תוכנה 1

תרגול 3: מחלקות

נעמה מאיר ומתי שמרת

מה בתוכנית

- המשך קצר על מבני בקרה
- שימוש במחלקות והעמסת פונקציות
- המחלקות String ו- StringBuffer

ריבוי תנאים (switch)

- תחביר מיוחד לריבוי תנאים

```
switch ( expression ) {
    case ConstantExpression : BlockStatements
    case ConstantExpression : BlockStatements
    ...
}
```

- טיפוס הביטוי הוא שלם שאינו long
- מתבצעת השוואה בינו ובין כל אחד מערכי ה **`case`** ומתבצעת קפיצה לשורה המתאימה אם קיימת
- לאחר הקפיצה מתחיל ביצוע סדרתי של המשך התוכנית, תוך התעלמות משורות ה **`case`**

ריבוי תנאים (switch)

```
System.out.print("The month is: ");

switch (month) {
  case 1: System.out.println("January");
  case 2: System.out.println("February");
  case 3: System.out.println("March");
  case 4: System.out.println("April");
  case 5: System.out.println("May");
  case 6: System.out.println("June");
  case 7: System.out.println("July");
  case 8: System.out.println("August");
  case 9: System.out.println("September");
  case 10: System.out.println("October");
  case 11: System.out.println("November");
  case 12: System.out.println("December");
}
...
```

- מה יודפס אם month == 9?
- ואם month == 13?

משפט break

- משפט ה- break נועד "לשבור" את בלוק הביצוע הנוכחי
- יכול להופיע בתוך לולאות או ב switch

```
switch (month) {
  case 1: System.out.println("January"); break;
  case 2: System.out.println("February"); break;
  case 3: System.out.println("March"); break;
  case 4: System.out.println("April"); break;
  case 5: System.out.println("May"); break;
  case 6: System.out.println("June"); break;
  ...
}
```

- מה יודפס אם month == 6?
- ואם month == 13?

משפט continue

- יכול להופיע רק בתוך לולאות
- כאשר מופיע בלולאות while ו do-while התכנית "תקפוץ" לשיערוך מחדש של תנאי הלולאה ומשם תמשיך בהתאם לתוצאה
- כאשר מופיע בלולאת for התכנית "תקפוץ" לחלק ה increment של הלולאה ומשם תמשיך בביצוע הלולאה

מחלקות - תזכורת

- המחלקה כספרייה של שירותים
  - אוסף של פונקציות בעלות מכנה משותף
    - Arrays – פעולות על מערכים
    - Math – פעולות מתימטיות
    - System – ממשק עם המערכת
- תבנית ליצירת אובייקטים

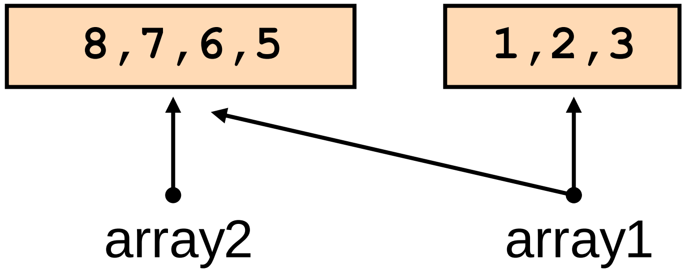
המחלקה Arrays

- פעולות על מערכים – חיפוש, מיון, העתקה וכדומה
- העתקה:

```
int[] array1 = {1,2,3};
int[] array2 = {8,7,6,5};
```

- העתקה נאיבית:

```
array1 = array2;
```



- כיצד נייצר עותק חדש?

העתקה בעזרת Arrays

- **`Arrays.copyOf(...)`**
  - the original array
  - the length of the copy (new array)

```
int[] arr1 = {1, 2, 3};
int[] arr2 = Arrays.copyOf(arr1, arr1.length);
```

- **`Arrays.copyOfRange(...)`**
  - the original array
  - initial index of the range to be copied, inclusive
  - final index of the range to be copied, exclusive

דוגמא

- מה הפלט של הקוד הבא

```
int[] odds = {1, 3, 5, 7, 9, 11, 13, 15};
int newOdds[] =
    Arrays.copyOfRange(odds, 1, odds.length);
for (int odd: newOdds) {
    System.out.print(odd + " ");
}
```

Output: 3 5 7 9 11 13 15

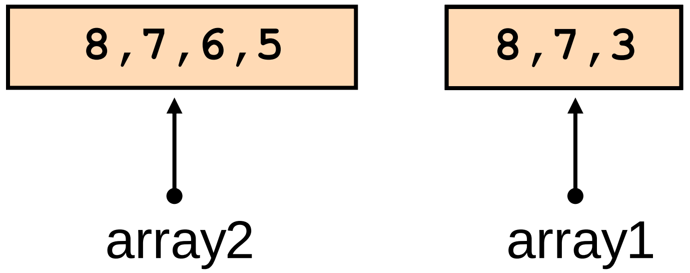
דרכים נוספות להעתקה

- הפונקציה arraycopy במחלקה java.lang.System מאפשרת העתקת תוכנו של מערך אחד לאחר

```
public static void arraycopy(Object src, int srcPos,
                             Object dest, int destPos,
                             int length)
```

```
System.arraycopy(array2, 0, array1, 0, 2);
```

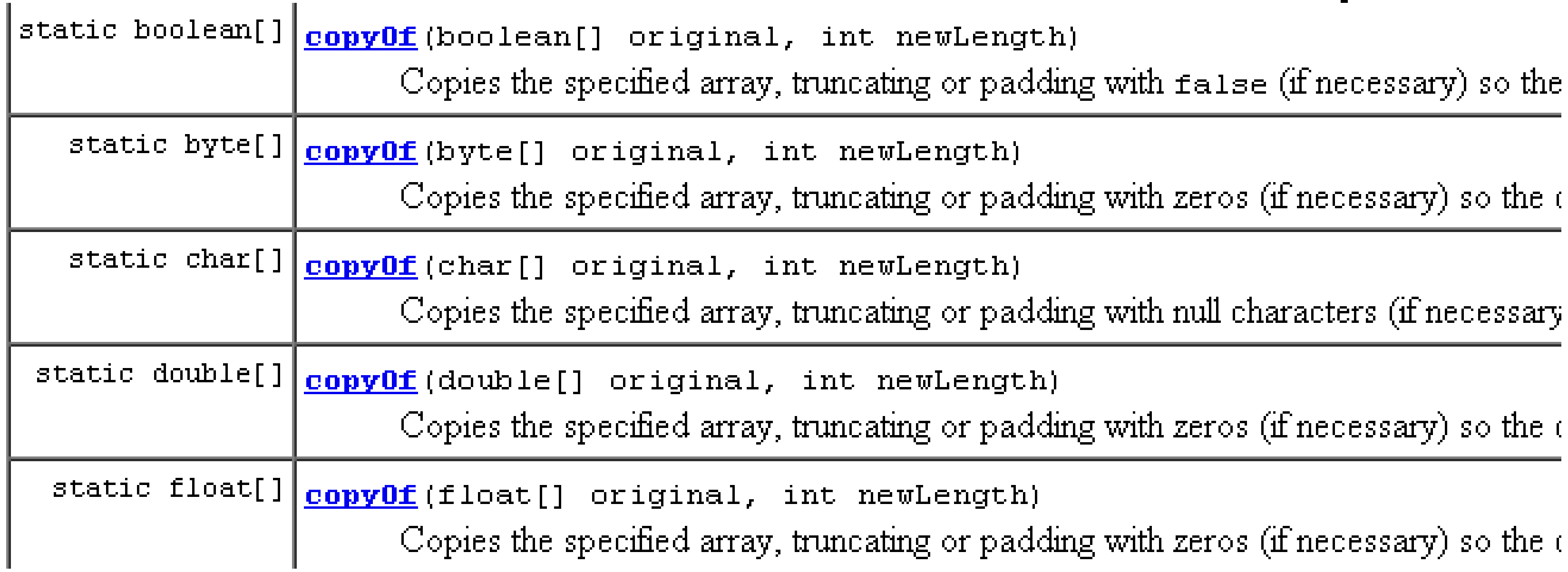
1,2 in array1 are replaced with 8,7



- Details: http://java.sun.com/javase/6/docs/api/java/lang/System.html

העמסה

- חתימה של פונקציה מורכבת משם הפונקציה ומהפרמטרים (מספרם והטיפוס שלהם בלבד) .
- שתי פונקציות נקראות מועמסות (overloaded) אם יש להן אותו שם אבל חתימה שונה

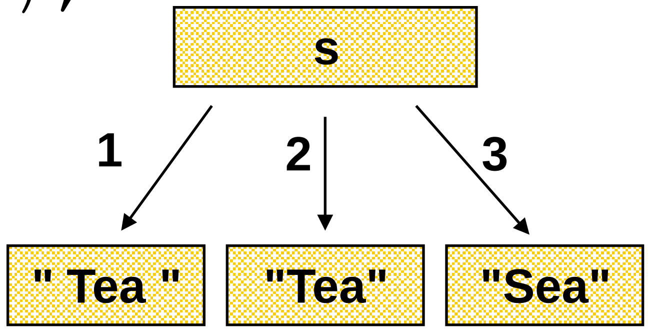


static boolean[]	**copyOf**(boolean[] original, int newLength) Copies the specified array, truncating or padding with `false` (if necessary) so the
static byte[]	**copyOf**(byte[] original, int newLength) Copies the specified array, truncating or padding with zeros (if necessary) so the
static char[]	**copyOf**(char[] original, int newLength) Copies the specified array, truncating or padding with null characters (if necessary
static double[]	**copyOf**(double[] original, int newLength) Copies the specified array, truncating or padding with zeros (if necessary) so the
static float[]	**copyOf**(float[] original, int newLength) Copies the specified array, truncating or padding with zeros (if necessary) so the

מחרוזות

- מרגע שנוצרה המחרוזת היא אינה ניתנת לשינוי (immutable)
  - ההפניה למחרוזת כמובן יכולה להשתנות

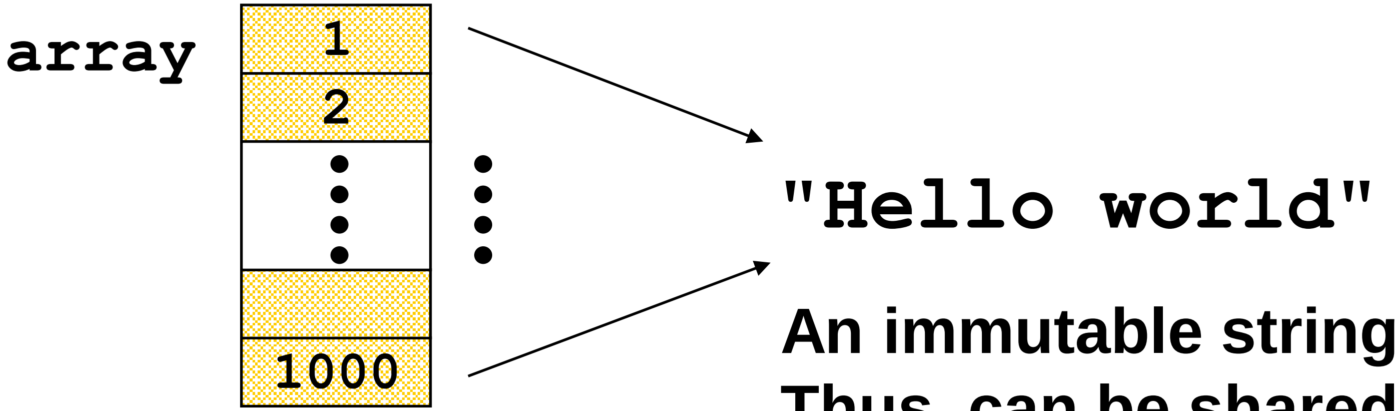
```
String s = " Tea ";
s = s.trim();
s = s.replace('T', 'S');
```



Interning

- מכיוון שמחרוזות הן קבועות ניתן לשתף אותן

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



An immutable string.
Thus, can be shared.

דוגמא ל-Interning

```
String hello = "Hello", lo = "lo";
```

String literals

```
System.out.println(hello == "Hello");

System.out.println(Other.hello == hello);

System.out.println(hello == ("Hel"+"lo"));

System.out.println(hello == ("Hel"+lo));

System.out.println(hello == ("Hel"+lo).intern());
```

דוגמא ל-Interning

```
String hello = "Hello", lo = "lo";

System.out.println(hello == "Hello");

Sy
```

Literal strings within the same class represent references to the same String

```
System.out.println(hello == ("Hel"+"lo"));

System.out.println(hello == ("Hel"+lo));

System.out.println(hello == ("Hel"+lo).intern());
```

דוגמא ל-Interning

```
String hello = "Hello", lo = "lo";

System.out.println(hello == "Hello");

System.out.println(Other.hello == hello);

Sys

System.out.println(hello == ("Hel"+lo));

System.out.println(hello == ("Hel"+lo).intern());
```

Literal strings within different classes represent references to the same String object

דוגמא ל-Interning

```
String hello = "Hello", lo = "lo";

System.out.println(hello == "Hello");

System.out.println(Other.hello == hello);

System.out.println(hello == ("Hel"+"lo"));

Syst
```

Strings computed by constant expressions are computed at compile time and then treated as if they were literals

```
System.out.println(hello == ("Hel"+lo).intern());
```

דוגמא ל-Interning

```
String hello = "Hello", lo = "lo";

System.out.println(hello == "Hello");

System.out.println(Other.hello == hello);

System.out.println(hello == ("Hel"+"lo"));

System.out.println(hello == ("Hel"+lo));

Syst
```

Strings computed by concatenation at run time are newly created and therefore distinct

דוגמא ל-Interning

```
String hello = "Hello", lo = "lo";

System.out.println(hello == "Hello");

System.out.println(Other.hello == hello);

System.out.println(hello == ("Hel"+"lo"));

System.out.println(hello == ("Hel"+lo));

System.out.println(hello == ("Hel"+lo).intern());
```

Explicitly interning a String returns a reference to the interned String object. If such a String was previously interned the retuned value will refer to that object

String Constructors

- Use implicit constructor:

```
String s = "Hello";
```

(string literals are interned)

Instead of:

```
String s = new String("Hello");
```

(causes extra memory allocation)

The StringBuffer Class

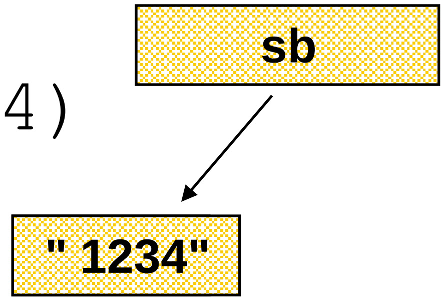
- Represents a **mutable** character string
- Main methods: `append()` & `insert()`
  - accept data of any type
  - If: `sb = new StringBuffer("123")`

    Then: `sb.append(4)`

    is equivalent to

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

    Both yields `"1234"`



StringBuffer 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

StringBuffer 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

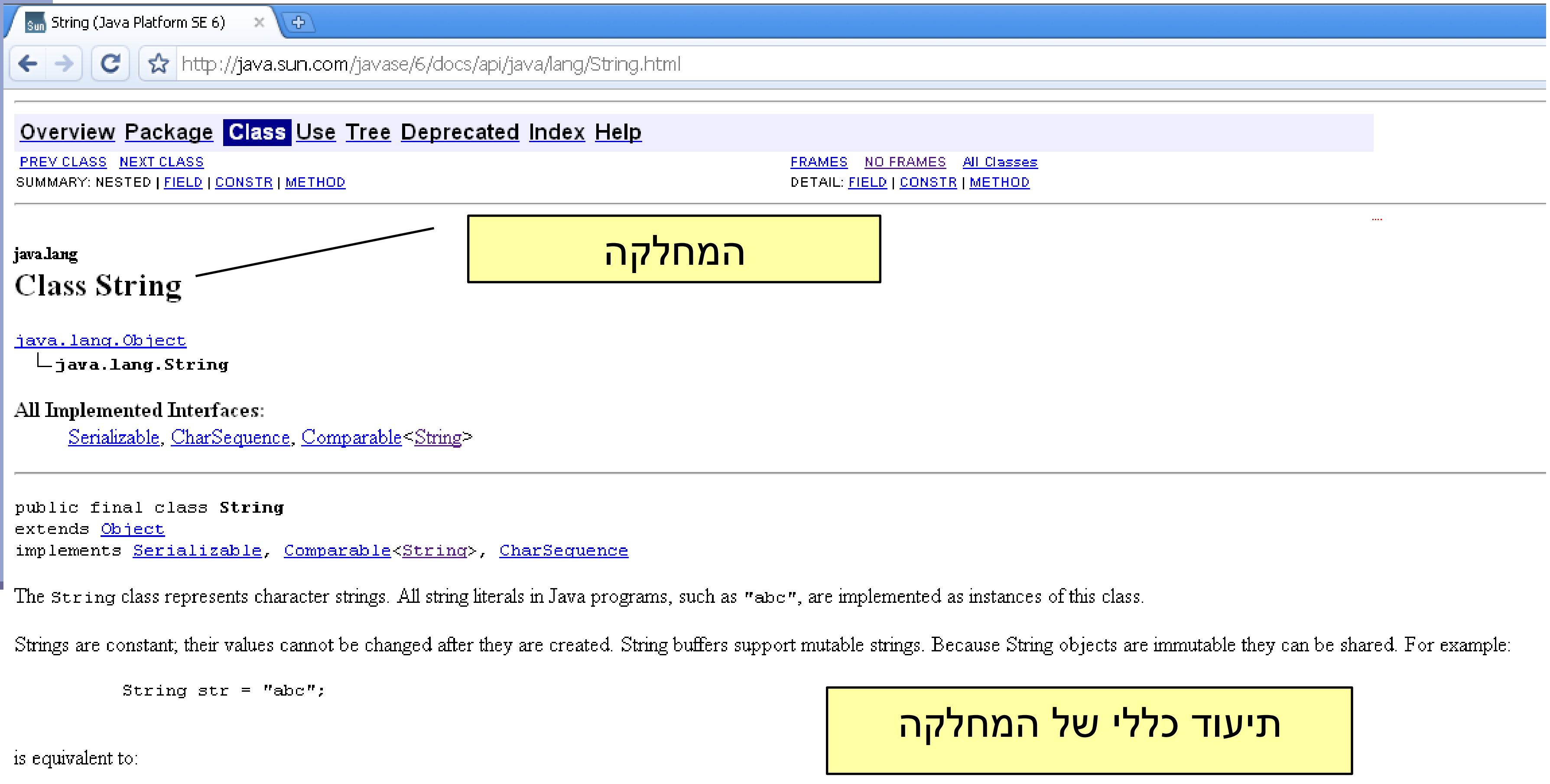
StringBuffer vs. String (cont.)

- 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

כיצד לקרוא Javadoc



String (Java Platform SE 6)

http://java.sun.com/javase/6/docs/api/java/lang/String.html

Overview Package **Class** Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

java.lang

Class String

המחלקה

```
java.lang.Object
  └java.lang.String
```

All Implemented Interfaces:

Serializable, CharSequence, Comparable<String>

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
```

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, are implemented as instances of this class.

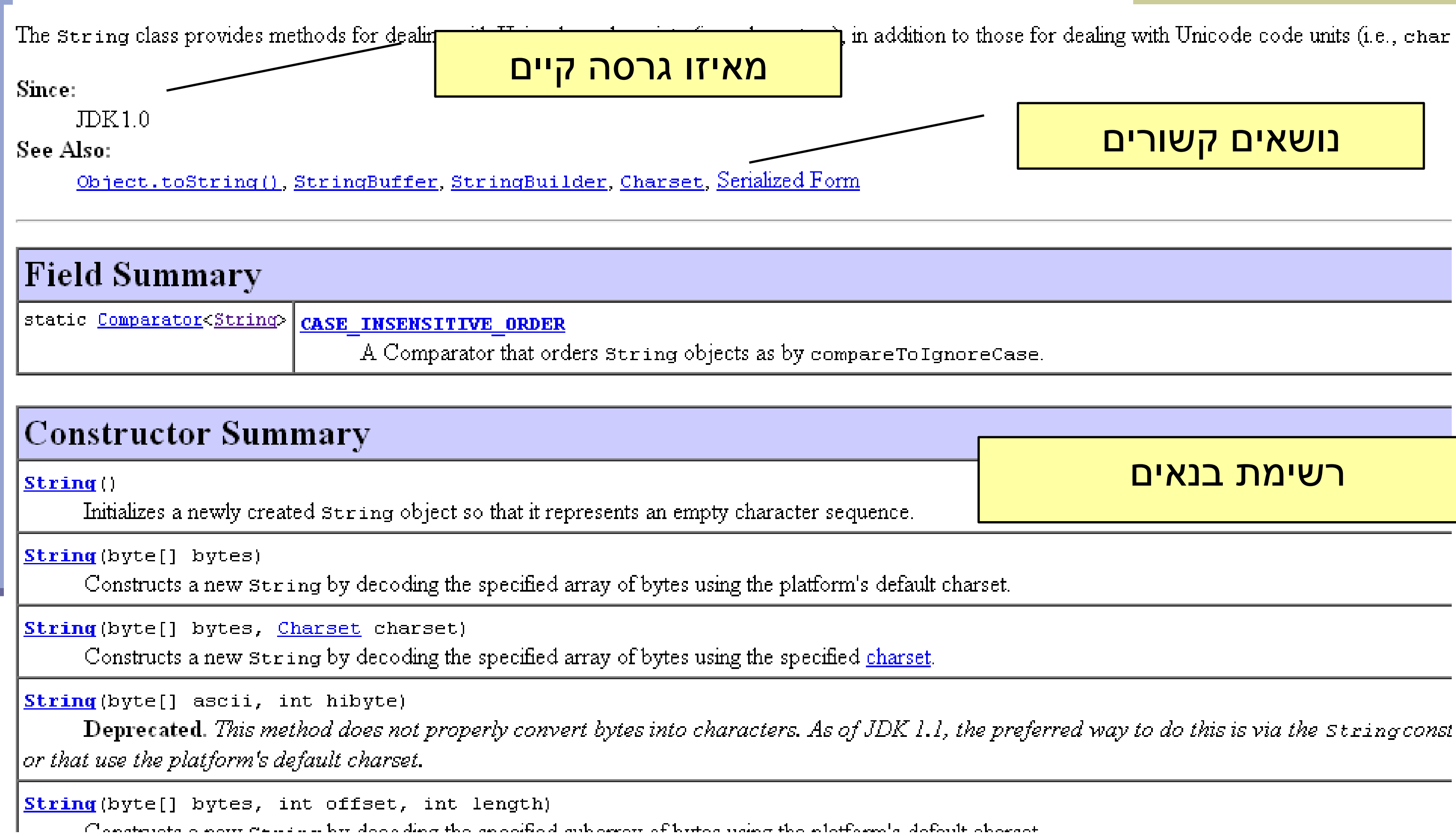
Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
     String str = "abc";
```

is equivalent to:

תיעוד כללי של המחלקה

כיצד לקרוא Javadoc



The String class provides methods for dealin , in addition to those for dealing with Unicode code units (i.e., char

Since:

JDK1.0

See Also:

Object.toString(), StringBuffer, StringBuilder, Charset, Serialized Form

מאיזו גרסה קיים

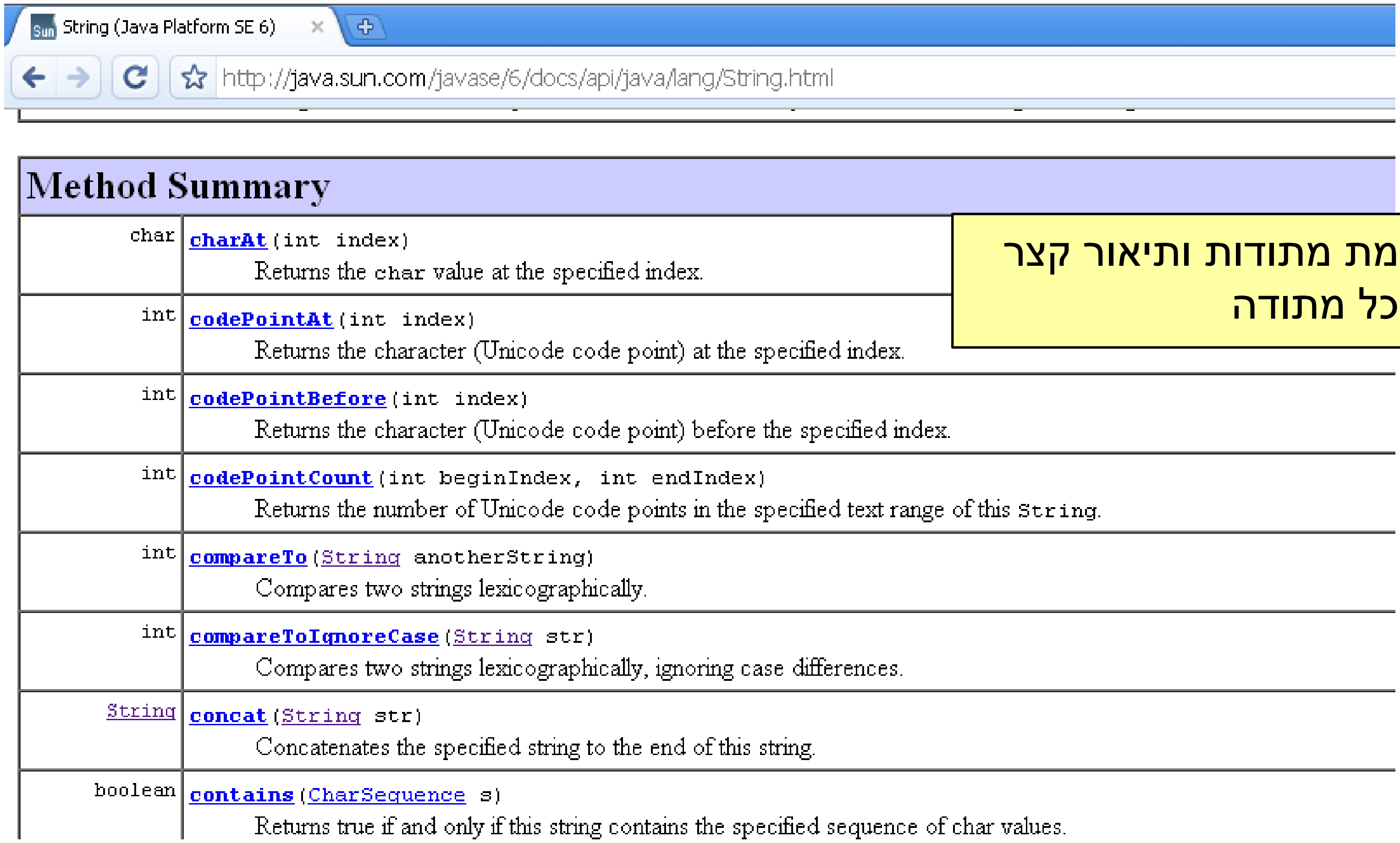
נושאים קשורים

Field Summary	
static Comparator<String>	**CASE_INSENSITIVE_ORDER** A Comparator that orders String objects as by compareToIgnoreCase.

Constructor Summary
String() Initializes a newly created String object so that it represents an empty character sequence.
String(byte[] bytes) Constructs a new String by decoding the specified array of bytes using the platform's default charset.
String(byte[] bytes, Charset charset) Constructs a new String by decoding the specified array of bytes using the specified charset.
String(byte[] ascii, int hibyte) **Deprecated.** *This method does not properly convert bytes into characters. As of JDK 1.1, the preferred way to do this is via the String const or that use the platform's default charset.*
String(byte[] bytes, int offset, int length) Constructs a new String by decoding the specified subarray of bytes using the platform's default charset

רשימת בנאים

כיצד לקרוא Javadoc

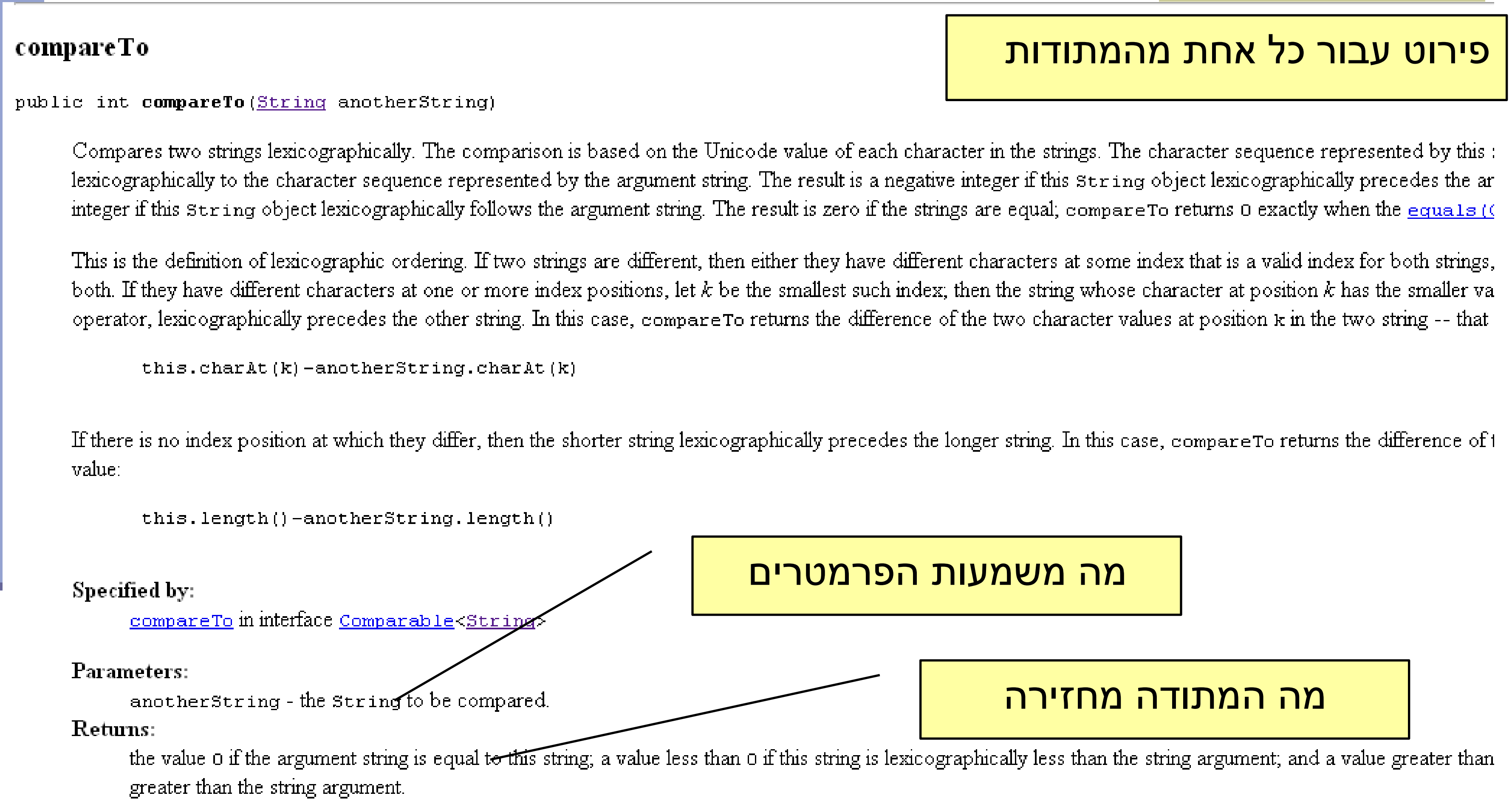

String (Java Platform SE 6)
http://java.sun.com/javase/6/docs/api/java/lang/String.html

Method Summary

char	**charAt**(int index) Returns the char value at the specified index.
int	**codePointAt**(int index) Returns the character (Unicode code point) at the specified index.
int	**codePointBefore**(int index) Returns the character (Unicode code point) before the specified index.
int	**codePointCount**(int beginIndex, int endIndex) Returns the number of Unicode code points in the specified text range of this String.
int	**compareTo**(String anotherString) Compares two strings lexicographically.
int	**compareToIgnoreCase**(String str) Compares two strings lexicographically, ignoring case differences.
String	**concat**(String str) Concatenates the specified string to the end of this string.
boolean	**contains**(CharSequence s) Returns true if and only if this string contains the specified sequence of char values.

רשימת מתודות ותיאור קצר של כל מתודה

כיצד לקרוא Javadoc



compareTo

```
public int compareTo(String anotherString)
```

Compares two strings lexicographically. The comparison is based on the Unicode value of each character in the strings. The character sequence represented by this :
lexicographically to the character sequence represented by the argument string. The result is a negative integer if this `String` object lexicographically precedes the ar
integer if this `String` object lexicographically follows the argument string. The result is zero if the strings are equal; `compareTo` returns `0` exactly when the `equals(`

This is the definition of lexicographic ordering. If two strings are different, then either they have different characters at some index that is a valid index for both strings,
both. If they have different characters at one or more index positions, let *k* be the smallest such index; then the string whose character at position *k* has the smaller va
operator, lexicographically precedes the other string. In this case, `compareTo` returns the difference of the two character values at position `k` in the two string -- that

```
this.charAt(k)-anotherString.charAt(k)
```

If there is no index position at which they differ, then the shorter string lexicographically precedes the longer string. In this case, `compareTo` returns the difference of t
value:

```
this.length()-anotherString.length()
```

Specified by:
`compareTo` in interface `Comparable<String>`

Parameters:
`anotherString` - the `String` to be compared.

Returns:
the value `0` if the argument string is equal to this string; a value less than `0` if this string is lexicographically less than the string argument; and a value greater than
greater than the string argument.

פירוט עבור כל אחת מהמתודות

מה משמעות הפרמטרים

מה המתודה מחזירה