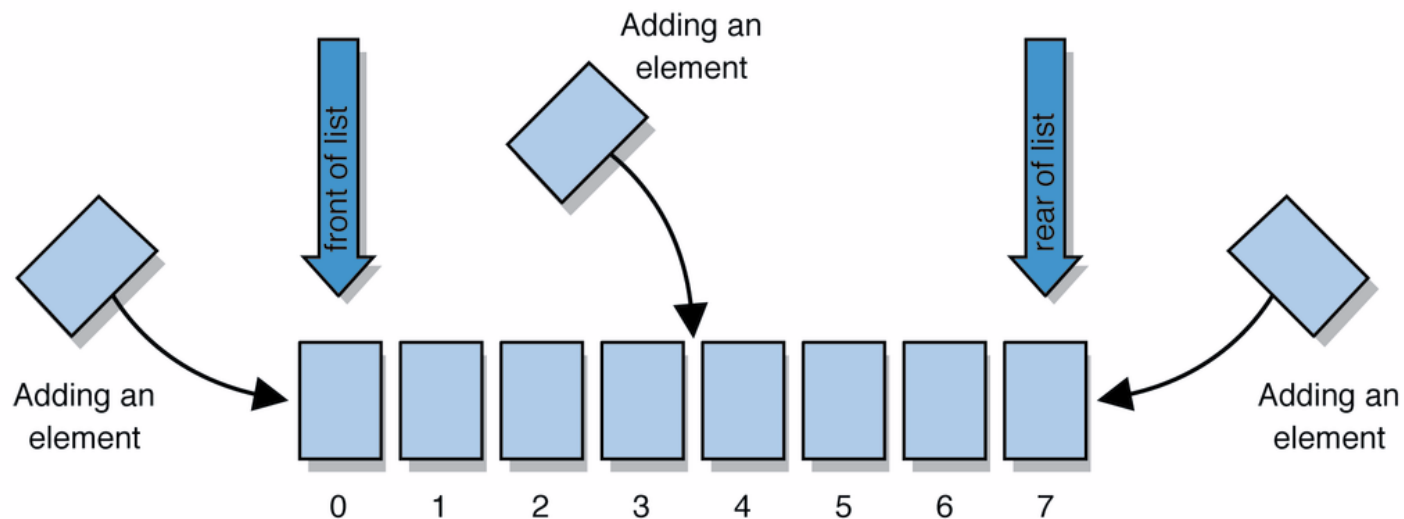


# Lists

- a collection storing a sequence of elements
  - each element is accessible by a 0-based **index**
  - a list has a **size** (# of elements present)
  - element can be added to the front, back, or elsewhere
  - in Java, a list can be represented as an **ArrayList** object



# Linear List

## Method Summary

void **addToFront**(java.lang.Object element)

Adds the given element to the front of the current list

void **addToRear**(java.lang.Object element)

Adds the given element to the rear of the current list

boolean **isEmpty**()

Predicate returns true if the list is empty and false otherwise

java.lang.Object **removeFront**()

Removes the first element in the list and returns a reference to it.

java.lang.Object **removeRear**()

Removes the last element in the list and returns a reference to it.

int **size**()

Returns the number of nodes in the list

java.lang.String **toString**()

Returns a string representation of the list.

# Create an empty List

```
ArrayLinearList a = new ArrayLinearList(100);
```

```
    b = new ArrayLinearList()
```

# The class ArrayLinearList

```
/** array implementation of LinearList */  
import java.util.*; // has Iterator interface  
import utilities.*; // has array resizing class  
  
public class ArrayLinearList // implements LinearList  
{ // data members  
    protected Object element[]; // array of elements  
    protected int size; // number of elements in array  
    // constructors and other methods come here  
}
```

# A Constructor

```
/** create a list with initial capacity initialCapacity
 * @throws IllegalArgumentException when
 * initialCapacity < 1 */
public ArrayList(int initialCapacity)
{
    if (initialCapacity < 1)
        throw new IllegalArgumentException
            ("initialCapacity must be >= 1");
    // size has the default initial value of 0
    element = new Object [initialCapacity];
}
```

## Another Constructor

```
/** create a list with initial capacity 10 */  
public ArrayLinearList()  
{// use default capacity of 10  
    this(10);  
}
```

## The Method isEmpty

```
/** @return true iff list is empty */  
public boolean isEmpty()  
{return size == 0;}
```

## The Method size()

```
/** @return current number of elements in list */  
public int size()  
    {return size;}
```



## The Method checkIndex

```
/** @throws IndexOutOfBoundsException when  
    * index is not between 0 and size - 1 */  
void checkIndex(int index)  
{  
    if (index < 0 || index >= size)  
        throw new IndexOutOfBoundsException  
            ("index = " + index + " size = " + size);  
}
```

# The Method get

```
/** @return element with specified index  
 * @throws IndexOutOfBoundsException when  
 * index is not between 0 and size - 1 */
```

```
public Object get(int index)  
{  
    checkIndex(index);  
    return element[index];  
}
```

## The Method indexOf

```
/** @return index of first occurrence of theElement,  
    * return -1 if theElement not in list */  
public int indexOf(Object theElement)  
{  
    // search element[] for theElement  
    for (int i = 0; i < size; i++)  
        if (element[i].equals(theElement))  
            return i;  
  
    // theElement not found  
    return -1;  
}
```

# The Method remove

```
public Object remove(int index)
{
    checkIndex(index);
    // copy the element to be deleted
    Object removedElement = element[index];
    // valid index, shift elements with higher index
    for (int i = index + 1; i < size; i++)
        element[i-1] = element[i];

    size=size-1;
    element[size] = null; // enable garbage collection
    return removedElement;
}
```



# The Method add

```
// shift elements right one position
```

```
for (int i = size - 1; i >= index; i--)
```

```
    element[i + 1] = element[i];
```

```
element[index] = theElement; // insert element
```

```
size++;
```

```
}
```

# Convert To A String

```
public String toString()
{
    StringBuffer s = new StringBuffer("[");
    // put elements into the buffer
    for (int i = 0; i < size; i++)
        if (element[i] == null) s.append("null, ");
        else s.append(element[i].toString() + ", ");
    if (size > 0) s.delete(s.length() - 2, s.length()); // remove last ", "
    s.append("]");
    // create equivalent String
    return new String(s);
}
```

# Conclusions

## Advantages

- Quick insertion – insert at last –  $O(1)$
- Very fast access if index known –  $O(1)$

## Disadvantages

- Slow search -  $O(n)$
- Slow deletion -  $O(n)$
- Fixed size- need dynamic array