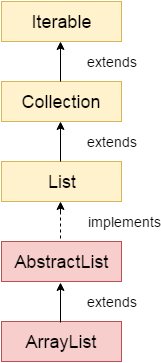
# Java ArrayList



* Java ArrayList class uses a dynamic array for storing the elements.
* It is like an array, but there is no size limit.
* We can add or remove elements anytime.
* So, it is much more flexible than the traditional array.
* It is found in the java.util package.
* It is like the Vector in C++.
* The ArrayList in Java can have the duplicate elements also.
* It implements the List interface so we can use all the methods of the List interface here.
* The ArrayList maintains the insertion order internally.
* It inherits the AbstractList class and implements [List interface](https://www.javatpoint.com/java-list).
* Java ArrayList class can contain duplicate elements.
* Java ArrayList class maintains insertion order.
* Java ArrayList class is non [synchronized](https://www.javatpoint.com/synchronization-in-java).
* Java ArrayList allows random access because the array works on an index basis.
* In ArrayList, manipulation is a little bit slower than the LinkedList in Java because a lot of shifting needs to occur if any element is removed from the array list.
* We can not create an array list of the primitive types, such as int, float, char, etc. It is required to use the required wrapper class in such cases. For exampl
* Java ArrayList gets initialized by the size. The size is dynamic in the array list, which varies according to the elements getting added or removed from the list.

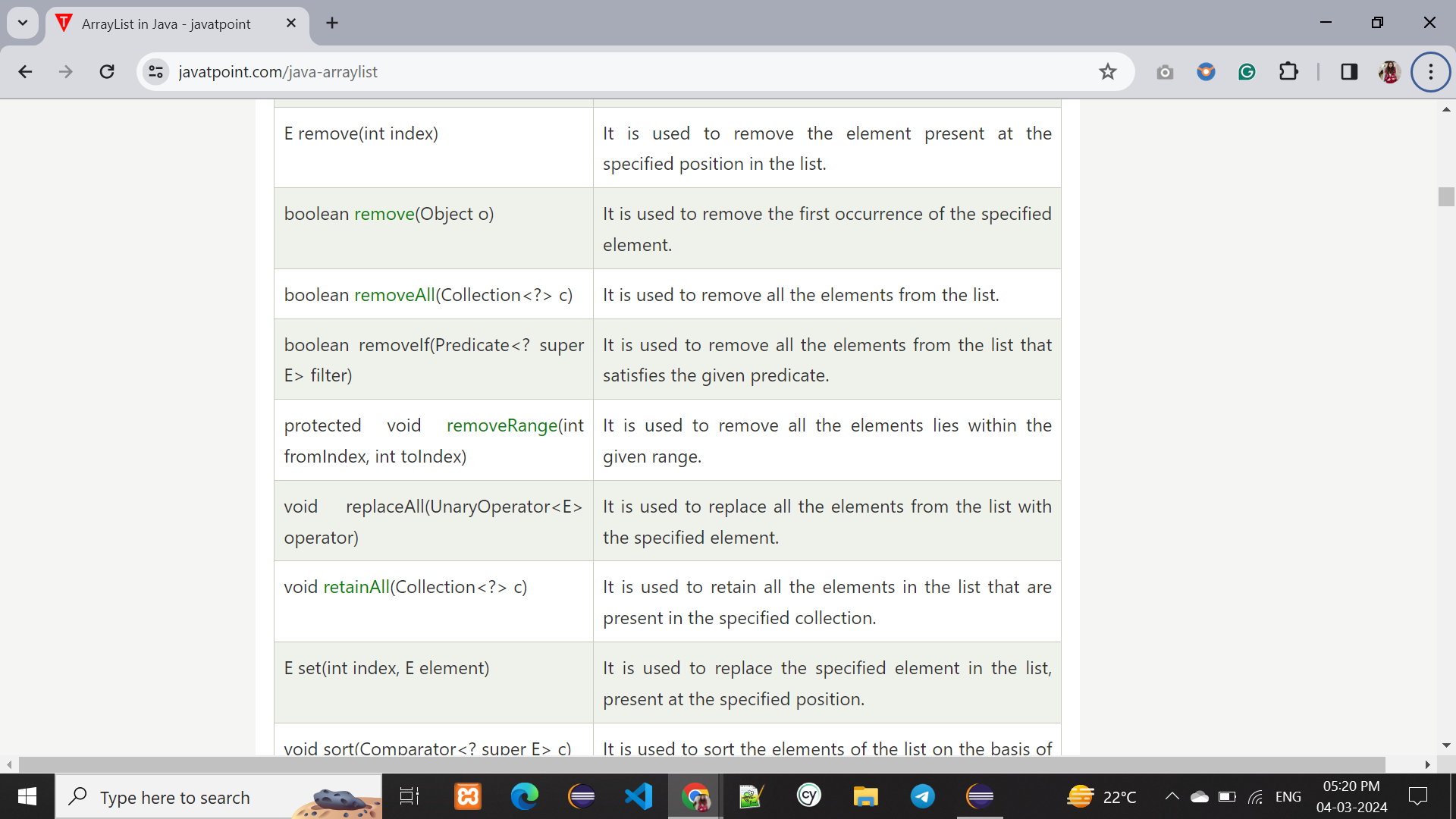
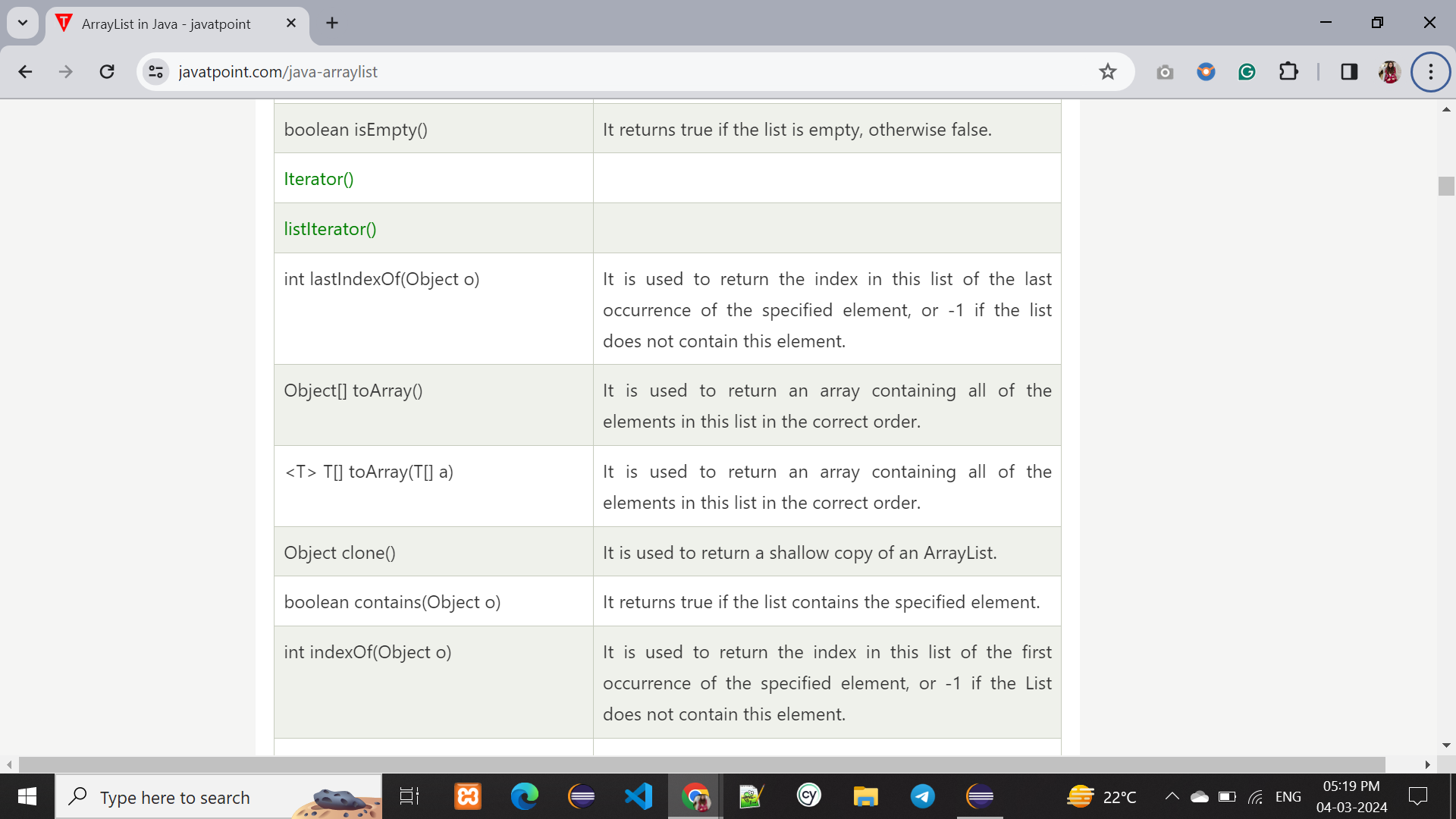
### ArrayList class declaration

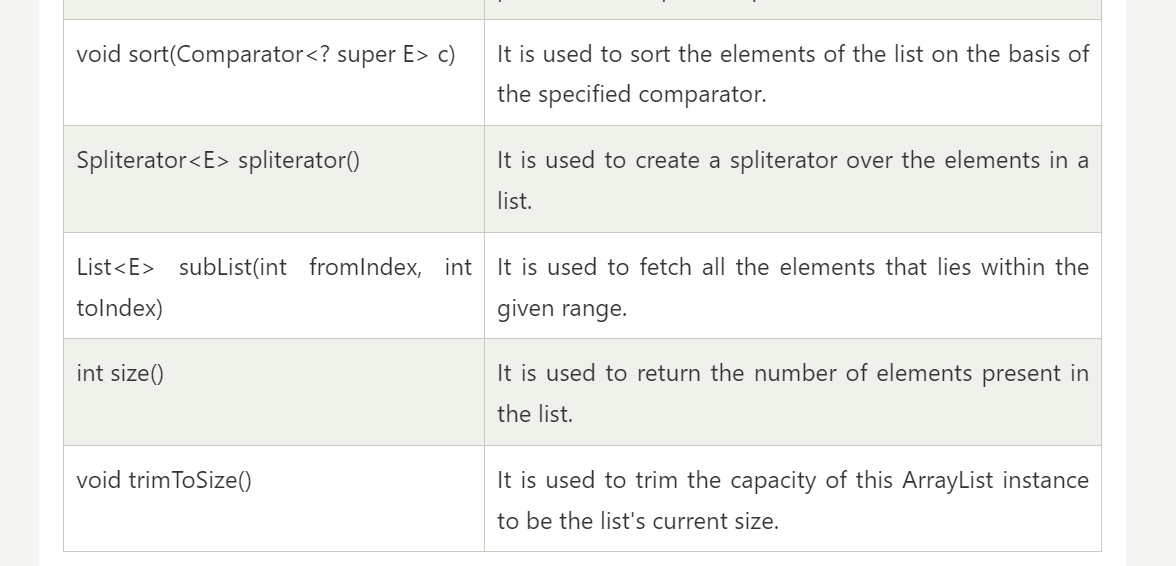
**public** **class** ArrayList<E> **extends** AbstractList<E> **implements** List<E>, RandomAccess, Cloneable, Serializable



### Methods of ArrayList







### Java Non-generic Vs. Generic Collection

Java collection framework was non-generic before JDK 1.5. Since 1.5, it is generic.

Java new generic collection allows you to have only one type of object in a collection. Now it is type-safe, so typecasting is not required at runtime.

Let's see the old non-generic example of creating a Java collection

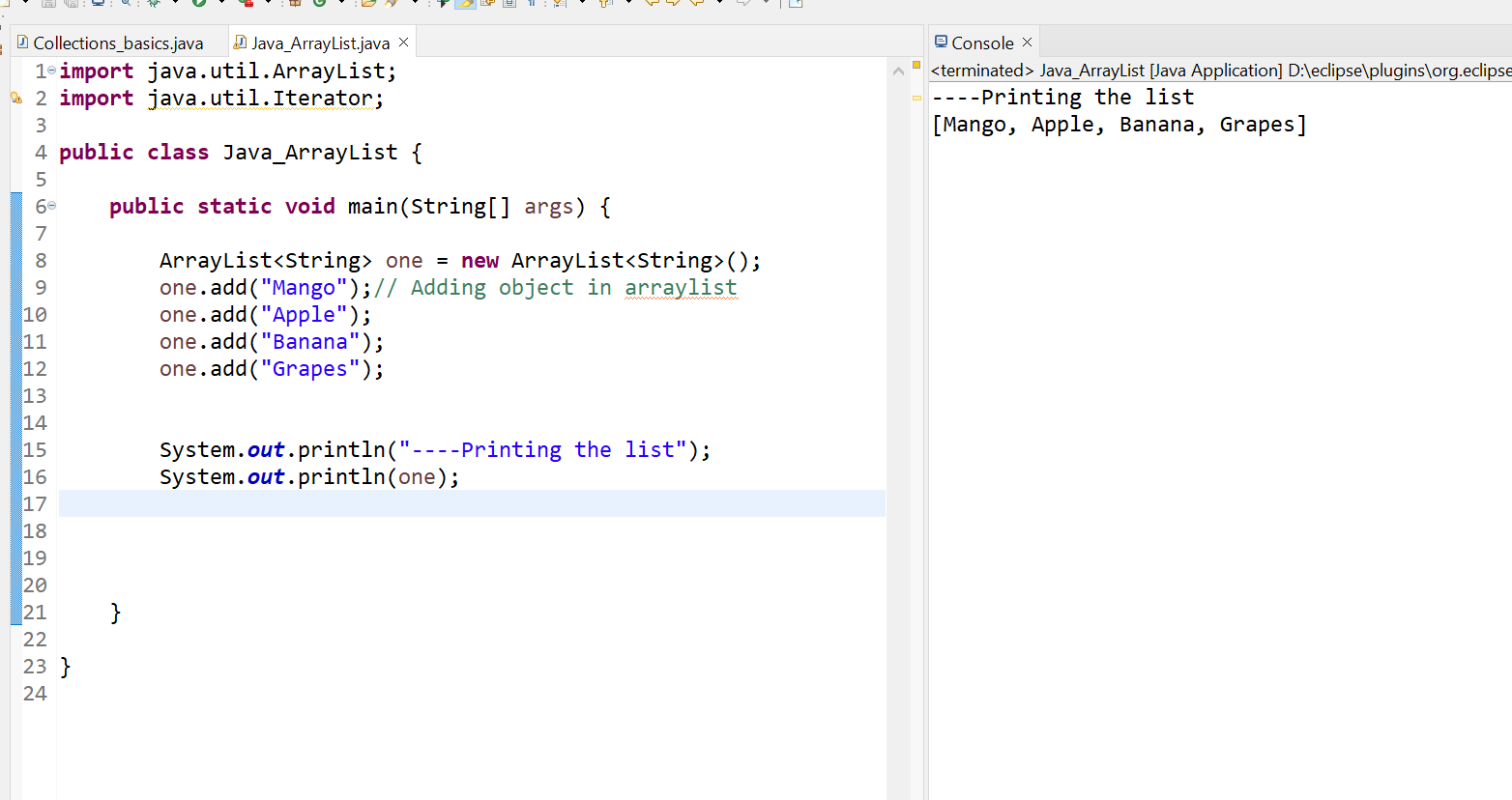
1. ArrayList list=**new** ArrayList();//creating old non-generic arraylist

Let's see the new generic example of creating java collection.

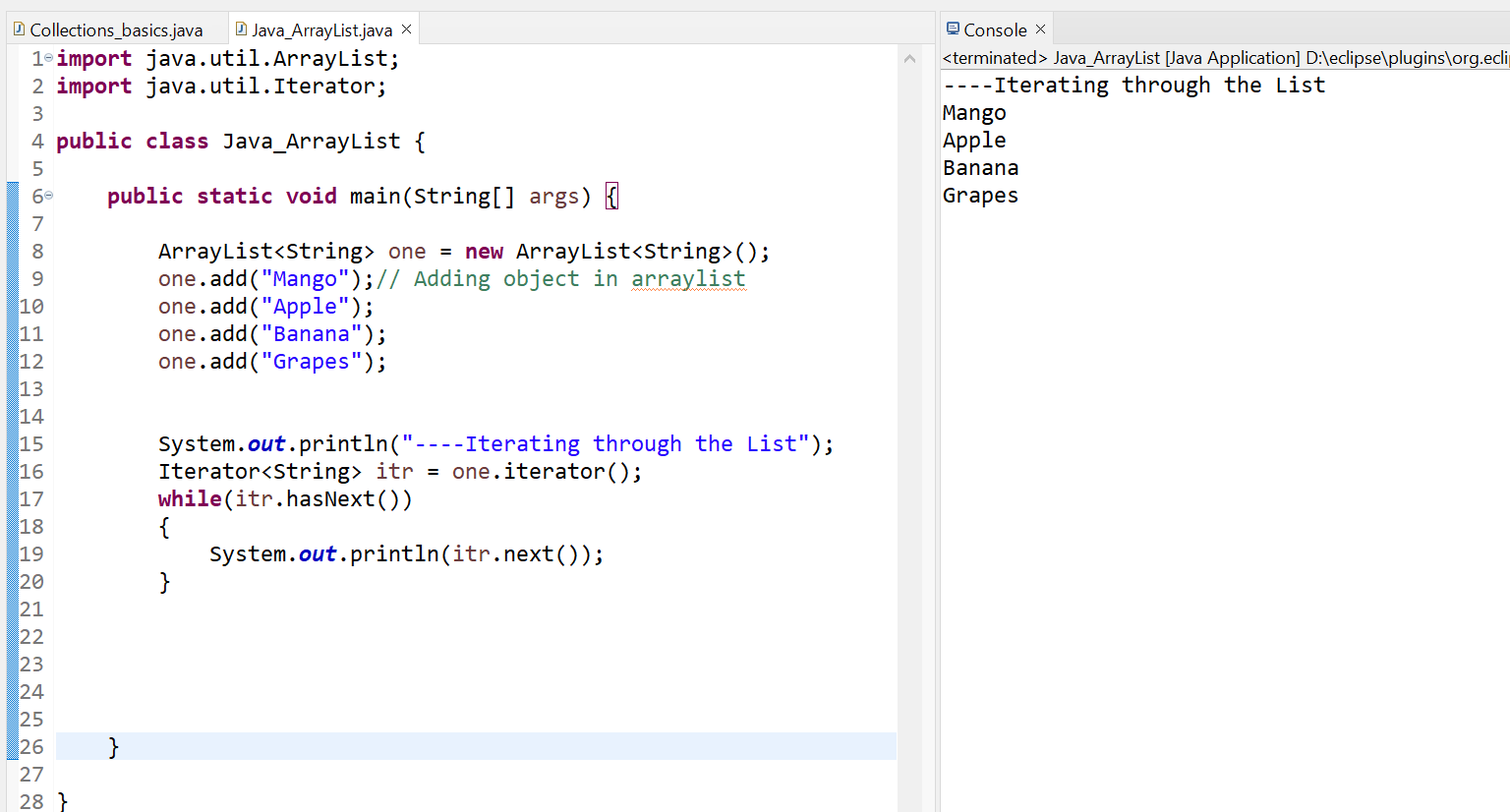
1. ArrayList<String> list=**new** ArrayList<String>();//creating new generic arraylist

In a generic collection, we specify the type in angular braces. Now ArrayList is forced to have the only specified type of object in it. If you try to add another type of object, it gives a *compile-time error*.

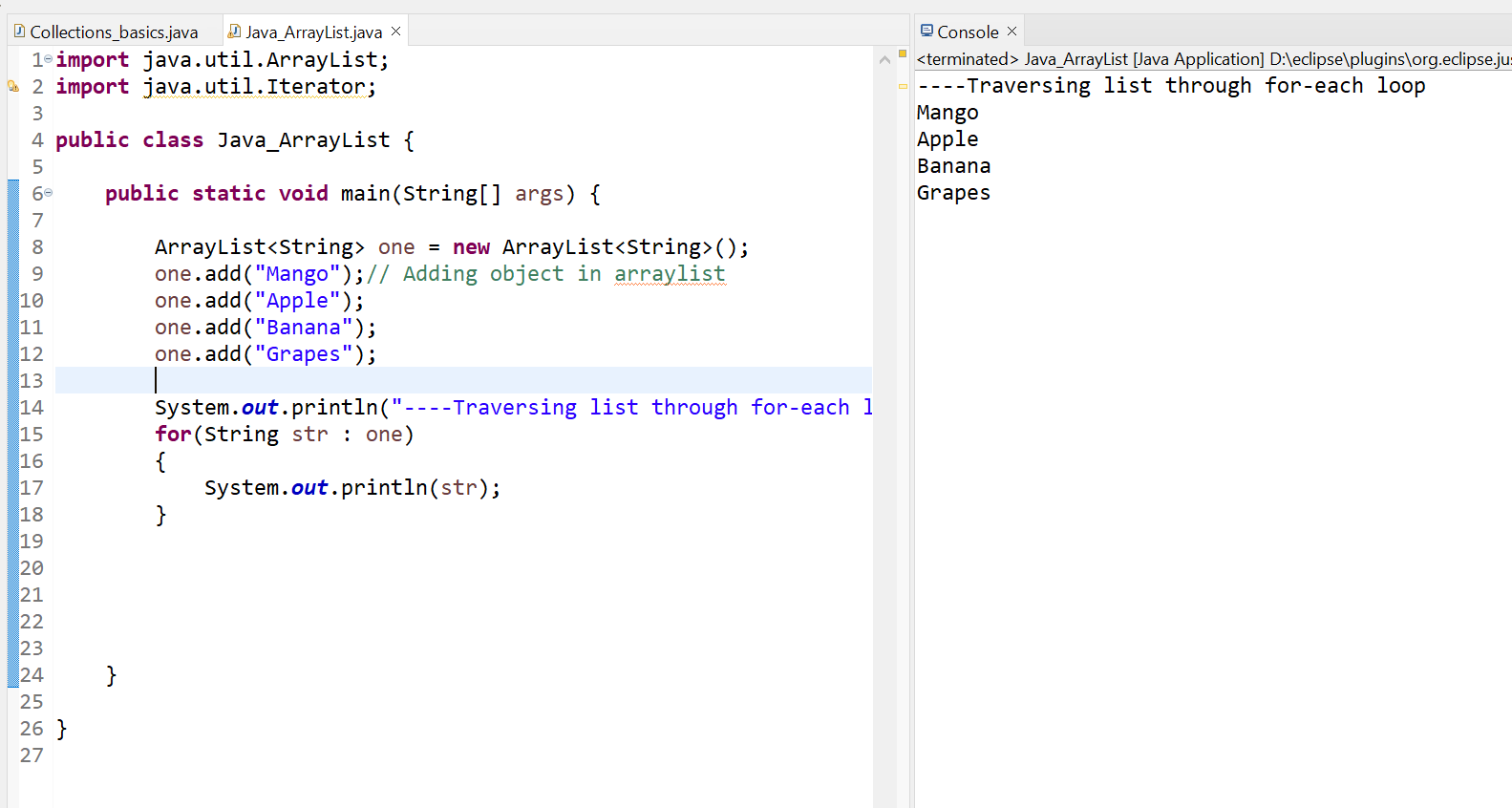
### Java ArrayList Example



### Iterating ArrayList using Iterator

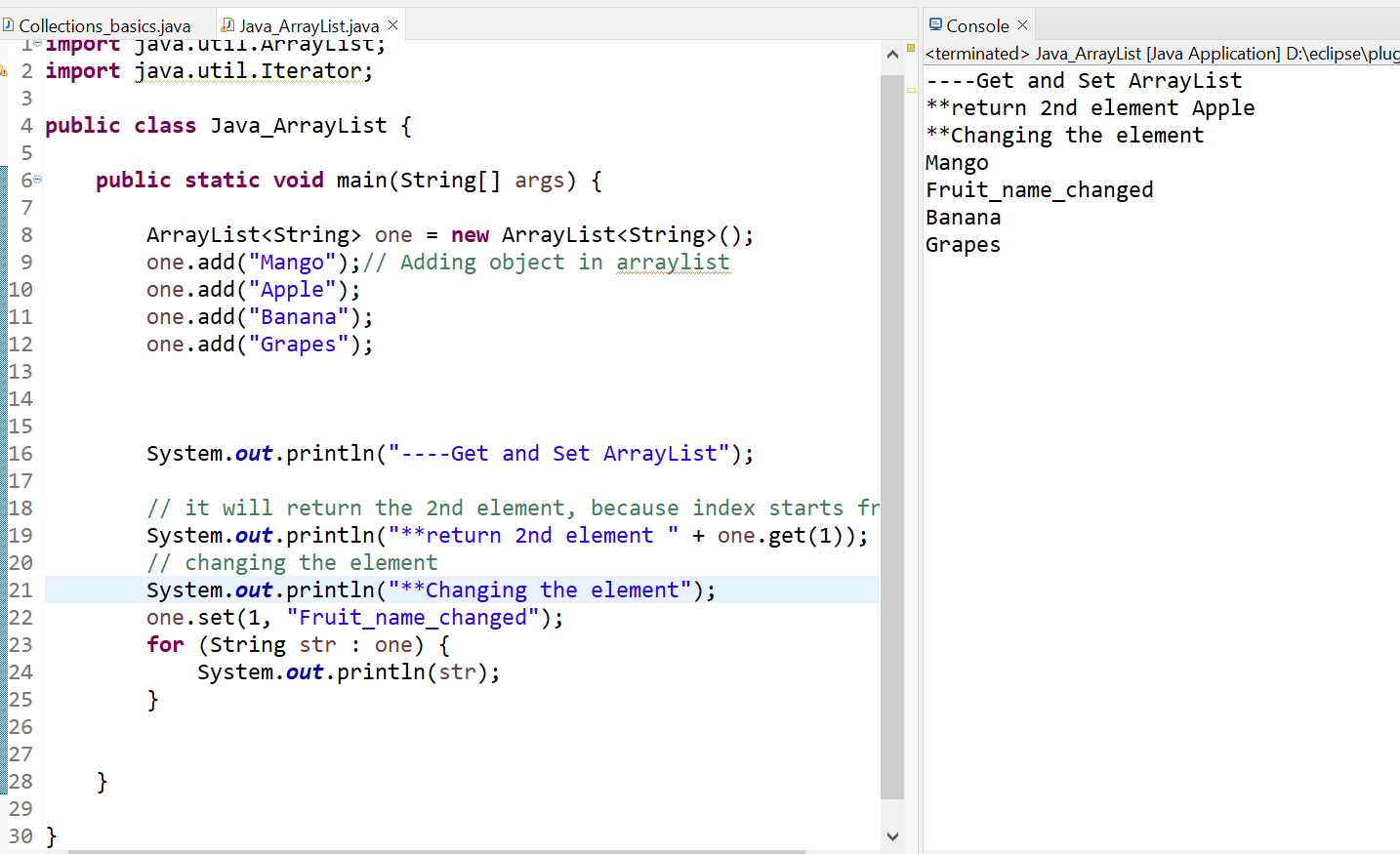


### Iterating ArrayList using For-each loop



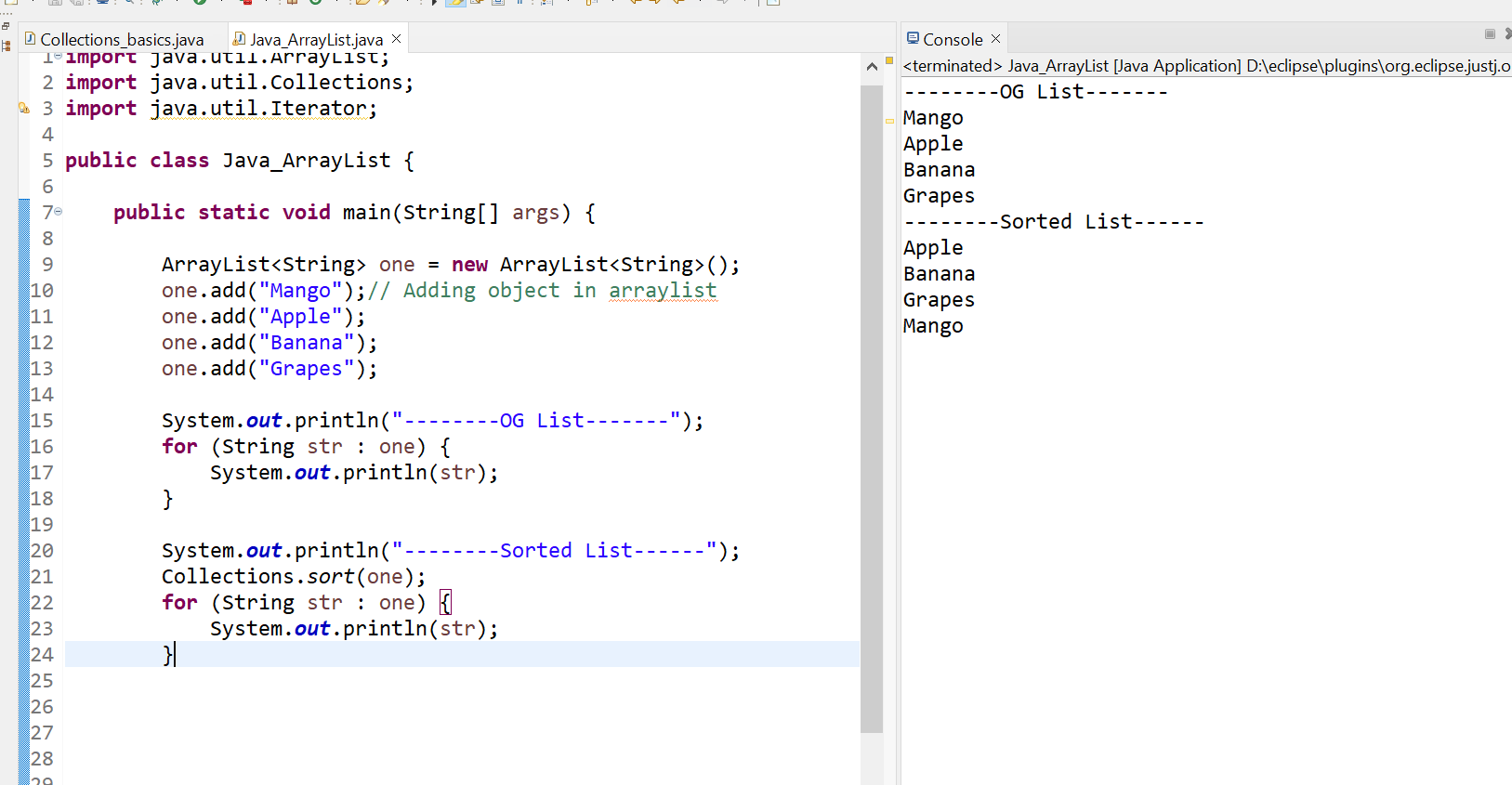
### Get and Set ArrayList

The *get() method* returns the element at the specified index, whereas the *set() method* changes the element.



### How to Sort ArrayList

The *java.util* package provides a utility class **Collections**, which has the static method sort(). Using the **Collections.sort()** method, we can easily sort the ArrayList.



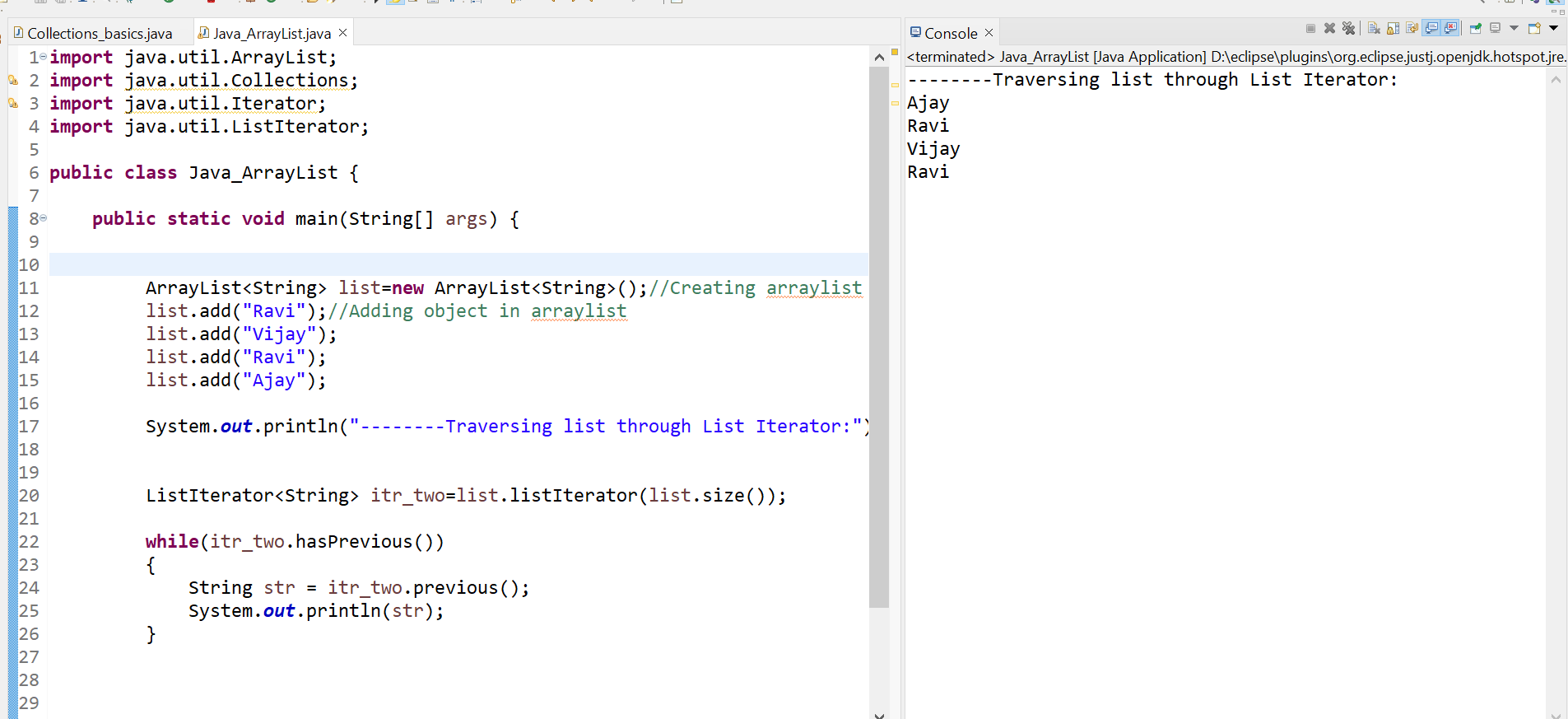
### Ways to iterate the elements of the collection in Java

There are various ways to traverse the collection elements:

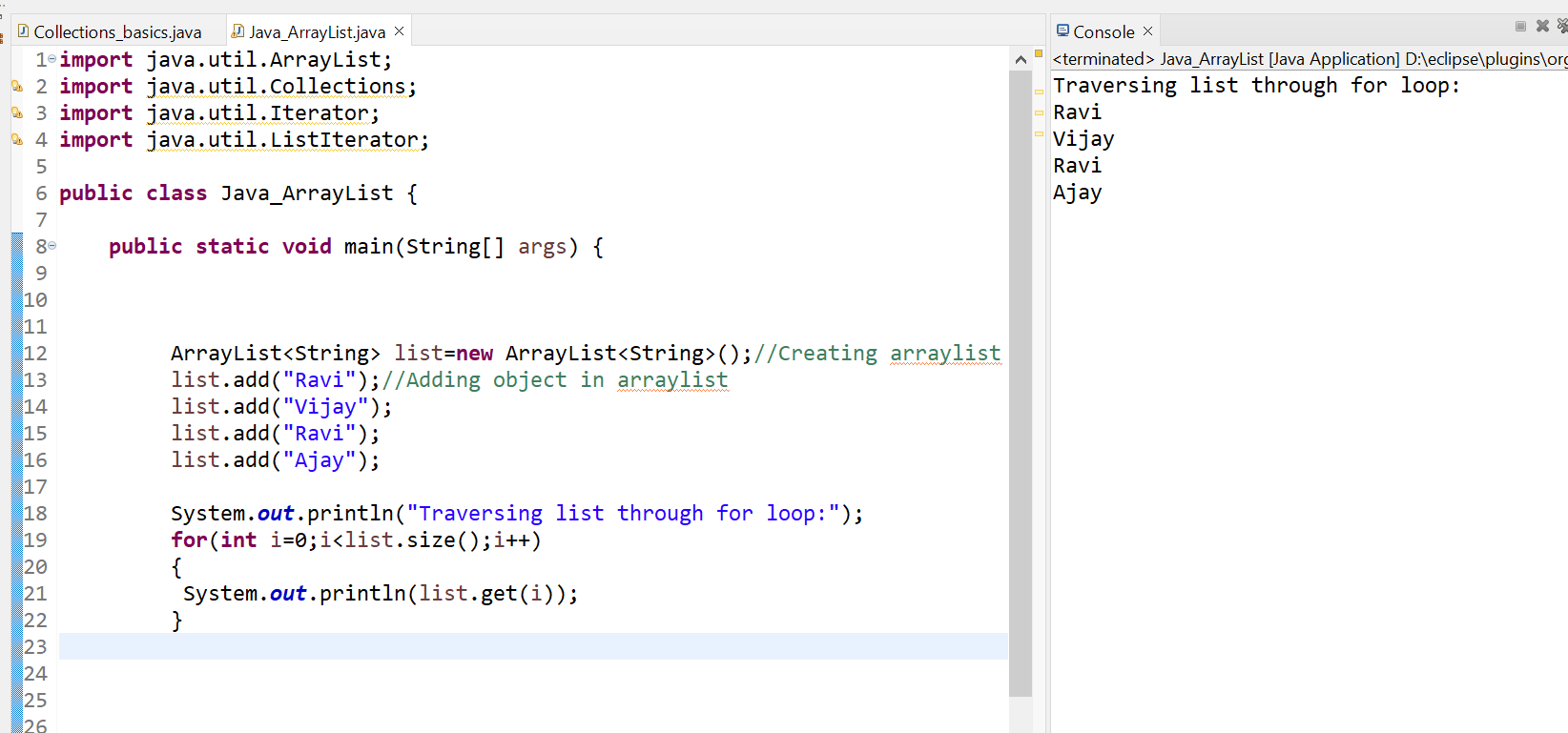
1. By Iterator interface.
2. By for-each loop.
3. By ListIterator interface.
4. By for loop.
5. By forEach() method.
6. By forEachRemaining() method.

Some of the methods are die in obove examples , for rest :

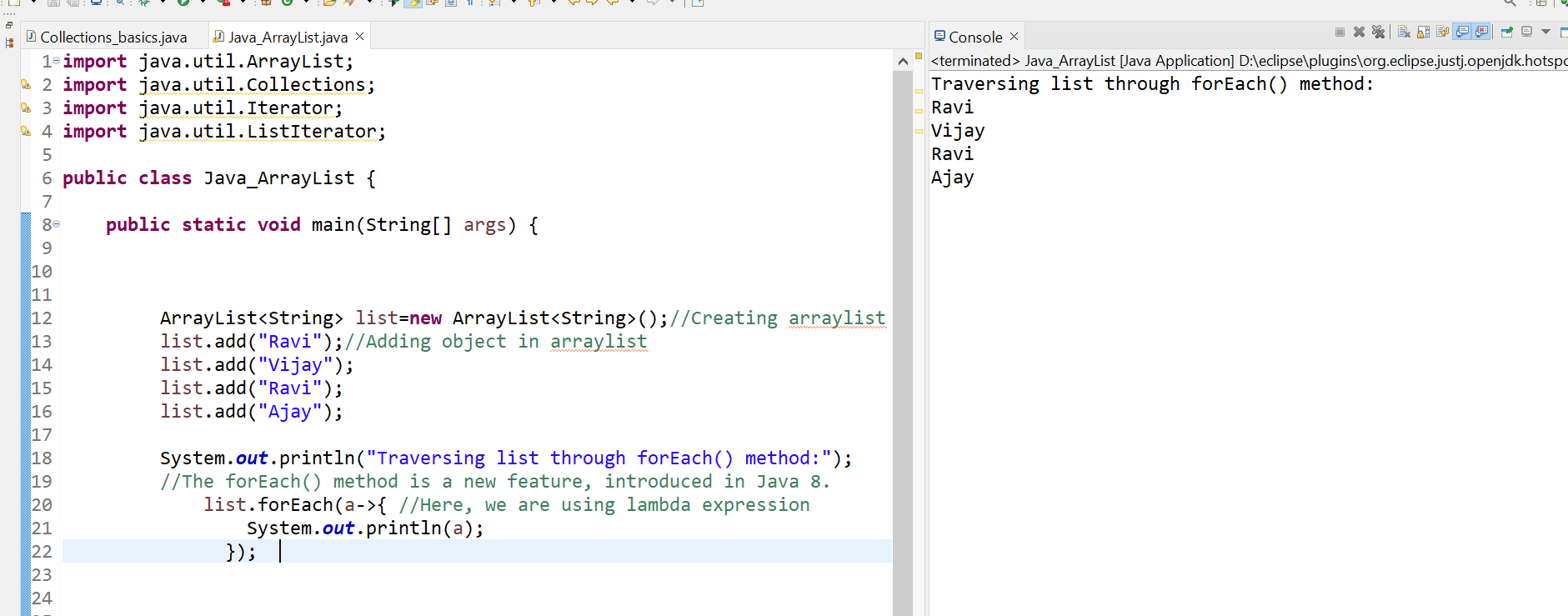
### Iterating Collection through remaining ways



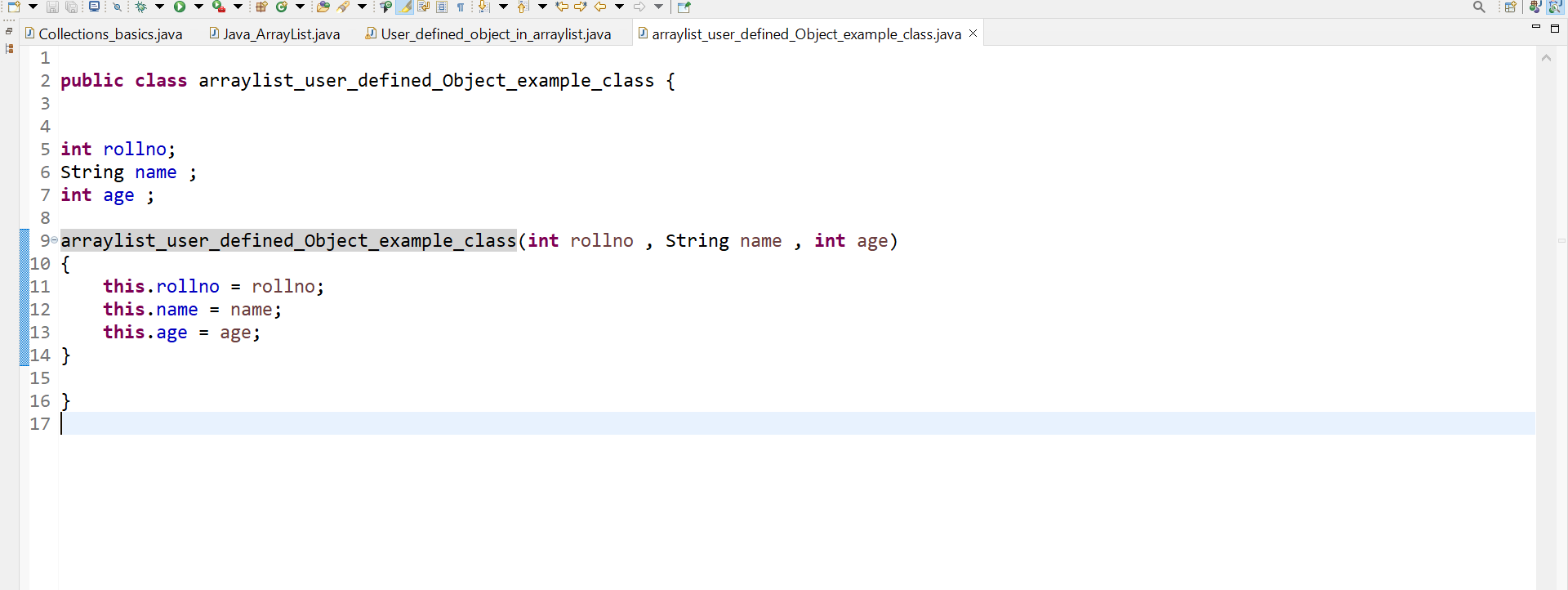
**Traversing list through for loop**

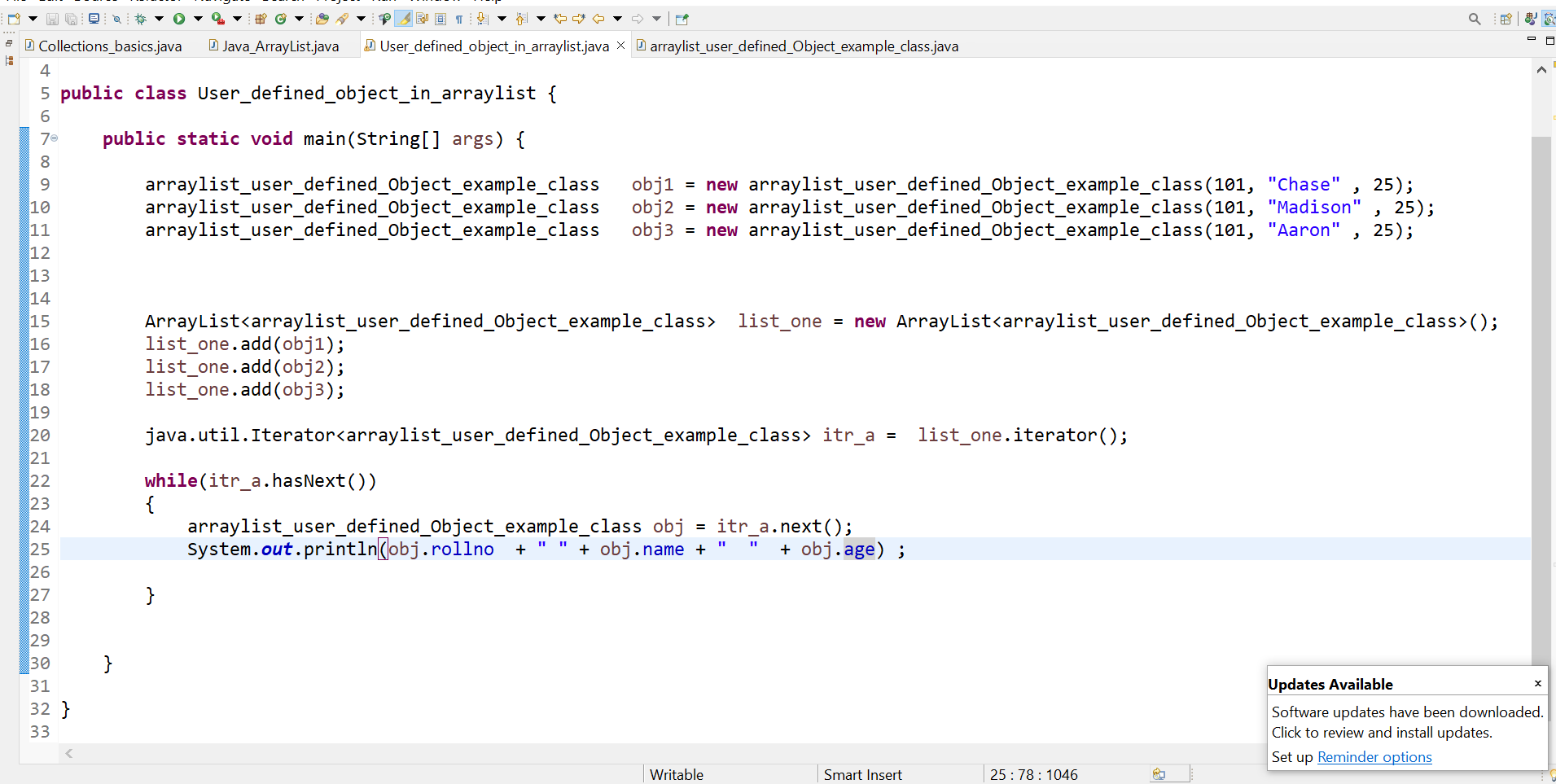
****

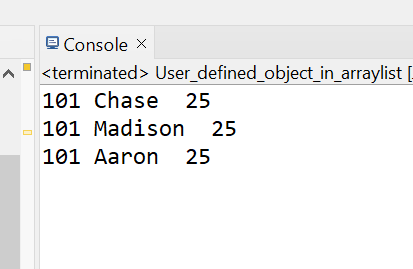
**Traversing list through forEach() method:**

****

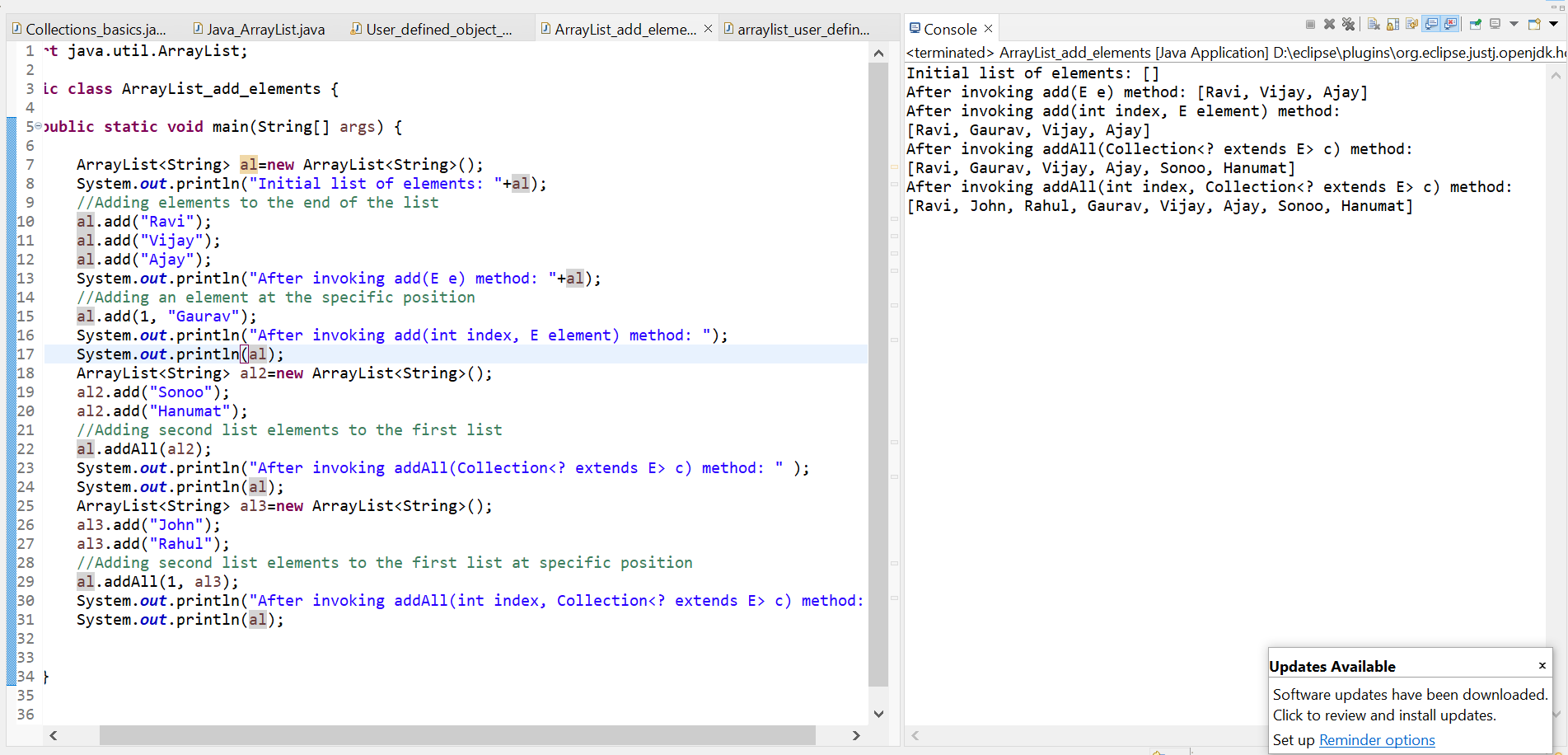
### User-defined class objects in Java ArrayList

****

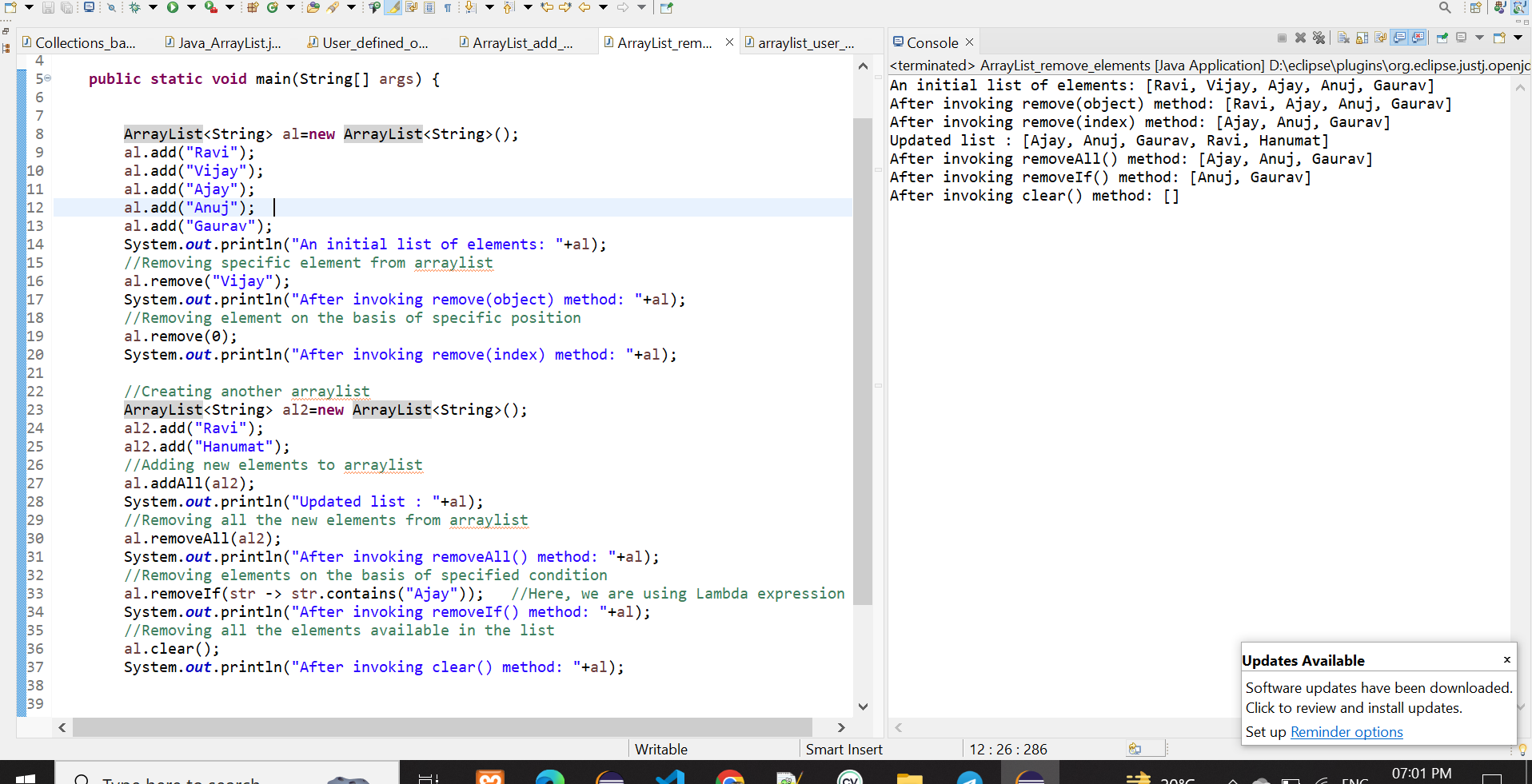
****

****

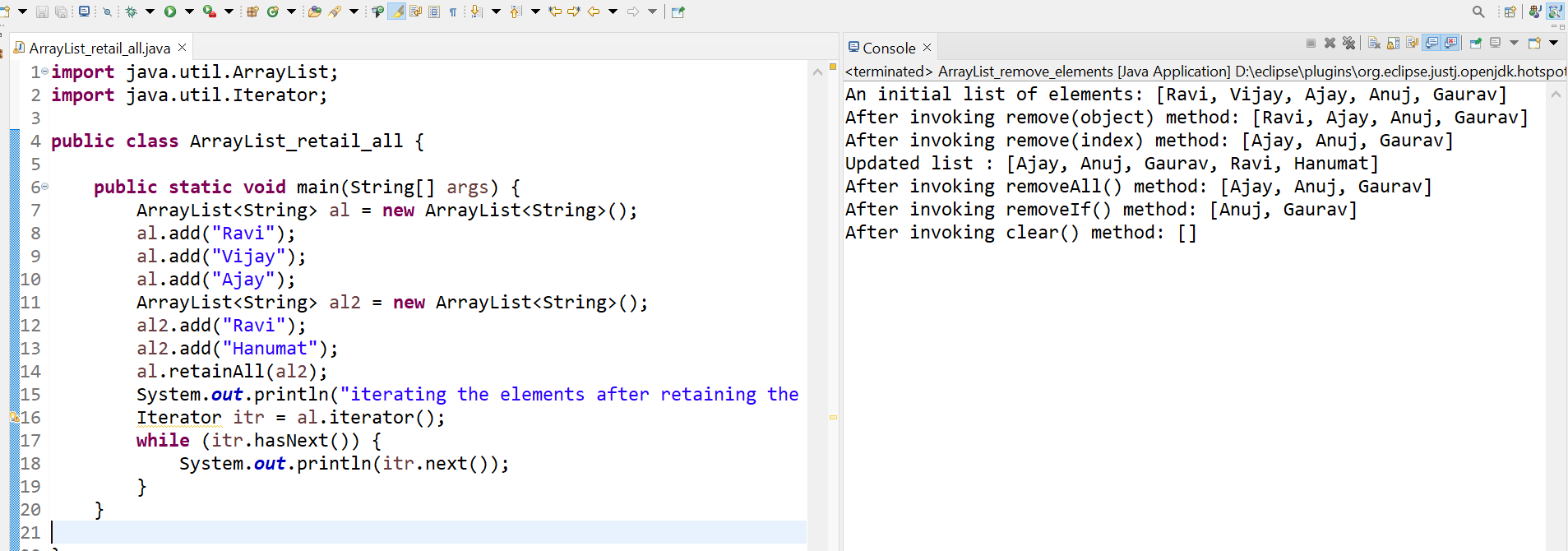
### Java ArrayList example to add elements

****

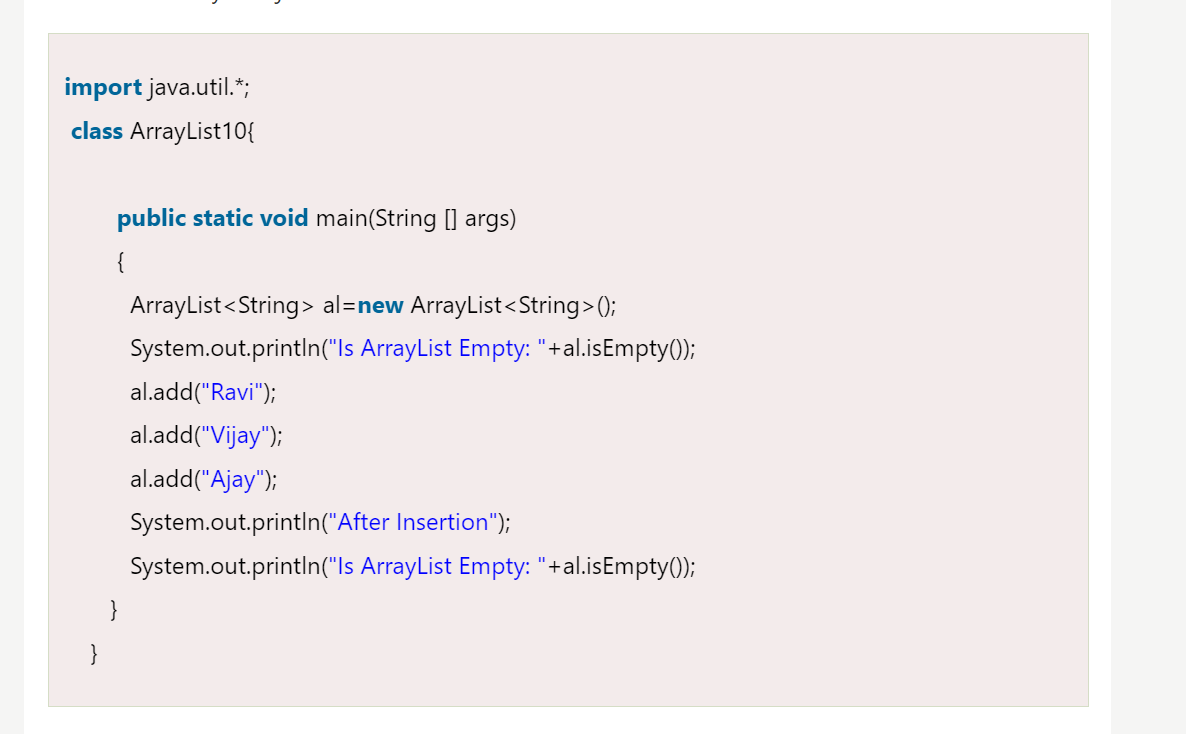
### Java ArrayList example to remove elements

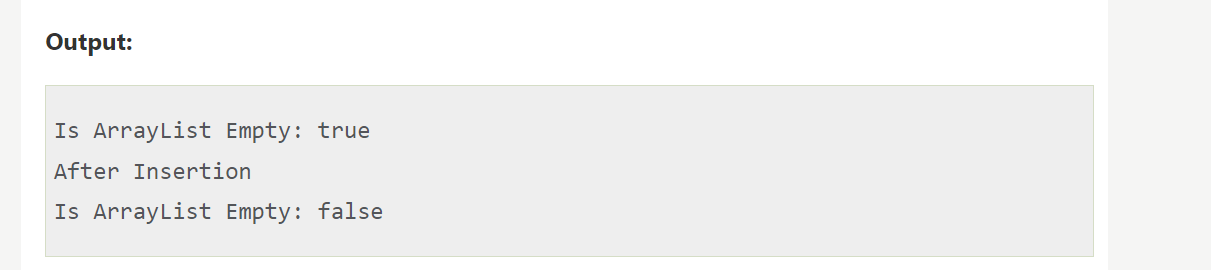
****

### Java ArrayList example of retainAll() method

****

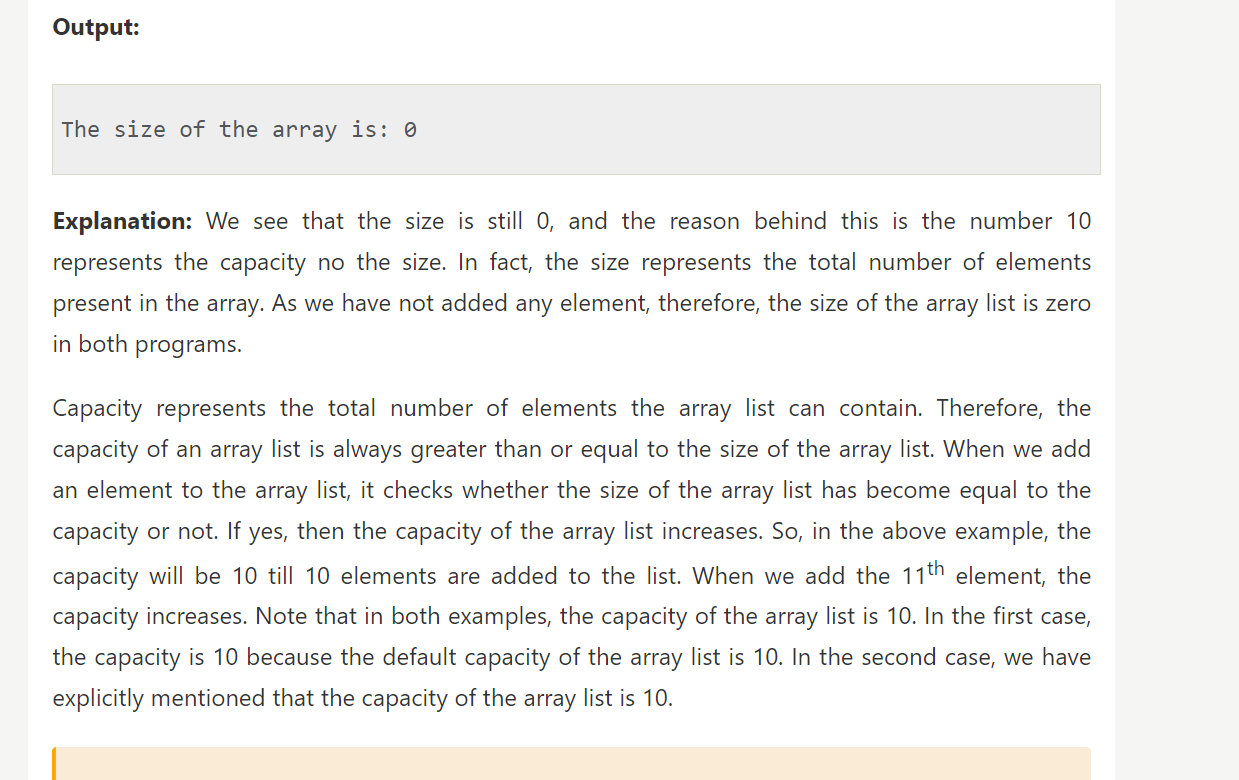
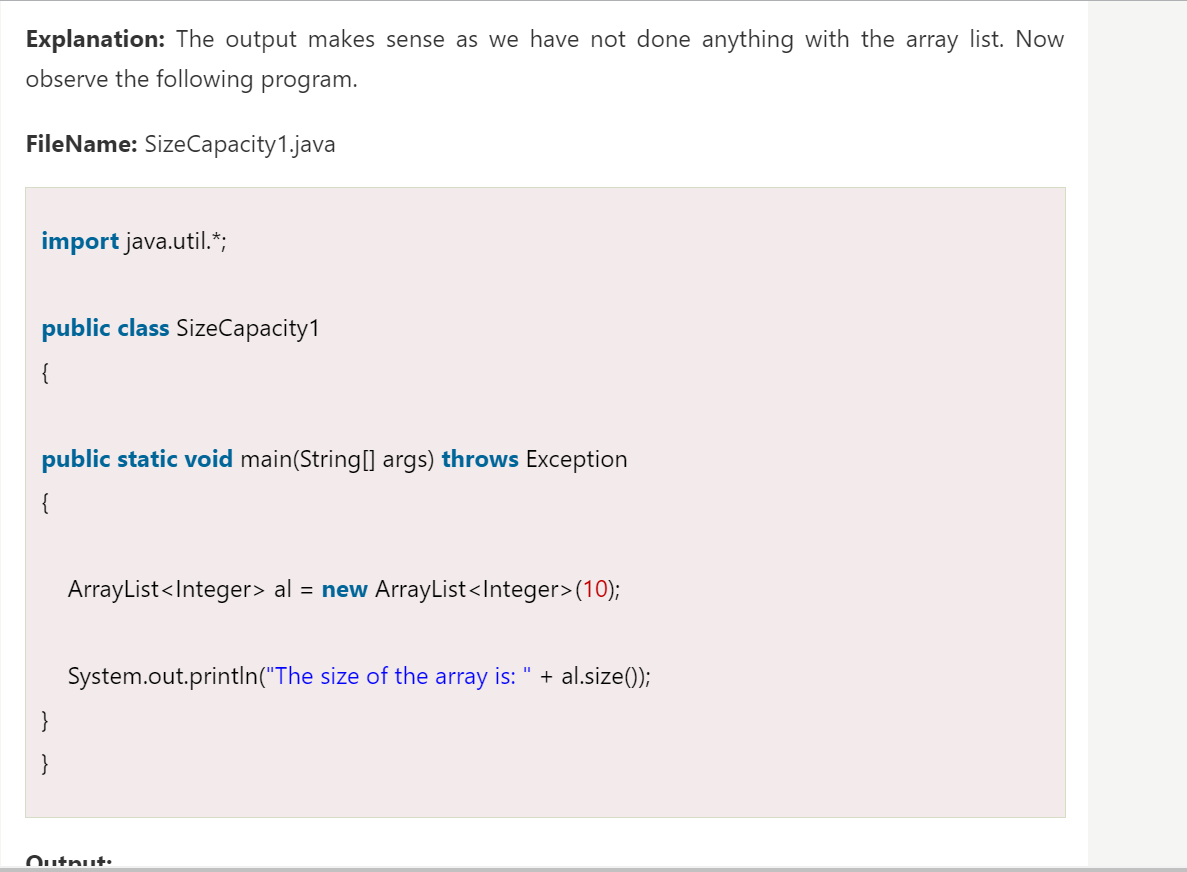
### Java ArrayList example of isEmpty() method

****

****

### Size and Capacity of an ArrayList

**Size and capacity of an array list are the two terms that beginners find confusing. Let's understand it in this section with the help of some examples. Consider the following code snippet.**

****Note: There is no any standard method to tell how the capacity increases in the array list. In fact, the way the capacity increases vary from one GDK version to the other version. Therefore, it is required to check the way capacity increases code is implemented in the GDK. There is no any pre-defined method in the ArrayList class that returns the capacity of the array list. Therefore, for better understanding, use the capacity() method of the Vector class. The logic of the size and the capacity is the same in the ArrayList class and the Vector class.