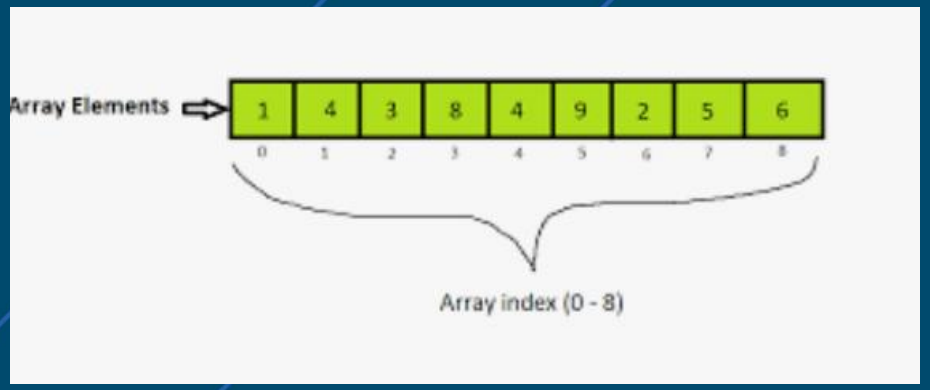
Féidearthachtaí as Cuimse
Infinite Possibilities



Array vs ArrayList

Object Oriented programming

OLLSCOIL TEICNEOLAÍOCHTA
BHAILE ÁTHA CLIATH
DUBLIN
TECHNOLOGICAL
UNIVERSITY DUBLIN

A constant requirement in programming:

- To be able to hold a collection of “data “ to be used by your class(es)
  - An application to find a bus route for your journey – and display it as points on a map
  - An application to read text files to find the number of matching words.
  - An application to display a list of countries and allow you to click on one to find out tourism details etc

Basic Arrays

An **array** in Java is a **container object** that holds a **fixed number of values** of a **single type**. Each element is stored in a **contiguous block of memory** and can be accessed by its **index** (position).

- Think of it like a row of boxes:
  - `String[]numbers = {“one”, “two”, “three”};`
  - What happens if you need more entries than you have allocated?

    E.g. Add entry “four”.
  - `Student[] students = new Students[4];`

    Length (i.e number of entries) is declared up front!

ArrayList

Flexible size (initially 20) expands or shrinks automatically

Methods to add / remove/ set etc

Can take any type of object

Provides many **built-in methods** (add, remove, get, set, size, etc.).

Syntax: `ArrayList<Class> = new ArrayList<Class>;`

```
// ArrayList example

import java.util.ArrayList;

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

Array versus ArrayLIst

- Arrays can store primitive types or reference types (i.e. objects)*

  Student[] students = new Student[4];

- ArrayList can only store objects (same type)

- ArrayLists can expand flexibly

* Note: Primitive versus reference types..!

Java API

The **Java Application Programming Interface (API)** is a library of **pre-built classes and methods** provided by Java.

It helps developers reuse standard functionality
(e.g. ArrayList, Scanner, Math, String).

Making ArrayList Available:

- The class is located in **the java.util package.**
- To use it, **import** the class at the top of your file:

Creating an array List

ArrayList can only store objects (not primitive data types)

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

```
ArrayList<Animal> animalList = new ArrayList<Animal>();
```

```
ArrayList<CurrentAccount> accountList = new
ArrayList<CurrentAccount>();
```

Can specify the initial capacity -

```
ArrayList<String> list = new ArrayList<String>(100);
```

Some methods of class `ArrayList` *

- constructors:

```
public ArrayList<Base_Type> ( int initialCapacity)

public ArrayList<Base_Type> ()
```

- Things an ArrayList can *do*..

Add a new element to the **end** of the array

```
public void add(Base_Type newElement);
```

Add a new element somewhere in the ArrayList (**based on index)**

```
public void add(int index, Base_Type newElement);
```

Returns the element at a particular index position

```
public Base_SType get(int index)
```

What else? e.g. Delete an entry? Check if it contains a particular entry? Length of it? etc? – look at the API

Class Demo

Class ArrayList<E>

java.lang.Object
  java.util.AbstractCollection<E>
    java.util.AbstractList<E>
      java.util.ArrayList<E>

Type Parameters:

E - the type of elements in this list

All Implemented Interfaces:

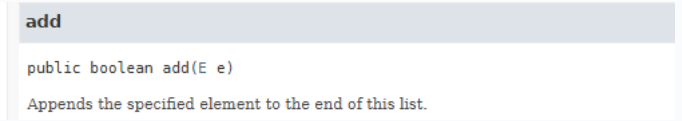
Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

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

list.add("Alice");

list.add("Bob");

System.out.println(list.size()); // prints 2
```



add

```
public boolean add(E e)
```

Appends the specified element to the end of this list.

Why does the `ArrayList` use **element (E)** and not **object (Object)**?

Looping through an ArrayList

```
For (String element:  namesList)

 system.out.println(element)
```

Maybe, easier to write than:

```
for (int i = 0; int <
 namesList.size(); i++)
{    ….
```

Copying an ArrayList ?

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

Populate it etc..

If you execute this:

```
ArrayList secondList = aList;
```

How many ArrayLists do you now have?

Copying an ArrayList

Ans: One (but you have two names for it) **only one ArrayList object exists**, but there are **two reference variables** pointing to it.

Need to use clone() method to a second copy of the arrayList and its contents:

```
ArrayList secondList = aList.clone();
```

Covered:

- Arrays
- Difference to ArrayLists
- ArrayList behaviour (i.e. methods you can call on an arrayList)
- Iterating