LinkedList

La clase LinkedList es casi idéntica a ArrayList

ArrayList frente a LinkedList

La clase LinkedList es una colección que puede contener muchos objetos del mismo tipo, al igual que ArrayList.

La clase LinkedList tiene todos los mismos métodos que la clase ArrayList porque ambos implementan la interfaz List. Esto significa que puede agregar elementos, cambiar elementos, eliminar elementos y borrar la lista de la misma manera.

ArrayList es la favorita para realizar búsquedas en una lista y podemos decir que ArrayList es más rápida para búsquedas que LinkedList. Si tenemos una lista y lo que nos importa no es buscar la información lo más rápido posible, sino que la inserción o eliminación se hagan lo más rápidamente posible, LinkedList resulta una implementación muy eficiente

Sin embargo, aunque la clase ArrayList y la clase LinkedList se pueden usar de la misma manera, se construyen de manera muy diferente.

Cómo funciona ArrayList

La clase ArrayList tiene una matriz regular dentro de ella. Cuando se agrega un elemento, se coloca en la matriz. Si la matriz no es lo suficientemente grande, se crea una nueva matriz más grande para reemplazar la anterior y se elimina la anterior.

Cómo funciona LinkedList

LinkedList almacena sus elementos en "contenedores". La lista tiene un enlace al primer contenedor y cada contenedor tiene un enlace al siguiente contenedor de la lista. Para agregar un elemento a la lista, el elemento se coloca en un nuevo contenedor y ese contenedor se vincula a uno de los otros contenedores de la lista.

Cuándo usar

Es mejor usar ArrayList cuando:

- Quieres acceder a elementos aleatorios con frecuencia
- Solo necesita agregar o eliminar elementos al final de la lista

Es mejor usar una LinkedList cuando:

- Solo usa la lista recorriéndola en lugar de acceder a elementos aleatorios
- Con frecuencia es necesario agregar y eliminar elementos desde el principio o la mitad de la lista

Métodos LinkedList

Para muchos casos, ArrayList es más eficiente ya que es común necesitar acceso a elementos aleatorios en la lista, pero LinkedList proporciona varios métodos para realizar ciertas operaciones de manera más eficiente:

addFirst () Agrega un elemento al principio de la lista.

addLast () Agrega un elemento al final de la lista

removeFirst () Elimina un elemento del principio de la lista.

removeLast () Elimina un elemento del final de la lista

getFirst () Obtiene el elemento al principio de la lista

getLast () Obtiene el elemento al final de la lista

Java HashMap

En el uso de ArrayList, se aprendió que las matrices almacenan elementos como una colección ordenada y debe acceder a ellos con un número de índice (tipo int). Sin embargo, un HashMap almacena elementos en pares "clave / valor" y puede acceder a ellos mediante un índice de otro tipo (por ejemplo, una cadena).

Un objeto se utiliza como clave (índice) para otro objeto (valor). Puede almacenar diferentes tipos: claves de cadena y valores enteros, o el mismo tipo, como: claves de cadena y valores de cadena:

Ejemplo

Cree un objeto HashMap llamado capitalCities que almacenará claves de cadena y valores de cadena:

```
import java.util.HashMap; // import the HashMap class

HashMap<String, String> capitalCities = new HashMap<String, String>();
```

Agregar elementos

La clase HashMap tiene muchos métodos útiles. Por ejemplo, para agregarle elementos, use el método put ():

```
// Import the HashMap class
import java.util.HashMap;

public class MyClass {
  public static void main(String[] args) {
    // Create a HashMap object called capitalCities
    HashMap<String, String> capitalCities = new HashMap<String, String>();
```

```
    // Add keys and values (Country, City)
    capitalCities.put("England", "London");
    capitalCities.put("Germany", "Berlin");
    capitalCities.put("Norway", "Oslo");
    capitalCities.put("USA", "Washington DC");
    System.out.println(capitalCities);
  }
}
```

Acceder a un ítem

Para acceder a un valor en el HashMap, use el método get () y consulte su clave:

```
capitalCities.get("England");
```

Eliminar un item

Para eliminar un elemento, utilice el método remove () y consulte la clave:

```
capitalCities.remove("England");
```

Para eliminar todos los elementos, utilice el método clear ():

```
capitalCities.clear();
```

HashMap Size

```
capitalCities.size();
```

Recorrer un HashMap

Recorra los elementos de un HashMap con un bucle para cada uno.

Nota: Use el método keySet () si solo desea las claves, y use el método values () si solo desea los valores:

```
// Print keys
for (String i : capitalCities.keySet()) {
  System.out.println(i);
}

// Print values
for (String i : capitalCities.values()) {
  System.out.println(i);
}

// Print keys and values
for (String i : capitalCities.keySet()) {
```

```
  System.out.println("key: " + i + " value: " + capitalCities.get(i));
}
```

La clase **TreeMap** es idéntica a HashMap con la salvedad que mantiene ordenado los datos por la clave.

Java HashSet

Un HashSet es una colección de elementos donde cada elemento es único y se encuentra en el paquete java.util:

```
Ejemplo
import java.util.HashSet; // Import the HashSet class

HashSet<String> cars = new HashSet<String>();
```

Agregar elementos

La clase HashSet tiene muchos métodos útiles. Por ejemplo, para agregarle elementos, use el método add ():

```
Ejemplo
import java.util.HashSet;

public class MyClass {
  public static void main(String[] args) {
    HashSet<String> cars = new HashSet<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("BMW");
    cars.add("Mazda");
    System.out.println(cars);
  }
}

Salida por Consola:
[Volvo, Mazda, Ford, BMW]
```

Nota: En el ejemplo anterior, aunque BMW se agrega dos veces, solo aparece una vez en el conjunto porque cada elemento de un conjunto debe ser único.

Comprobar si existe un artículo.

Para verificar si un elemento existe en un HashSet, use el método contains ():

cars.contains("Mazda");

TreeSet realiza todas las operaciones del HashSet. Además**,** mantiene todos los elementos ordenados en su orden natural o de acuerdo a como indique el Comparator que indique el constructor