Introduction to ArrayLists

(based on Ch. 4, Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling)

Produced by: Ms. Mairéad Meagher
Dr. Siobhán Drohan
Ms Siobhan Roche

Ollscoil Teicneolaíochta an Oirdheiscirt
South East Technological University

Department of Computing and Mathematics
http://www.setu.ie/

Topic list

1. Grouping Objects
   - Developing a basic personal notebook project using **Collections** e.g. **ArrayList**
2. **Indexing** within Collections
   - Retrieval and removal of objects
3. **Generic classes**
   - e.g. ArrayList
4. **Iteration**
   - Using the for loop
   - Using the while loop
   - Using the **for each** loop

- Next SlideDeck:
  coding a Shop Project that stores an ArrayList of Products.

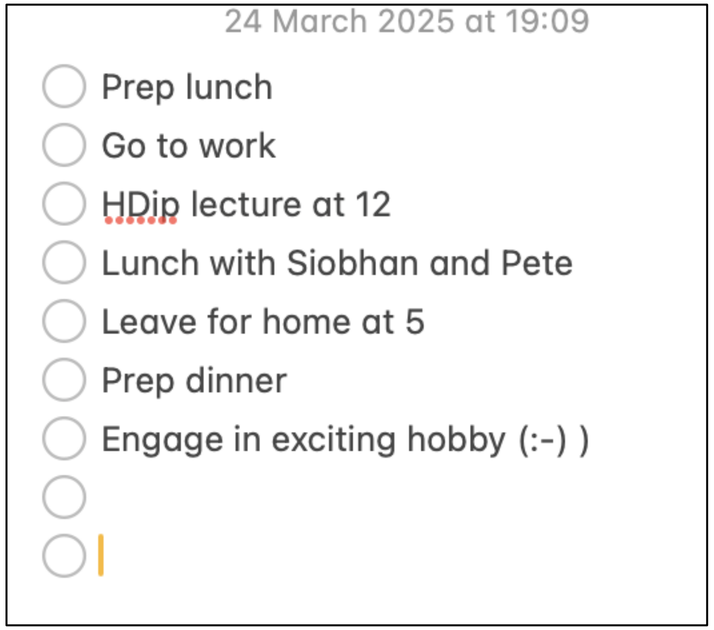
The requirement to **group objects**

- Many applications involve **collections** of objects:
  - Personal organizers.
  - Library catalogs.
  - Student-record system.

- The **number of items** to be stored <u>**varies**</u>:
  - Items added.
  - Items deleted.

Example: Personal Note Keeper

- Notes may be **stored**.
- Individual notes can be **viewed**.
- There is **no limit** to the number of notes.
- It generally **tells you how many** notes are stored.

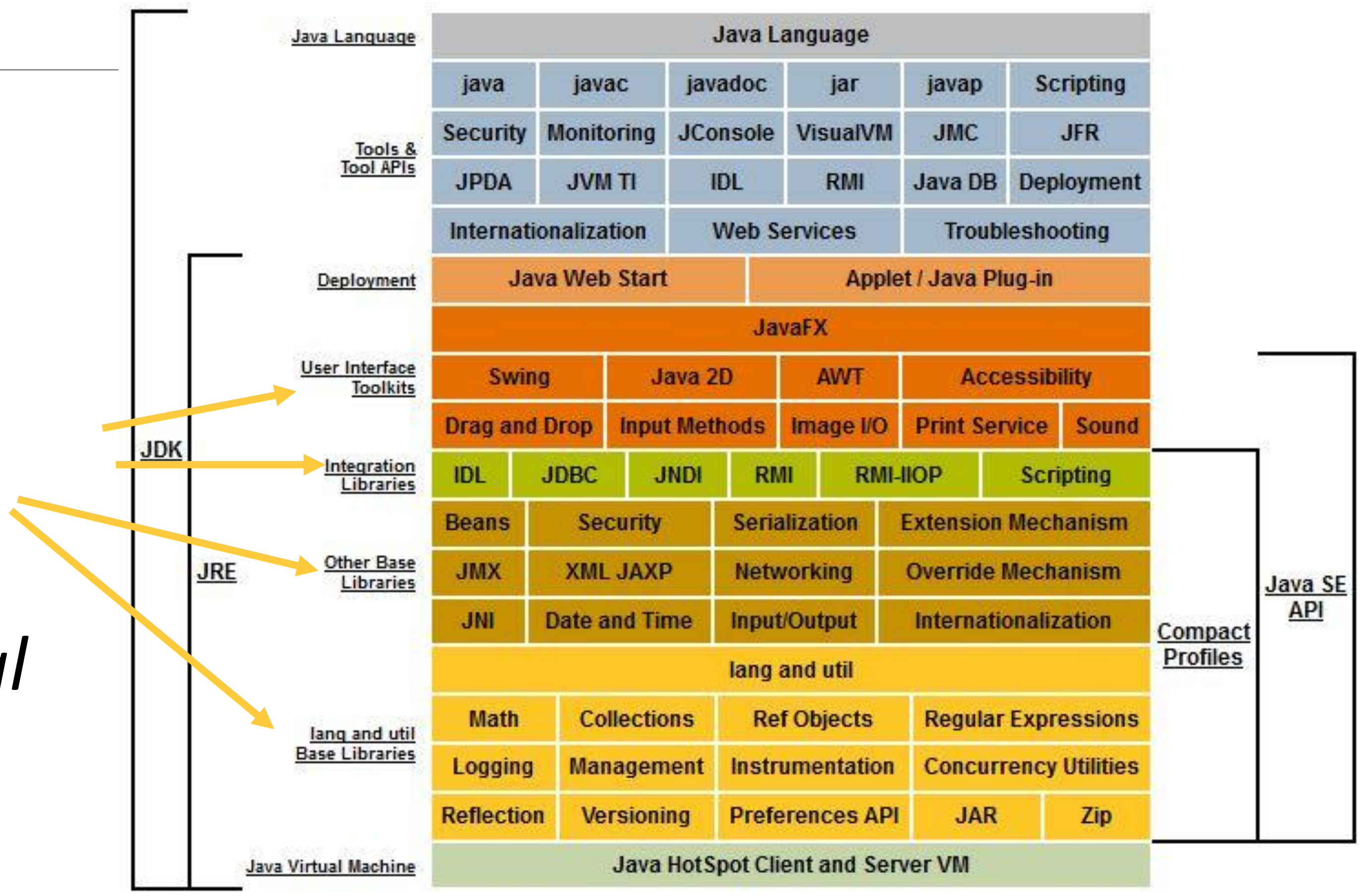


24 March 2025 at 19:09

- Prep lunch
- Go to work
- HDip lecture at 12
- Lunch with Siobhan and Pete
- Leave for home at 5
- Prep dinner
- Engage in exciting hobby (:-))

Java API – The Class Library

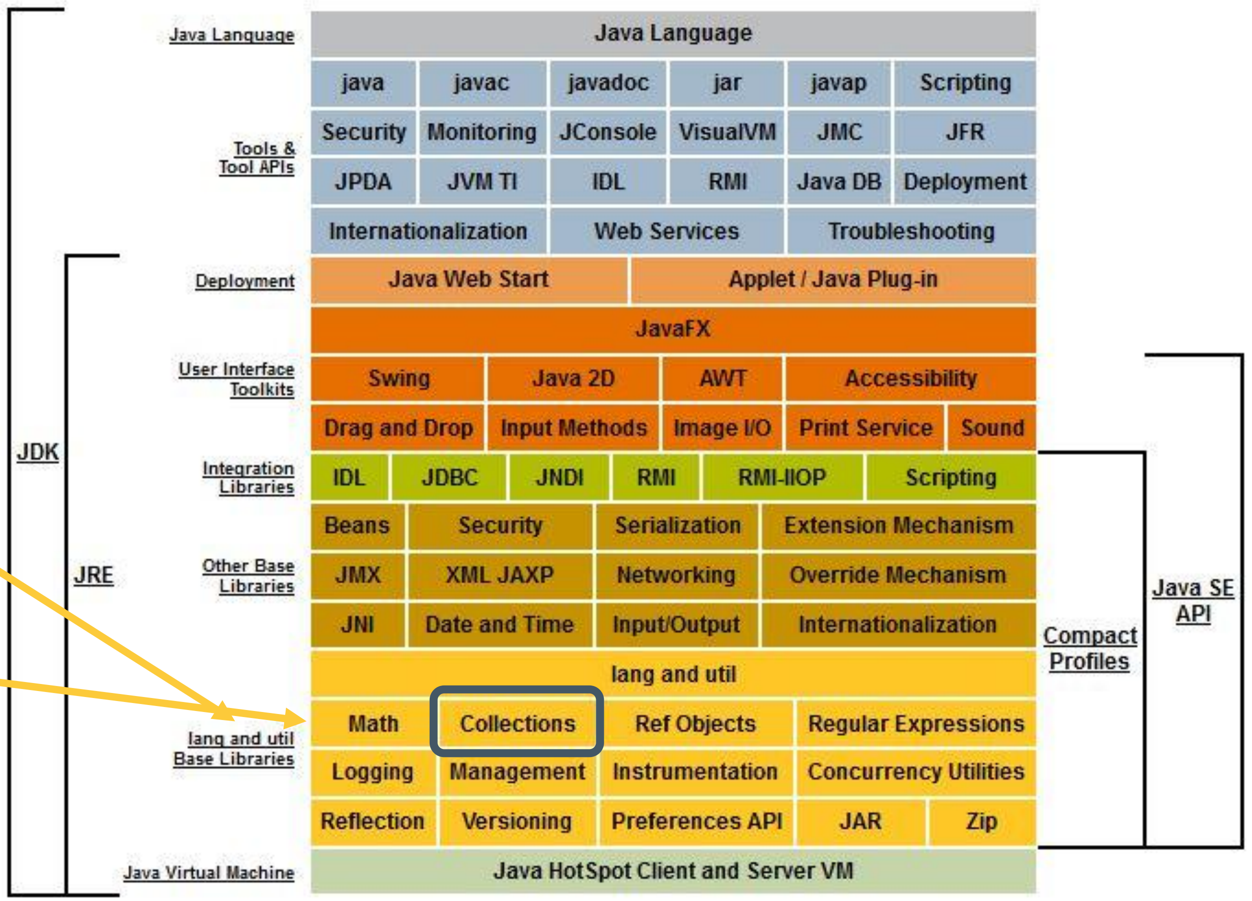
- Many useful classes.
- We don’t have to write everything from scratch.
- Java calls its libraries, ***packages***.
- *Packages contain individual **classes***



https://www.oracle.com/java/technologies/platform-glance.html

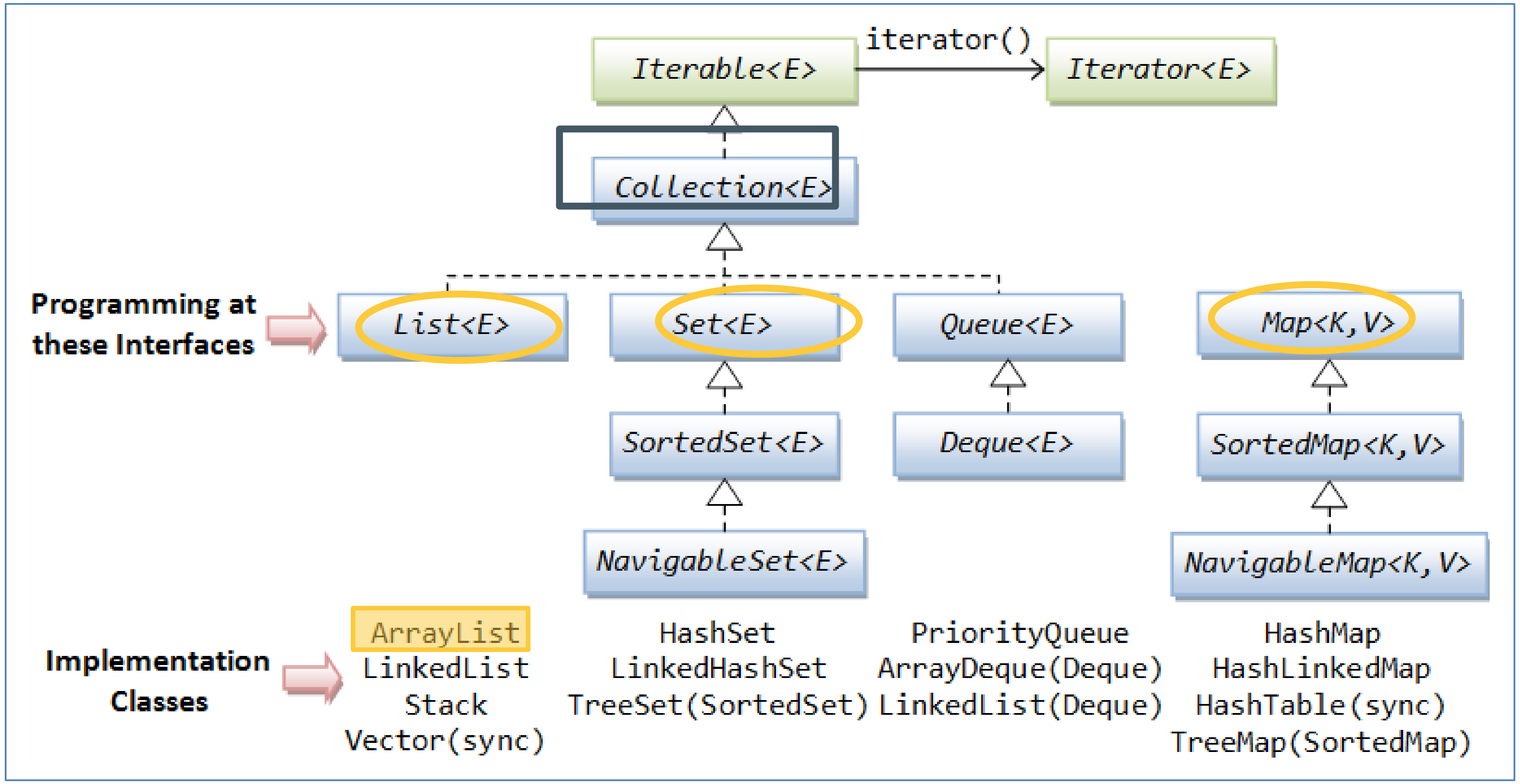
Java API – The Class Library

- Grouping objects is a recurring requirement.
  - The **`java.util`** package contains classes for doing this
  - …the **Collections Framework**



https://www.oracle.com/java/technologies/platform-glance.html

Java's Collection Framework

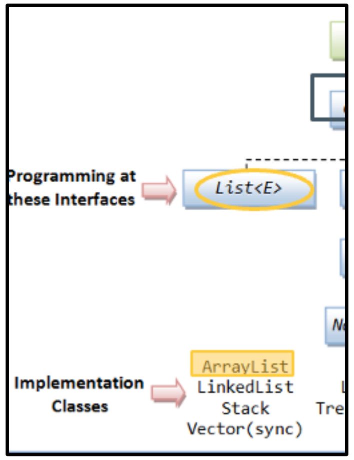


Lists

What properties do lists have?

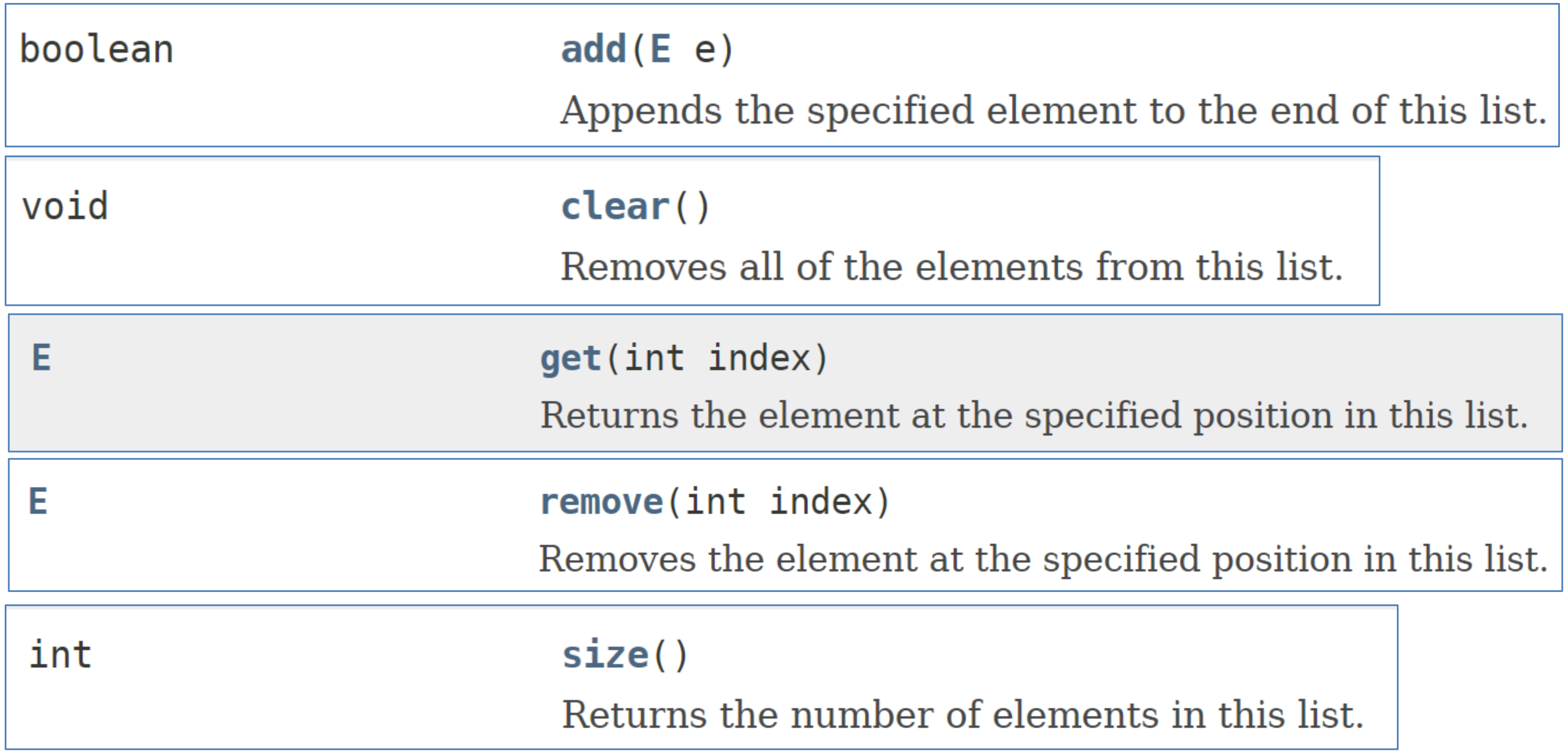
- Order – this may be chronological (in the order the elements were added);
- You can access any element in a list;
- You can add or remove any element in the list;
- You can have duplicates in a list.

We will look at ArrayLists



ArrayList – methods we will use

- **`ArrayList`** implements list functionality:



`boolean`	`add(E e)` Appends the specified element to the end of this list.
`void`	`clear()` Removes all of the elements from this list.
`E`	`get(int index)` Returns the element at the specified position in this list.
`E`	`remove(int index)` Removes the element at the specified position in this list.
`int`	`size()` Returns the number of elements in this list.

ArrayList Collection

- We specify:
  - the **type of collection**
    - e.g.: **`ArrayList`**
  - the **type of objects** it will contain
    - e.g.: **`<String>`**
- We say
  - **“ArrayList of String”**

```java
import java.util.ArrayList;

public class Notebook
{

      // Storage for an arbitrary number of notes.
      private ArrayList <String> notes;



      // Perform any initialization required for the notebook.
      public Notebook()
      {
            notes = new ArrayList <String>();
      }

}
```

```
import java.util.ArrayList;

public class Notebook
{

      // Storage for an arbitrary number of notes.
      private ArrayList <String> notes;



      // Perform any initialization required for the notebook.
      public Notebook()
      {
            notes = new ArrayList <String>();
      }

}
```

import the ArrayList package

```
import java.util.ArrayList;

public class Notebook
{

      // Storage for an arbitrary number of notes.
      private ArrayList <String> notes;


      // Perform any initialization required for the notebook.
      public Notebook()
      {
            notes = new ArrayList <String>();
      }

}
```

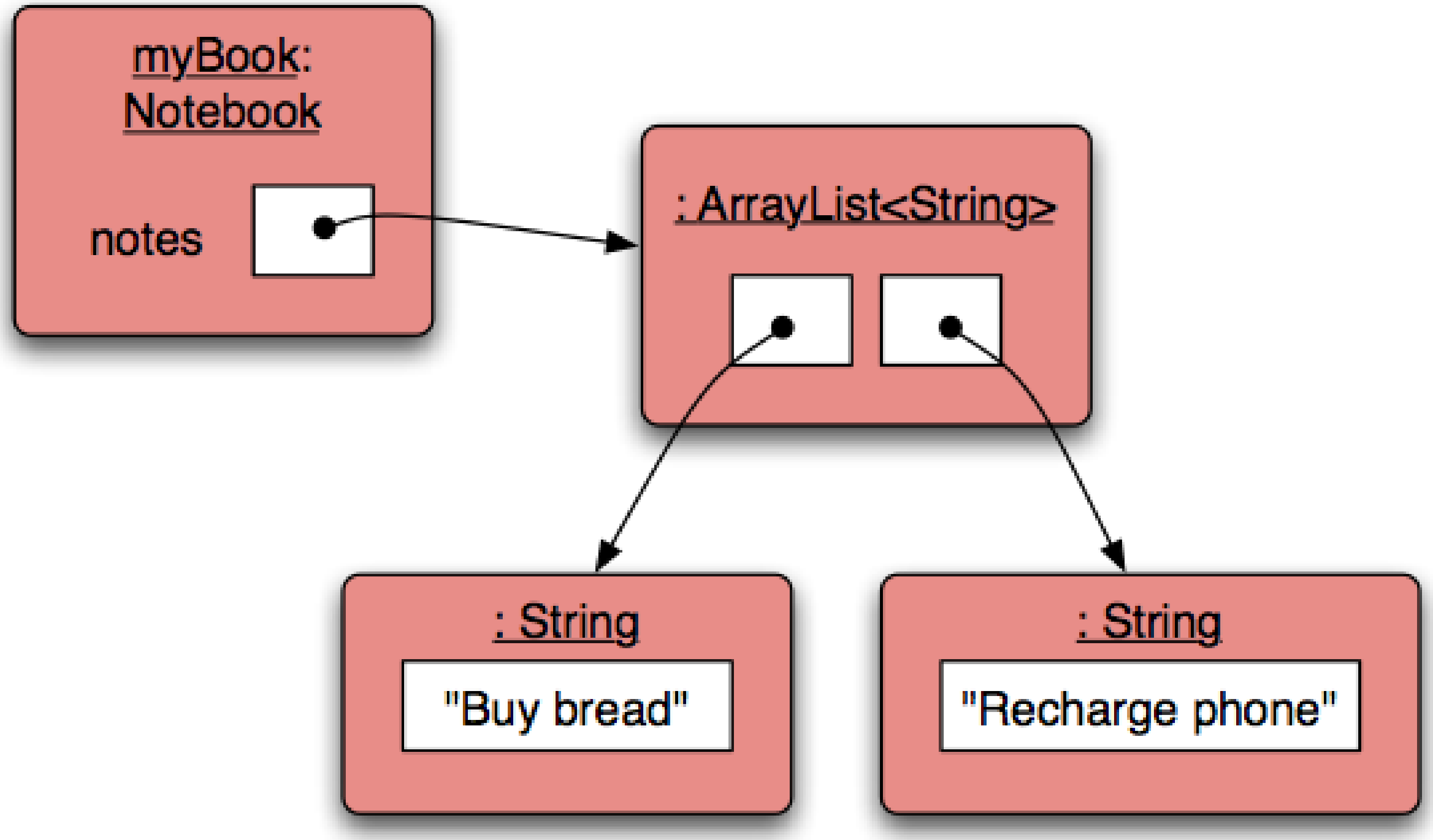
declares notes as a private "ArrayList of <String>"

```
import java.util.ArrayList;

public class Notebook
{

        // Storage for an arbitrary number of notes.
        private ArrayList <String> notes;



        // Perform any initialization required for the notebook.
        public Notebook()
        {
            notes = new ArrayList <String>();
        }

}
```

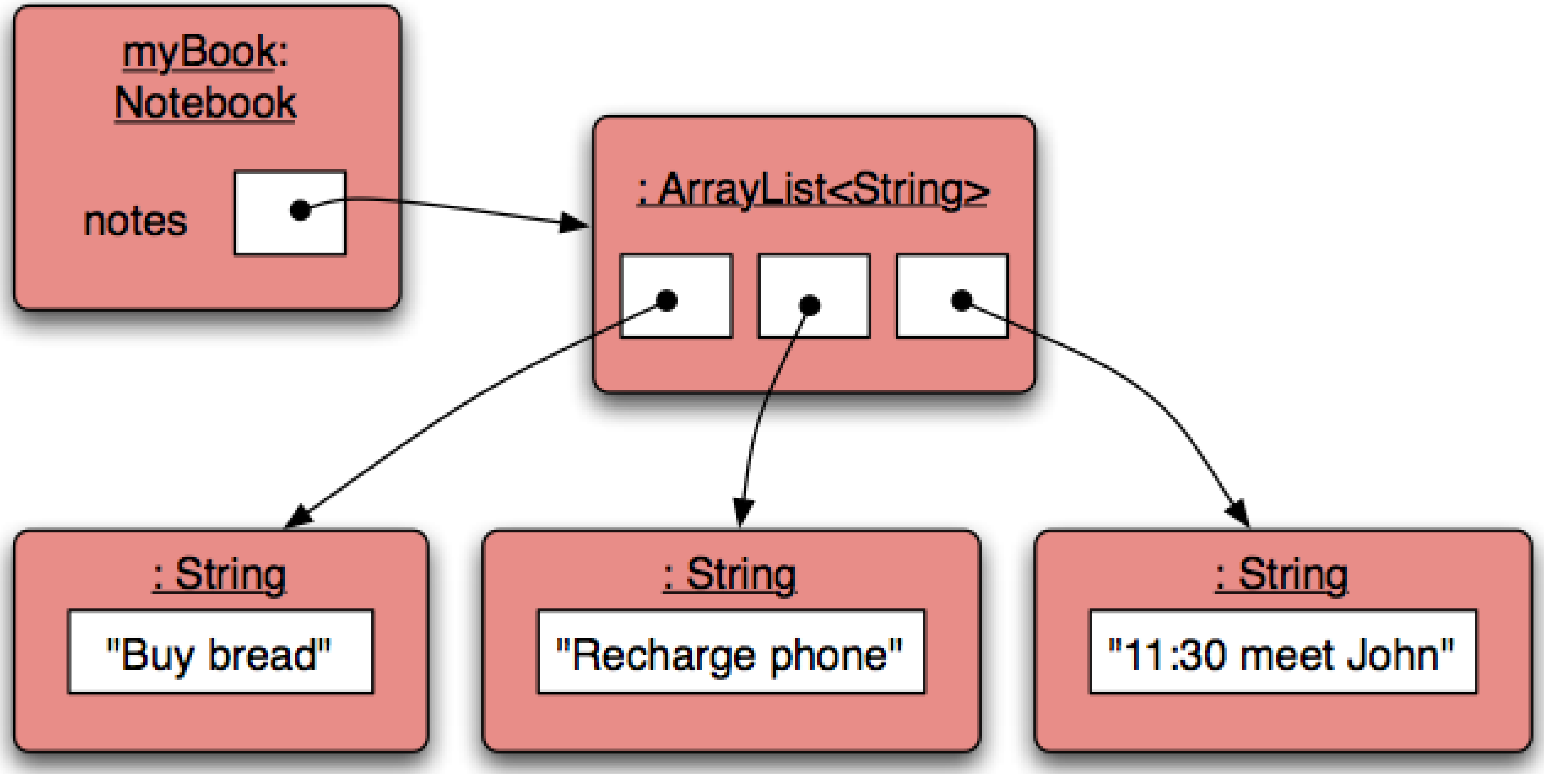
notes is initialised by calling the constructor using new

Note **new** and **()**

Object structures with ArrayList



Adding a third note



Features of the **ArrayList** Collection

- **It increases its capacity as necessary**.
- It keeps a private count
  - `size()` accessor.
- It keeps the objects in **order**.

?

Details of how all this is done are hidden.

- Does that matter?
- Does not knowing how, prevent us from using it?

```
import java.util.ArrayList;

public class Notebook
{
    private ArrayList <String> notes;

    public Notebook(){
            notes = new ArrayList <String> ();
    }

    public void storeNote(String note){
        notes.add(note);
    }

    public int numberOfNotes(){
        return notes.size();
    }
}
```

Adding a new note of type String

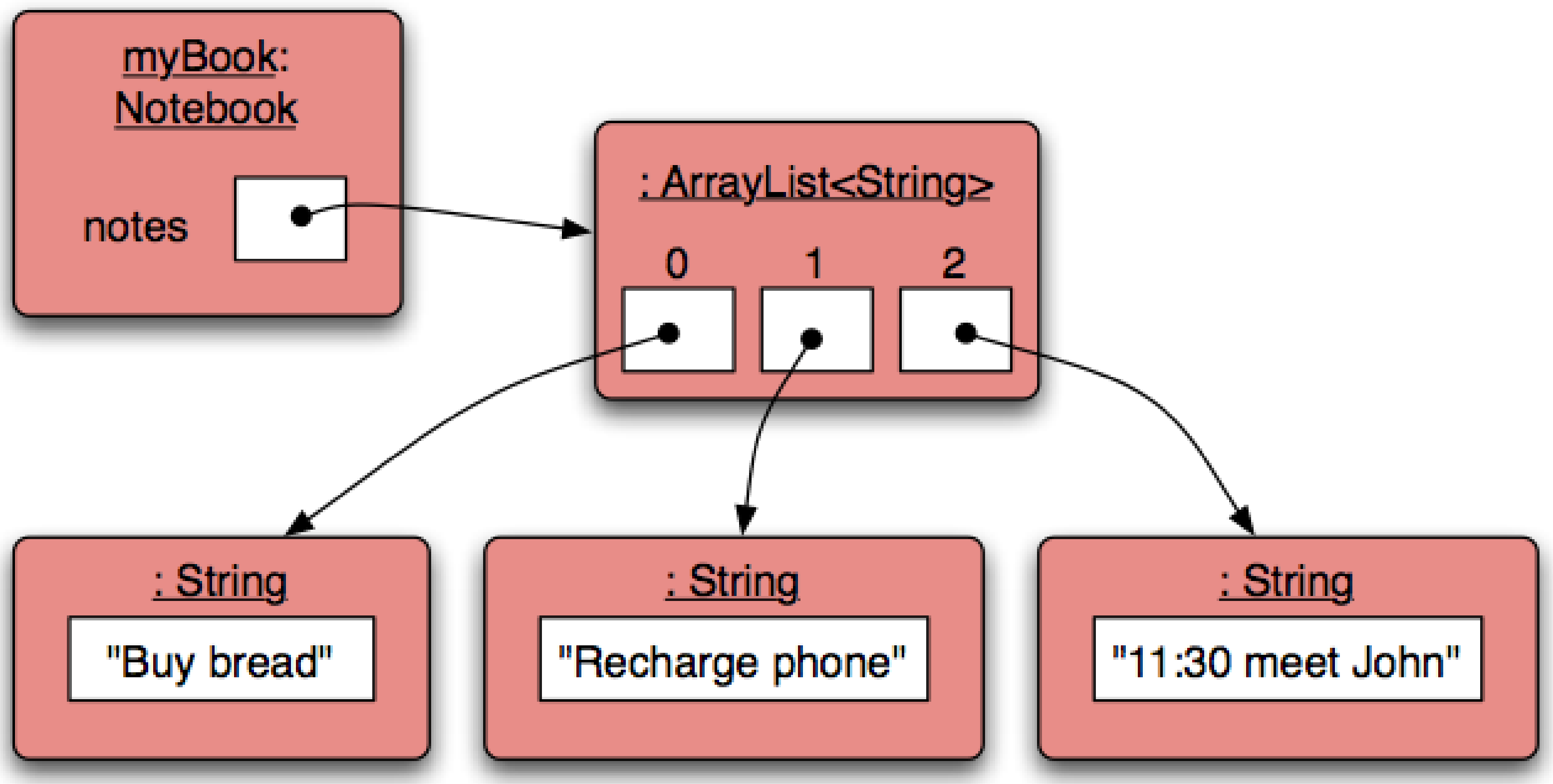
Returning the number of notes

Topic list

1. Grouping Objects
   - Developing a basic personal notebook project using **Collections** e.g. **ArrayList**
2. **Indexing** within Collections
   - Retrieval and removal of objects
3. **Generic classes**
   - e.g. ArrayList
4. **Iteration**
   - Using the for loop
   - Using the while loop
   - Using the **for each** loop

- Next SlideDeck:
  coding a Shop Project that stores an ArrayList of Products.

ArrayList: **Index numbering**



Retrieving an object – **showNote()**

```
public void showNote (int noteNumber)
{
   if(noteNumber < 0) {
      // This is not a valid note number.
   }
   else if(noteNumber < numberOfNotes()) {
      System.out.println(notes.get(noteNumber));
   }
   else {
      // This is not a valid note number.
   }
}
```

Index validity checks

Retrieve and print the note

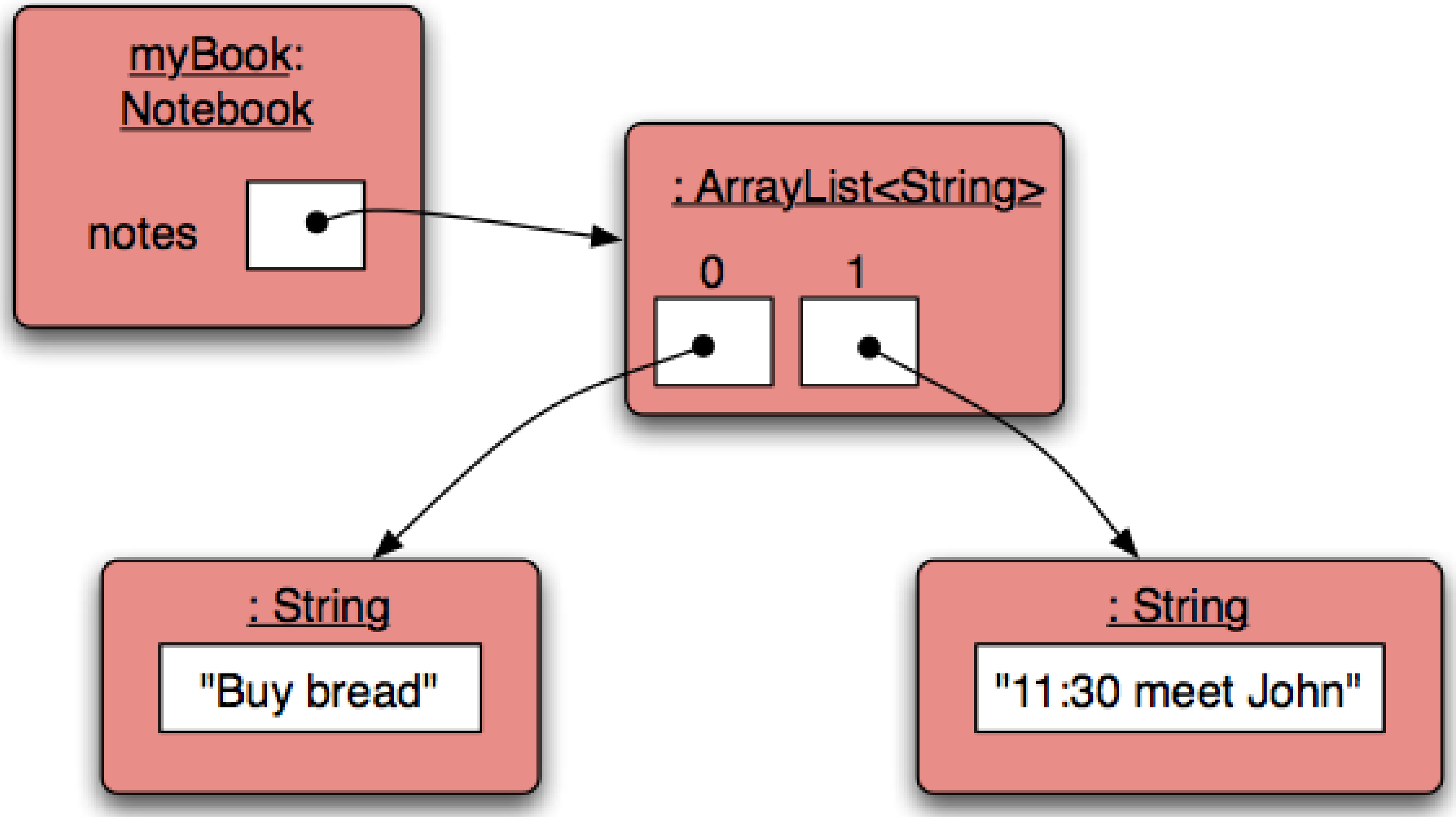
Removing an object

```
public void removeNote(int noteNumber)
  {
     if(noteNumber < 0) {
        // This is not a valid note number, so do nothing.
     }
     else if(noteNumber < numberOfNotes()) {
        // This is a valid note number.
        notes.remove(noteNumber);
     }
     else {
        // This is not a valid note number, so do nothing.
     }
  }
```

Index validity checks

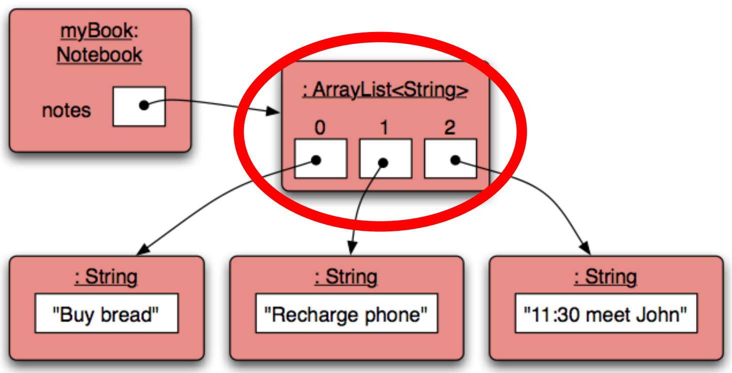
Delete the note at the specific index

Removal may **affect numbering**



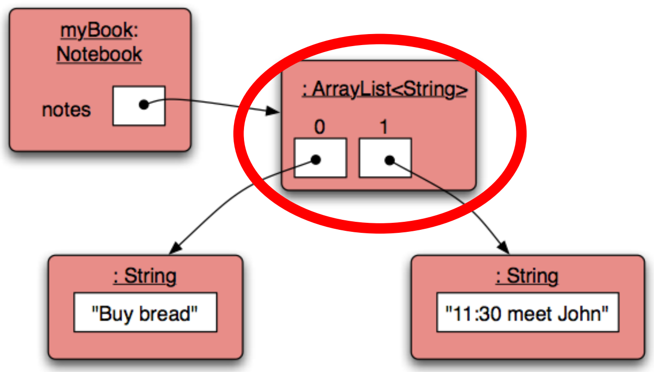
Removal may **affect numbering**

BEFORE



1 2 3

AFTER



1 2

NOTE the change in numbering

Questions?