Grouping Objects

ArrayList (generic classes) and Iteration

Produced by: Dr. Siobhán Drohan

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

Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

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

Topic list

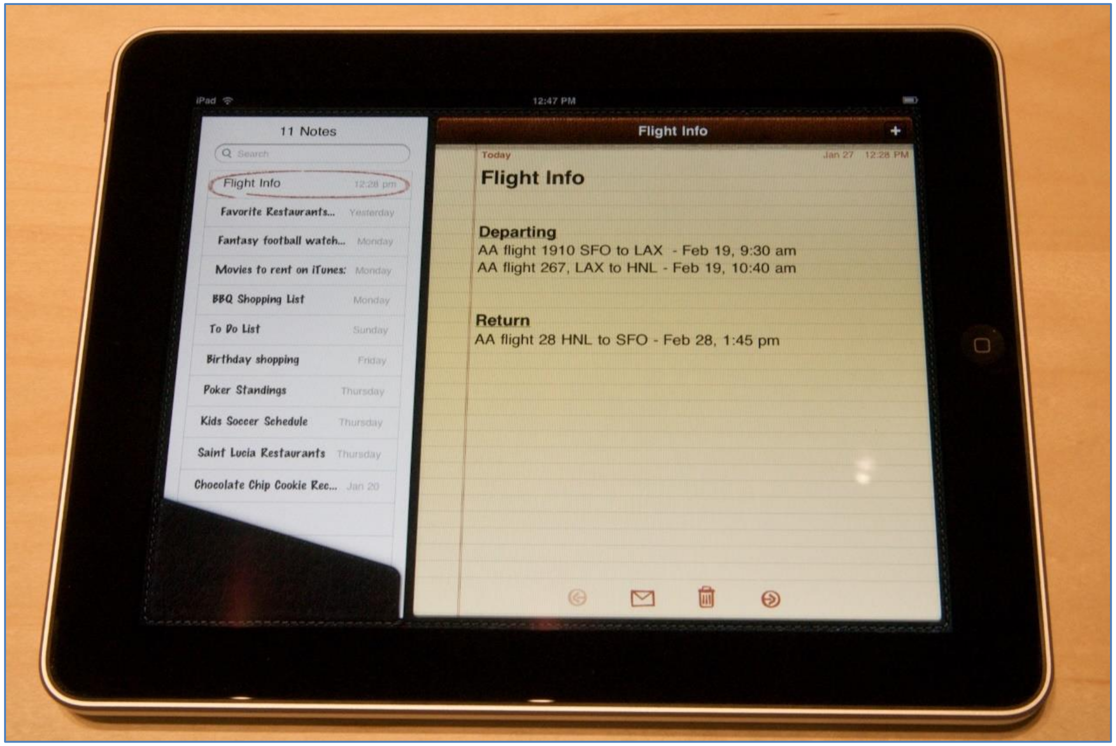
- Grouping Objects
  - Developing a basic personal notebook project using Collections e.g. ArrayList
- Indexing within Collections
  - Retrieval and removal of objects
- Generic classes e.g. ArrayList
- Iteration
  - Using the for loop
  - Using the while loop
  - Using the for each loop
  - Using the Iterator
- 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 will tell how many notes are stored.



iPad
12:47 PM
11 Notes
Search
Flight Info 12:28 pm
Favorite Restaurants... Yesterday
Fantasy football watch... Monday
Movies to rent on iTunes: Monday
BBQ Shopping List Monday
To Do List Sunday
Birthday shopping Friday
Poker Standings Thursday
Kids Soccer Schedule Thursday
Saint Lucia Restaurants Thursday
Chocolate Chip Cookie Rec... Jan 20
Flight Info
+
Today
Jan 27 12:28 PM

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

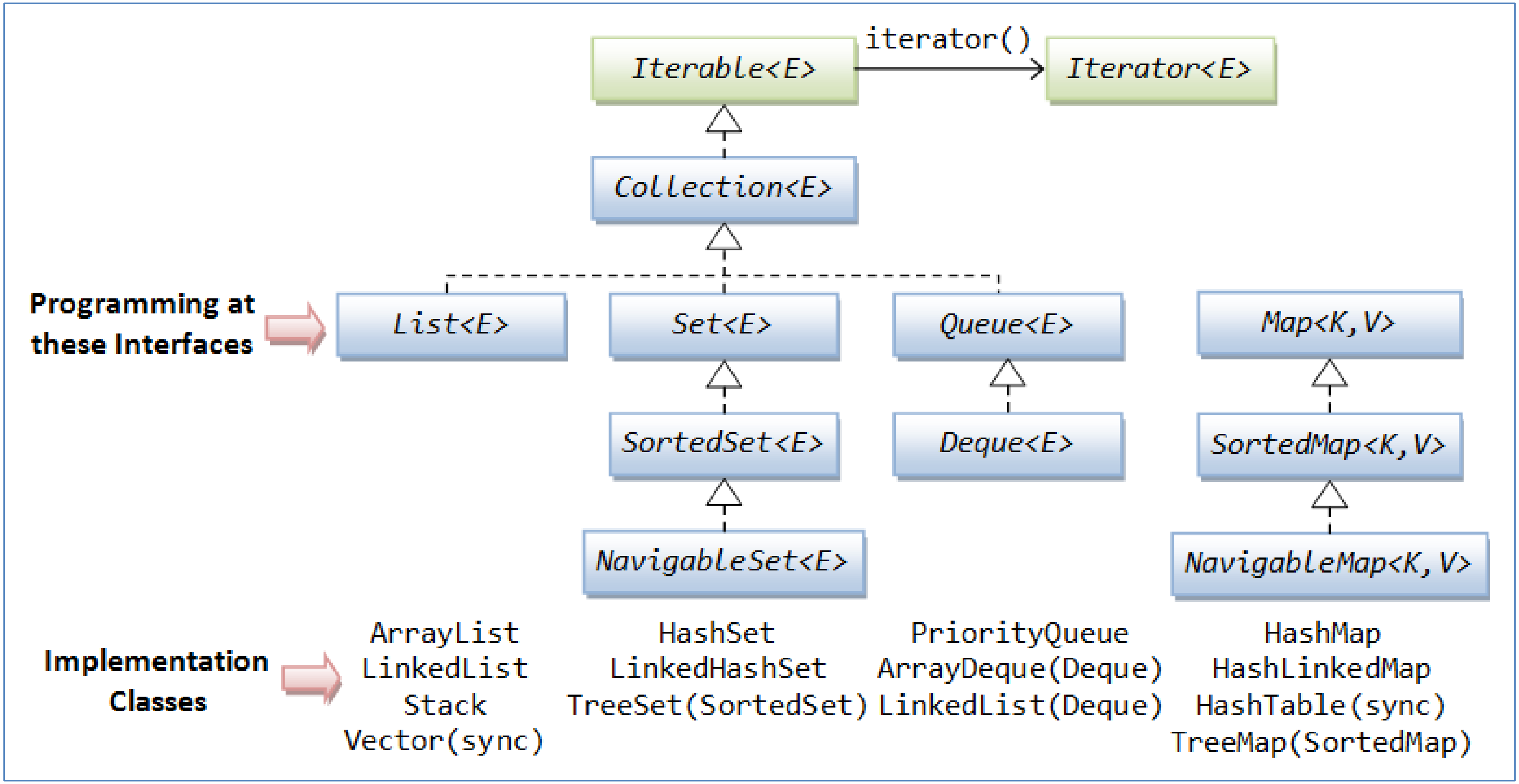
Java API: the class library

- 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



Iterable<E>

iterator()

Iterator<E>

Collection<E>

Programming at these Interfaces

List<E>

Set<E>

Queue<E>

Map<K,V>

SortedSet<E>

Deque<E>

SortedMap<K,V>

NavigableSet<E>

NavigableMap<K,V>

Implementation Classes

ArrayList
LinkedList
Stack
Vector(sync)

HashSet
LinkedHashSet
TreeSet(SortedSet)

PriorityQueue
ArrayDeque(Deque)
LinkedList(Deque)

HashMap
HashLinkedMap
HashTable(sync)
TreeMap(SortedMap)

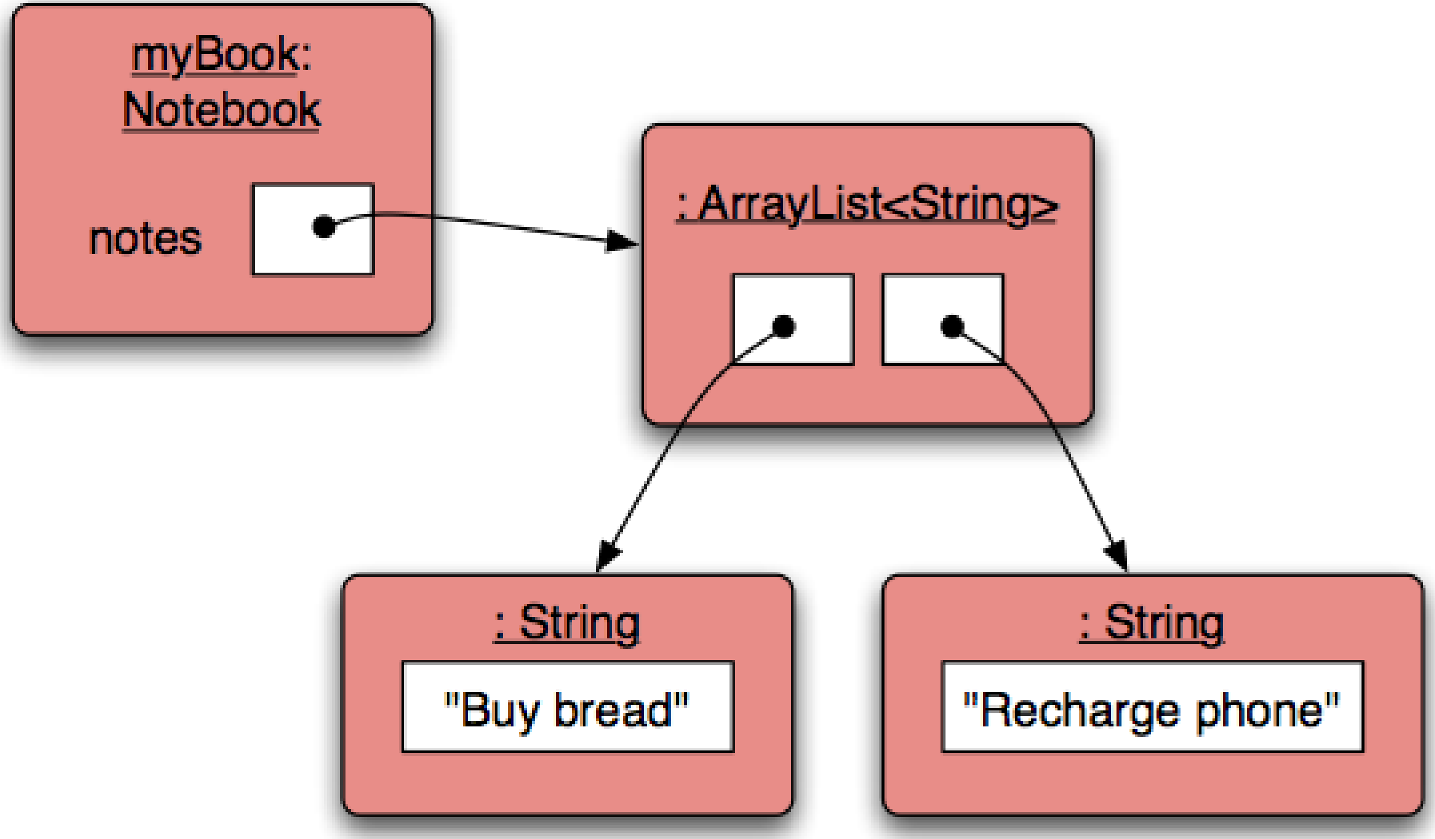
```java
import java.util.ArrayList;

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>();
        }

}
```

ArrayList Collection

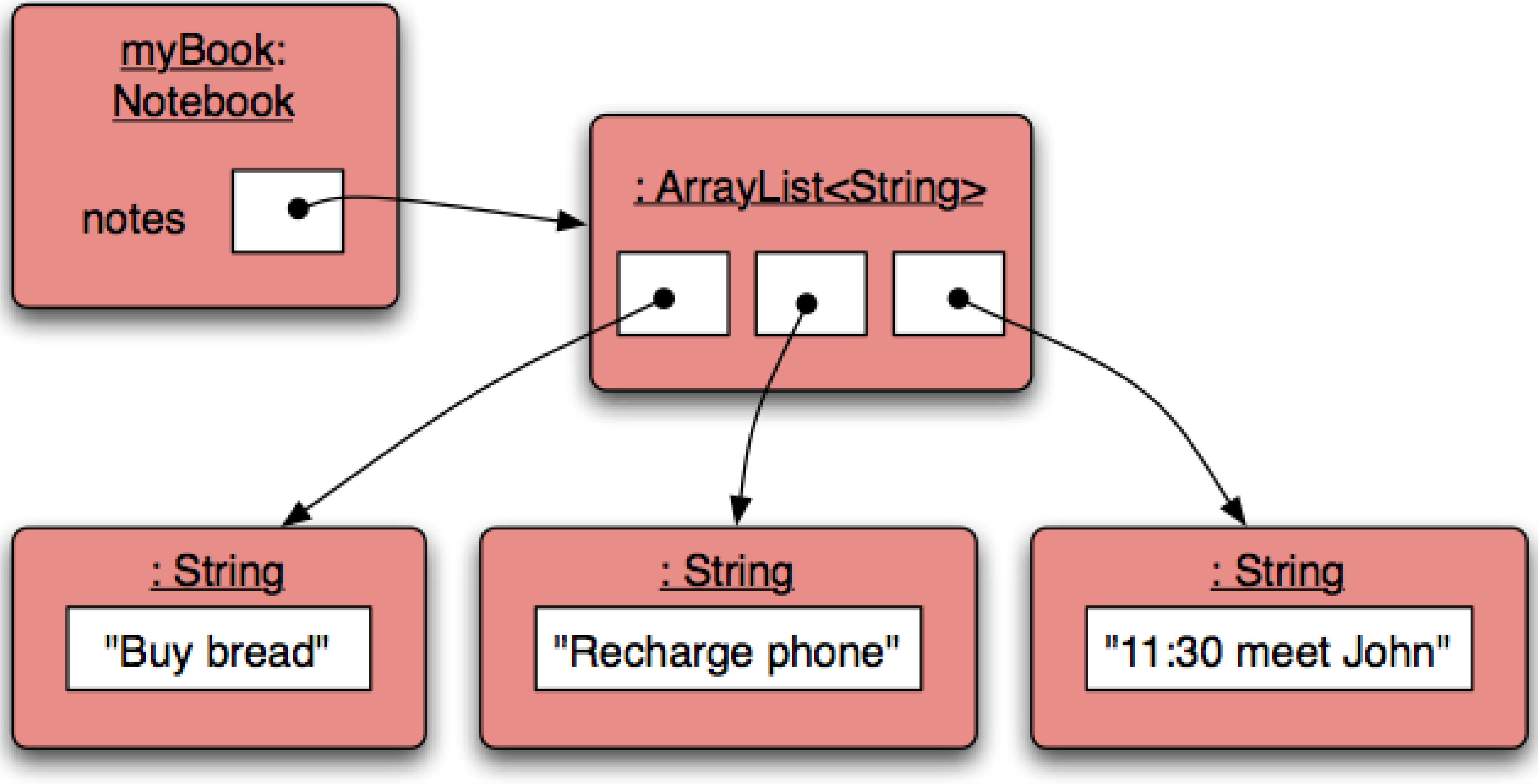
- We specify:
  - the type of collection: `ArrayList`
  - the type of objects it will contain: `<String>`

- We say, “ArrayList of String”.

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();
    }
}
```

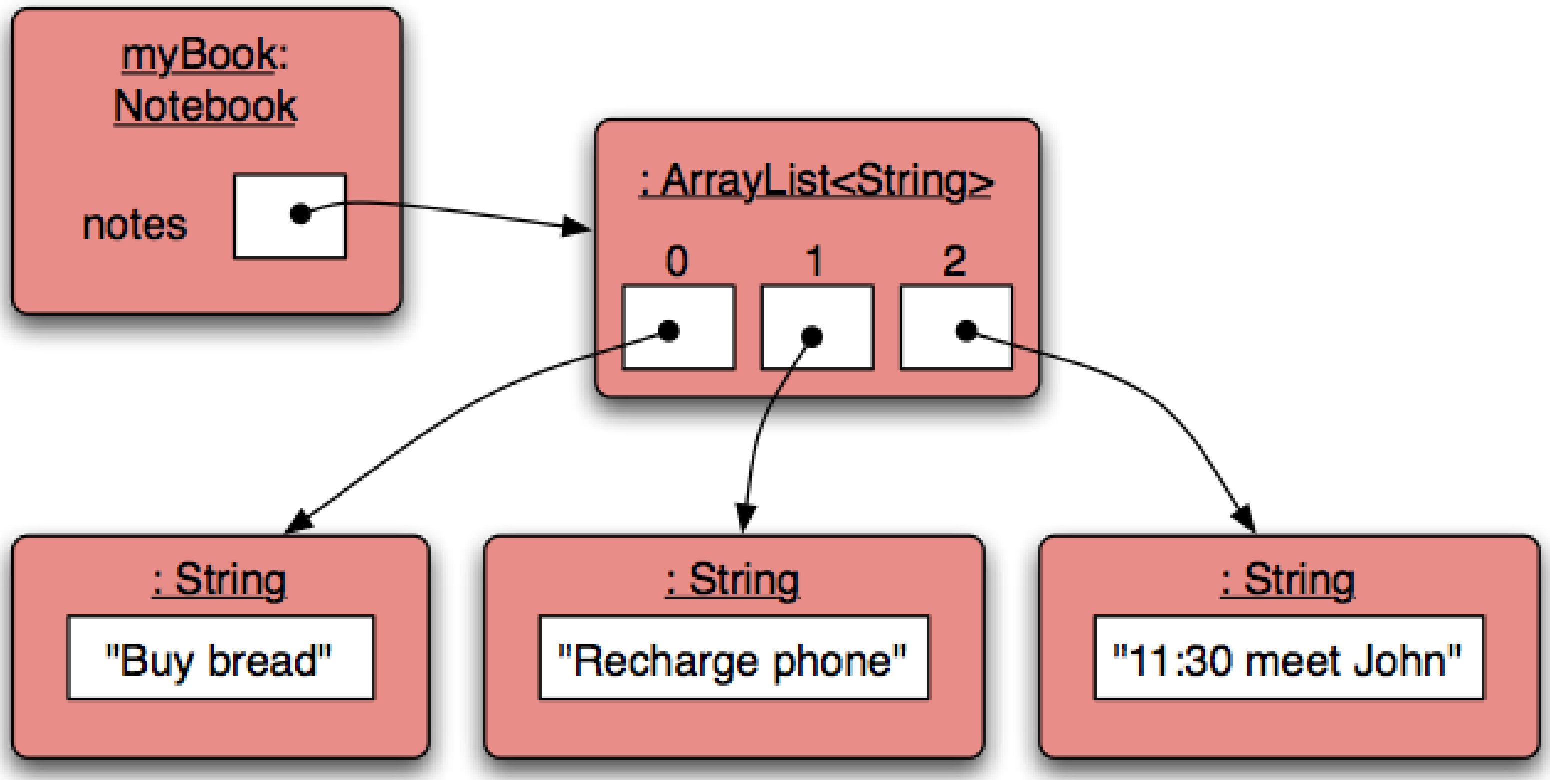
Adding a new note

Returning the number of notes

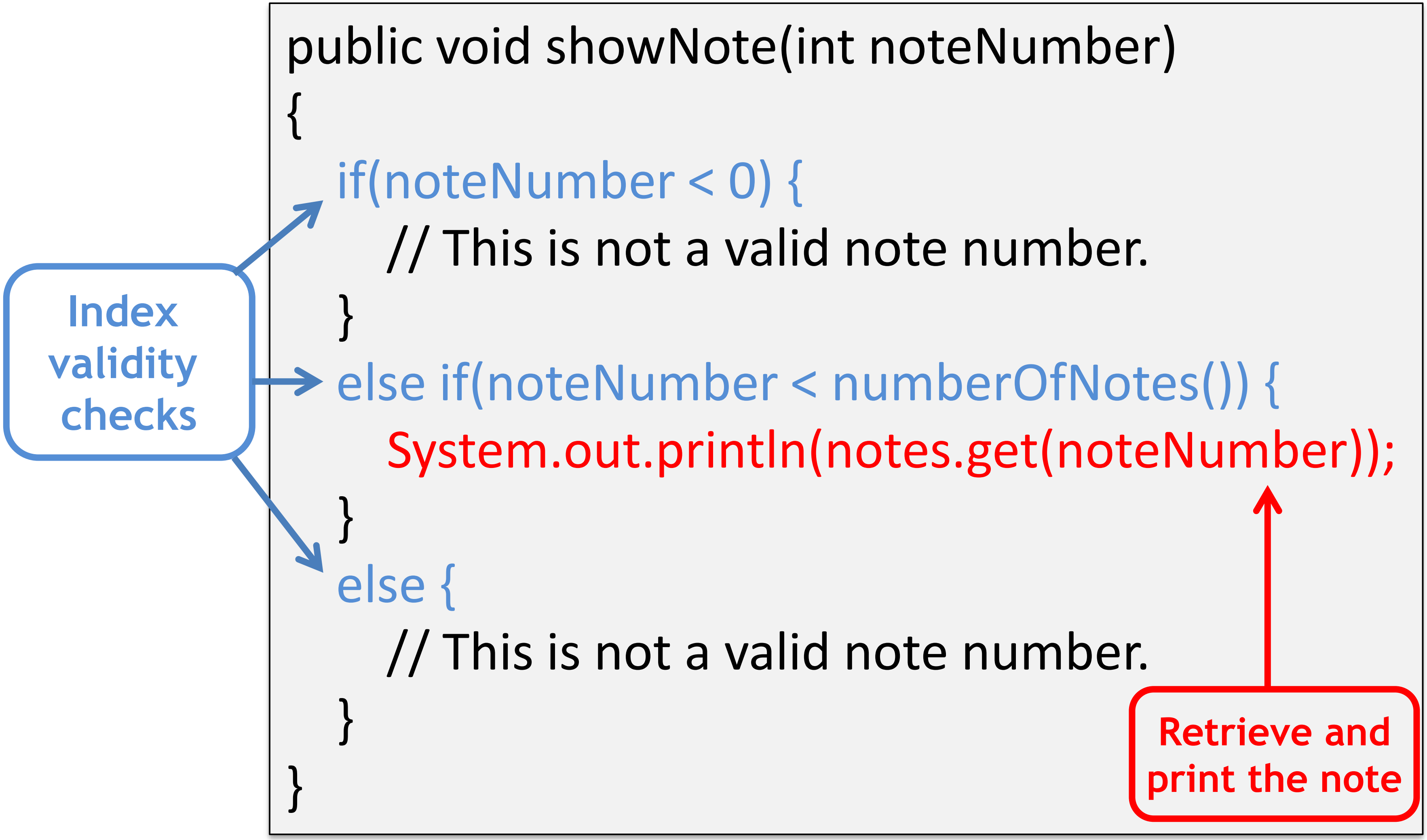
Topic list

- Grouping Objects
  - Developing a basic personal notebook project using Collections e.g. ArrayList
- Indexing within Collections
  - Retrieval and removal of objects
- Generic classes e.g. ArrayList
- Iteration
  - Using the for loop
  - Using the while loop
  - Using the for each loop
  - Using the Iterator
- Coding a Shop Project that stores an ArrayList of Products.

ArrayList: Index numbering



Retrieving an object



```
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

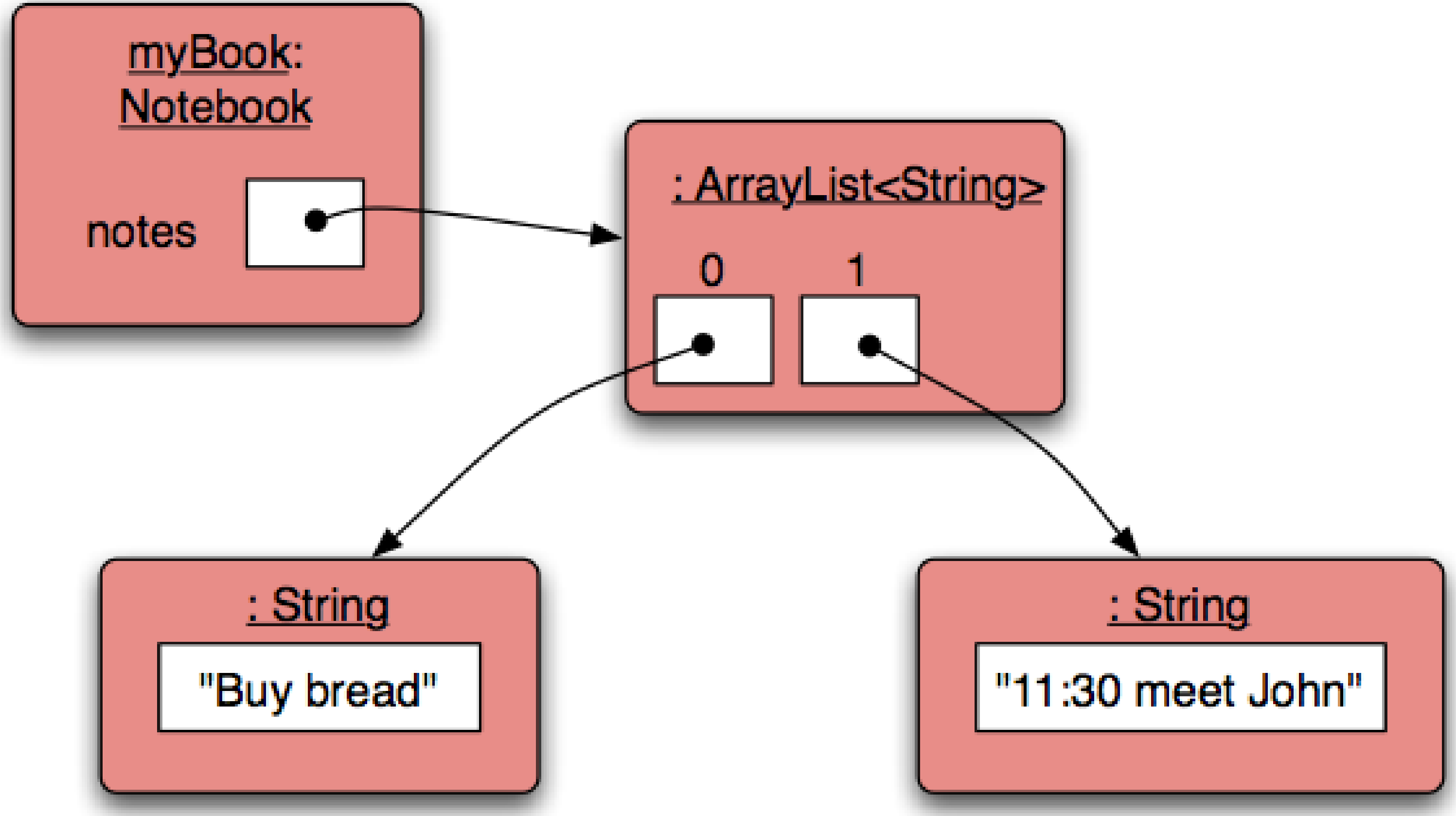
Removing an object

```
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

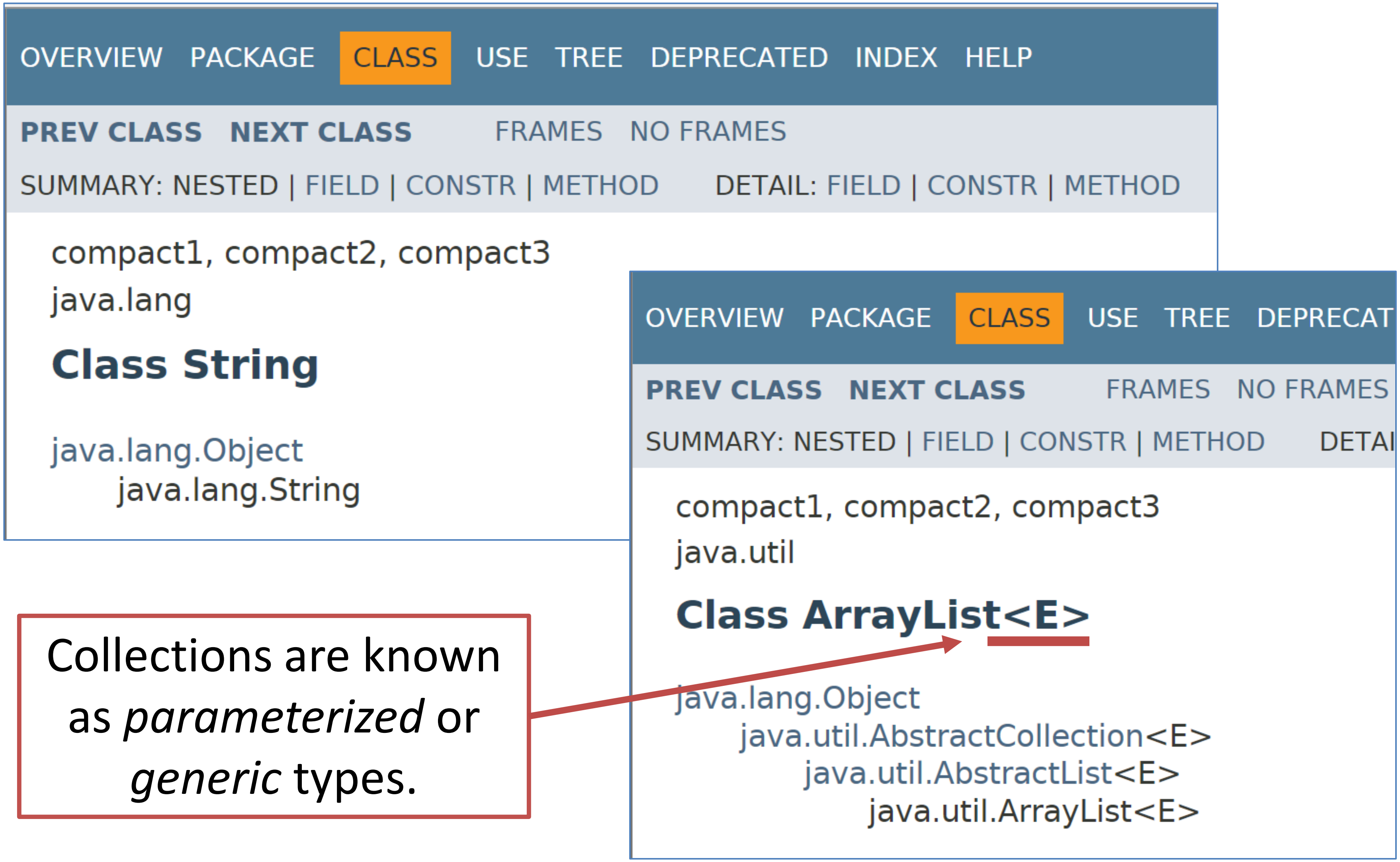
Removal may affect numbering



Topic list

- Grouping Objects
  - Developing a basic personal notebook project using Collections e.g. ArrayList
- Indexing within Collections
  - Retrieval and removal of objects
- Generic classes e.g. ArrayList
- Iteration
  - Using the for loop
  - Using the while loop
  - Using the for each loop
  - Using the Iterator
- Coding a Shop Project that stores an ArrayList of Products.

Generic Classes



OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3

java.lang

Class String

java.lang.Object
    java.lang.String

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

compact1, compact2, compact3

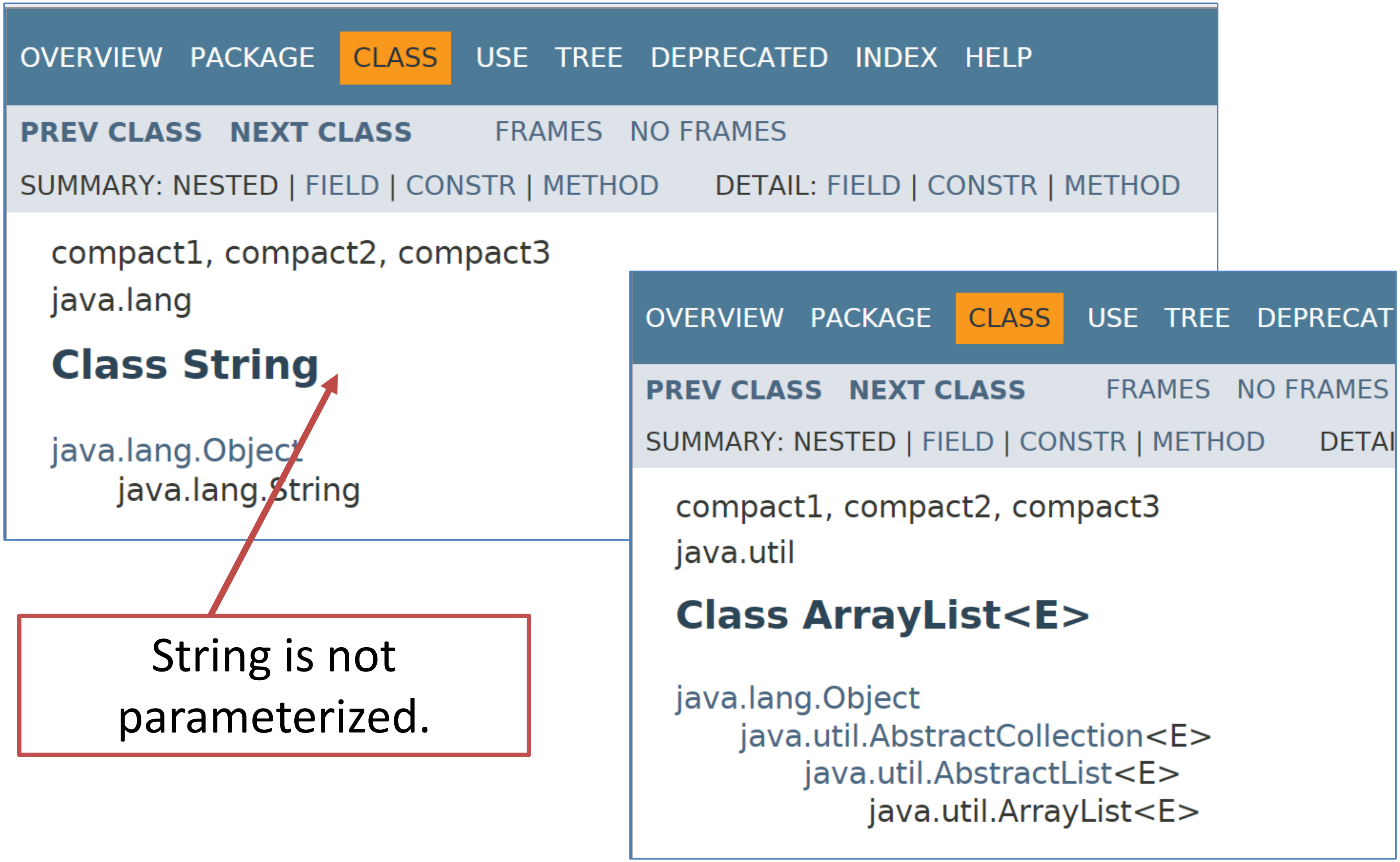
java.util

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.

Generic Classes


OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3
java.lang

Class String

java.lang.Object
    java.lang.String

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

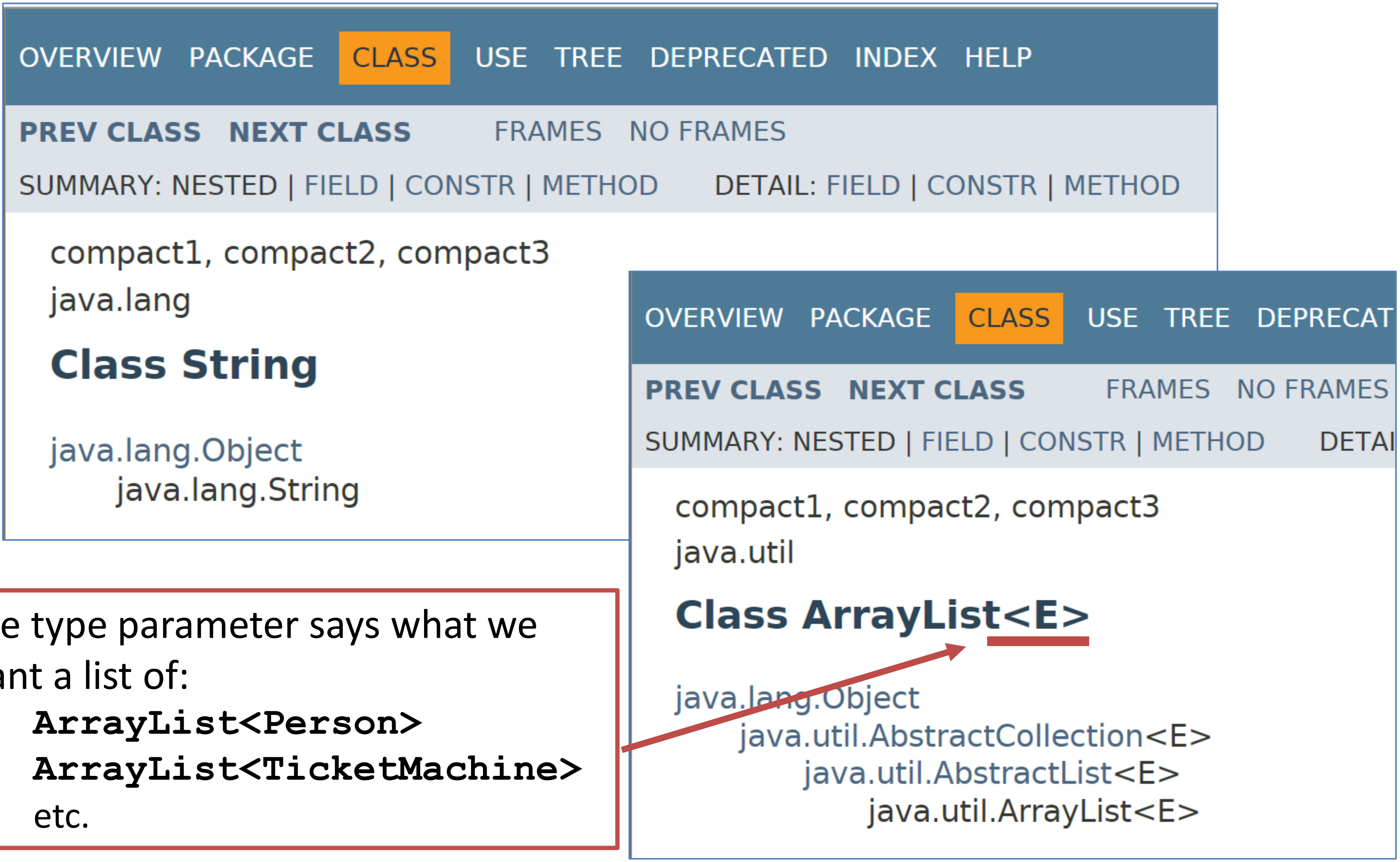
compact1, compact2, compact3
java.util

Class ArrayList<E>

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

String is not parameterized.

Generic Classes



compact1, compact2, compact3

java.lang

Class String

java.lang.Object
    java.lang.String

compact1, compact2, compact3

java.util

Class ArrayList<E>

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

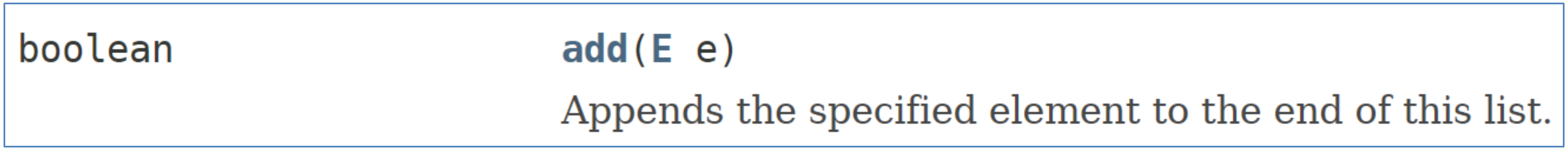
The type parameter says what we want a list of:

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

etc.

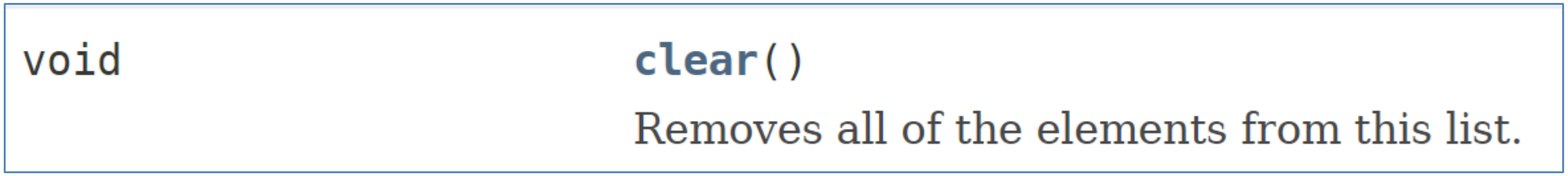
Generic classes

- `ArrayList` implements list functionality:



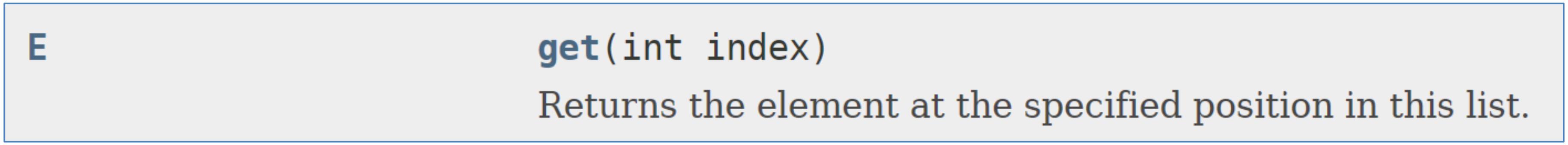
boolean add(E e)

Appends the specified element to the end of this list.



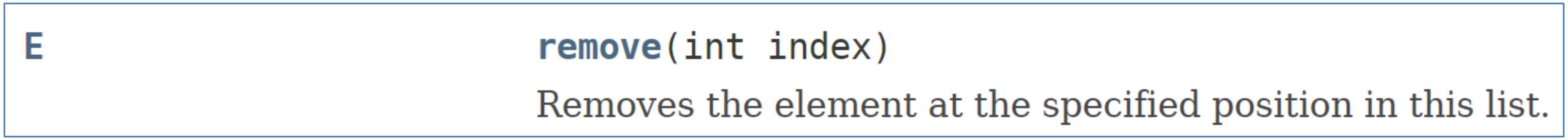
void clear()

Removes all of the elements from this list.



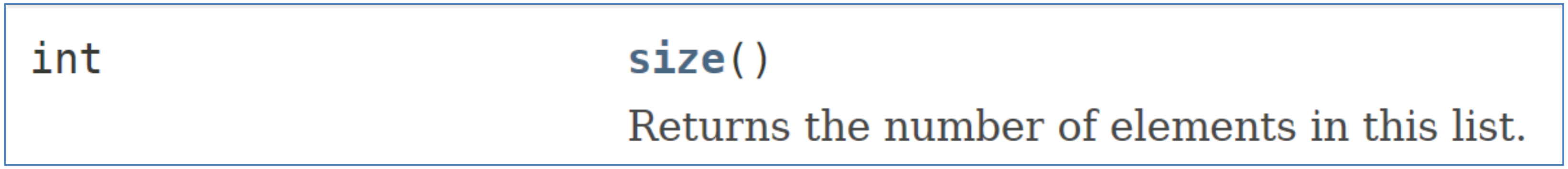
E get(int index)

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

- Grouping Objects
  - Developing a basic personal notebook project using Collections e.g. ArrayList
- Indexing within Collections
  - Retrieval and removal of objects
- Generic classes e.g. ArrayList
- Iteration
  - Using the for loop
  - Using the while loop
  - Using the for each loop
  - Using the Iterator
- 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 provide us with a way 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 (more on this in later lectures)

- You can use **for** and **while** loops to iterate over your ArrayList collection, or you can use two other special constructs associated with Collections:
  - for each
  - Iterator

Topic list

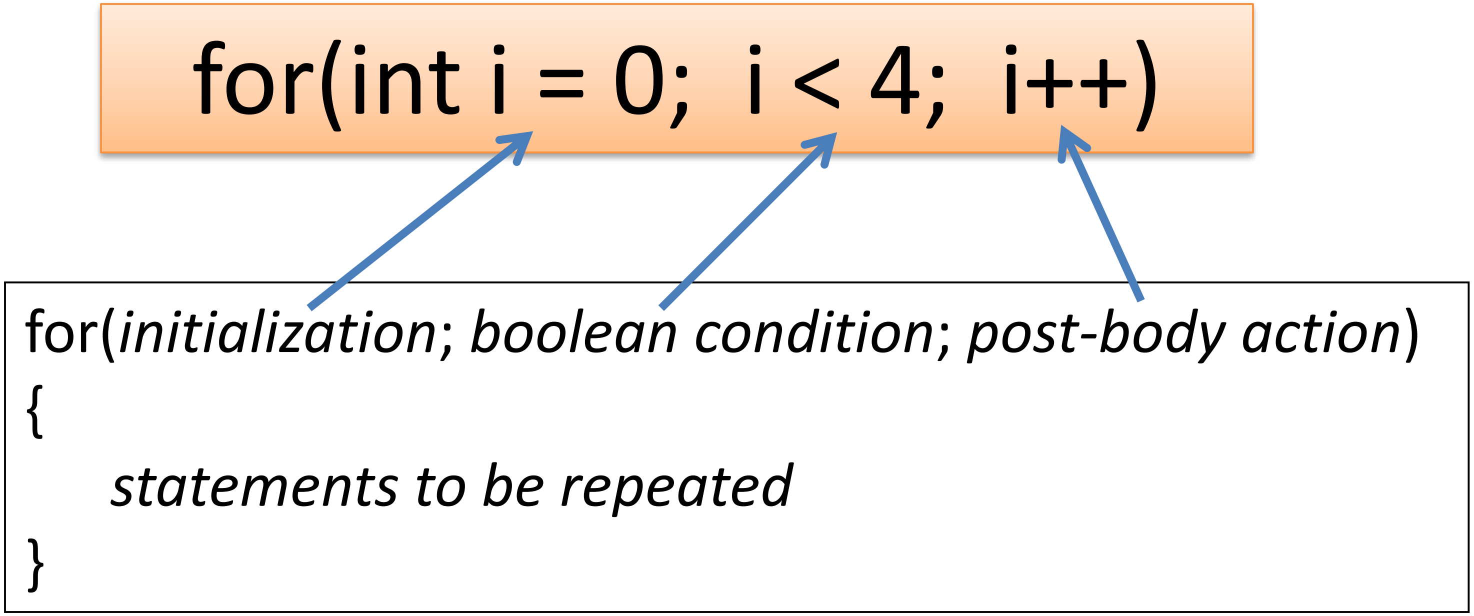
- Grouping Objects
  - Developing a basic personal notebook project using Collections e.g. ArrayList
- Indexing within Collections
  - Retrieval and removal of objects
- Generic classes e.g. ArrayList
- Iteration
  - Using the for loop
  - Using the while loop
  - Using the for each loop
  - Using the Iterator
- Coding a Shop Project that stores an ArrayList of Products.

for loop: pseudo-code

General form of a for loop

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

for loop: syntax



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

for(*initialization*; *boolean condition*; *post-body action*)
{

    statements to be repeated

}

for loop: syntax

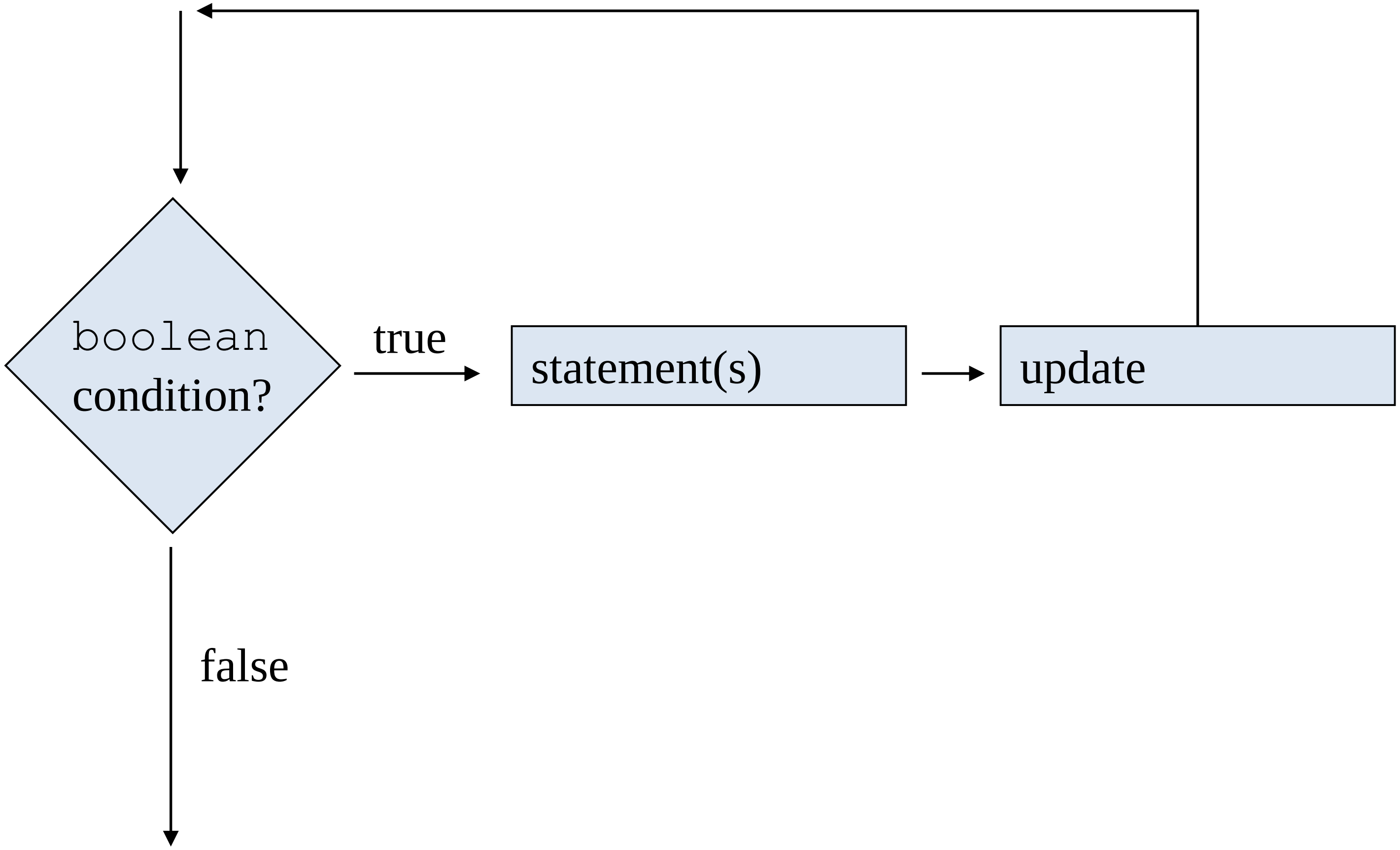
```
for(int i = 0;  i < 4;  i++)
```

i is the Loop Control Variable (LCV); three things must happen to it. It must be:

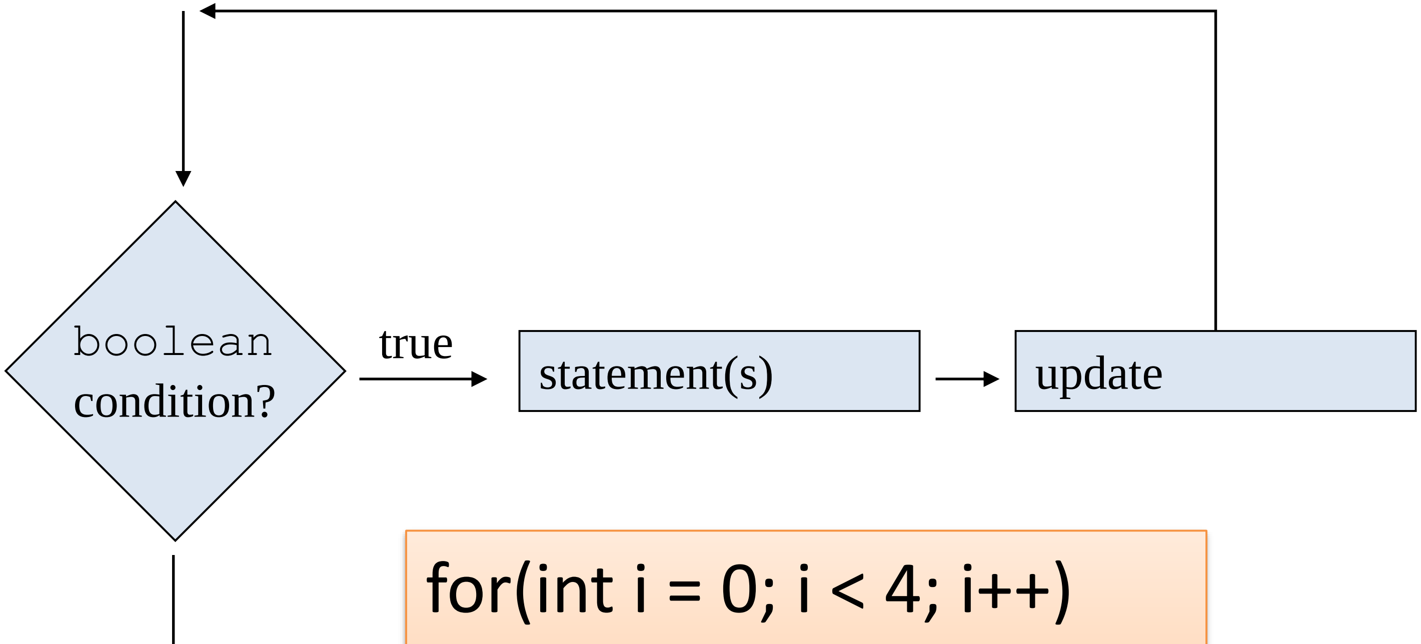
- Initialised
- Tested
- Updated

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

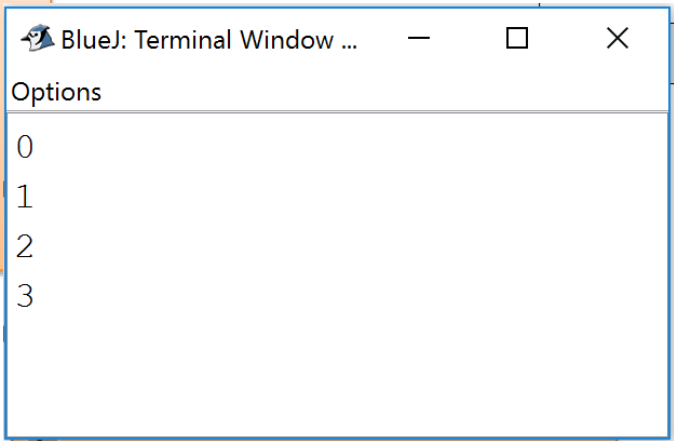
for loop: flowchart



for loop: flowchart



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



```
0
1
2
3
```

for loop: all parts are optional

```
for (  ;  ;  )
{
    // statements  here
}
```

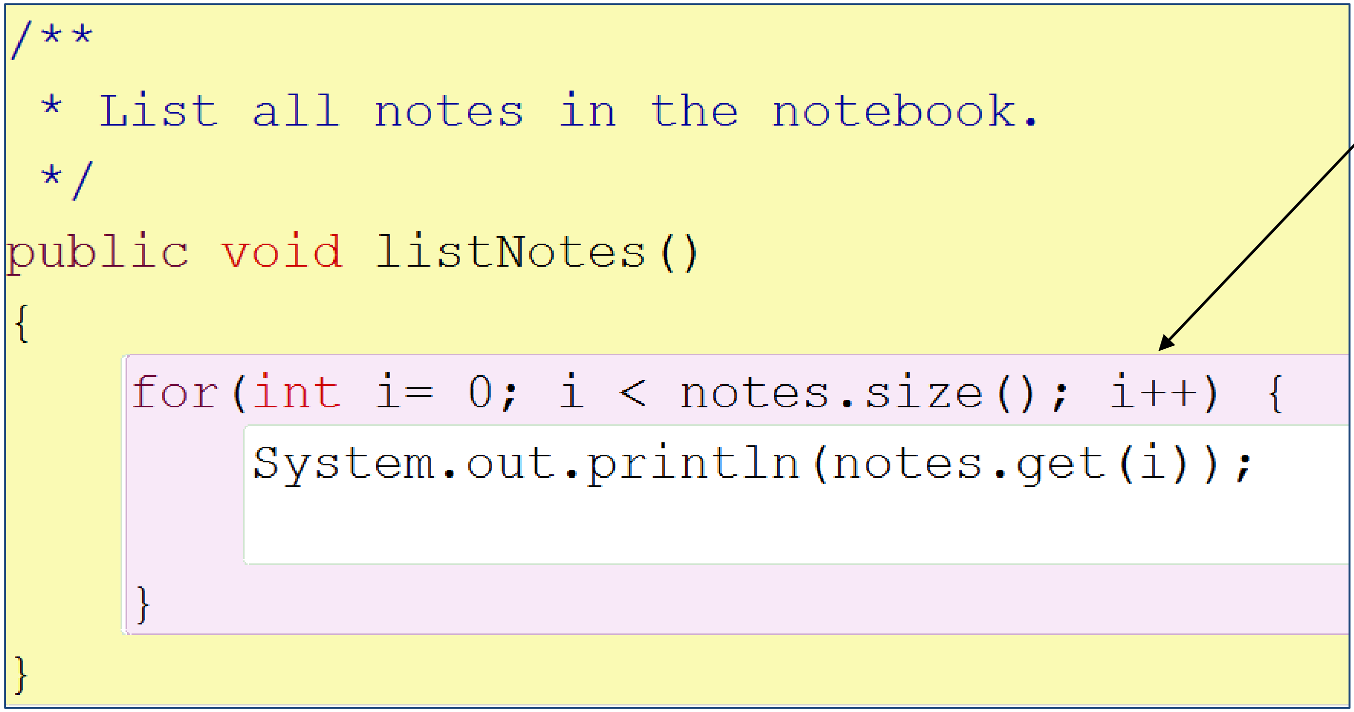
This is an infinite loop…

For loops can be nested

```
for (int i=0; i < 4; i++)
    for (int j=0; j < 4; j++)
      println("The value of i is: " + i + " and j is: " + j);
```

```
The value of i is: 0 and j is: 0
The value of i is: 0 and j is: 1
The value of i is: 0 and j is: 2
The value of i is: 0 and j is: 3
The value of i is: 1 and j is: 0
The value of i is: 1 and j is: 1
The value of i is: 1 and j is: 2
The value of i is: 1 and j is: 3
The value of i is: 2 and j is: 0
The value of i is: 2 and j is: 1
The value of i is: 2 and j is: 2
The value of i is: 2 and j is: 3
The value of i is: 3 and j is: 0
The value of i is: 3 and j is: 1
The value of i is: 3 and j is: 2
The value of i is: 3 and j is: 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));

    }
}
```

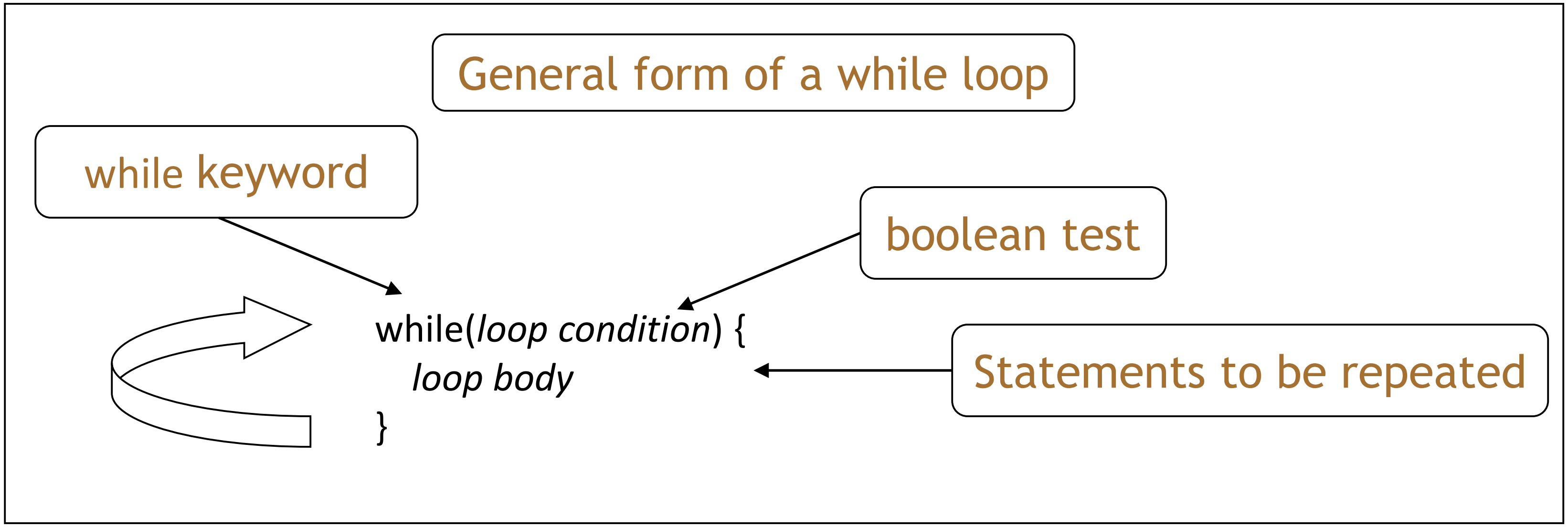
Increment *index* by 1

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

Topic list

- Grouping Objects
  - Developing a basic personal notebook project using Collections e.g. ArrayList
- Indexing within Collections
  - Retrieval and removal of objects
- Generic classes e.g. ArrayList
- Iteration
  - Using the for loop
  - Using the while loop
  - Using the for each loop
  - Using the Iterator
- Coding a Shop Project that stores an ArrayList of Products.

while loop: pseudo code



General form of a while loop

while keyword

boolean test

Statements to be repeated

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

Pseudo-code expression of the actions of a while loop

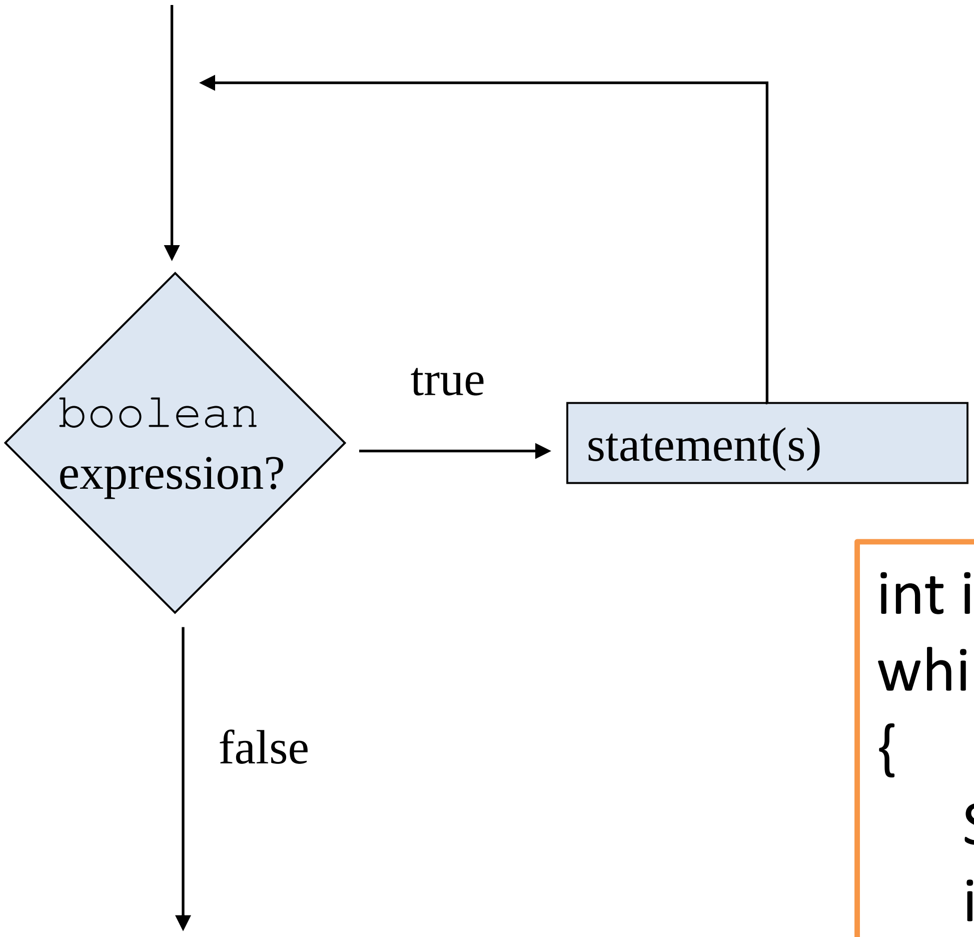
while we wish to continue, do the things in the loop body

while loop: construction

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

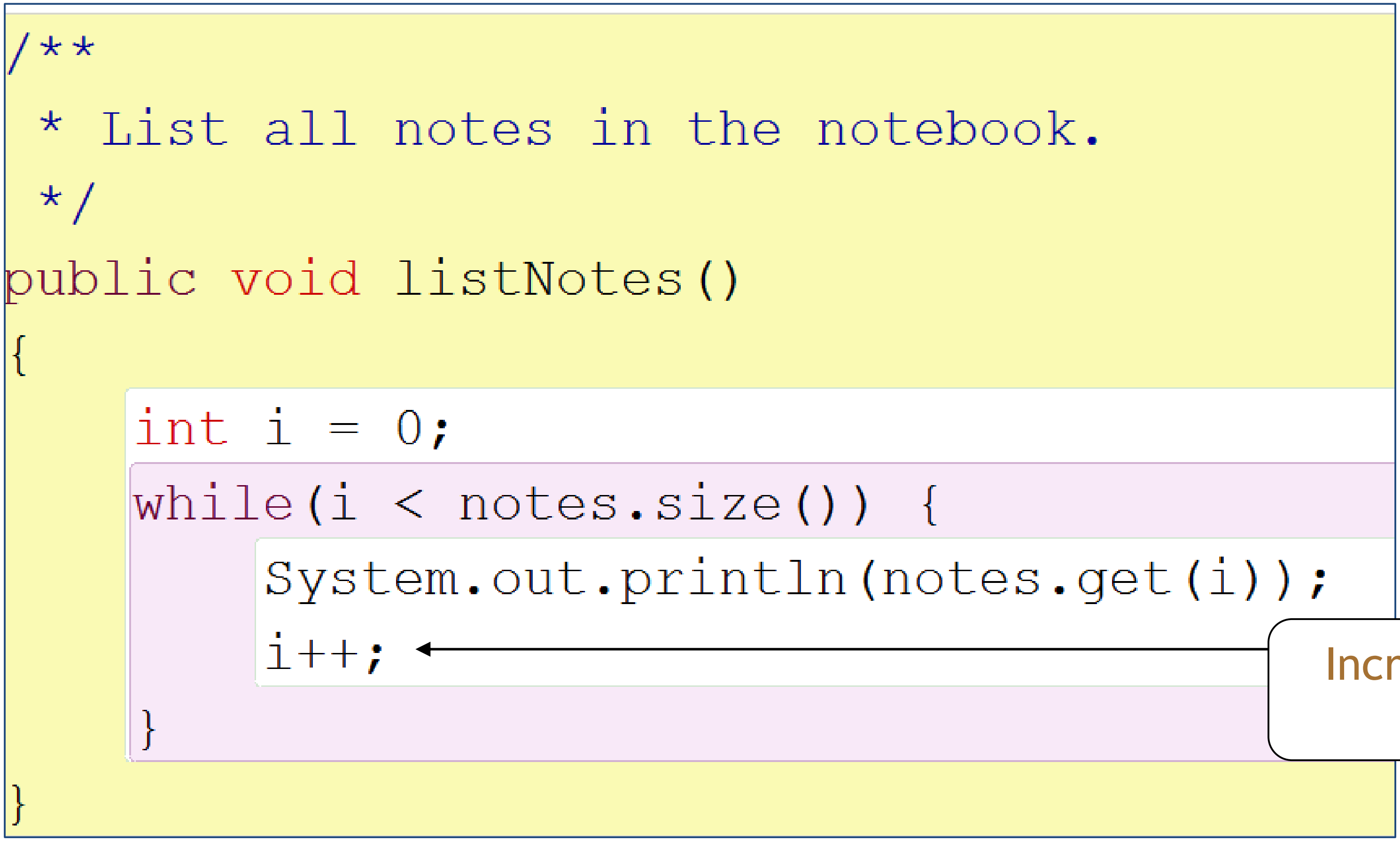
This structure should always be used

while loop: flowchart



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

while loop: iterating over a collection

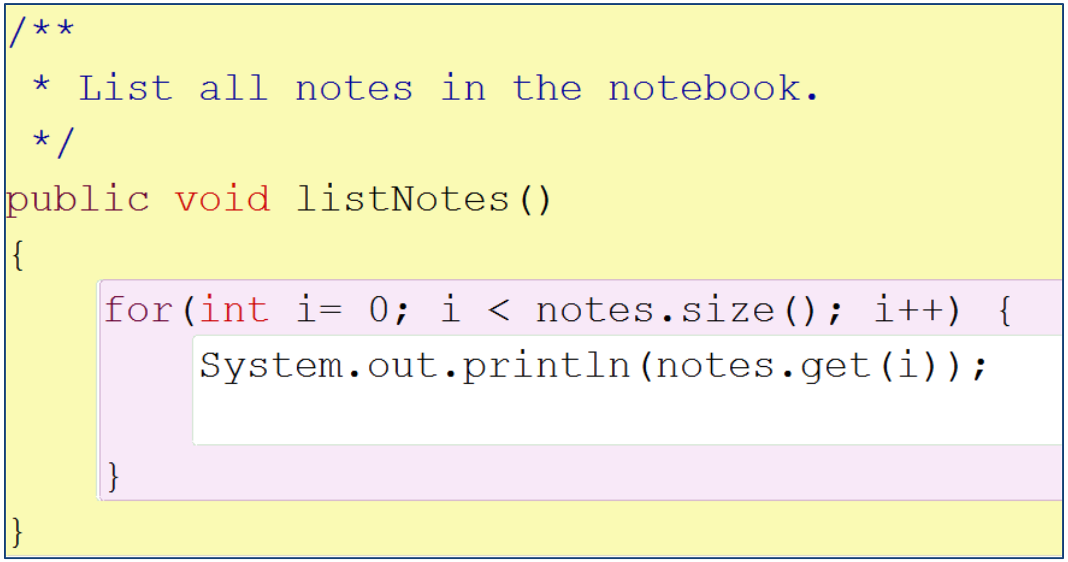


```
/**
 * List all notes in the notebook.
 */
public void listNotes()
{
    int i = 0;
    while(i < notes.size()) {
        System.out.println(notes.get(i));
        i++;
    }
}
```

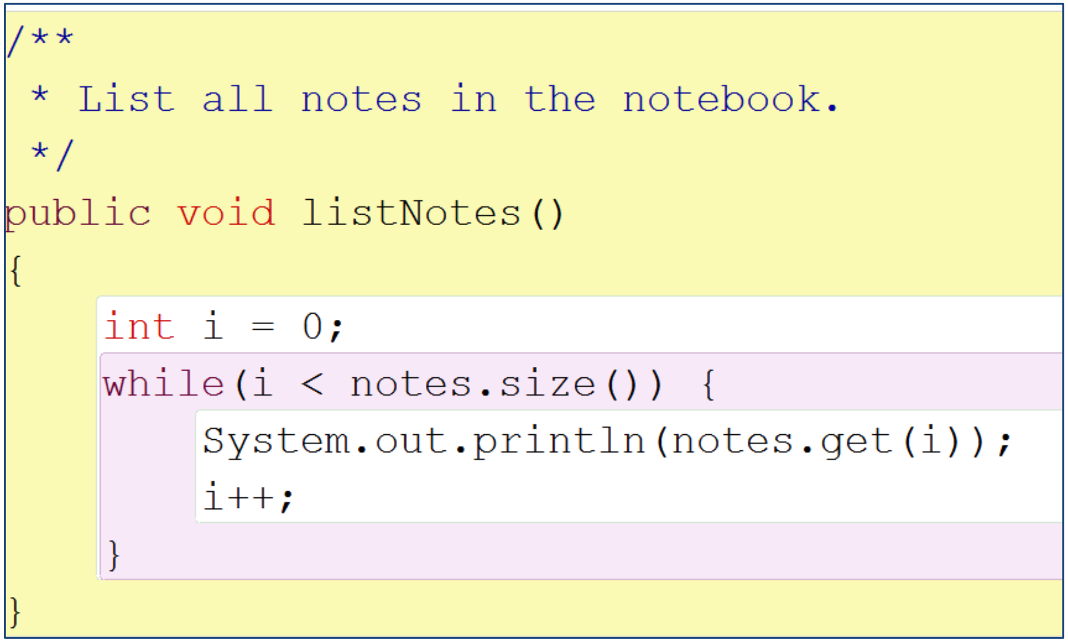
Increment *i* by 1

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.

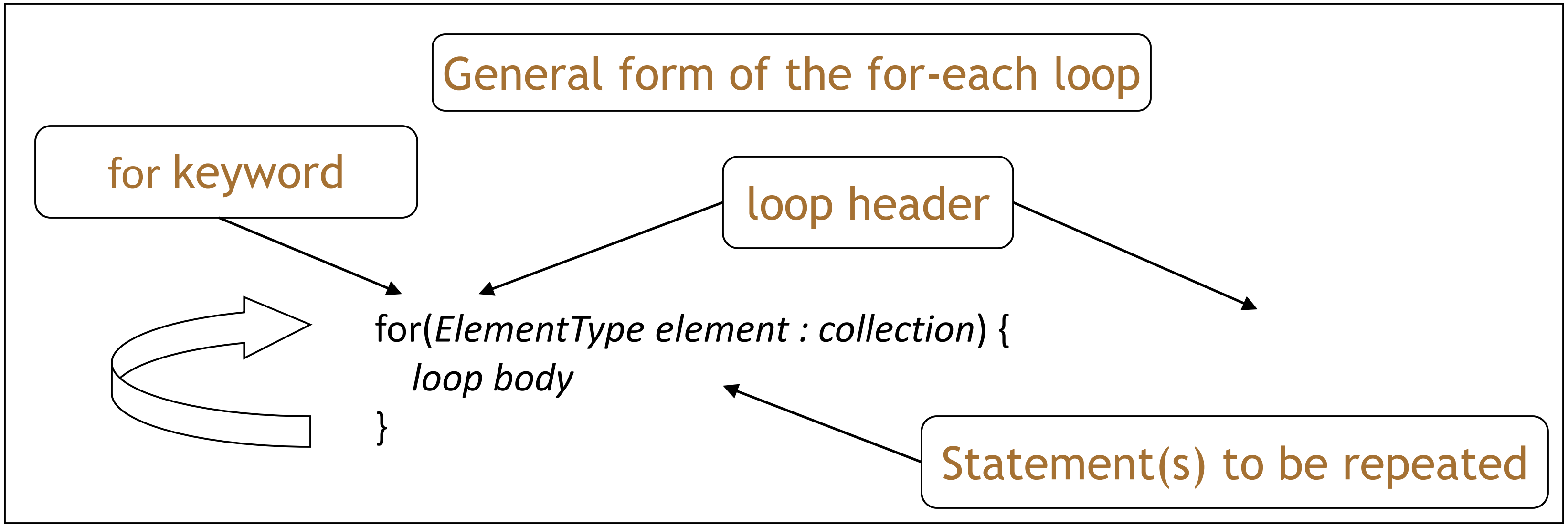
i < notes.size() is the test.

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

Topic list

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

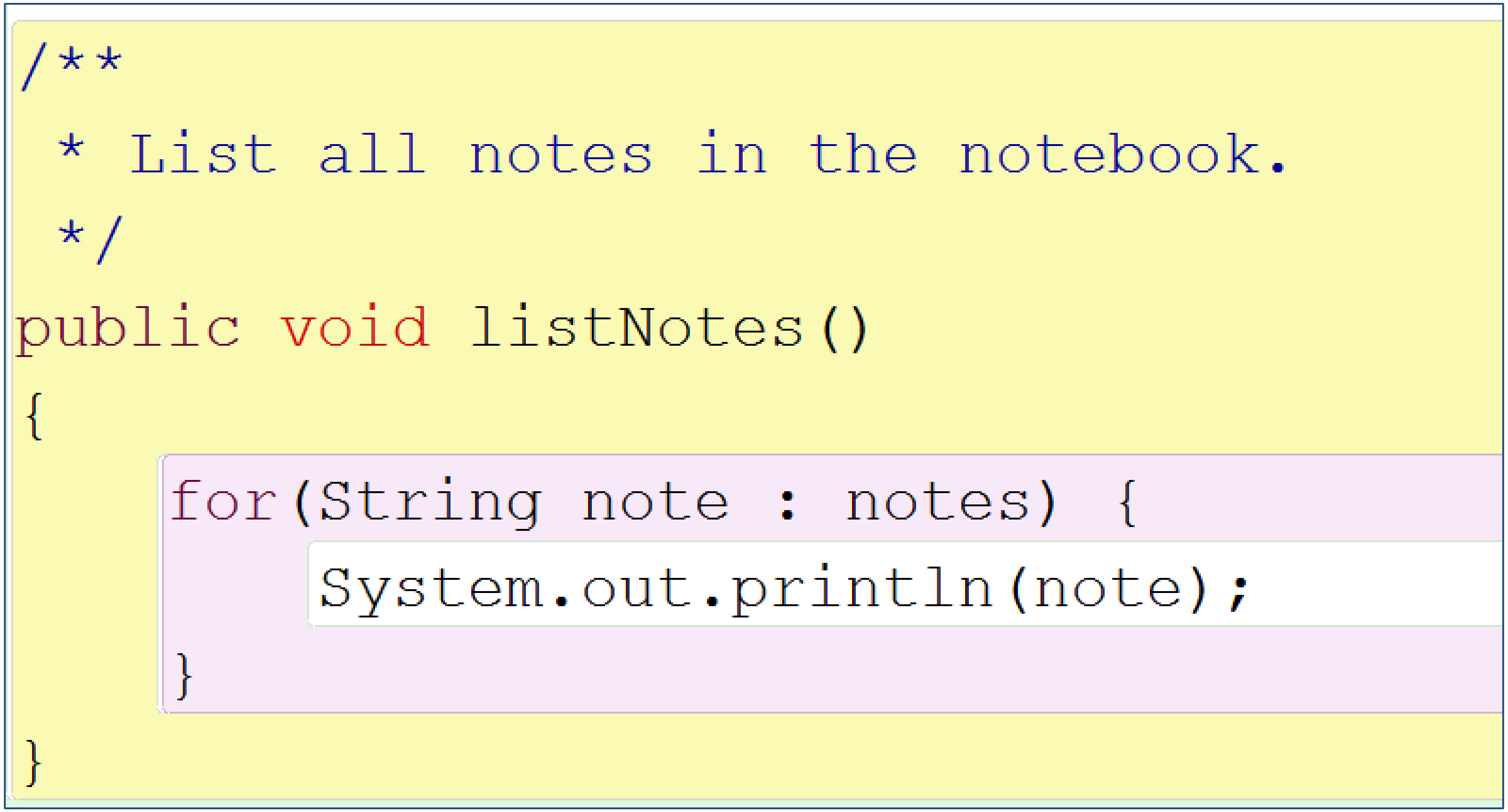
```
for(ElementType element : collection) {
    loop body
}
```

Statement(s) to be repeated

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 *notes*, 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*.

Topic list

- Grouping Objects
  - Developing a basic personal notebook project using Collections e.g. ArrayList
- Indexing within Collections
  - Retrieval and removal of objects
- Generic classes e.g. ArrayList
- Iteration
  - Using the for loop
  - Using the while loop
  - Using the for each loop
  - Using the Iterator
- Coding a Shop Project that stores an ArrayList of Products.

Iterator

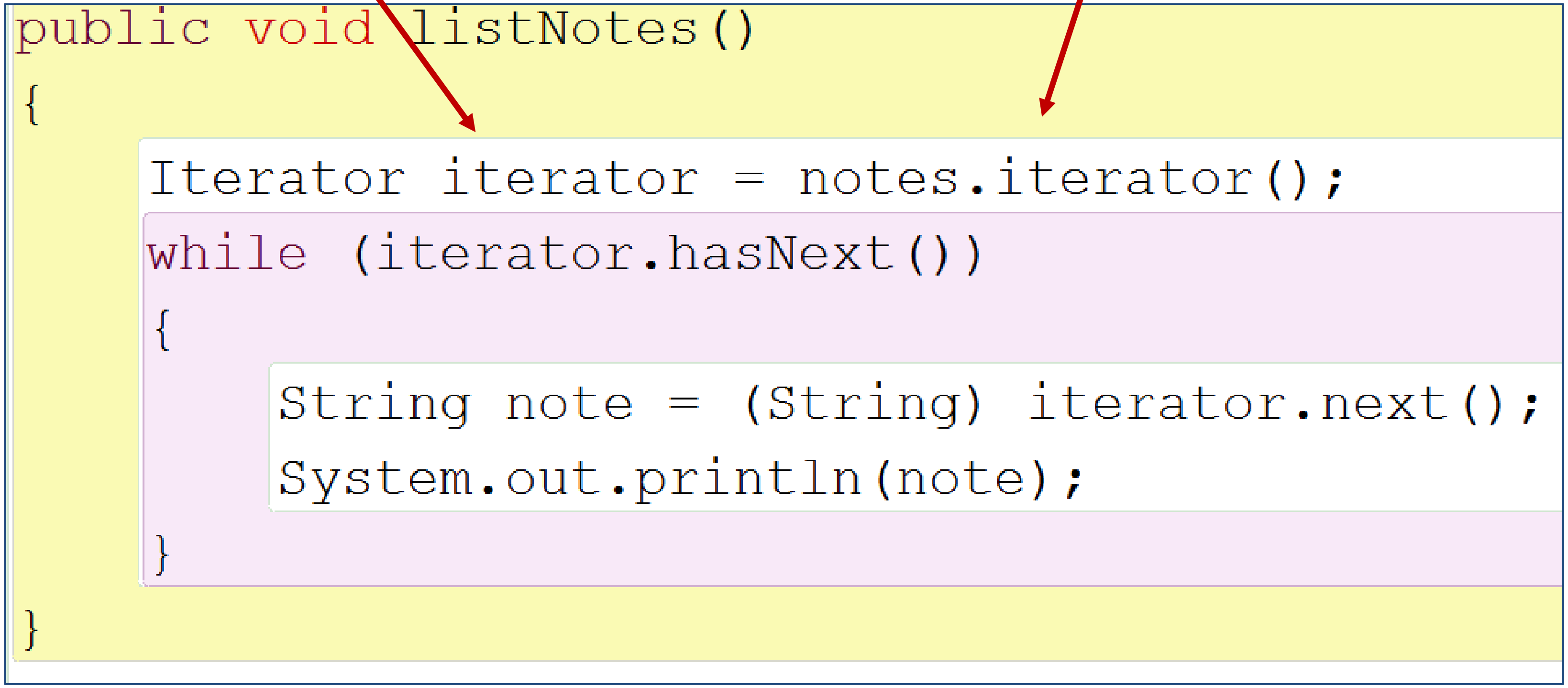
Defines a protocol for iterating through a collection.

```
public interface Iterator
{
  /**
   * Returns whether or not the underlying collection has next
   * element for iterating.
   */
  boolean hasNext();

  /**
   * Returns next element from the underlying collection.
   */
  Object next();

  /**
   * Removes from the underlying collection the last element returned
  by next.
   */
  void remove();
}
```

iterator

Attach an **iterator** object to our **notes** ArrayList.

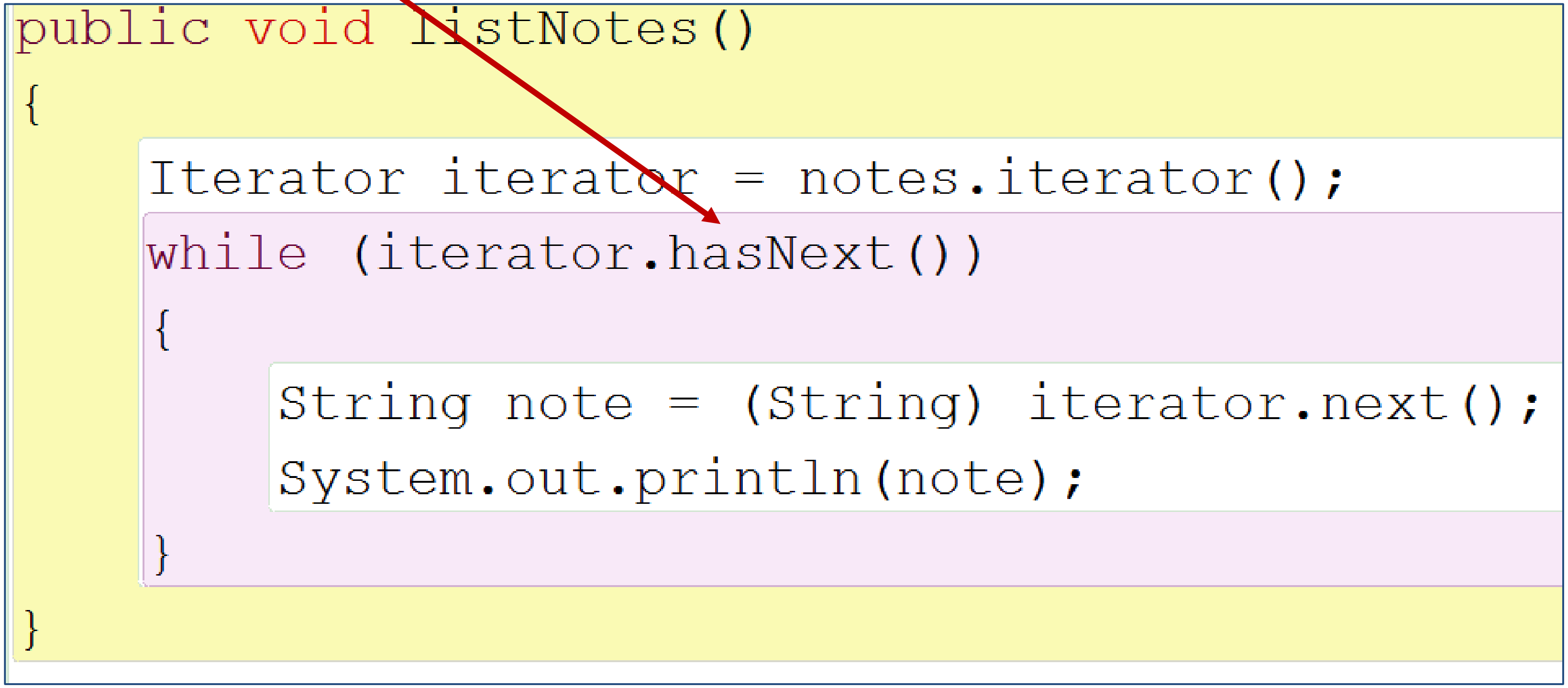
We will use this **iterator** object to traverse our ArrayList and retrieve each note in turn.



```
public void listNotes()
{
    Iterator iterator = notes.iterator();
    while (iterator.hasNext())
    {
        String note = (String) iterator.next();
        System.out.println(note);
    }
}
```

iterator

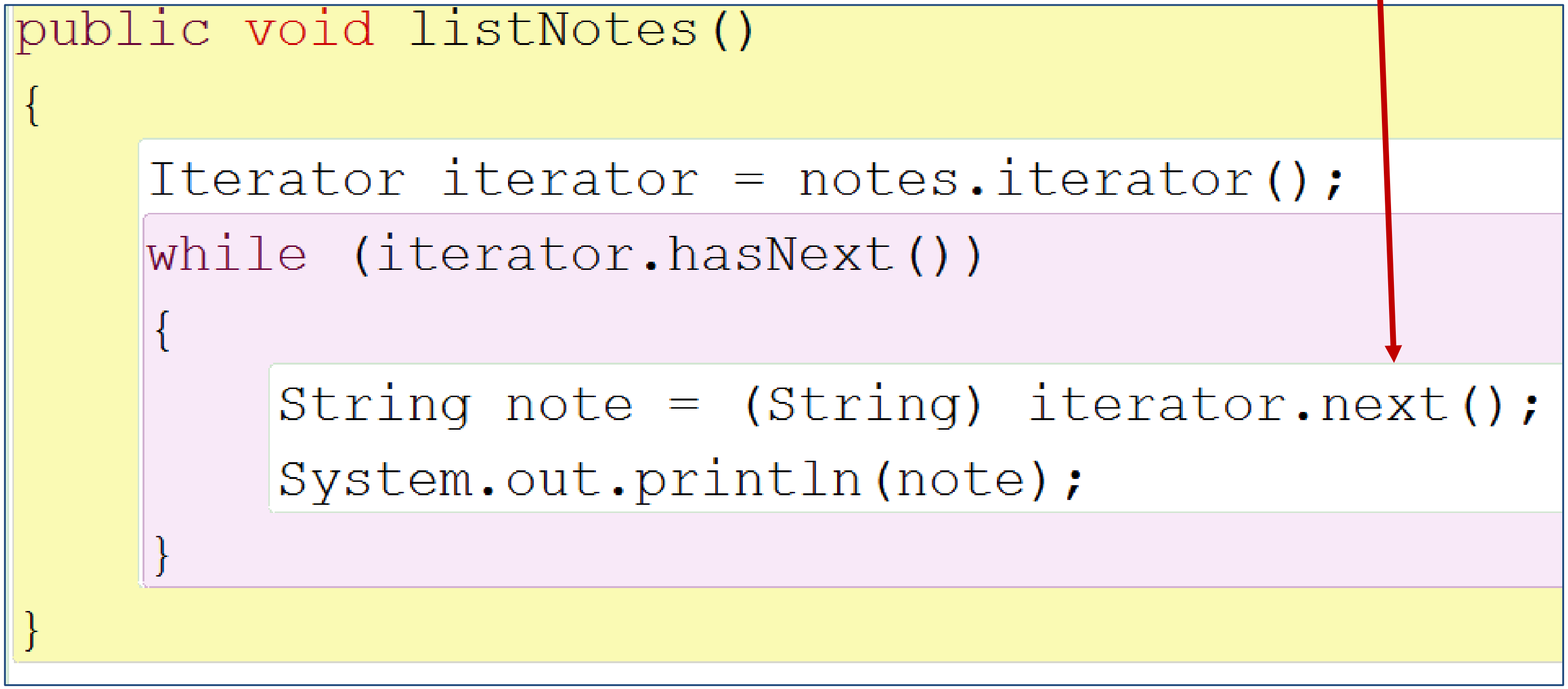
hasNext() returns **true** if the **iterator** for **notes** has more elements to view. This method only checks that there are more items; it doesn't retrieve any items.



```
public void listNotes()
{
    Iterator iterator = notes.iterator();
    while (iterator.hasNext())
    {
        String note = (String) iterator.next();
        System.out.println(note);
    }
}
```

iterator

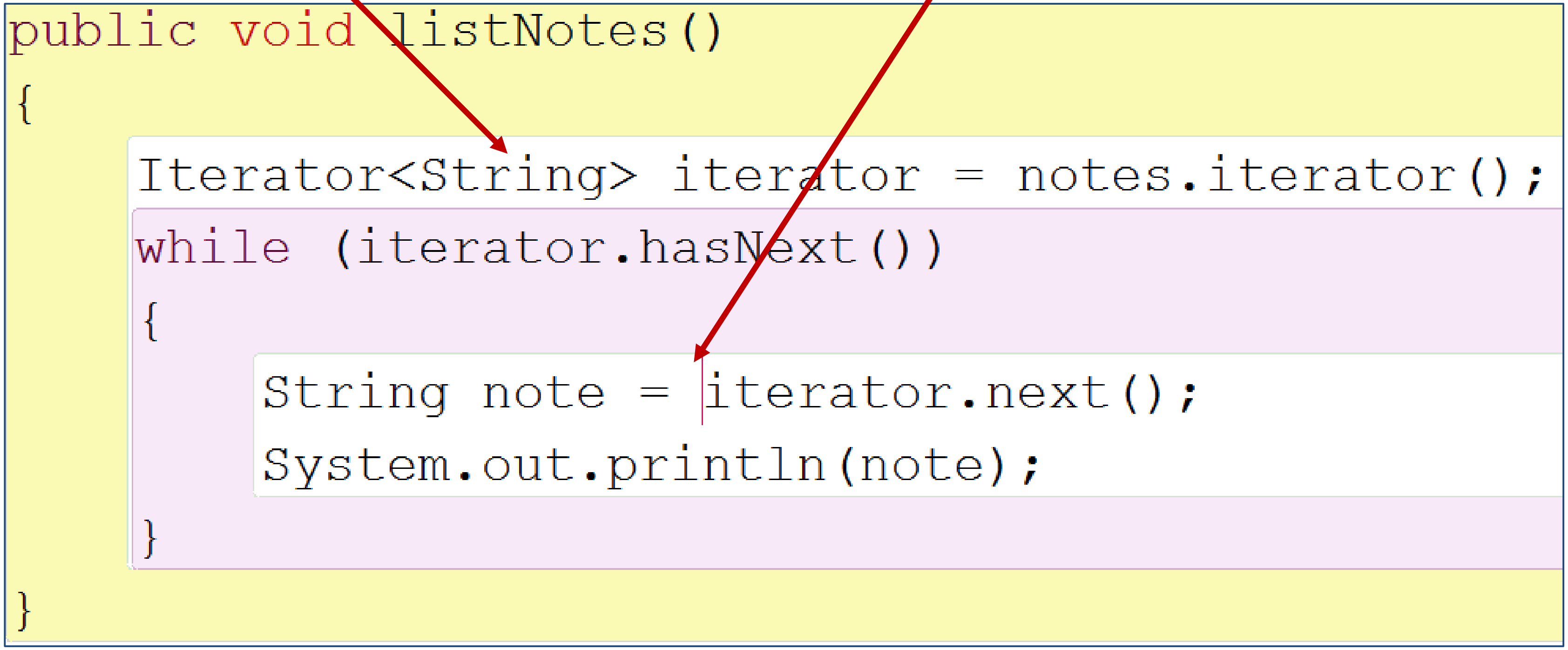
next() returns the next element from the **notes iterator.** However, we need to ***cast*** the returned element as String because we didn’t **type** our **Iterator**.



```
public void listNotes()
{
    Iterator iterator = notes.iterator();
    while (iterator.hasNext())
    {
        String note = (String) iterator.next();
        System.out.println(note);
    }
}
```

iterator

Let’s **type** our **Iterator** so that we don’t have to cast the retrieved elements…they will now be Strings.



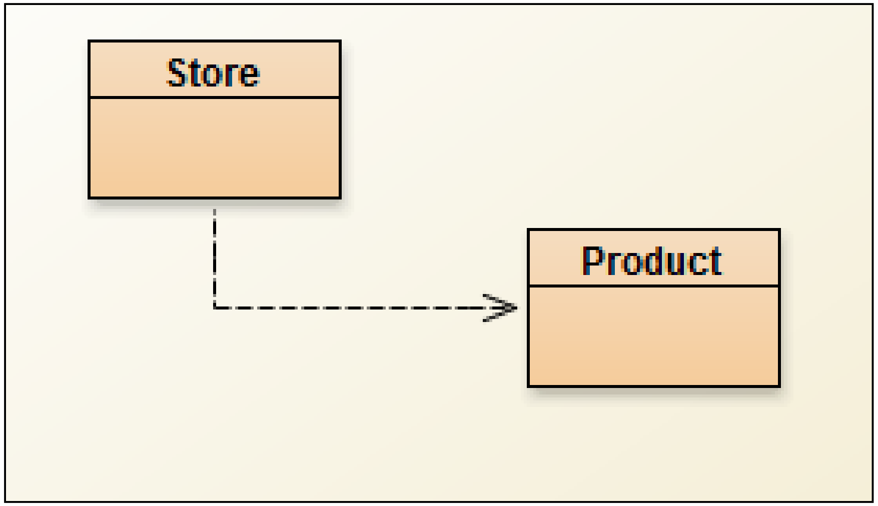
```
public void listNotes()
{
    Iterator<String> iterator = notes.iterator();
    while (iterator.hasNext())
    {
        String note = iterator.next();
        System.out.println(note);
    }
}
```

Topic list

- Grouping Objects
  - Developing a basic personal notebook project using Collections e.g. ArrayList
- Indexing within Collections
  - Retrieval and removal of objects
- Generic classes e.g. ArrayList
- Iteration
  - Using the for loop
  - Using the while loop
  - Using the for each loop
  - Using the Iterator
- Coding a Shop Project that stores an ArrayList of Products.

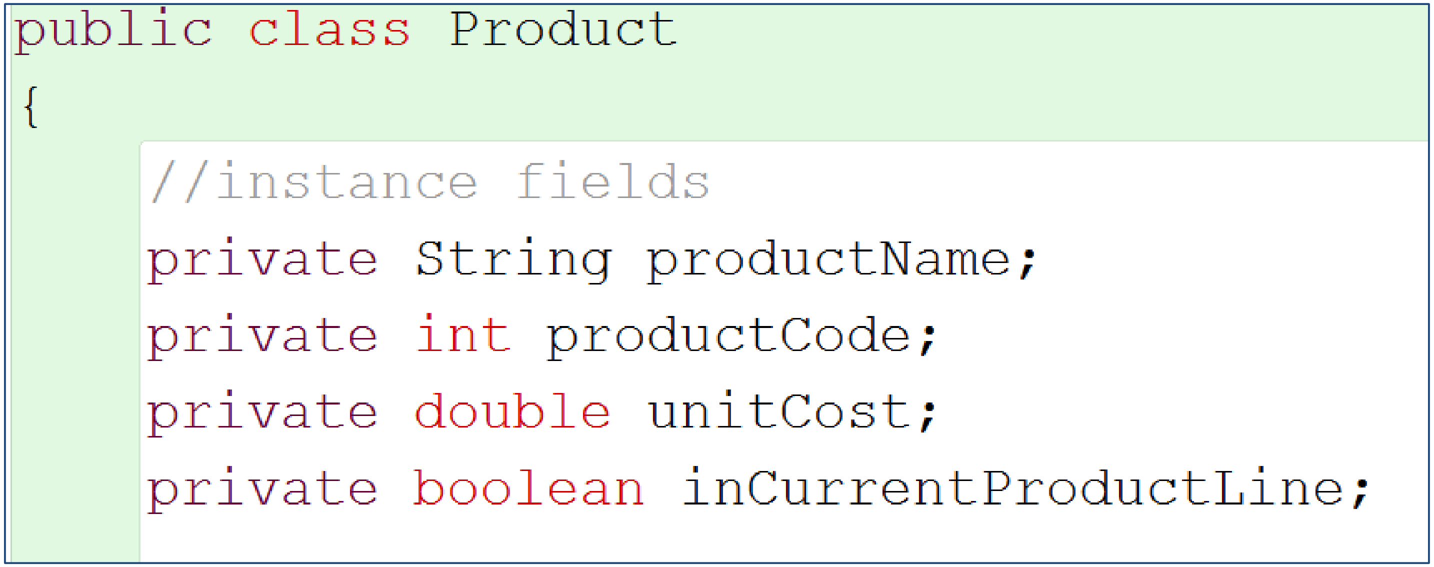
A basic example of a Shop

A Store has an ArrayList of Product.



Product class

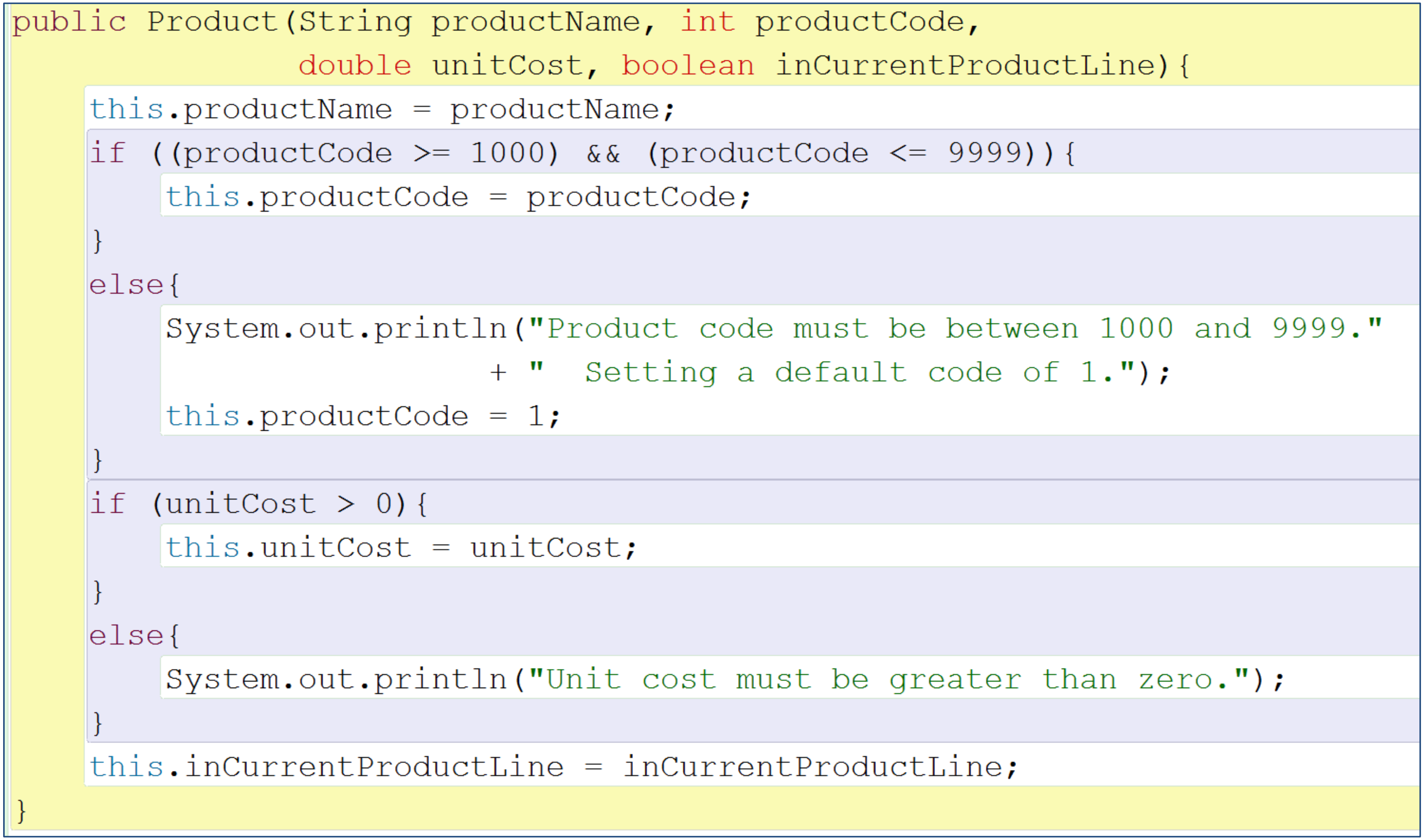
Our Product class contains four instance variables



```
public class Product
{
    //instance fields
    private String productName;
    private int productCode;
    private double unitCost;
    private boolean inCurrentProductLine;
```

Product class

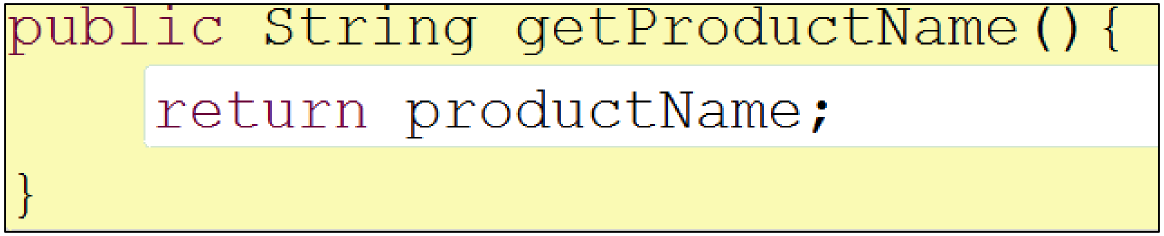
The constructor uses the data passed in the four parameters to update the instance fields.



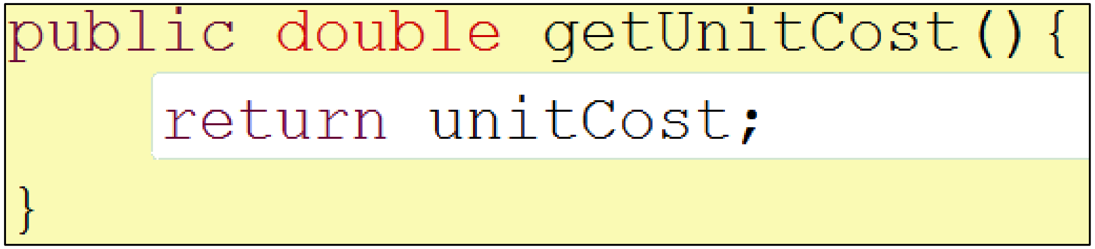
```
public Product(String productName, int productCode,
               double unitCost, boolean inCurrentProductLine){
    this.productName = productName;
    if ((productCode >= 1000) && (productCode <= 9999)){
        this.productCode = productCode;
    }
    else{
        System.out.println("Product code must be between 1000 and 9999."
                         + "  Setting a default code of 1.");
        this.productCode = 1;
    }
    if (unitCost > 0){
        this.unitCost = unitCost;
    }
    else{
        System.out.println("Unit cost must be greater than zero.");
    }
    this.inCurrentProductLine = inCurrentProductLine;
}
```

Product class

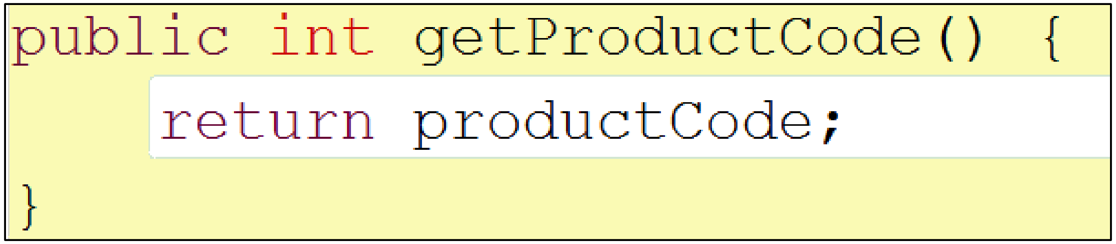
The class has getters for each instance field.



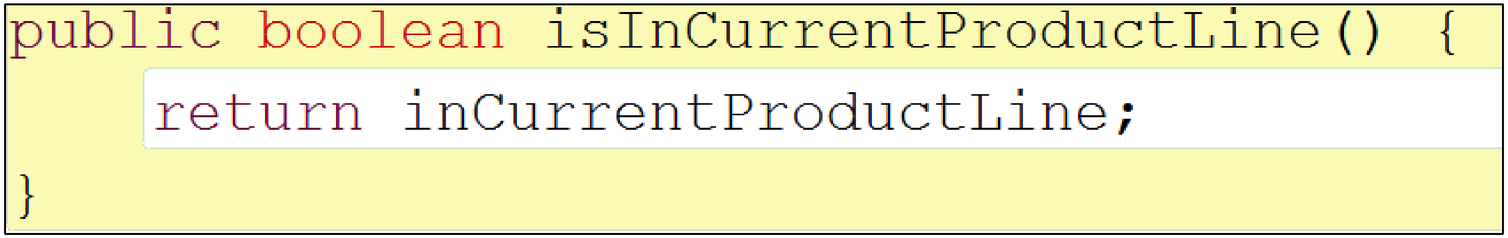
```
public String getProductName(){
    return productName;
}
```



```
public double getUnitCost(){
    return unitCost;
}
```



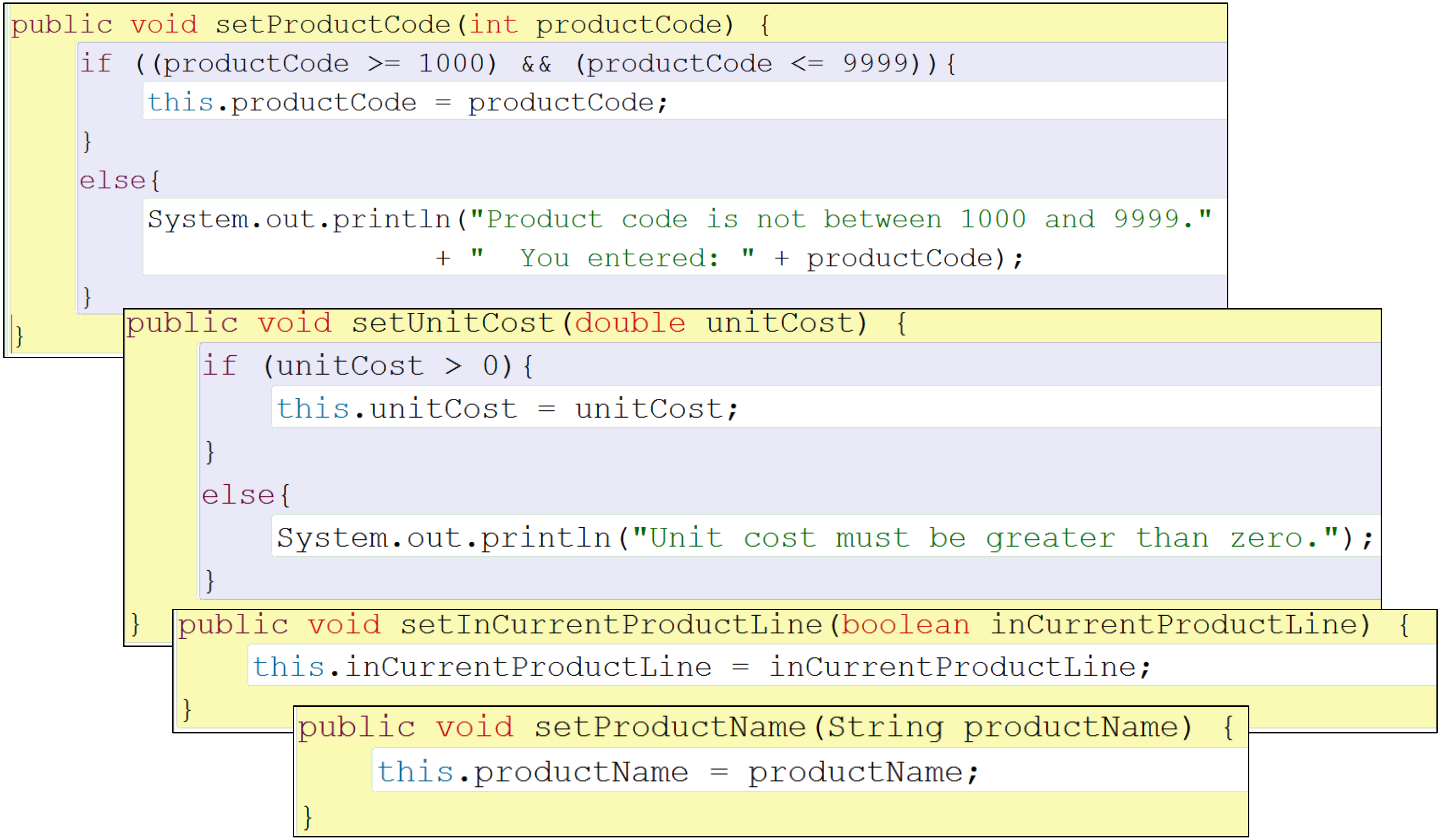
```
public int getProductCode()  {
    return productCode;
}
```



```
public boolean isInCurrentProductLine()  {
    return inCurrentProductLine;
}
```

Product class

The class has setters for each instance field.



```
public void setProductCode(int productCode) {
    if ((productCode >= 1000) && (productCode <= 9999)){
        this.productCode = productCode;
    }
    else{
        System.out.println("Product code is not between 1000 and 9999."
                  + "  You entered: " + productCode);
    }
}
```

```
public void setUnitCost(double unitCost) {
    if (unitCost > 0){
        this.unitCost = unitCost;
    }
    else{
        System.out.println("Unit cost must be greater than zero.");
    }
}
```

```
public void setInCurrentProductLine(boolean inCurrentProductLine) {
    this.inCurrentProductLine = inCurrentProductLine;
}
```

```
public void setProductName(String productName) {
    this.productName = productName;
}
```

Product class: toString()

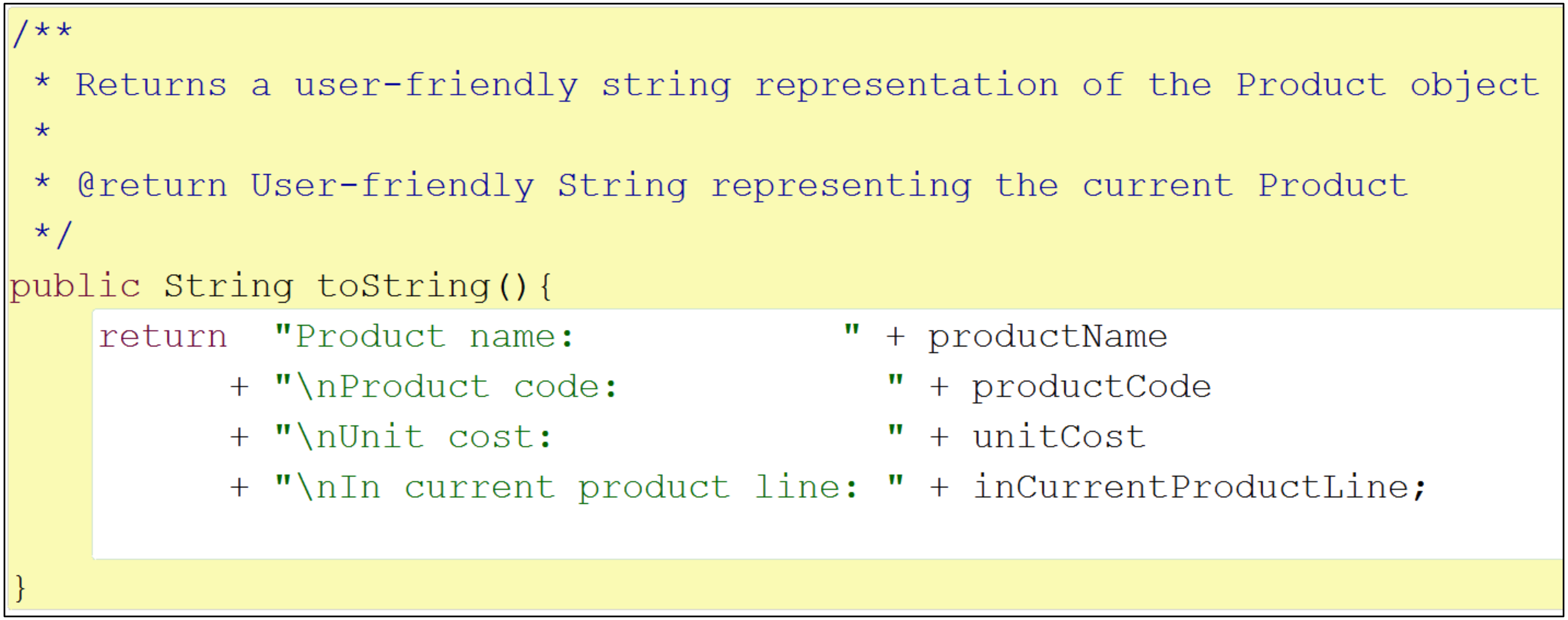
- Let’s now add a **toString()** method.

- Java has a special method with the signature:

  public String toString()

- You use this method to return a String that represents an object’s state, in a user-friendly manner.

Product class: toString()



```
/**
 * Returns a user-friendly string representation of the Product object
 *
 * @return User-friendly String representing the current Product
 */
public String toString(){
    return  "Product name:            " + productName
          + "\nProduct code:            " + productCode
          + "\nUnit cost:                " + unitCost
          + "\nIn current product line: " + inCurrentProductLine;

}
```

We will call this method from the Store class that we will construct over the next few slides.

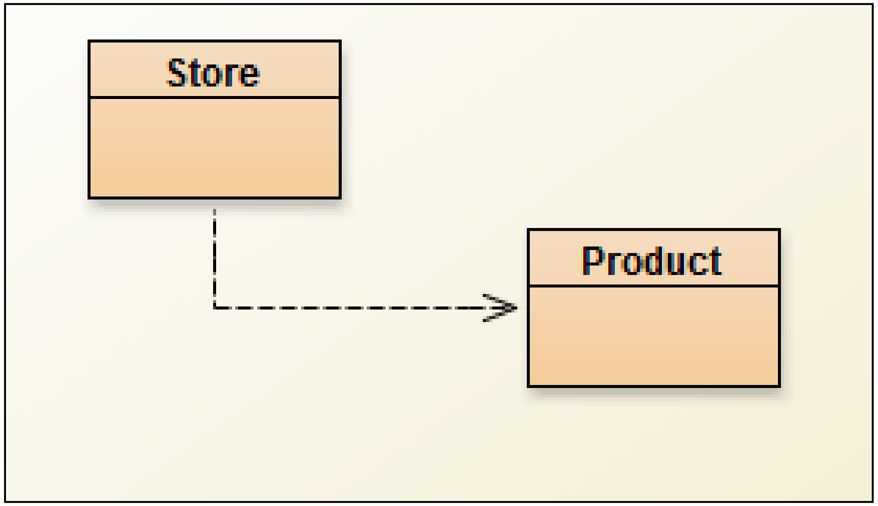
Product class: toString()

- When you print an object using code similar to **System.out.println(someObject)**, java will check the class for a toString method.

- If the toString() method:
  – exists, java will automatically call it and the user friendly object state will be printed.
  – doesn’t exist, java will print the class name followed by the memory location of the object.

A basic example of a Shop

A Store has an ArrayList of Product.



Store class

- The Store class will contain:
  1. an ArrayList of Product.
  2. a method to add Products to the ArrayList.
  3. a method to print out the contents of the ArrayList.
  4. a method that will print out the cheapest product in the ArrayList.

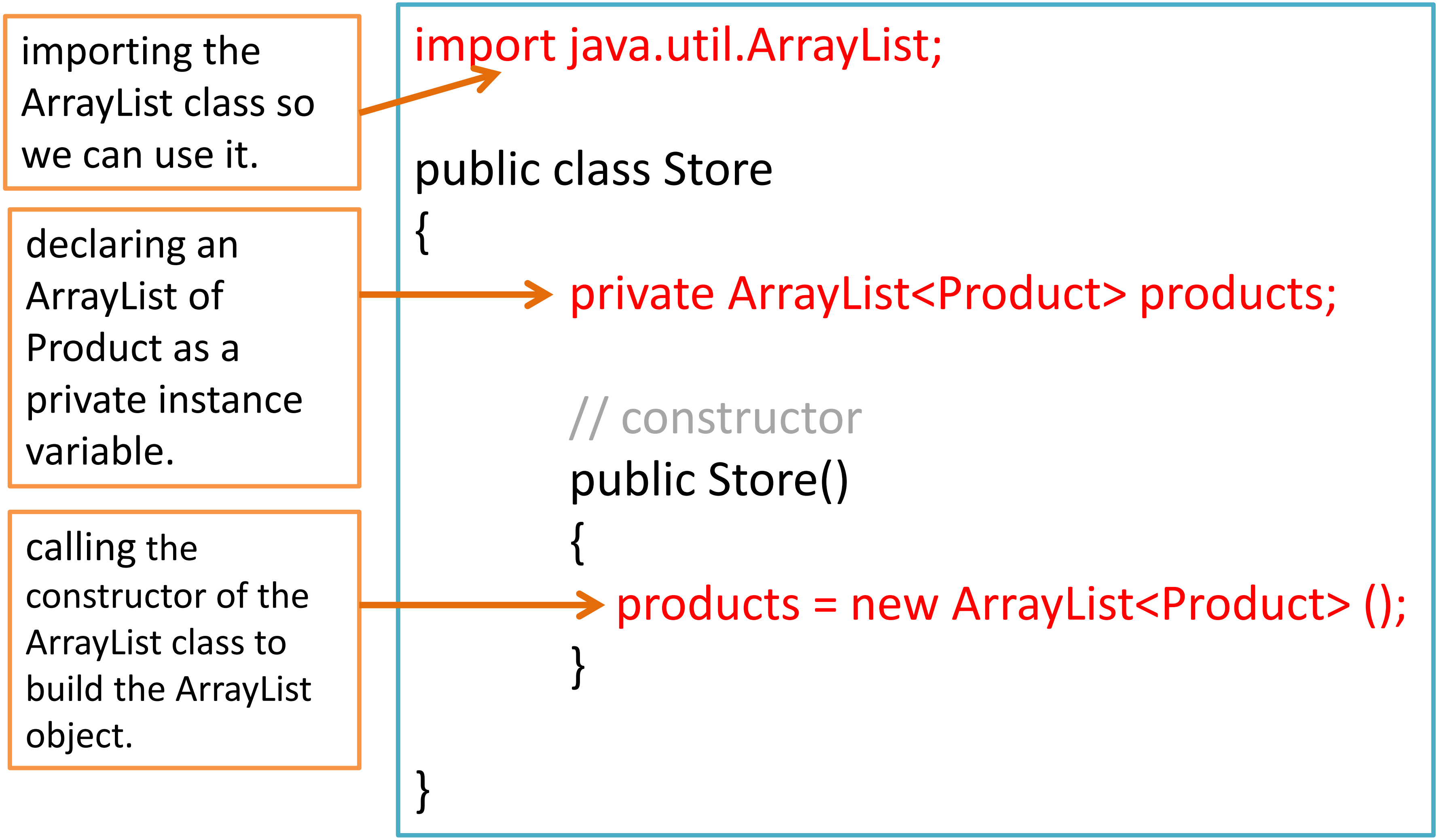
Store class

- The Store class will contain:
  1. an ArrayList of Product.
  2. a method to add Products to the ArrayList.
  3. a method to print out the contents of the ArrayList.
  4. a method that will print out the cheapest product in the ArrayList.

1. Declaring an ArrayList of Product

```
import java.util.ArrayList;

public class Store
{
        private ArrayList<Product> products;

        // constructor
        public Store()
        {
          products = new ArrayList<Product> ();
        }

}
```

1. Declaring an ArrayList of Product



importing the ArrayList class so we can use it.

declaring an ArrayList of Product as a private instance variable.

calling the constructor of the ArrayList class to build the ArrayList object.

```
import java.util.ArrayList;

public class Store
{
          private ArrayList<Product> products;

          // constructor
          public Store()
          {
             products = new ArrayList<Product> ();
          }

}
```

Store class

- The Store class will contain:
  1. an ArrayList of Product.
  2. a method to add Products to the ArrayList.
  3. a method to print out the contents of the ArrayList.
  4. a method that will print out the cheapest product in the ArrayList.

2. Add a Product object to an ArrayList of Product

```
public void add (Product product)
{
            products.add (product);
}
```

add: This is add method from the ArrayList class that we just imported.

products: This is the ArrayList of Product.

Product: The ArrayList holds objects of this type, Product.

product: This is object of type Product that we want to add to the ArrayList.

2. Add a Product object to an ArrayList of Product

```
import java.util.ArrayList;

public class Store{


        private ArrayList<Product> products;

        public Store(){
            products = new ArrayList<Product> ();
        }

        public void add (Product product){
            products.add (product);
        }
}
```

Store class

- The Store class will contain:
  1. an ArrayList of Product.
  2. a method to add Products to the ArrayList.
  3. a method to print out the contents of the ArrayList.
  4. a method that will print out the cheapest product in the ArrayList.

3. Printing all Products in an ArrayList of Product

```
public void listProducts(){
        for (Product product: products){
              System.out.println(product.toString());
        }
}
```

Product: This is the type of object that is stored in the ArrayList.

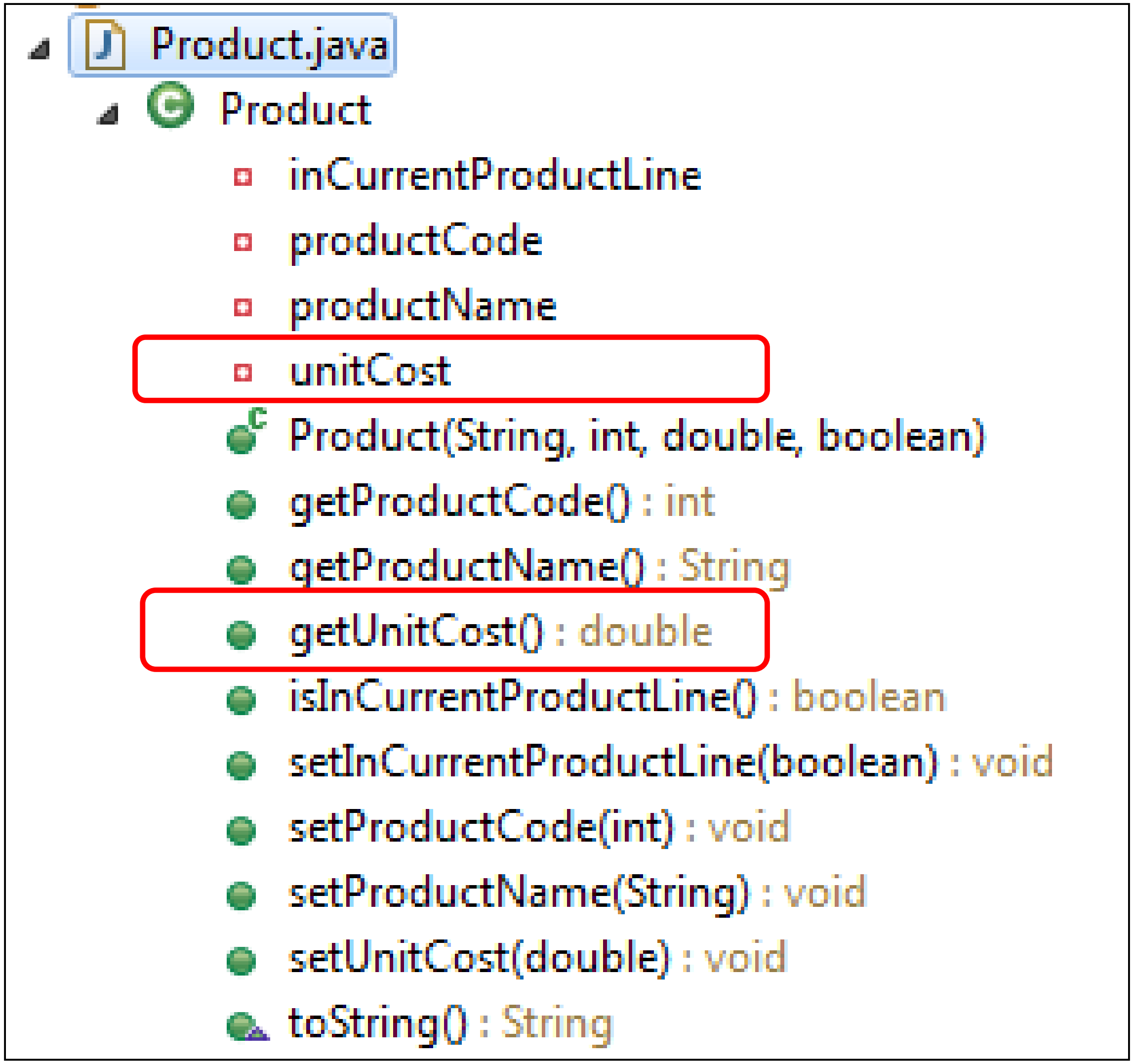
product: This is object reference pointing to the current object we are looking at in the ArrayList. As we iterate over each object in the ArrayList, this reference will change to point to the next object, and so on.

products: This is the ArrayList of Product.

Store class

- The Store class will contain:
  1. an ArrayList of Product.
  2. a method to add Products to the ArrayList.
  3. a method to print out the contents of the ArrayList.
  4. a method that will print out the cheapest product in the ArrayList.

Finding the Cheapest Product



```
Product.java
    Product
        inCurrentProductLine
        productCode
        productName
        unitCost
        Product(String, int, double, boolean)
        getProductCode() : int
        getProductName() : String
        getUnitCost() : double
        isInCurrentProductLine() : boolean
        setInCurrentProductLine(boolean) : void
        setProductCode(int) : void
        setProductName(String) : void
        setUnitCost(double) : void
        toString() : String
```

Finding the Cheapest Product

```
1.  If products have been added to the ArrayList
      1.1  Assume that the first Product in the ArrayList is the cheapest (set a local variable to store this object).
      1.2  For all product objects in the ArrayList
            1.2.1  if the current product cost is lower than the cost of the product object stored in the local variable,
                     1.2.1.1  update the local variable to hold the current product object.
                  end if
         end for
      1.3 Return the name of the cheapest product.
    else
      1.4 Return a message indicating that no products exist.
    end if
```

Finding the Cheapest Product

Working on the outer if statement

```
if products have been added to the ArrayList
        // return the cheapest product
else
        return a message indicating that no products exist.
end if
```

How do we write the code for this algorithm?

Finding the Cheapest Product

Working on the outer if statement

```
if products have been added to the ArrayList
        //return the cheapest product
else
        return a message indicating that no products exist.
end if
```

```
if (products.size() > 0){
        //return the cheapest product
}
else{
        return “No products added to the ArrayList”;
}
```

Working on step 1.1

```
if products have been added to the ArrayList
        // 1.1   Assume that the first Product in the ArrayList is the
        // cheapest (set a local variable to store this object).
else
        return a message indicating that no products exist.
end if
```

How do we write the code for this step?

Working on step 1.1

```
if products have been added to the ArrayList
        // 1.1   Assume that the first Product in the ArrayList is the
        // cheapest (set a local variable to store this object).
else
        return a message indicating that no products exist.
end if
```

```
if (products.size() > 0){
        Product cheapestProduct = products.get(0);
 }
else{
        return “No products added to the ArrayList”;
}
```

Working on the for loop

```
if products have been added to the ArrayList
        // 1.1   Assume that the first Product in the ArrayList is the
        // cheapest (set a local variable to store this object).
        // 1.2   For all product objects in the ArrayList
        //          determine the cheapest product
        //  end for
else
        return a message indicating that no products exist.
end if
```

How do we write the code for this step?

Working on the for loop

```
if (products.size() > 0){
        Product cheapestProduct = products.get(0);
        for (Product product : products)
        {
        }
 }
else{
        return “No products added to the ArrayList”;
}
```

Working on the code inside the for loop

```
1.  If products have been added to the ArrayList
    1.1   Assume that the first Product in the ArrayList  is the cheapest
          (set a local variable to store this object).
    1.2  For all product objects in the ArrayList
          1.2.1  if the current product cost is lower than the cost of
                   the product object stored in the local variable,
                        1.2.1.1  update the local variable to hold the
                                   current product object.
                 end if
          end for
    1.3 Return the name of the cheapest product.
    else
    1.4 Return a message indicating that no products exist.
    end if
```

How do we write the code for this?

Working on the code inside the for loop

```
if (products.size() > 0){
        Product cheapestProduct = products.get(0);
        for (Product product : products){
            if (product.getUnitCost() < cheapestProduct.getUnitCost() )
            {
            }
        }
 }
else{
        return “No products added to the ArrayList”;
}
```

Working on the code inside the for loop

```
1.  If products have been added to the ArrayList
    1.1  Assume that the first Product in the ArrayList  is the cheapest
         (set a local variable to store this object).
    1.2  For all product objects in the ArrayList
         1.2.1  if the current product cost is lower than the cost of
                the product object stored in the local variable,
                1.2.1.1  update the local variable to hold the
                         current product object.
                end if
         end for
    1.3 Return the name of the cheapest product.
    else
    1.4 Return a message indicating that no products exist.
    end if
```

How do we write the code for this step?

Working on the code inside the for loop

```
if (products.size() > 0){
        Product cheapestProduct = products.get(0);
        for (Product product : products){
           if (product.getUnitCost() < cheapestProduct.getUnitCost() ){
                cheapestProduct = product;
           }
        }
 }
else{
        return “No products added to the ArrayList”;
}
```

Working on the last step, 1.3

```
1.   If products have been added to the ArrayList
     1.1   Assume that the first Product in the ArrayList  is the cheapest
           (set a local variable to store this object).
      1.2  For all product objects in the ArrayList
           1.2.1  if the current product cost is lower than the cost of
                     the product object stored in the local variable,
                            1.2.1.1  update the local variable to hold the
                                           current product object.
                  end if
           end for
     1.3 Return the name of the cheapest product.
     else
     1.4 Return a message indicating that no products exist.
     end if
```

How do we write the code for this step?

Working on the last step, 1.3

```
if (products.size() > 0){
        Product cheapestProduct = products.get(0);
        for (Product product : products){
            if (product.getUnitCost() < cheapestProduct.getUnitCost()){
                cheapestProduct = product;
             }
        }
        return cheapestProduct.getProductName();
 }
else{
        return “No products added to the ArrayList”;
}
```

Questions?

Review

- Collections allow an arbitrary number of objects to be stored.
- Class libraries usually contain tried-and-tested collection classes.
- Java’s class libraries are called *packages*.
- We have used the `ArrayList` class from the `java.util` package.

Review

- Items may be added and removed.
- Each item has an index.
- Index values may change if items are removed (or further items added).
- The main `ArrayList` methods are `add`, `get`, `remove` and `size`.
- `ArrayList` is a parameterized or generic type.

Review

- Loop statements allow a block of statements to be repeated.
- The for-each loop allows iteration over a whole collection.
- The while loop allows the repetition to be controlled by a boolean expression.