Object-Orientated Programming

CMM024

RGU (2023)

Session 4 (Procedural Programming)

Introduction to Collections:

Arrays

ArrayLists

CMM024

Summary

- **What is an Array**
- **How we create and use arrays**
- **Why we use arrays (Algorithms)**
- **What is an ArrayList**
- **How we create and use array lists**
- **Why we use arraylists (Algorithms)**
- **Next Session**

Arrays: Declaration

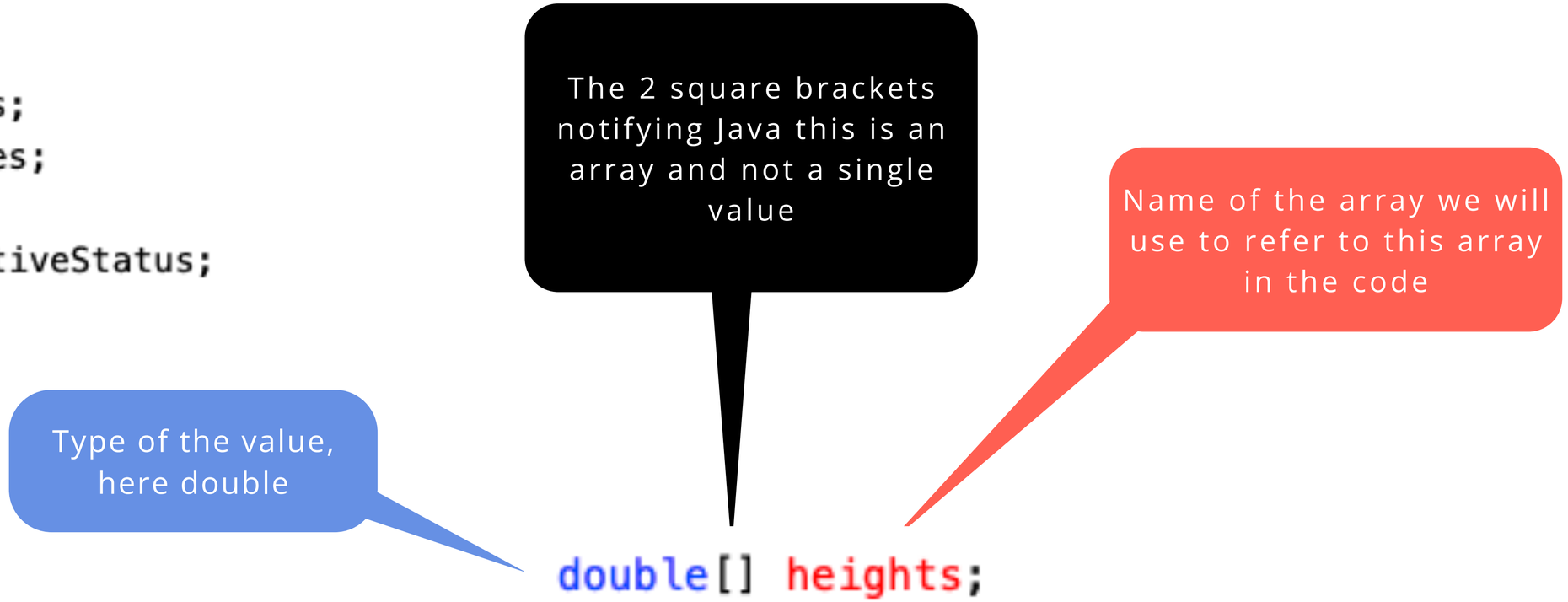
- Arrays are a collection of things
  - Collection of primitive values (variables) like int, double, float, String
  - A collection of Objects
- Arrays are used when you know:
  - the amount of values you want to store.
  - The data type of the values to store (int, double, float, String etc.)
- Before you can use an array, Java needs to know about this array
  - You MUST declare this array first!

Note: an array can store many types of objects, from primitives to more complex objects like Shapes, Connection etc. We will focus here on primitives only I.e. int, double, float, String, boolean.

- String is a primitive but it is actually a special object!

Arrays: Declaration

```
double[] heights;
float[] prices;
int[] ages;
char[] studentGrades;
String[] studentNames;
Shape[] usedShapes;
boolean[] studentActiveStatus;
```



Arrays: Declaration

- In the previous slide we declared the array using this syntax

```
double[] heights;
```

- However this is an alternative that does exactly the same, where the brackets are located after the array name

```
double heights[];
```

- Which one you want to use depends of the next declaration, below we declare 3 **arrays** of double values

```
double[] heights, prices, ages.
```

- Below we declare an array **heights**, a double single variable **prices** and a double single variable **ages**

```
double heights[], prices, ages.
```

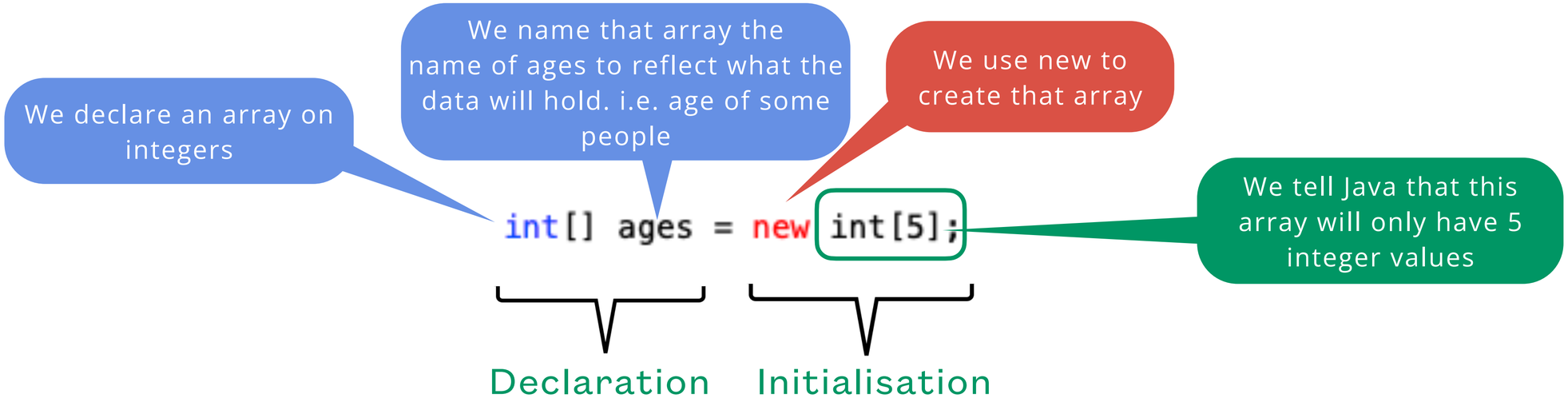
Arrays: Instantiation

- In the previous slide we declared the array using this syntax

```
int[] ages;
```

- Now we need to tell java how many values to create. We call this the length of the array. We use the keyword new to do this. Here we are creating an array of integer values that can only contain 5 values to store the age of some individuals

```
int[] ages = new int[5];
```



Arrays: Declaration

Declaration	Notes	Initial values
`int[] ages = new int[5];`	An array of 5 integers. We use a static value (5) for the length. 5 is called a Magic Number. It is not advised to do that.	0
`final int ARRAYLENTH = 202;` `double[] heights = new double[ARRAYLENTH];`	Here we want to store 202 double values that are human heights. Instead of using a magic number, we declare a constant (final) value storing the number of values that will be storedin the array	0.0
`int ARRAYLENTH = in.nextInt();` `double[] heights = new double[ARRAYLENTH];`	Instead of using a constant, we can use the Scanner class to get an integer value the can be used to set array length	0.0
`double[] prices` `= {28.99, 4.99, 1.25, 20,50.70};`	If your array contains fewer values, we can initialise it like this using curly brackets. Hence instead of having 0 as initial values, the array will contain real values	Values inside {}
`double[] weights = new int[5];`	Here we are declaring a array to hold doubles but we create it using integers. Note the red underline, meaning there is an error!	N/A

Arrays; an example

```
double x[] = new double[5];
    x[0] = 141;
    x[1] = 183.5;
    x[2] = 191.3;
    x[3] = 182.3;
    x[4] = 184.3;
```

Array declaration

Initialising the array with values. Storing the values into the array

If you have small amount of values you can initialise array like this

```
double y[] = new double[]{141,183.5,191.3,182.3,184.3};
```

LET'S CODE

Arrays, example

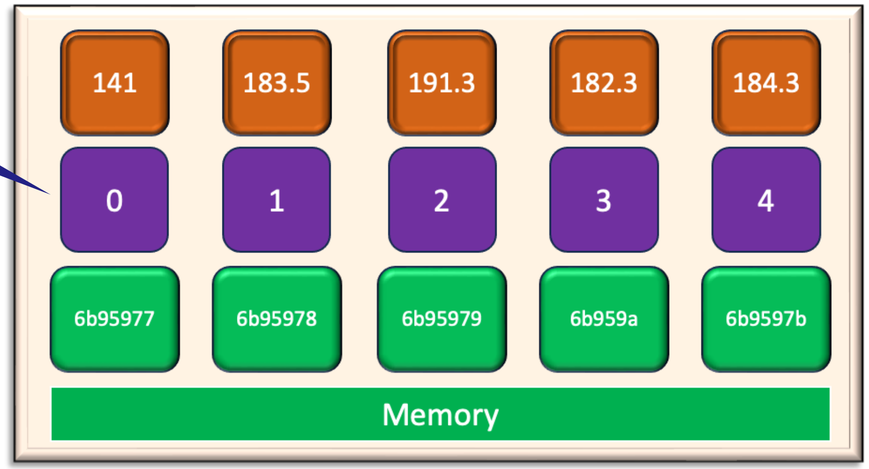
- You want to store Male height decimal values into an arrays for analysis
- We know the type (double) and the number of values (5)

	A	B
1	Sample	Height
2	1	141
3	1	183.5
4	1	191.3
5	1	182.3
6	1	184.3

DATA

```
public class ArrayDemo1 {
    public static void main(String[] args){
        int arrayLength = 5;
        double[] heights = new double[arrayLength];
        heights[0] = 141;
        heights[1] = 183.5;
        heights[2] = 191.3;
        heights[3] = 182.4;
        heights[4] = 184.3;
        double[] myHeights = {141, 183.5,  191.3, 182.4, 184.4 };
}
```

ARRAYS INDEXING ALWAYS STARTS AT ZERO



ARRAY VALUES

ARRAY INDEX

MEMORY ADRESSES

```
System.out.println("Memory location: " + heights);
```

```
dc393f5c38c54d508e00eff0a5/redhat.java/jdt_ws/CMM024_C
Memory location: [D@7344699f
jean-claude@MacBook-Pro-2 CMM024 Code %
```

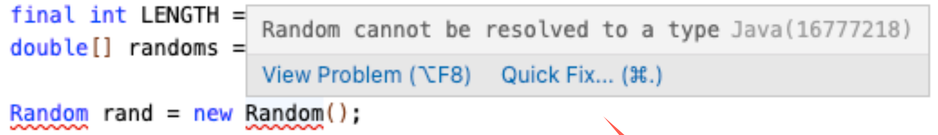
```
System.out.println("First value: " + heights[0]);
```

```
dc393f5c38c54d508e00eff0a5/redhat.java/jdt_ws/CMM024_C
First value: 141.0
jean-claude@MacBook-Pro-2 CMM024 Code %
```

Note: continuous memory block to store all the values

LET'S CODE

Arrays: Practice (1)



Click Fix and then import

You want to create 10 random decimal values and store them into an array.

- In Visual Studio Code we first create the array to store these values., but first create a class called ArrayTest

```
final int LENGTH = 10;
double[] randoms = new double[LENGTH];
```

- Then we create a Random object that will create these random values. You must import the library
  - ✳ Remember that Random generate value from 0 to 1 (1 not included) I.e. 0.0 <= rand < 1.0

```
Random rand = new Random();
```

- Then we can use a for loop to assign these random values into the array.

```
for (int i = 0; i < LENGTH; i++){
      randoms[i] = rand.nextDouble();
}
```

We ask rand to provide us with a random value

- We can use another for loop to display the values

```
for (int i = 0; i < LENGTH; i++){
      System.out.println(randoms[i]);
}
```

```
jean-claude@MacBook-Pro-2 CMN
ExceptionMessages -cp /Users/
de_186eb31c/bin ArrayTest2
0.07047163829681158
0.7929427821629763
0.2109440809383316
0.7412691683425184
0.5636492601122223
0.3977449302444209
0.49518452748703146
0.9632261712947783
0.96743502545501
0.07902588958324908
jean-claude@MacBook-Pro-2 CMN
```


LET'S CODE

Arrays, Practice (2)

```
jean-claude@MacBook-Pro-2 CMN
ExceptionMessages -cp /Users/
de_186eb31c/bin ArrayTest2
0.07047163829681158
0.7929427821629763
0.2109440809383316
0.7412691683425184
0.5636492601122223
0.3977449302444209
0.49518452748703146
0.9632261712947783
0.96743502545501
0.07902588958324908
jean-claude@MacBook-Pro-2 CMN
```

We have our 10 random decimal values (red). But how can we format the printout of these values?

- We can do it in two ways. We can change how the output is done or we can format the number before printing it out

2 decimals

```
for (int i = 0; i < LENGTH; i++){
    System.out.printf("%.2f %n",randoms[i]);
}
```

```
-full.jdk/Con
dc393f5c38c54
0.40
0.00
0.72
0.69
0.76
0.63
0.05
0.52
0.93
0.01
jean-claude@M
```

We first create a formate, which we use to format before printing it

3 decimals

```
DecimalFormat doubleFormater = new DecimalFormat("0.000");
for (int i = 0; i < LENGTH; i++){
    System.out.println(doubleFormater.format(randoms[i]));
}
```

```
-full.jdk/Con
dc393f5c38c54
0.001
0.978
0.087
0.370
0.124
0.703
0.675
0.416
0.627
0.090
jean-claude@M
```

What can we do to have randoms values from 0 to 10 instead of 0 to 1?

LET'S CODE

Arrays, Practice (3)

Create a new class called RandomTest, and copy the code listed on the right, so that you use a while loop instead of for loop to print 5 random values using printf, with 4 decimal digits

```
public class ArrayTest2 {

    public static void main(String[] args){

        final int LENGTH = 10;
        double[] randoms = new double[LENGTH];
        Random rand = new Random();
        for (int i = 0; i < LENGTH; i++){
            randoms[i] = rand.nextDouble();
        }
        for (int i = 0; i < LENGTH; i++){
            System.out.printf("%.2f %n",randoms[i]);
        }
        DecimalFormat doubleFormater = new DecimalFormat("0.000");
        for (int i = 0; i < LENGTH; i++){
            System.out.println(doubleFormater.format(randoms[i]));
        }
    }

}
```

LET’S CODE

ArraysList, what are they?

- Like Arrays, ArrayLists are a collection of values. The difference is:
  - ArrayList can grow or shrink as needed
  - The class ArrayList comes with numerous methods to perform common tasks like adding values, removing them etc.
  - ArrayLists cannot be used to store primitives (int, double, float, boolean) but can use be used with Integer, Double, Boolean, Float and String. Notice the capital letter! There are the Class form of the primitives, which provided extended functionality compared to primitives

```
ArrayList<String> people;
people = new ArrayList<>();
people.add("Jean");
people.add("John");
people.add("Catherine");
people.add("David");
System.out.println(people);
```

ArrayList declaration

ArrayList creation

ArrayList Initialisation

Printing content of the ArrayList

```
jean-claude@MacBook-Pro-2 CMM024_Code %
ExceptionMessages -cp /Users/jean-claude/
de_186eb31c/bin ArrayListTest
[Jean, John, Catherine, David]
jean-claude@MacBook-Pro-2 CMM024_Code % [
```

ArrayLists: Declaration

- ArrayList declaration is different from an Array.
  - ✳ You need to pass the data type between the **< >** brackets.
  - ✳ The brackets **< >** are called Diamond Operator

ArrayList	Array

`ArrayList<Integer> labCapacity;`
`ArrayList<Double> prices;`
`ArrayList<Boolean> status;`
`ArrayList<String> studentNames;`
	`int[] labCapacity;`
	`double[] prices;`
	`boolean[] status;`
	`String[] studentNames;`

Note that String is a Class (Uppercase S) and not a primitive but as already mentioned, it is a special type that is treated by Java as a primitive

ArrayLists: instantiating

- To instantiate an ArrayList we need to use the Diamond Operator
  - ✳ You need to pass the data type between the **< >** brackets on the left and leave it empty on the right.
- In older versions of Java you would pass the parameter on both sides!

ArrayList NEW	ArrayList OLD
`ArrayList<String> studentNames = new ArrayList<>();`	`ArrayList<String> studentNames = new ArrayList<String>();`

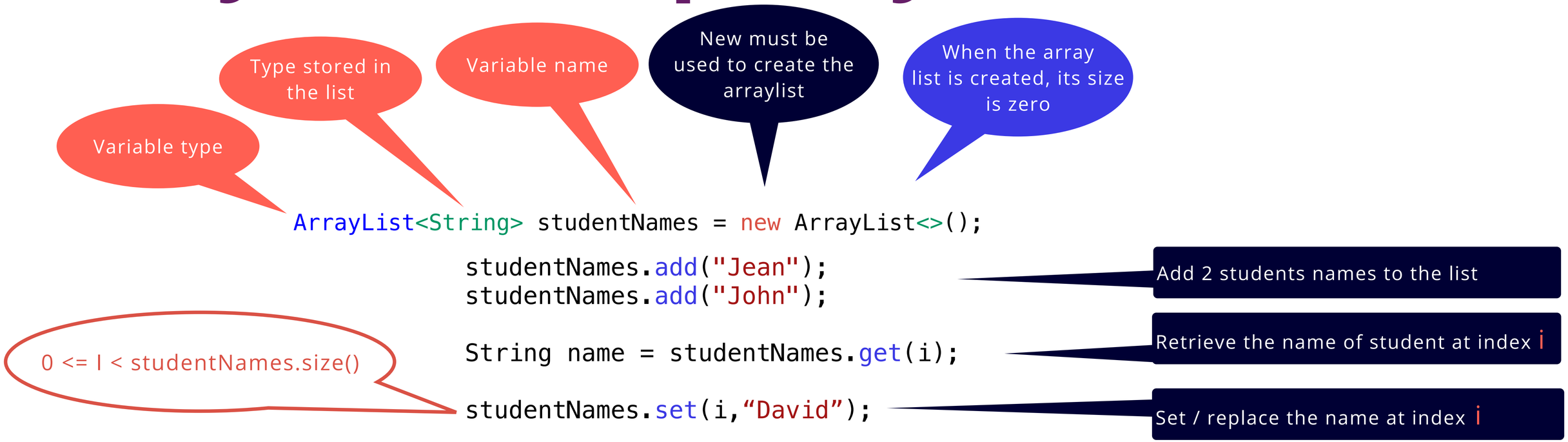
- As you can see we don't need the length as we do with Arrays, as the ArrayList will grow or shrink automatically

ArrayList	Array
`ArrayList<String> studentNames = new ArrayList<>();`	`String[] studentNames = new String[LENGTH];`

Nerd Alert: An ArrayList is what we call a Generic Type, which is a class that is parametrised over types. The chosen type here is String

```
public class ArrayList<T> {
    // T stands for "Type"
    private T t;

}
```

ArrayLists: Proper Syntax



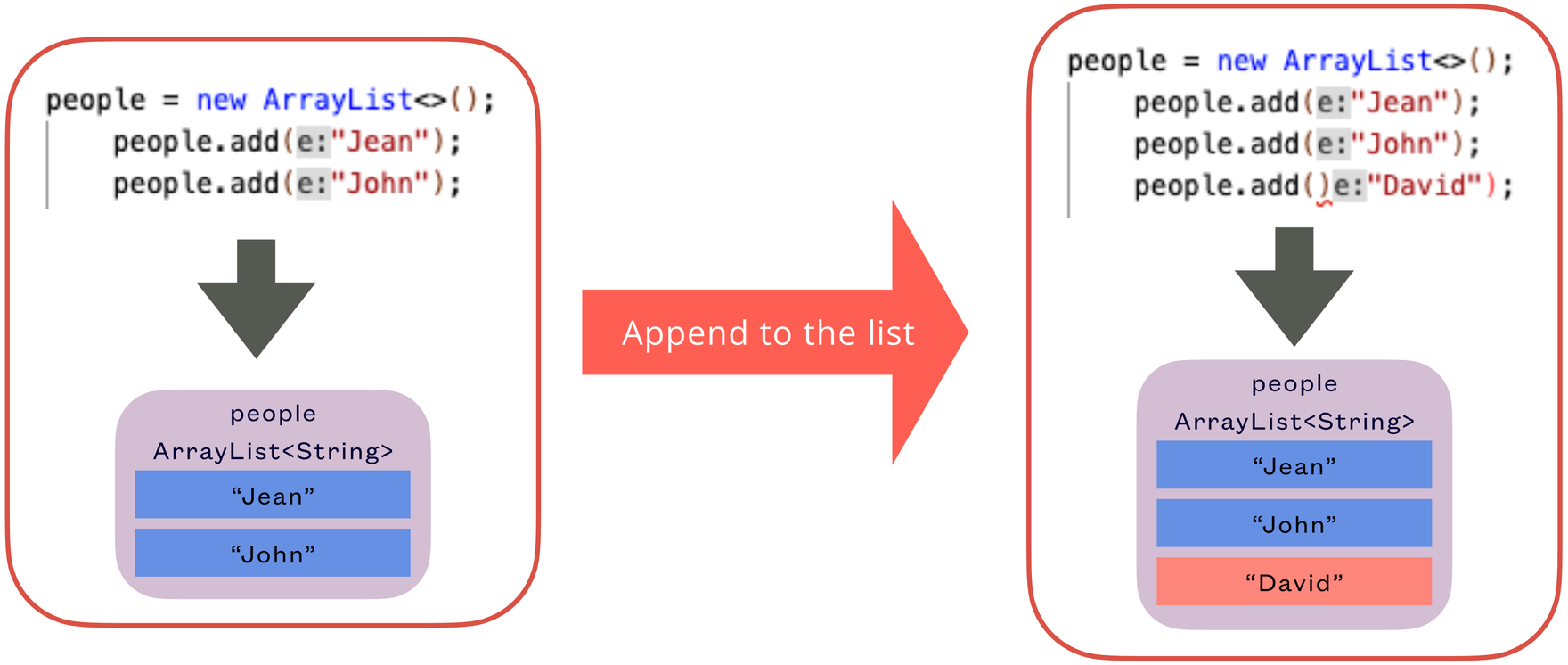
- There are some useful methods in the ArrayList class

add: add an element	remove: delete an element	size: current length
get: return an element	set: change an element	

ArrayLists: Adding an element

- To add an element to a ArrayList, you must use the method provided by the class
  - ✳ add(element) method will **append** the new element at the END of the list.

```
people.add("David");
```



ArrayLists: Adding an element

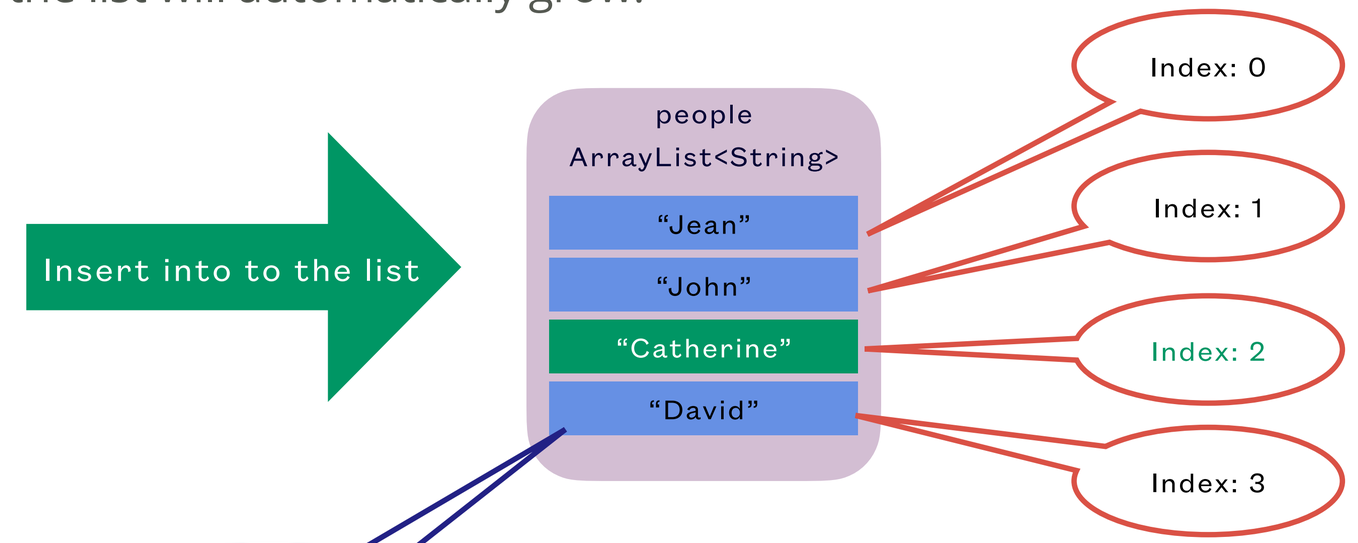
- There is another method that allows you to choose where to place an element
  - add(index, element) method will **insert** the new element in the location at the location referred by the index, Hence, all the existing elements will move to make some room. And the list will automatically grow.

```
people.add(2, ”Catherine”);
```

```
people = new ArrayList<>();
people.add("Jean");
people.add("John");
people.add(“David");
people.add(2,"Catherine");
```

Index:

Insert into to the list



David gets pushed down the

Visual Studio Code

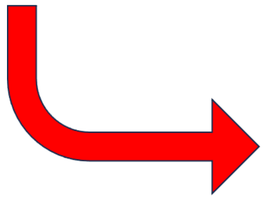
```
people = new ArrayList<>();
    people.add(e:"Jean");
    people.add(e:"John");
    people.add(e:"David");
    people.add(index:2,element:"Catherine");
```

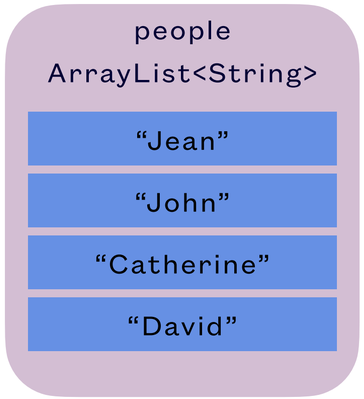
ArrayLists: Replacing an element

- There are times when you want to replace an existing element by another
  - set(index, element) method will **replace** an existing element referred by the index, The list size does not change as we are replacing an element
  - Must make sure the index is valid!

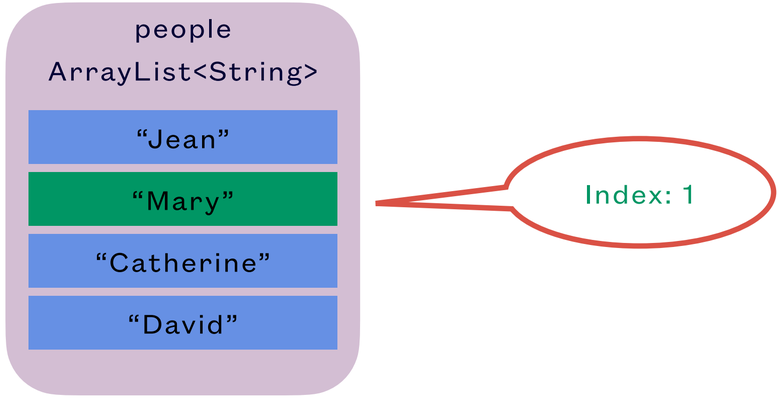
```
people.set(1, ”Mary”);
```

```
people = new ArrayList<>();
people.add("Jean");
people.add("John");
people.add(“David");
people.add(2,”Catherine");
```






```
people.set(1,"Mary");
```



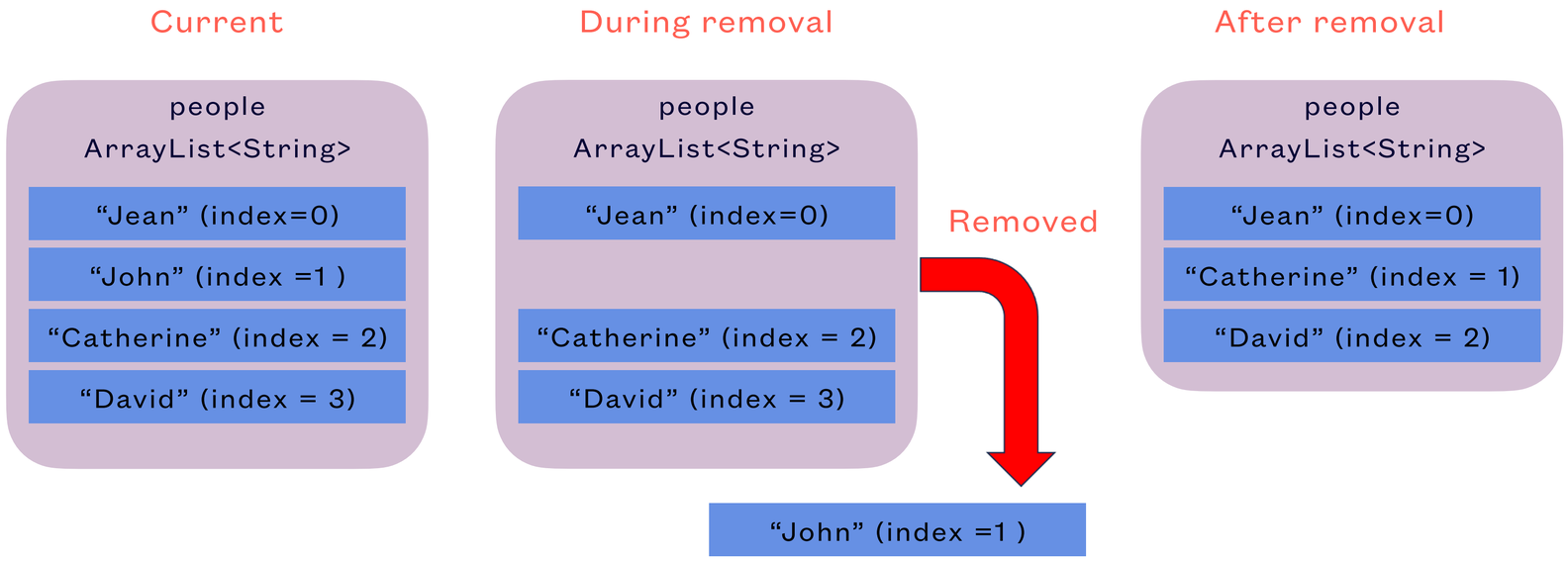
ArrayLists: Removing an element

- A method that is very useful is for removing an element from the list
  - remove(index) method will **remove** an element from the list referred by the index. The size of the list will automatically shrink.
  - Returns the element that was removed as a return value

```
people.remove(1);
```

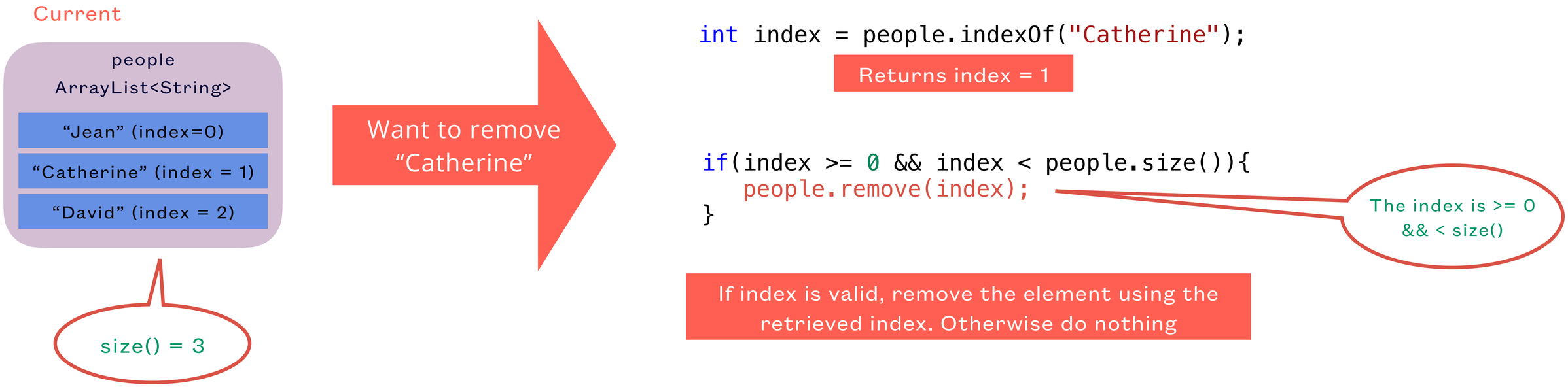
```
people = new ArrayList<>();
    people.add("Jean");
    people.add("John");
    people.add("David");
    people.add(2,"Catherine");

    people.remove(1);
```

Index: 1



ArrayLists: Removing an element

- At times you want to remove an element, not by using the index but by the actual value stored
  - You must find the index (using indexOf)
  - You must check the index is valid (if bigger or equal to 0 and smaller than size())
  - You must remove the element using the valid index



ArrayLists: Iterations

- There are may instances when you need to iterate through a list to perform some tasks. There are three main type of loops that allows you to do that.
    - Standard loops
    - For-each loops
    - Lambda Expressions

Very standard but you need to code quite a few lines

Much easier. No need to invoke any methods

For loop

```
for (int i = 0; i < people.size(); i++){
    String firstName = people.get(i);
    System.out.println(firstName);
}
```

For-each loop

```
for (String firstName: people){
    System.out.println(firstName);
}
```

More efficient both in terms of coding and speed

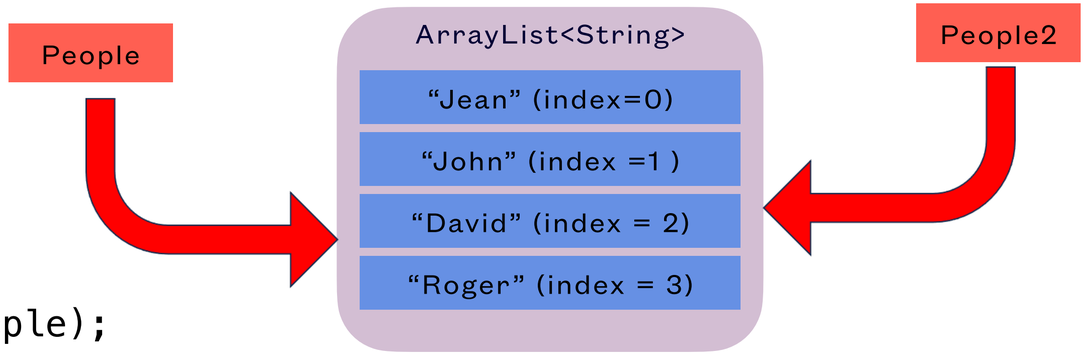
Lambda expression

```
people.forEach((firstName) -> System.out.println(firstName));
```

ArrayLists: Referencing (Practice1)

- There are many instances when you want to have a copy of an ArrayList.
  - You may want to have two ArrayList variables referring to the same list

```
public static void main(String[] args){
    ArrayList<String> people;
    people = new ArrayList<>();
        people.add("Jean");
        people.add("John");
        people.add("David");
        people.add("Roger");
        System.out.println("people : " + people);
 }
```




```
ArrayList<String> people2 = people;
System.out.println(people2);
```

**** Output ****

```
c44fdc393f5c38c54d508e00ef70a5/redhat
people : [Jean, John, David, Roger]
jean-claude@MacBook-Pro-2 CMM024 Code
```

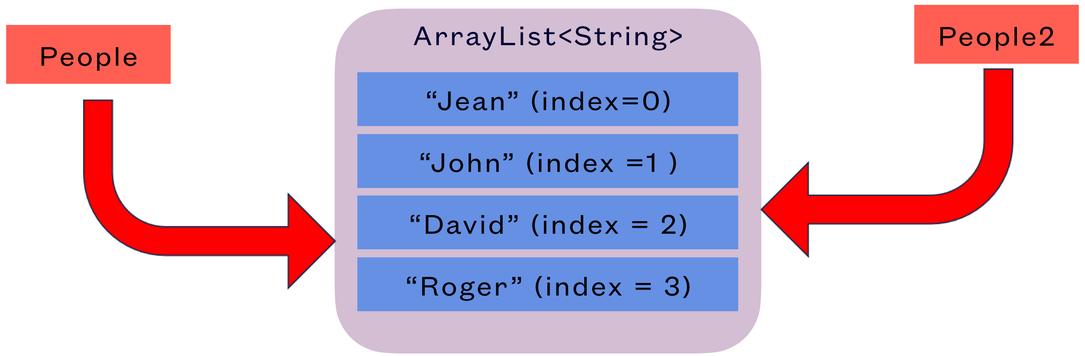
**** Output ****

```
c44fdc393f5c38c54d508e00ef70a5/redhat
people : [Jean, John, David, Roger]
people2: [Jean, John, David, Roger]
jean-claude@MacBook-Pro-2 CMM024 Code
```


LET’S CODE

ArrayLists: Referencing (Practice2)

- How do we prove people & people2 are linked to the same elements?
  - Lets replace a value in people and see if it is reflected in people2!




```
……

people.add("John");
people.add("Roger");
//add the line below
//we replace John with Mary in people
people.set(1,”Mary”);

……
```

**** Output ****

```
people : [Jean, Mary, David, Roger]
people2: [Jean, Mary, David, Roger]
jean-claude@MacBook-Pro-2 CMM024 Code
```

- Another way is to look at the actual memory location for both people and pleople2 Arraylists
  - To get an object memory location use the object hashCode() method. This returns an integer. To get the Hexadecimal we use Integer.toHexString(…) method.

```
System.out.println("people : " + Integer.toHexString((people.hashCode())));
System.out.println("people2: " + Integer.toHexString((people2.hashCode())));
```

```
people : [Jean, Mary, David, Roger]
people2: [Jean, Mary, David, Roger]
people : 3be77f7
people2: 3be77f7
jean-claude@MacBook-Pro-2 CMM024 Code
```


LET’S CODE

ArrayLists: Referencing (Practice2)

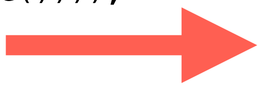
- What about the elements of these ArrayLists?. Just add the code below!

```
System.out.println("people : " + Integer.toHexString((people.hashCode())));
System.out.println("people2: " + Integer.toHexString((people2.hashCode())));
```

```
people : [Jean, Mary, David, Roger]
people2: [Jean, Mary, David, Roger]
people : 3be77f7
people2: 3be77f7
jean-claude@MacBook-Pro-2 CMM024 Code
```

- We can add the line below

```
for(String name: people){
   System.out.print(" " + name + " " + Integer.toHexString((name.hashCode())));
}
System.out.println();
for(String name: people2){
   System.out.print(" " + name + " " + Integer.toHexString((name.hashCode())));
}
System.out.println();
```



```
 Jean 232ac8 Mary 247afb David 3ec1e54 Roger 4b78bd7
 Jean 232ac8 Mary 247afb David 3ec1e54 Roger 4b78bd7
jean-claude@MacBook-Pro-2 CMM024 Code %
```

This tells us that people and people2 share the same data because they refer to the same memory locations!

LET'S CODE

ArrayLists: copying

- There are many instances when you want to have a copy (clone) of an ArrayList

```
ArrayList<String> people2 = people;
//add line below
ArrayList<String> people3 = new ArrayList<>(people);
```

To begin with the lists and elements have the same memory addresses

```
people : 3be77f7
people3: 3be77f7
 Jean 232ac8 Mary 247afb David 3ec1e54 Roger 4b78bd7
 Jean 232ac8 Mary 247afb David 3ec1e54 Roger 4b78bd7
```

```
ArrayList<String> people2 = people;
//add line below
ArrayList<String> people3 = new ArrayList<>(people);
//add this line below to replace David with Romeo
people3.add("Romeo");
```

Java creates an independent ArrayList as soon as changes occurs in people3

```
people : 78c82943
people3: 78c82943
 Jean 232ac8 Mary 247afb David 3ec1e54 Roger 4b78bd7
 Jean 232ac8 Mary 247afb David 3ec1e54 Roger 4b78bd7 Romeo 4b7a25a
```

```
ArrayList<String> people2 = people;
ArrayList<String> people3 = new ArrayList<>(people);
//add this line below to replace David with Romeo in people
//people3.add("Romeo");
people.set(2,"Simon");
```

The same occurs when changing an element of people

```
people : 1dc3e5b5
people3: 3be77f7
 Jean 232ac8 Mary 247afb Simon 4c300d6 Roger 4b78bd7
 Jean 232ac8 Mary 247afb David 3ec1e54 Roger 4b78bd7
```

Java will always ensure that people and people2 are always are independent lists

```
//changing the line below to replace people2 with people3
System.out.println("people : " + Integer.toHexString((people.hashCode())));
System.out.println("people3: " + Integer.toHexString((people3.hashCode())));
for(String name: people){
    System.out.print(" " + name + " " + Integer.toHexString((name.hashCode())));
}
System.out.println();
for(String name: people3){
    System.out.print(" " + name + " " + Integer.toHexString((name.hashCode())));
}
System.out.println();
```

LET'S CODE

ArrayLists: Useful Algorithms

- It is often the case you wish to know the minimum or maximum values in numerical data

```
public static void main(String[] args) {

        final int LENGTH = 10;
        ArrayList<Double> dnum = new ArrayList<>();
        Random rand = new Random();

        for (int i = 0; i < LENGTH; i++)
            dnum.add(i, rand.nextDouble());

        for (Double i : dnum)
            System.out.printf("%.3f \t", i);
        System.out.println();

        double smallest = dnum.get(0);
        double highest = dnum.get(0);
        for(Double d : dnum){
            if(d < smallest){
                smallest = d;
            }
            if(d > highest){
                highest = d;
            }
        }
        System.out.printf("Smallest %.3f Highest %.3f \n", smallest, highest);

    }
```

```
jean-claude@macbook-pro-2 CMM024_Code % cd /Users/jean-claude/Teaching/2023-20
17-full.jdk/Contents/Home/bin/java -XX:+ShowCodeDetailsInExceptionMessages -cp
c44fdc393f5c38c54d508e00eff0a5/redhat.java/jdt_ws/CMM024_Code_186eb31c/bin Nume
0.372   0.516   0.965   0.088   0.378   0.003   0.270   0.143   0.638   0.091
Smallest 0.003 Highest 0.965
```

ArrayLists are very cleaver in term of functionality.

Remember you can only add class object, Double, Integer, Object and no primitives like double, integer etc.

However, the functionality allows you to typecast them automatically to primitives when reading them if possible and if you wish so

Change:

for(Double d : dnum){

To (small cap double)

for(double d : dnum){

And run program again. It work!

LET'S CODE

Next week

- **Reflection week!**
- **We will be using Arraylist in the next session!**
- **The next sessions are dedicated to classes**