

# ArrayLists

---

---

# Using arrays to store data

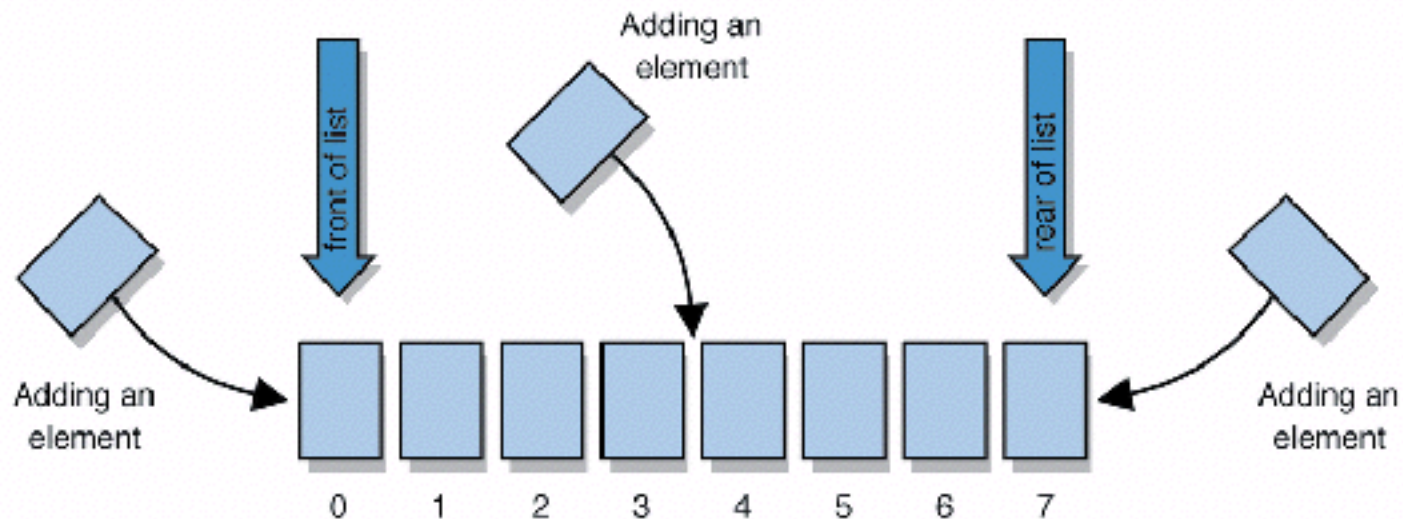
- ❑ Arrays: store multiple values of the same type.
- ❑ Conveniently refer to items by their index
- ❑ Need to know the size before declaring them:

```
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



# Exercise

- Let's write a class called `ArrayIntList` that implements 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

# Using ArrayList

- construction

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

- storing a value

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

- retrieving a value

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

- searching for the value 27

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

# Pros/cons of `ArrayIntList`

- pro (benefits)
  - ❑ simple syntax
  - ❑ don't have to keep track of array size and capacity
  - ❑ has powerful methods (`indexOf`, `add`, `remove`, `toString`)
- con (drawbacks)
  - ❑ `ArrayIntList` only works for `ints` (arrays can be any type)
  - ❑ Need to learn

# Java Collections and ArrayLists

- Java includes a large set of powerful **collections** classes.
- The most basic, `ArrayList`, is 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(<b>value</b>)</code>	appends value at end of list
<code>add(<b>index</b>, <b>value</b>)</code>	inserts given value at given index, shifting subsequent values right
<code>clear()</code>	removes all elements of the list
<code>indexOf(<b>value</b>)</code>	returns first index where given value is found in list (-1 if not found)
<code>get(<b>index</b>)</code>	returns the value at given index
<code>remove(<b>index</b>)</code>	removes/returns value at given index, shifting subsequent values left
<code>set(<b>index</b>, <b>value</b>)</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(<b>list</b>)</code>	adds all elements from the given list at the end of this list
<code>addAll(<b>index</b>, <b>list</b>)</code>	inserts the list at the given index of this list
<code>contains(<b>value</b>)</code>	returns true if given value is found somewhere in this list
<code>containsAll(<b>list</b>)</code>	returns true if this list contains every element from given list
<code>equals(<b>list</b>)</code>	returns true if given other list contains the same elements
<code>remove(<b>value</b>)</code>	finds and removes the given value from this list
<code>removeAll(<b>list</b>)</code>	removes any elements found in the given list from this list
<code>retainAll(<b>list</b>)</code>	removes any elements not found in given list from this list
<code>subList(<b>from</b>, <b>to</b>)</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

# 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 onwards

In version 7 of Java and later, rather than doing:

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

You can save a few keystrokes:

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

# Modifying a 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?

index	0	1	2	3	4	5
value	"she"	"sells"	"seashells"	"by"	"the"	"seashore"
size	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

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

- 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);
```

# Wrapper classes - continued

- 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);  
}  
...
```