More on ArrayLists

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

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

Ollscoil Teicneolaíochta an Oirdheiscirt
South East Technological University

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

Topic list

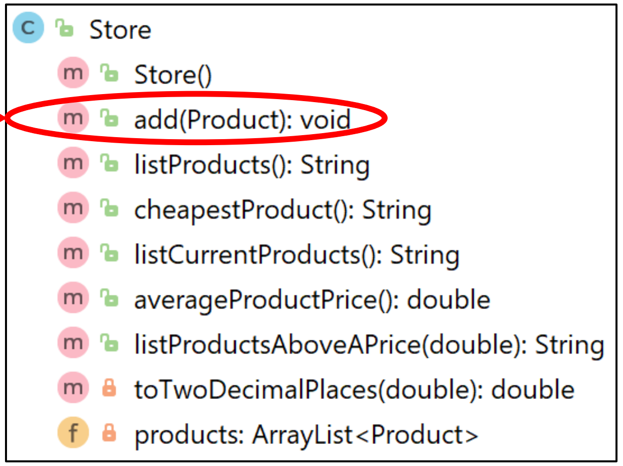
- Grouping Objects
  - Developing a basic personal notebook project using Collections e.g. ArrayList
- Indexing within Collections
  - Retrieval and removal of objects
- Generic classes e.g. ArrayList
- Iteration
  - Using the for loop
  - Using the while loop
  - Using the for each loop
- Shop – use an **ArrayList of Products – more methods**

Methods (1)

Store

These methods work on the **ArrayList** to:

1. **add** Products
2. print out the contents
3. print out the cheapest product



```
Store
    Store()
    add(Product): void
    listProducts(): String
    cheapestProduct(): String
    listCurrentProducts(): String
    averageProductPrice(): double
    listProductsAboveAPrice(double): String
    toTwoDecimalPlaces(double): double
    products: ArrayList<Product>
```

Add a product object to an ArrayList of Product

Store

The ArrayList holds objects of this type

This is an object variable of type Product that we want to add to the ArrayList.

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

Add a product object to an ArrayList of Product

Store

The ArrayList holds objects of this type

This is an object variable of type Product that we want to add to the ArrayList.

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

The ArrayList of Product

This is the **.add()** method from the ArrayList class that we just imported.

Add a product object to an ArrayList of Product

Store

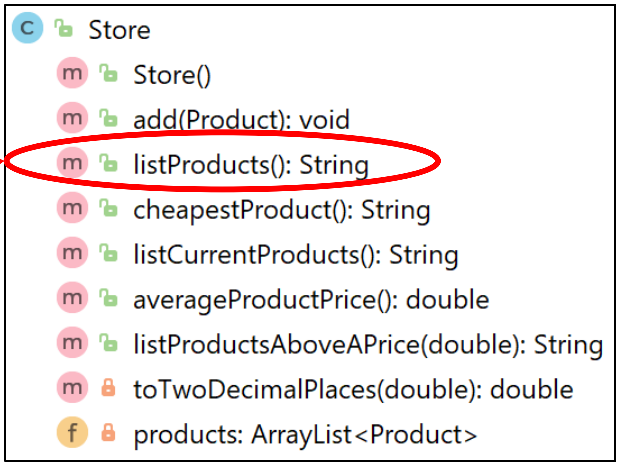
```
import java.util.ArrayList;

public class Store{

          private ArrayList<Product> products;

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

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

Methods (2)

Store

These methods work on the **ArrayList** to:

1. **add** Products
2. print out the contents
3. print out the cheapest product



```
Store
  Store()
  add(Product): void
  listProducts(): String
  cheapestProduct(): String
  listCurrentProducts(): String
  averageProductPrice(): double
  listProductsAboveAPrice(double): String
  toTwoDecimalPlaces(double): double
  products: ArrayList<Product>
```

Store

Print out the contents

```java
public String listProducts() {
    if (products.size() == 0) {
        return "No products";
    } else {
        String listOfProducts = "";
        for (int i = 0; i < products.size(); i++) {
            listOfProducts += i + ": " + products.get(i) + "\n";
        }
        return listOfProducts;
    }
}
```

Sample Output

```
0: Product description: Product1, product code: 1, unit cost: 45.99, currently in product line: true
1: Product description: Product2, product code: 2, unit cost: 12.99, currently in product line: false
2: Product description: Product3, product code: 3, unit cost: 23.5, currently in product line: true
```

Store

Print out the contents

If the size of the products ArrayList is **zero**,
return the String “No products” to the Driver class to be printed.

```java
public String listProducts() {
    if (products.size() == 0) {
        return "No products";
    } else {
        String listOfProducts = "";
        for (int i = 0; i < products.size(); i++) {
            listOfProducts += i + ": " + products.get(i) + "\n";
        }
        return listOfProducts;
    }
}
```

Sample Output

```
0: Product description: Product1, product code: 1, unit cost: 45.99, currently in product line: true
1: Product description: Product2, product code: 2, unit cost: 12.99, currently in product line: false
2: Product description: Product3, product code: 3, unit cost: 23.5, currently in product line: true
```

Store

Print out the contents

```
public String listProducts() {
    if (products.size() == 0) {
        return "No products";
    } else {
        String listOfProducts = "";
        for (int i = 0; i < products.size(); i++) {
            listOfProducts += i + ": " + products.get(i) + "\n";
        }
        return listOfProducts;
    }
}
```

If there are products in the ArrayList…
return a String containing the index number of each product & the product details.

Sample Output

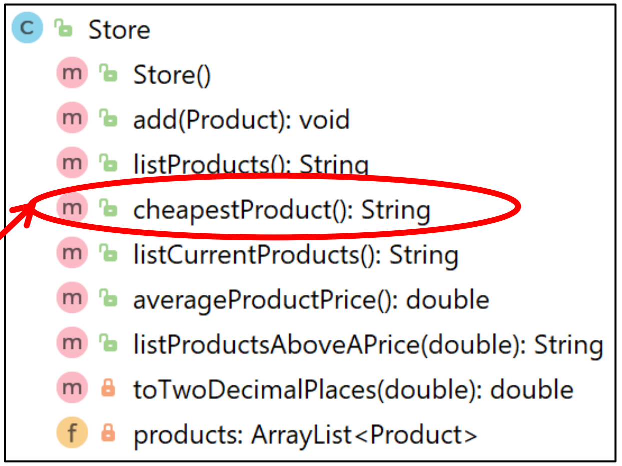
```
0: Product description: Product1, product code: 1, unit cost: 45.99, currently in product line: true
1: Product description: Product2, product code: 2, unit cost: 12.99, currently in product line: false
2: Product description: Product3, product code: 3, unit cost: 23.5, currently in product line: true
```

Store

Methods (3)

These methods work on the **ArrayList** to:

1. **add** Products
2. print out the contents
3. print out the cheapest product

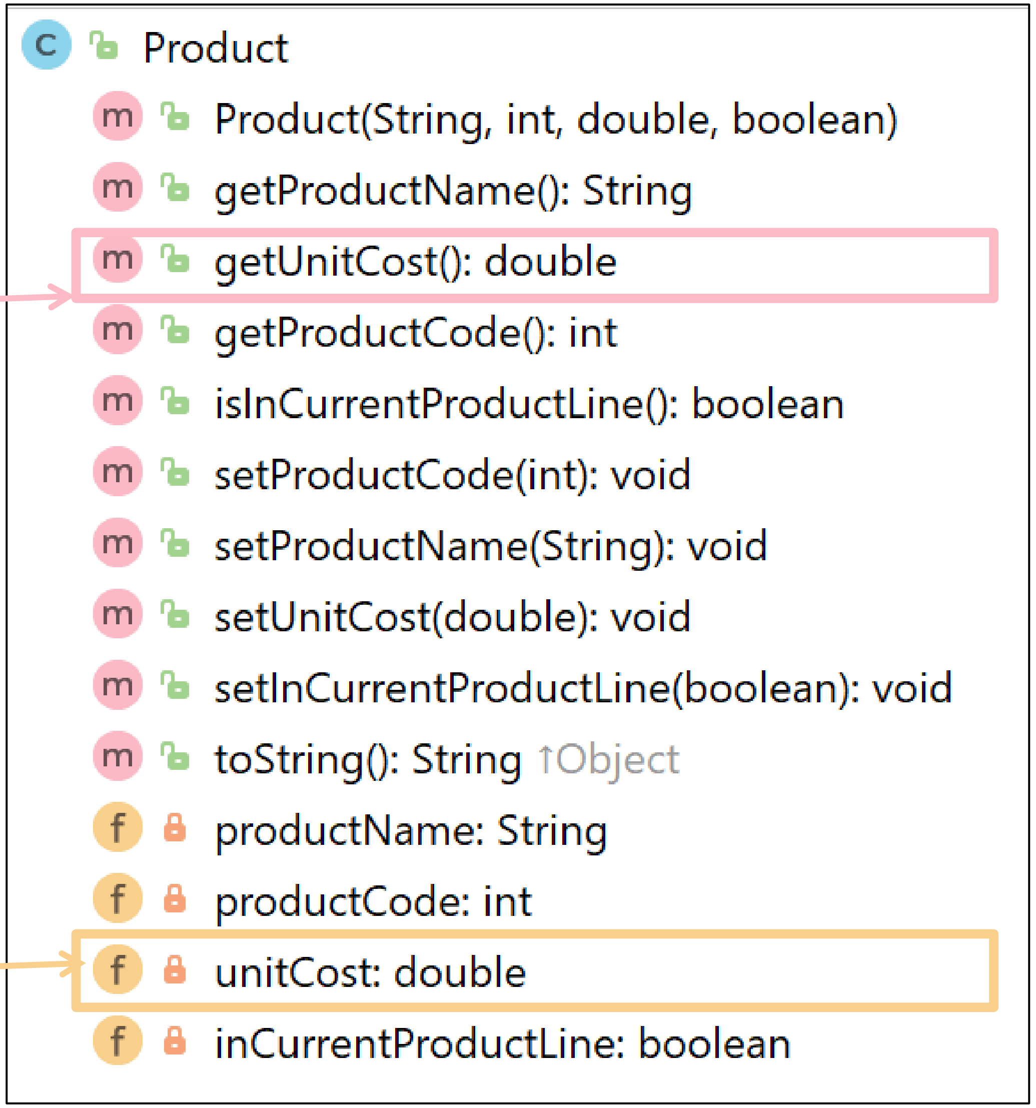


```
Store
  Store()
  add(Product): void
  listProducts(): String
  cheapestProduct(): String
  listCurrentProducts(): String
  averageProductPrice(): double
  listProductsAboveAPrice(double): String
  toTwoDecimalPlaces(double): double
  products: ArrayList<Product>
```

Product Finding the Cheapest **Product**

getter

private field – unit cost



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

Finding the Cheapest Product

Store

Algorithm (numbered steps)

```
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 (step 1.)

Store

Working on the outer if statement (step 1.)

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

Q: How do we write the code for this algorithm?

Store

```
if (products.size() != 0){
     //return the cheapest product
}
else{
     return "No products are in the ArrayList";
}
```

Store

Another valid approach!

```
if (!products.isEmpty()){
     //return the cheapest product
}
else{
     return "No products are in the ArrayList";
}
```

Store

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
```

Q: How do we write the code for this step?

Store

step 1.1

```
if (products.size() != 0){
       Product cheapestProduct = products.get(0);
 }
else{
      return "No products are in the ArrayList";
}
```

Working on the for loop **step 1.2**

Store

```
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
```

Q: How do we write the code for this step?

Store

step 1.2

```
if (products.size() > 0){
        Product cheapestProduct = products.get(0);
        for (Product product : products)
        {
        }
 }
else{
        return "No products are in the ArrayList";
}
```

Store

for each loop

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

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

product: This is the reference 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.

step 1.2.1

Store

```
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
```

Q: How do we write the code for this step?

Store

step 1.2.1

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

Store

Step 1.2.1.1

```
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).
      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
```

Q: How do we write the code for this step?

Store

Step 1.2.1.1

```
if (products.size() > 0){
        Product cheapestProduct = products.get(0);
        for (Product product : products){
             if (product.getUnitCost() < cheapestProduct.getUnitCost()){
                cheapestProduct = product;
              }
        }
 }
else{
        return "No products are in the ArrayList";
}
```

Store

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
```

Q: How do we write the code for this step?

Store

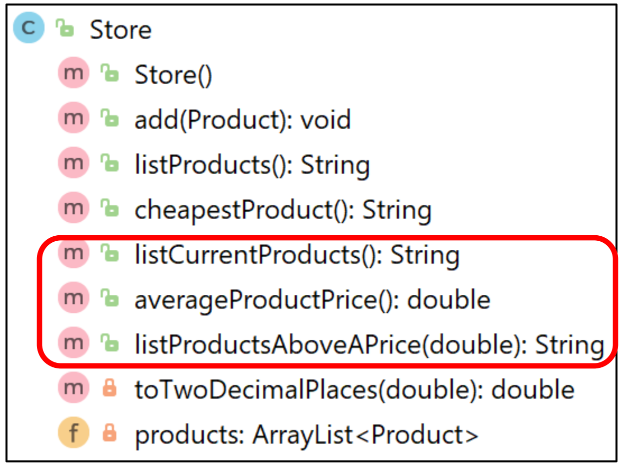
step, 1.3

```
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 are in the ArrayList";
}
```

Methods

We have look at the following methods:

1. add Products
2. print out the contents
3. print out the cheapest product



```
Store
    Store()
    add(Product): void
    listProducts(): String
    cheapestProduct(): String
    listCurrentProducts(): String
    averageProductPrice(): double
    listProductsAboveAPrice(double): String
    toTwoDecimalPlaces(double): double
    products: ArrayList<Product>
```

There are other methods in Store that operate on the ArrayList. They have a similar approach to the methods above; we can look at these in labs if you wish.

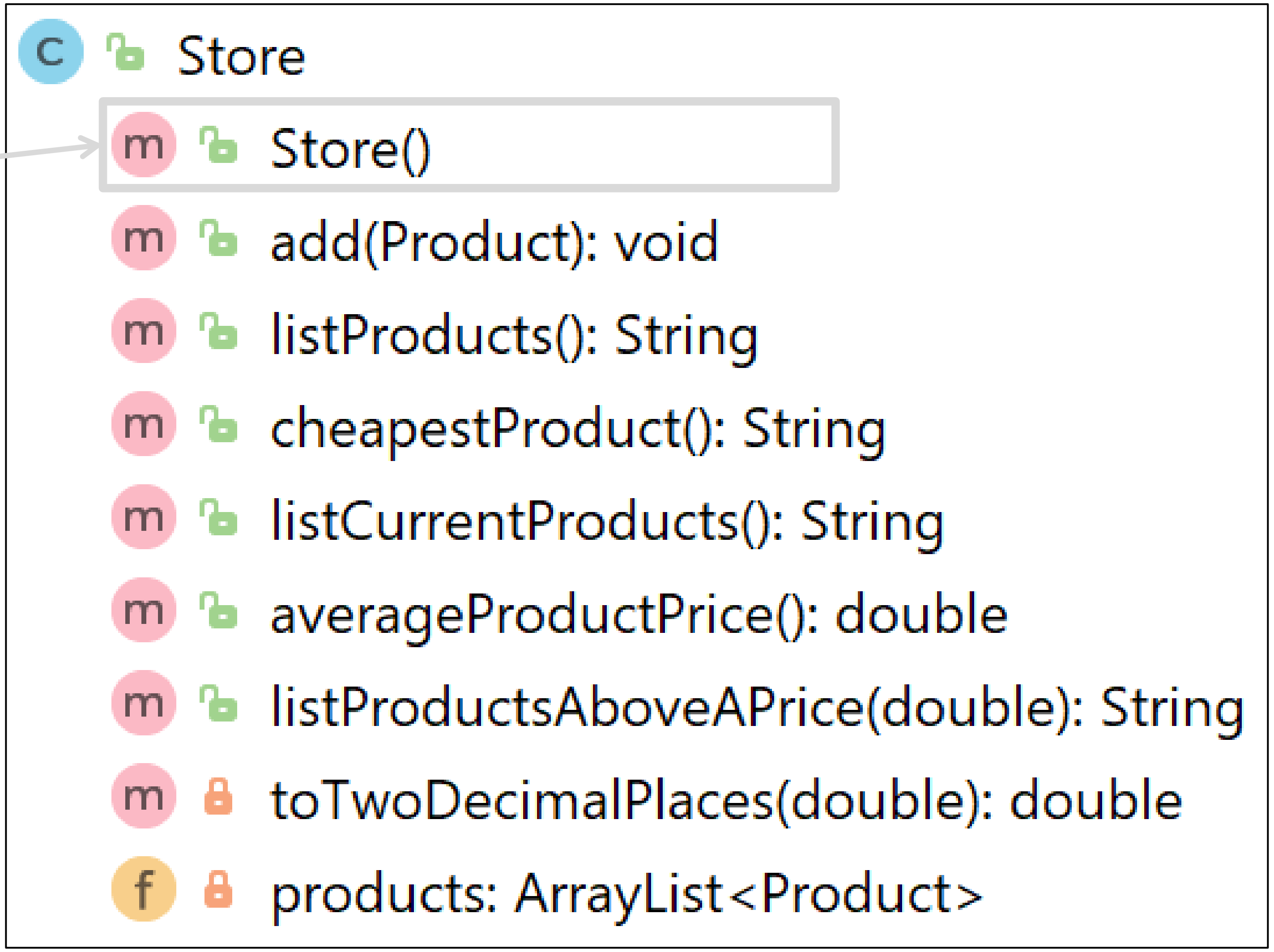
DRIVER

Refactor this class to handle a small change to the **Store** “interface”.

Store

Constructor

This was the only change to the public interface; we removed the parameter!



Store
Store()
add(Product): void
listProducts(): String
cheapestProduct(): String
listCurrentProducts(): String
averageProductPrice(): double
listProductsAboveAPrice(double): String
toTwoDecimalPlaces(double): double
products: ArrayList<Product>

Driver

Previously our Shop used an array and we needed to know how many Products to store:

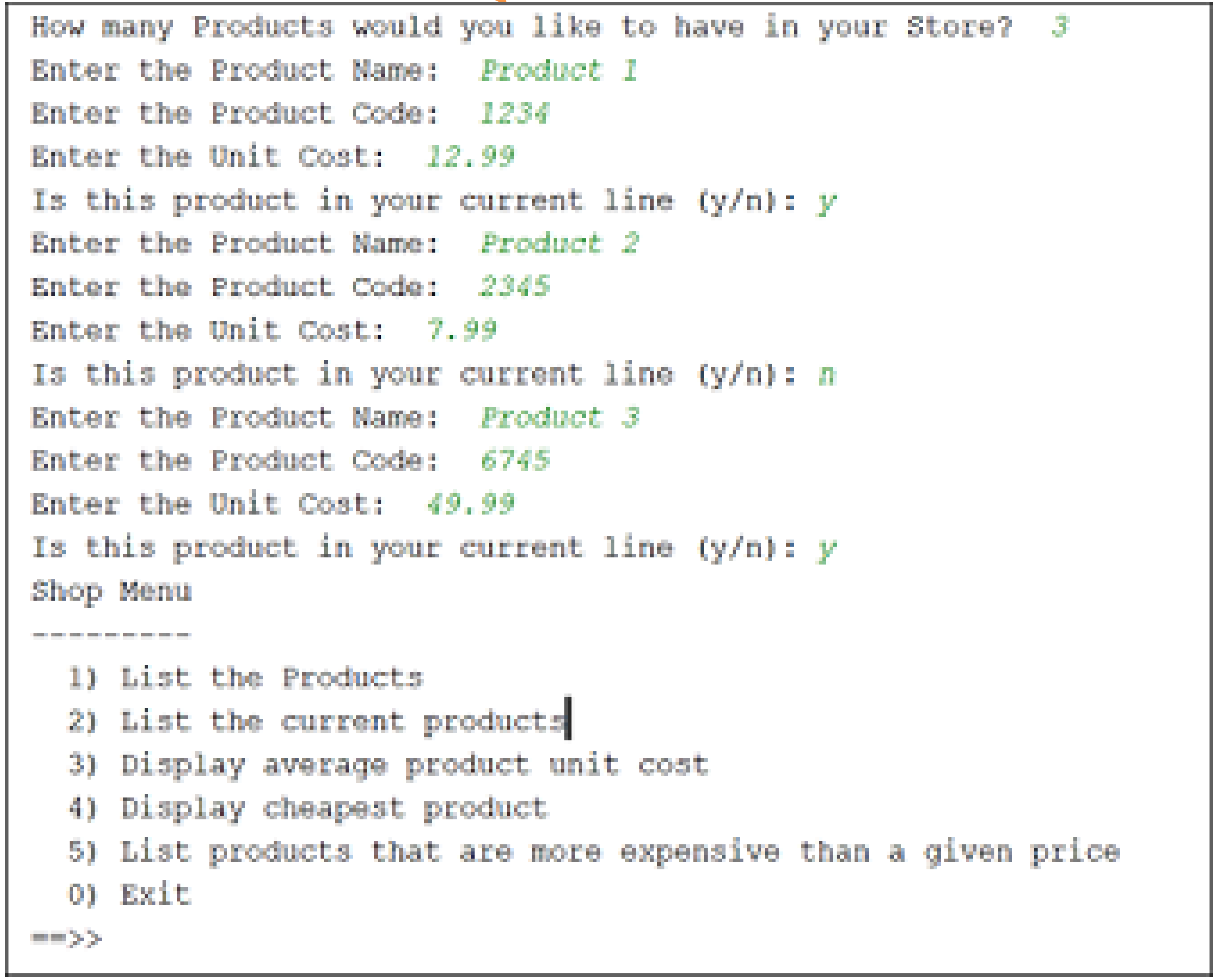
```
store = new Store(numberProducts);
```

Now that we are using an ArrayList, we don't need to set a capacity, so our constructor call becomes:

```
store = new Store();
```

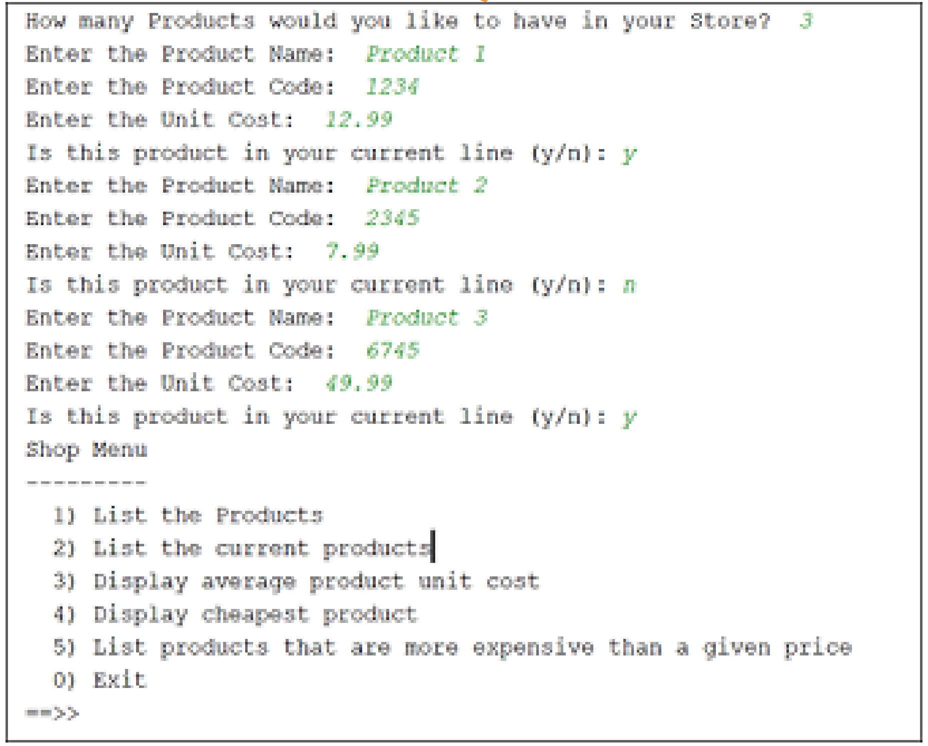
Driver

Our output when we run V3.0 of the app:



```
How many Products would you like to have in your Store?  3
Enter the Product Name:  Product 1
Enter the Product Code:  1234
Enter the Unit Cost:  12.99
Is this product in your current line (y/n): y
Enter the Product Name:  Product 2
Enter the Product Code:  2345
Enter the Unit Cost:  7.99
Is this product in your current line (y/n): n
Enter the Product Name:  Product 3
Enter the Product Code:  6745
Enter the Unit Cost:  49.99
Is this product in your current line (y/n): y
Shop Menu
---------
  1) List the Products
  2) List the current products
  3) Display average product unit cost
  4) Display cheapest product
  5) List products that are more expensive than a given price
  0) Exit
==>>
```

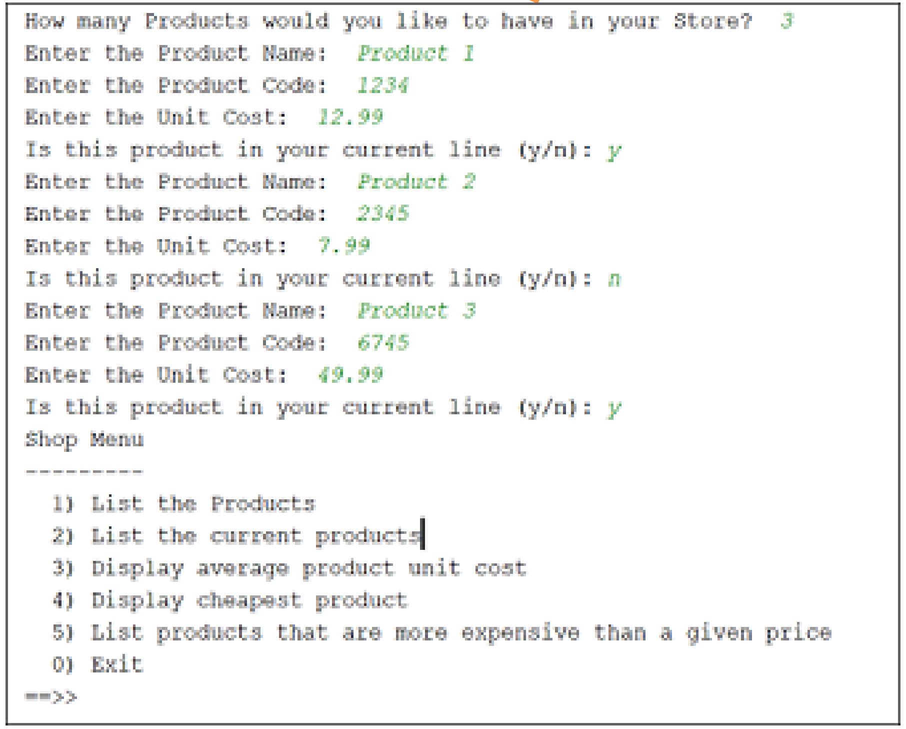
V3.0 output:



```
How many Products would you like to have in your Store?  3
Enter the Product Name:  Product 1
Enter the Product Code:  1234
Enter the Unit Cost:  12.99
Is this product in your current line (y/n): y
Enter the Product Name:  Product 2
Enter the Product Code:  2345
Enter the Unit Cost:  7.99
Is this product in your current line (y/n): n
Enter the Product Name:  Product 3
Enter the Product Code:  6745
Enter the Unit Cost:  49.99
Is this product in your current line (y/n): y
Shop Menu
---------
  1) List the Products
  2) List the current products
  3) Display average product unit cost
  4) Display cheapest product
  5) List products that are more expensive than a given price
  0) Exit
==>>
```

No difference in output…we only changed the internal storage mechanism!

V2.2 output:



```
How many Products would you like to have in your Store?  3
Enter the Product Name:  Product 1
Enter the Product Code:  1234
Enter the Unit Cost:  12.99
Is this product in your current line (y/n): y
Enter the Product Name:  Product 2
Enter the Product Code:  2345
Enter the Unit Cost:  7.99
Is this product in your current line (y/n): n
Enter the Product Name:  Product 3
Enter the Product Code:  6745
Enter the Unit Cost:  49.99
Is this product in your current line (y/n): y
Shop Menu
---------
  1) List the Products
  2) List the current products
  3) Display average product unit cost
  4) Display cheapest product
  5) List products that are more expensive than a given price
  0) Exit
==>>
```

Collections

- Allow an **arbitrary number** of objects to be stored.
- Are implemented in **Java's Class libraries** which contain tried-and-tested collection classes.
- In Java's class libraries are called ***packages***.
- We have used the `ArrayList` class from the `java.util` **package**.

ArrayList

- 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()`**
  - **`size()`**
- `ArrayList` is a parameterized or generic type.

Questions?