1. Go to **Groups on Wide** and copy the folder **CSCU9Y4/Practicals/Practical2** into your Y4 Practicals folder.

The aim of this practical is to reinforce some of the ideas about types you have met in the lectures, and to reinforce ideas about the difference between compile-time and run-time checking. In addition, you will practice your skills in coding with lists and arrays.

You are going to experiment with the contents of a **LinkedList** to illustrate some points about the types of elements of the list. You will see how troublesome it can be when a list structure is of type **Object**.

2. Open **CastTest**. The code is very simple. Run the application to see what it does. Use the debugger to inspect objects and step through the code. (You may get a warning from the compiler about unsafe operations; that's what this example is about.) Notice the use of type casts, for example the statement

(String) a.get(0)

where the result of **a.get(0)** is made into a String in the **result.append** statements.

Now try the following changes (note that the suggested changes have already been implemented in the program; you just need to uncomment the right lines).

Run the program after each change and examine which things cause compiler errors, and which things cause runtime errors. Pay particular attention to what the error messages say about types.

a. Add an assignment to a newly defined variable **String s** of the first element of the list **a**. Why do we need the cast?
b. Assign a value to an Integer **i** of the third element of the list. Note again the requirement for a cast.
c. Try increasing the third element of the list by 1, see the code. Try to remove the cast.
d. Try assignments to **String** or **Integer** where you know the cast won't match the actual element in the list.
e. Remember that list contents can change. Set the third element to be a **String**. Now the broken assignment of the previous step should be fine.

Our list **a** is an object of the **LinkedList** class. Go to the Java API documentation and make sure you understand how it works. Check again the error messages you obtained previously. What is the type of items in the LinkedList? How does this type relate to others in the Java hierarchy? Write your answers below.

Often, structures such as **Arrays, ArrayList** and **LinkedList** are intended to be "homogeneous", i.e. all the elements are of the same (rather general) type. Java provides better support (than shown above) of this through generics, or type parameters. For instance, you may have a type like **LinkedList<String>**, which represents a linked list of

strings and is "an instance" of a more generic type **LinkedList<T>**, i.e. a "list of elements of some type" represented by the type parameter **T**. Similarly, we can have a type **LinkedList<Integer>** or even a type like **LinkedList<StudentRecord>** (where we assume the class **StudentRecord** has been previously declared.

The rest of the practical is about playing with different ways of representing an ordered set of elements using Java library components, based on generics.

3. Copy Groups on Wide\CSCU9Y4\Practicals\Practical2\ArrTokenTest into your Y4Pracs folder.

Run ArrTokenTest. A window with two text fields, a text area and a button appears. What does the code do? Write the answer below.

In method **actionPerformed**, **split** produces an array of strings which is stored in the array **anArray**. We then call method print to display the contents of **anArray** in the textarea. You are going to adapt this code to deal with lists of strings instead of arrays.

4. In the class **ArrTokenTest** there is the stub of the method **mkList**. This method takes an array of strings and returns a **LinkedList<String>.** Implement the method body to traverse the array that is passed in to the method, taking each array element and inserting it into the list.

You can traverse the array in the usual way (as in the print method). You may need to look up some things in the Java API (e.g. how to add an object to a List).

Now add a statement to **actionPerformed** to call **mkList** and to store the result in a new variable **LinkedList<String> aList**. Compile your program to make sure there are no errors.

5. Now we want to print out this new data structure. In class **ArrTokenTest** there is a method **print** which will print out an array of **Strings**. Using this as a template, construct a new method **printList** which displays the contents of a **LinkedList<String>.**

You will have to change the type of the parameter and therefore make some changes to the body as you can no longer refer to **sa[i]**. The structure of the method can remain the same (a **for** loop) but you need to think about how to obtain an item in a list, how to tell that you've reached the end of the list, and how to obtain the length of the list. As above, check the Java API guide if you don't remember the details of these method calls.

Add a statement to **actionPerformed** to call **printList** to display the contents of **aList**. Your program should now be displaying two lists of tokens from your input string. Test your program and ensure the same list of tokens is printed each time.

6. You know about **LinkedList**. **ArrayList** provides a different implementation of lists. What would you do to change your program to use **ArrayList** instead of **LinkedList**? (Has your answer changed since last week?) Don't make the change, just discuss with your neighbours what you think the answer is, and write it here.

Furthermore, **List** is an interface (essentially, a list of method definitions), which is implemented by both **ArrayList** and **LinkedList**. This means you can declare something to be of type **List<String>** and only need to specify the actual class you want to use, i.e. **LinkedList** or **ArrayList**, when using the constructor, as required in the following code.

7. Change the declaration of **aList** to be **List<String>** in **actionPerformed**. Compile your code and see what the compiler makes of this change. You will probably get a compiler error. List is defined in both the packages **awt** and **util** and the compiler does not know which version to use. In order to tell it, you have to qualify the declaration of the **List** by prefixing with the one you actually want to use. To fix this, replace

List<String> with **java.util.List<String>**

Now the compiler knows which one to use. Change the use of **LinkedList<String>** to **List<String>** in both **mkList** and **printList**. The only use which remains as **LinkedList** is the constructor for **st** in **mkList**.

Above, you constructed **printList** by using a for loop. Now that you're using **List<String>** you can make access to the list more generic using an iterator. An iterator allows you to repeatedly step through (iterate) a list in a uniform way.

8. The code for a modified **printItList** using an iterator (with calls of **hasNext** and **next**) is given in comments in class **ArrTokenTest**. Uncomment this version and call it instead of **printList**. Again test your program to make sure you're still getting the same output as before.

Now a test to check that you are getting comfortable with iterators and lists.

9. Make a copy of the **ArrTokenTest** project and call the new project **ExtendedListTokenTest**. Delete the line in **actionPerformed** relating to printing the array of tokens. Run your program and confirm that it operates as before, but only prints the list once.

10. Modify **printItList** so that it prints out the length of the list passed in to it.

11. Define a method **removeEveryOtherToken** that will use an iterator to traverse the list, removing every other token from the list. That is, if the input list is "a b c d e f" then the output list should be "a c e". You can use the method **printItList** as a template. Add a call of **removeEveryOtherToken** to **actionPerformed** and display your answer with a call to **printItList**. Note: you must remove elements from the list, not just skip over them when printing. The goal is to change the stored list.

Checkpoint
Now demonstrate to a tutor that you have completed all this week's tasks. You should show us ArrTokenTest with the methods for printList and mkList, and the two outputs of the list of tokens. You should also show us the ExtendedListTokenTest which should print the length of the list and call the method removeEveryOtherToken. **If you didn't get to this point, finish the work off during the coming week and get the checkpoint marked next time. Checkpoints are dealt with as Canvas quizzes.**

Further tasks to develop your understanding:

Add one line of code to sort the contents of your **aList** (before passing it into **printItList**).

The **Arrays** class has many static methods for handling arrays of different types. There is a method **asList** which will view an array as a list. Note that this view is dynamic; if you subsequently change something in the list view, the underlying array will change too. Add a statement to **actionPerformed** to assign the list view of **anArray** to the variable **vList**. Check the syntax of the method call using the Java API.

Add a call of **printList** to **actionPerformed** to display the contents of **vList**. Test your modified program. What happens if you modify one of the elements of **aList**? Do you also change **vList**? Print both lists again to check.

So, the data you're holding is the same, but now you view it both as an array and as a list.