

## Character Strings

```
class String
class StringBuffer
  (Java 1.4 and before : thread-safe)
class StringBuilder
  (Java 5 : thread-unsafe)
(Cf. array of characters)
Based on 2-byte Unicode characters
```

## class String vs class StringBuffer/StringBuilder

- immutable
  - contents of `String` instance unchangeable
  - cannot *insert* or *append* characters
- mutable
  - contents of `StringBuffer` instance modifiable
- fixed length
  - number of characters in the string
- growable
  - grows automatically when characters added
- `s.length()`
  - number of characters in the string
- `sb.length()`
- `sb.capacity()`
  - total allocated capacity

## String literals and concatenation

- `String s = "abc";`
- `System.out.println( s + "def" );`
  - “+” stands for string concatenation.
  - Built-in *operator overload*.
- Left associativity of “+”
  - `(s + 3 + 5)` evaluates to `"abc35"`.
  - `(s + (3 + 5))` evaluates to `"abc8"`.
  - `(5 + 3 + s)` evaluates to `"8abc"`.

## String conversions

- `class Object` supports `toString()`-method to convert a class instance into a printable string form: `classname@hashcode`
- `toString()` can be *overridden*.

```
public String toString() { ... };
```
- `println/print` methods in `System.out`, the “+”-expressions, and the “+=”-statements invoke `toString()` implicitly, to coerce an instance to a string.

## Assignment

```
String s = "abc";
s += 2+3+5;
s.equals("abc10") == true
s.equals("abc235") == false
(Recall that "+" is left-associative.)
```

**Type E1;**

**E1 += E2;**

**E1 = (Type) ((E1) + (E2))**

TYPE	TO STRING	FROM STRING
<b>boolean</b>	<code>String.valueOf(boolean)</code>	<code>new Boolean(String).booleanValue()</code>
<b>int</b>	<code>String.valueOf(int)</code>	<code>Integer.parseInt(String)</code>
<b>long</b>	<code>String.valueOf(long)</code>	<code>Long.parseLong(String)</code>
<b>float</b>	<code>String.valueOf(float)</code>	<code>new Float(String).floatValue()</code>
<b>double</b>	<code>String.valueOf(double)</code>	<code>new Double(String).doubleValue()</code>

## Comparison

- `"abc".equals("abc")` is true.
- `"abc".equalsIgnoreCase("ABC")` is true.
- `"abc" == new String("abc")` is false.
- `s == s` is true.
- `s1.compareTo(s2)`
  - *negative*: s1 lexicographically precedes s2
  - *zero*: s1 is equal s2
  - *positive*: s1 lexicographically follows s2

## Useful Methods

- `String t = "abca";`
- `String s = new String(t);`
- `s.charAt(2)` is 'c'.
- `s.indexOf('a')` is 0.
- `s.indexOf("bc")` is 1.
- `s.indexOf('a', 2)` is 3.
- `s.lastIndexOf('a')` is 3.
- `regionMatches`, `startsWith`, `endsWith`, etc.
  - Pure ASCII applications can be inefficient because Java uses 16-bit Unicode characters.