Iterator in Java Collections Framework

לפניכם הסבר על Iterator בהקשר של ספריית Collections בג'אווה.

המטרות של מדריך זה הן:

1. להציג בפניכם את השימוש של Iteratorים במסגרת ספריית Collections.
2. להראות דוגמת שימוש ונקודות בעייתיות

יש לדעת את החומר המוצג במדריך זה גם עבור תרגילי הבית אשר משתמשים באוספים וגם כחומר למבחן.

תוכן עניינים

1. חזרה על Iterator וIterable מעבודה עצמית 6
2. המנשק Iterable בספריית Collections
3. שימוש מפורש באיטרטור לעומת שימוש בfor each
4. הסרת איברים ממבנה הנתונים בזמן שימוש באיטרטור
5. שאלה לדוגמה

חזרה על Iterator וIterable

בעבודה עצמית 6 הוצגו בפניכם המנשקים Iterable וItrator. תזכורת מאותה עבודה עצמית -

Iterator

מחלקה המממשת את המנשק Iterator היא מחלקה שכל מופע שלה יהווה איטרטור למחלקה אחרת שהאיטרטור מומש עבורה. Iterator זה כלי שיכול להיות קיים עבור מחלקה. אם יש לנו מחלקה A כלשהי, אנו יכולים ליצור עבור המחלקה הזאת- Iterator . אנחנו ניצור מחלקה חדשה, ונקרא לה MyIteratorForClassA.אנחנו נדאג שMyIteratorForClassA תממש את המנשקIterator, ואז נוכל ליצור אובייקט חדש מהמחלקה הזאת והוא יהיה איטרטור למחלקה A.

המנשק Iterator מחייב את המחלקות המממשות אותו לממש 3 מתודות:

hasNext() - מתודה שמחזירה האם קיים עוד איבר שהאיטרטור יכול להחזיר.

Next() – מתודה שמחזירה את האיבר הבא אם קיים כזה.

Remove() – מתודה שמסירה את האיבר האחרון שהוחזר. קיים לה מימוש ברירת מחדל במנשק שזורק שגיאה ואין חובה לממש אותה במחלקה המממשת.

Iterable

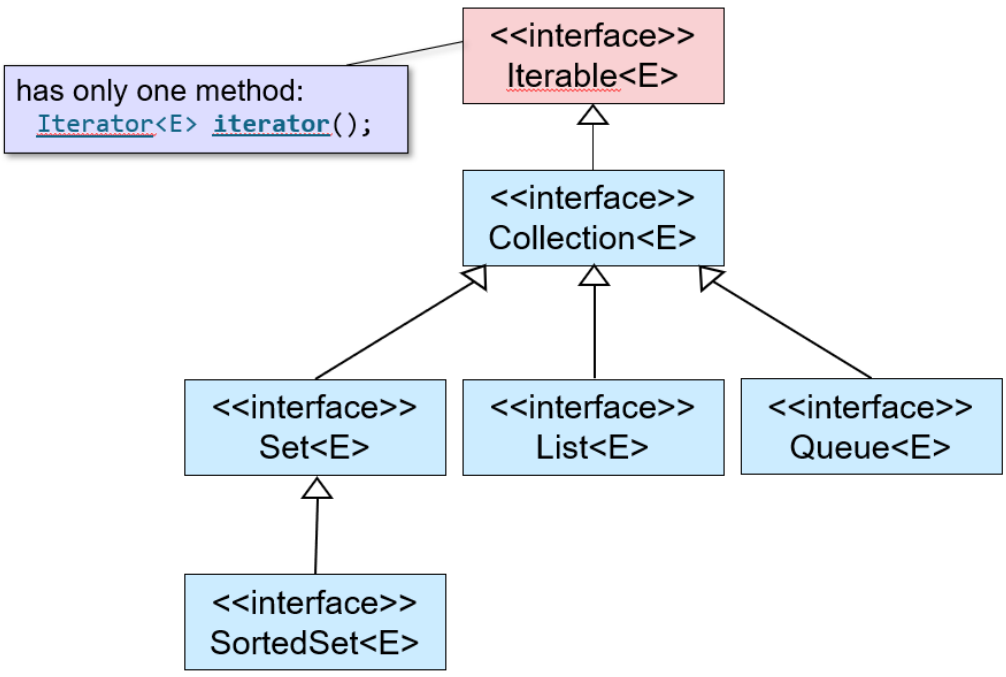
מחלקה המממשת את המנשק Iterable זו מחלקה שצריכה שיהיה לה מימוש למתודה Itrerator<E> ()iterator שלמעשה מחזירה מופע של איטרטור (מחלקה המממשת את המנשק Iterator). ניתן יהיה לעבור בלולאת for each על אובייקטים מטיפוס מחלקה המממשת את המנשק.

נקודה שחשוב לזכור לגבי Iterator לעומת Iterable - Iterator יכול להתקיים בלי קשר לIterable. אנחנו יכולים ליצור המון איטרטורים למחלקה A, לכתוב פונקציה שתייצר לנו איטרטור למחלקה A ולקרוא לה ()getIterator, וכל זה לא קשור לIterable. המנשק Iterator קיים ללא תלות ב-Iterable.
ההפוך הוא הנכון לגבי Iterable – מימוש מנשק אומר שהמחלקה המממשת מחזירה אובייקט מטיפוס Iterator, ולכן Iterable תלוי לחלוטין במנשק Iterator.

המנשק Iterable בספריית Collections

המנשק Collection מרחיב את המנשק Iterable, ולכן כל המנשקים שמרחיבים את Collection מרחיבים בעצמם את Iterable, ולכן לכולם ישנה מתודה ()iterator שמחזירה מופע של איטרטור מתאים לאותו אוסף.

חשוב לזכור – המנשק Map<K,V> והמנשק היורש ממנו SortedMap<K,V> אינם Iterables! כלומר הם אינם מממשים את המנשק Iterable, לא ניתן לעבור עליהם בלולאת foreach. כדי לעבור איברים בMap ניתן להשתמש במתודות שמחזירות Setים של המפתחות, ערכים, או זוגות של המפתחות ערכים כפי שראינו בתירגול.



שימוש מפורש באיטרטור לעומת שימוש בfor each

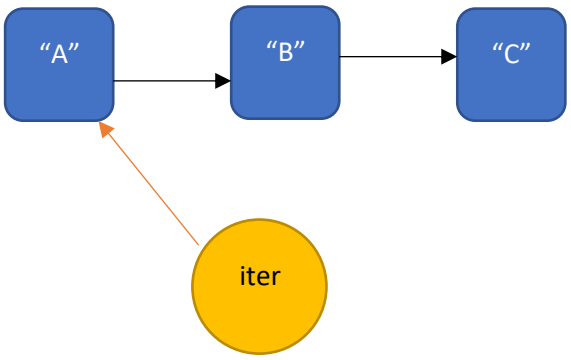
השימוש הבסיסי ביותר בIterator של מחלקה שמייצגת אוסף של אלמנטים הוא לרוץ על האלמנטים בלולאה. ישנן 2 דרכים לעשות זאת:

1. שימוש מפורש באיטרטור:

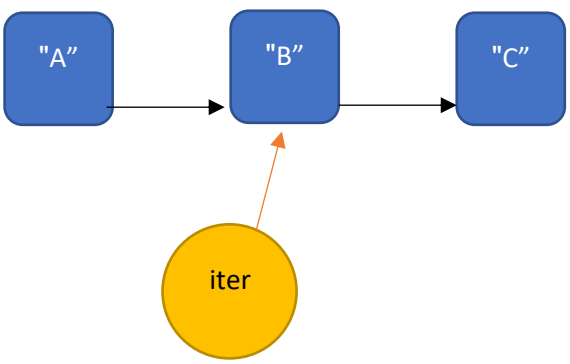
```
for (Iterator<String> iter = stringCollection.iterator(); iter.hasNext(); ) {
      System.out.println(iter.next());
}
```

נניח כי המשתנה stringCollection מצביע על אוסף מחרוזות כלשהו (למשל רשימה של מחרוזות). הפונקציה iterator() מחזירה אובייקט מטיפוס איטרטור שהוא מאפשר לרוץ על אוסף המחרוזות. תנאי הלולאה הוא כל עוד יש לאיטרטור עוד איברים לרוץ עליהם (באמצעות הבדיקה iter.hasNext()), ושימו לב שאין תנאי מקדם (החלק של i++ כפי שמופיע בלולאה הבאה for (int i=0; i<N; i++)). הקידום של האיטרטור נעשה ע"י קריאה ל()next בגוף הלולאה שהיא עושה 2 פעולות – מקדמת את האיטרטור לאיבר הבא ומחזירה את האיבר הקודם שהאיטרטור הצביע עליו. ניתן לראות זאת בדוגמה הבאה עבור רשימה מקושרת.

לאחר יצירת האיטרטור:



לאחר הפעלת ()next:



2. שימוש בforeach:

```
for (String str : stringCollection) {
      System.out.println(str);
}
```

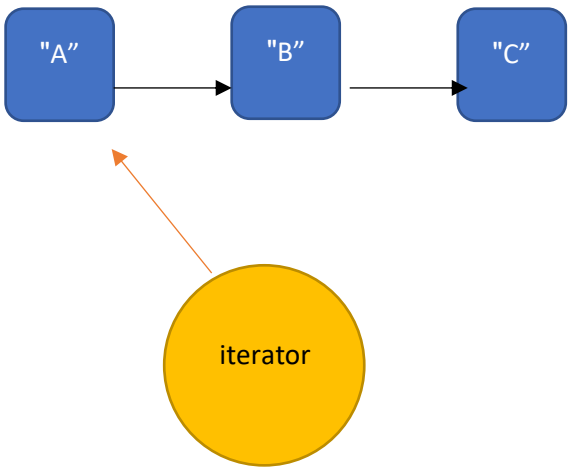
את המעבר של foreach אנחנו מכירים, בכל איטרציה של הלולאה האיבר הבאה מושם למשתנה הזמני str. כאשר אנחנו משתמשים בforeach, נוצר איטרטור ע"י ג'אווה והיא מטפלת בקריאות לhasNext() וnext() באופן שקוף לנו, ולכן שימוש בforeach זה שימוש באיטרטור באופן עקיף. נתין להשתמש בforeach עם מחלקות שהן Iterable ולכן ניתן להשתמש עם foreach עם כל מחלקה שממממשת את המנשק Collection.

הסרת איברים ממבנה הנתונים בזמן שימוש באיטרטור

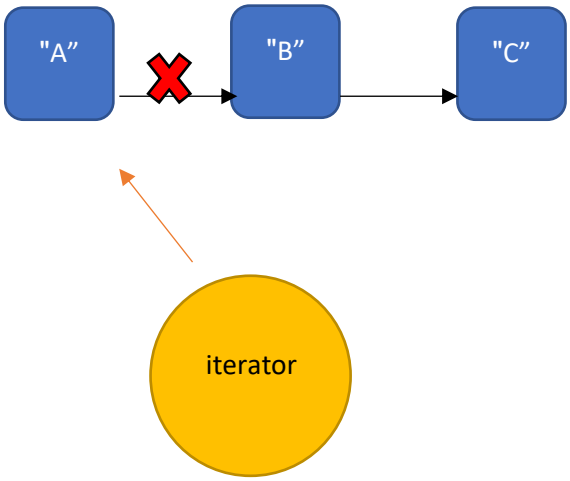
השימוש בforeach הוא נוח יותר פעמים רבות, אך הוא עשוי להיות בעייתי במקרים שבהם אנחנו רוצים לשנות את מבנה הנתונים עליו אנחנו רצים. התבוננו בדוגמה הבאה:

```
for (String str : stringCollection) {
      if (someCondition){
            stringCollection.remove(str);
            //can possible cause ConcurrentModificationException
      }
}
```

יש לנו כאן ריצה של foreach על אוסף של מחרוזות, ובהינתן תנאי מסוים (למשל כל המחרוזות שמכילות את התו 'A'), אנחנו מסירים את אותה המחרוזת. במצב כזה, אנו עלולים לגרום לשגיאה בריצת האיטרציה. נניח למשל שאוסף המחרוזות שלנו הוא הרשימה המקושרת ממקודם.



הפעלה של remove על האיבר "A" במצב כזה, עלול לגרום לכך שהiterator של הforeach לפתח יצביע על איבר שנמחק מהרשימה, מה שעלול לגרום לשגיאה ConcurrentModificationException.כיוון שהאיבר שהאיטרטור מצביע עליו לא חלק מהרשימה יותר. (שימו לב למילה עלול – שגיאה כזו יכולה לקרות כתלוי במימוש של האיטרטור באותו מבנה נתונים, ריצה כזו עשויה להסתיים ללא שגיאה).



לכן, כאשר אנחנו רוצים לערוך את מבנה הנתונים שעליו אנחנו עוברים רצוי להשתמש בפונקציה ()remove של האיטרטור במידה וישנה מתודה כזאת, ולשם כך יש לעשות שימוש מפורש באיטרטור כפי שמופיע בדוגמה הבאה. השימוש ב()remove של האיטרטור הוא בטוח כיוון שכותבת מחלקת האיטרטור ודאי דאגנ לקידום של האיטרטור לאיבר הבא לפני ההסרה בצורה בטוחה.

```
for (Iterator<String> iter = stringCollection.iterator(); iter.hasNext(); ) {
	iter.next(); /*we must advance the iterator, otherwise we will get
IllegalStateException */
	if (someCondition){
		iter.remove(); /*this is safe, unless remove is not supported*/
	}
}
```

שאלה לדוגמה

לסיום, התבוננו במתודה הבאה. היא נועדה להחזיר את האיבר השלילי הראשון מבין רשימה של מספרים.

האם היא מבצעת את תפקידה בצורה נכונה?

```
public static int getFirstNegativeNumInList(List<Integer> lst){
	Integer item = null;
	do {
		Iterator<Integer>it = lst.iterator();
		item = it.next();
	}
	while (item > 0)
	return item;
}
```

ישנו באג במתודה זו. האיתחול של האיטרטור נעשה בתוך לולאת הwhile, ולכן בכל איטרציה של הלולאה האיטרטור מאותחל מחדש בהצבעה לאיבר הראשון ברשימה. אם האיבר הראשון הוא שלילי, המתודה תחזיר תשובה נכונה. אך אם האיבר הראשון הוא חיובי באיטרציה הראשונה item יצביע על האיבר הראשון החיובי, אך באיטרציה הבאה יהיה איתחול מחדש של האיטרטור וitem יצביע שוב לאותו איבר חיובי, ולכן המתודה לעולם לא תסתיים.