



دانشکده مهندسی و علوم کامپیوتر

# برنامه نویسی پیشرفته وحیدی اصل

Collections-بخش لول

- Collections در جاوا چارچوبی است که امکان ذخیره سازی و دستکاری مجموعه ای از اشیا را فراهم می کند.
- همه عملیاتی که می توانید بر روی 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.
- Collection جاوا واسطه‌های (interface) بسیاری نظیر (Set, List, Queue, Deque ,...) و همچنین کلاس‌هایی نظیر (ArrayList, Vector, 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.

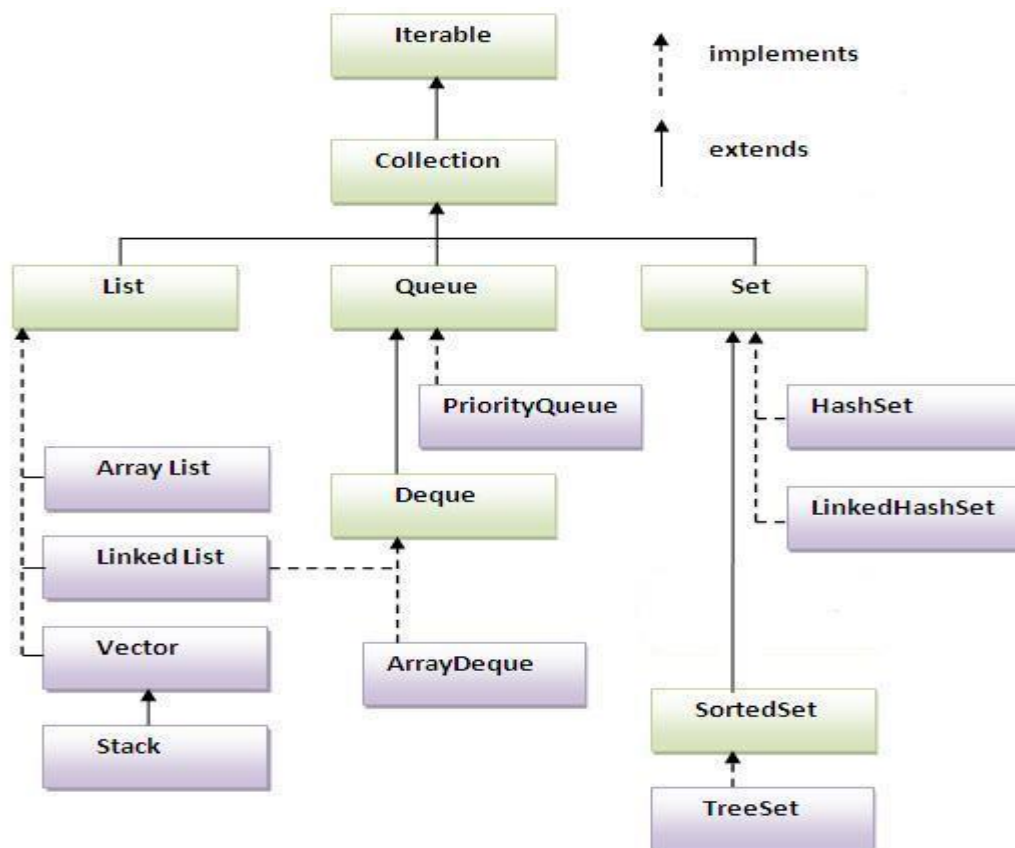
- یک **Collection** (*container*) شیئی است که حاوی گروهی از اشیا بوده و به این گروه در قالب یک واحد نگریسته می شود.
- انواع مختلفی از اشیا می توانند به عنوان عناصر **collections** ذخیره سازی، بازیابی و دستکاری شوند.



- یک **collection** را شیئی تصور کنید که ریموت کنترل‌هایی به عناصر (اشیایی دیگر) را نگهداری می‌کند.
- چارچوب **collections** جاوا بخشی از پکیج **java.util** را تشکیل می‌دهد.
- یک چارچوب **collections** از سه بخش اصلی تشکیل شده است:
  - **Interfaces**: هر واسطه، عملیات و قواعدی برای یک نوع **collection** مشخص (مانند **List**، **Set**، **Queue** ...) تعریف می‌کند.
  - با دانستن متدهای واسطه یک **collection** از عملکرد و خصوصیات آن آگاه می‌شوید.
  - **Implementations**: شامل کلاس‌هایی است که واسطه‌های بالا را پیاده‌سازی کرده‌اند (برای مثال **LinkedList**، **HashSet** ...).
  - **Algorithms**: متدهای پلی مورفیک سودمند برای ایجاد و دستکاری اشیایی از کلاسهای بالا مانند **Sorting**، **index searching**، **replacing**، **reversing** و غیره

# سلسله مراتب چارچوب Collections

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

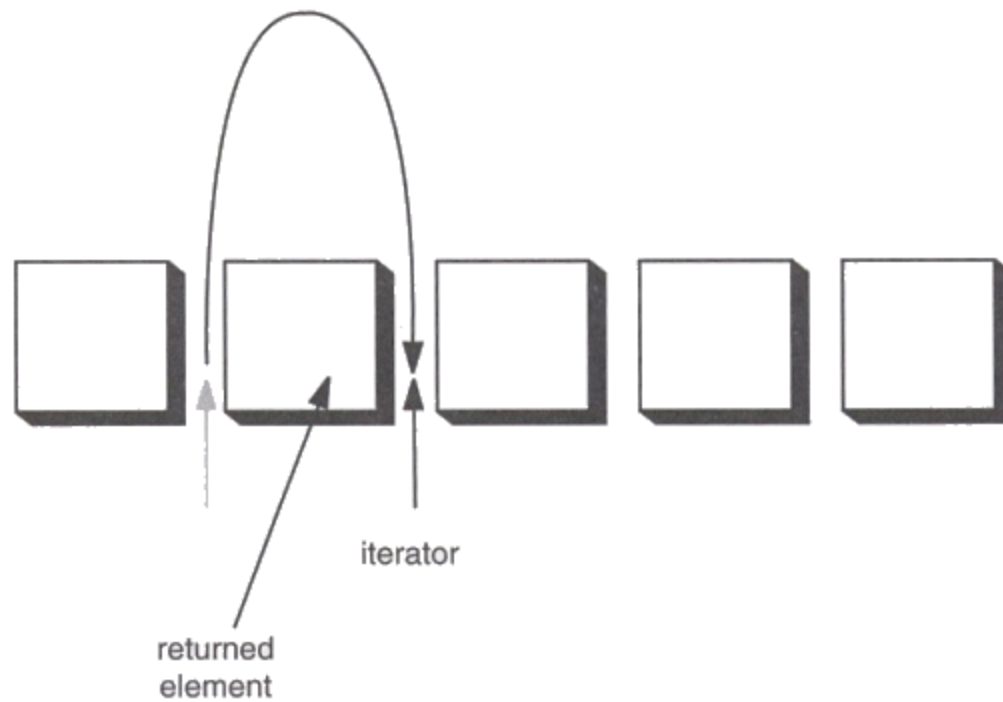


# Methods of Collection interface

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

- سه متد اصلی را تعریف کرده است:
  - `Object next()`
  - `boolean hasNext()`
  - `void remove()`
- این سه متد امکان دسترسی به عناصر ذخیره شده در `collections` را فراهم می کنند:
- یک `Iterator` محل عناصر را در یک `Collections` می داند.
- هربار فراخوانی `next()` سبب می شود عنصر بعدی در `collection` خوانده شود.
- می توانید این عنصر را استفاده یا حذف کنید.

# Iterator Position



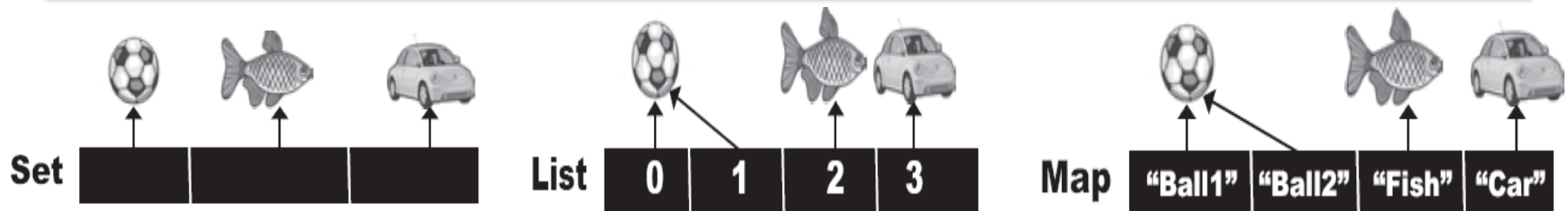


- یک Iterator شیئی است که به شما امکان می دهد یک collection را پیمایش کنید و با قرار گرفتن روی عنصر ذخیره شده در آن collection، در صورت تمایل عنصر را حذف کنید.
- برای ایجاد شیئی از Iterator برای پیمایش عناصر یک collection، کافیست متد iterator() را بر روی شیء collection، فراخوانی کنید.
- واسط Iterator به صورت زیر تعریف شده است:

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

- در این درس واسطه‌ها و کلاسهای زیر را بررسی می‌کنیم:

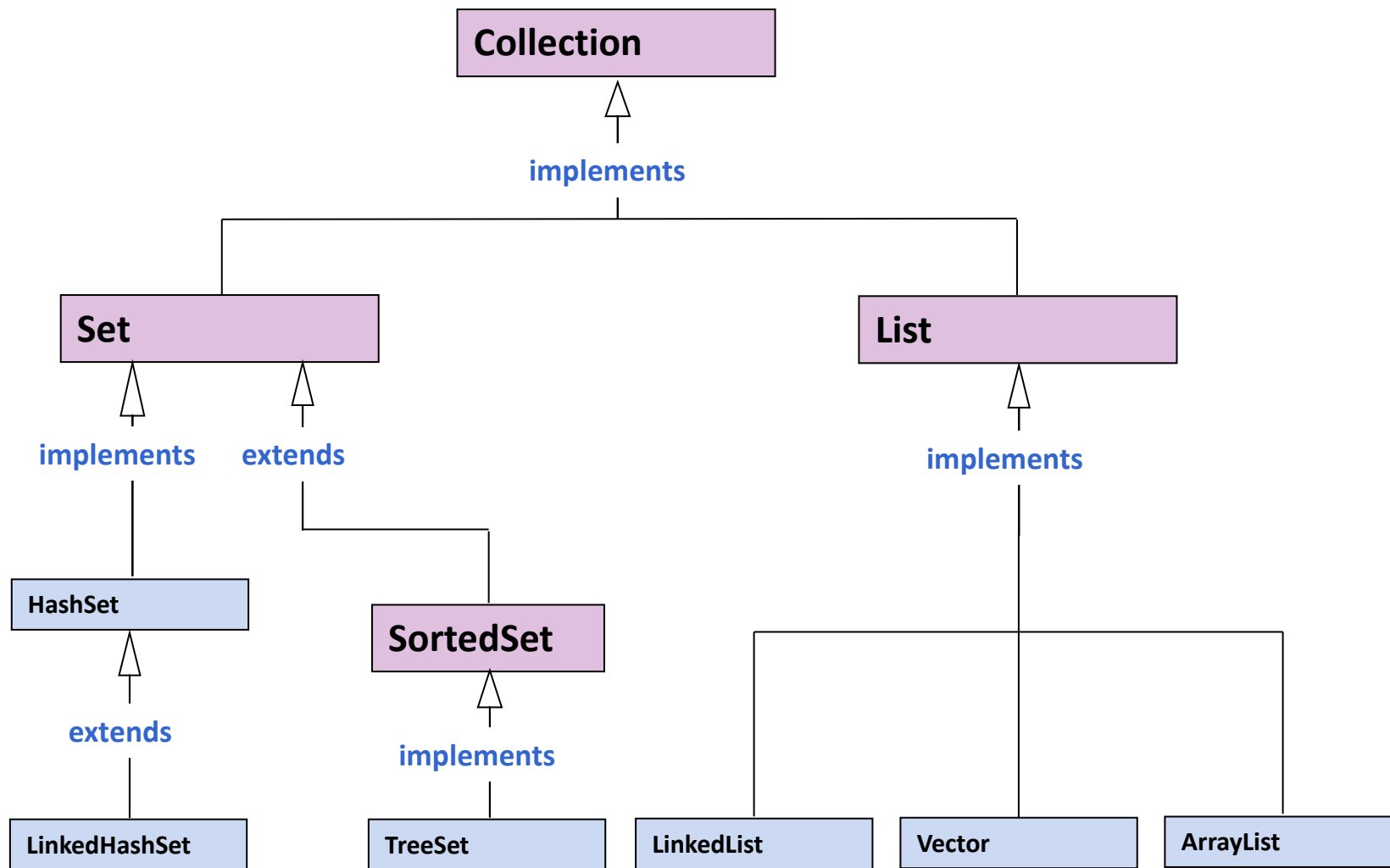
Set	List	Map
HashSet	ArrayList	HashMap
LinkedHashSet	LinkedList	LinkedHashMap
TreeSet	Vector	Hashtable
		TreeMap



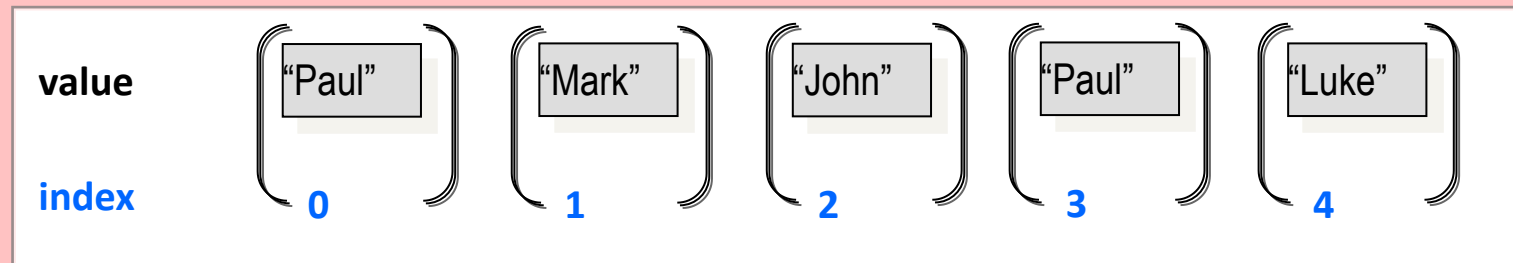
- بررسی خالی بودن collection
- بررسی اینکه آیا شیئی در collection موجود است.
- بازیابی یک شیئی از collection
- اضافه نمودن یک شیئی به collection
- حذف یک شیئی از collection
- پیمایش collection و بررسی هر شیئی
- هریک از انواع collection هریک از عملیات بالا را در قالب یک متد منحصر به خود پیاده سازی کرده اند.

- 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؟

# Set and List: Collections سلسله مراتب



# لیست چیست؟



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 -۱ 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 -۱ 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[ ]) {

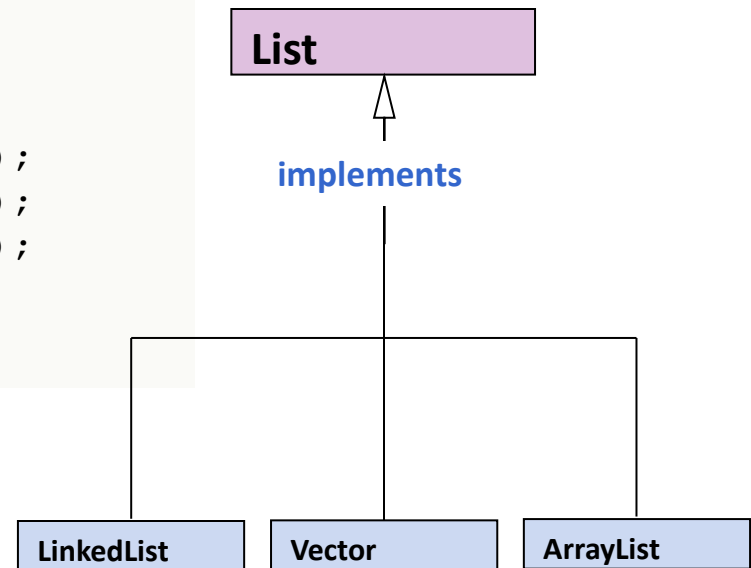
        ArrayList alist = new ArrayList( );

        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));

    }
}
```

One  
Two  
Three





- 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؟

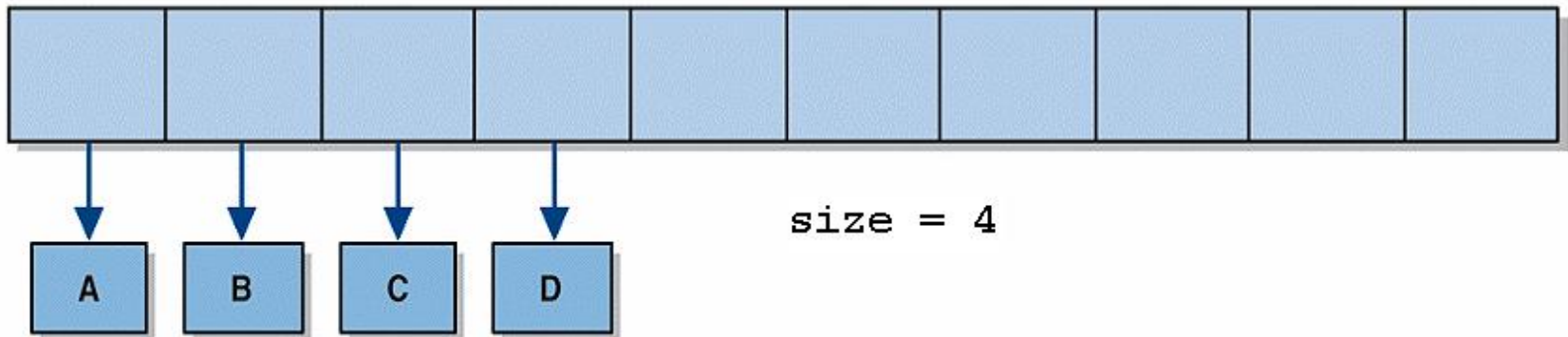
# ArrayList

• **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.*;`

`elements =`

0      1      2      3      4      5      6      7      ...



- در حقیقت یک آرایه دارای قابلیت تغییراندازه خودکار می باشد که می تواند هر نوع شیئی را با استفاده از متدهای تعریف شده اش نگهداری و دستکاری کند.
- ArrayList بیشتر مزایای یک آرایه معمولی را دارد، به ویژه دسترسی تصادفی!
- برنامه نویس دغدغه کارهایی مانند شیفต์ عناصر یا تغییراندازه آرایه داخلی را ندارد. اندازه اولیه ArrayList نیاز نیست از ابتدا مشخص شود!
- با فراخوانی toString بر روی یک ArrayList عناصر آن در قالب یک لیست برگردانده می شوند.

[1, 2.65, Marty Stepp, Hello]

- کلاس `java.util.Collections` تعداد زیادی متد استاتیک کارآمد دارد که بر روی اشیای **collections** (نظیر لیستها) اعمال می شود.
- این کلاس **utility**، واسطه **Collection** را پیاده سازی کرده است.

```
public static void copy(List dest, List src)
public static void fill(List list, Object value)
public static Object max(Collection coll)
public static Object min(Collection coll)
public static void reverse(List list)
public static void shuffle(List list)
public static void sort(List list)
public static void swap(List list, int i, int j)
```

- مثال:

```
Collections.sort(myArrayList);
```

برای مثال، متد `copy` در کلاس `Collections` محتوای یک لیست را در لیستی دیگر کپی می کند. بنابراین این کلاس مستقل از اشیای ساخته شده از `collection` می باشد.

## چگونه از ArrayList یک زیرلیست بگیریم؟

- چگونه از ArrayList یک زیرلیست بگیریم؟
  - اگر بخواهیم از ArrayList مان یک زیرلیست استخراج کنیم از متد `subList` در کلاس `ArrayList` استفاده می کنیم:
- `List subList(int fromIndex, int toIndex)`
- این متد زیرلیستی از اندیس `fromIndex` تا به `toIndex` (اندیس `toIndex` را شامل نمی شود) لیست فراخواننده این متد را استخراج می کند.

```
import java.util.ArrayList;
import java.util.List;
public class SublistExample {

    public static void main(String a[]){
        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);
    }
}
```

Original ArrayList Content: [Steve, Justin, Ajeet, John, Arnold, Chaitanya]

SubList stored in ArrayList: [Justin, Ajeet, John]

SubList stored in List: [Justin, Ajeet, John]

## استثنای ایجاد شده در subList

- متد subList استثنای IndexOutOfBoundsException را پرتاب می کند اگر اندیسهای مشخص شده در محدوده اندیسهای ArrayList نباشند ( $\text{fromIndex} < 0 \parallel \text{toIndex} > \text{size}$ )
- اگر اندیس شروع بزرگتر از اندیس پایانی متد باشد ( $\text{fromIndex} > \text{toIndex}$ )، 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);
        }
    }
}
```

• از متد addAll() استفاده کنید.

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

```
AL1: E1
AL1: E2
AL1: E3
AL2: E1
AL2: E2
AL2: E3
```



- For Loop
- Advanced for loop

```
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
- Iterator

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

# 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());
```

```
        }
```

```
    }
```

```
ArrayList elements:
```

```
C
```

```
C++
```

```
Java
```

```
DotNet
```

```
Perl
```

# تبدیل یک آرایه به ArrayList – روش اول

- تبدیل با استفاده از Arrays.asList()
- قاعده نحوی:

`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);
        }
    }
}
```

## تبدیل یک آرایه به ArrayList - روش دوم

- استفاده از `addAll()` - سریعتر از روش قبلی می باشد.
- قاعده نحوی:

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

یا:

```
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");
```

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

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

```
Hi
Hello
Howdy
Bye
String1
String2
```

# تبدیل 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);
        }
    }
}
```

Text1

Text2

Text3

Text4

# تبدیل 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[frnames.size()]);

        /*Displaying Array elements*/
        for(String k: frnames)
        {
            System.out.println(k);
        }
    }
}
```

# تبدیل 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);
        }
    }
}
```

# سنکرون سازی ArrayList

- ArrayList آسنکرون بوده و در محیطی چندترییدی می تواند 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 می گیرد:
  - استفاده از متد `Collections.synchronizedList()`
  - استفاده از نسخه `thread-safe` لیست: `CopyOnWriteArrayList`
- در هنگام استفاده از `iterator` بر روی یک شیء `ArrayList` بهتر است آن را در بلاک سنکرون قرار دهیم.
- مثال در اسلاید بعدی آمده است.



# سنکرون سازی ArrayList

```
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());
        }
    }
}
```

# استفاده از نسخه thread-safe لیست: CopyOnWriteArrayList

```
import java.util.Iterator;

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());
    }
}
```

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

## معایب 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
----	----	---	----	----

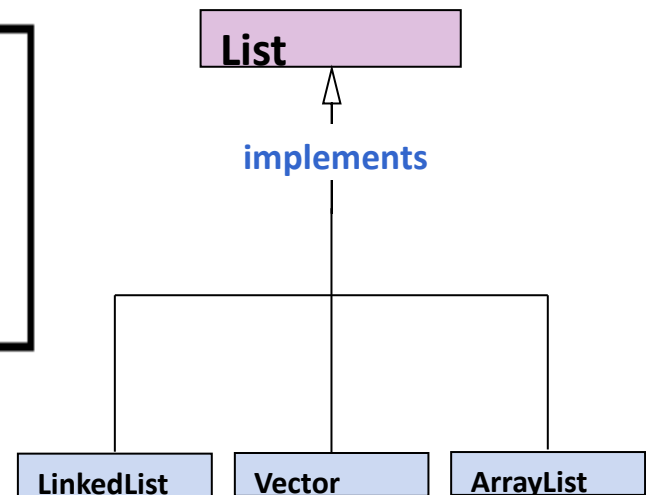
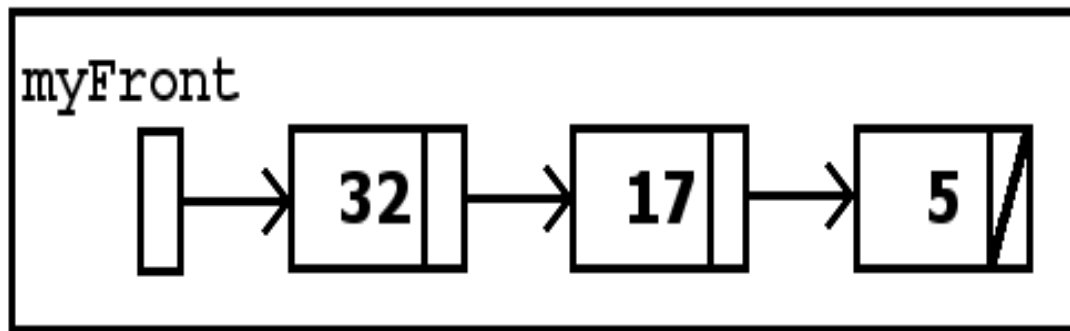
32	17	5	24	-3
----	----	---	----	----

- یک 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 گوئیم)
- آخرین **node** دارای **next == null** می باشد (که در شکل به صورت / نمایش داده می شود)

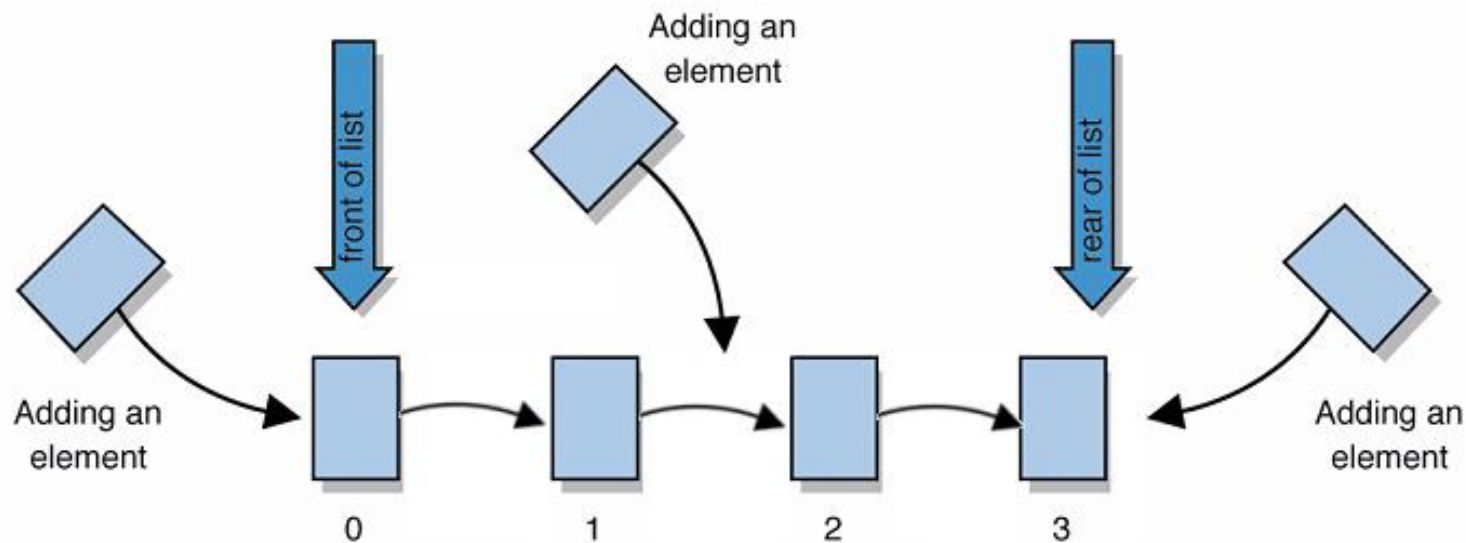


## لیست پیوندی (linked list)

- لیست پیوندی به **collection** ای گفته می شود که عملیات لیست را بر روی دنباله ای زنجیر شده از گرهها پیاده سازی می کند.
- برای یک لیست پیوندی کافیت 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.

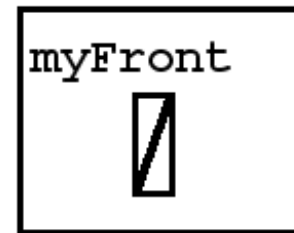


- کلاس `java.util.LinkedList` اشیایی از یک لیست پیوندی را با پیاده سازی داخلی خود، ایجاد می کند.



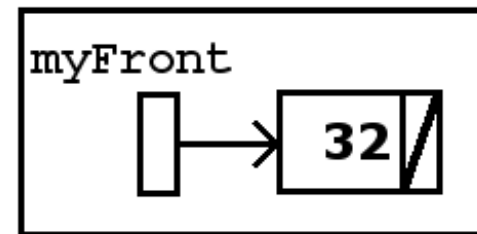
- لیست خالی

(myFront == null)



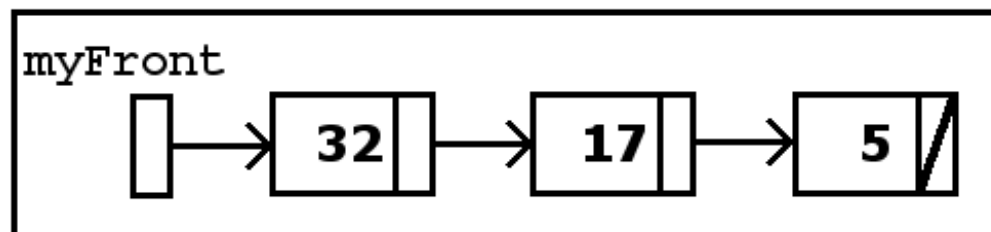
- لیستی با یک عنصر

- list with one element



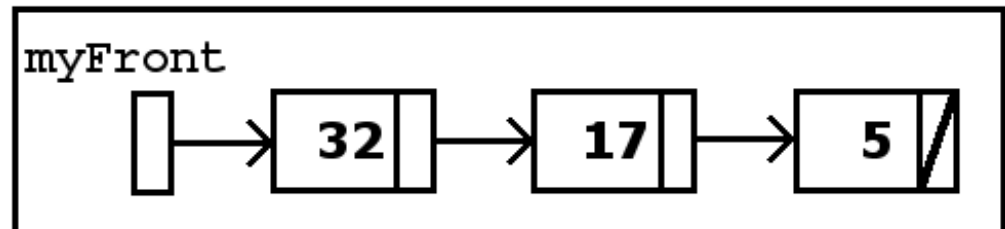
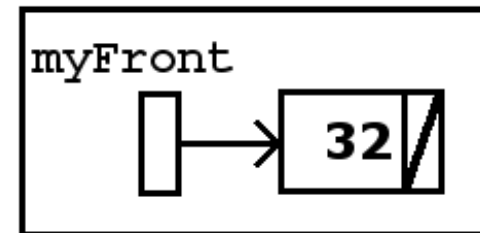
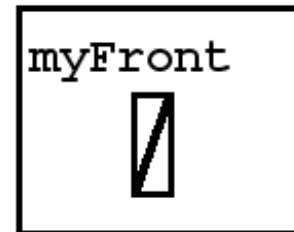
- لیستی با چندین عنصر

- list with many elements





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



```

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

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

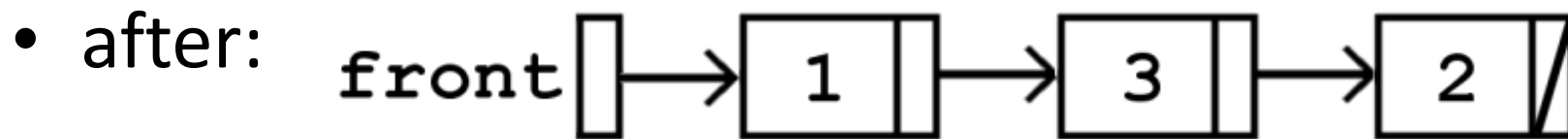
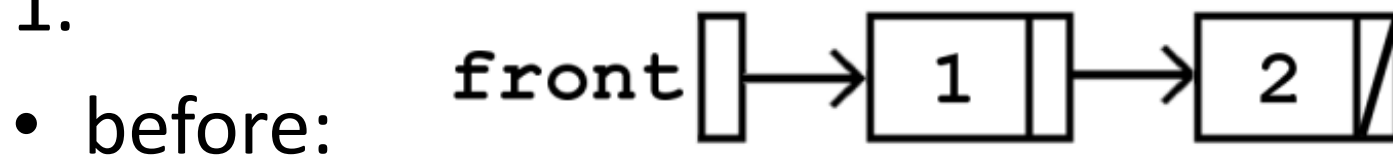
    public Node(Object element, Node next)
    {
        this.element = element;
        this.next = next;
    }
}

```

## مثالهای (a) Linked node

- هر گره حاوی شیئی از Integer است:


1.

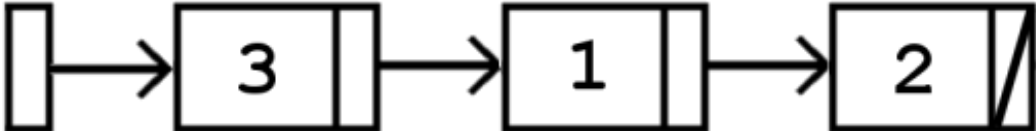


```
front.next = new Node(new Integer(3), front.next);
```

## مثالهای (b) Linked node

2.

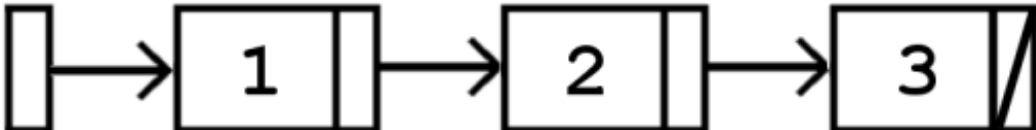
• before: `front` 

• after: `front` 

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

3.

• before: `front` 

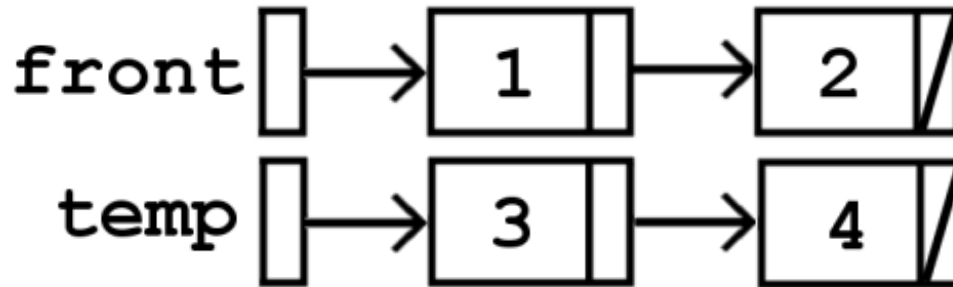
• after: `front` 

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

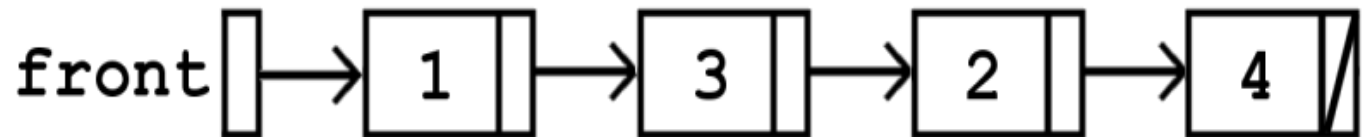
## (d) Linked node مثالهای

6.

- before:



- after:



7.

- before:



- after:



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

```

/* Models an entire linked list. */
public class IntLinkedList {
    private Node myFront;

    public IntLinkedList() {
        myFront = null;
    }

    /* Methods go here */
}

```

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

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

```
// returns number of elements
public int size()
```



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

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

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

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

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

```

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

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

// removes all values from list
public void clear()

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

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

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

```

```
import java.util.*;

public class TestCollection7{

    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());
        }
    }
}
```

## Test it Now

Output:Ravi  
 Vijay  
 Ravi  
 Ajay

## خصوصیات لیستهای 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 متد indexOf و contains کند هستند.
- دستکاری (مانند حذف و اضافه یک عنصر) در آن سریع است چون به شیف نیاز ندارد.
- برای پیاده سازی پشته ها، صفها، درختها و گرافها استفاده می شود.

# مقایسه ArrayList و LinkedList

- جستجو: عملیات جستجو در ArrayList در مقایسه با LinkedList سریعتر می باشد. زمان  $get(int\ index)$  در ArrayList دارای پیچیدگی  $O(1)$  می باشد در حالی که در LinkedList دارای پیچیدگی  $O(n)$  است.
- دلیل: سیستم ArrayList مبتنی بر اندیس می باشد که کارایی ساختار آرایه سنتی در زبانهای برنامه نویسی را دارد.
- اما لیست پیوندی doubly linked list را پیاده سازی کرده که نیاز به پیمایش همه عناصر برای جستجوی یک عنصر را دارد.
- حذف: عملیات حذف در LinkedList با در نظر نگرفتن زمان رسیدن به اندیس مورد نظر، دارای پیچیدگی  $O(1)$  است در حالی که ArrayList دارای پیچیدگی زمانی  $O(n)$  در بدترین حالت (حذف اولین عنصر) و  $O(1)$  در بهترین حالت است (حذف عنصر آخر)
  - توجه داشته باشید اگر پیمایش LinkedList جهت رسیدن به اندیس عنصری که می خواهیم آن را حذف کنیم، در نظر گرفته شود، زمان حذف افزایش خواهد یافت و ممکن است بهبود زمانی LinkedList نسبت به ArrayList قابل توجه نباشد.
  - اما در مواقعی که حذف عناصر از اندیسهای اول LinkedList به طور متوالی انجام شود، به دلیل زمان پیمایش کمتر و عدم نیاز به عملیات شیفت سرعت LinkedList به مراتب از بیشتر از ArrayList خواهد شد.
- نتیجه گیری: حذف عنصر LinkedList در حالت متوسط (اغلب اوقات) سریعتر از ArrayList می باشد.
- دلیل: لیست پیوندی دارای دو ریموت کنترل در هر گره است که دو عنصر مجاورش را کنترل می کند. در نتیجه حذف تنها به تغییر مقادیر ریموت کنترلها در گرههای مجاور عنصر حذف شونده نیاز دارد.
- در حالی که پس از حذف یک عنصر در ArrayList سایر عناصر باید شیفت داده شوند تا فضای خالی عنصر حذف شده را پر کنند!

# مقایسه ArrayList و LinkedList

- عملیات insert در لیست پیوندی دارای پیچیدگی  $O(1)$  است (با در نظر گرفتن زمان پیمایش) ؛
- در حالی که در ArrayList در بدترین حالت پیچیدگی  $O(n)$  دارد.
- دلیل آن مشابه حذف یک عنصر است.
- سربار حافظه: ArrayList اندیسها و ریموت کنترل عناصر داده ای را نگهداری می کند، در حالی که LinkedList عناصر داده ای و دو ریموت برای هر گره را نگهداری می کند در نتیجه لیست پیوندی سربار حافظه ای بیشتری دارد.

## شباهتهای میان دو کلاس لیست پیوندی و ArrayList:

- هر دو واسط List را پیاده سازی کرده اند.
- در هر دو ساختار، ترتیب عناصر اضافه شده به ساختار حفظ می شود. یعنی در هنگام پیمایش و نمایش عناصر، ترتیب نهایی با ترتیب اولیه insert شدن عناصر، یکی است.
- هر دو کلاس آسنکرون هستند و می توانند با متد Collections.synchronizedList سنکرون شوند.
- iterator و listIterator برگردانده شده توسط این کلاسها fail-fast هستند. اگر لیست پس از ایجاد iterator تغییر پیدا کند، در این حالت استثنای ConcurrentModificationException پرتاب خواهد شد!

- اگر تعداد insert و deletion زیاد است، لیست پیوندی اولویت دارد.
- جستجو و متد get در Arraylist به مراتب سریعتر از LinkedList هستند. بنابراین در هنگام نیاز به دسترسی های متوالی به عناصر و نیاز اندک به insert و delete، ArrayList اولویت دارد.

# Vector in Java

- **Vector** نیز واسط لیست را پیاده سازی کرده است.
- در این کلاس نیز ترتیب ورود عناصر حفظ می شود.
- چون سنکرون است به ندرت در محیطهای تک تریدی استفاده می شود. زیرا به سبب داشتن سازوکارهای قفل بندی عملیات جستجو، اضافه کردن ، حذف و ...عناصر در آن، کند است.
- روش اول ایجاد یک **Vector**:

**Vector vec = new Vector();**

- با این دستور یک **Vector** خالی با ظرفیت اولیه ۱۰ می سازید.
- اگر تعداد عناصر از ۱۰ بیشتر شود، ظرفیت **Vector** دوبرابر خواهد شد. یعنی اگر ۱۱ عنصر به **Vector** خالی اضافه شود، ظرفیت ۲۰ و نه ۱۱ خواهد شد.
- در **ArrayList** ظرفیت هر بار 1.5 برابر می شود.



# Vector in Java

- روش دوم ایجاد یک Vector:

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

- برداری به ظرفیت داده شده ایجاد می کند.

- روش سوم ایجاد یک Vector:

```
Vector object= new vector(int initialcapacity, capacityIncrement)
Vector vec= new Vector(4, 6)
```

- آرگومان اولی ظرفیت بردار و آرگومان دوم میزان افزایش ظرفیت در صورت پر شدن بردار را مشخص می کند.
- مثلاً اگر بردار با ۴ عنصر پر شد و بخواهیم عنصر پنجم را اضافه کنیم، ظرفیت (4+6) خواهد شد و در هنگام اضافه کردن عنصر یازدهم، ظرفیت (10+6) خواهد شد.

# مثالی از vector

```
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() + " ");
```

```
        }
```

```
    }
```

```
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
```

- **void addElement(Object element):** It inserts the element at the end of the Vector.
- **int capacity():** This method returns the current capacity of the vector.
- **int size():** It returns the current size of the vector.
- **void setSize(int size):** It changes the existing size with the specified size.
- **boolean contains(Object element):** This method checks whether the specified element is present in the Vector. If the element is been found it returns true else false.
- **boolean containsAll(Collection c):** It returns true if all the elements of collection c are present in the Vector.
- **Object elementAt(int index):** It returns the element present at the specified location in Vector.
- **Object firstElement():** It is used for getting the first element of the vector.
- **Object lastElement():** Returns the last element of the array.
- **Object get(int index):** Returns the element at the specified index.
- **boolean isEmpty():** This method returns true if Vector doesn't have any element.
- **boolean removeElement(Object element):** Removes the specifed element from vector.
- **boolean removeAll(Collection c):** It Removes all those elements from vector which are present in the Collection c.
- **void setElementAt(Object element, int index):** It updates the element of specifed index with the given element.

# Vector در مقایسه با ArrayList

- سنکرون بودن: همانگونه که پیشتر گفتیم، ArrayList آسنکرون است.
- به این معنا که همزمان چندین ترید می توانند بر روی یک شیء ArrayList کار کنند.
- برای مثال، اگر تریدی عملیات add را بر روی شیئی از ArrayList انجام می دهد، همزمان ترید دیگری می تواند عملیات remove را بر روی آن انجام دهد.
- اما vector سنکرون بوده و اگر تریدی از آن استفاده کند، همزمان ترید دیگری امکان دسترسی به آن شیء را ندارد.
- تغییر اندازه: هر دو کلاس ArrayList و Vector به صورت پویا منبسط و منقبض می شوند، اما میزان آن در دو کلاس متفاوت می باشد.
- کارایی: ArrayList به خاطر آسنکرون بودن کارایی بهتری دارد. قفل بندی در vector کارایی آن را کاهش می دهد.
- Fail-fast: هر دو کلاس Fail-Fast هستند.
- Enumeration در Vector این مشکل را ندارد.

## چه زمانی از ArrayList و چه زمانی از Vector 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.