CO452
Programming Concepts

Lecture 4

Collections (ArrayList) and Generics

Collections

In this lecture we'll explore some of the differences between the following data structures, and also discuss the String class in more detail

- List
- LinkedList
- ArrayList
- Arrays

Importing packages

Importing packages

Because the collection classes (ArrayList) exist in another folder (package) we have to ‘import’ the package so we can refer to the ArrayList class (create objects of it) in our class

```
import java.util.ArrayList;
/**
* Class comment…
*/
public class Student
{
    …
}
```

List

Java’s List

A List describes a collection of objects which are ordered.

In Java, the List is built as an interface, which cannot be instantiated (as it’s not a class).

However, List contains abstract forms of methods which are overridden in the ArrayList and LinkedList classes, providing the implementation (method bodies) which enable the behaviour of the data structure.

LinkedList

Linked List Introduction

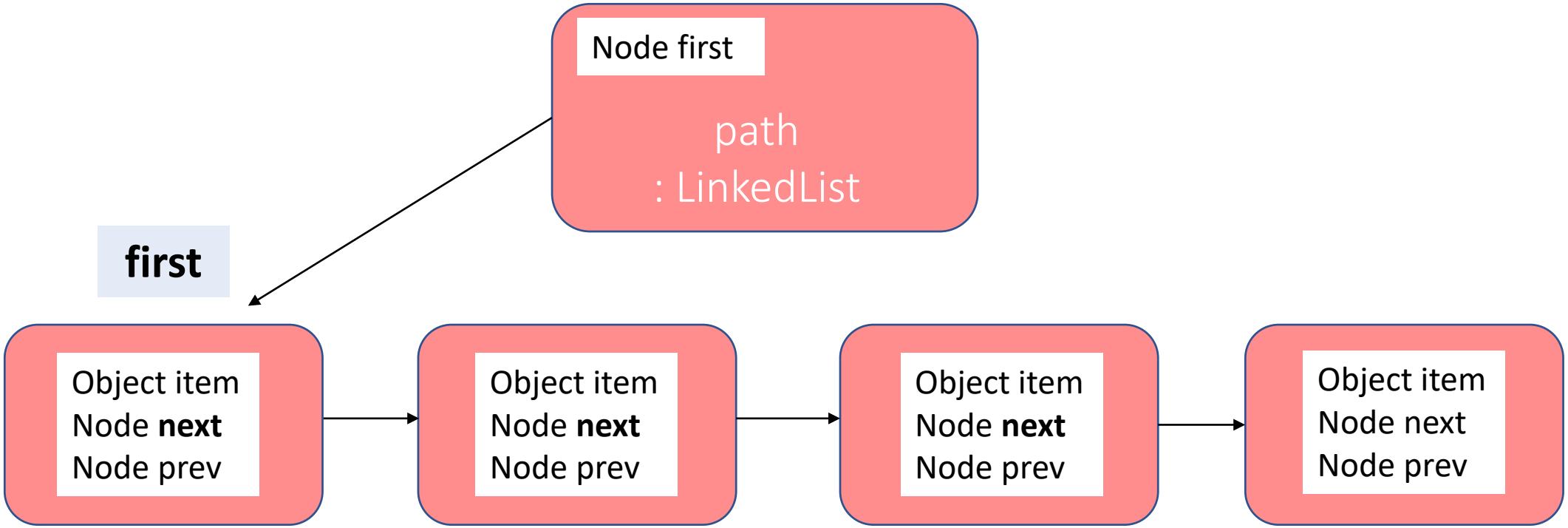
An instantiation of a singly Linked List will typically hold a pointer to the first object in the list.

The first object in the list then holds a pointer to the next object in the list and that object points to the next and so on. Therefore, objects are accessed sequentially starting from the first object (in a singly Linked List).

Objects are appended (added to the end; the last) or prepended (added to start of the list; the first).

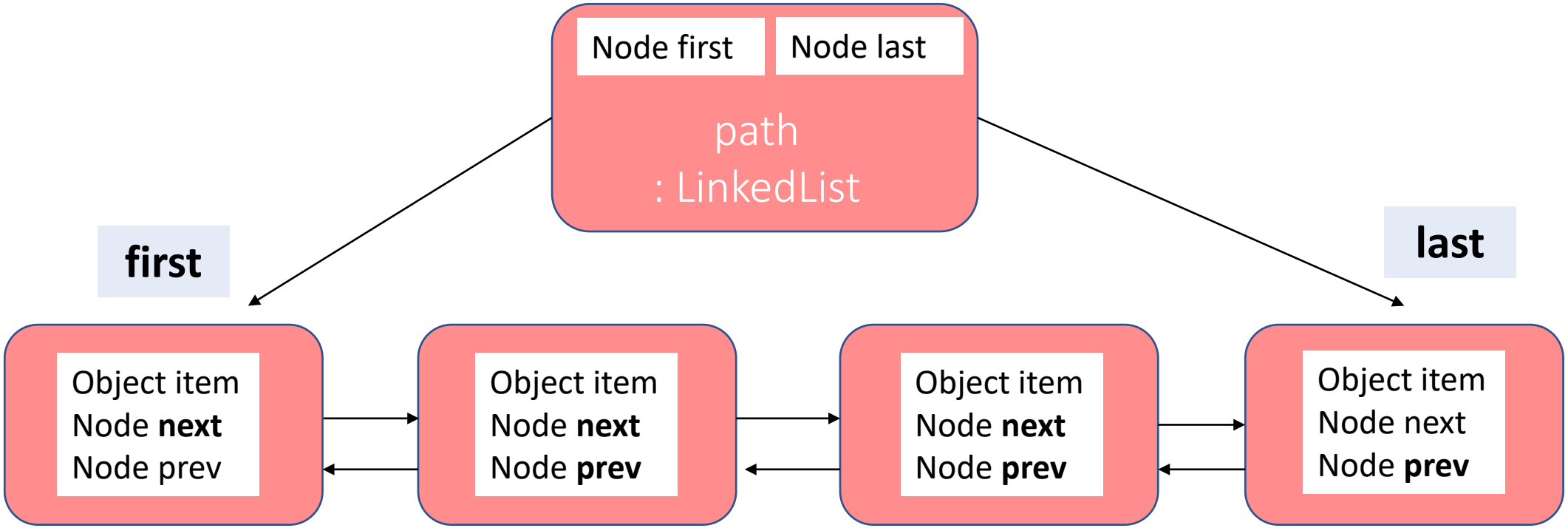
Visualisation of a (singly) LinkedList

A singly linked list would a pointer to the first item
and the individual nodes point to the next node in sequence



Visualisation of a doubly LinkedList

Java's LinkedList is also an example of a doubly linked list where objects also hold pointers to the previous object as well as next



Array

Array

The items in an array are called **elements.**

We specify how many elements an array will have when we declare the **size** of the array (if '**fixed-size**'), unlike flexible sized collections (ArrayList).

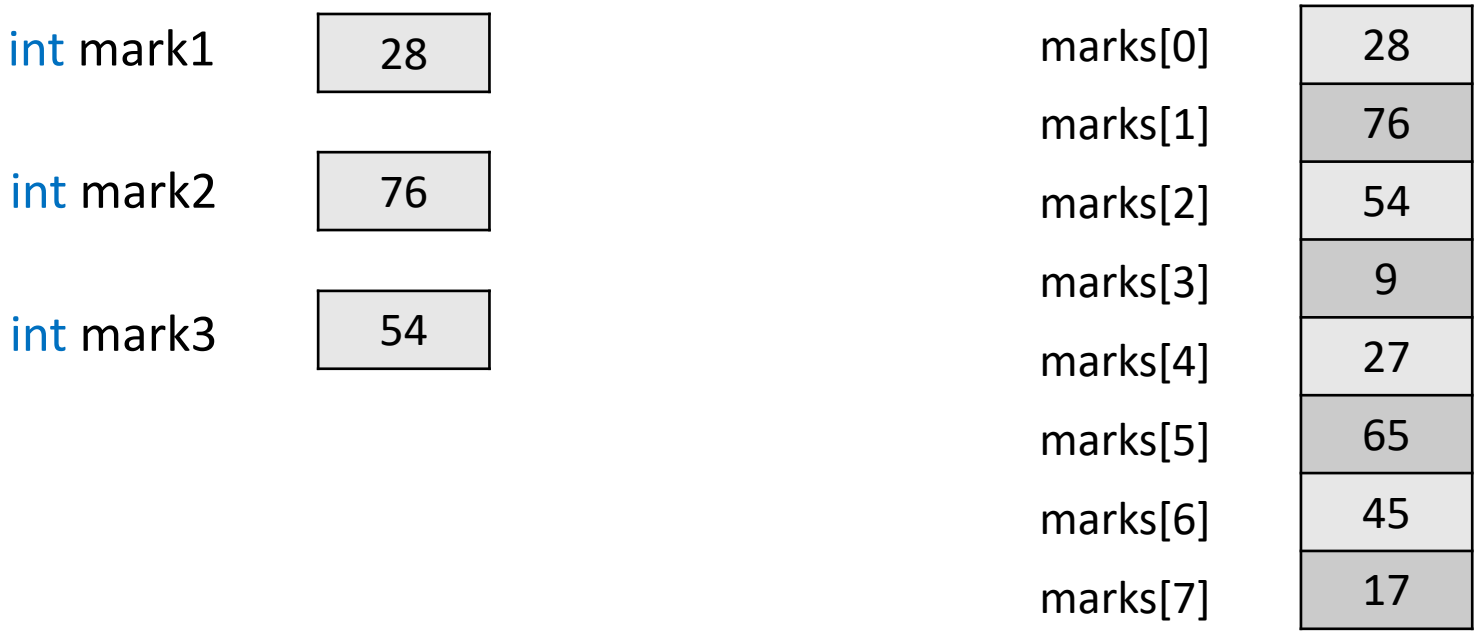
Elements are numbered and can referred to by number inside the [ ] is called the **index.** This is used when data is input and output.

Can only store data if it **matches the type** the array is declared with.

Visualisation of an Array

An Array is a structure that can hold multiple values in individual elements (positions)

```
int[] marks = new int[8];
```



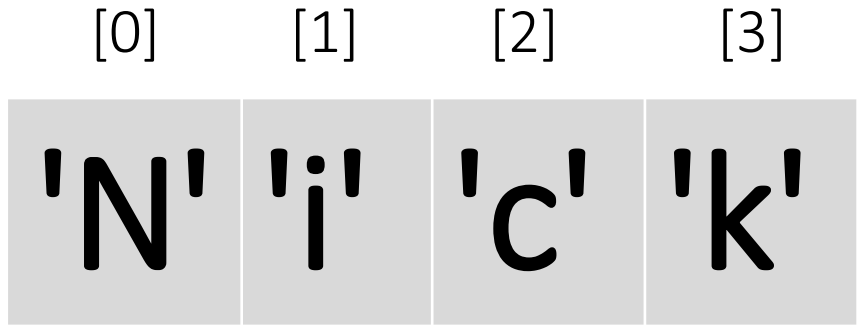
Strings

A String object is an array

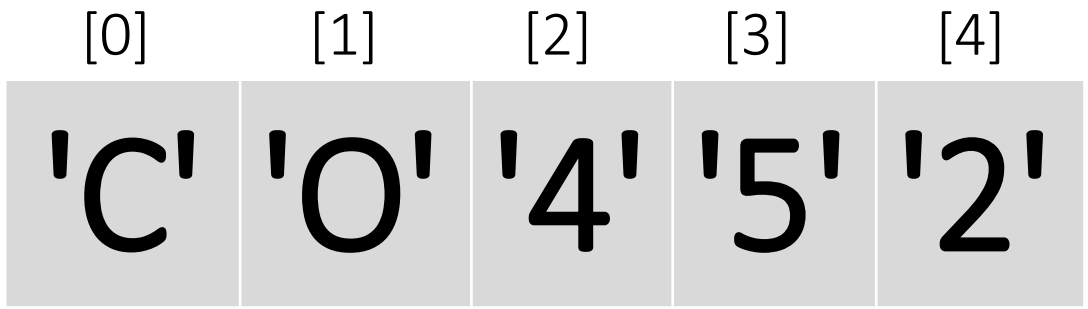
A String object is an **immutable array of characters**.

Each character has a numbered position in the array (index):

String name = "Nick";



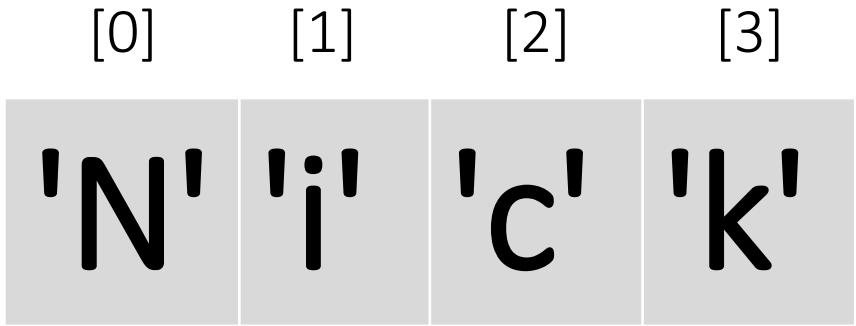
String code = "CO452";



Referring to characters in a String

You can refer to letters of a String through the index value. In Java you can pass the index value as a parameter to the method **charAt()**

String name = "Nick";



System.out.println(name.**charAt**(0)); // displays 'N'

String methods

Reminder on the equals() method

Whilst the equality operator (==) can be applied to primitive data (int, char, boolean), Strings are classes, so **the equality operator would compare memory addresses of String objects** rather than the values stored in each object

Use the method **equals** to compare the values stored at String variables rather than comparing memory addresses

```
if(name.equals("Nick"))
```

toUpperCase() and toLowerCase()

Methods which remove ‘casing’ can make validation easier when performing comparisons:

```
String name = "Nick";
System.out.println(name.toUpperCase());     // NICK
System.out.println(name.toLowerCase());      // nick
```

contains()

The contains method of the String library can be called to evaluate whether a part of one String exists in another. Be aware though, that the comparison is case sensitive.

```
String name = "Nick Day";

System.out.println(name.contains("Day"));       // true
System.out.println(name.contains("day"));       // false (case sensitive)

String title = "Programming Concepts";

System.out.println(title.contains("min"));      // true
```

ASCII

Under the American Standard Code for Information Interchange (ASCII), each letter, number and symbol on a keyboard has a decimal integer value. These can be used to evaluate and compare characters.

0-9 keys : ASCII decimal values : 48 - 57

A-Z keys : ASCII decimal values : 65 - 90

a-z keys : ASCII decimal values : 97 - 122

Other useful String methods

compareTo()

hashCode()

indexOf()

join()

length()

split()

trim()

valueOf()

Full documentation available at:
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html

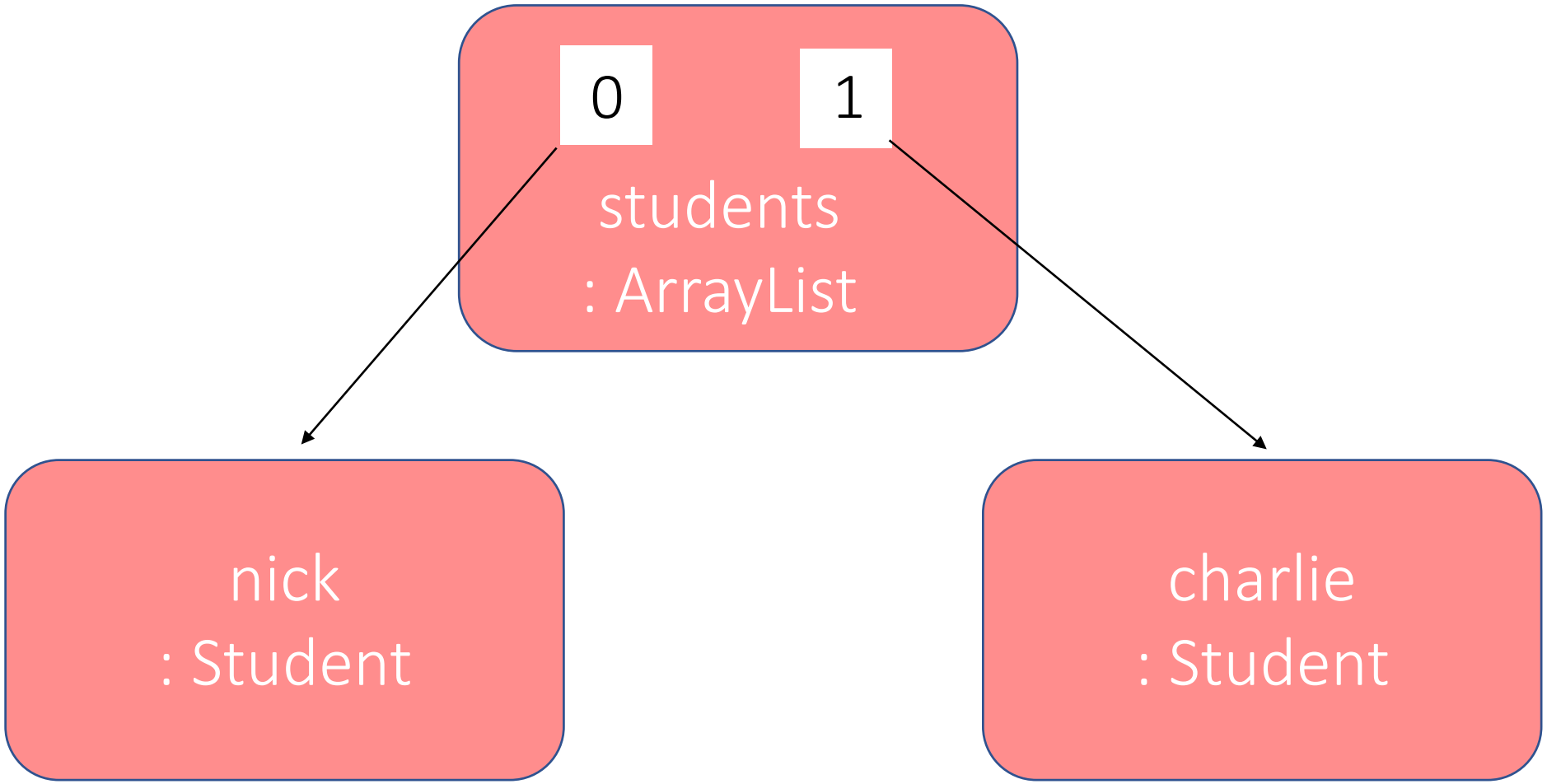
ArrayList

ArrayList

An ArrayList is a **convenient way** to store related objects in one collection.

For example, we could create a collection of student objects called '**students**'.

Visualisation of an ArrayList



Some methods of the ArrayList

add()

remove()

clear()

get()

size()

How to create an ArrayList

How to instantiate an object of the ArrayList

```
private ArrayList<Student> students;
private ArrayList<Product> products;
```

How to instantiate an object of the ArrayList

Scope **Class** **Type of objects** **object (collection)**

```
private ArrayList<Student> students;
private ArrayList<Product> products;
```

How to instantiate an object of the ArrayList

Scope **Class** **Type of objects** **object (collection)**

```
private ArrayList<Student> students = new ArrayList<Student>();
private ArrayList<Product> products = new ArrayList<Product>();
```

How to instantiate an object of the ArrayList

Scope **Class** **Type of objects** **object (collection)** **Constructor**

```
private ArrayList<Student> students = new ArrayList<Student>();
private ArrayList<Product> products = new ArrayList<Product>();
```

Comparison with object syntax

Scope | Class | Type of objects | object (collection) | Constructor

```
private ArrayList<Student> students = new ArrayList<Student>();
```

Scope | Class | object | Constructor

```
private Student student = new Student();
```

Generics

What is a generic class

Collections such as the ArrayList are an example of a generic class (also known as a ‘parameterized class’).

These generic classes utilise the ‘diamond notation’ **< > and substitute the placeholders in the class definition for the type placed in the < >**

This reduces duplication as there is no need to create separate classes or methods which only work with one type.

Portion of the ArrayList class

```
/**
*   A portion of the ArrayList generic class
*   @param <E> e short for element
*/
public class ArrayList<E>…
{
   public boolean add(E e)
   {
      …
   }

   public E remove(int index)
   {
      …
   }
}
```

Full documentation available at:
https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

Add to an ArrayList

Adding objects through the method

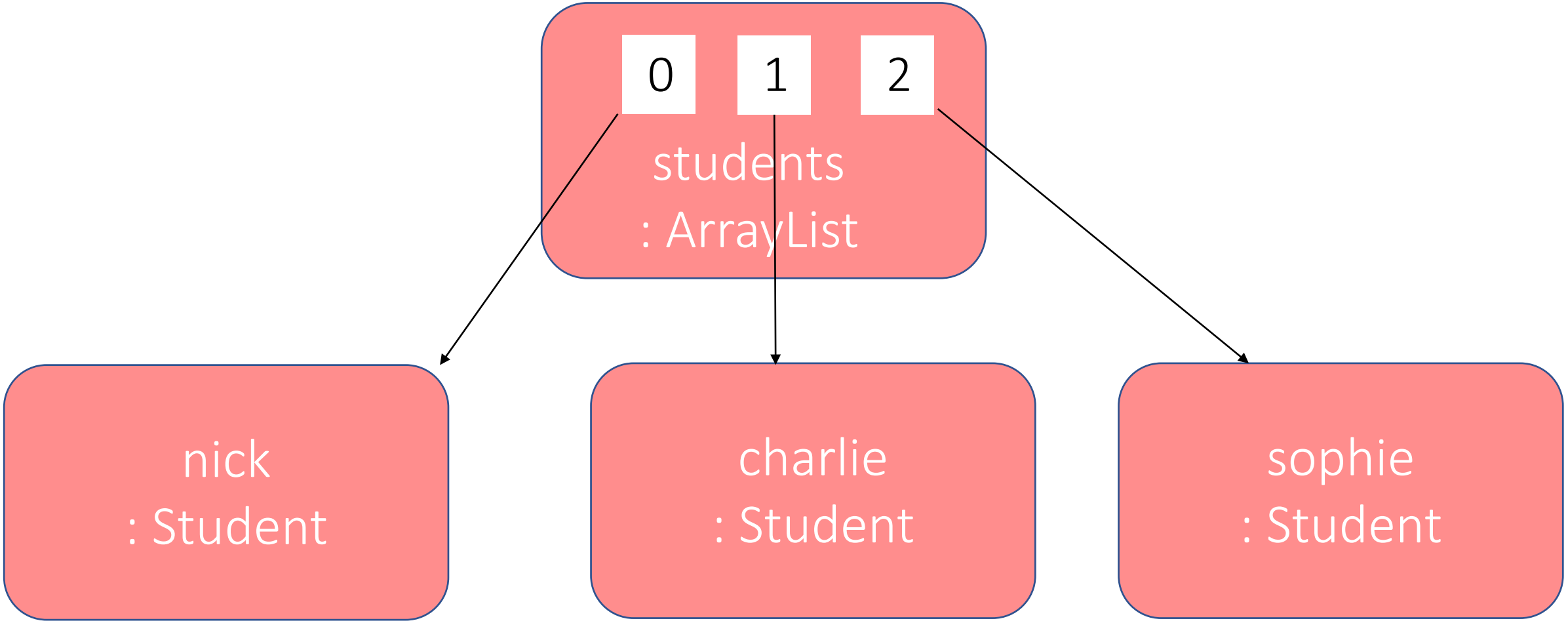
Can call the ‘add’ method through an ArrayList object

students.**add**(nick)

students.**add**(charlie)

students.**add**(sophie)

Adding an item to an ArrayList



Iterating through an ArrayList (for each)

for each loop with collection

The for each loop can be used to iterate through collections of objects.

Requires an object to be declared of the type of item that is in the collection:

```
for(Student student : students)
{
    student.print();
}
```

for each loop with collection

The for each loop can be used to iterate through collections of objects.

Requires an object to be declared of the type of item that is in the collection:

Class **object** **ArrayList**

```
for(Student student : students)
{
    student.print();
}
```

call print on each object in the ArrayList

Finding an item in an ArrayList

Finding an item in an ArrayList

We can take the same for each loop and use to check each object individually…

```
public Student findByID(int id)
{
    for(Student student : students)
    {
        if(student.getID() == id)
            return student;
    }
    return null;
}
```

Finding an item in an ArrayList

… and can check to see whether the value we are searching for matches a value in an item of the ArrayList

```
public Student findByID(int id)
{
    for(Student student : students)
    {
        if(student.getID() == id)
            return student;
    }
    return null;
}
```

Finding an item in an ArrayList

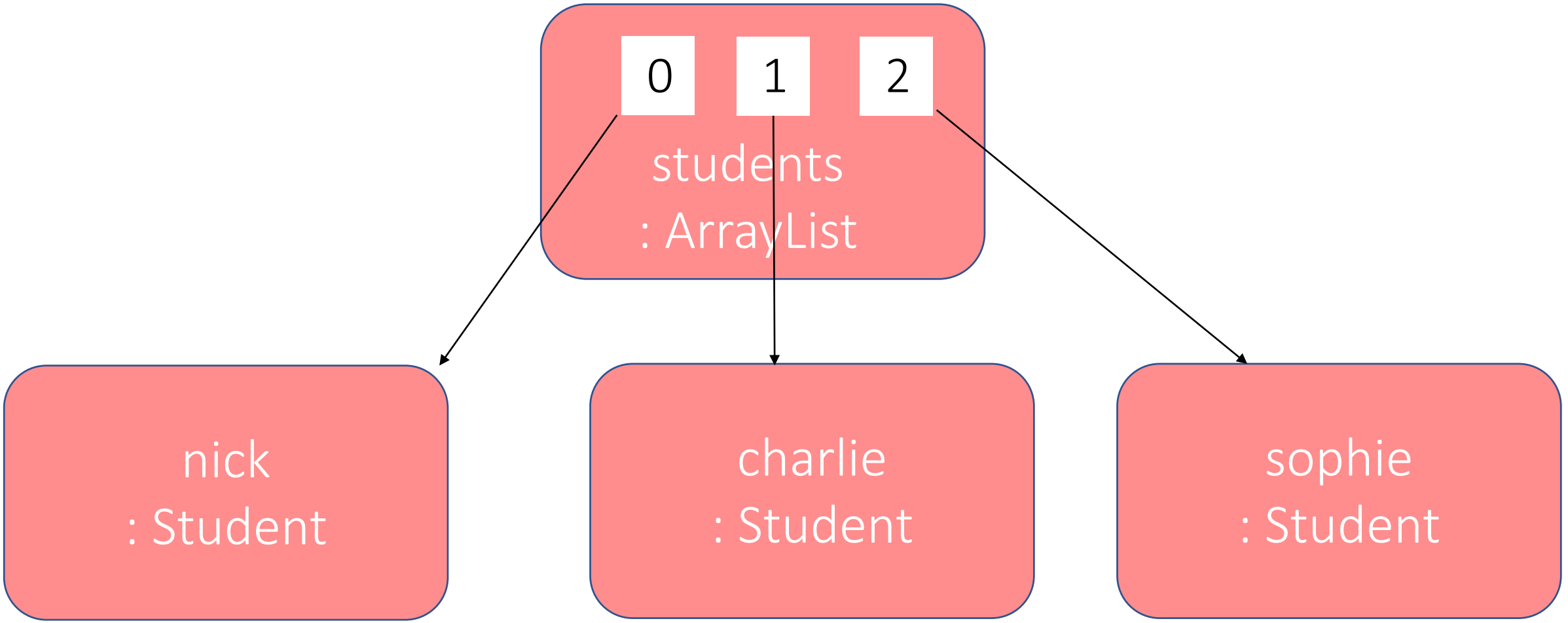
… and can check to see whether the value we are searching for matches a value in an item of the ArrayList

```
public Student findByID(int id)
{
    for(Student student : students)
    {
        if(student.getID() == id)
            return student;
    }
    return null;
}
```

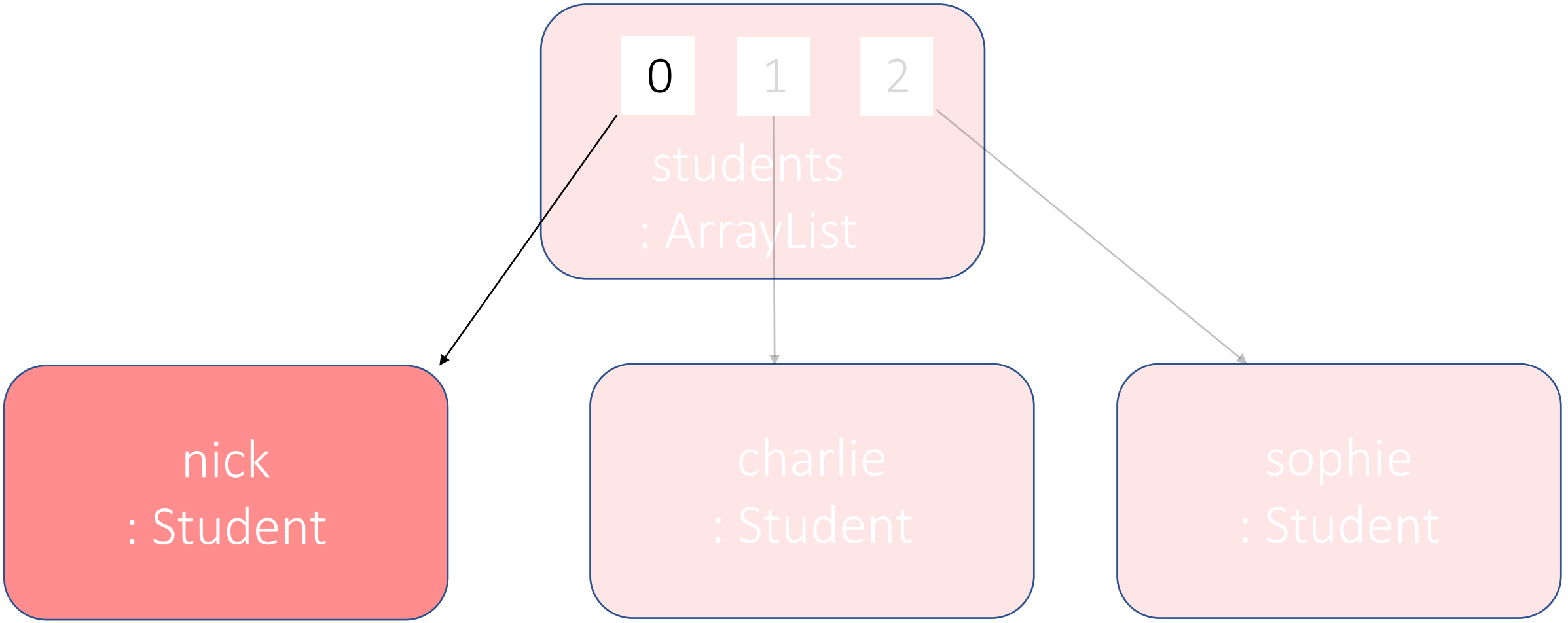
Example

Let’s say that we are looking to **return the student object that has the id value of 4231**

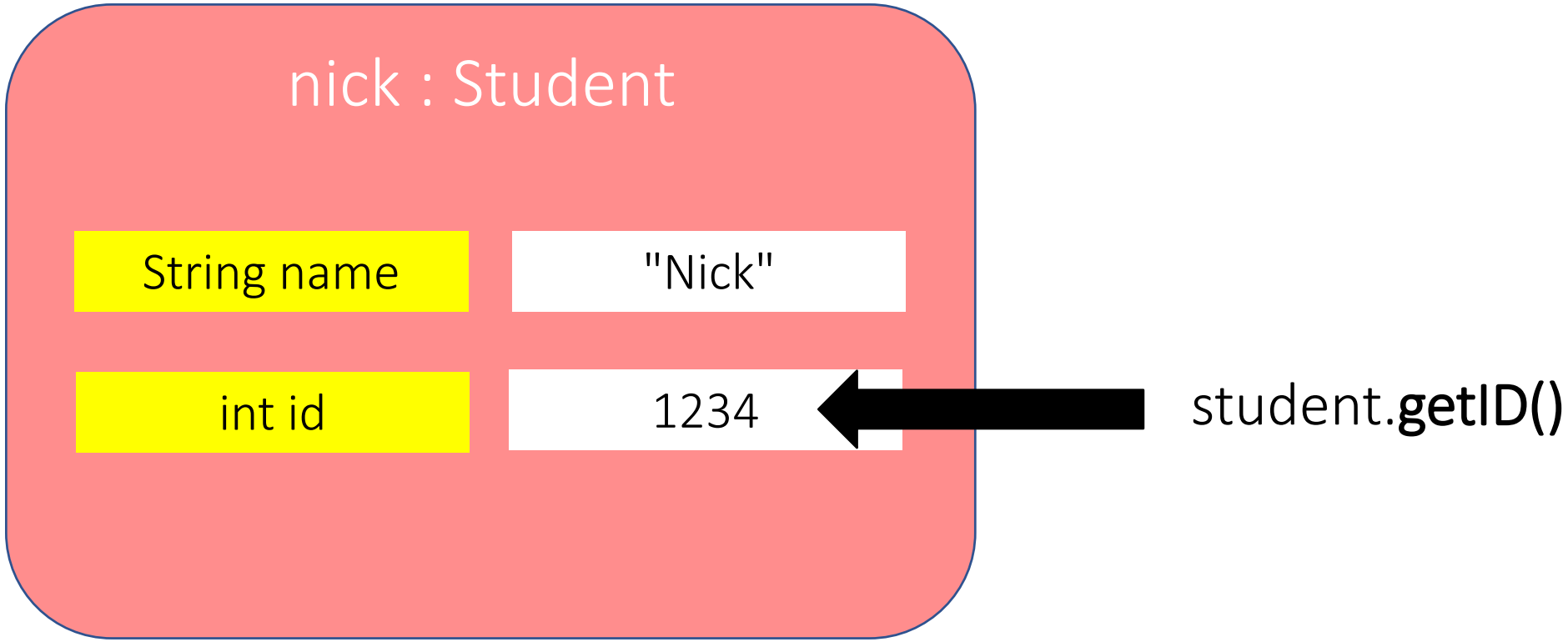
Check each item in the ArrayList



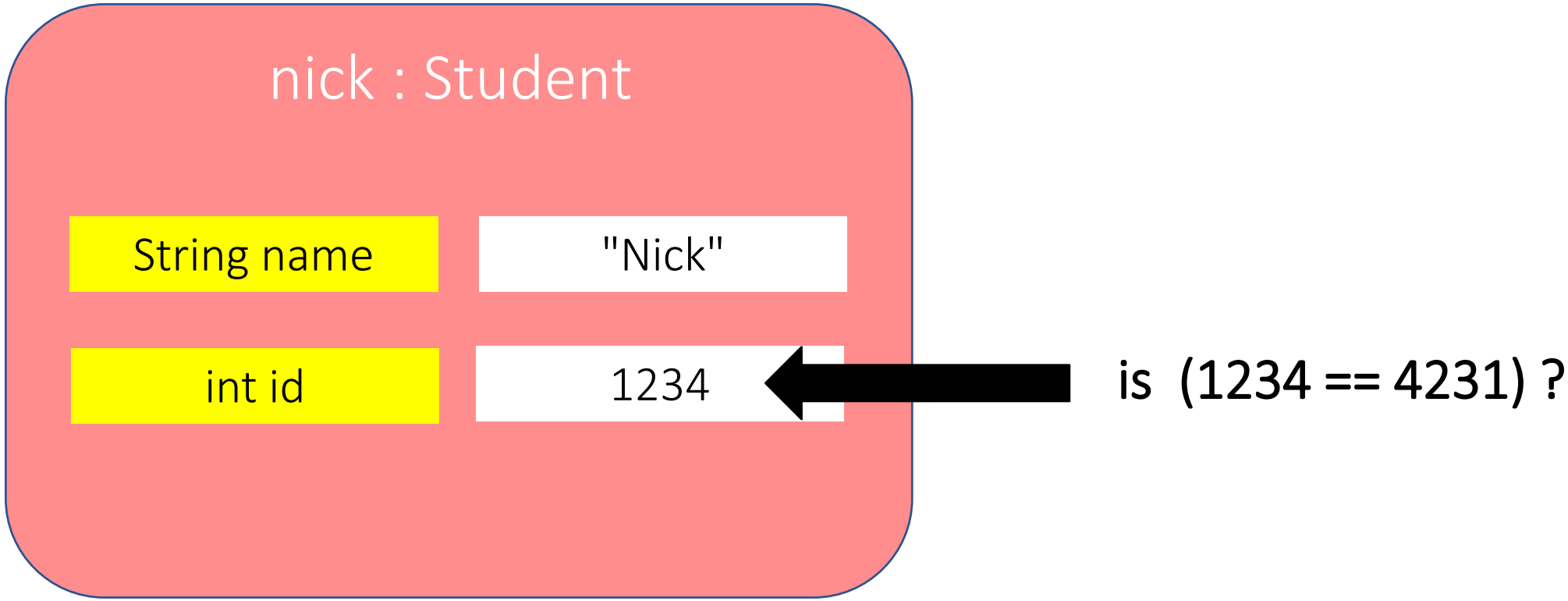
Check the first item



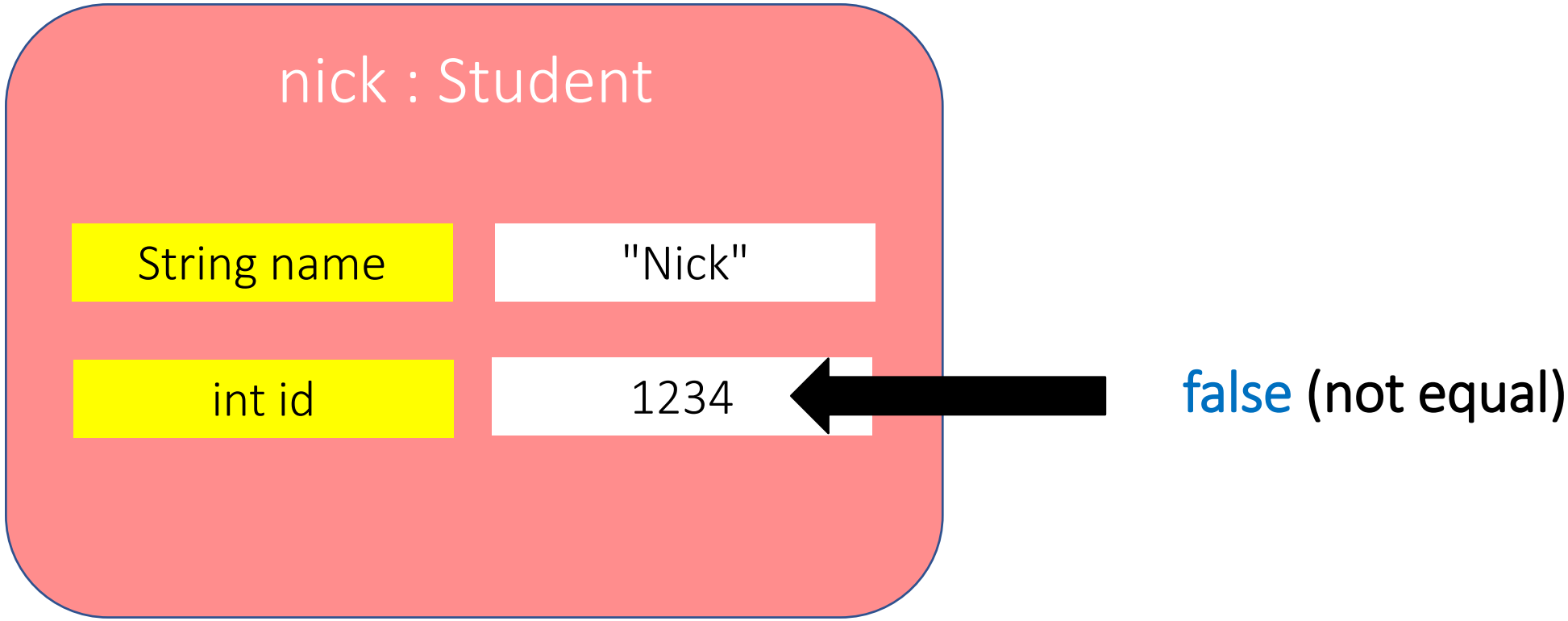
Return the id



Check the id



Not equal so move onto next item

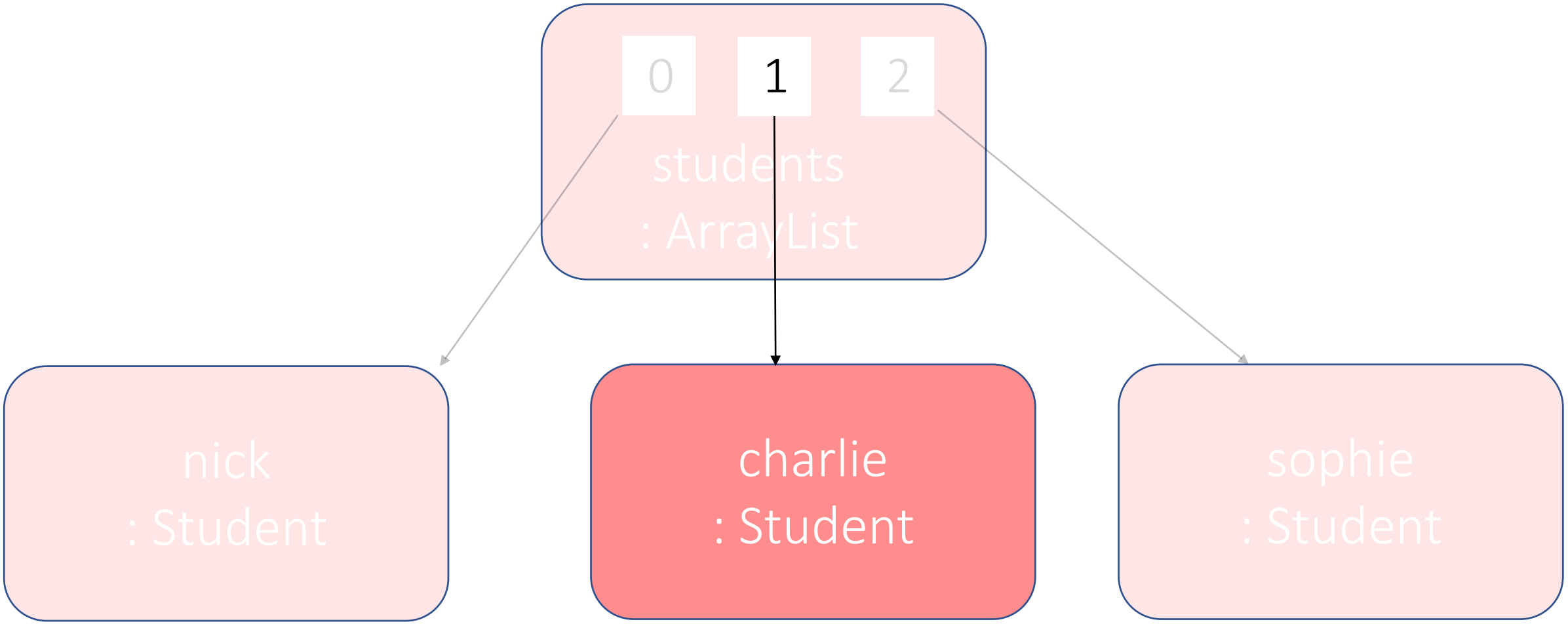


Reminder about the for each loop

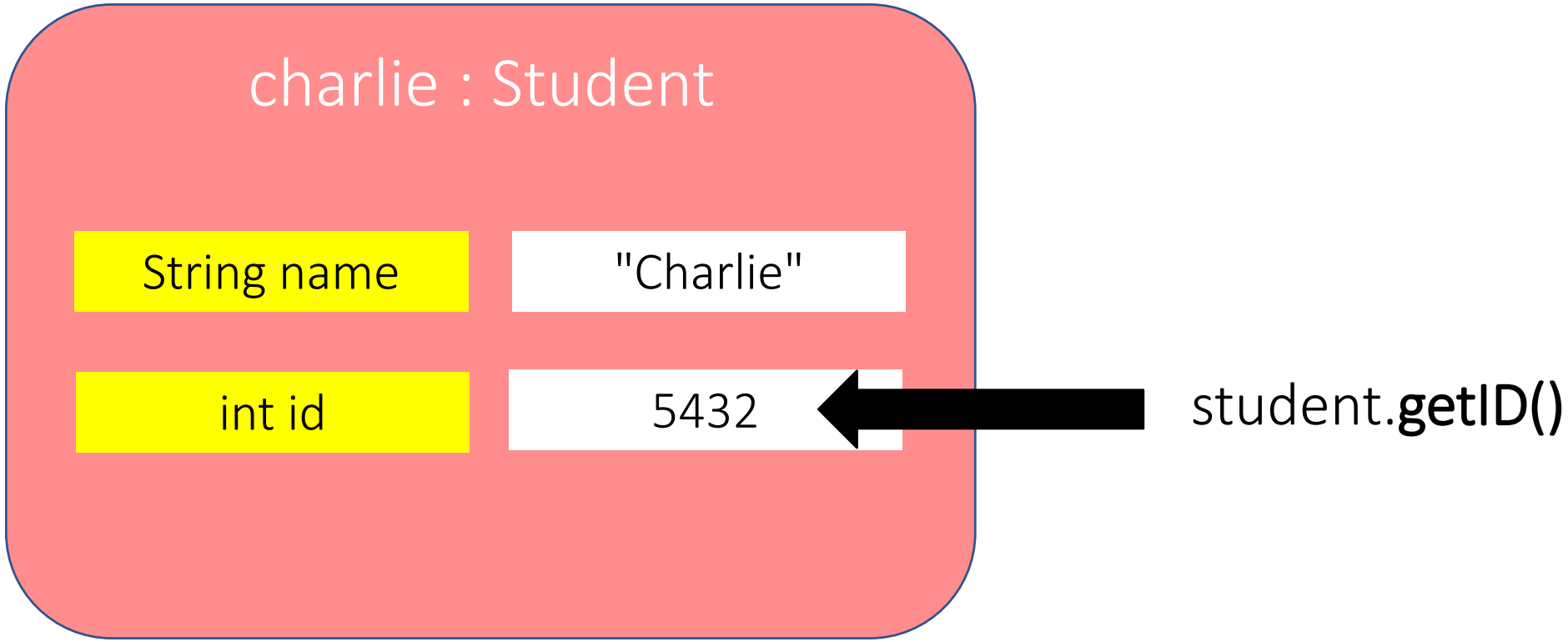
… and can check to see whether the value we are searching for matches a value in an item of the ArrayList

```
public Student findByID(int id)
{
    for(Student student : students)
    {
        if(student.getID() == id)
            return student;
    }
    return null;
}
```

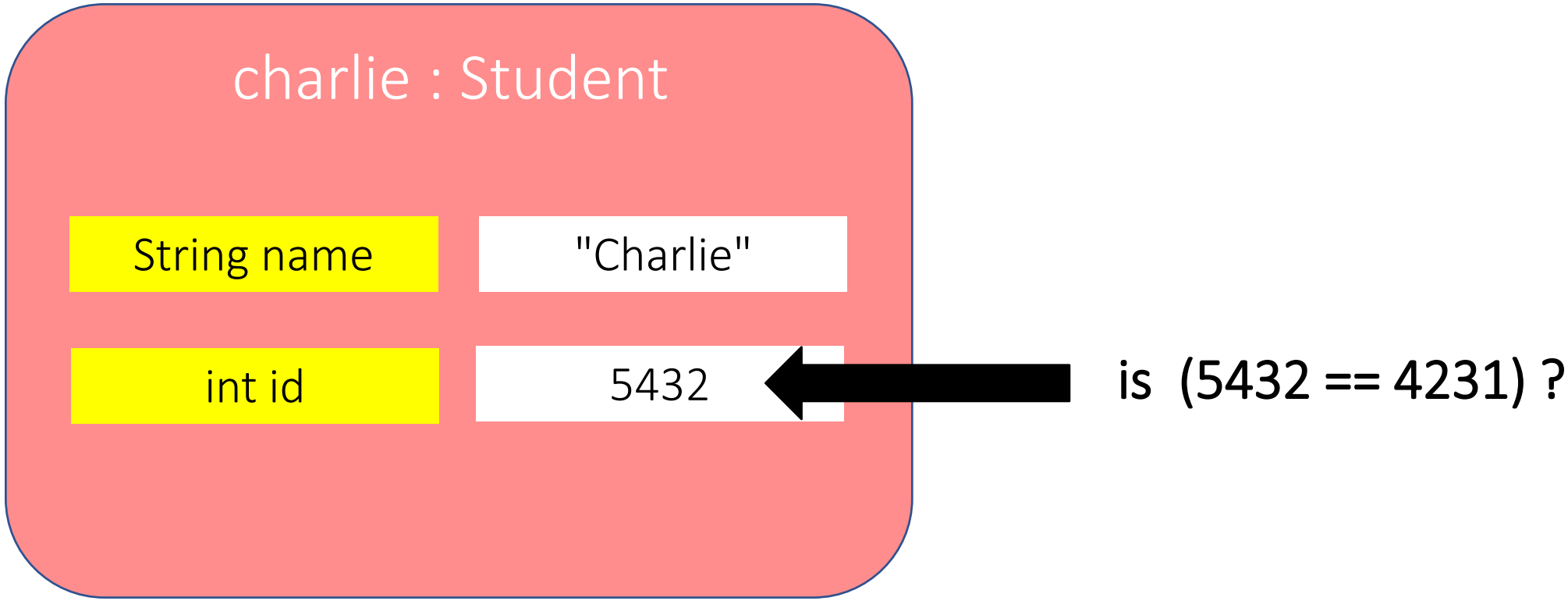
Check the next item



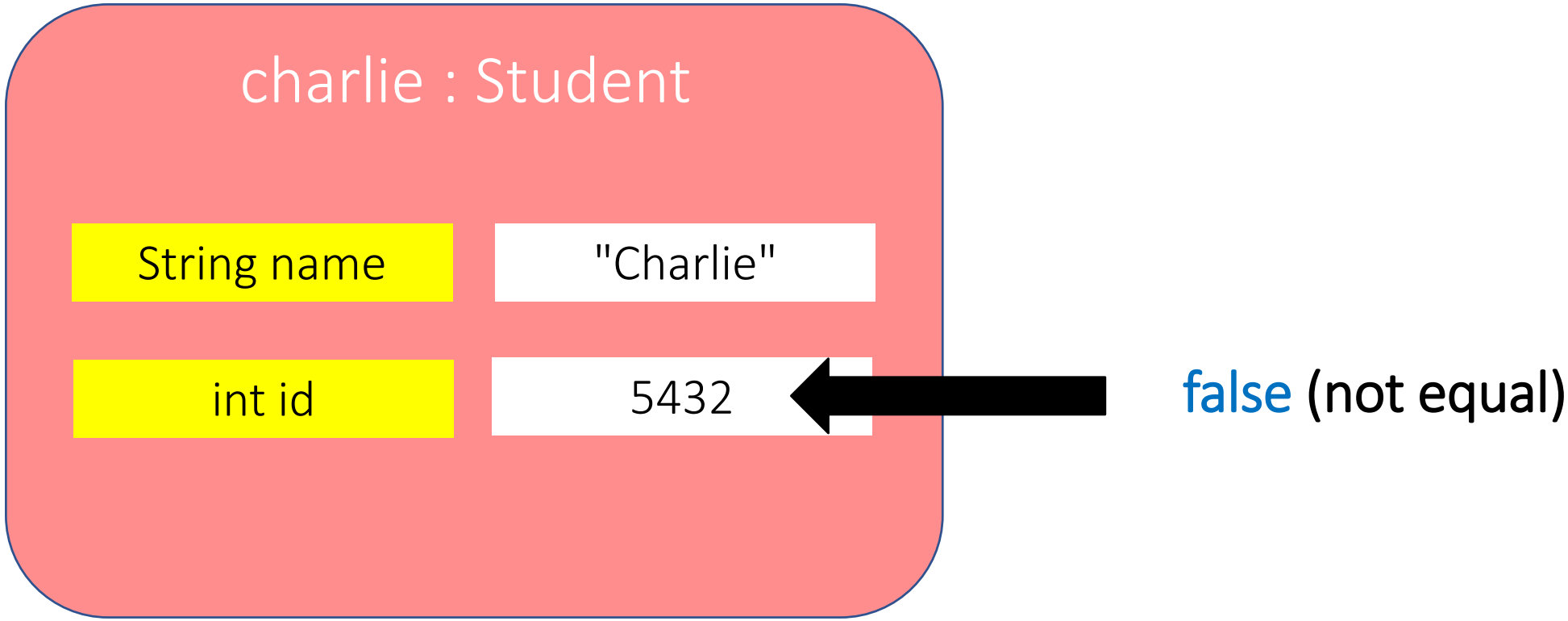
Return the id



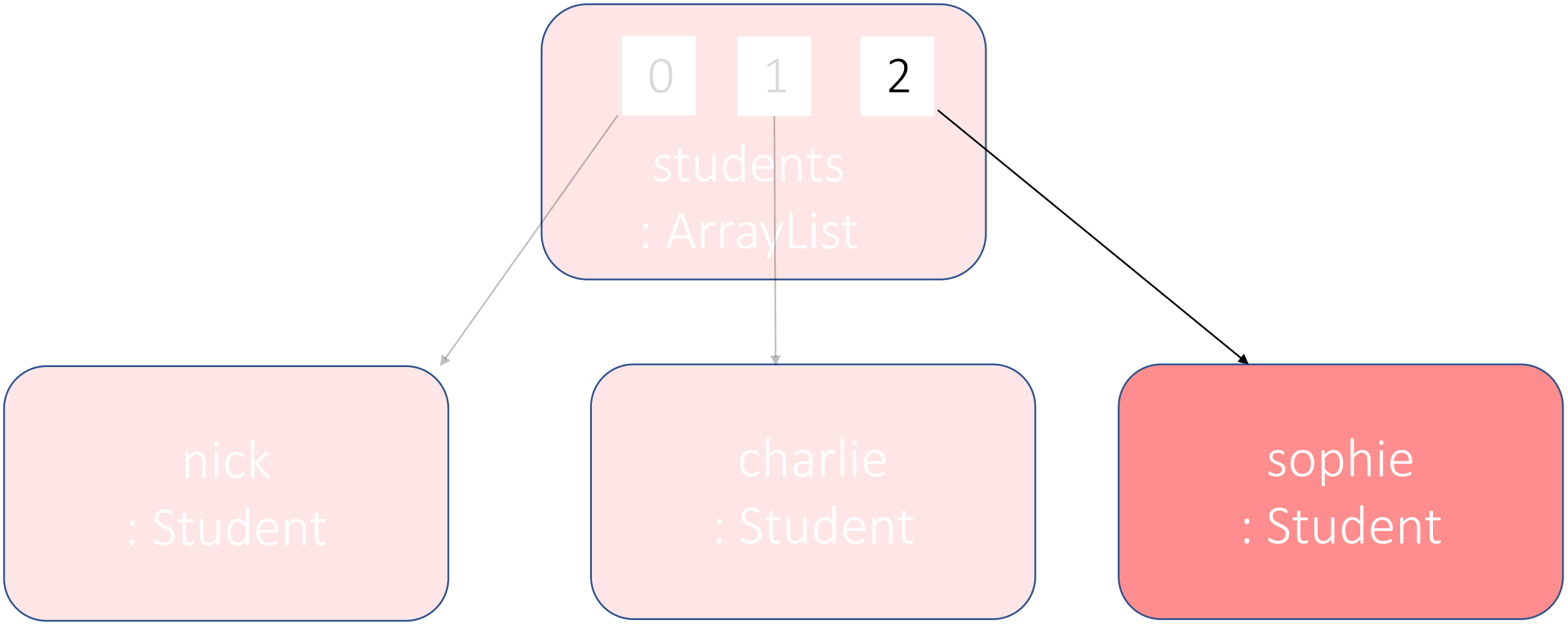
Check the id


charlie : Student
String name
"Charlie"
int id
5432
is (5432 == 4231) ?

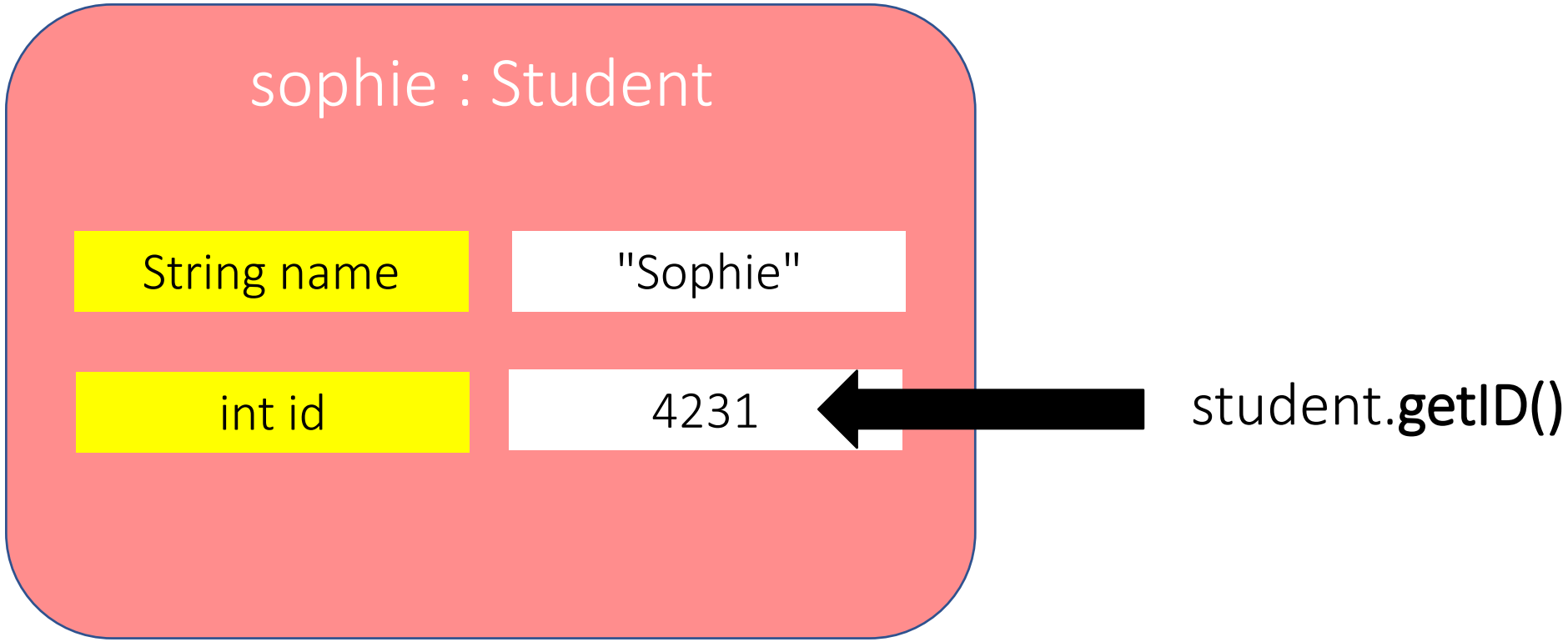
Not equal so move on to next item



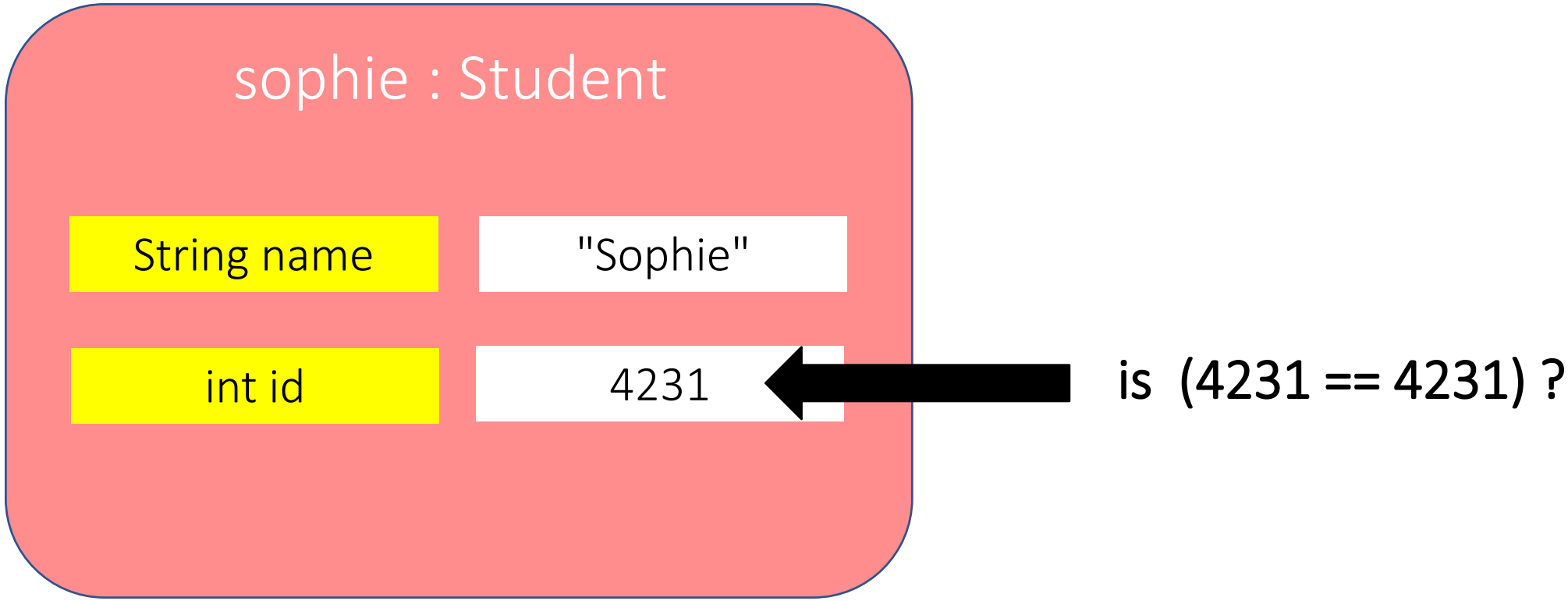
Check the next item



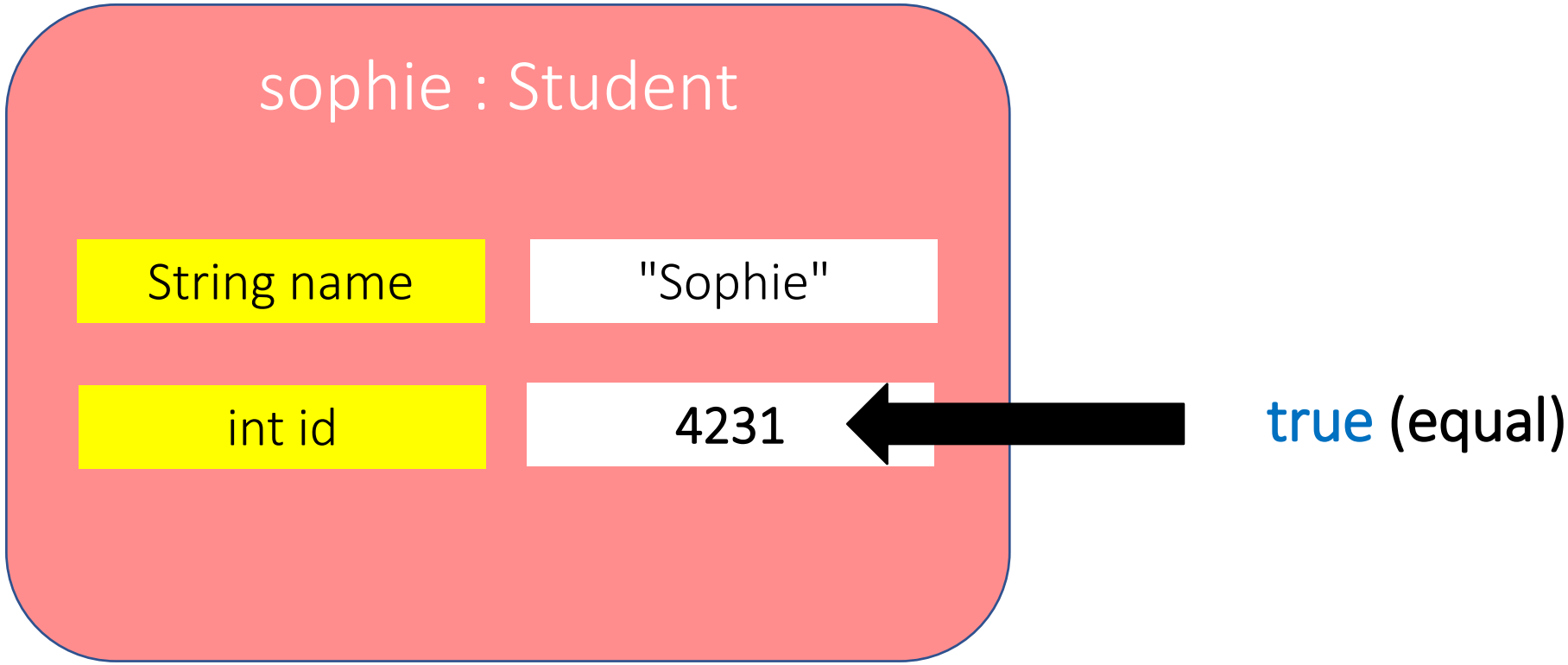
Return the id



Check the id



Match! So return object



Return the object ‘found’

Once an object with a value that matches the sought value is found, return that object and end the search

```
public Student findByID(int id)
{
    for(Student student : students)
    {
        if(student.getID() == id)
            return student;
    }
    return null;
}
```

Removing from an ArrayList

Removing an item in an ArrayList

Step 1 – **find** the object we want to remove

Step 2 – **remove** the located object (if found)

```
public Student remove(int id)
{
    Student student = findByID(id);
    if(student != null)
        students.remove(student);
    else
        System.out.println("Could not find student");
}
```