תוכנה 1 בשפת Java

More Generics

רובי בוים ומתי שמרת

Generic Methods

- 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<T>(s1);
        result.addAll(s2);
        return result;
    }
}
```


קוד הלקוח

- בקוד הלקוח לא נצטרך לציין מהו הטיפוס הגנרי
- הקומפיילר "יבין" זאת בעצמו מתוך ההקשר

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


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

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?

תזכורת

- מערכים הם קו-וריאנטים
  - אם Sub הוא תת-טיפוס של Super אז []Sub הוא תת-טיפוס של []Super
  - ✓

```
Sub[] sub = ...
Super[] sup = sub;
```

- טיפוסים גנריים הם וריאנטים
  - אם T1 ו T2 טיפוסים שונים אז, לדוגמה, בין הטיפוסים List<T1> ו List<T2> לא מתקיים יחס של תת-טיפוסים גם אם יחס כזה מתקיים בין T1 ו T2

✗

```
List<Sub> sub = new ArrayList<sub>();
List<Super> sup = sub;
```

Wildcards - פתרון

- 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 טיפוסי העל של T (כולל T)

הבעיה

- מה קורה עבור הקוד הבא:
  - זיכרו Integer יורש מ Number

```
Stack<Number> numberStack = new Stack<Number>();
Collection<Integer> integers = ...
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?

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?

? 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

קוד הלקוח

- האם יש בעיה בקוד הלקוח?

✓
```
Stack<Number> numberStack = new Stack<Number>();
Object o = numberStack.pop();
```

✗
```
Collection<Object> objects = ...
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

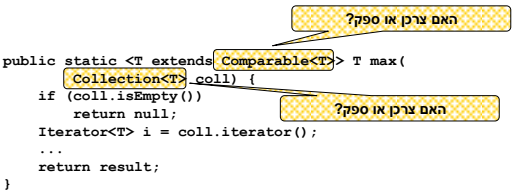
PECS

Producer 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

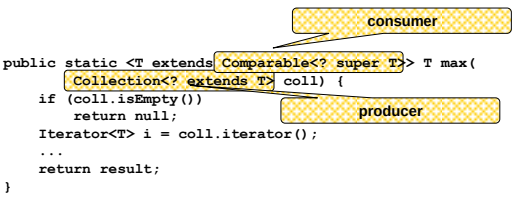
בחזרה ל- max

- נבחן מחדש את max לפי עקרון ה PECS



בחזרה ל- max

- נבחן מחדש את max לפי עקרון ה PECS



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, …)

```
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 raw collection – לא בטוח
- לא ניתן להוסיף אובקייטים בכלל ל collection<?>
  - חוץ מ null
  - שגיאת קומפילציה