Índice

Assuntos: JAVA Collections, JAVA Estrutura de dados.

Abertura Alunos:
Reflexão diária: Autocontrole
Giovanni Torelli e Samuel Ferreira.

JAVA Collections

Para maiores informações acesse o Cookbook MD15, na aba de Links.

A estrutura de dados serve para melhor armazenamento e organização de dados.

Elas não suportam dados primitivos: byte, short, int, long, char, float, double, boolean.

Ela acaba por usar umas versões modificadas desses dados, sendo encapsulados em um objeto ou das classes Wrapper.

Elemento é o conteúdo de uma collection, índice é a posição desse elemento.

Não se esqueça de importar as bibliotecas.

A Classe Wrapper transforma um primitivo em Objeto e adiciona Métodos.

Tabela Wrapper

Tipo Primitivo	Tamanho	Wrapper
boolean	*true* ou *false*	**Boolean**
char	16 bits	**Character**
byte	08 bits	**Byte**
short	16 bits	**Short**
int	32 bits	**Integer**
long	64 bits	**Long**
float	32 bits	**Float**
double	64 bits	**Double**
String	Não é primitivo	

Comandos

sort() Ordena a lista.
max() Retorna o maior elemento.
min() Retorna o menor elemento.

Iterator

Iterator<Tipo> nomeDoIterator = nomeDaCollection.iterator();

Permite que um usuário percorra todos os objetos de uma collection.

Deve ser usado sempre que quisermos enumerar elementos em todas as interfaces implementadas pelas collections: **Set**, **List**, **Queue**, **Deque**, e todas implementadas da interface **Map**.

hasNext( ) Retorna verdadeiros e a iteração tiver mais elementos.

next( ) Retorna o próximo elemento na iteração.

Ele lança "**NoSuchElementException**" se nenhum outro elemento estiver presente.

List

É uma interface que herda da interface Collection. Ela funciona como um vetor, mas com mais funcionalidades.

Como qualquer Array, a lista começa em 0.
Ele aceita elementos repetidos.

Comandos

ArrayList<*Tipo*> *nome* = new ArrayList<*Tipo*>(); Subclasse de List.
Usada para se criar um vetor redimensionável, que é muito mais eficiente para leitura, por ser implementado internamente com vetores, o que a torna ideal para o acesso aleatório aos dados armazenados.

.add (*objeto*) Adiciona um objeto ao final da lista.

.add(*índice, objeto*) Adiciona um objeto na posição de índice indicada. Demais itens são empurrados para frente.

.get(*índice*) Recupera um objeto pelo índice.

.indexOf(objeto) Procura um objeto, e retorna o índice da primeira ocorrência do objeto.

.set(*índice,objeto*) Grava um objeto na posição indicada. Apaga qualquer um que ocupava esta posição.

.remove(*índice*) Apaga o objeto nesta posição.

.clear( ) Limpa a lista.

.size( ) Retorna o tamanho da lista.

.isEmpty( ) Retorna true se a lista está vazia.

.contains(*Objeto*) Retorna true se existe uma ocorrência do elemento na lista.

.sort(null) Ordena os elementos em ordem crescente.

.sort(Comparator.reverseOrder()) Ordena os elementos em ordem decrescente.

Collections.max(*lista*)); Retorna o maior valor na lista.

system.out.println(*lista*.to*Tipo*()); Define uma forma da lista ser representada na tela. **lista** representa o nome da sua lista, **Tipo** representa o tipo de variável, como String, float, int, etc.

Set

Coleção não ordenada de objetos, que permite apenas valores únicos.
Como qualquer Array, ele começa no 0.
NÃO aceita elementos repetidos.

Os objetos inseridos não serão inseridos necessariamente na mesma ordem, e inclusive essa ordem não é constante ao longo do tempo.

Hashcode, HashSet, HashMap

Os objetos são inseridos na Collection Set com base em seu **Hashcode**. O **Hashcode** é um número inteiro com 7 dígitos, calculado a partir do Método **Hashcode( )**.
A partir do Hashcode do Objeto, obtido pelo Método **hashcode( )**, a Collection Set determina a posição onde o objeto será armazenado no **Hashmap**.
Por se tratar de um número calculado, a posição do elemento será aleatória.

O **HashSet** usa internamente o **HashMap** para armazenar seus elementos. Sempre que você cria um objeto HashSet, um objeto HashMap associado a ele também é criado.
Este objeto HashMap é usado para armazenar os elementos inseridos no HashSet.
Os elementos adicionados ao HashSet são armazenados como chaves desse objeto HashMap.
O valor associado a essas chaves será uma constante (**PRESENT**).

Comandos

Set<*Tipo*> *nome* = new HashSet<*Tipo*>(); Criação de um Set com HashSet.

.add(Objeto) Adiciona um Objeto na Collection Set.

.remove(Objeto) Apaga o objeto armazenado na Collection Set.

.clear() Limpa a Collection Set

.size() Retorna o tamanho da Collection Set (número de elementos armazenados).

.isEmpty() Retorna true se a Collection Set está vazia.

.contains(Objeto) Retorna true se o Objeto existe na Collection Set.

hashcode() Retorna o Hashcode do elemento (tipo um id do elemento).

Estrutura de dados

Queue

Uma estrutura de collection em fila, os dados entram por ordem de chegada, não se especifica uma posição.
O último a entrar é o primeiro a sair.
Ao remover ele removerá o primeiro.
Ao adicionar, ele adiciona no final.
Como uma fila comum. Se você chega primeiro, é atendido primeiro, e vice-versa.

Comandos

Queue <*Tipo*> *nome* = new LinkedList <*Tipo*> ( );
Comando para criar uma nova estrutura **Queue**.

.add ( ) Adiciona um elemento ao final da fila.

.size ( ) Retorna o tamanho da fila.

.clear ( ) Remove todos os elementos da fila.

.remove (*elemento*) Remove e retorna o primeiro elemento da fila

.isEmpty( ) Verifica se a fila está vazia ou não. Caso vazia retorna true. Caso contrário, false.

.contains (*elemento*) Verifica se a fila contém o elemento especificado. Caso sim, retorna true. Caso não, retorna false.

.peek ( ) Exibe o PRIMEIRO elemento da fila (HEAD). Caso a fila esteja vazia, retorna nulo (*null*).

.poll ( ) Exibe e remove o PRIMEIRO elemento da fila (HEAD). Caso a fila esteja vazia, retorna nulo (*null*).

Stack

Uma estrutura collection em pilha, os dados entram por ordem de chegada, não se especifica uma posição.
O último a entrar é o primeiro a sair.
Ao remover ele removerá o último.
Ao adicionar ele adiciona no final.
Imagine uma pilha de pratos. O primeiro que foi colocado fica inacessível, para removê-lo você deve mexer em todos os que estão acima dele.

Comandos

Stack <*Tipo*> *nome* = new Stack <*Tipo*> ( );
Comando para criar uma nova estrutura **Stack**.

.push ( ) Adiciona um elemento no TOPO da pilha.

.size ( ) Retorna o tamanho da pilha.

.clear ( ) Remove todos os elementos da pilha.

.pop ( ) Remove e retorna o elemento no TOPO da pilha.

.isEmpty ( ) Verifica se a pilha está vazia ou não. Caso sim ele retorna *true*, caso contrário retorna *false*.

.contains (elemento) Verifica se a pilha contém o elemento em específico. Caso sim retorna *true*, caso contrário retorna *false*.

.peek ( ) Exibe, sem remover, o elemento no TOPO da pilha. Caso a pilha esteja vazia, retorna nulo (*null*).

Links e afins

Cookbook MD15, Collections:

https://github.com/conteudoGeneration/cookbook_java_fullstack/blob/main/01_java/15.md

Cookbook MD18, Estrutura de Dados:

https://github.com/conteudoGeneration/cookbook_java_fullstack/blob/main/01_java/18.md

---

Dúvidas

tudo kkkkkkkkkkkkkkk #help #help2

não entendi nada nada nada KKKKKKKKKKKKKKK