

תוכנה 1 בשפת Java

Wildcards and JVM Optimizations

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

```

מחסנית

```

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

```

■ נתונה המחלקה:

■ נרצה להוסיף

```

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

```

■ מה הבעיה במימוש?

הבעיה

- מה קורה עבור הקוד הבא:
- זיכור 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?

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

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?

? super E

■ טיפוס הקלט ל popAll

■ במקום "Collection of E"
"Collection of **some supertype of E**"

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

- סום תחתון על טיפוס הקלט
- E הוא תת טיפוס של עצמו

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

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

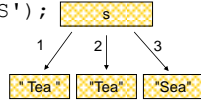
■ ניתן להוסיף כל אובייקט ל raw collection – לא בטוח

■ לא ניתן להוסיף אובייקטים בכלל ל collection<?>
■ חוץ מ null
■ שגיאת קומפילציה

String Immutability

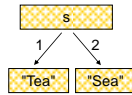
- Strings are constants

```
String s = " Tea ";
s = s.trim();
s = s.replace('T', 'S');
```



- A string reference may be set:

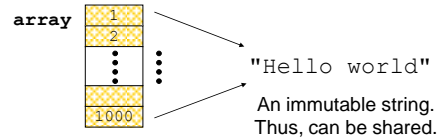
```
String s = "Tea";
s = "Sea";
```



String Interning

- Avoids duplicate strings

```
String[] array = new String[1000];
for (int i = 0; i < array.length; i++) {
    array[i] = "Hello world";
}
```

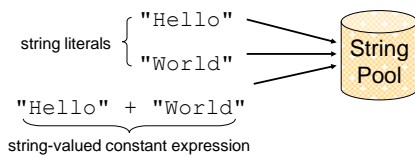


An immutable string.
Thus, can be shared.

14

String Interning (cont.)

- All string literals and string-valued constant expressions are interned.



15

String Constructors

- Use implicit constructor:

```
String s = "Hello";
// (string literals are interned)
```

Instead of:

```
String s = new String("Hello");
// (causes extra memory allocation)
```

16

The StringBuilder Class

- Represents a **mutable** character string
- Main methods: **append()** & **insert()**

- accept data of any type

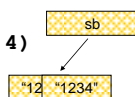
- If: `sb = new StringBuilder("123")`

Then: `sb.append(4)`

is equivalent to

`sb.insert(sb.length(), 4)`

Both yield "1234"



17

The Concatenation Operator (+)

- String conversion and concatenation:

- "Hello " + "World" is "Hello World"
- "19" + 8 + 9 is "1989"

- Concatenation by StringBuilder

- String `x = "19" + 8 + 9;`

is compiled to the equivalent of:

```
String x =
    new StringBuilder().append("19").
        append(8).append(9).toString();
```

18

StringBuilder vs. String

■ Inefficient version using String

```
public static String duplicate(String s, int times) {  
    String result = s;  
    for (int i = 1; i < times; i++) {  
        result = result + s;  
    }  
    return result;  
}
```

A new String object is created each time.

19

StringBuilder vs. String (cont.)

■ More efficient version with StringBuffer:

```
public static String duplicate(String s, int times) {  
    StringBuffer result = new StringBuffer(s);  
    for (int i = 1; i < times; i++) {  
        result.append(s);  
    }  
    return result.toString();  
}
```

no new Objects

20

StringBuilder vs. String (cont.)

■ Even more efficient version:

```
public static String duplicate(String s, int times) {  
    StringBuffer result =  
        new StringBuffer(s.length() * times);  
    for (int i = 0; i < times; i++) {  
        result.append(s);  
    }  
    return result.toString();  
}
```

created with the correct capacity

21