

---

# ArrayLists

---

---

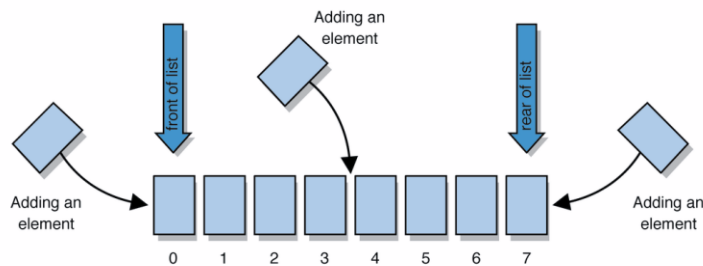
## Using arrays to store data

- Arrays: store multiple values of the same type.
  - Conveniently refer to items by their index
  - Need to know the size before declaring them:

```
int[] numbers = new int[100];
```
  - We often need to store an unknown number of values.
    - Need to either count the values or resize as additional storage space is needed.
-

## Lists

- **list**: a collection storing an ordered sequence of elements, each accessible by a 0-based index
  - a list has a **size** (number of elements that have been added)
  - elements can be added at any position



## ArrayList

- Let's consider the methods of a class called `ArrayList` that represents a list using `int[]`
  - behavior:
    - `add(value)`,            `add(index, value)`
    - `get(index)`,            `set(index, value)`
    - `size()`
    - `remove(index)`
    - `indexOf(value)`
    - ...
  - The list's *size* will be the number of elements added to it so far

---

## ArrayList

- construction

```
int[] numbers = new int[5];
ArrayList list = new ArrayList();
```

- storing a given value:

```
numbers[0] = 42;
list.add(42);
```

- retrieving a value

```
int val = numbers[0];
int val = list.get(0);
```

- searching for a given value

```
for (int i = 0; i < numbers.length; i++) {
    if (numbers[i] == 27) { ... }
}
if (list.indexOf(27) >= 0) { ... }
```

---



---

## Pros/cons of ArrayList

- pro (benefits)

- simple syntax
- don't have to keep track of array size and capacity
- has powerful methods (`indexOf`, `add`, `remove`, `toString`)

- con (drawbacks)

- `ArrayList` only works for `ints` (arrays can be any type)
  - Need to learn how to use the class
-

---

## Java Collections and ArrayLists

- Java includes a large set of powerful classes that provide functionality for storing and accessing collections of objects
- The most basic, `ArrayList`, can store any type of `Object`.
- All collections are in the `java.util` package.

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

---

---

## Type Parameters (Generics)

```
ArrayList<Type> name = new ArrayList<Type>();
```

- When constructing an `ArrayList`, you can specify the type of elements it will contain between `<` and `>`.
  - We say that the `ArrayList` class accepts a *type parameter*, or that it is a *generic* class.

```
ArrayList<String> names = new ArrayList<String>();  
names.add("Asa");  
names.add("Nathan");
```

---

## ArrayList methods

<code>add (value)</code>	appends value at end of list
<code>add (index, value)</code>	inserts given value at given index, shifting subsequent values right
<code>clear ()</code>	removes all elements of the list
<code>indexOf (value)</code>	returns first index where given value is found in list (-1 if not found)
<code>get (index)</code>	returns the value at given index
<code>remove (index)</code>	removes/returns value at given index, shifting subsequent values left
<code>set (index, value)</code>	replaces value at given index with given value
<code>size ()</code>	returns the number of elements in list
<code>toString ()</code>	returns a string representation of the list such as "[3, 42, -7, 15]"

## ArrayList methods 2

<code>addAll (list)</code>	adds all elements from the given list at the end of this list
<code>addAll (index, list)</code>	inserts the list at the given index of this list
<code>contains (value)</code>	returns true if given value is found somewhere in this list
<code>containsAll (list)</code>	returns true if this list contains every element from given list
<code>equals (list)</code>	returns true if given other list contains the same elements
<code>remove (value)</code>	finds and removes the given value from this list
<code>removeAll (list)</code>	removes any elements found in the given list from this list
<code>retainAll (list)</code>	removes any elements <i>not</i> found in given list from this list
<code>subList (from, to)</code>	returns the sub-portion of the list between indexes <b>from</b> (inclusive) and <b>to</b> (exclusive)
<code>toArray ()</code>	returns an array of the elements in this list

## Learning about classes

- The Java API specification website contains detailed documentation of every Java class and its methods.

The screenshot shows the Java API documentation for the `ArrayList` class. The left sidebar contains a search bar and a list of Java packages. The main content area is titled "Constructors and Description" and lists three constructors for `ArrayList`:

- `ArrayList()`: Constructs an empty list with an initial capacity of ten.
- `ArrayList(Collection<E> c)`: Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.
- `ArrayList(int initialCapacity)`: Constructs an empty list with the specified initial capacity.

Below the constructors is a "Method Summary" section with tabs for "All Methods", "Instance Methods", and "Concrete Methods". The "All Methods" tab is selected, showing a table of methods:

Modifier and Type	Method and Description
boolean	<code>add(E e)</code> Appends the specified element to the end of this list.
void	<code>add(int index, E element)</code> Inserts the specified element at the specified position in this list.
boolean	<code>addAll(Collection&lt;E&gt; c)</code> Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean	<code>addAll(int index, Collection&lt;E&gt; c)</code> Inserts all of the elements in the specified collection into this list, starting at the specified position.
void	<code>clear()</code> Removes all of the elements from this list.
Object	<code>clone()</code> Returns a shallow copy of this <code>ArrayList</code> instance.

<https://docs.oracle.com/javase/8/docs/api/>

## Iterating through an array list

- Suppose we want to look for a `value` in an `ArrayList` of `Strings`.

```
for (int i = 0; i < list.size(); i++) {
    if (value.equals(list.get(i))) {
        //do something
    }
}
```

- Alternative:

```
for (String s : list) {
    if (value.equals(s)) {
        //do something
    }
}
```

## Note on generics in Java 7 and above

In version 7 of Java, rather than doing:

```
ArrayList<Type> name = new ArrayList<Type>();
```

You can save a few keystrokes:

```
ArrayList<Type> name = new ArrayList<>();
```

## Modifying while looping

- Consider the following flawed pseudocode for removing elements that end with 's' from a list:

```
removeEndS(list) {
    for (int i = 0; i < list.size(); i++) {
        get element i;
        if it ends with an 's', remove it.
    }
}
```

- What does the algorithm do wrong?

<i>index</i>	0	1	2	3	4	5
<i>value</i>	"she"	"sells"	"seashells"	"by"	"the"	"seashore"
<i>size</i>	6					

## ArrayList of primitives?

- The type you specify when creating an `ArrayList` must be an **object** type; it cannot be a primitive type.

- The following is illegal:

```
// illegal -- int cannot be a type parameter
ArrayList<int> list = new ArrayList<int>();
```

- But we can still use `ArrayList` with primitive types by using special classes called *wrapper* classes in their place.

```
ArrayList<Integer> list = new ArrayList<Integer>();
```

## Wrapper classes: Example

- Every java primitive has a class dedicated to it.

Example:

```
int x = 3;
Integer y = new Integer(5);

int z = x + y;

int z = x + y.intValue(); // convert wrapper to primitive

// can also construct an Integer from a string:

y = new Integer("5");
```



## ArrayLists of wrapper type objects

Primitive Type	Wrapper Type
int	Integer
double	Double
char	Character
boolean	Boolean
float	Float

- A wrapper is an object whose purpose is to hold a primitive value and to provide more functionality.
- Once you construct the list, use it with primitives as normal (autoboxing):

```
ArrayList<Double> grades = new ArrayList<Double>();
grades.add(3.2);
grades.add(2.7);
```

## ArrayLists of wrapper type objects

- Autoboxing:

```
ArrayList<Double> grades = new ArrayList<Double>();
// Autoboxing: create Double from double 3.2
grades.add(3.2);
grades.add(2.7);
double sum = 0.0;
for (int i = 0; i < grades.size(); i++) {
    //AutoUNboxing from Double to double
    sum += grades.get(i);
}
...
```

---

## Java Collections

- ArrayList belongs to Java's Collections framework.
  - Other classes have a very similar interface, so it will be easier to learn how to use those classes once you've learned ArrayList
- 

---

## Looking ahead: Interfaces

- A Java **interface** specifies which public methods are available to a user
  - A class **implements** an interface if it provides all the methods in the interface
  - Interfaces allow for common behavior amongst classes. Example: the **List** interface is implemented by several Collections classes (LinkedList, ArrayList, Vector, Stack)
-