Day 12 – Abstraction, Stacks, Collections, and the Stack Interface

Chapter 02, 2.1 – Abstraction, Review Problem(s): ***3, 4, 6, 8***

Chapter 02, 2.2 – Stack, Review Problem(s): ***12, 14***

Chapter 02, 2.3 – Collections

Chapter 02, 2.4 – Stack Interface, Review Problem(s): ***19, 20***

Time for some important vocabulary:

- ***Abstraction*** – A model of a system that includes only the details essential to the perspective of the viewer of the system
- ***Information hiding*** – The practice of hiding details within a module with the goal of controlling access to the details from the rest of the system
- ***Data abstraction*** – The separation of a data type's logical properties from its implementation
- ***Abstract data type (ADT)*** – A data type whose properties (domain and operations) are specified independently of any particular implementation

There are different levels to consider and work with ADTs:

- ***Application*** (or user or client or external) level: We use the ADT to solve a problem.
- ***Abstract*** (or logical) level: Provides an abstract view of the data values (the domain) and the set of operations to manipulate them.
- ***Implementation*** (or concrete or internal) level: Provides a specific representation of the structure to hold the data and the implementation of the operations.

We'll learn these terms by using them. Our first ADT will be a stack.

We first consider the stack at the application level, and then proceed with the abstract level and finally the implementation. There are multiple ways to implement a stack, and we will consider a few.

Defining the stack at the application level means we are just defining the ***Interface***, and the methods are ***Java Abstract Methods***, which only have descriptions of the parameters. We can use an interface to formally specify the logical level of an ADT, it provides a template for classes to fill, and then a separate class implements it.

Previously we considered Inheritance based Polymorphism. But in using interfaces, we now consider ***Interface Polymorphism***, where an object variable can reference objects of different classes at different times. So, a variable declared to be of an interface "type" T can hold a reference to any object of class that implements that interface.

And finally, a stack is an example of a ***Collection ADT***. A Collection is an object that holds other objects. Typically we are interested in inserting, removing, and obtaining the elements of a collection.

How can we implement stacks that hold different types of elements?

- Implement a separate stack class for each element type, but this approach is too redundant and not generally useful.
- Implement a stack class that can hold elements of class Object, but this approach requires extra effort to keep track of, and indicate, exactly what is in the stack.
- Implement a stack class that can hold elements of a particular interface, but this approach could work in certain specific situations but it is not a general solution.
- ***We will implement a Generic stack, it's implemented once and the type of element will be indicated at instantiation.***