תוכנה 1 בשפת Java

תרגול 12: wildcards, enums

הדס צור ואסף זריצקי

מערכים ורשימות

- האם הקוד הבא מתקמפל?

```
String[] as = new String[10];
Object[] ao = as;
```

- וזה?

```
List<String> ls = new ArrayList<String>(10);
List<Object> lo = ls;
```


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 תתי-טיפוסים גם אם יחס כזה מתקיים בין T1 ו T2

✗

```
List<T1> sub = new ArrayList<T1>();
List<T2> sup = sub;
```


משימה

- נממש פונקציה המדפיסה את כל האברים שבאוסף

```
static void printCollection(Collection<Object> c) {
    for (Object o : c)
        System.out.println(o);
}
```

- האם ניתן להעביר לפונקציה List<String> כפרמרטר?

```
public static void main(String[] args) {
    List<String> ls = new ArrayList<String>();
    // populate list
    printCollection(ls);
}
```

✗

- נרצה פונקציה שמקבלת collection של "כל טיפוס"

האמא של כל ה-Collections

- טיפוס העל של כל האוספים הוא:

`Collection<?>` - collection of unkown

- זהו אוסף שטיפוס האברים שבו מתאים להכל
- הטיפוס של האברים נקרא *wildcard type* מסיבות ברורות

```
static void printCollection(Collection<?> c) {
    for (Object o : c)
        System.out.println(o);
}
```


Unbounded Wildcard

```
static void printCollection(Collection<?> c) {
    for (Object o : c)
        System.out.println(o);
}

Collection<?> ls = new ArrayList<String>();
c.add(new Object());
```

✗

תמיד נוכל לקרוא איברים ולהתייחס אליהם כ- Object

מכיוון שאיננו יודעים מה טיפוס האברים באוסף c לא ניתן להוסיף אלמנטים.
כל אובייקט שנעביר כפרמטר ל-add חייב להיות מתת-טיפוס של האבר, אבל איננו יודעים מהו טיפוס האבר.
החריג היחיד הוא null

שימושים

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

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


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:
  - 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?

? 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<Number>();
Object o = numberStack.pop();
```

✗
```
Collection<Object> objects = ...
numberStack.popAll(objects);
```

- האם השימוש ב extend מתאים גם פה?

get-put principal*

- 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

* "*Java Generics and Collections*" by Naftalin and Wadler

? 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

13

Demystifying Enums

- Enums are just syntactic sugar
- We could emulate an Enum with a class
  - This is what the compiler does

```
public enum Operation {
  PLUS("+") { public double apply(double x, double y) {return x + y;} },
  MINUS("-") { public double apply(double x, double y) {return x - y;} },
  TIMES("*") { public double apply(double x, double y) {return x * y;} },
  DIVIDE("/") { public double apply(double x, double y) {return x / y;} };

  private final String symbol;

  Operation(String symbol) { this.symbol = symbol; }
  public String toString() { return symbol; }

  public abstract double apply(double x, double y);
}
```

16

סיכום Wildcards

- שלושה סוגים של wildcards:

1. **?**

   קבוצת "כל הטיפוסים" או "טיפוס לא ידוע כלשהו"

2. **? extends T**

   משפחת תתי הטיפוס של T (כולל T)

3. **? super T**

   משפחת טיפוסי העל של T (כולל T)

Disassembling Operation

```
public abstract class Operation extends Enum {
    private Operation(String s, int i, String symbol) {
        super(s, i);
        this.symbol = symbol;
    }

    public static Operation[] values() {
        Operation aoperation[];
        int i;
        Operation aoperation1[];
        System.arraycopy(aoperation = ENUM$VALUES, 0,
    aoperation1 = new Operation[i = aoperation.length], 0, i);
       return aoperation1;
    }

   ...
}
```

See the code on the course site.

17