Collections

מבנים דינמים לאחסון אובייקטים

הצורך באוספים וספריית ה Collection

- עד כה השתמשנו במערך לשמירת נתונים(פרימיטיבים ואובייקטים).
- הבעיה היתה בהיותו מבנה סטטי - גודלו היה קבוע מרגע יצירתו.
- הגדלה דרשה הקצאת זכרון מחדש והעתקת אובייקטים - בזבזני.
- ספריית ה Collections אשר מאגדת יחדיו ממשקים שונים ומימושים שלהם ומטרתה לספק מסגרת מאחדת למבני נתונים (אוספים) דינמיים.

ספריית ה Collection (המשך)

- כל האוספים הללו הם אובייקטים אשר יכולים להכיל בתוכם אובייקטים אחרים וגודלם יכול להשתנות בזמן הריצה ובאופן דינמי.
- הספרייה מכילה ממשק אחד כללי הנקרא Collection ממנו יורשים ממשקים שונים כל אחד מיישם אותו באופן מעט שונה ולפי לוגיקה פנימית שונה.
- הספריה גם מכילה מחלקה סטטית המכילה אלגורתמים הניתנים להפעלה על האוספים.

Collection interface subclasses

- בראש כל המבנים השונים בספריה נמצא ה Collection interface שהוא מגדיר מבנה כללי לניהול אובייקטים בצורה דינמית ממנו יורשים כמה ממשקים חשובים ובינהם:
- **List** לאחסון אוסף **סדור** של אובייקטים (לכל אובייקט יש מיקום באוסף).
- **Set** לאחסון אוסף **יחודי וללא חזרות** של אובייקטים.
- **Map** המאחסנת **זוגות של אובייקטים לפי מבנה של key-value** כאשר ה key הוא יחודי.

Collection’s classes

- **ArrayList** ו **LinkedList** אשר יורשות מ **AbstractList** שמיישמת את ממשק ה List ומייצגים מערך דינמי ורשימה מקושרת.
- **HashSet** ו **TreeSet** אשר יורשות מ **AbstractSet** המיישם את הממשק Set ומייצגים טבלת Hash רגילה וממויינת (בהתאמה).
- **HashMap** ו **TreeMap** אשר יורשות מ **AbstractMap** המיישם את הממשק Map ונותנות בטבלת hash או בעץ (בהתאמה) מבנה לאחסון אובייקטים לפי key - value

Hash code

“aba” $\xrightarrow{\text{hash code}}$ 292

“ima” $\xrightarrow{\text{hash code}}$ 311

“abc” $\xrightarrow{\text{hash code}}$ 294

“aab” $\xrightarrow{\text{hash code}}$ 292

292	“aba” “aab”
311	“ima”
294	“abc”

אלגוריתמים עיקריים ל Collections

הספריה Collections מכילה מחלקה בעלת אוסף פונקציות סטטיות הניתנות להרצה על כל Collection:

- חיפוש בינארי, בהינתן List **ממויינת** וערך כלשהו, מחזירה את המיקום שלו ברשימה באמצעות חיפוש יעיל ומהיר.
- פונקציות למיון מציאת איבר מינימלי ומקסימלי ברשימה לפי ה Natural ordering (יישום Comparable) או בהתאם ל instance של Comparator.
- פונצקיות נוספות להעתקה מרשימה אחרת, להפוך סדר, להחלפה...

Generics

- Generics הוכרזו בג׳אווה 5 ונפוצים מאוד בשימוש ב Collections.
- הטיפוס המאוחסן במבנה יוגדר כטיפוס כללי ויקבע בעת היצירה.
- לולאת for שאינה עושה אבחנה בין הטיפוסים השונים.
- פונקציה למיון שיכולה למיין כל מערך.
- המטרה היא לגלות שגיאות בזמן קומפליציה (בזמן כתיבת הקוד) שאחרת היו מתגלות בזמן ריצה וגורמות לקריסה.

Generics function example

```
// generic method printArray
   public static < E > void printArray( E[] inputArray )
   {
        for ( E element : inputArray ){
           System.out.println(element);
        }
        System.out.println();
    }
```

```
public static void main( String args[] )
   {
      // Create arrays of Integer, Double and Character
      Integer[] intArray = { 1, 2, 3, 4, 5 };
      Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
      Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };

      printArray( intArray  ); // pass an Integer array

      printArray( doubleArray ); // pass a Double array

      printArray( charArray ); // pass a Character array
   }
```

הפונקציה כתובה
פעם אחת בלבד
ומבצעת אותו
אלגוריתם ללא
תלות בטיפוס עליו
היא עובדת

Generic classes

הגדרת המחלקה הגנרית

```
class MyClass<E> {

  E item;

  public void setItem(E value) {
    this.item = value;
  }

  public E getItem() {
    return this.item
  }

}
```

היתרון הגדול במקרה הזה על פני שימוש ב object כללי הוא שלאחר יצירת ה instance הקומפיילר יצפה שנכניס String ואם לא אז הוא יעיר לנו בזמן קומפילציה.

בעת יצירת המופע שלה נגדיר מי יהיה הטיפוס של המופע הספציפי שניצור

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

  MyClass<String> x = new MyClass<String>();
}
```

פונקציות עיקריות ArrayList

שם הפונקציה	תפקיד
add(index,object) add(object) add(collection)	מוסיפה את האובייקט למקום המצויין במערך הדינמי מוסיפה את האובייקט לסוף המערך הדימני מוסיפה את כל האיברים מהאוסף המועבר לסוף המערך שלנו
contains(object)	מחזירה אמת אם המערך מכיל את האובייקט המועבר על קריאה ל equals
get(index)	מחזירה אובייקט במקום המבוקש, זורקת IndexOutOfBoundsException
indexOf(object) lastIndexOf(object)	מחזירה את האינדקס של המופע הראשון של האובייקט המועבר מחזירה את האינדקס של המופע האחרון של האובייקט המועבר
remove(index)	מסירה מהרשימה את האובייקט במקום המבוקש ומחזירה אותו
size()	מחזירה את מספר האיברים במערך הדינמי
toArray()	מחזירה מערך של כל העצמים ברשימה בסדר הופעתם

פונקציות עיקריות HashMap

שם הפונקציה	תפקיד
put(K,V)	משייכת את הערך V למפתח K במפה, מחזירה את הערך שנדרס אם המפתח כבר היה בטבלה או null אם הוא חדש.
get(K)	מחזירה את הערך V המשוייך למפתח K או null אם אין ערך כזה
contains(K) contains(V)	מחזירה אמת אם המפה מכילה מיפוי למפתח המועבר מחזירה אמת אם המפה מכילה אחד או יותר מפתחות עבור הערך המועבר
keySet() values()	מחזירה Set של כל המפתחות במפה מחזירה Collection של הערכים במפה
remove(K)	מסירה את המיפוי למפתח זה אם היה קיים ומחזירה אותו אם לא אז null
size()	מחזירה את מספר המיפויים הקיימים במפה
putAll(map)	מוסיפה את כל המיפויים מהמפה המועברת למפה הזו תוך דריסת ערכים קודמים אם המפתחות כבר קיימים.

Iterator

- האיטרטור מאפשר מעבר סידרתי על הרשימה.
- האיטרטור מצביע לתחילת הרשימה ומתקדם עד שמגיע לסופה.
- כדי לקבל איטרטור לרשימה נקרא לפונקציה iterator()
  או listIterator() אם מדובר ב List.
- ListIterator מאפשר לא רק לעבור על האיברים, ולבדוק את ערכם
  אלא גם לשנות אותם.
- הפונקציות החשובות הן:

Iterator (המשך)

hasNext()	פונקציה המחזירה אמת אם יש עוד איברים באים, שקר אחרת.
next()	מחזירה את האיבר הבא או זורקת NoSuchElementException אם אין איבר כזה
remove()	מסירה את האיבר הנוכחי ברשימה וזורקת שגיאה אם לא נקרא next בפעם הראשנה
add(object) set(object)	הפונקציות הללו שייכות ל ListIterator ומטרתן להוסיף איברים ולשנות אותם

iterator example

```
// Create an array list
    ArrayList<String> al = new ArrayList();
    // add elements to the array list
    al.add("C");
    al.add("A");
    al.add("E");
    al.add("B");
    al.add("D");
    al.add("F");
```

```
Iterator itr = al.iterator();

while(itr.hasNext()) {
        String element = itr.next();
        System.out.print(element + " ");
 }
```

```
// Modify objects being iterated
ListIterator litr = al.listIterator();

while(litr.hasNext()) {
        Object element = litr.next();
        litr.set(element + "+");
}
```

Comparator

- Comparator הוא ממשק המכיל פונקציה סטטית compare אשר מקבלת שני אובייקטים ומחזירה מספר חיובי אם הראשון גדול מהשני, אפס אם הם שווים ושלילי אם השני גדול מהראשון.
- הייתרון ביצירת comparator הוא שניתן להגדיר סדר מיון והשוואה אחר מן ה natural ordering כלומר לא צריך לשנות את המחלקה מבפנים אם נרצה באופן זמני למיין אחרת.

comparator example

נניח ונרצה באופן זמני לייצר רשימה של סטודנטים הממויינת בסדר **יורד** של ציונים. נצטרך ראשית להגדיר את ה Comparator

נניח קיימת המחלקה Student אשר מיישמת את Comparable וסדר המיון הטבעי שלה הוא על פי ציונים בסדר **עולה**.

```
class Student implements
Comparable<Student>
{
   private String name;
   private double gpa;
   ….
   @Override
   public int compareTo(Student o) {
      if(o==null)
         return 1;
      if(gpa>o.gpa)
         return 1;
      if(o.gpa>gpa)
         return -1;
      return name.compareTo(o.name);
```

```
class MyStComparator implements
Comparator<Student>
{
   @Override
   public int compare(Student arg0, Student arg1) {
      return (int)-(arg0.getGpa()-arg1.getGpa());
   }
```

ואז לייצר instance שלו ולהעביר לפונקציית המיון

```
ArrayList<Student> studentim=new ArrayList<Student>();
studentim.add(new Student("Yael",92.3));
studentim.add(new Student("Adi",94.5));
studentim.add(new Student("Noa",97.7));
Collections.sort(studentim, new MyStComparator());
```

AutoBoxing - unBoxing

- ראינו כי כל ה Collections יכולים להכיל רק אובייקטים. מה נעשה כאשר נרצה מערך דינמי של פרימיטיבים כדוגמת int, float, double, char, boolean ?
- לכל פרימיטיב קיימת מחלקה שנועדה "לעטוף" אותו באובייקט. ל int קיימת Integer, ל double קיימת Double וכך הלאה.
- בגלל שהשימוש במחלקות הוא הוא נרחב מאוד ג'אווה עוזרת לנו ומייצרת את האובייקטים הללו עבורנו במקרה הצורך וגם "מוציאה" בעצמה את הערך מהאובייקט.

Examples - boxing & unboxing

```
Integer num = 5; // new Integer(5)

ArrayList<Double> numbers=new ArrayList<Double>();
numbers.add(4.7) // number.add(new Double(4.7));

static void printd(double b) { System.out.println(b); }

printd(new Double(7.9))
        //printd((new Double(7.9)).doubleValue())
```

Collection legacy classes

- כל האוספים שקדמו לספריית ה Collections נקראים Legacy classes למרות היותם ישנים הם עברו התאמה לתמוך ב Generics שהוצגו בג׳אווה 5 ועדיין נמצאים בשימוש.

- בין המחלקות הללו ניתן למצוא מבנים מאוד שימושיים את **HashTable** אשר דומה מאוד ל HashMap אך בנגוד אליו, הראשון הוא Thread safe כלומר ניתן לגשת אל המבנה מכמה תהליכים במקביל מבלי להסתכן ב Race condition.

Legacy classes (המשך)

- **Stack** אשר מספקת מבנה לאיחסון אובייקטים
- LIFO - האחרון שנכנס הוא הראשון לצאת
- פונקציות שימושיות כגון ()**pop** המוציאה את האיבר בראש המחסנית ו ()**peek** אשר מאפשרת לראות את האיבר שבראש מבלי להוציא אותו, ו ()**push** אשר דוחפת איבר לראש המחסנית. דוגמה לשימוש במחסנית יכולה להיות שמירת הפעולות ב Undo, הפעולה הראשונה להיות מבוטלת היא האחרונה שעשינו.