Software 1 with Java

Recitation No. 4
Strings and Arrays

The String Class

- Represents a character string (e.g. "Hi")
- Implicit constructor:

```
String quote = "Hello World";
```

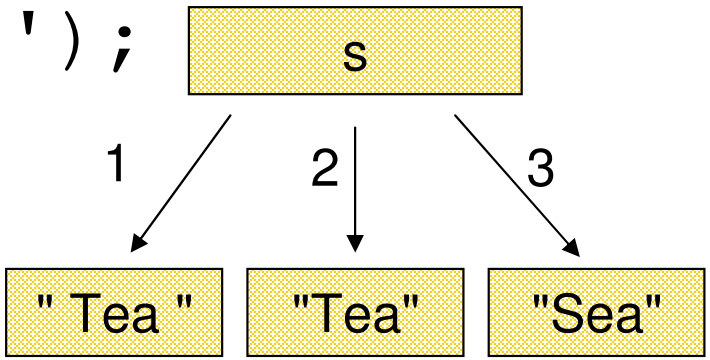
string literal

- All string literals are `String` instances
- `Object` has a `toString()` method
- More details in JDK 5.0 documentation
  http://java.sun.com/j2se/1.5.0/docs/

String Immutability

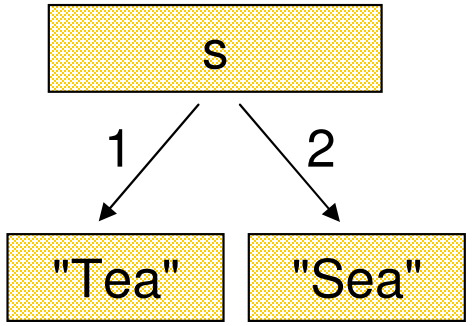
- Strings are constants

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



- A string reference may be set:

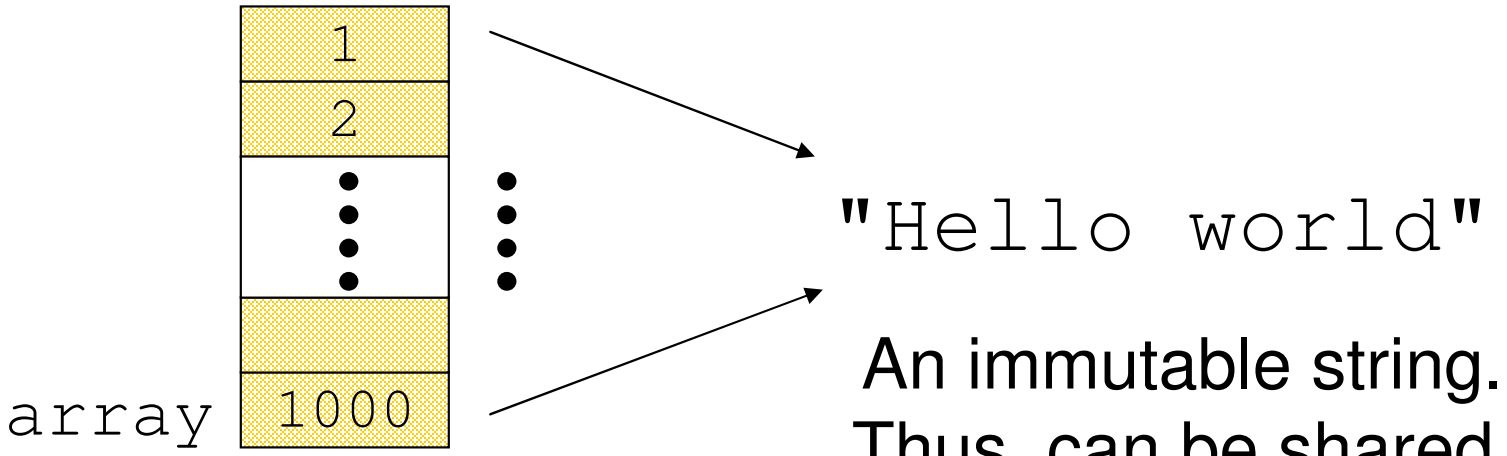
```
String s = "Tea";
s = "Sea";
```



String Interning

- Avoids duplicate strings

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



An immutable string.
Thus, can be shared.

String Interning (cont.)

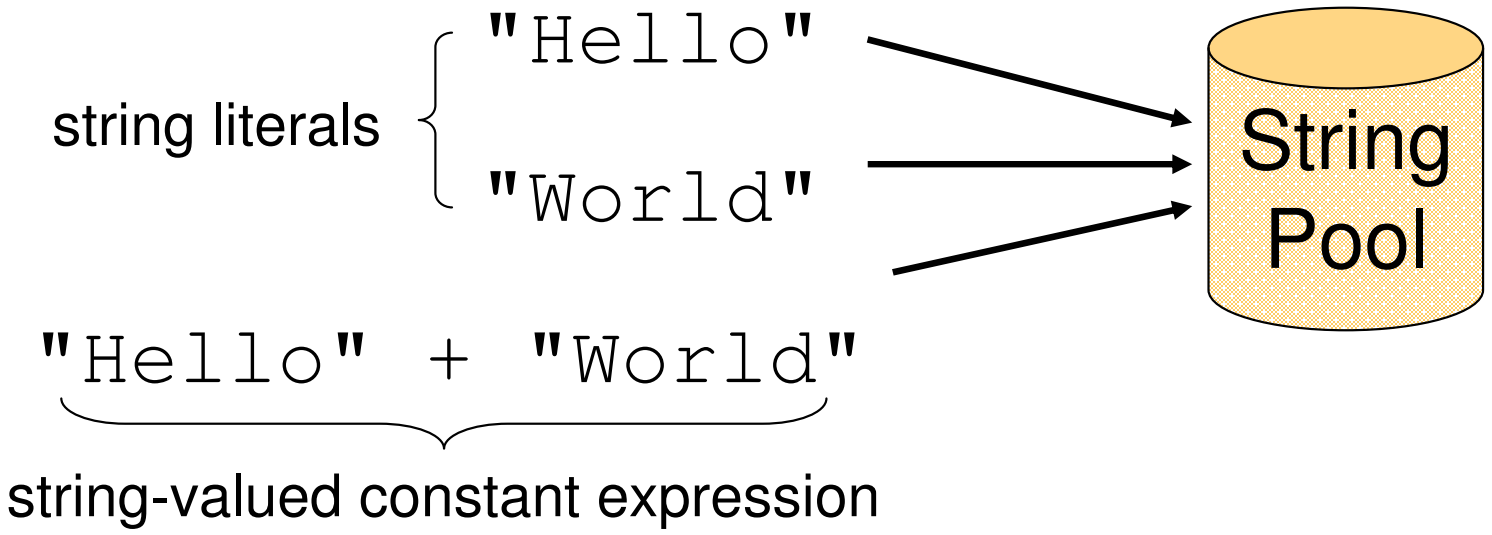
- The `String` class has a static private **pool** of internal strings.
- `myString.intern()` implementation:

if $\exists s \in pool$: *myString.equals(s) == true*
return s;
else
add myString to the pool
return myString;

`equals`: compares characters
`==`: compares references

String Interning (cont.)

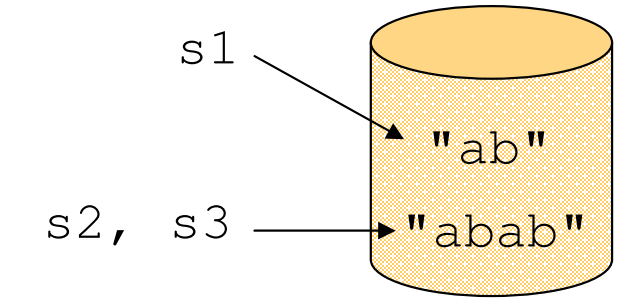
- All string literals and string-valued constant expressions are interned.



String Interning (cont.)

If:

```
String s1 = "ab";
String s2 = "ab" + "ab";
String s3 = "aba" + "b";
String s4 = s1 + s1;
String s5 = s1 + s1;
String s6 = s1 + "ab";
```



Then:

```
s4.equals(s2) is true
(s4 == s2) is false
(s4 == s5) is false
(s2 == s3) is true
(s2 == s6) is false
(s4.intern() == s2) is true
(s4.intern() == s5.intern()) is true
```

s4 → "abab"

s5 → "abab"

s6 → "abab"

String Constructors

- Use implicit constructor:

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

(string literals are interned)

Instead of:

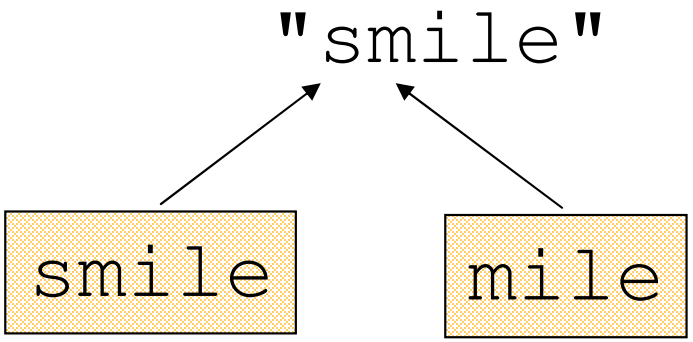
```
String s = new String("Hello");
```

(causes extra memory allocation)

Substrings

- Substrings are created without copying
- `String.substring()` is very efficient

```
String smile = "smile";
String mile = "smile".substring(1,4);
```



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"`

The Concatenation Operator (+)

- String conversion and concatenation:
  - `"Hello " + "World"` is `"Hello World"`
  - `"19" + 8 + 9` is `"1989"`
- Conversion by `toString()`
- Concatenation by `StringBuffer`
- `String x = "19" + 8 + 9;`

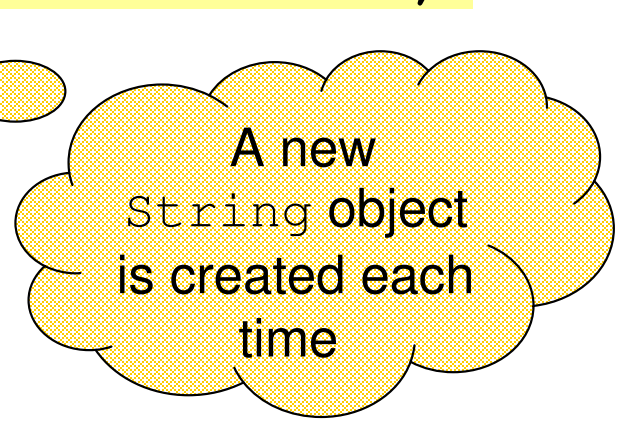
  is compiled to the equivalent of:

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

StringBuffer vs. String

- Inefficient version using String:

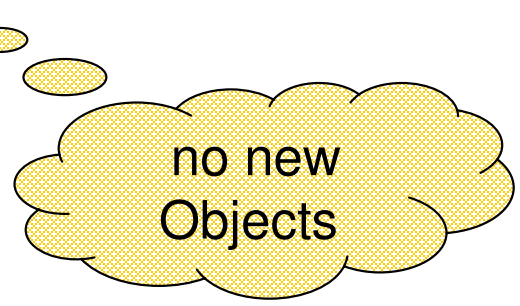
```
public static String
duplicate(String s, int times) {
    String result = s;
    for (int i=1; i<times; i++) {
        result = result + s;
    }
    return result;
}
```



StringBuffer vs. String (cont.)

- More efficient version with StringBuffer:

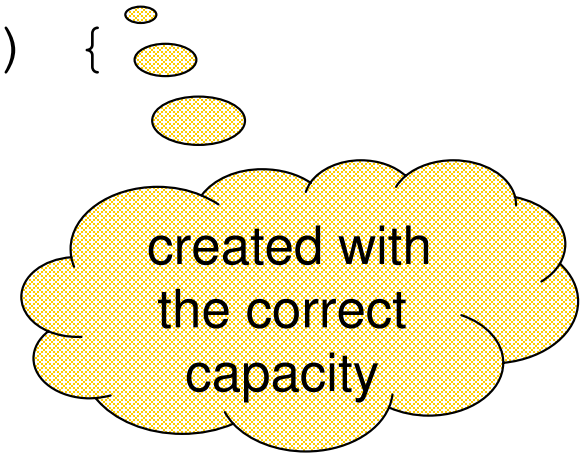
```
public static String
duplicate(String s, int times) {
    StringBuffer result = new StringBuffer(s);
    for (int i=1; i<times; i++) {
        result.append(s);
    }
    return result.toString();
}
```



StringBuffer vs. String (cont.)

- Much 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();
}
```

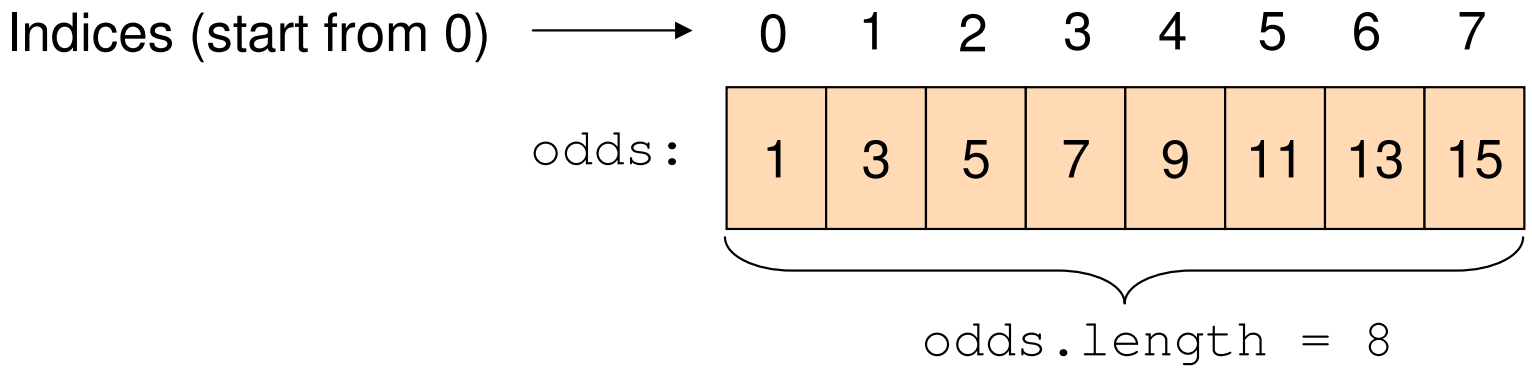


StringBuffer vs. StringBuilder

- `StringBuilder` class provides an API compatible with `StringBuffer`, but with no guarantee of synchronization.
- This class is designed for use as a drop-in replacement for `StringBuffer` in places where the string buffer was being used by a single thread (as is generally the case).
- Where possible, it is recommended that this class be used in preference to `StringBuffer` as it will be faster under most implementations.

Arrays

- **Array:** A fixed-length data structure for storing multiple values of the same type
- Example: An array of odd numbers:

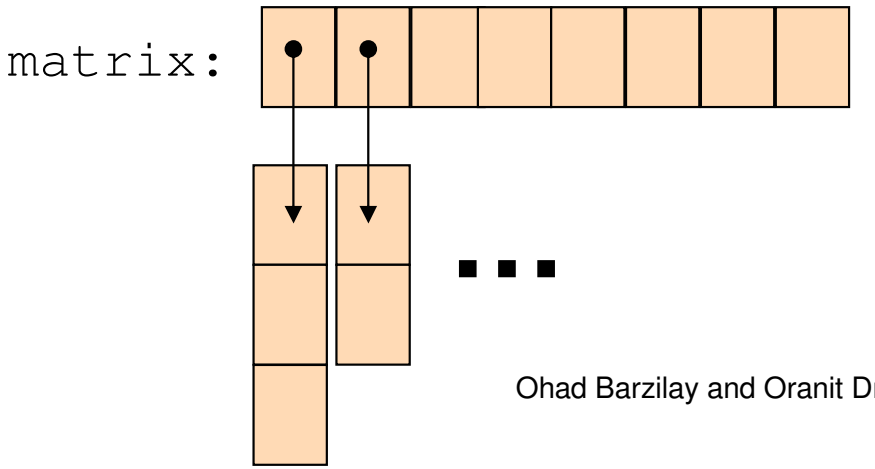


The type of all elements is `int`

The value of the element at index 4 is 9: `odds[4] == 9`

Array Declaration

- An array is denoted by the `[]` notation
- Examples:
  - `int[] odds;`
  - `int odds[];` // legal but discouraged
  - `String[] names;`
  - `int[][] matrix;` // an array of arrays



Array Creation and Initialization

- What is the output of the following code:

```
int[] odds = new int[8];
for (int i=0 ; i < odds.length ; i++) {
  System.out.print(odds[i] + " ");
  odds[i] = 2*i+1;
  System.out.print(odds[i] + " ");
}
```

Array creation: all elements get the default value for their type (`0` for `int`)

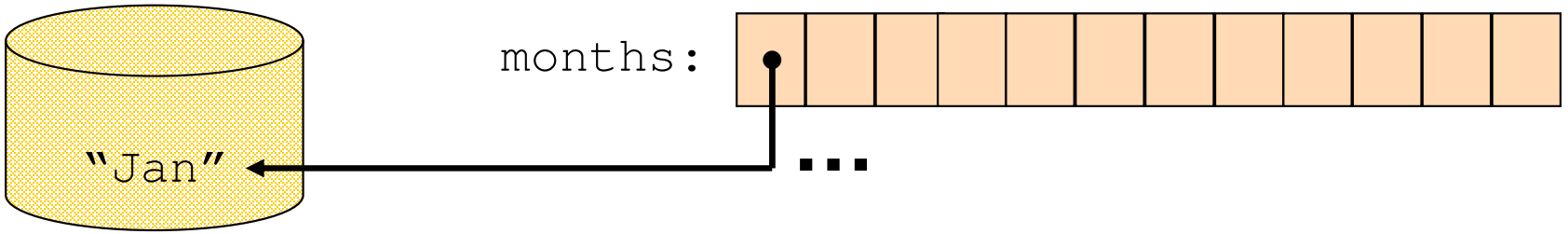
- Output:

```
0 1 0 3 0 5 0 7 0 9 0 11 0 13 0 15
```

Array Creation and Initialization

- Creating and initializing small arrays with *a-priori* known values:
  - `int[] odds = {1,3,5,7,9,11,13,15};`
  - `String months[] =`

```
{"Jan", "Feb", "Mar", "Apr",
 "May", "Jun", "July", "Aug",
 "Sep", "Oct", "Nov", "Dec"};
```



Loop through Arrays

- By promoting the array's index:

```
for (int i=0 ; i < months.length ; i++) {
     System.out.println(months[i]);
}
```

- In Java 5.0:

```
for (String month: months) {
     System.out.println(month);
}
```

The variable month is assigned the next element in each iteration

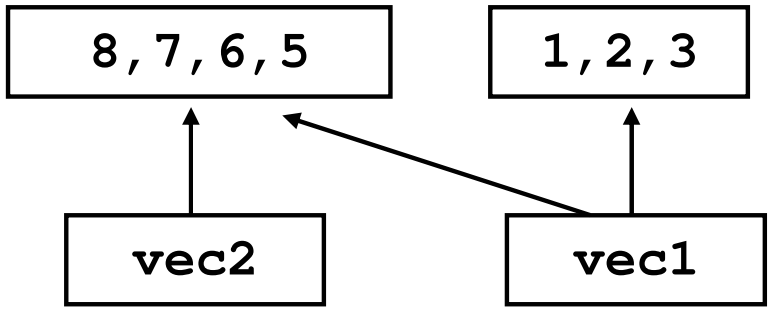
Manipulating arrays

- The java.util.Arrays class has methods for sorting and searching arrays e.g.
  - public static void **sort**(int[] a)
  - public static int **binarySearch**(int[] a, int key)
- In the java.lang.System class:
  - public static void **arraycopy**(Object src, int srcPos, Object dest, int destPos, int length)

Copying Arrays

- Naïve copy:

```
int vec1[] = {1,2,3};
int vec2[] = {8,7,6,5};
vec1 = vec2;
```



- Doing it "by hand" is inefficient
- **`arraycopy`**`(vec2, 0, vec1, 0, 3);`
  will replace 1,2,3 in `vec1` with 8,7,6

Manipulating arrays (cont.)

- What is the output of the following code:

```
int[] odds = {1,3,5,7,9,11,13,15};
int[] newOdds  = new int[8];
System.arraycopy(odds, 1, newOdds, 1, 7);
for (int odd: odds) {
     System.out.print(odd + " ");
}
```

- Output:

```
0 3 5 7 9 11 13 15
```

Reading command line arguments

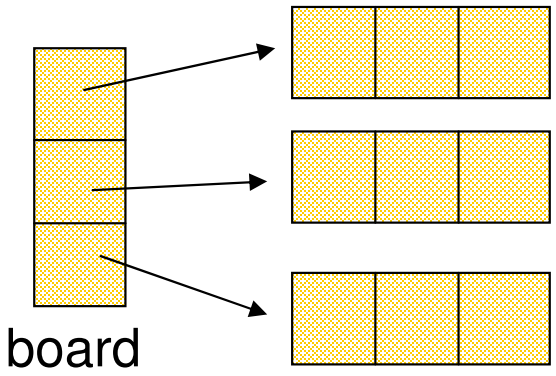
```
class CommandLineDemo
{
   public static void main(String args[])
   {
        for (int i=0; i<args.length; i++)
          System.out.println("Argument number " + (i+1)
               + " is : " + args[i]);
   }
}

> java CommandLineDemo Apple      Box Car
Argument number 1 is : Apple
Argument number 2 is : Box
Argument number 3 is : Car
```

2 Dimensional Arrays

- There are no 2D arrays in Java but you can build array of arrays:

```
char[] board[] = new char[3][];
for (int i = 0; i<3; i++)
    board[i] = new char[3];
```



Or equivalently:

```
char[] board[] = new char[3][3];
```

2 Dimensional Arrays (con't)

Building a Tic-Tac-Toe board:

```
for (int i = 0; i < 3; i++)
   for (int j = 0; j < 3; j++)
       board[i][j] = (i + j) % 2 == 0 ? 'x' : 'o';

for (int i = 0; i < 3; i++) {
   for (int j = 0; j < 3; j++) {
       System.out.print(board[i][j] + " ");
   }
   System.out.println(" ");
}
```