

به نام خدا



برنامه‌سازی پیشرفته

دانشگاه شهید بهشتی · دانشکده مهندسی و علوم کامپیوتر

دکتر مجتبی وحیدی اصل

نازنین زهرا فرهنگ

فهرست مطالب

- 1. در جاوا Collections .
- 2. چارچوب Collections .
- 3. پیمایشگر (Iterator) .
- 4. لیست .
- 5. ArrayList .
- 6. LinkedList .
- 7. Vector .

مقدمه: Collections در جاوا

- Collections در جاوا چارچوبی است که امکان ذخیره سازی و دستکاری مجموعه ای از اشیا را فراهم می کند.
- همه عملیاتی که می توانید ببروی داده ای انجام دهید: مانند جستجو (searching) ، مرتب سازی (sorting) ، دستکاری، اضافه نمودن داده جدید، حذف و غیره با استفاده از این چارچوب قابل انجام می باشد.
- جاوا واسطهای Collection (ArrayList, Vector, ..., Set, List, Queue, Deque) بسیاری نظیر (interface) و همچنین کلاسهایی نظیر (LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc) را در اختیار برنامه نویس گذاشته است.
- با collections می توانید داده ها را از یک ساختار ذخیره سازی (مانند ArrayList) به یک ساختار دیگر (مانند Array) انتقال دهید.

چیست؟ Collection

- یک Collection شیء است که حاوی گروهی از اشیا بوده و به این گروه در قالب یک واحد نگریسته می‌شود.
- انواع مختلفی از اشیا می‌توانند به عنوان عناصر collections ذخیره سازی، بازیابی و دستکاری شوند.

مزایای Collections در جاوا

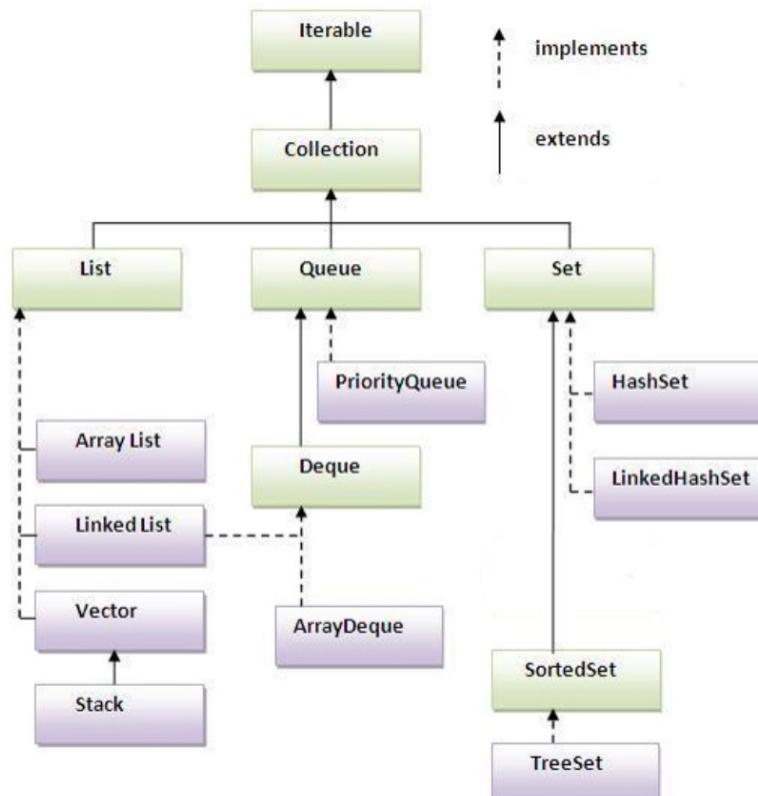
1. کلاس‌های Collection مانند ArrayList، HashSet، HashMap فراهم می‌کنند.
2. در مقایسه با آرایه‌ها که اندازه‌ی ثابتی دارند، بسیاری از کلاس‌های Collection مانند ArrayList یا HashSet به صورت پویا (Dynamic) رشد یا کاهش پیدا می‌کنند.
3. کلاس‌های Collection شامل متدهای مفیدی مثل sort(), reverse(), contains(), addAll(), removeAll() و ... هستند که برنامه‌نویس را از نوشتن کدهای تکراری بینیاز می‌کنند.
4. استفاده از Collection‌ها باعث می‌شود که شما منظم‌تر، خواناتر و آسان‌تر برای نگهداری و توسعه باشد.

چارچوب Collections

- یک collection را شیئی تصور کنید که ریموت کنترلهاي به عناصر (اشیایی دیگر) را نگهداری می کند.
- چارچوب collections جاوا بخشی از پکیج `java.util` را تشکیل می دهد.
- یک چارچوب collections از سه بخش اصلی تشکیل شده است:
 - یک واسط، عملیات و قواعدی برای یک نوع collection مشخص (مانند Queue, Set, List ...) تعریف می کند.
 - با دانستن متدهای واسط یک collection از عملکرد و خصوصیات آن آگاه می شوید.
 - شامل کلاسهاي است که واسطهای بالا را پیاده سازی کرده اند (برای مثال (...LinkedList, HashSet
- Sorting, index searching, replacing: متدهای پلی مورفیک سودمند برای ایجاد و دستکاری اشیایی از کلاسهاي بالا مانند ، Algorithms و غیره reversing

سلسله مراتب چارچوب Collections

پکیج `java.util` حاوی همه کلاسها و interface های لازم برای چارچوب collection می باشد.



Methods of Collection interface

1	public boolean add(Object element)	is used to insert an element in this collection.
2	public boolean addAll(Collection c)	is used to insert the specified collection elements in the invoking collection.
3	public boolean remove(Object element)	is used to delete an element from this collection.
4	public boolean removeAll(Collection c)	is used to delete all the elements of specified collection from the invoking collection.
5	public boolean retainAll(Collection c)	is used to delete all the elements of invoking collection except the specified collection.
6	public int size()	return the total number of elements in the collection.
7	public void clear()	removes the total no of element from the collection.
8	public boolean contains(Object element)	is used to search an element.
9	public boolean containsAll(Collection c)	is used to search the specified collection in this collection.
10	public Iterator iterator()	returns an iterator.
11	public Object[] toArray()	converts collection into array.
12	public boolean isEmpty()	checks if collection is empty.
13	public boolean equals(Object element)	matches two collection.
14	public int hashCode()	returns the hashcode number for collection.

Iterator Interface

تا قبل از جاوا ۵، امکان استفاده از `for-each` برای پیمایش وجود نداشت.

قبل از جاوا ۵، با کمک `Iterator` پیمایش روی `Collection`‌ها انجام می‌شد.

سه متد اصلی را تعریف کرده است:

`Object next()` : عنصر بعدی را در `Collection` برمی‌گرداند.

`boolean hasNext()` : بررسی می‌کند آیا عنصر دیگری برای پیمایش وجود دارد یا خیر.

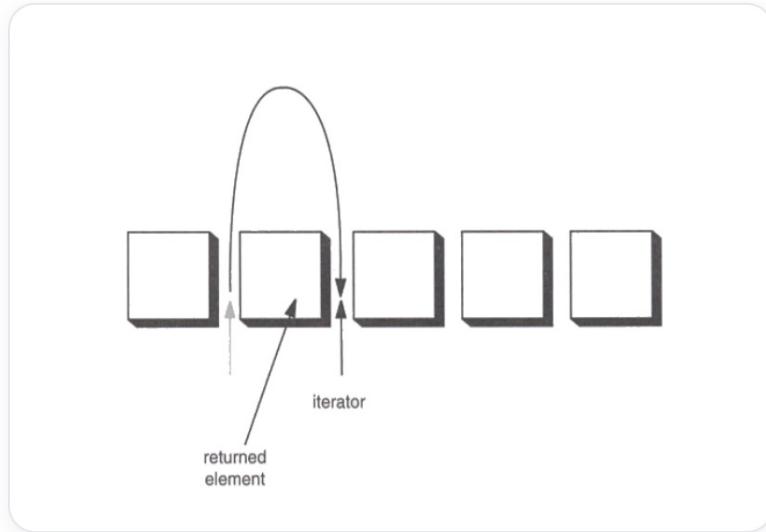
`void remove()` : آخرین عنصری را که توسط `next()` بازگردانده شده، از مجموعه حذف می‌کند.

این سه متد امکان دسترسی به عناصر ذخیره شده در `Collection`‌ها را فراهم می‌کنند.

یک `Iterator` محل عناصر را در یک `Collection` می‌داند.

هر بار فراخوانی `next()` باعث می‌شود عنصر بعدی در `Collection` خوانده شود.

می‌توانید از این عنصر استفاده کنید یا آن را حذف نمایید.



شیئی است که به شما امکان می‌دهد یک Collection را پیمایش کنید و با قرار گرفتن روی عنصر ذخیره شده در آن Collection، در صورت تمایل عنصر را حذف کنید. برای ایجاد شیئی از Iterator جهت پیمایش عناصر یک Collection، کافی است متدهای `iterator()` را روی شیء فراخوانی کنید.

واسطه Iterator به صورت زیر تعریف شده است:

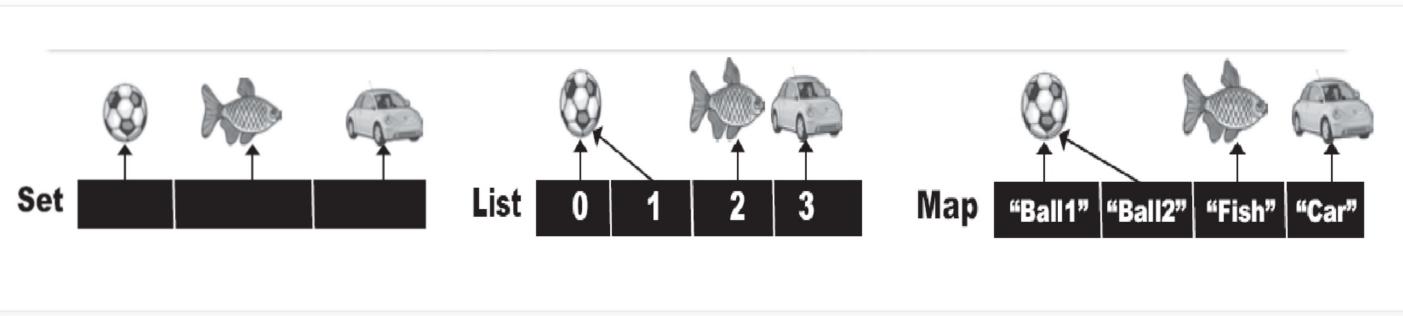
```
public interface Iterator<E> {
    boolean hasNext();
    E next();
    void remove(); //optional
}
```

مباحث مورد بررسی در چارچوب Collection

در این درس واسط ها و کلاس های زیر را بررسی می کنیم:

Set	List	Map
HashSet	ArrayList	HashMap
LinkedHashSet	LinkedList	LinkedHashMap
TreeSet	Vector	Hashtable
		TreeMap

عملیات اولیه در Collections

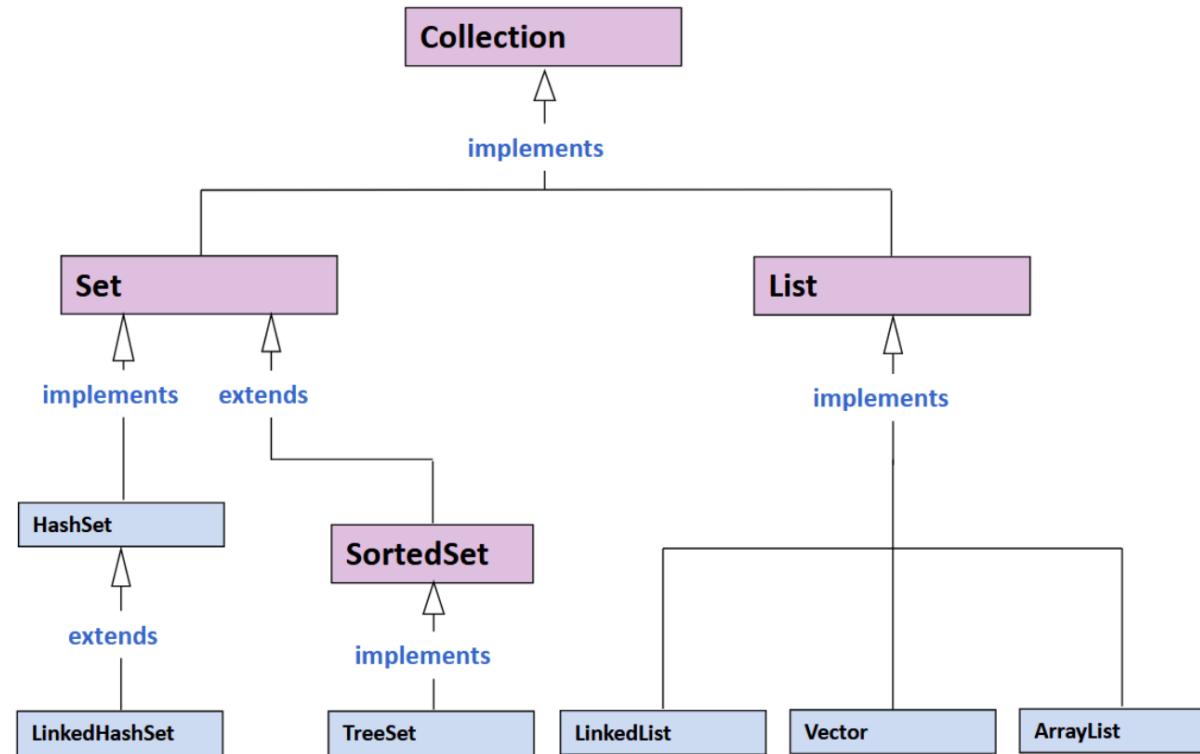


1. بررسی خالی بودن collection
 2. بررسی اینکه آیا شیئی در collection موجود است.
 3. بازیابی یک شیء از collection
 4. اضافه نمودن یک شیء به collection
 5. حذف یک شیء از collection
 6. پیمایش collection و بررسی هر شیء
- هریک از انواع collection هریک از عملیات بالا را در قالب یک متده منحصر به خود پیاده سازی کرده اند.

Collections خصوصیات

- دارای ترتیب مشخص (Ordered) است یا خیر
 - عناصر برخی انواع collections ، اغلب در یک ترتیب مشخص، ذخیره سازی و دستیابی می شوند.
 - عناصر برخی انواع دیگر مانند Hashtable از این قاعده تبعیت نمی کند.
- دارای اندیس است (Indexed) یا خیر
 - عناصر با استفاده از اندیس قابل دسترسی هستند یا دسترسی بدون اندیس انجام می شود.
- عناصر منحصر بفرد (Unique) هستند یا خیر
 - آیا تکرار عناصر در collection مجاز است؟

سلسله مراتب Collections



محدودیت آرایه‌ها

- اگر طول موردنیاز آرایه (size) را پیش‌اپیش ندانیم، چه کنیم؟
- اگر بخواهیم بعد از ساختن یک آرایه، طول آن را افزایش دهیم چه کنیم؟
- اگر بخواهیم بعضی از عناصر و اعضای آرایه را حذف کنیم، چه کنیم؟

: نکته

راه حل ساده‌ای برای موارد فوق در آرایه‌ها وجود ندارد. مثلًاً متدى که یک خانه از آرایه را حذف کند یا طول آرایه را بیشتر کند.

لیست چیست؟

value	(“Paul”)	(“Mark”)	(“John”)	(“Paul”)	(“Luke”)
index	0		1		2		3		4						

در یک لیست، اندیس اشیای ذخیره شده اهمیت دارد!

در نتیجه حاوی متدهایی است که اندیس عناصر را در هنگام اضافه نمودن، حذف و بازیابی آنها در نظر می گیرند.

ArrayList

Vector

LinkedList

واسط list در جاوا

جاوا واسطی به نام **java.util.List** دارد که بیانگر لیستی از اشیا می باشد. این واسط، متدهای زیر را به متدهای موجود در Collection اضافه می کند:

public void add(int index, Object o)

- در مکان مشخص شده عنصر مشخصی را به لیست اضافه می کند.

public Object get(int index)

- عنصری را در مکان مشخص شده در لیست برمی گرداند.

public int indexOf(Object o)

- اندیس اولین رخداد یک عنصر مشخص را در لیست برمی گرداند و در صورت عدم وجود عنصر، مقدار 1- را بر می گرداند.

public int lastIndexOf(Object o)

- اندیس آخرین رخداد یک عنصر مشخص را در لیست برمی گرداند و در صورت عدم وجود عنصر، مقدار 1- را بر می گرداند.

public Object remove(int index)

- عنصری را در مکان مشخص شده در لیست حذف می کند.

public Object set(int index, Object o)

- عنصری را در مکان مشخص شده در لیست را با عنصر مشخص شده تعویض می کند.

```
import java.util.ArrayList;

public class MyArrayList {

    public static void main(String args[ ]) {

        ArrayList alist = new ArrayList( );

        alist.add(new String("One"));
        alist.add(new String("Two"));
        alist.add(new String("Three"));

        System.out.println(alist.get(0));
        System.out.println(alist.get(1));
        System.out.println(alist.get(2));
    }
}
```

خروجی

One •

Two •

Three •

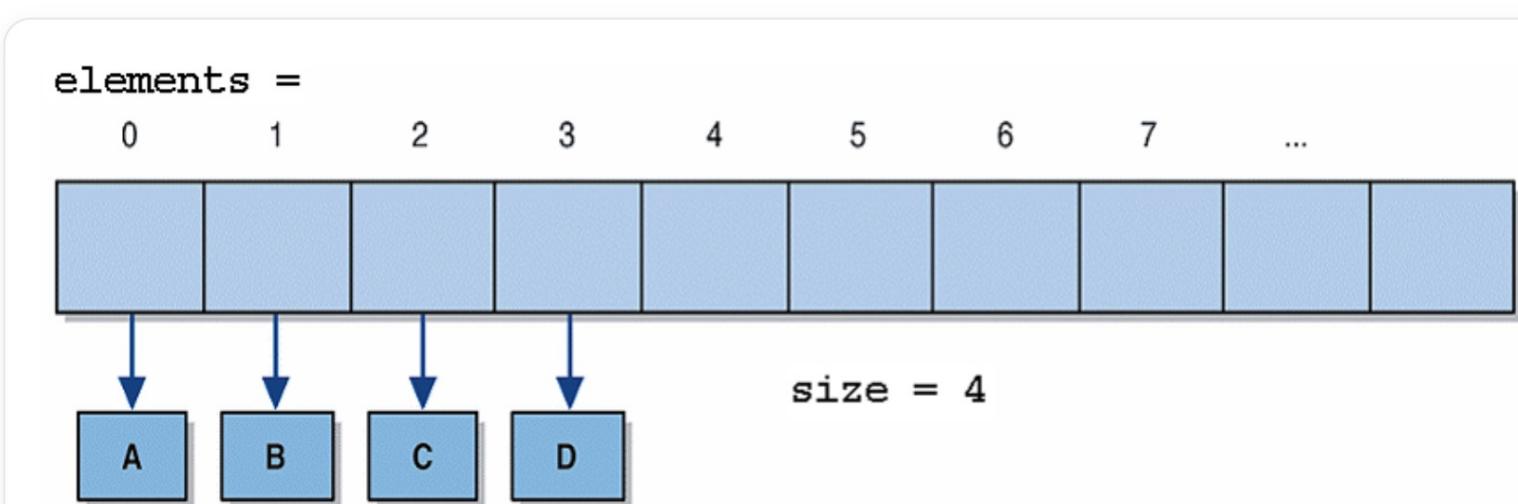
پرسش هایی درباره List

- بسیاری از عملیات اسلایدهای قبلی را می توان با استفاده از آرایه به جای لیست انجام داد.
- پرسش کلیدی: به چه دلیلی باید از List به جای آرایه برای ذخیره سازی داده هایمان استفاده کنیم؟
- یک لیست چگونه پیاده سازی می شود؟
- چرا همه متدهای List از نوع Object استفاده می کنند؟

ArrayList

لیستی که از یک آرایه داخلی برای ذخیره سازی داده ها استفاده می کند:

- encapsulates array and # of elements (size)
- in Java: `java.util.ArrayList`
- when you want to use ArrayList, remember to import `java.util.*;`



ArrayList خصوصیات

- در حقیقت یک آرایه دارای قابلیت تغییراندازه خودکار می باشد که می تواند هرنوع شیئی را با استفاده از متدهای تعریف شده اش نگهداری و دستکاری کند.
- در ابتدا، ArrayList خالی است، به مرور می توانیم عناصری به این فهرست اضافه یا کم کنیم.
- ArrayList بیشتر مزایای یک آرایه معمولی را دارد، به ویژه دسترسی تصادفی!
- برنامه نویس دغدغه کارهایی مانند شیفت عناصر یا تغییراندازه آرایه داخلی را ندارد. اندازه اولیه ArrayList نیاز نیست از ابتدا مشخص شود!
- لیستها، می توانند نوع اشیاء درون خود را مشخص کنند.
- در کد زیر، به ظرف students فقط اشیائی از نوع Student می توان اضافه کرد:
`ArrayList<Student> students = new ArrayList<Student>();`
- با فراخوانی `toString` برروی یک ArrayList عناصر آن در قالب یک لیست برگردانده می شوند.

[1, 2.65, Marty Stepp, Hello]

کلاس Collections برروی لیست

- کلاس **java.util.Collections** تعداد زیادی متده استاتیک کارآمد دارد که برروی اشیای collections (نظیر لیستها) اعمال می شود.
- این کلاس utility، واسط Collection را پیاده سازی کرده است.

public static void copy(List dest, List src)

- متده **copy** در کلاس Collections محتوای یک لیست را در لیست دیگر کپی می کند. بنابراین این کلاس مستقل از اشیای ساخته شده از collection می باشد.

public static void fill(List list, Object value)

- عنصری را در مکان مشخص شده در لیست برمی گرداند.

public static Object max(Collection coll)

- اندیس اولین رخداد یک عنصر مشخص را در لیست برمی گرداند و در صورت عدم وجود عنصر، مقدار 1- را برمی گرداند.

public static Object min(Collection coll)

- اندیس آخرین رخداد یک عنصر مشخص را در لیست برمی گرداند و در صورت عدم وجود عنصر، مقدار 1- را برمی گرداند.

public static void reverse(List list)

- عنصری را در مکان مشخص شده در لیست حذف می کند.

public static void shuffle(List list)

- عنصری را در مکان مشخص شده در لیست را با عنصر مشخص شده تعویض می کند.

public static void sort(List list)

- در مکان مشخص شده عنصر مشخص را به لیست اضافه می کند.

public static void swap(List list, int i, int j)

- عنصری را در مکان مشخص شده در لیست برمی گرداند.

```
import java.util.ArrayList;
import java.util.Collections;

public class MyArrayList {

    public static void main(String args[ ]) {

        ArrayList myArrayList = new ArrayList( );

        myArrayList.add(7);
        myArrayList.add(4);
        myArrayList.add(3);

        Collections.sort(myArrayList);
        System.out.println(myArrayList.toString());
    }
}
```

خروجی

[3 ,4 ,7] •

چگونه از ArrayList یک زیرلیست بگیریم؟

اگر بخواهیم از ArrayList مان یک زیرلیست استخراج کنیم از متدهای subList در کلاس ArrayList استفاده می‌کنیم:

`List subList(int fromIndex, int toIndex)`

این متدهای زیرلیستی از اندیس fromIndex تا به اندیس toIndex (اندیس toIndex را شامل نمی‌شود) لیست فراخواننده این متدهای استخراج می‌کند.

مثال

```
import java.util.ArrayList;
import java.util.List;

public class SublistExample {
    public static void main(String[] args) {
        ArrayList<String> al = new ArrayList<String>();

        //Addition of elements in ArrayList
        al.add("Steve");
        al.add("Justin");
        al.add("Ajeet");
        al.add("John");
        al.add("Arnold");
        al.add("Chaitanya");

        System.out.println("Original ArrayList Content: " + al);

        //Sublist to ArrayList
        ArrayList<String> al2 = new ArrayList<String>(al.subList(1, 4));
        System.out.println("Sublist stored in ArrayList: " + al2);

        //Sublist to List
        List<String> list = al.subList(1, 4);
        System.out.println("Sublist stored in List: " + list);
    }
}
```

خروجی

Original ArrayList Content: [Steve, Justin, Ajeet, John, Arnold, Chaitanya] •
Sublist stored in ArrayList: [Justin, Ajeet, John] •
Sublist stored in List: [Justin, Ajeet, John] •

استثناهای ایجاد شده در `subList`

- متد `subList` استثنای `IndexOutOfBoundsException` را پرتاب می کند اگر اندیسهای مشخص شده در محدوده اندیسهای `ArrayList` نباشند (`fromIndex < 0 || toIndex > size`)
- اگر اندیس شروع بزرگتر از اندیس پایانی متد باشد (`fromIndex>toIndex`), استثنای `IllegalArgumentException` پرتاب می شود.

اتصال(ترکیب) دو ArrayList

```
import java.util.ArrayList;
public class Details
{
    public static void main(String [] args)
    {
        //First ArrayList
        ArrayList<String> arraylist1 = new ArrayList<String>();
        arraylist1.add("AL1: E1");
        arraylist1.add("AL1: E2");
        arraylist1.add("AL1: E3");

        //Second ArrayList
        ArrayList<String> arraylist2 = new ArrayList<String>();
        arraylist2.add("AL2: E1");
        arraylist2.add("AL2: E2");
        arraylist2.add("AL2: E3");

        //New ArrayList
        ArrayList<String> al = new ArrayList<String>();
        al.addAll(arraylist1);
        al.addAll(arraylist2);

        //Displaying elements of the joined ArrayList
        for(String temp : al){
            System.out.println(temp);
        }
    }
}
```

در مثال زیر، دو ArrayList در قالب یک ArrayList جدید به هم متصل می شوند:

- از متد **addAll()** استفاده کنید.

خروجی

AL1: E1 •
AL1: E2 •
AL1: E3 •
AL2: E1 •
AL2: E2 •
AL2: E3 •

روش های مختلف دسترسی به عناصر ArrayList

```
import java.util.*;
public class LoopExample {

    public static void main(String[] args) {

        ArrayList<Integer> arrlist = new ArrayList<Integer>();
        arrlist.add(14);
        arrlist.add(7);
        arrlist.add(39);
        arrlist.add(40);

        /* For Loop for iterating ArrayList */
        System.out.println("For Loop");
        for (int counter = 0; counter < arrlist.size(); counter++) {
            System.out.println(arrlist.get(counter));
        }
        /* Advanced For Loop */
        System.out.println("Advanced For Loop");
        for (Integer num : arrlist) {
            System.out.println(num);
        }
    }
}
```

```
/* While Loop for iterating ArrayList */
System.out.println("While Loop");
int count = 0;
while (arrlist.size() > count) {
    System.out.println(arrlist.get(count));
    count++;
}

/* Looping ArrayList using Iterator */
System.out.println("Iterator");
Iterator iter = arrlist.iterator();
while (iter.hasNext()) {
    System.out.println(iter.next());
}
}
```

خروجی

While Loop •

14 •

7 •

39 •

40 •

Iterator •

14 •

7 •

39 •

40 •

For Loop •

14 •

7 •

39 •

40 •

Advanced For Loop •

14 •

7 •

39 •

40 •

دسترسی به عناصر ArrayList با واسط Enumeration

```
import java.util.Enumeration;
import java.util.ArrayList;
import java.util.Collections;

public class EnumExample {

    public static void main(String[] args) {
        //create an ArrayList object
        ArrayList<String> arrayList = new ArrayList<String>();

        //Add elements to ArrayList
        arrayList.add("C");
        arrayList.add("C++");
        arrayList.add("Java");
        arrayList.add("DotNet");
        arrayList.add("Perl");

        // Get the Enumeration object
        Enumeration<String> e = Collections.enumeration(arrayList);
        // Enumerate through the ArrayList elements
        System.out.println("ArrayList elements: ");
        while (e.hasMoreElements())
            System.out.println(e.nextElement());
    }
}
```

خروجی

- ArrayList elements:
- C
- C++
- Java
- DotNet
- Perl

تبديل یک آرایه به ArrayList - روش اول

تبديل با استفاده از `Arrays.asList()`

سینتکس

```
ArrayList<T> arraylist = new ArrayList<T>(Arrays.asList(arrayname));
```

```
import java.util.*;

public class ArrayToArrayList {
    public static void main(String[] args) {

        /* Array Declaration and initialization */
        String cityNames[] = {"Agra", "Mysore", "Chandigarh", "Bhopal"};

        /* Array to ArrayList conversion */
        ArrayList<String> cityList = new ArrayList<String>(Arrays.asList(cityNames));

        /* Adding new elements to the converted List */
        cityList.add("New City2");
        cityList.add("New City3");

        /* Final ArrayList content display using for */
        for (String str: cityList)
        {
            System.out.println(str);
        }
    }
}
```

خروجی

Agra •

Mysore •

Chandigarh •

Bhopal •

New City2 •

New City3 •

تبديل يك آرایه به ArrayList - روش دوم

- استفاده از `addAll()` - سريعتر از روش قبلی می باشد.

سينتكس

```
String array[]={new Item(1), new Item(2), new Item(3), new Item(4)}; •  
ArrayList arraylist = new ArrayList();  
Collections.addAll(arraylist, array);  
  
Collections.addAll(arraylist, new Item(1), new Item(2), new Item(3), new Item(4)); •
```

```
import java.util.*;

public class Example2 {
    public static void main(String[] args) {

        /* Array Declaration and initialization */
        String array[] = {"Hi", "Hello", "Howdy", "Bye"};

        /* ArrayList declaration */
        ArrayList<String> arrayList = new ArrayList<String>();

        /* Conversion */
        Collections.addAll(arrayList, array);

        /* Adding new elements to the converted List */
        arrayList.add("String1");
        arrayList.add("String2");

        /* Display array list */
        for (String str : arrayList) {
            System.out.println(str);
        }
    }
}
```

خروجی

Hi •
Hello •
Howdy •
Bye •
String1 •
String2 •

تبديل یک آرایه به ArrayList - روش سوم(دستی)

```
import java.util.*;  
  
public class Details {  
    public static void main(String[] args) {  
  
        /* ArrayList declaration */  
        ArrayList<String> arraylist = new ArrayList<String>();  
  
        /* Initialized Array */  
        String array[] = {"Text1", "Text2", "Text3", "Text4"};  
  
        /* array.length returns the current number of  
         * elements present in array */  
        for (int i = 0; i < array.length; i++) {  
            /* We are adding each array's element to the ArrayList */  
            arraylist.add(array[i]);  
        }  
  
        /* ArrayList content */  
        for (String str : arraylist) {  
            System.out.println(str);  
        }  
    }  
}
```

خروجی

Text1 •

Text2 •

Text3 •

Text4 •

تبديل یک ArrayList به آرایه - روش اول

- استفاده از متد `()toArray`

```
import java.util.*;  
  
public class Example {  
    public static void main(String[] args) {  
  
        /* ArrayList declaration and initialization */  
        ArrayList<String> friendsNames = new ArrayList<String>();  
        friendsNames.add("Ankur");  
        friendsNames.add("Ajeet");  
        friendsNames.add("Harsh");  
        friendsNames.add("John");  
  
        /* ArrayList to Array Conversion */  
        String frNames[] = friendsNames.toArray(new String[friendsNames.size()]);  
  
        /* Displaying Array elements */  
        for (String k : frNames) {  
            System.out.println(k);  
        }  
    }  
}
```

خروجی

Ankur •
Ajeet •
Harsh •
John •

تبديل یک ArrayList به آرایه - روش دوم (دستی)

```
import java.util.*;  
  
public class ArrayListToArray {  
    public static void main(String[] args) {  
  
        /* ArrayList declaration and initialization */  
        ArrayList<String> arrList = new ArrayList<String>();  
        arrList.add("String1");  
        arrList.add("String2");  
        arrList.add("String3");  
        arrList.add("String4");  
  
        /* ArrayList to Array Conversion */  
        String array[] = new String[arrList.size()];  
        for (int j = 0; j < arrList.size(); j++) {  
            array[j] = arrList.get(j);  
        }  
  
        /* Displaying Array elements */  
        for (String k : array) {  
            System.out.println(k);  
        }  
    }  
}
```

خروجی

String1 •
String2 •
String3 •
String4 •

سنکرون سازی `ArrayList`

- آسنکرون بوده و در محیطی چندتریدی می‌تواند همزمان توسط چندین ترید مورد دستیابی و تغییر قرار گیرد.
- اگر نخواهیم در یک لحظه دسترسی به آن توسط چند ترید صورت گیرد، باید آن را سنکرون سازی کنیم تا در یک لحظه میان چندین ترید به اشتراک گذاشته نشود.
- سنکرون سازی آشکار، به دو روش صورت می‌گیرد:
 - استفاده از متدهای `Collections.synchronizedList()`
 - استفاده از نسخه `CopyOnWriteArrayList` لیست: `thread-safe`
- در هنگام استفاده از `iterator` برروی یک شیء `ArrayList` بهتر است آن را در بلاک سنکرون قرار دهیم.

روش اول

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Collections;

public class Details {

    public static void main(String a[]){
        List<String> syncal = Collections.synchronizedList(new ArrayList<String>());
        //Adding elements to synchronized ArrayList
        syncal.add("Pen");
        syncal.add("NoteBook");
        syncal.add("Ink");

        System.out.println("Iterating synchronized ArrayList:");
        synchronized(syncal) {
            Iterator<String> iterator = syncal.iterator();
            while (iterator.hasNext())
                System.out.println(iterator.next());
        }
    }
}
```

خروجی

Iterating synchronized ArrayList:
• Pen
• NoteBook
• Ink

روش دوم

```
import java.util.Iterator;
import java.util.concurrent.CopyOnWriteArrayList;

public class Details {

    public static void main(String a[]) {
        CopyOnWriteArrayList<String> al = new CopyOnWriteArrayList<String>();

        //Adding elements to synchronized ArrayList
        al.add("Pen");
        al.add("NoteBook");
        al.add("Ink");

        System.out.println("Displaying synchronized ArrayList Elements:");

        //Synchronized block is not required in this method
        Iterator<String> iterator = al.iterator();
        while (iterator.hasNext())
            System.out.println(iterator.next());
    }
}
```

خروجی

Displaying synchronized ArrayList Elements:

- Pen
- NoteBook
- Ink

تحلیل زمان اجرای ArrayList

OPERATION	RUNTIME (Big-Oh)
add to start of list	O(n)
add to end of list	O(1) amortized
add at given index	O(n)
clear	O(n)
get	O(1)
find index of an object	O(n)
remove first element	O(n)
remove last element	O(1)
remove at given index	O(n)
set	O(1)
size	O(1)
toString	O(n)

ArrayList

- اضافه کردن به ArrayList حاوی تعداد زیادی عنصر، سربار زمانی دارد.
- اضافه کردن به ابتدای لیست منجر به شیفت همه عناصر دیگر می شود.
- حذف یک عنصر نیز پر هزینه بوده و به شیفت همه عناصر نیاز دارد.
- در بیشتر دستورات باید اندیس عنصر را بدانیم.
- عناصر بهم چسبیده هستند و دستکاری آنها زمانبر است.
- آیا می توانیم از ساختار ذخیره سازی انعطاف پذیرتری استفاده کنیم؟

32	17	5	24	-3
----	----	---	----	----

32	17	5	24	-3
----	----	---	----	----

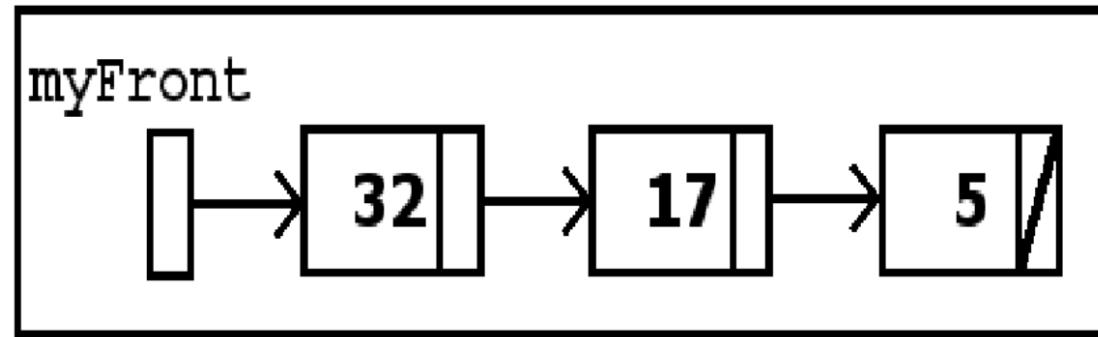
یک راه حل: لیستی از عناصر به هم زنجیره شده

- یک شیئی از کلاسی به نام Node ایجاد کنیم که یک مکان ذخیره سازی برای یک عنصر لیست را فراهم کند.
- هر گره لیست ریموت کنترلی به گره بعد از خود را نگهداری کند (که به این ریموت کنترل، next می گوییم)
- آخرین node دارای next == null می باشد (که در شکل به صورت / نمایش داده می شود)

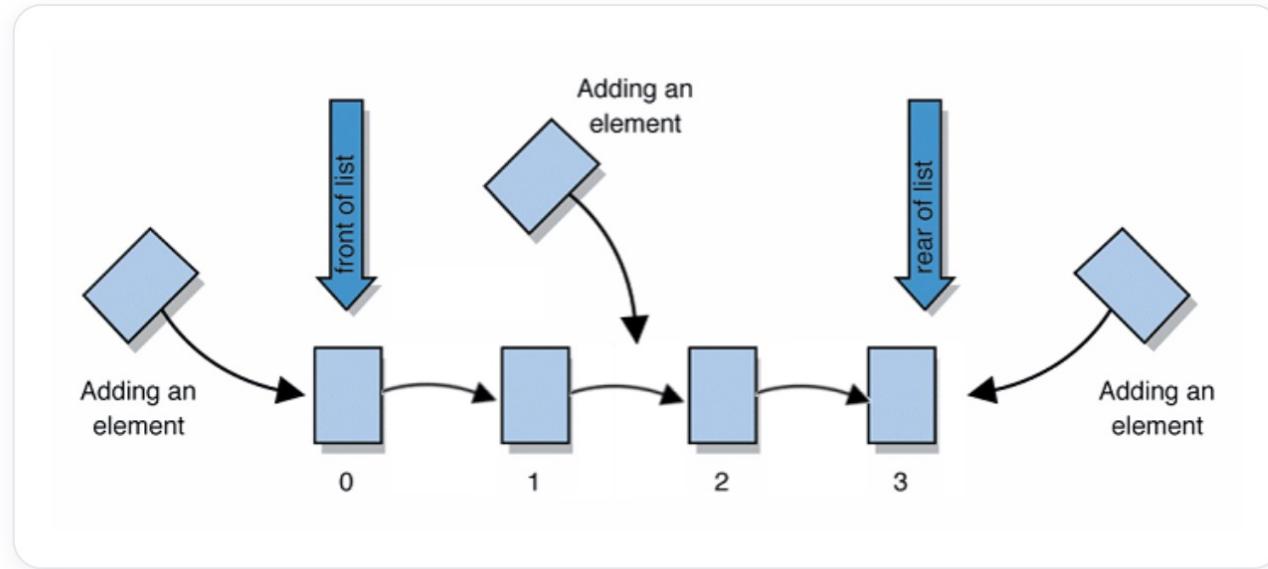


لیست پیوندی (Linked List)

- لیست پیوندی به collection ای گفته می شود که عملیات لیست را برروی دنباله ای زنجیر شده از گره ها پیاده سازی می کند.
- در آن، برخلاف آرایه، همه اعضا پشت سرهم در حافظه قرار نمی گیرند.
- برای اضافه کردن یک عضو به فهرست:
یک شیء جدید ایجاد می شود و آخرین ارجاع (اشاره گر) به این شیء جدید اشاره خواهد کرد.
- برای حذف یک عضو از فهرست: کافیست اشاره گر به این شیء، به شیء بعدی اشاره کند
- برای یک لیست پیوندی کافیست ریموت کنترلی به اولین گره را نگهداری کنیم (در شکل این ریموت را myFront نامگذاری کرده ایم)
- دسترسی به سایر گره ها از گره قبلی آن میسر خواهد شد.

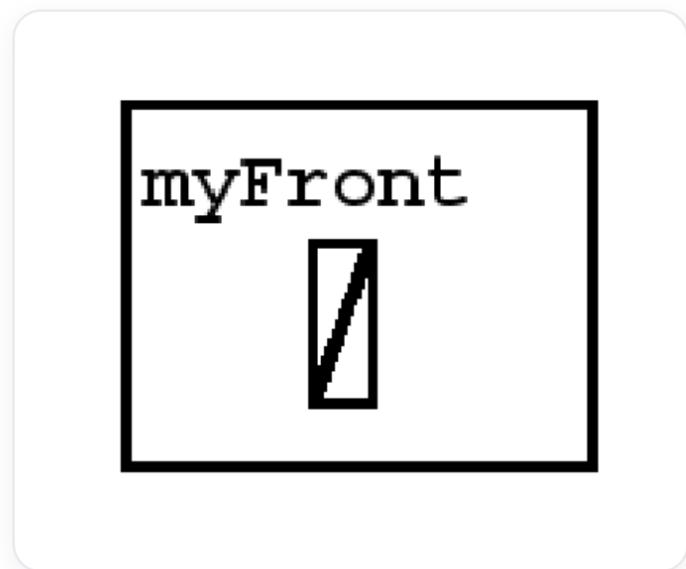


• کلاس `java.util.LinkedList` اشیایی از یک لیست پیوندی را با پیاده سازی داخلی خود، ایجاد می کند.

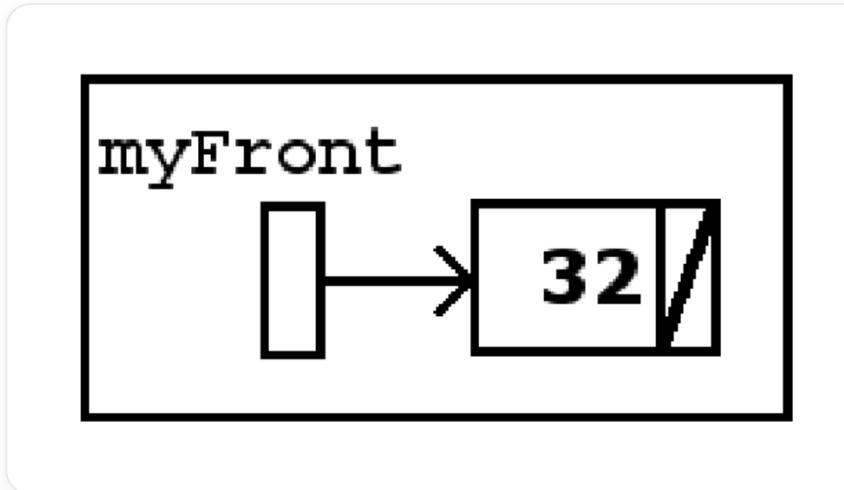


حالت های مختلف لیست پیوندی

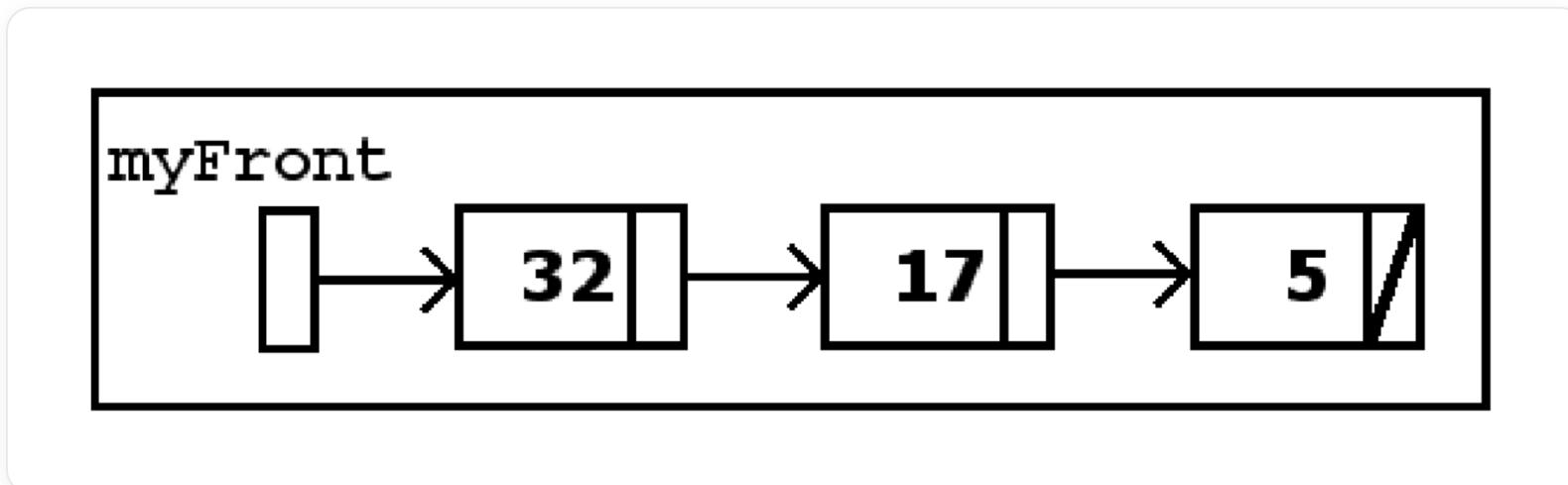
- لیست خالی ($\text{myFront} == \text{null}$)



- لیستی با یک عنصر (list with one element)

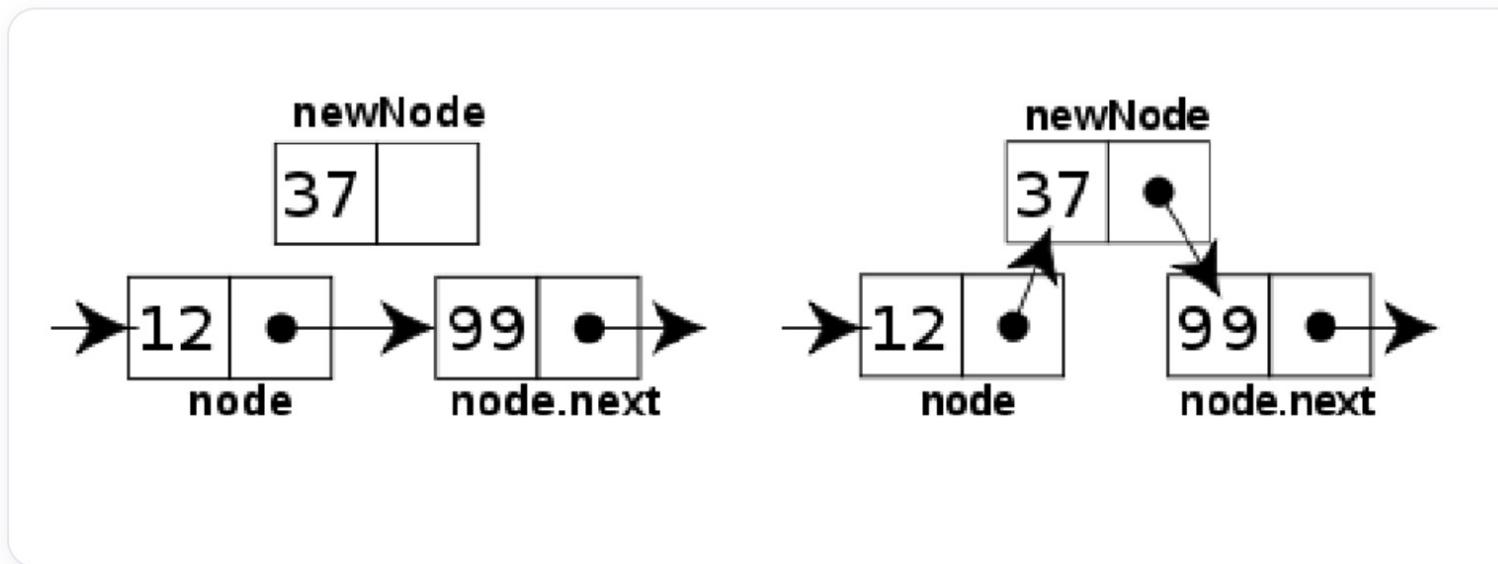


- لیستی با چندین عنصر (list with many elements)

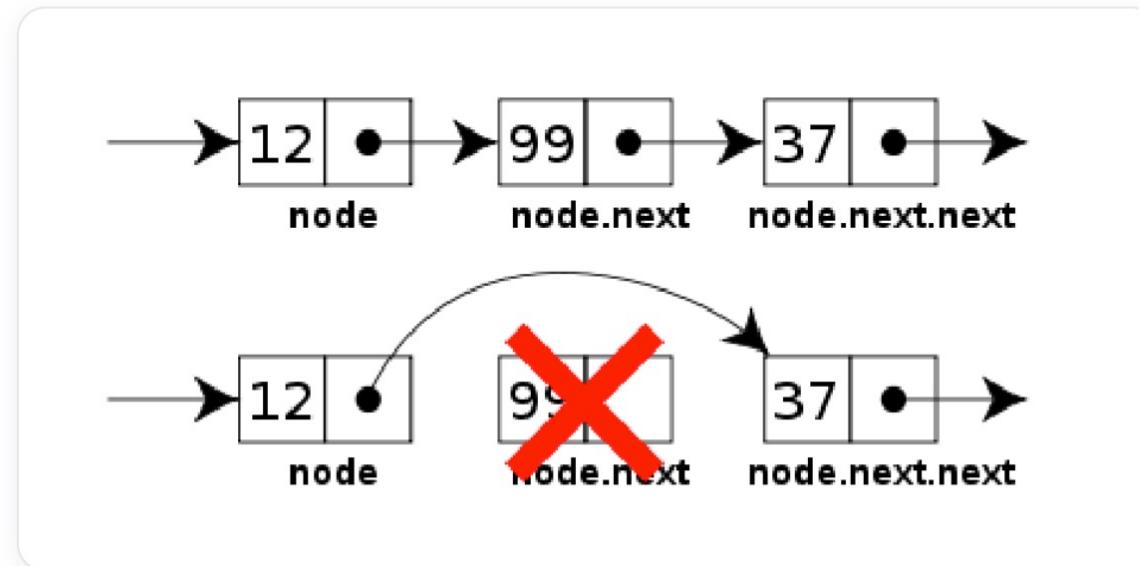


حذف و اضافه به لیست پیوندی

- اضافه به لیست:



- حذف از لیست:



یک لیست پیوندی چگونه پیاده سازی می شود؟

- an add operation (at the front, back, and middle)
- a remove operation
- a get operation
- a set operation
- an indexof (searching) operation

پیاده سازی کلاس Node

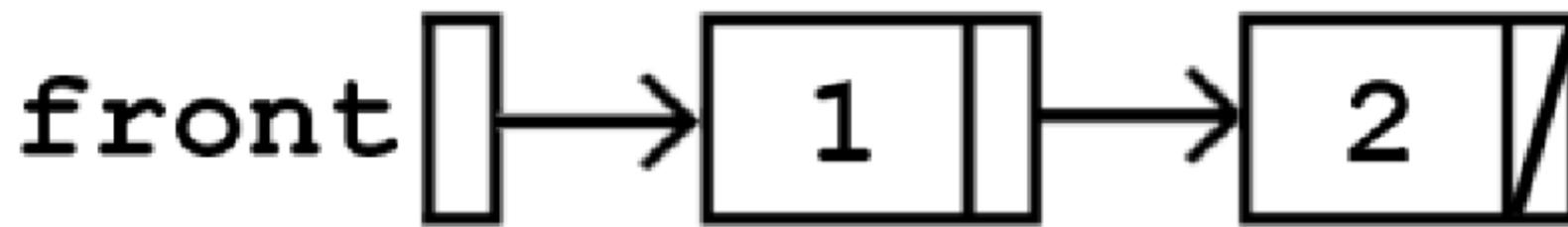
```
/* Stores one element of a linked list. */
public class Node {
    public Object element;
    public Node next;

    public Node(Object element) {
        this(element, null);
    }

    public Node(Object element, Node next) {
        this.element = element;
        this.next = next;
    }
}
```

مثال های Linked Node

Initialization: •



```
front.next = new Node(new Integer(3), front.next);
```



```
front = new Node(new Integer(3), front);
```



```
front.next.next = new Node(new Integer(3));
```



تحلیل زمان اجرای LinkedList

OPERATION	RUNTIME (Big-Oh)
add to start of list	O(1)
add to end of list	O(1)
add at given index	O(n)
clear	O(n)
get	O(n)
find index of an object	O(n)
remove first element	O(1)
remove last element	O(1)
remove at given index	O(n)
set	O(n)
size	O(1)
toString	O(n)

متد های لیست پیوندی

`public void addFirst(int value)`
inserts given value at front

`public boolean isEmpty()`
returns true if no nodes are in list

`public int size()`
returns number of elements

`public String toString()`
returns string representation of list

`public void addLast(int value)`
appends given val at end

`public void add(int index, int value)`
inserts given value at given index

`public int get(int index)`
returns value at given index (exception when index is OOB)

`public int set(int index, int value)`
sets element at given index to have the given value, and returns it (exception when index is OOB)

public boolean indexOf(int value)
returns index of value in list (-1 if value is not in the list)

public boolean contains(int value)
returns true if value is in list

public void clear()
removes all values from list

public int removeFirst()
removes and returns front value (exception when list is empty)

public int removeLast()
removes and returns rear value (exception when list is empty)

public void remove(int index)
removes and returns value at index (exception when index is OOB)

```
import java.util.*;

public class TestLinkedList {
    public static void main(String args[]){

        LinkedList<String> al = new LinkedList<String>();
        al.add("Ravi");
        al.add("Vijay");
        al.add("Ravi");
        al.add("Ajay");

        Iterator<String> itr = al.iterator();
        while(itr.hasNext()) {
            System.out.println(itr.next());
        }
    }
}
```

خروجی

Ravi •

Vijay •

Ravi •

Ajay •

خصوصیات لیست های پیوندی

- قادرند عناصر تکراری را نگهداری کنند.
- به دلیل ساختار زنجیره ای، از فضاهای خالی حافظه heap به صورت بهینه استفاده می کنند.
- دسترسی تصادفی به یک عنصر در آنها امکانپذیر نیست.
 - پیمایش باید از نود اولیه آغاز شود تا آدرس نود بعدی مشخص شود و در نود بعدی به دنبال آدرس نود سوم برود تا به عنصر مورد نظر برسد.
 - اگر ریموت کنترلی به نود آخر وجود داشته باشد، دسترسی به آخرین گره به طور آنی انجام می شود و نیاز به پیمایش کل لیست نخواهد بود.
 - در لیستهای پیوندی دوطرفه (doubly) پیمایش دوطرفه از ابتدا یا انتهای لیست به عناصر میانی امکان پذیر می شود.
- به دلیل نیاز به پیمایش غیرتصادفی، زمان جستجو در آن پایین نیست.
- دو متدهای contains و indexOf کند هستند.
- دستکاری (مانند حذف و اضافه یک عنصر) در آن سریع است چون به شیفت نیاز ندارد.
- برای پیاده سازی پشته ها، صفحه ها، درختها و گرافها استفاده می شود.

مقایسه LinkedList و ArrayList

جستجو: عملیات جستجو در `ArrayList` دارای پیچیدگی زمانی $O(n)$ است، در حالی که همین عملیات در `LinkedList` دارای پیچیدگی $O(1)$ می‌باشد.

دلیل: `ArrayList` مبتنی بر اندیس است و عملکردی مشابه آرایه‌های سنتی دارد. در مقابل، `LinkedList` (از نوع doubly linked list) برای دسترسی به یک عنصر نیازمند پیمایش عناصر از ابتدا یا انتهای آن می‌باشد.

حذف: عملیات حذف در `LinkedList` (در صورت صرفنظر از زمان رسیدن به اندیس موردنظر) دارای پیچیدگی زمانی $O(1)$ است. در حالی که حذف در `ArrayList` در بدترین حالت (حذف عنصر اول) دارای پیچیدگی $O(n)$ و در بهترین حالت (حذف عنصر آخر) دارای پیچیدگی $O(1)$ می‌باشد. توجه داشته باشید اگر زمان پیمایش `LinkedList` برای رسیدن به اندیس موردنظر در نظر گرفته شود، زمان حذف افزایش می‌یابد و برتری زمانی آن نسبت به `ArrayList` ممکن است چندان محسوس نباشد.

اما در سناریوهایی که حذف عناصر به صورت متوالی از ابتدای `LinkedList` انجام می‌شود، به دلیل عدم نیاز به عملیات شیفت، عملکرد `LinkedList` به مراتب بهتر از `ArrayList` خواهد بود.

نتیجه‌گیری: در حالت متوسط، عملیات حذف در `LinkedList` اغلب سریع‌تر از `ArrayList` است.

دلیل: در `LinkedList` هر گره دارای دو اشاره‌گر به عناصر مجاور است و حذف تنها با تغییر این اشاره‌گرها انجام می‌شود. در مقابل، در `ArrayList` پس از حذف یک عنصر، سایر عناصر باید برای پر کردن فضای خالی شیفت داده شوند.

درج: عملیات درج (`insert`) در `LinkedList` (با صرفنظر از زمان پیمایش) دارای پیچیدگی $O(1)$ است، در حالی که در `ArrayList` در بدترین حالت پیچیدگی زمانی $O(n)$ دارد.

دلیل: منطق درج مشابه عملیات حذف است؛ در `LinkedList` تنها اشاره‌گرها تغییر می‌کنند، اما در `ArrayList` نیاز به جابه‌جایی عناصر وجود دارد.

سربار حافظه: `ArrayList` تنها عناصر داده‌ای را به صورت پشت‌سرهم ذخیره می‌کند، اما `LinkedList` علاوه بر داده‌ها، برای هر گره دو اشاره‌گر (به گره قبلی و بعدی) نگهداری می‌کند. به همین دلیل، `ArrayList` نسبت به `LinkedList` سربار حافظه‌ای بیشتری دارد.

شباهت‌های میان دو کلاس `ArrayList` و `LinkedList`

هر دو کلاس، واسط `List` را پیاده‌سازی کرده‌اند.

در هر دو ساختار، ترتیب عناصر اضافه‌شده حفظ می‌شود؛ یعنی هنگام پیمایش و نمایش عناصر، ترتیب نهایی دقیقاً مطابق با ترتیب درج (`insert`) آن‌ها است. هر دو کلاس به صورت پیش‌فرض ناهمزمان (`non-synchronized`) هستند و می‌توان آن‌ها را با استفاده از متدهای `Collections.synchronizedList` به ساختار سنکرون تبدیل کرد.

بازگردانده شده توسط این کلاس‌ها از نوع `fail-fast` هستند؛ به این معنا که اگر پس از ایجاد `iterator` تغییری در لیست ایجاد شود، استثنای `ConcurrentModificationException` پرتاب خواهد شد.

چه زمانی از `ArrayList` و چه زمانی از `LinkedList` استفاده کنیم؟

اگر تعداد عملیات درج (`insert`) و حذف (`delete`) زیاد باشد، استفاده از `LinkedList` انتخاب مناسب‌تری است؛ زیرا این عملیات (با صرف‌نظر از زمان پیمایش) هزینه زمانی کمتری دارند.

عملیات جستجو و متدهای `get` در `ArrayList` به مراتب سریع‌تر از `LinkedList` است. بنابراین در سناریوهایی که نیاز به دسترسی‌های متوالی و مکرر به عناصر وجود دارد و تعداد عملیات درج و حذف کم است، استفاده از `ArrayList` اولویت دارد.

Java در Vector

کلاس **Vector** نیز واسط **List** را پیاده‌سازی می‌کند.

در این ساختار، ترتیب ورود عناصر حفظ می‌شود و هنگام پیمایش، عناصر به همان ترتیب درج شده نمایش داده می‌شوند.

از آنجایی که **Vector** به صورت پیش‌فرض **سنکرون** است، معمولاً در محیط‌های تک‌ریسمانی (Single-threaded) کمتر مورد استفاده قرار می‌گیرد؛ زیرا به دلیل استفاده از مکانیزم‌های قفل‌گذاری (Locking)، عملیات‌هایی مانند جستجو، اضافه‌کردن و حذف عناصر کندتر انجام می‌شوند.

روش اول ایجاد **:Vector**

```
Vector vec = new Vector();
```

با این دستور، یک **Vector** خالی با ظرفیت اولیه ۱۰ ایجاد می‌شود. اگر تعداد عناصر از ۱۰ بیشتر شود، ظرفیت **Vector** دو برابر خواهد شد. به عنوان مثال، با افزودن عنصر یازدهم، ظرفیت از ۱۰ به ۲۰ افزایش می‌یابد.
در مقایسه، در **ArrayList** ظرفیت هر بار حدود ۱,۵ برابر افزایش پیدا می‌کند.

روش دوم ایجاد **:Vector**

```
Vector vec = new Vector(int initialCapacity);  
Vector vec = new Vector(3);
```

در این حالت، یک **Vector** با ظرفیت اولیه مشخص شده ایجاد می‌شود.

روش سوم ایجاد **:Vector**

```
Vector vec = new Vector(int initialCapacity, int capacityIncrement);  
Vector vec = new Vector(4, 6);
```

آرگومان اول ظرفیت اولیه **Vector** و آرگومان دوم مقدار افزایش ظرفیت پس از پرشدن آن را مشخص می‌کند.
به عنوان مثال، اگر **Vector** با ظرفیت ۴ پر شود و بخواهیم عنصر پنجم را اضافه کنیم، ظرفیت به $10 = 6 + 4$ افزایش می‌یابد و هنگام افزودن عنصر یازدهم، ظرفیت به $16 = 10 + 6$ خواهد رسید.

```
import java.util.*;

public class VectorExample {

    public static void main(String args[]) {
        /* Vector of initial capacity(size) of 2 */
        Vector<String> vec = new Vector<String>(2);
        /* Adding elements to a vector*/
        vec.addElement("Apple");
        vec.addElement("Orange");
        vec.addElement("Mango");
        vec.addElement("Fig");
        /* check size and capacityIncrement */
        System.out.println("Size is: " + vec.size());
        System.out.println("Default capacity increment is: " + vec.capacity());
        vec.addElement("fruit1");
        vec.addElement("fruit2");
        vec.addElement("fruit3");
        /* size and capacityIncrement after two insertions */
        System.out.println("Size after addition: " + vec.size());
        System.out.println("Capacity after increment is: " + vec.capacity());

        /* Display Vector elements */
        Enumeration en = vec.elements();
        System.out.println("\nElements are:");
        while (en.hasMoreElements())
            System.out.print(en.nextElement() + " ");
    }
}
```

خروجی

Size is: 4 •
Default capacity increment is: 4 •
Size after addition: 7 •
Capacity after increment is: 8 •
Elements are: •
Apple Orange Mango Fig fruit1 fruit2 fruit3 •

متد های Vector

`void addElement(Object element)`

این متد عنصر داده شده را به انتهای Vector اضافه می کند.

`int capacity()`

ظرفیت فعلی Vector را برمی گرداند.

`int size()`

تعداد عناصر موجود در Vector را برمی گرداند.

`void setSize(int size)`

اندازه فعلی Vector را به مقدار مشخص شده تغییر می دهد.

`boolean contains(Object element)`

بررسی می کند آیا عنصر مشخص شده در Vector وجود دارد یا خیر؛ در صورت وجود مقدار `true` و در غیر این صورت `false` برمی گرداند.

`boolean containsAll(Collection c)`

اگر تمام عناصر موجود در Collection داده شده در Vector وجود داشته باشند، مقدار `true` بازمی گرداند.

`Object elementAt(int index)`

عنصر موجود در اندیس مشخص شده را برمی گرداند.

Object firstElement()
اولین عنصر Vector را برمی‌گرداند.

Object lastElement()
آخرین عنصر Vector را برمی‌گرداند.

Object get(int index)
عنصر موجود در اندیس مشخص شده را برمی‌گرداند.

boolean isEmpty()
اگر Vector هیچ عنصری نداشته باشد مقدار **true** برمی‌گرداند.

boolean removeElement(Object element)
عنصر مشخص شده را از Vector حذف می‌کند.

boolean removeAll(Collection c)
تمام عناصری را که در Collection داده شده وجود دارند از Vector حذف می‌کند.

void setElementAt(Object element, int index)
مقدار عنصر موجود در اندیس مشخص شده را با عنصر جدید جایگزین می‌کند.

مقایسه Vector و ArrayList

سنکرون بودن: همان‌طور که گفته شد، **ArrayList** به صورت پیش‌فرض آسنکرون (non-synchronized) است؛ یعنی چندین ترد می‌توانند به‌طور هم‌زمان روی یک شیء **ArrayList** عملیات انجام دهند. برای مثال، در حالی که یک ترد متدها `add` را اجرا می‌کند، ترد دیگری می‌تواند متدهای `remove` را فراخوانی کند. در مقابل، **Vector** سنکرون است و در هر لحظه تنها یک ترد اجازه دسترسی به شیء **Vector** را دارد؛ بنابراین دسترسی هم‌زمان چندتردی به آن ممکن نیست.

تغییر اندازه: هر دو کلاس **ArrayList** و **Vector** به صورت پویا افزایش یا کاهش اندازه می‌یابند، اما نحوه و میزان افزایش ظرفیت در این دو متفاوت است.

کارایی: **ArrayList** به دلیل آسنکرون بودن، کارایی بالاتری دارد. در مقابل، مکانیزم قفل‌گذاری (Locking) در **Vector** باعث کاهش کارایی آن می‌شود، به‌ویژه در محیط‌های تک‌ریسمانی یا با میزان رقابت کم.

: **Fail-Fast** هستند؛ به این معنا که در صورت تغییر ساختار لیست پس از ایجاد **iterator**، استثنای **ConcurrentModificationException** پرتاب خواهد شد.

چه زمانی از **Vector** و چه زمانی از **ArrayList** استفاده کنیم؟

به محیط و کاربرد شما بستگی دارد. اگر **thread-safe** بودن برایتان مهم نیست از **ArrayList** و در غیراینصورت از **Vector** استفاده کنید.

خودمون رو بسنجیم

این بخش برای این طراحی شده که بعد از مطالعه مباحث Collections، خودت رو محک بزنی و مطمئن بشی مفاهیم کلیدی رو یاد گرفتی. بدون نگاه کردن به متن درس، به پرسش‌ها جواب بد:

- چه نیازهایی با آرایه تامین نمی‌شود؟ راه حل چیه؟
- LinkedList و ArrayList رو باهم مقایسه کن.
- Vector و ArrayList رو باهم مقایسه کن.
- کدی به دفعات به سراغ اندیسی تصادفی در میانه لیست می‌رود؛ اگر list یک ArrayList باشد کد سریع‌تر اجرا می‌شود یا LinkedList؟
- برنامه‌ای بنویس که چند عنصر (لیست پیوندی جدید) را در جای مشخص (مثلاً اندیس 1) به یک LinkedList اضافه کند و عناصر به همراه اندیس‌ها را چاپ کند.
- برنامه‌ای بنویس که عناصر یک vector را به دیگری در اندیس مشخص (مثلاً 2) اضافه کند و عناصر را در یک آرایه کپی و چاپ کند.

پایان

در صورت هرگونه سوال یا پیشنهاد می‌تونی با من
در ارتباط باشی:)

gmail: nanafarhanj@gmail.com

telegram: [@nana_f83](https://t.me/nana_f83)