ArrayLists

Review

- An **array** is a block of memory that allows us to store a list of related data
- **Arrays** can store primitive data (int, double, boolean) and references to objects (String, any objects you make from a class)

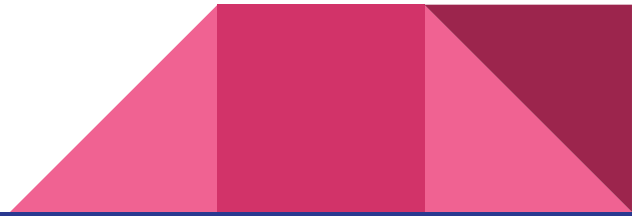
```
int[ ] array = {10, 20, 30};
String[] words = {“hello”, “cat”, “dog”};
```

- **Array limitation:**
  - the size of the array is set in stone on initialization
  - difficult to add more to the end of the array and delete elements from the array

ArrayLists

- **ArrayLists** are resizable arrays. These are sometimes referred to as lists (especially in other programming languages).
- **ArrayLists** must hold object data. It cannot hold primitive data (int, double char or boolean)

  We must use the wrapper classes. For int, we use Integer. For double, we use Double. For boolean, we use Boolean.
- **ArrayLists** have a number of methods that can be used to add, insert, delete and reorganize the data stored in them.



Declaring and Initializing ArrayLists

- To use an **ArrayList**, we need to import the library that provides access to the **ArrayList** class and its methods At the top of your program, you need to have this statement

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

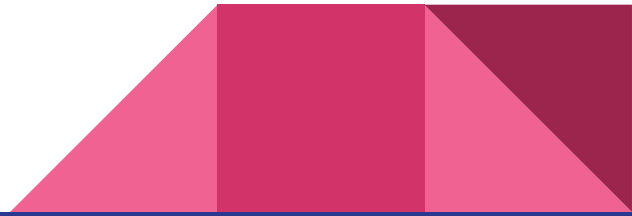
- The syntax to declare and initialize an ArrayList is as follows:

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

```
import java.util.ArrayList; // you may use java.util.*

// declaring and initializing an Arraylist
// that will hold integers
ArrayList<Integer> myList = new ArrayList<Integer>();
```

Common methods

1. void add(int index, TYPE value)
2. boolean add(TYPE value)
3. TYPE get(int index)
4. TYPE remove(int index)
5. boolean remove(TYPE value)
6. TYPE set(int index, TYPE value)
7. int size()

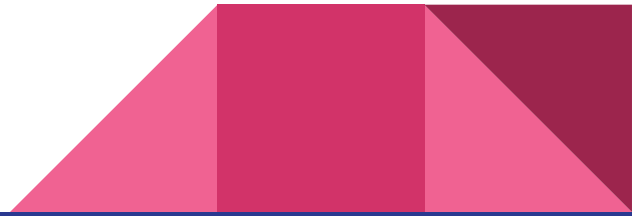


Comparing Arrays and ArrayLists

When to use a List or an Array?

Use an **array** when you want to store several items of the **same type** and **you know how many items** will be in the array and the items in the array won't change in order or number.

Use a **list** when you want to store several items of the **same type** or not and **you don't know how many items** you will need in the list or when you want to **remove** items from the list or **add** items to the list.



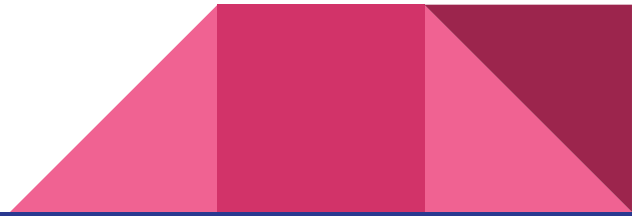
Comparing Arrays and ArrayLists

	Array	ArrayList
Declare	int[ ] highScores; String[ ] names;	ArrayList<Integer> highScoreList; ArrayList<String> nameList;
Create	int[ ] highScores = new int[5];	ArrayList<Integer> highScoreList = new ArrayList<Integer>();
Setting the value at an index	highScores[0] = 80;	highScoreList.set(0, 80);
Getting the value at an index	int score = highScores[0];	int score = highScoreList.get(0);
Getting the number of items	System.out.println(highScores.length);	System.out.println(highScoreList.size());

Initialize an ArrayList with elements

```
ArrayList<Integer> nums = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5));

System.out.println(nums);
```

Output: [1, 2, 3, 4, 5]



Adding Elements to an ArrayList

- Requires the syntax `arrayListName.add(value);`

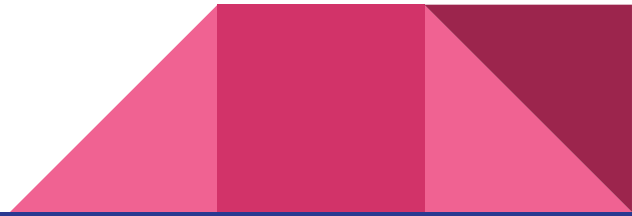
```
ArrayList<String> names = new ArrayList<String>();
names.add("Hume");
names.add("Locke");
names.add("Hegel");
System.out.println(names);
```

```
Output:
[Hume, Locke, Hegel]
```

- Adding elements to an ArrayList is slightly different than placing elements in an array

Traversing ArrayLists with Loops

1. Enhanced For Each Loop
2. For Loop
3. While Loop



Enhanced For Each Loop

You can use a enhanced for-each loop to traverse through all of the items in a list, just like you do with an array.

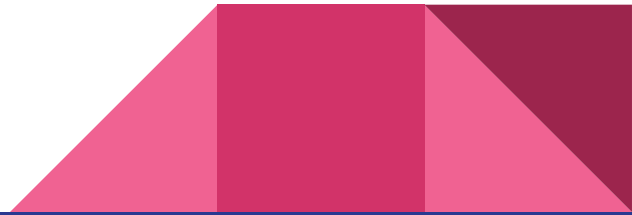
```
import java.util.ArrayList;

public class EnhancedForEachLoop
{
    public static void main(String[] args)
    {
        ArrayList<Integer> myList = new ArrayList<Integer>();
        myList.add(50);
        myList.add(30);
        myList.add(20);
        int total = 0;
        for (Integer value: myList)
        {
             total += value;
        }
        System.out.println("Sum of all elements: " + total);
    }
}
```

For Loop

For Loop and While Loop process list elements using the index.

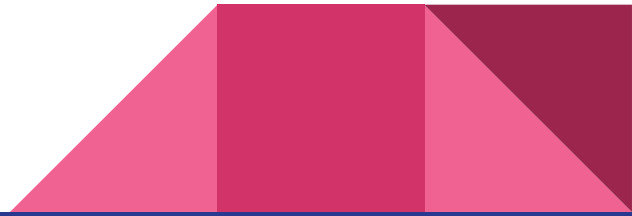
The ArrayList index starts at 0 just like arrays, but instead of using the square brackets [ ] to access elements, you use the get(index) to get the value at the index and set(index, value) to set the element at an index to a new value.

If you try to use an index that is outside of the range of 0 to the number of elements − 1 in an ArrayList, your code will throw an ArrayIndexOutOfBoundsException, just like in arrays.



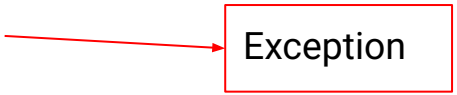
For Loop

```
import java.util.ArrayList;

public class ForLoop
{
    public static void main(String[] args)
    {
        ArrayList<Integer> myList = new ArrayList<Integer>();
        myList.add(50);
        myList.add(30);
        myList.add(20);
        int total = 0;
        for (int i=0; i < myList.size(); i++)
        {
            total = total + myList.get(i);
        }
        System.out.println(total);
    }
}
```



While Loop

```
import java.util.ArrayList;

public class Test{
     public static void main(String[] args){
          ArrayList<Integer> myList = new ArrayList<Integer>();
          myList.add(30);
          myList.add(40);
          myLst.add(20);
          int total = 0;
          int i=0;
          try{
               while (i <= myList.size()){
                    total += myList.get(i);
                    i++;
               }
          }catch(IndexOutOfBoundsException e){
               System.out.println("Error: index out of bounds");
          }
          System.out.println(total);
     }
}
```



ArrayList of any Object

You can have any kind of Objects in an ArrayList

```
class Student
{
  private String name;
  private String email;
  private int id;

  public Student(String name, String email, int id)
  {
        this.name = name;
        this.email = email;
        this.id = id;
  }

  public String toString()
  {
        return id + ": " + name + ", " + email;
  }
}
```

```
import java.util.ArrayList;

public class StudentList
{
  public static void main(String[] args)
  {
        ArrayList<Student> roster = new ArrayList<Student>();
        roster.add(new Student("Sophie", "sophie@myschool.com", 123456));
        roster.add(new Student("Anais", "anais@myschool.com", 789012));
        for (Student student: roster){
              System.out.println(student);
        }
  }
}
```

Example: Remove

Assume that nums is an ArrayList with these values [0, 0, 4, 2, 5, 0, 3, 0]. What will nums contain after executing the following code:

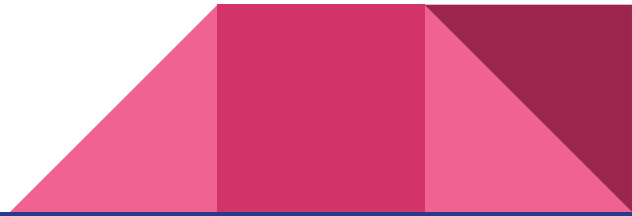
```
public static void numQuest(ArrayList<Integer> nums)
{
   int k = 0;
   int zero = 0;
   while (k < nums.size())
   {
      if (nums.get(k).equals(zero))
         nums.remove(k);
      k++;
   }
   System.out.println(nums);
}
```

nums will contain:
[0, 4, 2, 5, 3]

Warning!!!

- Be careful when you remove items from a list as you loop through it. Remember that removing an item from a list will shift the remaining items to the left.

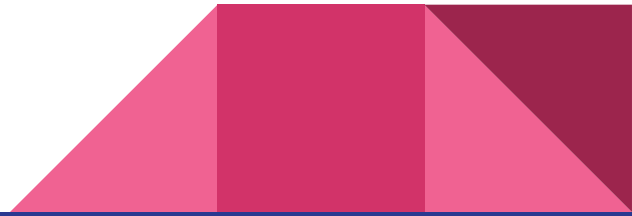
- Do not use the enhanced for each loop if you want to add or remove elements when traversing a list because it will throw a ConcurrentModificationException error.



Remove: index or element?

```
ArrayList<Integer> numbers = new ArrayList<>();
numbers.add(1);
numbers.add(2);
numbers.add(3);
numbers.add(4);
numbers.remove(1);
```

What would be removed? The index or the element?



Remove: index or element?

Remove by index:

numbers.remove(1);

Remove by element:

numbers.remove(Integer.valueOf(1));
numbers.remove((Integer) 1);