| **Exercise 4.54** Continue working with the *club* project from Exercise 4.40. Define a method in the **Club** class with the following description:**/\*\*****\* Determine the number of members who joined in the****\* given month.****\* @param month The month we are interested in.****\* @return The number of members who joined in that month.****\*/****public int joinedInMonth(int month)**If the **month** parameter is outside the valid range of 1 to 12, print an error message and return zero. |
| --- |
| **Exercise 4.55** Define a method in the **Club** class with the following description:**/\*\*****\* Remove from the club's collection all members who****\* joined in the given month, and return them stored****\* in a separate collection object.****\* @param month The month of the membership.****\* @param year The year of the membership.****\* @return The members who joined in the given month and year.****\*/****public ArrayList<Membership> purge(int month, int year)**If the **month** parameter is outside the valid range of 1 to 12, print an error message and return a collection object with no objects stored in it.*Note:* The **purge** method is significantly harder to write than any of the others in this class. You should walk through the list backward to avoid creating runtime errors. |
| **Exercise 4.56** Open the *product* project and complete the **StockManager** class through this and the next few exercises. **StockManager** uses an **ArrayList** to store **Product** items. Its **addProduct** method already adds a product to the collection, but the following **four**  methods need completing: **delivery (exercise 4.59)**, **findProduct (exercise 4.57)**, **printProductDetails (exercise 4.56)**, and **numberInStock (exercise 4.58)**. Each product sold by the company is represented by an instance of the **Product** class, which records a product’s ID, name, and how many of that product are currently in stock. The **Product** class defines the **increaseQuantity** method to record increases in the stock level of that product. The **sellOne** method records that one item of that product has been sold, by reducing the quantity field level by 1. **Product** has been provided for you, and you should not need to make any alterations to it. Start by implementing the **printProductDetails** method to ensure that you are able to iterate over the collection of products. Just print out the details of each **Product** returned, by calling its **toString** method. (Fadoir note: the **toString** method makes a string that summarizes the product.) |
| **Exercise 4.57** Implement the **findProduct** method. This should look through the collection for a product whose **id** field matches the ID argument of this method. If a matching product is found, it should be returned as the method’s result. If no matching product is found, return **null**. This differs from the **printProductDetails** method in that it will not necessarily have to examine every product in the collection before a match is found. For instance, if the first product in the collection matches the product ID, iteration can finish and that first **Product** object can be returned. On the other hand, it is possible that there might be no match in the collection. In that case, the whole collection will be examined without finding a product to return. In this case, the **null** value should be returned.When looking for a match, you will need to call the **getID** method on a **Product**. |
| **Exercise 4.58** Implement the **numberInStock** method. This should locate a product in the collection with a matching ID and return the current quantity of that product as a method result. If no product with a matching ID is found, return zero. This is relatively simple to implement once the **findProduct** method has been completed. For instance, **numberInStock** can call the **findProduct** method to do the searching and then call the **getQuantity** method on the result. Take care over products that cannot be found, though. |
| **Exercise 4.59** Implement the **delivery** method using a similar approach to that used in **numberInStock**. It should find the product with the given ID in the list of products and then call its **increaseQuantity** method. |
| **Exercise 4.60** *Challenge exercises* **One:** Implement a method in **StockManager** to print details of all products with stock levels below a given value (passed as a parameter to the method).**Two:** Modify the **addProduct** method so that a new product cannot be added to the product list with the same ID as an existing one.**Three:** Add to **StockManager** a method that finds a product from its name rather than its ID:**public Product findProduct(String name)**In order to do this, you need to know that two **String** objects, **s1** and **s2**, can be tested for equality with the boolean expression **s1.equals(s2)** |