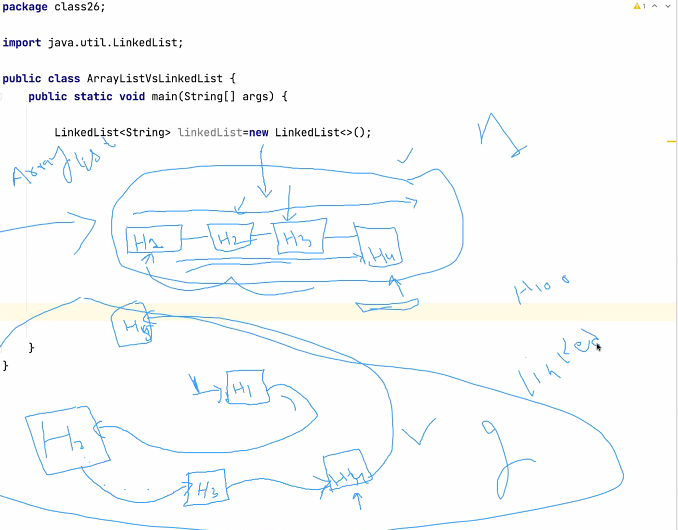
Array List Vs LinkedList



Notes

Collections Framework:

It's a collection of classes and interfaces that help us store and manipulate the data efficiently.

Why we should use Collections Framework?

1) It's flexible( we don't need to provide size beforehand).

2) It's dynamic(all the classes can increase and decrease their size at run time).

3) There are built-in methods to perform common tasks.

4) pre-built data structures.

downsides of Collections framework:

1)we can't use primitive data types we have to use wrapper classes which occupy 4 times more memory

than a primitive.

int num=10 // 4 memory locations

Integer num1=10; //16 memory locations

Wrapper:

Wrapper classes are object types for all the primitive types

why should we learn about the wrapper types?

1) Because collections framework only supports wrapper types.

2) There are built-in methods to perform common tasks.

Downsides:

1)we can't use primitive data types we have to use wrapper classes which occupy 4 times more memory

than a primitive.

int num=10 // 4 memory locations

Integer num1=10; //16 memory locations

Boxing:

Converting a primitive to its corresponding wrapper type by manually calling methods.

int pnum=10;

Integer wnum=new Integer(pnum);

AutoBoxing:

conversion from primitive to a wrapper type but here we don't need to call any methods explicitly.

int pnum=10;

Integer wnum=pnum;

Unboxing:

converting a wrapper to a primitive by calling a method.

Integer wnum=new Integer(10);

int pnum=wnum.intValue();

AutoUnboxing:

converting a wrapper to a primitive without calling any methods.

Integer wnum=new Integer(10);

int pnum=wnum;

ArrayList Class:

it is a classes in collections framework that allows us to store data like an array.

some common methods from this class are 1) add() 2) remove 3) contains 4) size().

Iterator:

its a mechanism that allows us to iterate and delete data from any class in collections framework.

it has hasNext() next() remove().

why we should learn about Iterator?

|  |  |
| --- | --- |
|  | Because loops are not safe when we perform an operation that can change the size of a data structure. |
|  | ArrayList Vs LinkedList |
|  | Both of the classes implement the list interface which means both of them will have same methods. |
|  | ArrayList: |
|  | uses an array behind the scenes to store the data. |
|  | we should use arrayList when we are retrieving the data most of the time and not inserting the data in the start or in-between |
|  | if we have to shift the data. |
|  | to the list. |
|  | LinkedList: |
|  | Linked-list does not store the data in consecutive memory locations that's why it is easier to insert and remove the data from anywhere |
|  | in a list. |
|  | if you are going to manipulate the data(shifting of elements) always use linked-list. |
|  |  |
|  | if most of the time you use add or remove method then use linkedList. |
|  | if most of the time you use get method use ArrayList. |
|  |  |
|  | Sets |
|  |  |

Create an array List of words.

package class26;

import java.util.ArrayList;

public class Task1 {

public static void main(String[] args) {

/\* Create an arrayList of words.

Remove every word that ends with “e”.

\*/

ArrayList<String> words=new ArrayList();

words.add("Java");

words.add("India");

words.add("Palestine");

words.add("USA");

words.add("France");

words.add("Pakistan");

words.add("Ukraine");

var iterator=words.iterator();

while (iterator.hasNext()){

var item=iterator.next();

if(item.endsWith("e")){

iterator.remove();

}}

System.out.println(words);

}}

create an arraylist of drinks

package class26;

import java.util.ArrayList;

public class Task2 {

public static void main(String[] args) {

/\*

create an arraylist of drinks.if any drink has letter "a" or "e" replace it with water.

\*/

ArrayList<String> drinks=new ArrayList<>();

drinks.add("Tea");

drinks.add("fresh Juice");

drinks.add("Iced Tea");

drinks.add("Soda");

drinks.add("canned Juice");

drinks.add("Milk");

drinks.add("Chocolate Milk");

for (int i = 0; i < drinks.size(); i++) {

if(drinks.get(i).contains("a")||drinks.get(i).contains("e")){

drinks.set(i,"Water"); //replace

}}

System.out.println(drinks);

}}

Create an arrayList of even numbers from 1 to 500.

package class26;

import java.util.ArrayList;

public class Task3 {

public static void main(String[] args) {

//Create an arrayList of even numbers from 1 to 500.

// Remove any number that is divisible by 5 from that arrayList.

ArrayList<Integer> numbers=new ArrayList<>();

for (int i = 2; i <=500 ; i=i+2) {

numbers.add(i);

}

System.out.println(numbers);

var it=numbers.iterator();

while (it.hasNext()){

Integer num=it.next();

if(num%5==0){

it.remove();

}

}

System.out.println(numbers);

}}

class Wrapper Demo

package class26;

public class WrapperDemo {

public static void main(String[] args) {

String num="10";

int number=Integer.parseInt(num);

System.out.println( Integer.MAX\_VALUE);

System.out.println( Double.MAX\_VALUE);

System.out.println( Long.MAX\_VALUE);

}}

class ArrayListVsLinkedList

package class26;

import java.util.ArrayList;

import java.util.LinkedList;

public class ArrayListVsLinkedList {

public static void main(String[] args) {

LinkedList<String> linkedList=new LinkedList<>();

long startTime = System.currentTimeMillis();

for (int i = 0; i < 1000000; i++) {

linkedList.add(0,"Test");

}

long endTime = System.currentTimeMillis();

System.out.println("LinkedList "+(endTime-startTime));

ArrayList<String> arrayList=new ArrayList<>();

long startTimeA = System.currentTimeMillis();

for (int i = 0; i < 1000000; i++) {

arrayList.add(0,"Test");

}

long endTimeA = System.currentTimeMillis();

System.out.println("ArrayList "+(endTimeA-startTimeA){

}}

ListDemo

package class26;

public class ListDemo {

public static void main(String[] args) {

}}

package class26;

import java.util.HashSet;

import java.util.LinkedHashSet;

import java.util.TreeSet;

public class TypesOfSets {

public static void main(String[] args) {

//There are three types of sets

/\*

HashSet:

DownSide=> it does not care about the insertion order

UpSide => It is the fastest type of Set

\*/

HashSet<String> hashSet=new HashSet<>();

hashSet.add("Mango");

hashSet.add("Apple");

hashSet.add("Mango");

hashSet.add("Kiwi");

hashSet.add("Orange");

hashSet.add("Banana");

System.out.println("HashSet");

System.out.println(hashSet);

//Whenever you need insertion order

LinkedHashSet<String> linkedHashSet=new LinkedHashSet<>();

linkedHashSet.add("Mango");

linkedHashSet.add("Apple");

linkedHashSet.add("Kiwi");

linkedHashSet.add("Mango");

linkedHashSet.add("Orange");

linkedHashSet.add("Banana");

System.out.println("linkedHashSet");

System.out.println(linkedHashSet);

// When ever you need sorted data use TreeSet

TreeSet<String> treeSet=new TreeSet<>();

treeSet.add("Mango");

treeSet.add("Apple");

treeSet.add("Kiwi");

treeSet.add("Mango");

treeSet.add("Orange");

treeSet.add("Banana");

System.out.println("TreeSet");

System.out.println(treeSet);

}}

package class26;

import java.util.ArrayList;

import java.util.HashSet;

import java.util.LinkedHashSet;

import java.util.LinkedList;

public class WhenToUseSets {

public static void main(String[] args) {

ArrayList<Integer> numbers=new ArrayList<>();

numbers.add(10);

numbers.add(20);

numbers.add(10);

numbers.add(40);

numbers.add(50);

numbers.add(50);

numbers.add(50);

numbers.add(50);

numbers.add(50);

System.out.println(numbers);

System.out.println(new LinkedHashSet<>(numbers));

HashSet<Integer> hashSet=new HashSet<>(numbers);

System.out.println(hashSet);

LinkedList<Integer> linkedList=new LinkedList<>(hashSet);

System.out.println(linkedList);

}}

package class26;

import java.util.ArrayList;

import java.util.LinkedHashSet;

public class WhySet {

public static void main(String[] args) {

ArrayList<Integer> numbers=new ArrayList<>();

numbers.add(10);

numbers.add(10);

numbers.add(10);

System.out.println(numbers);

LinkedHashSet<Integer> setNumbers=new LinkedHashSet<>();

setNumbers.add(10);

setNumbers.add(10);

setNumbers.add(10);

System.out.println(setNumbers);

}}