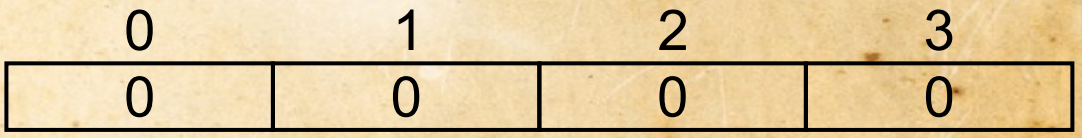
# Java Collections Hierarchy

## class ArraysList

# Lecture Contents

- Review of Arrays
- Limitations of Arrays

# What is an Array



# Declaring an Array

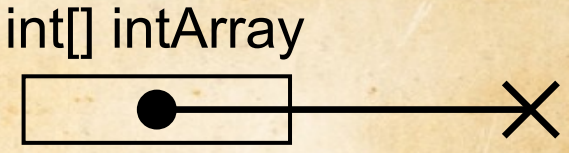
- An array is a collection of items of same data type stored at contiguous memory locations.
- An array of a type is declared by appending open and close square bracket, `[]`, after the type.

```
int[] intArray;
double[] doubleArray;
String[] bunchOfStrings;
```

# Allocating Memory for an Array

- Declaring an array only creates the space to store the ***reference*** to the array, it does not allocate memory to store the array.

```
int[] intArray;
```

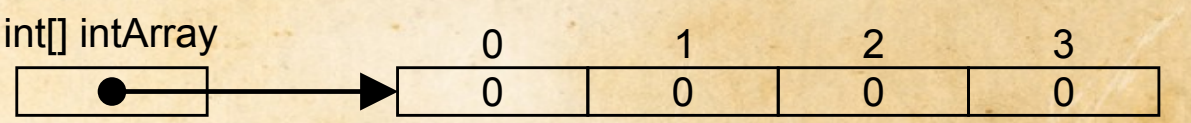


# Allocating Memory for an Array

- To allocate memory, we use the keyword `new`.

```
int[] intArray;
intArray = new int[4];
```

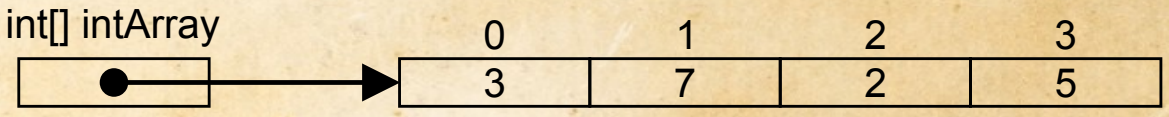
- `new` will allocate the array and initialize all values to zero.



# Allocating Memory for an Array

- We can also initialize an array with comma-separated sequence of elements enclosed in curly braces:

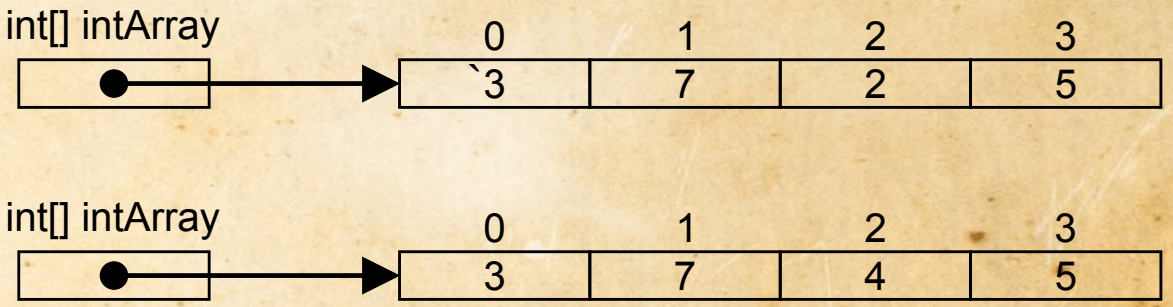
```
int[] intArray = { 3, 7, 2, 5 };
```



# Accessing Arrays

- We use the *index* to access the array:

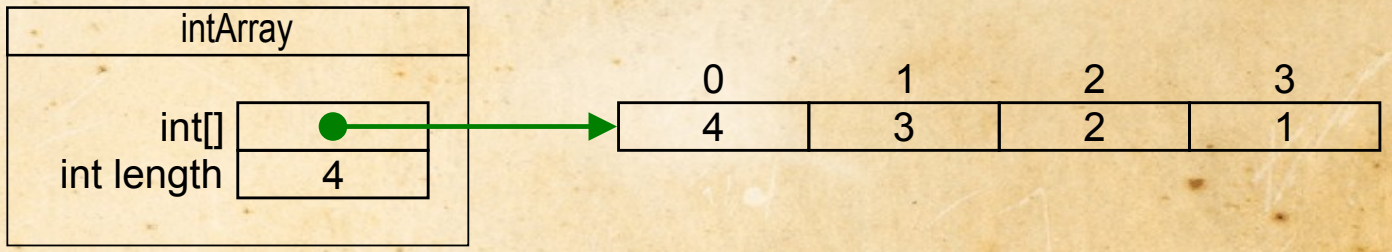
```
int[] intArray = { 3, 7, 2, 5 };
intArray[2] = 4;
```



# Finding the Length of an Array

- We use the `.length` field when we need to find the length of the array:

```
int[] intArray = { 3, 7, 2, 5 };
for(int i = 0; i < intArray.length; i++) {
   intArray[i] = 4 - i;
}
```



# Limitations of Arrays

- Arrays size is set when the array is instantiated.

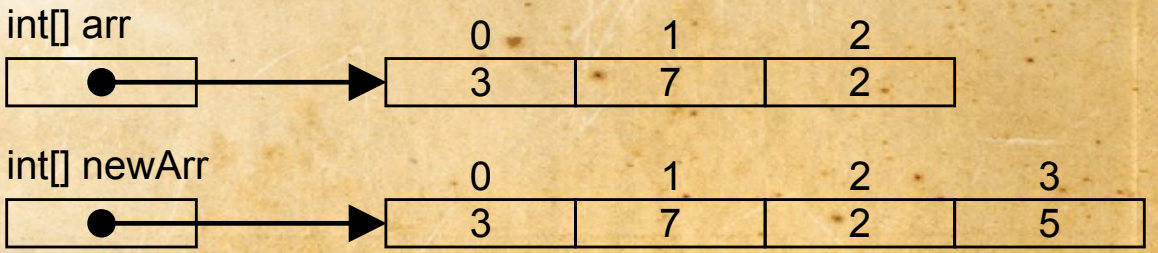
```
int[] intArray;
intArray = new int[4];
```


- The only way to add an additional element to the array is to create an entirely new array and copy the contents into the new array.

# Limitations of Arrays

- The only way to add an additional element to the array is to create an entirely new array and copy the contents into the new array.

```
public static int[] expandByOne(int[] arr, int value) {
    int[] newArr = new int[arr.length + 1];
    for(int i = 0; i < arr.length; i++) {
        newArr[i] = arr[i];
    }
    newArr[arr.length] = value;
    return newArr;
}
```



# Limitations of Arrays

- The only way to add an additional element to the array is to create an entirely new array and copy the contents into the new array.
- The only way to shorten an array is to create  an entirely new array and copy the desired contents into the new array.
  - The code for this is left as an exercise to the learner.

# class ArrayList

- A class that contains an array that automatically extends when needed.
- The type of object is declared using Java generic – a parameter enclosed in angle brackets rather than parenthesis.
  - Declaring and initializing a new `ArrayList` named list:

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

  - The second type is optional as of Java 7, so simplified:

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

# class ArrayList

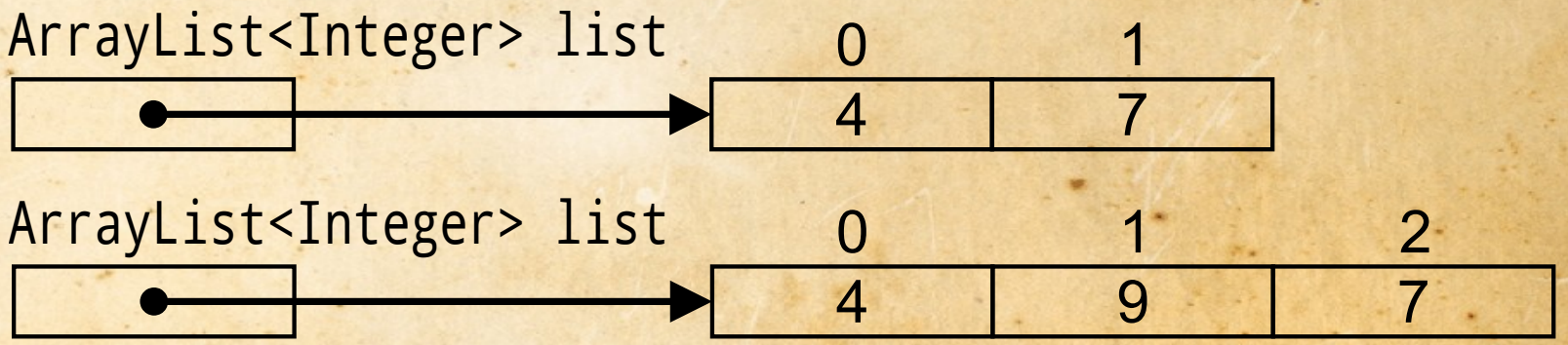
- The list is initially empty.
- **Adding** elements to the **end** of the list:

```
ArrayList<Integer> list = new ArrayList<>();
list.add(4);
list.add(7);
```

# class ArrayList

- **Inserting** elements into the list at an **index**:

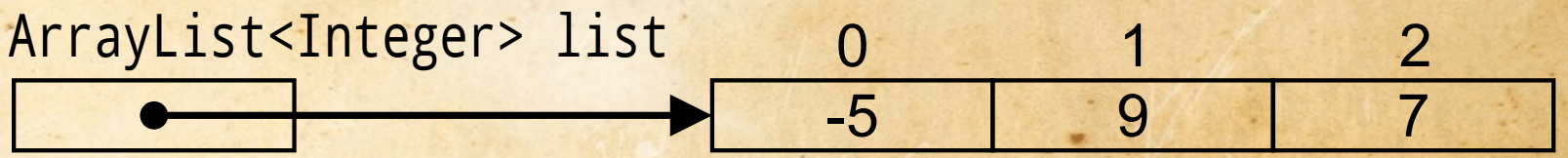
```
ArrayList<Integer> list = new ArrayList<>();
list.add(4);
list.add(7);
list.add(1,9);
```



# class ArrayList

- **Changing the value** of an element in the list:

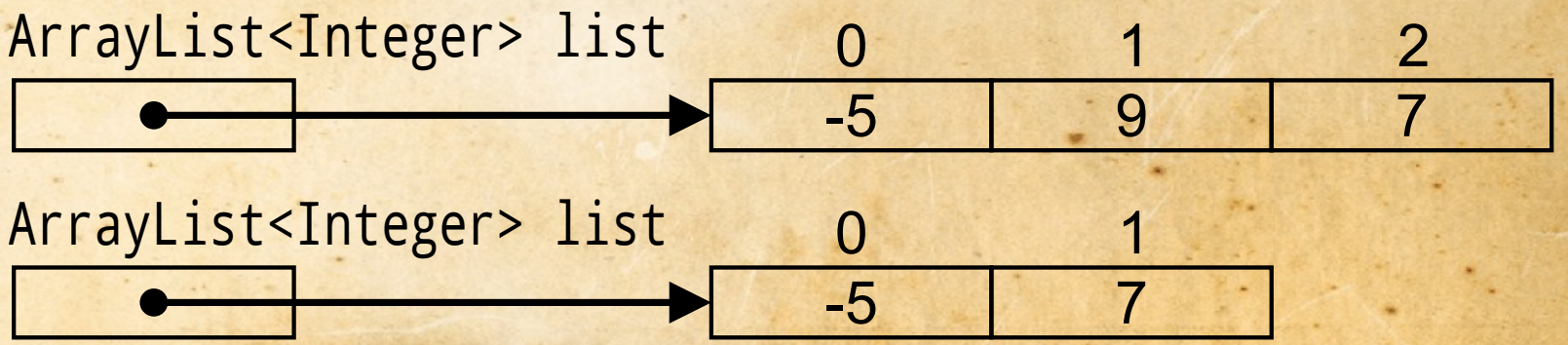
```
ArrayList<Integer> list = new ArrayList<>();
list.add(4);
list.add(7);
list.add(1,9);
list.set(0,-5);
```



# class ArrayList

- **Deleting** an element from the list (given an **index**):

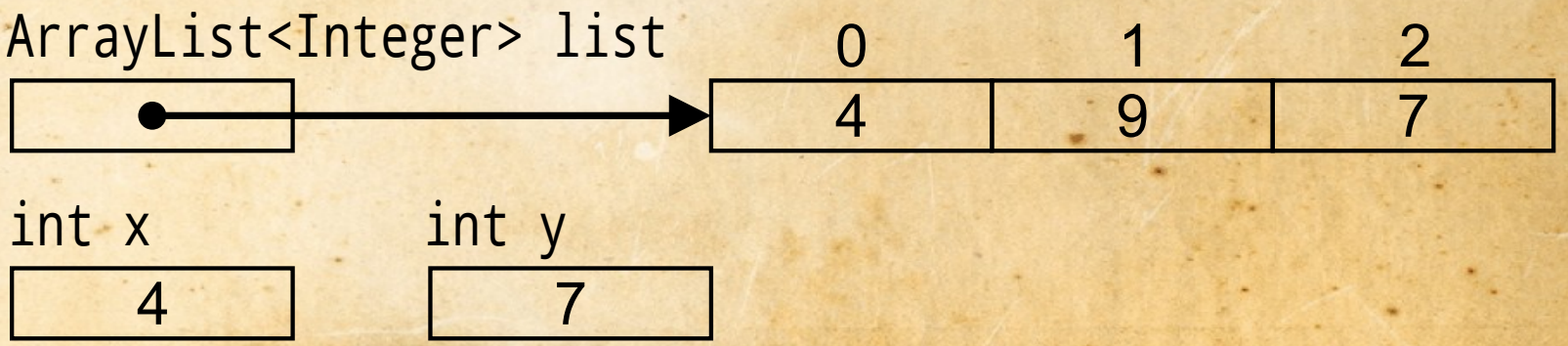
```
ArrayList<Integer> list = new ArrayList<>();
list.add(4);
list.add(7);
list.add(1,9);
list.set(0,-5);
list.remove(1);
```



# class ArrayList

- **Reading** elements at an index from a list:

```
ArrayList<Integer> list = new ArrayList<>();
list.add(4);
list.add(7);
list.add(1,9);
int x = list.get(0);
int y = list.get(2);
```



| Method | Description |
| --- | --- |
| `E` **`get`**`(int index)` | Returns the element at position `index` in the list |
| `boolean` **`add`**`(E obj)`<br>`void` **`add`**`(int index, E obj)` | If no `index` is given, the element, `obj`, is appended to the end of the list and `true` is returned. If an index is given, the element, `obj`, is inserted into the list at the index given by `index`, shifting all elements starting from that index one position to the right. |
| `E` **`set`**`(int index, E obj)` | Replaces the element at position `index` with the element `obj`; returns the element that was previously at that position. |
| `E` **`remove`**`(int index)` | Removes the element from position `index`, shifting all elements after that element to the left by one index position. |
| `int` **`size`**`()` | Returns the number of elements in the list. |

# class ArrayList

- ArrayList and for loop:

```
ArrayList<Vector> list = new ArrayList<>();
list.add(new Vector(1.1,2.2));
list.add(new Vector(3.3,4.4));
for(int i = 0; i < list.size(); i++) {
    System.out.println(list.get(i).getX());
}
```

```
1.1
3.3
```

# class ArrayList

- ArrayList and an enhanced for loop:

```
ArrayList<Vector> list = new ArrayList<>();
list.add(new Vector(1.1,2.2));
list.add(new Vector(3.3,4.4));
for(Vector v : list) {
    System.out.println(v.getY());
}
```

```
2.2
4.4
```

# Java Collections Hierarchy

## class ArraysList