

תוכנה 1



תרגול מספר 13:
עוד על טיפוסים מוכללים (גנריים)

חומר עזר על טיפוסים מוכללים

- Java Tutorials ■

<http://docs.oracle.com/javase/tutorial/java/generics/index.html>

Generic and raw types ■

Generic methods ■

Bounded types ■

Wildcards ■

על כל אלה נדבר היום ■

ירושה של טיפוסים גנריים

■ כשכותבים טיפוס גנרי או טיפוס שירש מטיפוס גנרי, יש לשים לב אילו פרמטרים גנריים צריך בכל טיפוס.

■ דוגמאות:

■ מאגר מחרוזות

```
public class MyStringPool extends HashSet<String>
```

■ Map עם מפתח שהוא תמיד Integer

```
public interface IntegerKeyMap<V> extends Map<Integer, V>
```

■ רשימה המאפשרת לשמור עבור כל איבר, בנוסף, ערך מטיפוס אחר P

```
public interface PayloadList<E, P> extends List<E>
```

■ רשימה דו-מימדית

```
public class ArrayList2D<E> extends ArrayList<List<E>>
```

שירותים גנריים

- כמו מחלקות גם מתודות יכולות להיות גנריות
- נגדיר את הטיפוס הגנרי לפני טיפוס הערך המוחזר

```
public class Sets {  
  
    public static <T> Set<T> union(Set<T> s1, Set<T> s2) {  
        Set<T> result = new HashSet<>(s1);  
        result.addAll(s2);  
        return result;  
    }  
    ...  
}
```

קוד הלקוח

- בקריאה לשירות ניתן להשמיט את הטיפוס הגנרי
- הקומפילר "יבין" זאת בעצמו מתוך ההקשר!
- ניתן לכפות טיפוס T ע"י: `Class.<T>method()`

```
public static void main(String[] args) {  
    Set<String> s1 = new HashSet<>();  
    Set<String> s2 = new HashSet<>();  
    Sets.union(s1, s2);  
}
```

מגבלות על T - הפונקציה max

■ כיצד נממש פונקציה המוצאת את האיבר המקסימלי באוסף כלשהו?

■ נדרוש יחס סדר על טיפוס האברים

```
public static <T extends Comparable<T>> T max(Collection<T> coll) {  
    if (coll.isEmpty())  
        return null;  
    Iterator<T> i = coll.iterator();  
    T result = i.next();  
    while (i.hasNext()) {  
        T t = i.next();  
        if (t.compareTo(result) > 0)  
            result = t;  
    }  
    return result;  
}
```

עוד נחזור ל-max

תזכורת

■ אם Sub הוא תת-טיפוס של Super
■ אז Sub[] הוא תת-טיפוס של Super[]
(זיכרו, יורש מ Integer יורש מ Number)

✓

```
Integer[] intArr = new Integer[10];  
Number[] numArr = intArr;  
numArr[0] = 2.3 // throws ArrayStoreException  
                // type safety was compromised
```

■ זה לא נכון לגבי טיפוסים גנריים!

■ לדוגמא, List<Sub> אינו תת-טיפוס של List<Super>

✗

```
List<Integer> intList = new ArrayList<Integer>();  
List<Number> numList = intList;  
// cannot break type safety during compilation
```

מחסנית

נתונה המחלקה: ■

```
public class Stack<E> {  
    public Stack() {...}  
    public void push(E e) {...}  
    public E pop() {...}  
    public boolean isEmpty() {...}  
}
```

נרצה להוסיף ■

```
public void pushAll(Collection<E> src) {  
    for (E e : src)  
        push(e);  
}
```

מה הבעיה במימוש? ■

הבעיה

מה קורה עבור הקוד הבא: ■

```
public static void main(String[] args) {  
    Stack<Number> numberStack = new Stack<>();  
    ✓ numberStack.push(6);  
  
    Collection<Integer> integers = Arrays.asList(1, 2, 3, 4);  
    ✗ numberStack.pushAll(integers);  
}
```

הודעת שגיאה ■

The method `pushAll(Collection<Number>)` in the type `Stack<Number>` is not applicable for the arguments `(Collection<Integer>)`

ממה נובעת הודעת השגיאה? ■

פתרון - Wildcards

■ שלושה סוגים של wildcards:

1. ?

קבוצת "כל הטיפוסים" או "טיפוס כלשהו"

2. **T extends ?**

משפחת תתי הטיפוס של T (כולל T)

3. **T super ?**

משפחת טיפוס העל של T (כולל T)

? extends E

טיפוס הקלט ל `pushAll` ■

במקום “Collection of E” נרצה ■

“Collection of **some subtype of E**”

```
public class Stack<E> {  
    ...  
    public void pushAll(Collection<? extends E> src) {  
        for (E e : src)  
            push(e);  
    }  
}
```

חסם עליון על טיפוס הקלט ■

E הוא תת טיפוס של עצמו ■

popAll

כעת נרצה להוסיף את popAll ■

```
public class Stack<E> {  
    ...  
    public void popAll(Collection<E> dst) {  
        while (!isEmpty())  
            dst.add(pop());  
    }  
}
```

בעיית קומפילציה? ■

מה עם קוד הלקוח? ■

קוד הלקוח

האם יש בעיה בקוד הלקוח? ■

- ✓ `Stack<Number> numberStack = new Stack<>();`
`Object o = numberStack.pop();`
- ✗ `Collection<Object> objects = new HashSet<>();`
`numberStack.popAll(objects);`

האם השימוש ב `extend` מתאים גם פה? ■

? super E

טיפוס הקלט ל popAll ■

במקום "Collection of E" נרצה ■

"Collection of **some supertype of E**"

```
public class Stack<E> {  
    ...  
    public void popAll(Collection<? super E> dst) {  
        while (!isEmpty())  
            dst.add(pop());  
    }  
}
```

חסם תחתון על טיפוס הקלט ■

E הוא תת טיפוס של עצמו ■

get-put principal

PECS

Producer **E**xtends Consumer **S**uper

■ השתמשו ב **extends** כאשר אתם קוראים נתונים ממבנה, ב **super** כאשר אתם מכניסים נתונים למבנה ואל תשתמשו ב wildcards כאשר אתם עושים את שניהם

■ ב pushAll קוראים נתונים מהמשתנה src

■ ב popAll מכניסים נתונים למשתנה dst

בחזרה ל- max

■ נבחן מחדש את max:

```
public static <T extends Comparable<T>> T max(Collection<T> coll) {}
```

■ נתונות המחלקות:

```
public class Employee implements Comparable<Employee> {...}
```

```
public class Manager extends Employee {...}
```

■ והקוד:

```
Collection<Manager> coll = ...;
```

```
✗ Manager e = max(coll);
```

```
✗ Employee e = Max.<Employee>max(coll);
```

לא ממש
Manager
Comparable<Manager>

לא ניתן להמיר מ-
Collection<Manager>
ל-Collection<Employee>

בחזרה ל- max

■ נבחן מחדש את max לפי עקרון ה PECS

האם צרכן או ספק?

```
public static <T extends Comparable<T>> T max(  
    Collection<T> coll) {  
    if (coll.isEmpty())  
        return null;  
    Iterator<T> i = coll.iterator();  
    ...  
    return result;  
}
```

האם צרכן או ספק?

בחזרה ל- max

■ נבחן מחדש את max לפי עקרון ה PECS

צרכן

```
public static <T extends Comparable<? super T>> T max(  
    Collection<? extends T> coll) {  
    if (coll.isEmpty())  
        return null;  
    Iterator<T> i = coll.iterator();  
    ...  
    return result;  
}
```

ספק

בחזרה ל- max

עוד שינוי אחד דרוש ■

התאמת האיטרטור לטיפוס האברים שבאוסף ■

```
public static <T extends Comparable<? super T>> T max(
    Collection<? extends T> coll) {
    if (coll.isEmpty())
        return null;
    Iterator<? extends T> i = coll.iterator();
    ...
    return result;
}
```

Unbounded Wildcard

- כשלא יודעים או לא אכפת לנו מהו הטיפוס האמיתי
- לדוגמא, פונקציות הפועלות על מבנה ה collection (shuffle, rotate, ...)

```
public static int numberOfElementsInCommon(Set<?> s1, Set<?> s2) {  
    int result = 0;  
    for (Object o : s1) {  
        if (s2.contains(o))  
            result++;  
    }  
    return result;  
}
```

שימוש ב ? הוא בטוח

■ ניתן להוסיף כל אובייקט ל- Collection בלי פרמטר גנרי (raw) -
לא בטוח!

■ לא ניתן להוסיף אובייקטים ל `Collection<?>`, בכלל!
■ חוץ מ null
■ שגיאת קומפילציה

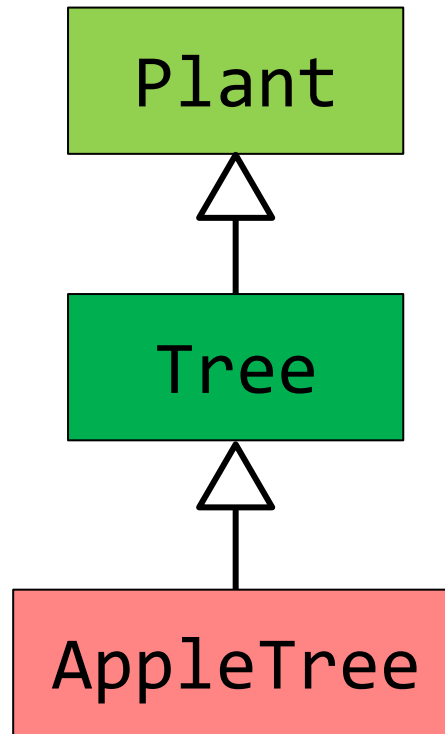
הבדלים בין ? ל-T

- **משמעות: <T>** – הגדרת פרמטר שנצרך לבחור לו ערך (ביצירת מופע של טיפוס גנרי או קריאה לפונק' גנרית)
 - **משמעות: <?>** - נקבע כבר ערך לפרמטר גנרי, אבל הוא לא ידוע
 - **צריך להצהיר על T** בתחילת המחלקה או המתודה
 - אפשר להשתמש ב-? תמיד
 - כל שימוש ב-T מתמפה **לאותו טיפוס** (באותו מופע או פונק' גנריים)
 - כל שימוש ב-? יכול להתמפות ל**טיפוס אחר**
 - אפשר ליצור עצמים עם פרמטר גנרי T, למשל
 - **אי אפשר** ליצור עצמים אלא רק מצביעים עם ?
 - אפשר להוסיף עצמים לאוסף עם פרמטר T
 - **אי אפשר** להוסיף עצמים לאוסף עם פרמטר ?
- ```
new ArrayList<T>()
```

# תרגול



■ נתונה לנו היררכיית המחלקות הבאה:



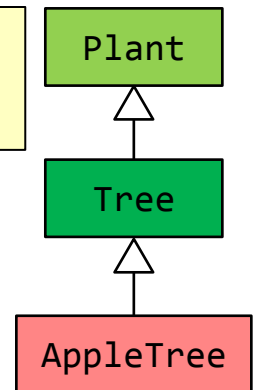
# תרגול

```
List<Tree> treeList = new ArrayList<>();
```

שאלה: לאילו מן הבאים ניתן לעשות השמה ל-treeList?

- ✗ List<Plant> l1 = treeList
- ✗ List<AppleTree> l2 = treeList
- ✗ ArrayList<Tree> l3 = treeList
- ✗ LinkedList<Tree> l4 = treeList
- ✓ Collection<Tree> l5 = treeList

ניתן לעשות המרה  
(casting)



- ✓ List<? extends Plant> l6 = treeList
- ✓ List<? extends Tree> l7 = treeList
- ✓ List<? super Tree> l8 = treeList
- ✗ List<? super Plant> l9 = treeList

?<T extends Tree> l10 📢

- ✓ Collection<?> l10 = treeList
- ✗ Collection<T> l11 = treeList
- ✓ 📢 Collection<? super T> l12 = treeList





**זהו להיום...**