1. ArrayList Class :: Motivation

Write a program that reads a text file (*ints-in.txt*), containing integers one per line, and writes the integers that were read to another text file (*ints-out.txt*) but in reverse order. We do not know anything more about the contents of *ints-in.txt* other than the number of integers in *ints-in.txt* is more than 0 and less than or equal to one million.

ints-in.txt	***ints-out.txt***
7	12
3	-16
5	4
8	...
...	8
4	5
-16	3
12	7

What would be helpful would be a data structure—similar to a 1D array—but that can grow in size as needed. That need can be met by the *java.util.ArrayList* class which offers two significant advantages:

- *ArrayList*s can grow and shrink as needed.
- The *ArrayList* class provides methods for common tasks such as inserting and removing elements.

1. ArrayList Class :: Declaring an *ArrayList* Object

The *ArrayList* class is a **generic class** meaning:

- It is designed to store objects of *any* class.
- We must provide the class of objects being stored as a **type parameter** during declaration.

The syntax for declaring an *ArrayList* object is:

```
import java.util.ArrayList;
...
ArrayList<class> name = new ArrayList<class>();
```

In Java 7, *class* does not have to be specified twice (the angle brackes <> are referred to as the **diamond operator**). Example: create an *ArrayList* where each object is a *String*:

```
ArrayList<String> names = new ArrayList<>();
```

It is important to note that the data type within the diamond operator must be a **class**, i.e., it cannot be one of the **primitive** data types (int, double, char, boolean, etc).

```
ArrayList<int> numbers = new ArrayList<>();  // Syntax error! int is not a class.
```

We will discuss shortly how to create an ArrayList object that stores **int**s and **double**s.

1. ArrayList Class :: Adding Elements

An *ArrayList* is empty and has **size** 0 when constructed. Elements are added to an *ArrayList* using the **boolean *add*(*E e*)** method which adds element *e* of class *E* to end of the ArrayList, i.e., this is an **append** operation. *add*() returns **true** if the element was successfully added and **false** if it was not. Examples,

```
ArrayList<String> names = new ArrayList<>();
names.add("Emily");
names.add("Bob");
names.add("Cindy");
```

1. ArrayList Class :: *size*(), *get*(), *set*() Methods

The size of an *ArrayList* may be obtained by calling the **int *size***() method,

```
int n = names.size();  // Assigns 3 to n.
```

Elements are accessed by calling the ***E get***(**int *index***) method:

```
String s = names.get(0);  // Assigns "Emily" to s.
```

Note that the elements are numbered starting at 0 and the last element is at *names.size*() - 1. To change an element call the ***E set***(**int *index*, *E e***) method:

```
String s = names.set(2, "Carolyn");
```