Prepared by Linan Qiu <lq2137@columbia.edu>, adapted from Open Data Structures (opendatastructures.org) for CS3134 Spring 2016.

List Abstract Data Type

List as an Abstract Data Type

An abstract data type is really an **interface**. It specifies what something should do, but never really goes into detail about how it should be done. For example, a List is an abstract data type. We never really specify how exactly to implement it (in fact we will see that there many ways to implement lists. In this chapter, we talk about using arrays and using a linked list). Instead, we just want lists have the following functionality:

- Store a, well, list of items
- Useres can run through this list of items one by one (**iterate** through the items)
- Add and remove items from this list
- Retrieve a specific item from the list either by value (in a list of fruits, get me "Apple") or index (get me the 4th fruit in the list)
- Expand and shrink as needed

Arrays?

These functionalities *seem* to be satisfied by the array object in Java already. Let's say we create a new array like this:

```
int[] array = new int[5];
```

Can we store a list of items? Sure!

```
array[0] = 1;
array[1] = 2;
array[2] = 4;
array[3] = 8;
array[4] = 16;
```

Can we run through the list one by one? Yup! (For those of you who insist that we need an iterator, bear with me for a moment.)

```java
for (int i : array) {
  // do something with i eg.
  System.out.println(i);
}
```

Can we add and remove items? Well, not perfectly but sort of.

```java
// remove an item
// let's say Integer.MIN_VALUE is the placeholder for empty
array[4] = Integer.MIN_VALUE;

// add an item
array[4] = 42;
```

This process is not perfect, since we have to know the exact index of where we are adding, and we cannot add beyond 5 elements. In fact, this kind of outlines our total failure at the last requirement – arrays cannot expand or shrink.

> In fact, what you're doing when you're declaring an array is this: you're telling the computing to set aside a certain amount of memory in the RAM for you. That RAM is equivalent to the size of the elements in the array, so its a contiguous block of memory. Then, if you want to grow the array, you would have to declare a new (bigger) array and copy the old array's contents into the new one. You cannot just keep writing beyond the memory range occupied by the old array – there may be something else there (an object, another program etc), and you'd be doing something really really bad. Older languages like C allows you to write beyond the `length - 1` of an array, and this often causes problems for programmers. Java will throw you a `IndexOutOfBoundsException` preventing you from doing this.

This shows that arrays just by themselves are an insufficient implementation of the list ADT.

By the way, an implementation of an ADT is called a **data structure**. That's why this course is called data structures hehehe.