How to Join two ArrayList<E>?

This question is part of [Java Collections Framework](http://data-structure-learning.blogspot.com/2015/05/java-collections-part-1.html). Java Collections provides [various interfaces](http://data-structure-learning.blogspot.com/2015/05/java-collections-part-2-interfaces.html) and its concrete implementations that we use every day.

Now let us see how to join two [ArrayList<E>](http://data-structure-learning.blogspot.com/2015/05/java-collections-part-5list-interface.html). ArrayList<E> is a class that provided better flexibility over array. ArrayList<E> class extends AbstractList<E> class and implements List<E> interface. ArrayList<E> class is concrete implementation of [List<E>](http://data-structure-learning.blogspot.com/2015/05/java-collections-part-5list-interface.html) interface. Whereas [AbstractList<E>](http://data-structure-learning.blogspot.com/2015/05/java-collections-part-11-abstractlist.html) is abstract implementation of List<E> interface.

List<E> interface has a method

**boolean** addAll(Collection<? **extends** E> c);

This method is used to append all the elements to the specified collection in order of the elements returned by [iterator](http://data-structure-learning.blogspot.com/2015/05/java-collections-part-8-iterator.html).

We will use this method to add all the elements from one list to another list.

So let us just populate one list with list of computer languages and other list with databases.

Then we will use addAll(Collection<? extends E> c) method to append all the elements from both the lists.

Below is method to populate list of different languages.

**public** **static** List<String> populateLanguages() {

List<String> languages = **new** ArrayList<String>();

languages.add("Java");

languages.add("JavaScript");

languages.add("C#");

languages.add("Python");

**return** languages;

}

Below is method to populate list of databases.

**public** **static** List<String> populateDatabases() {

List<String> databases = **new** ArrayList<String>();

databases.add("MySql");

databases.add("SqlLite");

databases.add("MongoDB");

databases.add("Neo4j");

databases.add("Oracle");

**return** databases;

}

Now we define a new method addAllDemo(..) that will take both of the above returned lists as arguments. Then we will define a new list that will use addAll method to append all the elements from list1 and then from list2.

**public** **static** List<String> addAllDemo(List<String> languages, List<String> databases) {

List<String> languagesAndDatabases = **new** ArrayList<String>();

languagesAndDatabases.addAll(languages);

languagesAndDatabases.addAll(databases);

**return** languagesAndDatabases;

}

We can also do this operation by lambda operator.

forEach() method is defined as default method in Iterable<E> interface. Let’s see the hierarchy here. List<E> interface extends [Collection<E> interface](http://data-structure-learning.blogspot.com/2015/05/java-collections-part-4collection.html) and Collection<E> interface extends Iterable<E> interface.

So we can directly use forEach method.

Below is the code for joining two lists by Lambda Operator.

**public** **static** List<String> addListByLambda(List<String> languages, List<String> databases){

List<String> languagesAndDatabases = **new** ArrayList<String>();

languages.forEach(str -> languagesAndDatabases.add(str));

databases.forEach(str -> languagesAndDatabases.add(str));

**return** languagesAndDatabases;

}

Below is the code for entire program.

**package** org.collections;

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** JoinTwoArrayLists {

**public** **static** **void** main(String[] args) {

List<String> languages = *populateLanguages*();

System.***out***.println("List of languages " + languages);

List<String> databases = *populateDatabases*();

System.***out***.println("List of databases " + databases);

List<String> joinLists = *addAllDemo*(languages, databases);

System.***out***.println("joined lists by addAll(..) " + joinLists);

List<String> joinListsByLambda = *addListByLambda*(languages, databases);

System.***out***.println("joined lists by lambda operator " + joinListsByLambda);

}

**public** **static** List<String> populateLanguages() {

List<String> languages = **new** ArrayList<String>();

languages.add("Java");

languages.add("JavaScript");

languages.add("C#");

languages.add("Python");

**return** languages;

}

**public** **static** List<String> populateDatabases() {

List<String> databases = **new** ArrayList<String>();

databases.add("MySql");

databases.add("SqlLite");

databases.add("MongoDB");

databases.add("Neo4j");

databases.add("Oracle");

**return** databases;

}

**public** **static** List<String> addAllDemo(List<String> languages, List<String> databases) {

List<String> languagesAndDatabases = **new** ArrayList<String>();

languagesAndDatabases.addAll(languages);

languagesAndDatabases.addAll(databases);

**return** languagesAndDatabases;

}

**public** **static** List<String> addListByLambda(List<String> languages, List<String> databases){

List<String> languagesAndDatabases = **new** ArrayList<String>();

languages.forEach(str -> languagesAndDatabases.add(str));

databases.forEach(str -> languagesAndDatabases.add(str));

**return** languagesAndDatabases;

}

}

Below is the output for the program.

List of languages [Java, JavaScript, C#, Python]

List of databases [MySql, SqlLite, MongoDB, Neo4j, Oracle]

Joined lists by addAll(..) [Java, JavaScript, C#, Python, MySql, SqlLite, MongoDB, Neo4j, Oracle]

Joined lists by lambda operator [Java, JavaScript, C#, Python, MySql, SqlLite, MongoDB, Neo4j, Oracle]

That’s all on joining two lists.