Software Development (CS2514) Assignment 4

Generics (Due: 22 April. Marks: 5)

Learning Objectives

You will learn how to implement a class that depends on a generic type and how to create instances of the class for an actual type. You will implement a linked list class that uses the *visitor* design pattern to carry out common tasks. The visitor design pattern allows client classes to carry out tasks that use the data in the lists without knowing about the representation of the lists.

Main Details

For this assignment you will implement a generic linked list class. You should use the linked list representation that was studied in Lectures 13 & 14. You will use the *Visitor Design Pattern,* which is explained further on, to implement the following tasks: print the list, compute the length of the list, copy the list, and reverse the list. The following are the details.

- You should implement your linked lists in a class called `MyList`. The implementation should use a nested class for representing the `Link`s in the list.
- The visitor design pattern allows client classes of collections access to the data in these collections without disclosing any information about the implementation of the collection.
- For the purpose of this assignment the visitor pattern has two generic interfaces `Visitor` and `Visitable`.
- `Visitable` is implemented by the `MyList` class and client classes can see all data elements in the `MyList` instances by visiting them with a `Visitor` instance.
- The `Visitor` interface defines a single method called `show( final T data )`.
- The `Visitable` interface defines a single method called `visitAll( Visitor<T> visitor )`. When this method is called, the method visits all elements in the collection. The order of visiting is the same as the order of the collection.

  For each element `data` in the collection, the method calls `visitor.show( data )`, thereby showing `data` to the `visitor` instance without disclosing the implementation of the `MyList` class.
- For example, a client can now print all things in a `MyList<T>` instance, `list`, by creating a `Visitor<T>` instance `visitor` that prints a single `T` and by calling `list.visitAll( visitor )`.
- For example, with the following you may print the elements of a `MyList` instance called `list`, which has type `MyList<Integer>`.

```
final Visitor<Integer> visitor = new Visitor<Integer>( ) {
    @Override
```

```
    public void show( final Integer data ) {
        System.out.println( data );
    }
};
list.visitAll( visitor );
```

- **Important**: the `Visitor` instance only takes in a `T` data element as its argument. When a `Visitor` references a variable or attribute from outside the body of the `show( )` method, the variable or attribute must be constant (`final`). This poses a problem when you want to compute the length of the list because you cannot increment a `final` variable. Fortunately, there is an easy solution to this problem:
  - Define a generic `Holder` class that has an attribute that references (holds) an instance of a class;
  - Define a getter and a setter for this attribute;
  - You can now access `final Holder` elements from inside the `visit( )` method.
  - Using the getter and setter you can change the object that is held by the `Holder` instance.
- The main client class, is called `Main`. It should define the `main( )`. The `main( )` should create a `MyList<Integer>` with some `Integer`s in it and should then carry out the tasks of printing, computing the length, copying, and reversing the list. The `MyList` is properly encapsulated and doesn't provide an API for these tasks, so the `main( )` has to create `Visitor` instances that carry out these tasks.
- The `main( )` should not be too long, so it may be a good idea to create each `Visitor` by calling a method. Alternatively, you can define class constant `Visitor` attributes.
- The `MyList` class should have only two instance methods. The first instance method is for adding a member to the list. The second method is a method called `visitAll( )`.
- You are *not* allowed to import external classes and your implementation should not use arrays.

Hint: you can implement `copy( )` by calling `reverse( )` twice.

General Comments

Carefully read the submission guidelines before you submit the assignment.

Do not ignore compiler warnings: should you get them, it (almost always) means you have an error in your implementation, which may result in a client run-time error. Such errors should be fixed before deployment.

Like all other exercises, this is an exercise about implementing *maintainable* classes. You should always assume the specifications may change (slightly) and make sure your implementation can implement these changes with the minimum amount of effort.

Before you start implementing your classes, please make sure you understand the API. If you don't, you'll make your life much more difficult.

Submission Details

- Please remember that all classes, attributes, and public methods should be commented using a proper `JavaDoc` comment.
- The `JavaDoc` class comment should explain the purpose of the class; *not the purpose of the assignment.*

- Please provide your name and student ID as part of your class `JavaDoc`. You can use the `@author` tag for this:

```Java
@author Java Joe (ID 12345678)
```

- Use the CS2514 `moodle` site to upload your program as a single *.tgz* archive called *Lab-4.tgz* before 23.55pm, 22 April, 2018. To create the *.tgz* archive, do the following:
  - Create a directory `Lab-4` in your working directory.
  - Copy `MyList.java`, `Visitable.java`, `Visitor.java`, and `Main.java` into the directory. Do not copy any other files into the directory.
  - Run the command '`tar cvfz Lab-4.tgz Lab-4`' from your working directory. The option '`v`' makes `tar` very chatty: it should tell you exactly what is going into the `.tgz` archive. Make sure you check the `tar` output before submitting your archive.
  - Note that file names in `Unix` are case sensitive and should not contain spaces.
- Note that the format of your submission should be `.tgz`: do *not* submit `zip` files, do *not* submit `tar` files, do *not* submit `bzip` files, and do *not* submit `rar` files. If you do, it may not be possible to unzip your assignment.
- No marks shall be awarded for programs that do not compile.