# אוספים - Collections

## Collection

collection הוא ממשק ב-java ומייצג קבוצה של אובייקטים שהם איברי האוסף. collection מייצג ממשק שורש בהיררכיה של מבני נתונים, כלומר כל מבני הנתונים האחרים חייבים לממש אותו. יש שלושה ממשקים נוספים שממשים את collection ומרחיבים אותו: List, Set, Map.

הערה – map לא מממש את collection. אמנם יש בו בנאי שמאפשר לקבל אותו כפרמטר.

### ממשק List

מבני נתונים (מחלקות) שמממשים את List הם מבני נתונים שהאיברים בהם מסודרים בסדר מסוים. כלומר ניתן לגשת אל האיבר במקום ה-i במבני נתונים אלו. בנוסף, מבני נתונים אלו מאפשרים שכפול של אובייקטים, כלומר ניתן לשמור את אותו אובייקט במספר מקומות ב-List.

המחלקות מסוג List הן: LinkedList, Vector, Stack, ArrayList.

### ממשק Set

מבני נתונים שממשים את Set הם מבני נתונים שמחזיקים רק שעתוק אחד מכל אובייקט, כלומר לא יכול להיות אובייקט שיופיע יותר מפעם אחת. האיברים במבני נתונים אלו אינם מסודרים.

המחלקות מסוג Set הן:

HashSet – ממשק שהחיפוש בו מאוד מהיר, ממומש באמצעות HashTable.

TreeSet – בנוסף לכך שממש את Set גם האיברים בו מסודרים. שילוב בין Set ו-List.

### ממשק Map

מבני נתונים שממשים את Map הם מבני נתונים שבהם לצד האובייקט המייצג value נשמר אובייקט נוסף שהוא ה-key שלו.

המחלקות מסוג Map הן: HashMap, TreeMap.

## איטרטור Iterator

Iterator<T> זהו ממשק גנרי ב-Java שמטרתו לעבור על כל האיברים של מבנה נתונים באופן סדרתי. אובייקט מסוג מחלקה שמממשת את איטרטור, זהו אובייקט שיחזיק איבר כלשהו במבנה נתונים, וכל פעם נאמר לאיטרטור לקפוץ לאיבר הבא עד שנעבור על כל האיברים במבנה הנתונים.

הממשק מורכב משלוש שיטות:

**hasNext** - שיטה בוליאנית שמחזירה אמת אם יש עוד איבר לעבור עליו.

**next** - מחזירה את האיבר הבא במבנה הנתונים ומכניסה אותו לאיטרטור. נצטרך להפעיל שיטה זו בפעם הראשונה כדי לקבל את האיבר הראשון.

**remove** - מוחקת את האיבר הנוכחי שמוחזק באיטרטור ומחזירה את האיטרטור לאיבר הקודם שהחזיק. כדי להפעיל שיטה חייבים לבצע לפחות פעולת next אחת על האיטרטור.

השימוש באיטרטור יעשה בדרך כלל באופן הבא:

Iterator<Integer> iterC = c.iterator();

while(iterV.**hasNext**()){

elem = iterV.**next**();

//do something

}

כל מבנה נתונים מסוג collection מממש את איטרטור, ולכן כדי לקבל אובייקט מסוג איטרטור של מבני נתונים c נשתמש בפקודה:

Iterator<Integer> iterC = c.iterator();

### מימוש איטרטור

עבור מחלקה שיצרנו שמכילה מבנה נתונים, כמו לדוגמא עץ בינארי, נוכל ליצור איטרטור משלנו שיעבור על כל הנתונים במחלקה על ידי שנכתוב מחלקה נוספת שתממש את הממשק Iterator<T>. במחלקה זו נצטרך לממש את שלושת השיטות לעיל ולהגדיר עבור כל אחת מה נרצה שתעשה. לאחר מכן נגדיר במחלקה הראשית שיטה iterator() שתחזיר אובייקט מסוג המחלקה שמממשת את Iterator.

public class MyClass{

public MyIterator **iterator**(){

return new MyIterator();

}

private class MyIterator **implements Iterator<T>**{

public MyIterator(){…} //constructor

public boolean **hasNext**(){…}

public T **next**(){…}

public void **remove**(){…}

}

}

### ListIterator

זהו ממשק שמממש את Iterator ומרחיב אותו לשיטות נוספות, כמו גישה אל האיבר הקודם, שינו והוספת איבר ולא רק מחיקה ועוד. כל מבנה נתונים מסוג List מממש את ListIterator.

## Comparator

כאשר יש לנו מבנה נתונים המכיל אובייקטים כלשהם, מהמחלקה Player לדוגמא, שהם אינם פרימיטיביים, נרצה למיין את מבנה הנתונים לפי אחד השדות השייכים למחלקה. יש שתי שיטות לעשות זאת:

### Comparable<T>

בשיטה הראשונה בתוך המחלקה שיצרה את האובייקטים Player נממש את הממשק Comparable<T>, כאשר במקום T נכתוב את שם המחלקה Player. לאחר מכן נצטרך לממש את השיטה: compareTo(Player otherPlayer), המחזירה ערך int. בתוך שיטה זו נעשה את ההשוואה שתחליט איזה אובייקט יופיע קודם במבנה הנתונים, האובייקט שהפעיל את השיטה (this) או האובייקט שהתקבל כפרמטר. אם הערך המוחזר מהשיטה חיובי אזי this יופיע ראשון, אם שלילי הפרמטר יופיע ראשון, ואם 0 שניהם שווים ואין הבדל ביניהם. לאחר מימוש שיטה זו נוכל לבצע על מבנה הנתונים d, המחזיק את האובייקטים, את השיטה הסטטית Collections.sort(d), המקבלת אובייקט מסוג Collection וממיינת אותו לפי שיטת compareTo.

### Comparator<T>

בשיטה זו ניצור מחלקה **חדשה** PlayerComparator שתממש את הממשק Comparator<T>, כאשר T הוא שם המחלקה שמייצרת את האובייקטים Player. לאחר מכן נצטרך לממש רק שיטה אחת והיא: compare(Player firstPlayer, Player secondPlayer), המחזירה ערך int. בתוך שיטה זו נעשה את ההשוואה שתחליט איזה אובייקט משני הפרמטרים יופי קודם במבנה הנתונים. אם הערך המוחזר מהשיטה חיובי הפרמטר הראשון יופי קודם, ואם שלילי הפרמטר השני יופיע קודם, ואם 0 שניהם שווים. לאחר שנסיים זאת נוכל לבצע על מבנה הנתונים d, המחזיק את האובייקטים, את השיטה הסטטית Collections.sort(d, PlayerComparator), המקבלת אובייקט מסוג Collection ואובייקט מסוג Comparator, וממיינת את האוסף לפי הלוגיקה של הקומפרטור.

בדרך כלל נעדיף להשתמש בשיטה השנייה של Comparator ולא ב-Comparable, משתי סיבות: בשיטה השנייה לא צריך לשנות דבר במחלקה המקורית, ולכן אנו מסתכנים בפחות שגיאות. סיבה שנייה היא מפני שניתן ליצור מספר מחלקות Comparator, שכל אחת תמיין לפי שדה מחלקה שונה. ואילו ב-Comparable ניתן למיין רק לפי שדה מחלקה אחד, ואם נרצה לשנות זאת נהיה חייבים לשנות את הקוד.

## enum

כאשר אנו רוצים לייצג בתוכנית שלנו מספר ידוע של מחרוזות קבועות ולא משתנות, כמו לדוגמא: אפשרויות בתפריט, שמות החודשים, סוגי קלפים בחפיסה וכו', יש כמה דרכים לעשות זאת. הדרך הפשוטה היא לייצג אותן במחרוזות String, אמנם בשיטה זאת יכולים להיווצר תקלות רבות בגלל שינוי קל של תווים, וגם השוואה בין שתי מחרוזות היא כבדה יחסית. פתרון אחר הוא להשתמש ב-enum.

enum זוהי דרך קלה לבנות מחלקה שהשדות שלה הם ערכים קבועים וסטטיים. במחלקה זו בהצהרת המחלקה במקום להשתמש במילה השמורה Class נרשום enum, ובכך ניצור מחלקה מסוג זה. ניתן גם ליצור מחלקה זו כמחלקה פנימית. כל שדה ב-enum מחזיק איזשהו מספר קבוע, כך שהשוואה בין שתי ערכים היא בעצם השוואה בין שני מספרים המתבצעת בקלות. נהוג שכל הערכים ב-enum רשומים באותיות גדולות.

מכיוון שכל הערכים סטטיים, כדי לגשת לערך נרשום את השם של מחלקת enum, לאחר מכן נקודה '.' , ולבסוף את שם הערך. ניתן גם ליצור אוסף של כל הערכים על ידי השיטה הסטטית values על שם מחלקת ה-enum. בדרך כלל נשתמש ב-enum בתוך תנאי switch.

ההבדל בי מחלקת enum למחלקה רגילה הוא שבמחלקת enum אוטומטית כל השדות הם ציבוריים(public) , סטטיים (static), וקבועים (finale). ניתן גם להוסיף למחלקת enum בנאים ושיטות.

enum Day {

SUNDAY, MONDAY, TUESDAY, WEDNESDAY,

THURSDAY, FRIDAY, SATURDAY;

}

public void main(String[] args) {

switch (new Day()) {

case FRIDAY:

System.out.println(Day.FRIDAY + " are better.");

break;

default:

System.out.println("Midweek days are so-so.");

break;

}

}

}