OLLSCOIL NA hÉIREANN
THE NATIONAL UNIVERSITY OF IRELAND, CORK

COLÁISTE NA hOLLSCOILE, CORCAIGH
UNIVERSITY COLLEGE, CORK

Examination Session and Year	Semester II Exam, 2019 – 2020
Module Code	CS2514
Module Title	Introduction to Java
Paper Number	1
External Examiner	Professor Omer Rana
Head of Department	Professor Cormac J. Sreenan (head@cs.ucc.ie)
Internal Examiners	Dr M. R. C. van Dongen (dongen@cs.ucc.ie)
Instructions to Candidates	Instructions are provided on Page 3
Duration of Paper	12 hours
Special Requirements	No special requirements

PLEASE DO NOT TURN THIS PAGE UNTIL INSTRUCTED TO DO SO.

ENSURE THAT YOU HAVE THE CORRECT PAPER.

Instructions

- Please refer to the document `CS2514-instructions.docx` for instructions related to exam duration and submission.
- The exam consists of 4 questions. Each question contributes a different mark.
- Check your code prior to submission.
- This is an open-book exam, you may consult with any material provided by the lecturer, your own notes, books, and API documentation. Besides API documentation, you may not use any material found online.
- You are allowed to use "standard" `Java` classes, provided they do not require an `import` statement. E.g. you are not allowed to use the `ArrayList` class.
- You are allowed to use any kind of predefined `Java` interface, even if they require an `import` statement.
- You are not allowed to use arrays.
- You are allowed to use strings but you should not use them to implement arrays or lists.

Please remember the following.

- Respect encapsulation.
- Use meaningful attribute, variable, and method names.
- Pay attention to the layout, and make sure your coding style is clear.
- **For questions 1 and 3 all public methods require `JavaDoc` comments. Other methods for these questions require regular comments.**
- Do *not* ignore compile-time warning messages—warnings like these usually indicate *errors* and you should make sure you eliminate them.

Question 1: A Non-Generic Pair Class. *(20/80 marks)*

Implement a non-generic class called `NonGenericPair`. Instances of the class should consist of two object references. The class should define getters, setters, and two constructors.[1] Instances of the class should be capable of comparing themselves for equality with other object references using the `equals( )` method. Instances should be able to convert themselves to `String`; the result of such conversions should clearly show the two object references which are owned by the instances.

Question 2: A Generic Pair Class. *(5/80 marks)*

Using the `NonGenericPair` class as your reference implementation, implement a *generic* class called `Pair`, which should depend on two type parameters. The `Pair` class is needed for Question 4. There is no need to compare members of the class for equality.

There is no need to provide comments.

Question 3: Interfaces. *(25/80 marks)*

Consider the following generic interface definitions.

```java
/**
 * Generic interface which defines an <code>apply</code> method
 * which applies a task to its argument.
 */
public interface Task<T> {
    /**
     * Apply this instance's task to an object argument.
     * @param object The object to which this task is applied.
     */
    public void apply( final T object );
}
```

```java
/**
 * Generic interface for collections which allows for a traversal
 * of the members of the collection and applying a task to each member
 * of the collection.
 */
public interface Traversable<T> extends Iterable<T> {
    /**
     * Apply a task to each member of this <code>Iterable</code> collection.
     * @param task The task which is applied to each individual
     *        member of this <code>Iterable</code> collection.
     */
    public void traverse( final Task<T> task );
}
```

Using the `Task` interface you may define a polymorphic `Task` object whose instance method `apply( )` applies a task to an object reference argument. E.g. consider the (polymorphic) `Task` objects `print1` and `print2`. Assume that the `apply( )` method of the `print1` object prints its argument once and that the `apply( )` method of the `print2` object prints its argument twice. Then `print1.apply( object )` prints `object` once and `print2.apply( object )` prints `object` twice.

The `Traversable` interface lets you define a polymorphic `Traversable` object which can traverse all members of an `Iterable` collection and apply a given task to each member of the collection. Furthermore, assume we have a `Traversable` object `traverser` which visits all objects in a list. Then `traverser.traverse( task1 )` prints each object in the list once and `traverser.traverse( task2 )` prints each object in the list twice.

For this question you will implement a concrete class which implements the `Traversable` interface. The name of the class should be `Traverser`. The class should have a constructor which takes one parameter, which should be a `Iterable` object reference whose members are traversed by the instance method `traverse( )` of the `Traverser` interface.

[1] You should be able to guess the signature of the constructors.

Question 4: Linked Lists. *(30/80 marks)*

In this question you will implement a class for linked lists. The list class implement the `Iterable` interface and should provide (1) a method for printing a list and (2) a method for sorting lists with the quicksort algorithm. You should use a similar representation as we used during the lectures, so the list class should depend on a `Link` class whose instances represent the links.

However, the implementation of the classes should be different. (The main differences will be in the `Link` class.) The following explains the details of the implementation.

- The instances of the `List` class should be capable of iterating over the *elements* in the list.
- The instances of the `Link` class should be capable of iterating over the *links* in the chain of `Link` elements.
- The `print( )` method in the `List` class should exploit the fact that the `List` class is `Iterable`.
- **All list traversals in the `Link` class must be implemented using the `Task` and the `Traversable` interfaces.** For example, this lets you implement quicksort's `partition( )` algorithm by traversing all the `Link` elements and by carrying out a `Task` for each element.
- You should use the `Pair` class from Question 2 to represent quicksort's partition.
- You should define a small `main` method which creates a list consisting of three integers, sorts the list and prints it before and after the sorting.

Remember that all attributes should be encapsulated. There is no need to provide comments.