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.

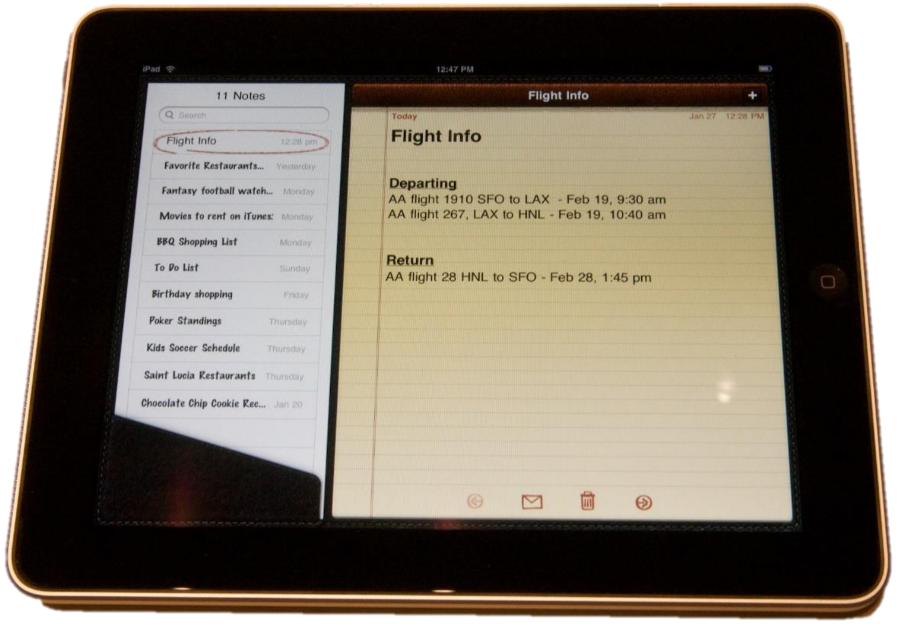
The requirement to **group objects**

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

- The **number of items** to be stored **varies**:
  - Items added.
  - Items deleted.

Example: A personal notebook

- 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.



11 Notes
Flight Info
Favorite Restaurants...
Fantasy football watch...
Movies to rent on iTunes:
BBQ Shopping List
To Do List
Birthday shopping
Poker Standings
Kids Soccer Schedule
Saint Lucia Restaurants
Chocolate Chip Cookie Rec...
Flight Info
Departing
AA flight 1910 SFO to LAX - Feb 19, 9:30 am
AA flight 267, LAX to HNL - Feb 19, 10:40 am
Return
AA flight 28 HNL to SFO - Feb 28, 1:45 pm

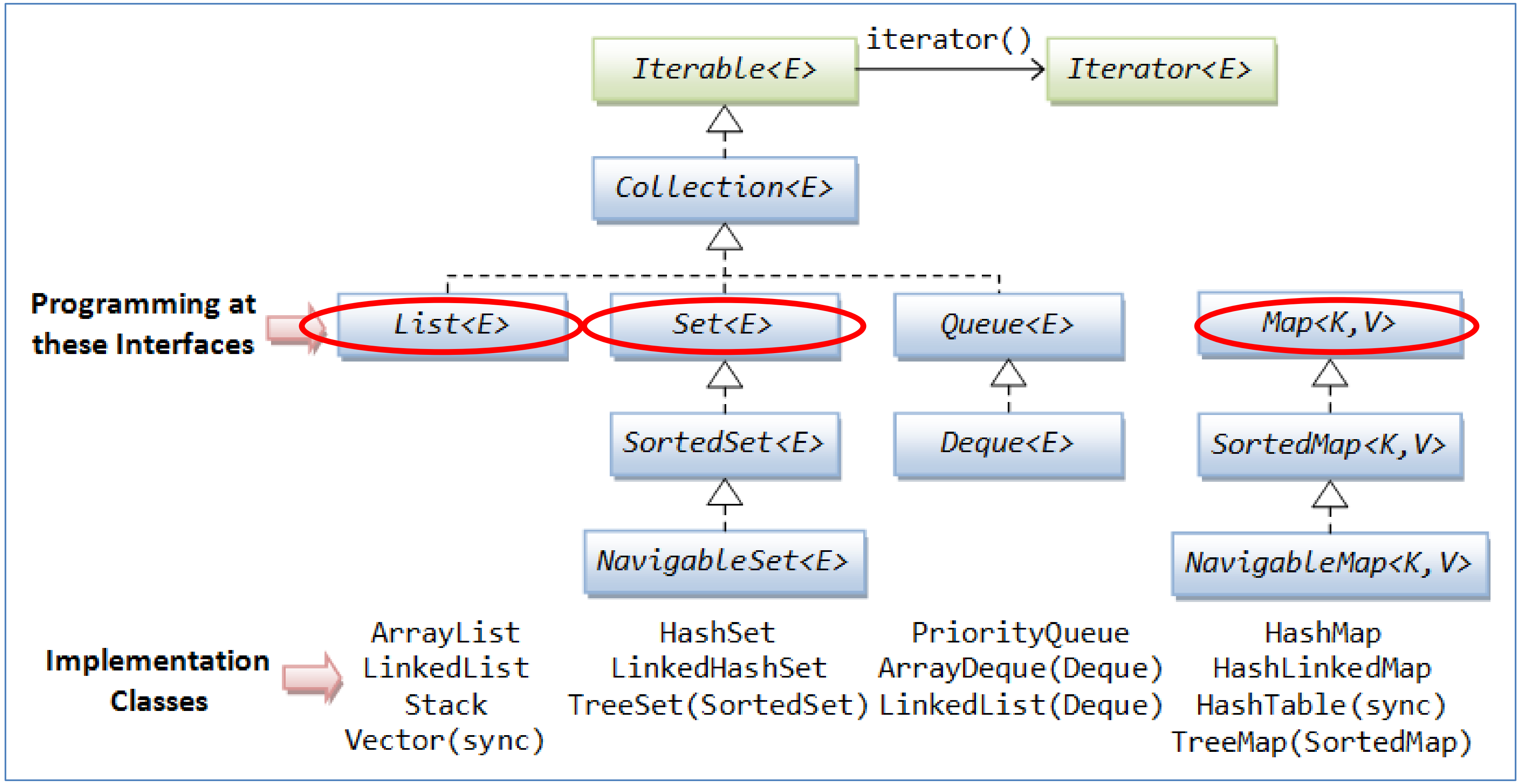
Java API: the class library

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

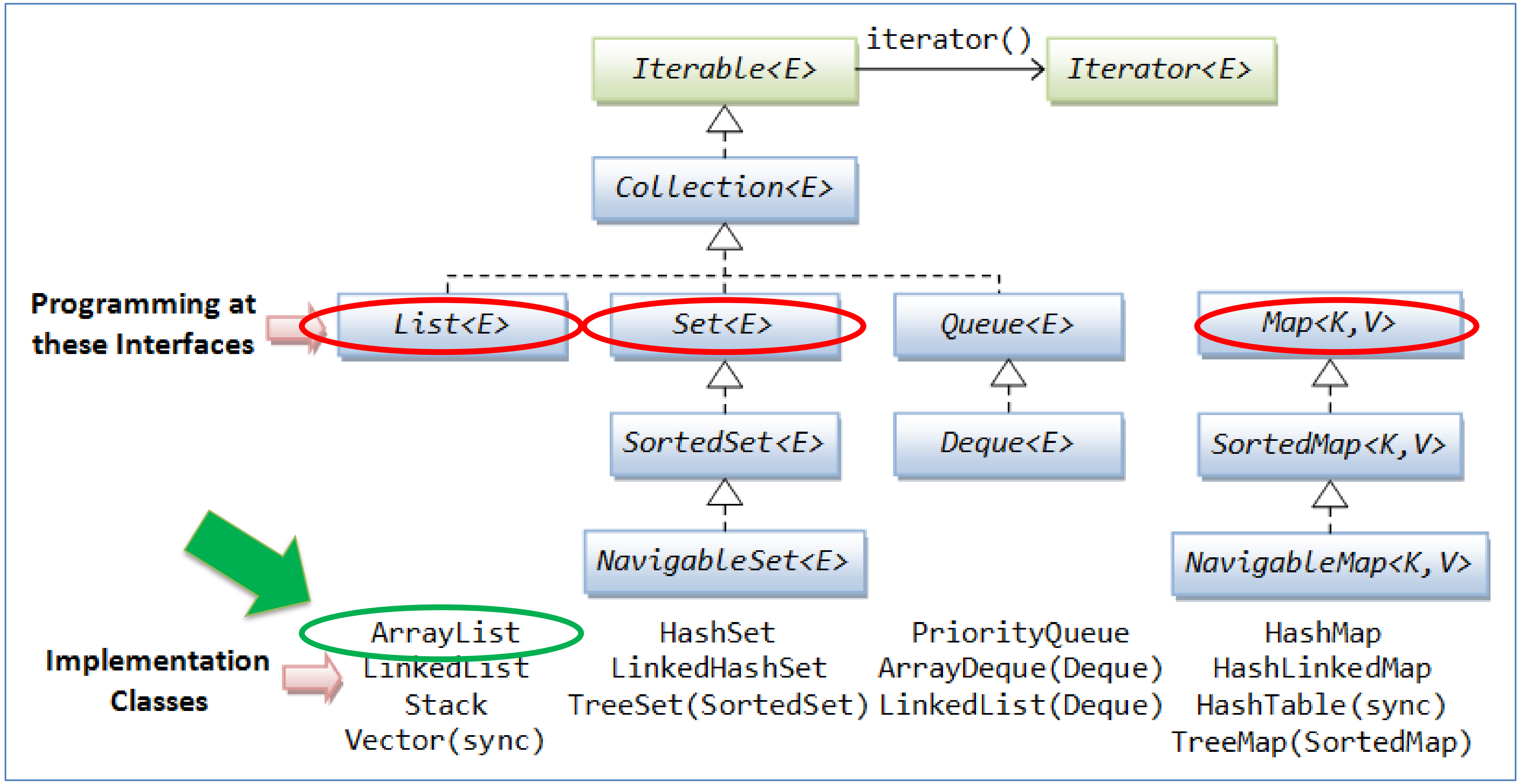
Back to the notebook:

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

Java’s Collections Framework



Java’s Collections Framework



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>();
      }

}
```

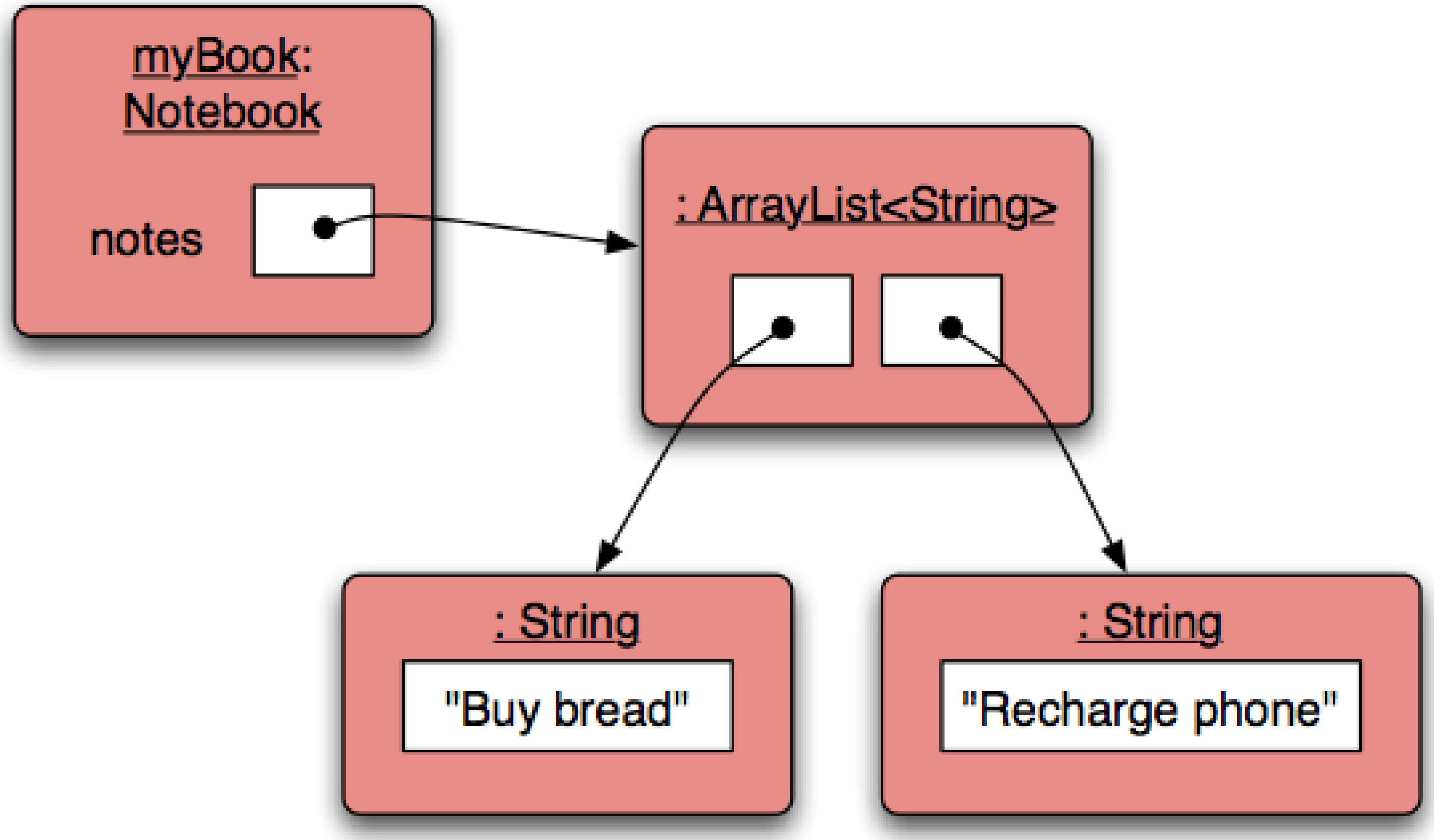
declares notes as a private "ArrayList of <String>"

```
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>();
        }

}
```

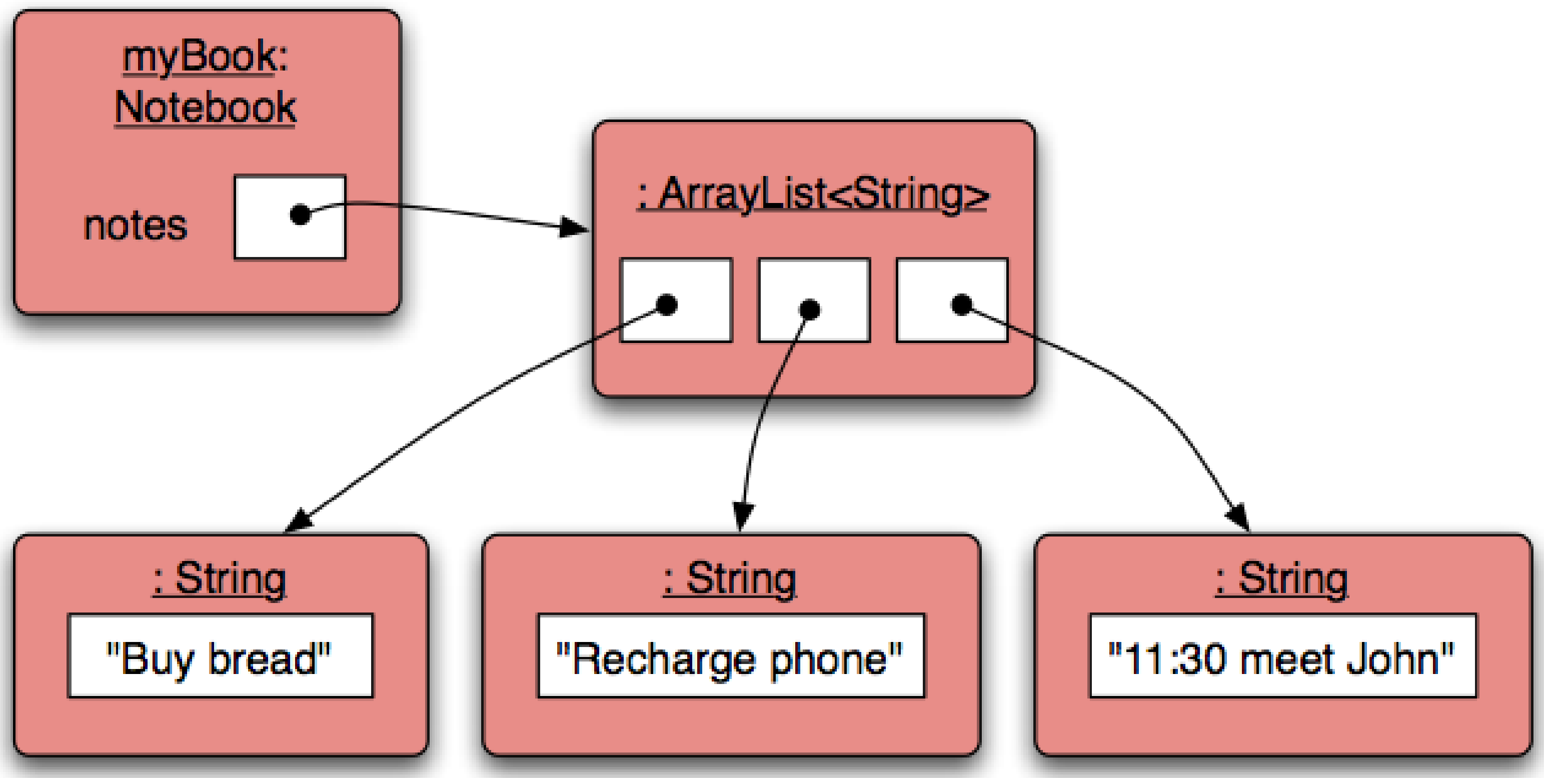
notes is initialised by calling the constructor using new

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

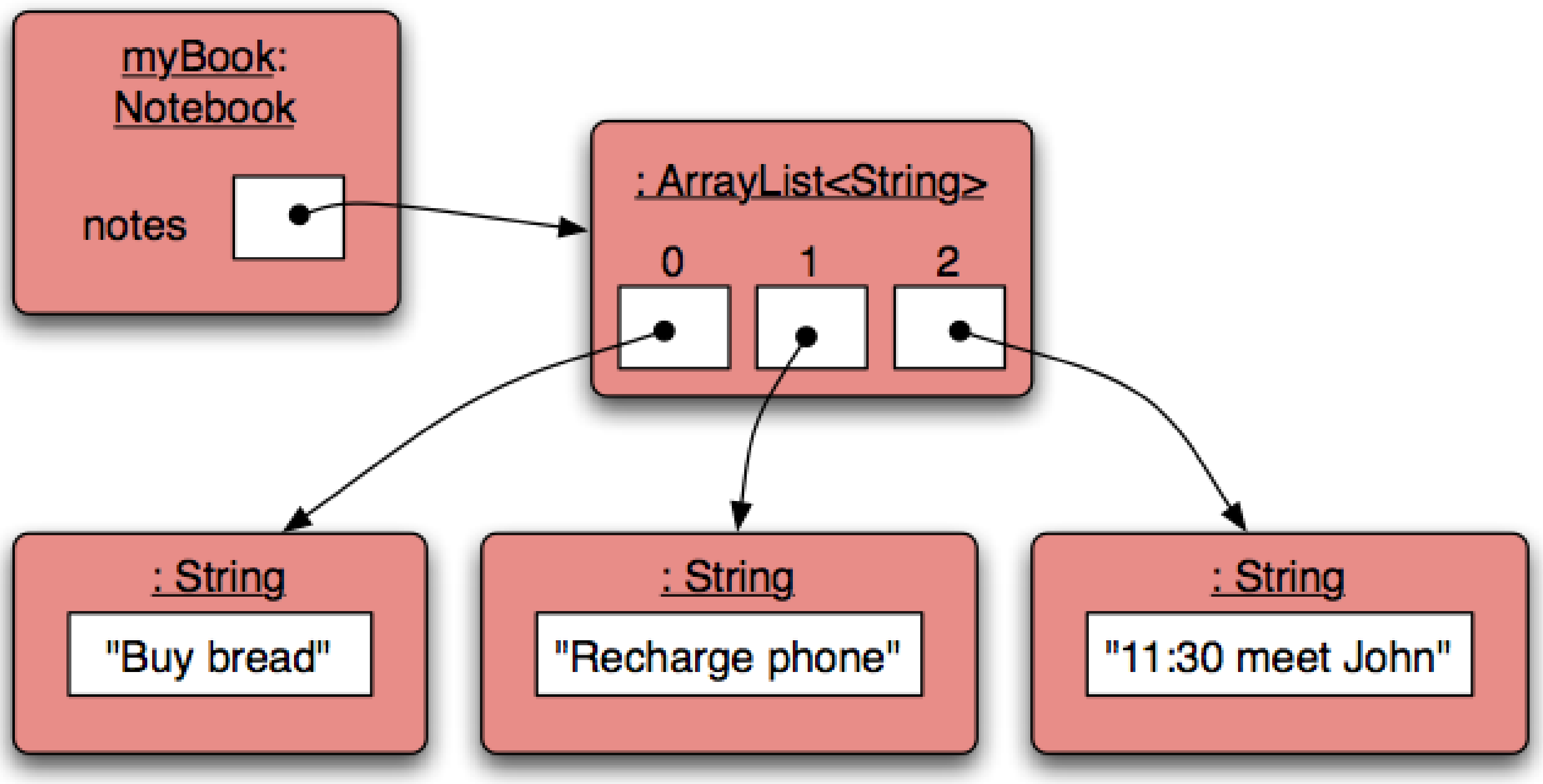
Returning the number of notes

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

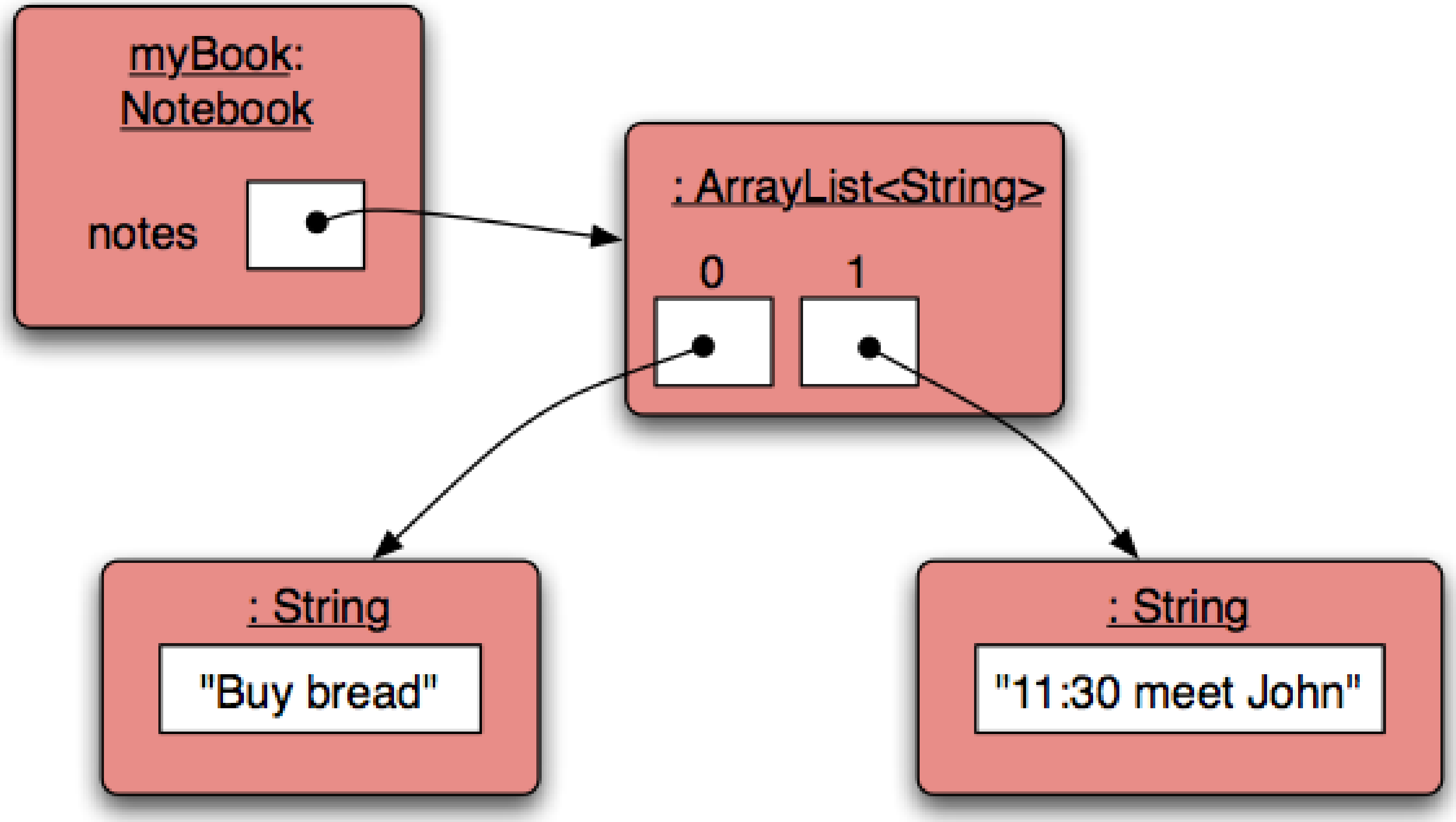
Removing an object

```
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

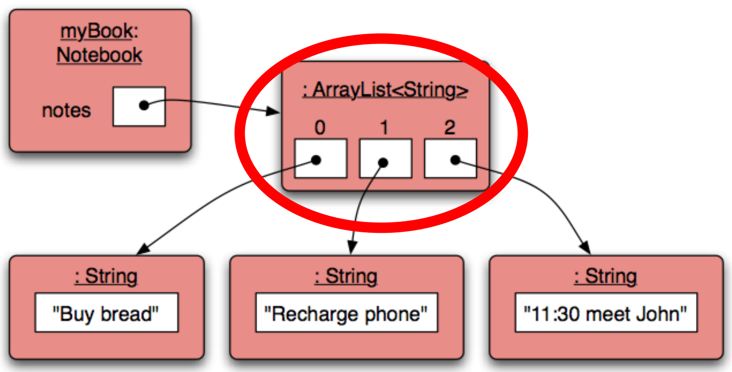
Delete the note at the specific index

Removal may **affect numbering**



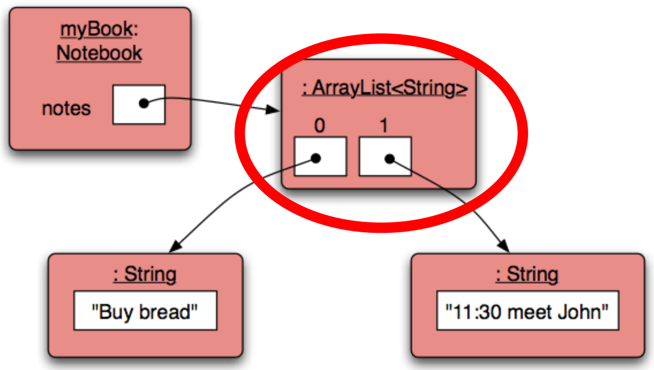
Removal may **affect numbering**

BEFORE



1 2 3

AFTER



1 2

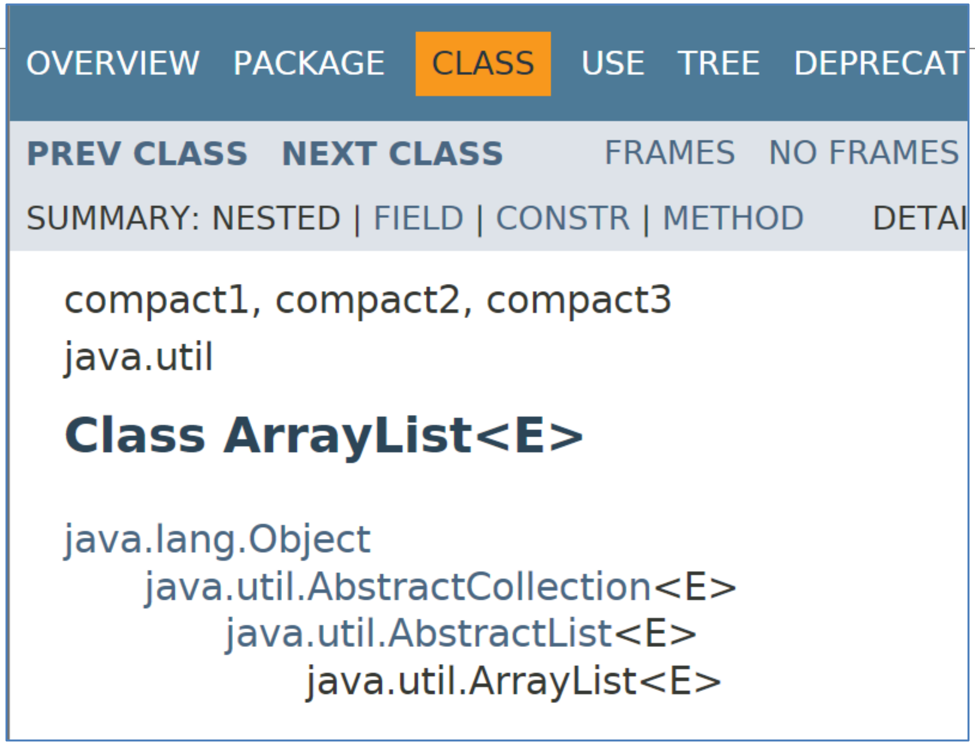
NOTE the change in numbering

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 / Parameterized 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.

Generic/Parameterized Classes


OVERVIEW PACKAGE CLASS USE TREE DEPRECAT
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAI

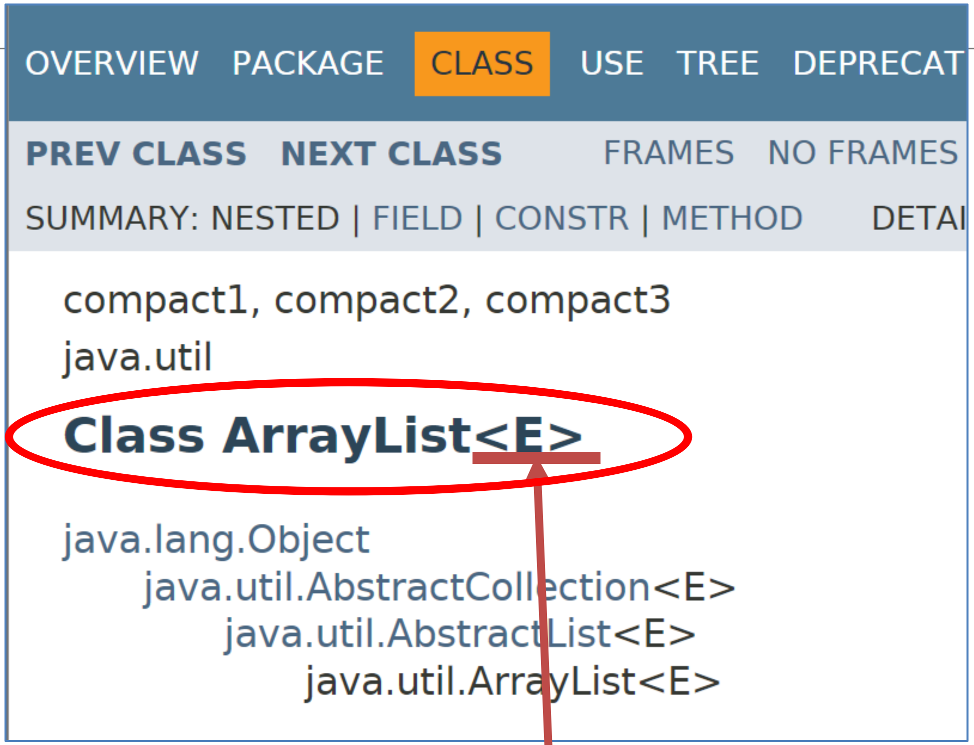
compact1, compact2, compact3

java.util

Class ArrayList<E>

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

Generic/Parameterized Classes


OVERVIEW PACKAGE CLASS USE TREE DEPRECAT

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAI

compact1, compact2, compact3

java.util

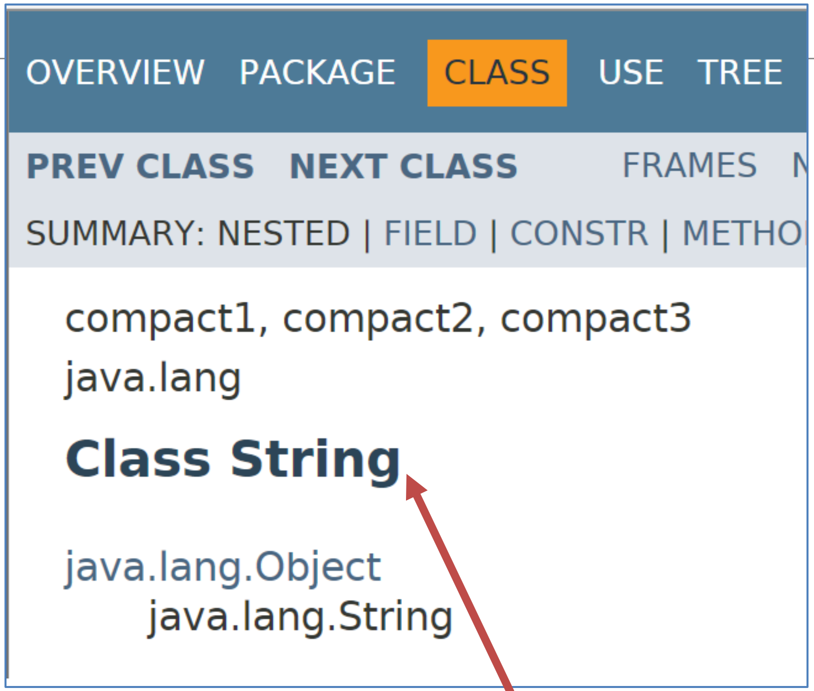
Class ArrayList<E>

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

Collections are known as ***parameterized*** or ***generic*** types.

Note **<E>** is the parameter.

E gets replaced with some Class or Type



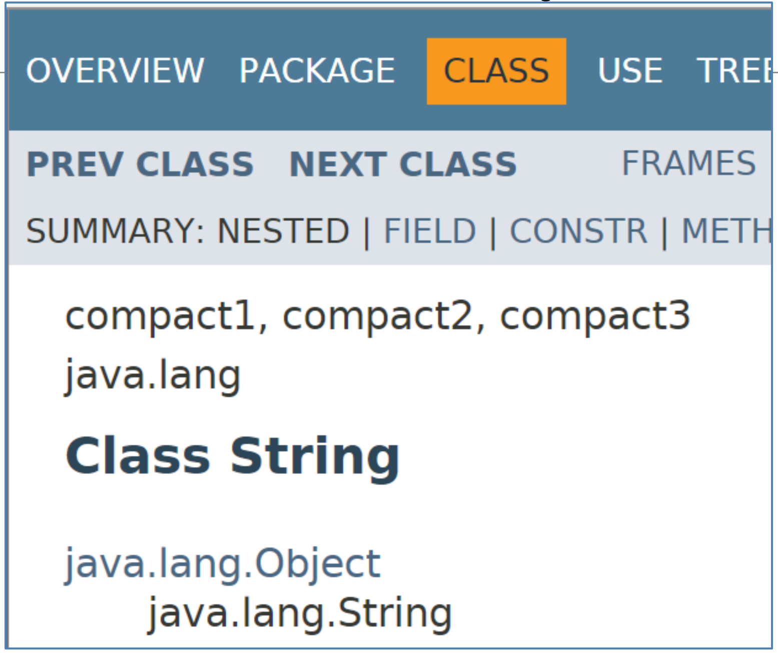
compact1, compact2, compact3
java.lang

Class String

java.lang.Object
    java.lang.String

String is not parameterized.

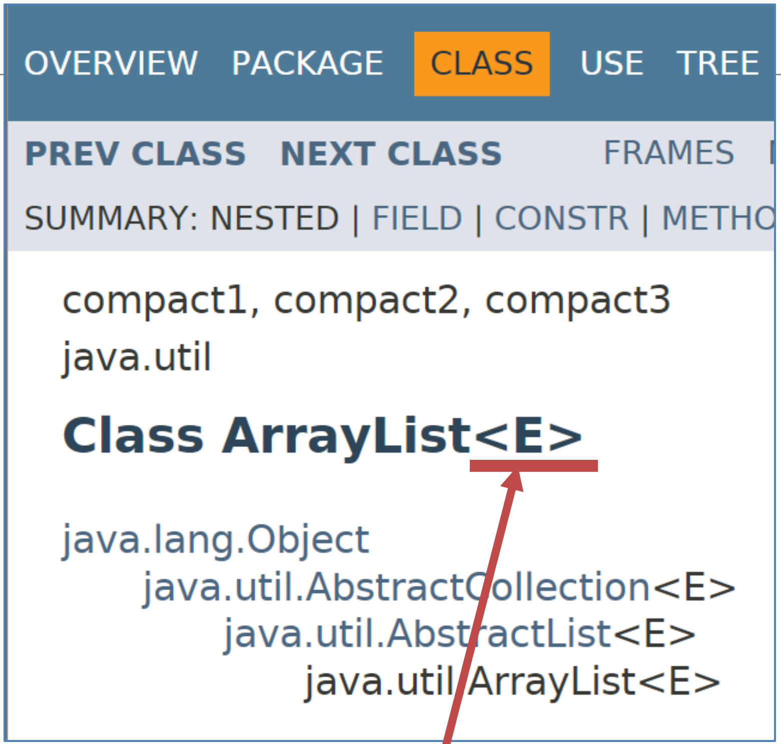
Generic/Parameterized Classes


OVERVIEW PACKAGE CLASS USE TREE
PREV CLASS NEXT CLASS FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METH

compact1, compact2, compact3
java.lang

Class String

java.lang.Object
    java.lang.String


OVERVIEW PACKAGE CLASS USE TREE
PREV CLASS NEXT CLASS FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHO

compact1, compact2, compact3
java.util

Class ArrayList<E>

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

String is not parameterized.

ArrayList is parameterized.

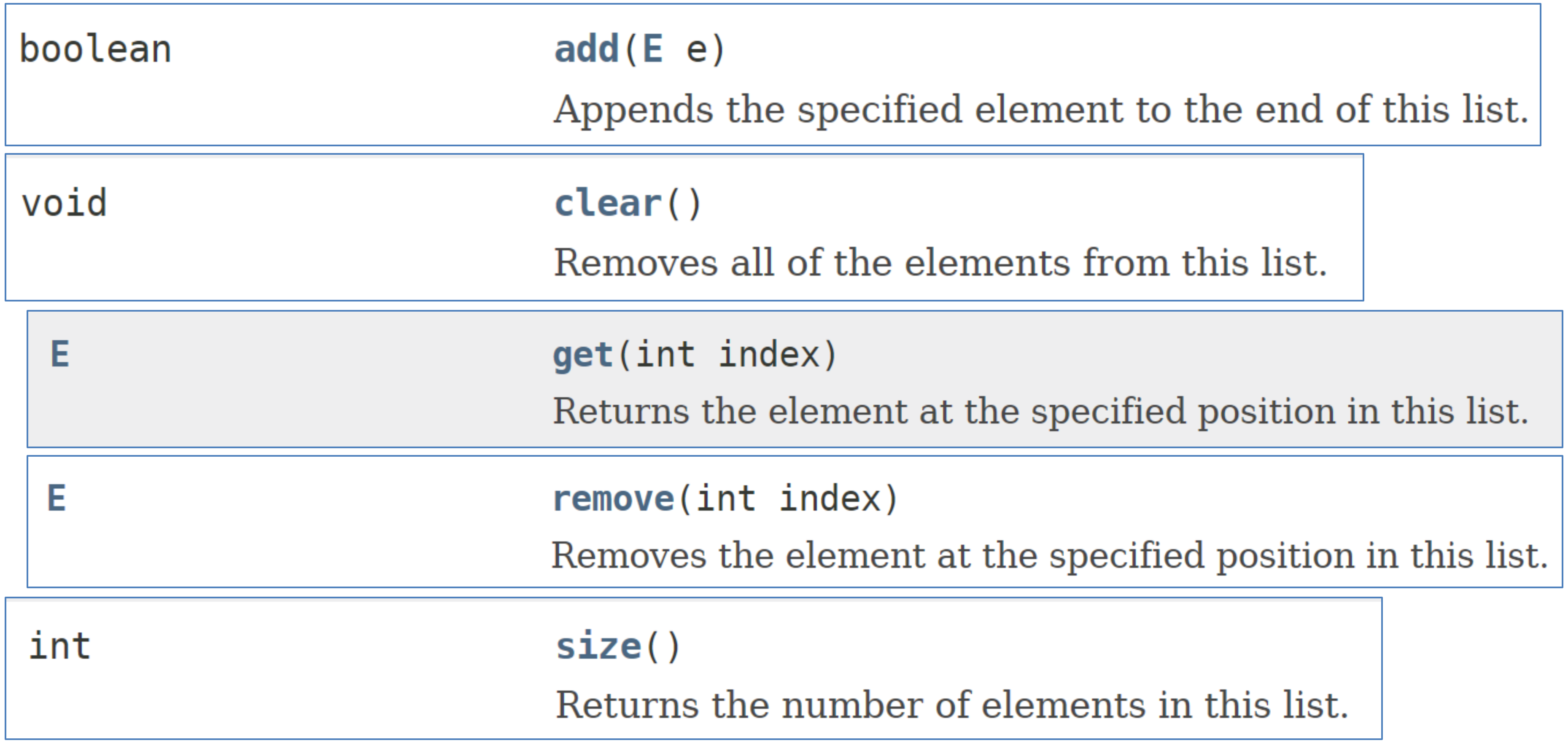
The **type parameter <E>**
says what we want a list of e.g.:

```
ArrayList<Person>
ArrayList<TicketMachine>
ArrayList<String>
```

etc.

Generic/Parameterized classes

- **`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.

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.

Processing a whole collection (**iteration**)

- We often want to perform some actions an **arbitrary** number of times.
  - E.g.,
    Print all the notes in the notebook.
    How many are there?
    Does the amount of notes in our notebook vary?

- Most programming languages include ***loop statements*** to make this possible.

- **Loops** enable us to **control how many times we repeat** certain actions.

Loops in Programming

- There are three types of standard loops in (Java) programming:
  - **while**
  - **for**
  - **do while**

- You typically use **for** and **while** loops to iterate over your ArrayList collection,

OR

- you can use another special construct associated with Collections:
  - **for each**

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

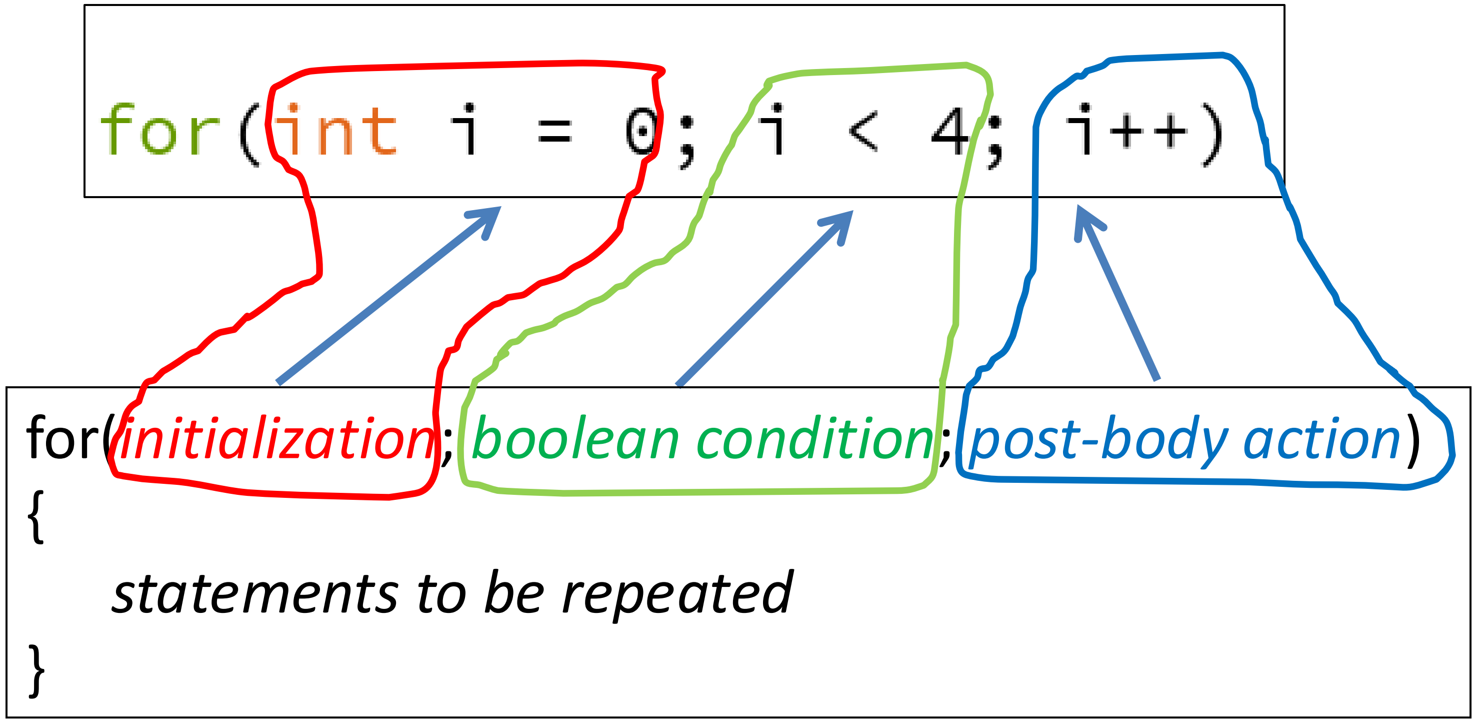
- Next SlideDeck:
  coding a Shop Project that stores an ArrayList of Products.

Recap: **for loop** pseudo-code

General form of a for loop

```
for(initialization; boolean condition; post-body action)
{
    statements to be repeated
}
```

Recap: **for loop** syntax



```
for(int i = 0; i < 4; i++)
```

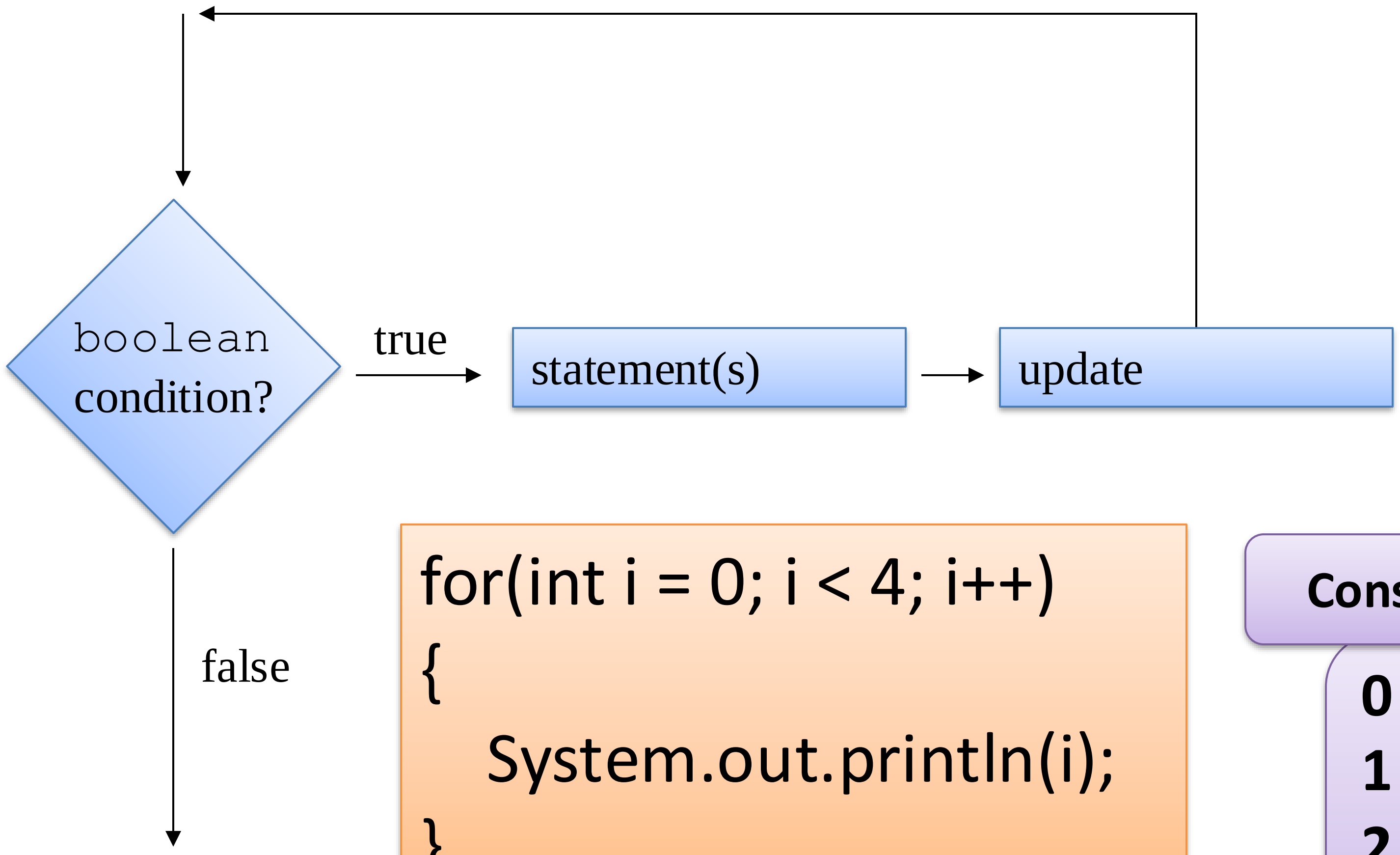
```
for(initialization; boolean condition; post-body action)
{
    statements to be repeated
}
```

Recap: for loop syntax

```
for(int i = 0; i < 4; i++)
```

initialization	int i = 0;	Initialise a loop control variable (LCV) e.g. i. It can include a variable declaration.
boolean condition	i < 4;	Is a valid boolean condition that typically tests the loop control variable (LCV).
post-body action	i++	A change to the loop control variable (LCV). Contains an assignment statement.

Recap: **for loop** flowchart

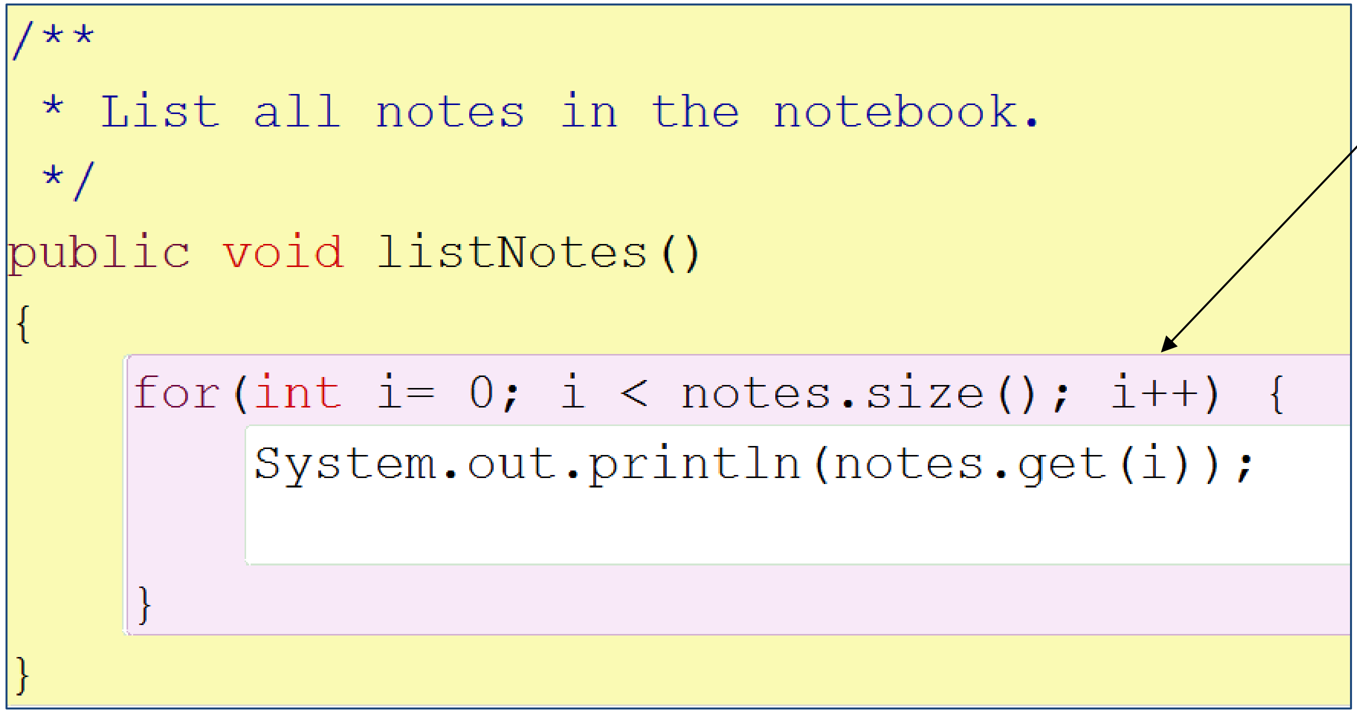


```
for(int i = 0; i < 4; i++)
{
    System.out.println(i);
}
```

Console Output:

```
0
1
2
3
```

for loop: for **iterating** over **a collection**



```
/**
 * List all notes in the notebook.
 */
public void listNotes()
{
    for(int i= 0; i < notes.size(); i++) {
        System.out.println(notes.get(i));

    }
}
```

Increment *index* by 1

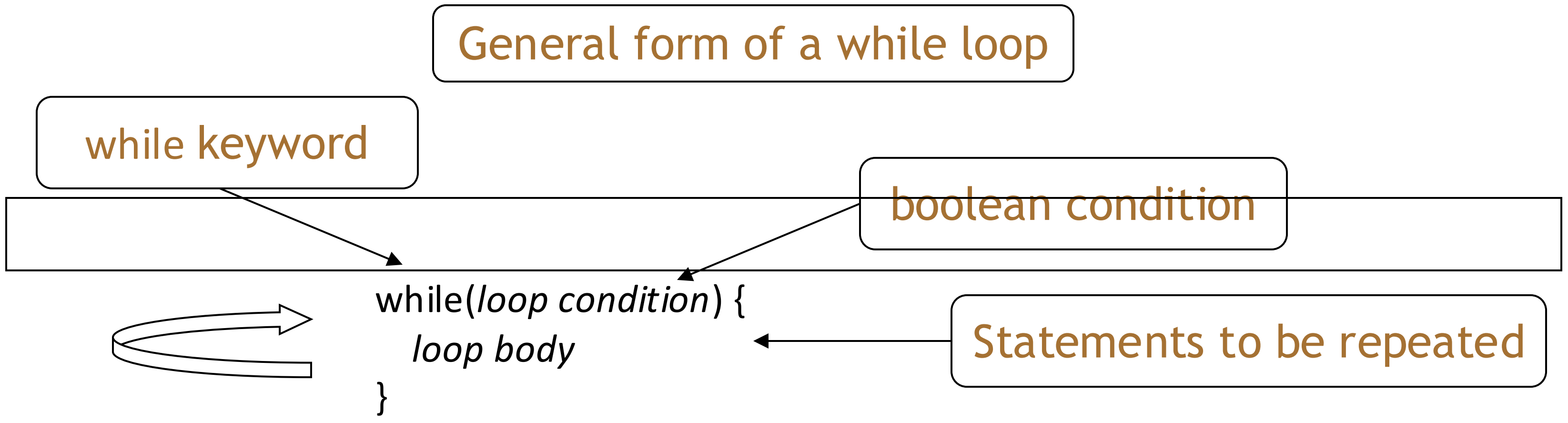
for each value of *i* less than the size of the collection, print the next note, and then increment *i*

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.

Recap: **while loop** pseudo code



```
while(*loop condition*) {
   *loop body*
}
```

Pseudo-code expression of the actions of a while loop

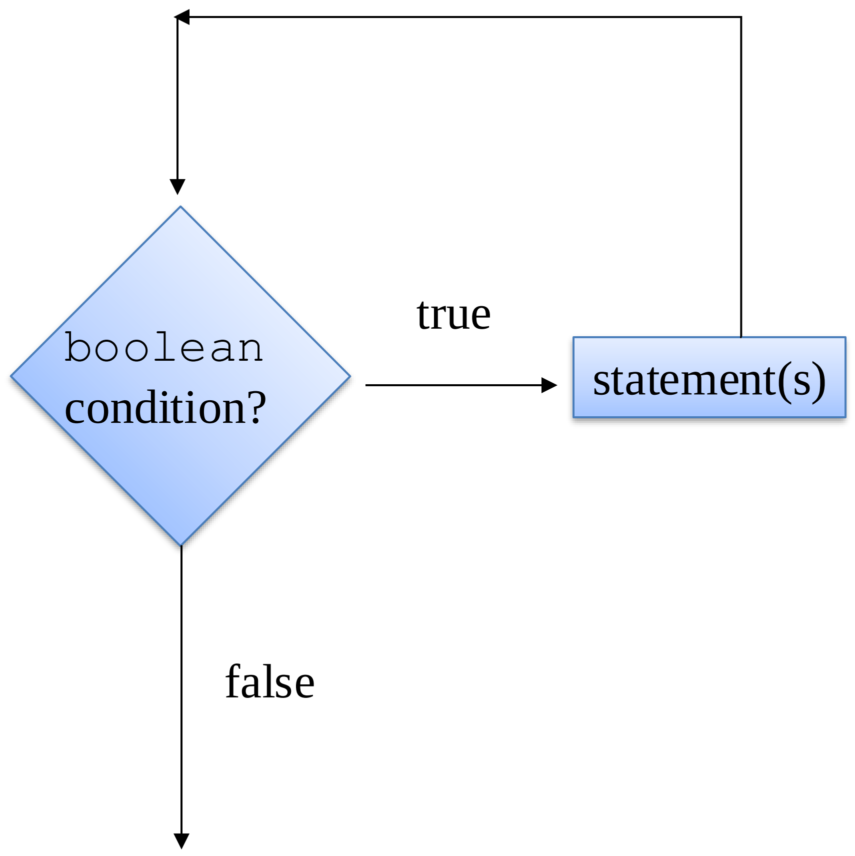
while we wish to continue, do the things in the loop body

Recap: **while loop** construction

```
Declare and initialise loop control variable (LCV)
while(condition based on LCV)
{
    “do the job to be repeated”
    “update the LCV”
}
```

This structure should always be used

Recap: **while loop** flowchart



```
int i = 1;
while (i <= 10)
{
     System.out.println(i);
     i++;
}
```

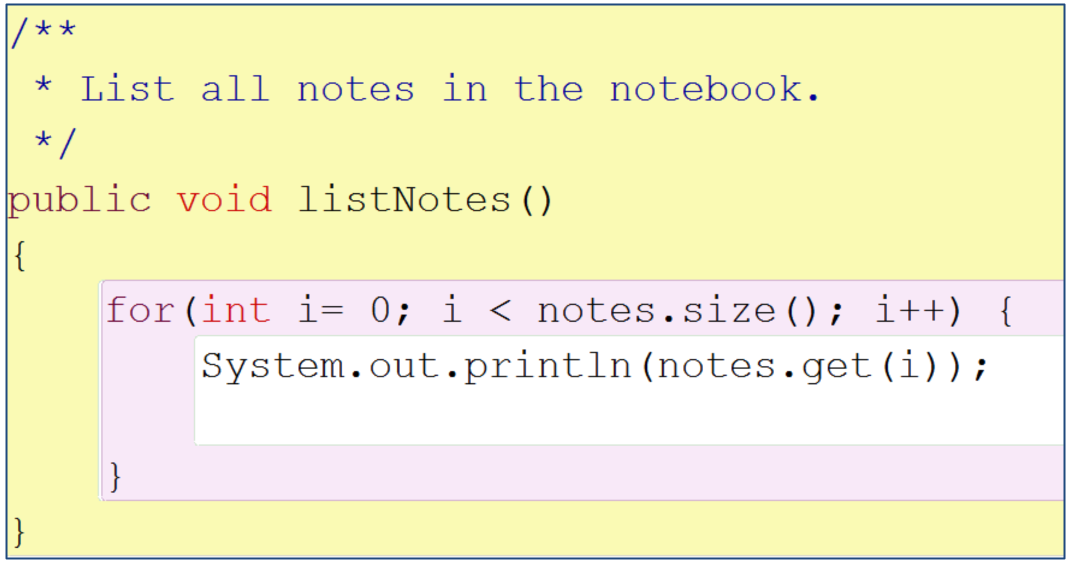
while loop: **iterating** over **a collection**

```
/**
 * List all notes in the notebook.
 */
public void listNotes()
{
    int i = 0;
    while(i < notes.size()) {
        System.out.println(notes.get(i));
        i++;
    }
}
```

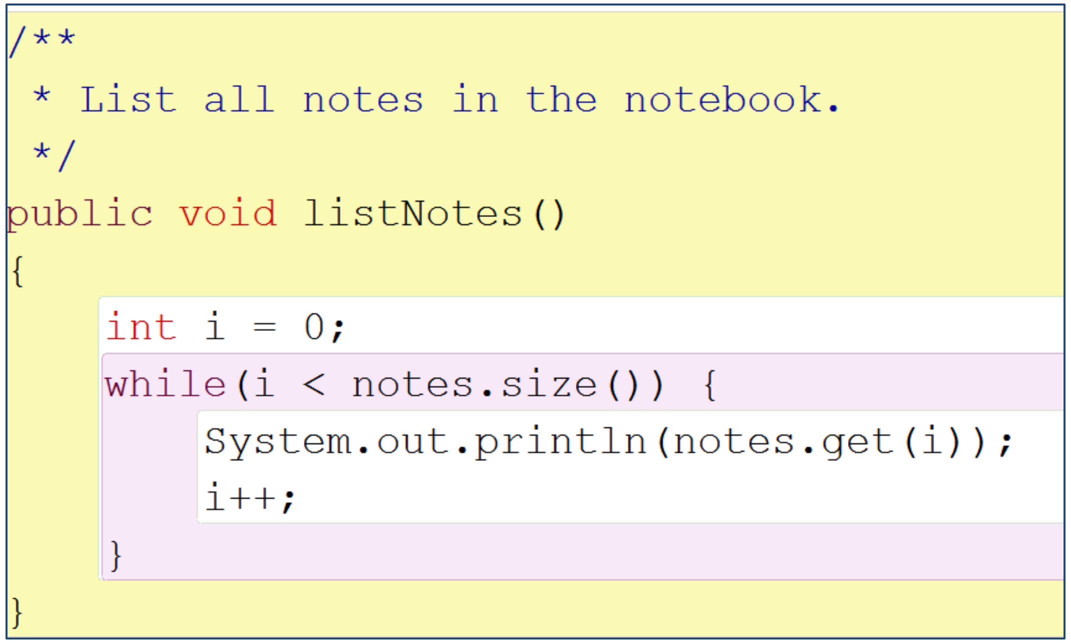
Increment *i* by 1

while the value of *i* is less than the size of the collection, print the next note, and then increment *i*

`for` versus `while`



```
/**
 * List all notes in the notebook.
 */
public void listNotes()
{
    for(int i= 0; i < notes.size(); i++) {
        System.out.println(notes.get(i));

    }
}
```



```
/**
 * List all notes in the notebook.
 */
public void listNotes()
{
    int i = 0;
    while(i < notes.size()) {
        System.out.println(notes.get(i));
        i++;
    }
}
```

Variable **i** is the
Loop Control Variable (**LCV**).
It must be initialised, tested and changed.

int i = 0 is the **initialisation**.

i < notes.size() is the **test**.

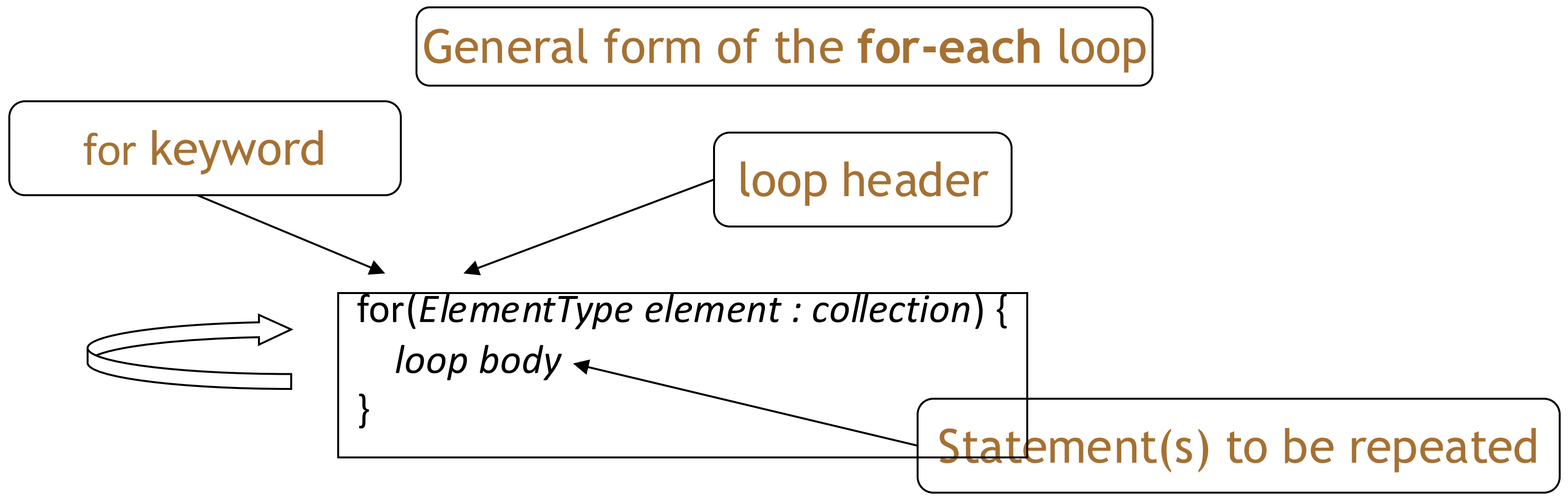
i++ is the post-body action
i.e. the **change**.

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.

for each loop: pseudo code

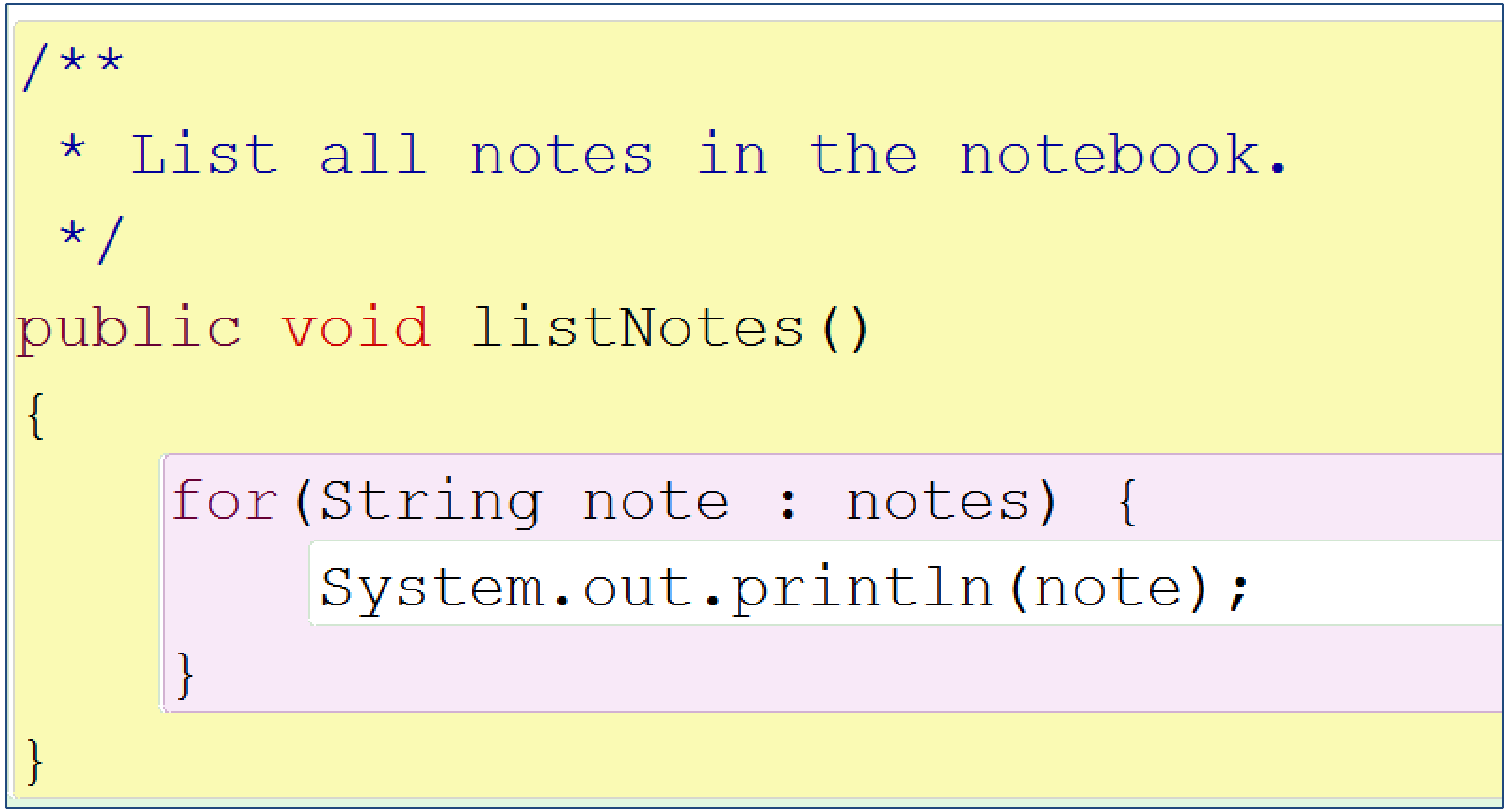

General form of the **for-each** loop
for keyword
loop header
Statement(s) to be repeated

```
for(ElementType element : collection) {
    loop body
}
```

Pseudo-code expression of the actions of a **for-each** loop

For each *element* in *collection*, do the things in the *loop body*.

for each loop: **iterating** over **a collection**



```
/**
 * List all notes in the notebook.
 */
public void listNotes()
{
    for(String note : notes) {
        System.out.println(note);
    }
}
```

for each *note* in the *notes* collection, print out *note*

for each loop

- Can only be used for **access**;
  - you can’t remove the retrieved elements.
- Can only loop forward in single steps.
- Cannot use to compare two collections.

for each **versus** while

- for-each:
  - **easier to write**.
  - **safer**: it is guaranteed to stop.

- while:
  - we **don’t *have* to process the whole collection**.
  - doesn’t even have to be used with a collection.
  - take care: could be an *infinite loop*.

Summary

- Java **Collections** Framework
  - ArrayList
- `import java.util.ArrayList;`
- `private ArrayList <String> notes;`
- `notes = new ArrayList <String>();`
- `notes.add(note);`
- `notes.size();`
- `notes.get(noteNumber)`
- `notes.remove(noteNumber);`
- **Iterating collections**
  - **for each**
    - `for (String note : notes) {System.out.println(note);}`

Questions?