Collections class replaceAll method

As the name suggests replace all method is used to replace all the old values from List with the new value.

Below is the code taken from JDK and I have commented it out so as to understand it properly.

**private** **static** **final** **int** ***REPLACEALL\_THRESHOLD*** = 11;

/\*\*

\* replaceAll() method replaces all the occurrences of oldVal

\* with newVal.

\* \*/

**public** **static** <T>

**boolean** replaceAll(List<T> list,

T oldVal,

T newVal) {

**boolean** result = **false**;

**int** size = list.size();

/\*\*

\* if size of list is less than REPLACEALL\_THRESHOLD i.e 11

\* or list is instance of RandomAccess(ArrayList and

\* CopyOnWriteArrayList) enter this condition.

\* \*/

**if (size < *REPLACEALL\_THRESHOLD* || list instanceof RandomAccess) {**

/\*\*

\* If the value to be replaced is null then search for

\* it and replace it with newVal using set() method of

\* List interface. Set result to true if List is changed.

\* \*/

**if** (oldVal == **null**) {

**for** (**int** i = 0; i < size; i++) {

**if** (list.get(i) == **null**) {

list.set(i, newVal);

result = **true**;

}

}

}

/\*\*

\* If the value is not null use Object#equals(Object)

\* method to find the element. If found set the newVal

\* at the current index. Set result to true if List

\* is changed.

\* \*/

**else** {

**for** (**int** i = 0; i < size; i++) {

**if** (oldVal.equals(list.get(i))) {

list.set(i, newVal);

result = **true**;

}

}

}

}

/\*\*

\* If the List is not instance of RandomAccess for example

\* LinkedList then enter this code of block.

\* \*/

**else** {

/\*\*

\* Use ListIterator to traverse the List.

\* \*/

ListIterator<T> itr = list.listIterator();

/\*\*

\* If the value to be replaced is null then search for

\* it and replace it with newVal using set() method of

\* List interface. Set result to true if List is changed.

\* \*/

**if** (oldVal == **null**) {

**for** (**int** i = 0; i < size; i++) {

**if** (itr.next() == **null**) {

itr.set(newVal);

result = **true**;

}

}

}

/\*\*

\* If the value is not null use Object#equals(Object)

\* method to find the element. If found set the oldVal

\* to newVal. Set result to true if List is changed.

\* \*/

**else** {

**for** (**int** i = 0; i < size; i++) {

**if** (oldVal.equals(itr.next())) {

itr.set(newVal);

result = **true**;

}

}

}

}

**return** result;

}

RandomAccess interface plays a very important role here. RandomAccess is a marker interface. Marker interfaces does not have any methods or variables declared in it. It is used for tagging purpose. Implementing this interface denotes specific behavior on part of implementing class. Currently I know about 4 classes ArrayList, CopyOnWriteArrayList, Stack and Vector which implements this interface.

For the above method if there is any class which implements RandomAccess interface then it will enter the first if condition which is in **bold font**.

Below is the program that uses replaceAll method. It replaces Integer 3 with 300.

**public** **class** CollectionsReplaceAll {

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

List<Integer> numbers = Arrays.*asList*(3, 2, 1, 4, 3, 2, 1, 3);

Integer oldVal = Integer.*valueOf*(3);

Integer newVal = Integer.*valueOf*(300);

**new** CollectionsReplaceAll()

.replaceAllDemo(numbers, oldVal, newVal);

}

**public** <T> **void** replaceAllDemo(**final** List<T> list,

**final** T oldVal,

**final** T newVal) {

**if**(list == **null**){

**throw** **new** NullPointerException("Argument is null");

}

System.***out***.println("List before replaceAll(): "+list);

Collections.*replaceAll*(list, oldVal, newVal);

System.***out***.println("List after replaceAll(): "+list);

}

}

Output

List before replaceAll(): [3, 2, 1, 4, 3, 2, 1, 3]

List after replaceAll(): [300, 2, 1, 4, 300, 2, 1, 300]

That’s all on replaceAll() method of Collections class.

In next posts we will saw other methods for Collections class.