | 10 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**LISTA DE TABELAS**

|  |  |  |  |
| --- | --- | --- | --- |
| Tabela 1 | — | Comparativo entre interfaces............................................................................... | 12 |
| Tabela 2 | — | Implementações de uso geral................................................................................ | 14 |

**LISTA DE SIGLAS**

|  |  |
| --- | --- |
| ABNT | Associação Brasileira de Normas Técnicas |
| ID | Identity |
| IFCE | Instituto Federal de Educação, Ciência e Tecnologia do Ceará |
| JAVA | Linguagem de programação |
| JDK | Java Development Kit |
| N2 | Nota 2 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**SUMÁRIO**

|  |  |  |
| --- | --- | --- |
| **1** | **INTRODUÇÃO**...................................................................................................... | 07 |
| **2** | **USABILIDADE**...................................................................................................... | 07 |
| **2.1** | **O que são Java Collections**...................................................................................... | 07 |
| **2.2** | **Para que são usadas**................................................................................................ | 08 |
| **2.3** | **Quais os principais tipos de Collections.**................................................................ | 10 |
| **2.4** | **Quais as vantagens e desvantagens de cada uma**................................................... | 12 |
| **2.5** | **Quando devem ser usadas**....................................................................................... | 14 |
| ***2.6*** | **Exemplos de utilização**............................................................................................ | 16 |
|  | **REFERÊNCIAS**..................................................................................................... | 19 |
|  | **APÊNDICE A — RELAÇÃO DE NORMAS TÉCNICAS VIGENTES UTILIZADAS NA NORMALIZAÇÃO DE TRABALHOS ACADÊMICOS**... | 20 |
|  | **ANEXO A — RESOLUÇÃO QUE APROVA A CRIAÇÃO DO CURSO SUPERIOR DE TECNOLOGIA EM GESTÃO AMBIENTAL NO IFCE CAMPUS PARACURU**......................................................................................... | 21 |

1. **INTRODUÇÃO**

Esse trabalho consiste na realização de uma pesquisa sobre Java Collections e elaboração de um relatório contemplando os seguintes aspectos: O que são Java Collections; Para que são usadas; Quais os principais tipos de Collections; Quais as vantagens de desvantagens de cada uma: Quando devem ser usadas; Mostrar exemplos de utilização.

O trabalho deve conter o máximo de detalhes possível, como explicações, figuras, diagramas, trechos de código de implementação, etc.

O documento também deverá conter uma capa com o nome do aluno, instituição, curso, disciplina, professor e título do trabalho.

1. **JAVA COLLECTIONS**
   1. **O que são Java Collections**

A estrutura de coleções Java é um conjunto de classes e interfaces que implementam estruturas de dados de coleção comumente reutilizáveis, ainda que seja referido como uma estrutura, ele funciona como uma biblioteca.

Em resumo é um conjunto bem definido de interfaces e classes para representar e tratar grupos de dados como uma única unidade.

A estrutura de coleções fornece as interfaces que definem várias coleções e classes que as implementam.

A Collections Framework contém os seguintes elementos:

*Interfaces*: tipos abstratos que representam as coleções. Permitem que coleções sejam manipuladas tendo como base o conceito “Programar para interfaces e não para implementações”, desde que o acesso aos objetos se restrinja apenas ao uso de métodos definidos nas interfaces;

*Implementações*: são as implementações concretas das interfaces;

*Algoritmos*: são os métodos que realizam as operações sobre os objetos das coleções, tais como busca e ordenação.

* 1. **Para que são usadas**

Desde as primeiras versões, o Java dispõe das estruturas de arrays e as classes Vector e Hashtable. No entanto, além da dificuldade em implementar estruturas de dados utilizando arrays, os desenvolvedores sentiam falta de classes que implementassem estruturas como listas ligadas e tabelas de espalhamento(hash), além da dificuldade que os usuários enfrentam para lembrar de todos os diferentes métodos, sintaxe e construtores presentes em cada classe de coleção.

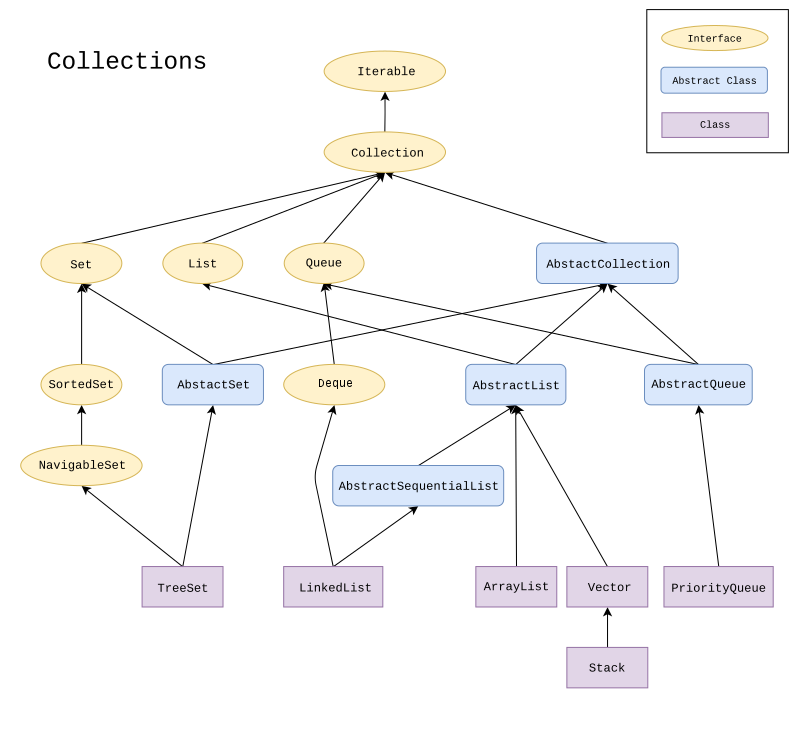
Todas essas coleções não tinham uma interface comum, embora o objetivo principal de todas as coleções seja o mesmo, a implementação de todas essas coleções foi definida de forma independente e não teve correlação entre elas.

Para atender a essas necessidades, a partir de Java JDK 1.2, foi criado um conjunto de interfaces e classes denominado Collections Framework, que faz parte do pacote java.util.

O pacote do utilitário (java.util) contém todas as classes e interfaces exigidas pela estrutura de coleta. A estrutura da coleção contém uma interface denominada interface iterável que fornece o iterador para iterar por todas as coleções.

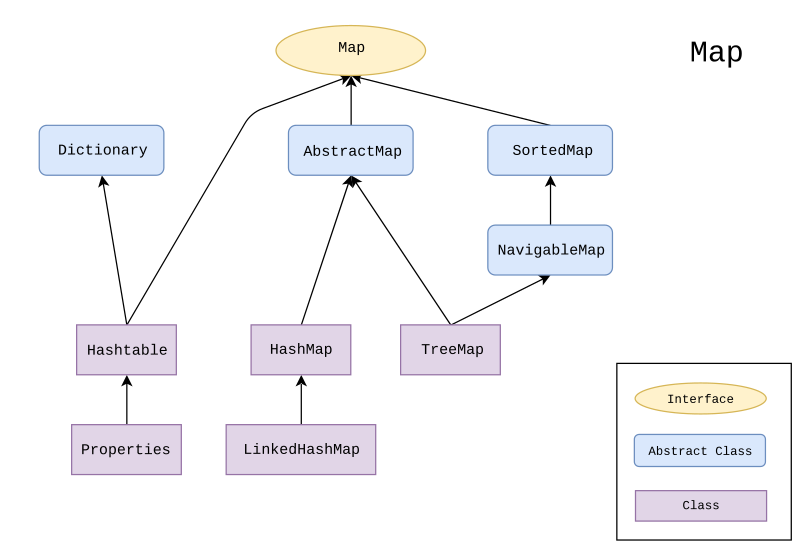
A interface Collection (java.util.Collection) e a interface Map (java.util.Map) são as duas interfaces “raiz” principais das classes de coleção Java.

Figura 1 — Classe de coleção e hierarquia de interface.



Fonte: Internet - <https://en.wikipedia.org/>, 2022.

Figura 2 — Classe de mapa e hierarquia de interface



Fonte: Internet - <https://en.wikipedia.org/>, 2022.

* 1. **Quais os principais tipos de Collections**

Temos quatro grandes tipos de coleções: Set (conjunto), List (lista), Queue (fila) e Map (mapa), a partir dessas interfaces, temos muitas subclasses concretas que implementam várias formas diferentes de se trabalhar com cada coleção.

Todas as interfaces e classes são encontradas dentro do pacote (package) java.util, embora a interface Map não ser filha direta da interface Collection ela também é considerada uma coleção devido a sua função.

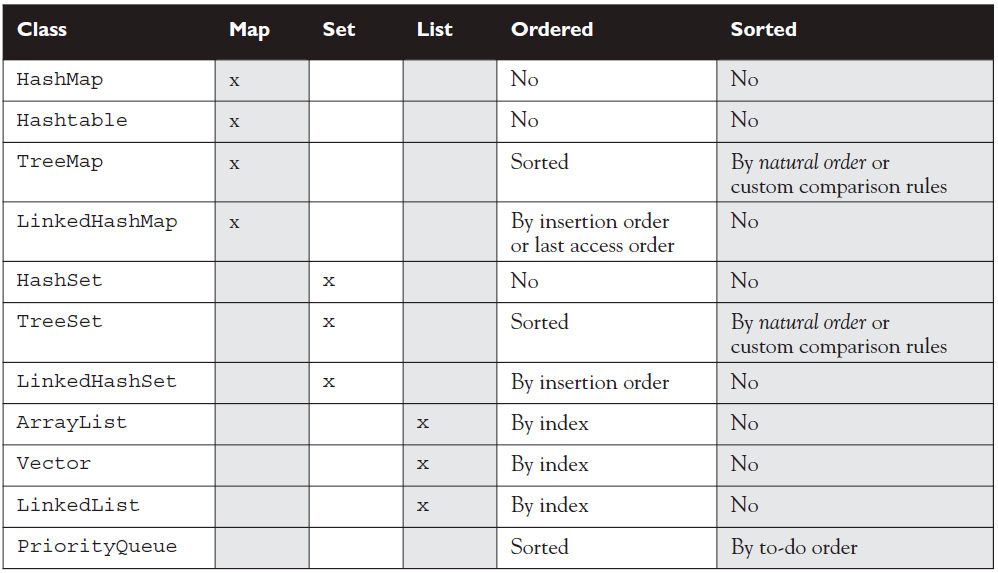
* + 1. *Set(Conjunto)* - é uma coleção do tipo conjunto de elementos. As características principais deste tipo de coleção são: os elementos não possuem uma ordem de inserção e não é possível ter dois objetos iguais.
    2. *List(Lista)* - é uma coleção do tipo lista, em que a ordem dos elementos é dado através de sua inserção dentro da lista.
    3. *Queue* - é uma coleção do tipo fila. As principais características deste tipo de coleção são: a ordem que os elementos entram na fila é a mesma ordem que os elementos saem da fila (FIFO - First In First Out), podemos também criar filas com prioridades.
    4. *Map(Mapa)* - é uma coleção do tipo mapa. As principais características deste tipo de coleção são: os objetos são armazenados na forma de chave / valor, não pode haver chaves duplicadas dentro do mapa.

As principais diferenças dentre as interfaces citadas acima são:

* *List*: Lista de coisas;
* *Set:* Lista de coisas sem repetição;
* *Map*: Lista com chave-valor, sendo que a chave deve ser única;
* *Queue*: Fila.

Cada interface possui classes que as implementam. Como são muitas vou apenas colocar o quadro comparativo de cada uma delas, que acho que resume muito bem a funcionalidade de cada uma:

##### Tabela 01 - Comparativo entre interfaces

Fonte: Internet - <https://pt.stackoverflow.com>, 2022.

* 1. **Quais as vantagens e desvantagens de cada uma**
     1. *Set* - O uso de um Set pode parecer desvantajoso: não armazena a ordem, e não aceita elementos repetidos, não há métodos que trabalham com índices, como o get(int) que as listas possuem.

A grande vantagem do Set é que existem implementações, como a HashSet, que possui uma performance incomparável com as Lists quando usado para pesquisa (método contains por exemplo).

* + 1. *Queue* - Para pegar um elemento é muito fácil: basta pegarmos a célula em que aquele elemento se encontra e acessar o elemento de dentro dela Perceba que este método consome tempo linear. Esta é uma grande desvantagem da Lista Ligada em relação aos Vetores. Vetores possuem o chamado acesso aleatório aos elementos: qualquer posição pode ser acessada em tempo constante. Apesar dessa grande desvantagem, diversas vezes utilizamos uma Lista e não é necessário ficar acessando posições aleatórias: comumente percorremos a lista por completa.
    2. *List* - Ao usarmos um tipo mais genérico como List ao declarar uma variável e um tipo específico como ArrayList apenas no momento de instanciar um objeto para aquela variável, estamos tornando nosso código mais flexível a mudanças, Vantagem: aceita qualquer tipo, Desvantagem: retorna tipo genérico e requer downcasting que só é verificado em tempo de execução.
    3. *LinkedList:* possui uma grande vantagem, utiliza a estrutura de dados chamada lista ligada, e é bastante rápido para adicionar e remover elementos na cabeça da lista, isto é, na primeira posição. Mas é lento se você precisar acessar um determinado elemento, pois a implementação precisará percorrer todos os elementos até chegar ao décimo quinto, por exemplo.
    4. *ArrayList*: internamente usa um array para guardar os elementos, consegue fazer operações de maneira muito eficiente, como invocar o método get(índice). Se você precisa pegar o décimo quinto elemento, ele te devolverá isso bem rápido. Quando um ArrayList é lento? Quando você for, por exemplo, inserir um novo elemento na primeira posição. Pois a implementação vai precisar mover todos os elementos que estão no começo da lista para a próxima posição
  1. **Quando devem ser usadas**

As interfaces apresentadas anteriormente possuem diversas implementações que são utilizadas para armazenar as coleções, na tabela abaixo estão resumidas as implementações mais comuns:

##### Tabela 02 - Implementações de uso geral

##### 

Fonte: Internet - <https://www.devmedia.com.br>, 2022).

Abaixo seguem apresentadas algumas características das implementações que podem ajudar a decidir qual delas utilizar em uma aplicação:

* + 1. *ArrayList* –é como um array cujo tamanho pode crescer. A busca de um elemento é rápida, mas inserções e exclusões não são. Podemos afirmar que as inserções e exclusões são lineares, o tempo cresce com o aumento do tamanho da estrutura. Esta implementação é preferível quando se deseja acesso mais rápido aos elementos. Por exemplo, se você quiser criar um catálogo com os livros de sua biblioteca pessoal e cada obra inserida receber um número sequencial (que será usado para acesso) a partir de zero;
    2. *LinkedList* – implementa uma lista ligada, ou seja, cada nó contém o dado e uma referência para o próximo nó. Ao contrário do ArrayList, a busca é linear e inserções e exclusões são rápidas. Portanto, prefira LinkedList quando a aplicação exigir grande quantidade de inserções e exclusões. Um pequeno sistema para gerenciar suas compras mensais de supermercado pode ser uma boa aplicação, dada a necessidade de constantes inclusões e exclusões de produtos;
    3. *HashSet* – o acesso aos dados é mais rápido que em um TreeSet, mas nada garante que os dados estarão ordenados. Escolha este conjunto quando a solução exigir elementos únicos e a ordem não for importante. Poderíamos usar esta implementação para criar um catálogo pessoal das canções da nossa discografia.
    4. *TreeSet* – os dados são classificados, mas o acesso é mais lento que em um HashSet. Se a necessidade for um conjunto com elementos não duplicados e acesso em ordem natural, prefira o TreeSet. É recomendado utilizar esta coleção para as mesmas aplicações de HashSet, com a vantagem de os objetos estarem em ordem natural.
    5. *LinkedHashSet* – é derivada de HashSet, mas mantém uma lista duplamente ligada através de seus itens. Seus elementos são iterados na ordem em que foram inseridos. Opcionalmente é possível criar um LinkedHashSet que seja percorrido na ordem em que os elementos foram acessados na última iteração. Poderíamos usar esta implementação para registrar a chegada dos corredores de uma maratona;
    6. *HashMap* – baseada em tabela de espalhamento, permite chaves e valores null. Não existe garantia que os dados ficarão ordenados. Escolha esta implementação se a ordenação não for importante e desejar uma estrutura onde seja necessário um ID (identificador). Um exemplo de aplicação é o catálogo da biblioteca pessoal, onde a chave poderia ser o ISBN (International Standard Book Number);
    7. *TreeMap* – implementa a interface SortedMap. Há garantia que o mapa estará classificado em ordem ascendente das chaves. Mas existe a opção de especificar uma ordem diferente. Use esta implementação para um mapa ordenado. Aplicação semelhante a HashMap, com a diferença que TreeMap perde no quesito desempenho;
    8. *LinkedHashMap* – mantém uma lista duplamente ligada através de seus itens. A ordem de iteração é a ordem em que as chaves são inseridas no mapa. Se for necessário um mapa onde os elementos são iterados na ordem em que foram inseridos, use esta implementação. O registro dos corredores de uma maratona, onde a chave seria o número que cada um recebe no ato da inscrição, é um exemplo de aplicação desta coleção
  1. **Exemplos de utilização**
     1. *Exemplo 01 -* java.util.List

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Classe utilizada para demonstrar o uso da estrutura

\* de dados Lista.

\*/

public class ExemploLista {

public static void main(String[] args) {

List nomes = new ArrayList();

nomes.add("Zezinho");

nomes.add("Luizinho");

nomes.add("Joãozinho");

for(int cont = 0; cont < nomes.size(); cont++) {

String nome = (String) nomes.get(cont);

System.out.println(nome);

}

}

}

* + 1. *Exemplo 02 -* interface List

import java.util.\*;

public class ListaAluno {

public static void main(String[] args) {

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

lista.add("João da Silva");

lista.add("Antonio Sousa");

lista.add("Lúcia Ferreira");

System.out.println(lista);

}

}

* + 1. *Exemplo 03 -* interface Queue

import java.util.\*;

class GfG {

public static void main(String args[])

{

// Creating empty priority queue

PriorityQueue<Integer> pQueue

= new PriorityQueue<Integer>();

// Adding items to the pQueue using add()

pQueue.add(10);

pQueue.add(20);

pQueue.add(15);

// Printing the top element of PriorityQueue

System.out.println(pQueue.peek());

// Printing the top element and removing it

// from the PriorityQueue container

System.out.println(pQueue.poll());

// Printing the top element again

System.out.println(pQueue.peek());

}

}

**REFERÊNCIAS**

ANDRADE, M. M. **Introdução à metodologia do trabalho científico**: elaboração de trabalhos na graduação. 10. ed. São Paulo: Atlas, 2010.

\_\_\_\_\_\_. **O que são Java Collections**. Rio de Janeiro, c2022. Disponível em: < <https://en.wikipedia.org/wiki/Java_collections_framework>>. Acesso em: 11 jun. 2022.

\_\_\_\_\_\_. **Para que são usadas**. Rio de Janeiro, c2022. Disponível em: <<https://www.devmedia.com.br/java-collections-como-utilizar-collections/18450>>. Acesso em: 11 jun. 2022.

\_\_\_\_\_\_. **Quais os principais tipos de Collections**. Rio de Janeiro, c2022. Disponível em: <<http://www.universidadejava.com.br/java/java-collection/>>. Acesso em: 11 jun. 2022.

\_\_\_\_\_\_. **Comparativo entre interfaces**. Rio de Janeiro, c2022. Disponível em: <<https://pt.stackoverflow.com/questions/34605/quais-tipos-de-cole%C3%A7%C3%B5es-e-suas-diferen%C3%A7as-em-java>>. Acesso em: 11 jun. 2022.

\_\_\_\_\_\_. **Quais as vantagens de desvantagens de cada uma**. Rio de Janeiro, c2022. Disponível em: <<https://www.alura.com.br/conteudo/java-collections>>. Acesso em: 11 jun. 2022.

\_\_\_\_\_\_. **Quando devem ser usadas**. Rio de Janeiro, c2022. Disponível em: <<https://www.devmedia.com.br/java-collections-como-utilizar-collections/18450>>. Acesso em: 11 jun. 2022.

\_\_\_\_\_\_. **Implementações de uso geral**. Rio de Janeiro, c2022. Disponível em: <<https://www.devmedia.com.br/java-collections-como-utilizar-collections/18450>>. Acesso em: 11 jun. 2022.

\_\_\_\_\_\_. **Exemplos de utilização**. Icapuí, c2022. Disponível em: <<https://acervolima.com/colecoes-em-java/>>. Acesso em: 11 jun. 2022.

\_\_\_\_\_\_. **Exemplos de utilização**. Rio de Janeiro, c2022. Disponível em: <<http://www.universidadejava.com.br/java/java-collection/> Acesso em: 11 jun. 2022.

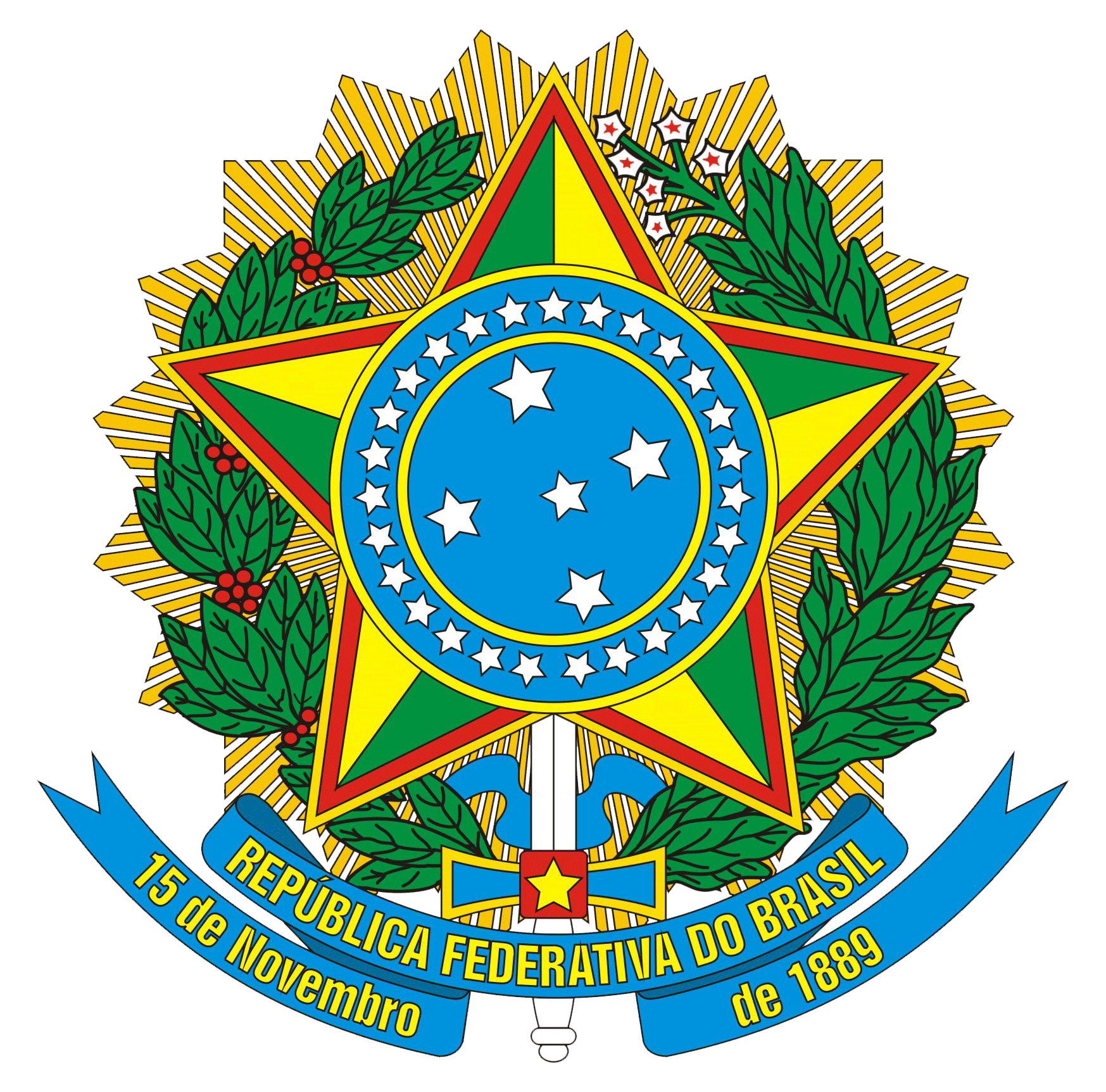
**APÊNDICE A** – **RELAÇÃO DE NORMAS TÉCNICAS VIGENTES UTILIZADAS NA NORMALIZAÇÃO DE TRABALHOS ACADÊMICOS**

Quadro 2 — Normas técnicas vigentes sobre normalização de trabalhos acadêmicos do ABNT/CB - 014

|  |  |
| --- | --- |
| **Número** | **Título** |
| 6022:2018 | Artigo em publicação periódica técnica e/ou científica - Apresentação |
| 6023:2002 | Referências - Elaboração |
| 6024:2012 | Numeração progressiva das seções de um documento - Apresentação |
| 6027:2012 | Sumário - Apresentação |
| 6028:2003 | Resumo - Apresentação |
| 6034:2004 | Índice - Apresentação |
| 10520:2002 | Citações em documentos - Apresentação |
| 10719:2015 | Relatório técnico e/ou científico - Apresentação |
| 12225:2004 | Lombada - Apresentação |
| 14724:2011 | Trabalhos acadêmicos - Apresentação |
| 15287:2011 | Projeto de pesquisa - Apresentação |
| 15437:2006 | Pôsteres técnicos e científicos - Apresentação |

Fonte: elaborado pelo autor, de acordo com o Catálogo da ABNT.

**ANEXO A — RESOLUÇÃO QUE APROVA A CRIAÇÃO DO CURSO SUPERIOR DE TECNOLOGIA EM GESTÃO AMBIENTAL NO IFCE CAMPUS PARACURU**

****

**SERVIÇO PÚBLICO FEDERAL**

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO CEARÁ**

**CONSELHO SUPERIOR**

**RESOLUÇÃO N° 01, DE 10 DE JANEIRO DE 2018**

Aprova *ad referendum* a criação do curso Superior de Tecnologia em Gestão Ambiental no *campus* Paracuru.

**O PRESIDENTE EM EXERCÍCIO DO CONSELHO SUPERIOR DO INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO CEARÁ**, no uso de suas atribuições legais e estatutárias e considerando o Memorando nº 001/2018/GDG da direção-geral do *campus* Paracuru,

**R E S O L V E:**

**Art. 1º -** Criar, *ad referendum* do Conselho Superior, o curso Superior de Tecnologia em Gestão Ambiental do *campus* Paracuru e autorizar a oferta de 35 vagas semestrais.

**Parágrafo único -** O curso será ofertado na modalidade presencial e nos turnos matutino e vespertino, conforme definido no projeto pedagógico em anexo.

**Art. 2º -** A interrupção da oferta e/ou a extinção do referido curso deverá ser submetida a este conselho para aprovação, com as devidas justificativas e a apresentação do planejamento de realocação de recursos humanos e de materiais vinculados ao curso.

José Wally Mendonça Menezes

**Presidente em exercício do Conselho Superior**