Unit 1: OO Programming

Exercise 3: Extending the Collections API

Aim of the Exercise
This exercise requires you to implement your own, custom data structure for the Java Collections API to meet some unusual requirements. **The aim is for you to gain a deeper understanding of how to create and use generic classes, and how to use and implement interfaces.**

Along the way, it should also make some of the data structures theory you learned in SCC120 a little more real…

The Task

Your task is to create an elegant and extensible data structure specifically designed for **error logging**. An error log is simply a record of all the errors, warnings and alerts a computer system encounters during its operation. The aim of the log is to then allow the staff who manage those computer systems to detect any issues that may affect the performance and security of that system. For example, a web server typically logs all web page requests. Internet routers will log periods of excessive internet traffic, as this may constitute a cyber attack, etc.

More specifically, you **must** adhere to the following requirements:

- You **must** use create your own data structure **from first principles**.
- You **must** use the Java Collections API interfaces, such that your data structure operates consistently with existing data structures.
- Your data structure must be **fully type safe**.
- Your data structure **must** be capable of storing objects of any Java class.
- The class representing your data structure must be a generic class called **OptimizedLog** that is compliant with the Java Collection interface.
- It should be possible to add entries into the log, but these entries are always added to the end of the log. It should be impossible to insert entries at any other location.
- It should be impossible to edit or remove an entry from the log once it has been added.
- Each log entry should also automatically record the time and date at which the entry was created.
- It is not uncommon for the same error to be logged multiple times in rapid succession. Your data structure should optimise this by detecting when consecutive identical log entries are made and record the number of times that message has been recorded instead of multiple copies. See overleaf for an example.
- Your data structure must accurately implement the Iterator interface, such that the original data logged by the user can be extracted via a standard foreach loop.
- Your data structure should respond to the toString() method that returns a string representation of each log entry recorded, in order, alongside the date/time and number of consecutive occurrences of that message.
- Two logs are considered equivalent if they contain the same messages in the same order (regardless of the time those were messages are created)

Step 1: Creating a Simple Test Harness

As you may not yet have used the collections API, you should start by creating a test harness using an existing, commonly used data structure – **ArrayList**.

Create a class containing the following code that creates an ArrayList capable of storing String objects, adds several items, then iterates through them using a foreach loop:

```
import java.util.*;

public class TestHarness
{
    public static void main(String[] args)
    {
        ArrayList<String> log = new ArrayList<>();

        log.add("Hello world!");
        log.add("SPAM...");
        log.add("The Eagle has landed...");
        log.add("SPAM...");
        log.add("SPAM...");
        log.add("SPAM...");

        for (String s : log)
            System.out.println(s);
    }
}
```

- Run the program and observe the output
- Experiment with the add() and remove() methods, and note how the collection can be iterated over using a foreach loop.
- Note also how this solution is fully type safe – the type returned from the collection is identical to that entered.
- Experiment to see how a collection holding a different type can be created and used. You may wish to create and use one of your own classes (e.g. Person, or Planet etc).

Step 2: Creating Your Own Collection

Create Java classes to meet the needs of the requirements at the start of this document, such that the following test code operates correctly:

```java
import java.util.*;

public class TestHarness
{
    public static void main(String[] args)
    {
        OptimizedLog<String> log = new OptimizedLog<>();

        log.add("Hello world!");
        log.add("Another entry...");

        log.add("SPAM...");
        log.add("SPAM...");
        log.add("SPAM...");

        System.out.println(log);
    }
}
```

The output of this test code should be something like the following:

```
[Wed Oct 16 19:31:41 BST 2019]: Hello world!
[Wed Oct 16 19:31:41 BST 2019]: Another entry...
[Wed Oct 16 19:31:41 BST 2019 - Wed Oct 16 19:31:43 BST 2019][3 TIMES]: SPAM...
```

Guidance Notes

You will need to use write at least two classes and accurately implement two interfaces to complete this task. You will also need to make use of generics, as discussed in the lectures.

- You will find two text files supplied along with this task (Collection.txt and Iterator.txt). These are (very slightly simplified) copies of the standard Java 11 collections interfaces. Study these interfaces. You will find these useful as a basis for your classes. (note that as these interfaces are already part of the Java SDK, you don't need to include these in your compilation – these text files are provided for reference only).

- Remember the constraints specified in the requirements. If you think this through, you will realise that these simplify the task (i.e. consider what your classes do and don't need to fully implement as a result of these constraints).

- Be sure to test that your classes operate properly.

Assessment

Submit your work by grouping your .java files into a zip file and submitting them to the submission point on the SCC212 moodle page. These will be tested offline against a test harness to determine how well you have completed the task.

Submission deadline: Friday Week 6.

You do not need to present this code for code review – it will be marked offline.

Automated testing will be performed on your code. Be careful to ensure you adhere to the guidelines provided in this document. Be particularly careful to ensure you use the correct name for your collection class: OptimizedLog

Place only your .java files into the zip file that you submit. Do not include class files, folders, or any other file.

Your classes should not be a member of any named package (i.e. do not use the package keyword anywhere in your java files).

We will use a command line build environment identical to that of the lab computers to test your code. Ensure your code compiles in that environment before submitting it. Any submission that fails to compile will receive a mark of zero.

Marking Scheme

Your work will be marked based on its ability to pass tests on the following functionality:

Ability to create an instance of a custom, type safe generic collection, as described in this document	20 %
Ability to add a log entry of any type	20 %
Ability to iterate over the collection using a foreach loop	20 %
Correct behaviour (ordered/append only/immutability)	20 %
Correct Implementation of toString() method, including time/date	10 %
Optimization of repeated entries	10 %