

# תוכנה 1



תרגול מספר 13:  
עוד על טיפוסים מוכללים (גנריים)

בית הספר למדעי המחשב  
אוניברסיטת תל אביב

# 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>
    Integer 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<ArrayList<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

- בקוד הלקוח לא נצטרך לציין מהו הטיפוס הגנרי של השירות
- הקומפיליר "יבין" זאת בעצמו מתוך ההקשר!

```

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[]  
■ (זיכור, Integer יורש מ Number)

✓ `Integer[] intArr = new Integer[10];`  
`Number[] numArr = intArr;`

■ זה לא נכון לגבי טיפוסים גנריים!  
■ לדוגמה, `List<Sub>` אינו תת-טיפוס של `List<Super>`

✗ `List<Integer> intList = new ArrayList<Integer>();`  
`List<Number> numList = intList;`

7

## 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?

8

## 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?

9

## פתרון - Wildcards

■ שלושה סוגים של wildcards:

1. ?  
קבוצת "כל הטיפוסים" או "טיפוס כלשהו"
2. `extends T`?  
משפחת תתי הטיפוס של T (כולל T)
3. `super T`?  
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

11

## 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

14

## get-put principal

PECS

Producer Extends Consumer Super

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` 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?

16

## 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

17

## 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;  
}
```

18

## 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;
}
```

19

## שימוש ב ? הוא בטוח

- ניתן להוסיף כל אובייקט ל- Collection בלי פרמטר גנרי (raw) - לא בטוח!
- לא ניתן להוסיף אובייקטים ל- Collection<?>, בכלל!
  - חוק מ null
  - שגיאת קומפילציה

20

## הבדלים בין ? ל-T

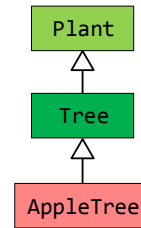
- |   |  |
|---|--|
| <ul style="list-style-type: none"> <li>משמעות: &lt;?&gt; - נקבע כבר ערך לפרמטר גנרי, אבל הוא לא ידוע</li> </ul> | <ul style="list-style-type: none"> <li>משמעות: &lt;T&gt; - הגדרת פרמטר שנוצר לבחור לו ערך (ביצירת מופע של טיפוס גנרי או קריאה לפונק' גנרית)</li> </ul> |
| <ul style="list-style-type: none"> <li>אפשר להשתמש ב-? תמיד</li> </ul>  | <ul style="list-style-type: none"> <li>צריך להצהיר על T בתחילת המחלקה או המתודה</li> </ul>   |
| <ul style="list-style-type: none"> <li>כל שימוש ב-? יכול להתמפות לטיפוס אחר</li> </ul>                          | <ul style="list-style-type: none"> <li>כל שימוש ב-T מתמפה לאותו טיפוס (באותו מופע או פונק' גנריים)</li> </ul>  |
| <ul style="list-style-type: none"> <li>אי אפשר ליצור עצמים אלא רק מצביעים עם ?</li> </ul>                       | <ul style="list-style-type: none"> <li>אפשר ליצור עצמים עם פרמטר גנרי T, למשל <code>new ArrayList&lt;T&gt;()</code></li> </ul>                         |
| <ul style="list-style-type: none"> <li>אי אפשר להוסיף עצמים לאוסף עם פרמטר ?</li> </ul>                         | <ul style="list-style-type: none"> <li>אפשר להוסיף עצמים לאוסף עם פרמטר T</li> </ul>   |

21

## תרגול



- נתונה לנו היררכיית המחלקות הבאה:



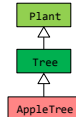
22

## תרגול

List<Tree> treeList = new ArrayList<>();  
 שאלה: לאילו מן הבאים ניתן לעשות השמה ל-treeList?

- ✗ List<Plant> 11
- ✗ List<AppleTree> 12
- ✗ ArrayList<Tree> 13
- ✗ LinkedList<Tree> 14
- ✓ Collection<Tree> 15

ניתן לעשות המרה (casting)



- ✓ List<? extends Plant> 16
- ✓ List<? extends Tree> 17
- ✓ List<? super Tree> 18
- ✗ List<? super Plant> 19

✗ <T extends Tree> 10

- ✓ Collection<?> 110
- ✗ Collection<T> 111
- ✓ Collection<? super T> 112

23



זהו להיום...

24