Exercises 04 – Linked lists in Java and Python

# Introduction

The last exercises involved using lists in Python and Java. This follows on but concentrates on Java and the subsequent material on searching for items in lists. Remember to consult the lecture slides for similar examples and the guide documents for anything you have trouble with.

## Java exercises - using the List interface type

1. Start a new Java program by creating a new class with a main method in your existing Java project, or if you prefer by creating a new project (see *00-Netbeans guide.pdf*). Call the class **ListExperiments**.
2. In the **main** method create a variable called ***mylist*** whose type is **ArrayList<String>** and assign a new instance of **ArrayList** to it. In order to do this you will need to add import statements to your class to import java.util.ArrayList - your IDE will probably offer to do this for you (alternatively you can add a single statement import java.util.\*; that imports everything in the *util* package). See *00-Netbeans guide.pdf* and *01-The basic strucure of a program.pdf* also *07-Arrays and Lists.pdf*.
3. Now add a new static method to your class as follows:

public static void demonstrateList(ArrayList<String> list) {

list.add("Apples");

list.add("Oranges");

list.add("Bananas");

System.out.println( list);

list.add(1, "Pineapples");

System.out.println( list);

list.remove(2);

System.out.println( list);

System.out.println( list.size());

}

1. In the main method, add a call to demonstrateList giving it your mylist object as the argument.
2. Run the program - you should see the following output:

[Apples, Oranges, Bananas]

[Apples, Pineapples, Oranges, Bananas]

[Apples, Pineapples, Bananas]

3

1. Make sure you understand how the code corresponds to the output.
2. Review the lecture slides from 04-Linked Lists and Abstact Data Types.pptx that discuss abstract data types and Java interfaces.
3. Examine the line you have that declares your ***mylist*** variable. It should read  
   ArrayList<String> mylist = new ArrayList<>();  
   Comment this line out (turn it into a comment that is ignored when you run the program) by placing // before it, and add a new line below it as follows:  
   LinkedList<String> mylist = new LinkedList<>();
4. You will need to add a statement to import java.util.LinkedList (or get the IDE to do it for you) but after this your IDE should still complain that you can't call *demonstrateList* with ***mylist*** as an argument because ***mylist*** is declared to be of type **LinkedList** not **ArrayList**.
5. Change the type of the parameter defined in *demonstrateList* so it is **LinkedList** instead of **ArrayList**. The program should now run normally, and produce the same output as before. The important thing is that apart from changing the type of the parameter, everything in the demonstrateList method works just as well with a LinkedList as with an ArrayList. To allow it to use either kind without any change, we can make its parameter into an abstract data type that describes "any kind of list". In Java we can do this using the **List** type. This is a Java ***interface*** (that defines the methods that will be available without specifying any implementation of them). Both **ArrayList** and **LinkedList** have all the methods defined in **List** and declare that they implement **List**, so anywhere we use the **List** type, we can use either kind.
6. Change the type of the parameter defined in *demonstrateList* so it is now **List<String>**. You will need to add a statement to import java.util.List (or get the IDE to do it for you) but after this your program should work just the same. The difference is that we can now call demonstrateList with either kind of list.
7. To prove this, change your main method to call demonstrateList with both kinds of list, something like this (note that we now have two variables mylist1 and mylist2):

ArrayList<String> mylist1 = new ArrayList<>();

demonstrateList(mylist1);

LinkedList<String> mylist2 = new LinkedList<>();

demonstrateList(mylist2);

## Python exercises - no formal abstract data types

1. Create a new Python program called linkedlists.py and add the following function to it:

def demonstrateList(list):

list.append("Apples")

list.append("Oranges")

list.append("Bananas")

print(list)

list.insert(1, "Pineapples")

print(list)

list.pop(2);

print(list)

print(len(list))

You should be able to recognise that this is equivalent to the Java code we used above.

1. Now add these lines to the bottom of your program:

print("Using a regular list")

demostrateList([])

1. Run the program - you should expect to see the same output we got in Java.
2. Now add an import statement as the first line of your program as follows:

from collections import deque

You have declared that you want to be able to use the ***deque*** class from the *collections* module.

1. Now add the following lines to your main program:

print("Now using a deque")

d = deque()

demonstrateList(d)

This creates a deque object and assigns it to the variable d, then passes that to the demonstrateList method.

1. Run the program and see what happens.
2. Although Python did not warn you in advance, the deque object does not have the functions used in the demonstrateList function. It does have *append* and *insert*, and they work just as they did on the regular list. The pop(2) call fails, because although deque has a pop() function, it takes no arguments and can only remove the last item on the linked list.

* Regular lists and deques *do not* share a common set of functions.
* There is no abstract data type (like Java's List interface) that formally relates them to each other.
* They cannot be used interchangeably UNLESS only the functions that they DO have in common are used.

1. Have a look at the official Python documentation of deque:  
   <https://docs.python.org/3/library/collections.html#collections.deque>
2. For a more in-depth look at the deque class, work through this tutorial:  
   <https://realpython.com/python-deque/>