Software 1

Recitation No. 7
Strings and Constructors

Lior Shapira and Ohad Barzilay

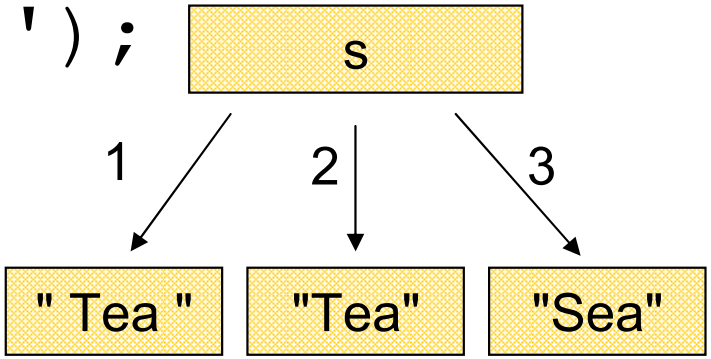
היום בתוכנית

- השלמת התרגול הקודם – מבני נתונים
- על מחרוזות מקובעות ושאינן מקובעות
- אתחול עצמים ואלגוריתם הבנאים

String Immutability

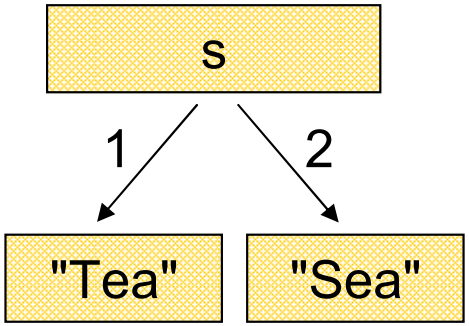
- Strings are constants

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



- A string reference may be set:

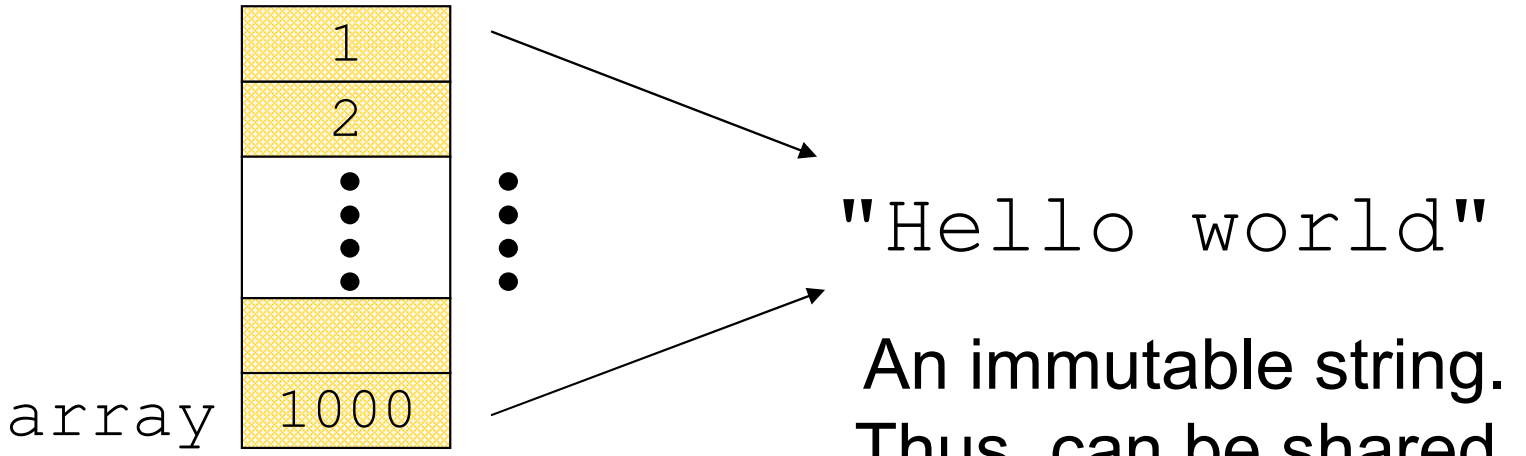
```
String s = "Tea";
s = "Sea";
```



String Interning

- Avoids duplicate strings

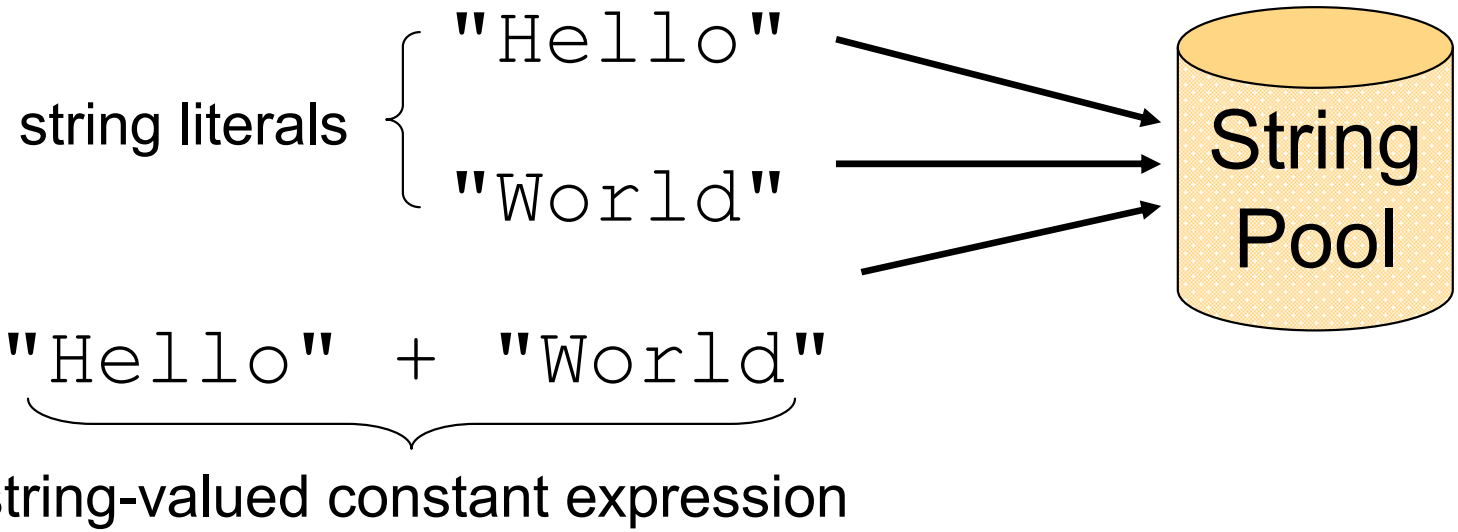
```
String[] array = new String[1000];
for (int i=0 ; i<1000 ; i++) {
   array[i] = "Hello world";
}
```



An immutable string.
Thus, can be shared.

String Interning (cont.)

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



String Constructors

- Use implicit constructor:

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

(string literals are interned)

Instead of:

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

(causes extra memory allocation)

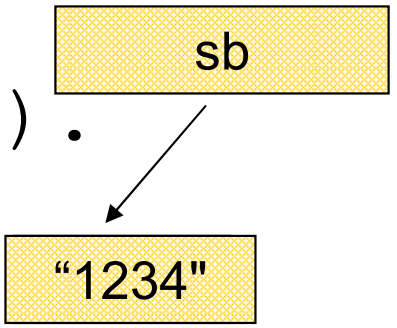
The StringBuffer Class

- 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"`



The Concatenation Operator (+)

- String conversion and concatenation:
  - `"Hello " + "World"` is `"Hello World"`
  - `"19" + 8 + 9` is `"1989"`

- Concatenation by `StringBuffer`
  - `String x = "19" + 8 + 9;`

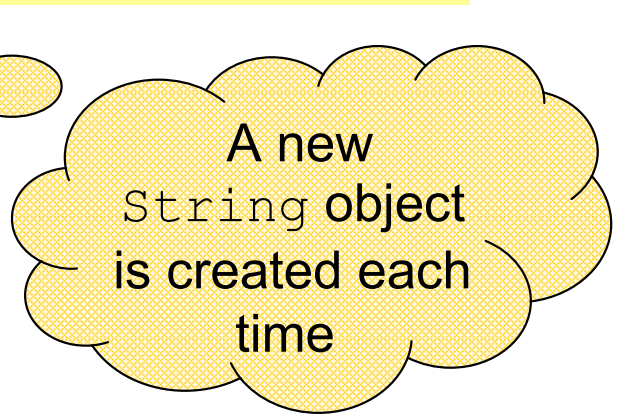
    is compiled to the equivalent of:

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

StringBuffer vs. String

- Inefficient version using String:

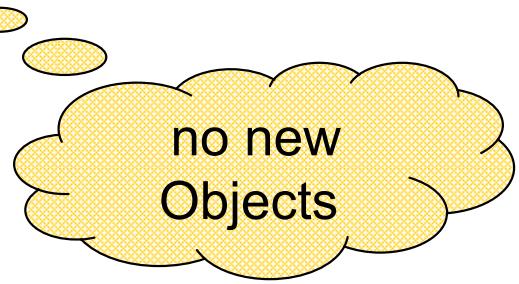
```
public static String
duplicate(String s, int times) {
    String result = s;
    for (int i=1; i<times; i++) {
        result = result + s;
    }
    return result;
}
```



StringBuffer vs. String (cont.)

- More efficient version with StringBuffer:

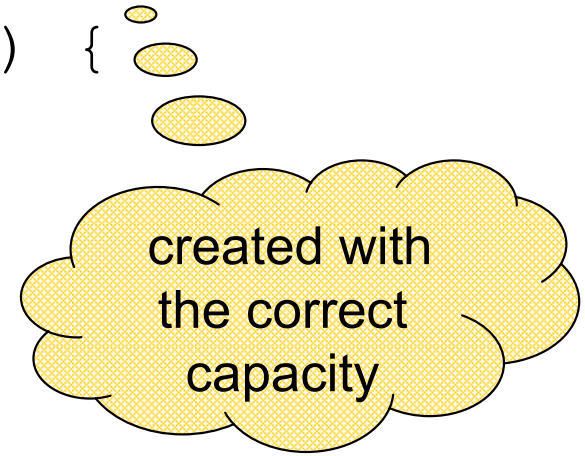
```
public static String
duplicate(String s, int times) {
    StringBuffer result = new StringBuffer(s);
    for (int i=1; i<times; i++) {
        result.append(s);
    }
    return result.toString();
}
```



StringBuffer vs. String (cont.)

- 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();
}
```



StringBuilder vs. StringBuffer

- StringBuilder has the same API as StringBuffer, but with no guarantee of synchronization.
- StringBuilder is a replacement for StringBuffer when there is only a single thread
- Where possible, it is recommended to use StringBuilder as it will be faster under most implementations.

אתחולים ובנאים

אתחולים ובנאים

- יצירת מופע חדש של עצם כוללת: הקצאת זכרון, אתחול, הפעלת בנאים והשמה לשדות
- במסגרת ריצת הבנאי נקראים גם הבנאי/ים של מחלקת הבסיס
- תהליך זה מבלבל כי לשדה מסוים ניתן לבצע השמות גם ע"י אתחול, וגם ע"י מספר בנאים (אחרון קובע)
- בשקפים הבאים נתאר במדויק את התהליך
- נעזר בדוגמא

מה הסדר ביצירת מופע של מחלקה?

1. **שלב ראשון**: הקצאת זיכרון לשדות העצם והצבת ערכי ברירת מחדל

2. **שלב שני**: נקרא הבנאי (לפי חתימת new) והאלגוריתם הבא מופעל:

1. Bind constructor parameters.
2. If explicit this(), call recursively, and then skip to Step 5.
3. Call recursively the implicit or explicit super(...), except for Object because Object has no parent class.
4. Execute the explicit instance variable initializers.
5. Execute the body of the current constructor.

דוגמא

```
public class Object {

   public Object() {}
   // …
}
```

```
public class Employee  {

   private String name;
   private double salary = 15000.00;
   private Date birthDate;

   public Employee(String n, Date DoB) {

      name = n;
      birthDate = DoB;
   }

   public Employee(String n) {
      this(n, null);
   }

}
```

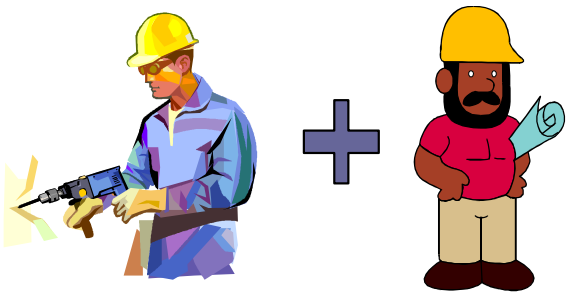
```
public class Manager extends Employee {

   private String department;

   public Manager(String n, String d) {
      super(n);
      department = d;
   }

}
```

הרצת הדוגמא

- מה קורה כאשר ה JVM מריץ את השורה

Manager m = new Manager(“Joe Smith”, “Sales”);

- שלב ראשון: הקצאת זיכרון לשדות העצם והצבת ערכי ברירת מחדל



(String)Name
(double)Salary
(Date)Birth Date
(String)Department

תמונת הזכרון

- **Basic initialization**
  - Allocate memory for the complete Manager object
  - Initialize all instance variables to their default values
- **Call constructor: Manager("Joe Smith", "Sales")**
  - Bind constructor parameters: n="Joe Smith", d="Sales"
  - No explicit this() call
  - Call super(n) for Employee(String)
    - Bind constructor parameters: n="Joe Smith"
    - Call this(n, null) for Employee(String, Date)
      - Bind constructor parameters: n="Joe Smith", DoB=null
      - No explicit this() call
      - Call super() for Object()
        - No binding necessary
        - No this() call
        - No super() call (Object is the root)
        - No explicit variable initialization for Object
        - No method body to call
      - Initialize explicit Employee variables: salary=15000.00;
      - Execute body: name="Joe Smith"; date=null;
    - Steps skipped
    - Execute body: No body in Employee(String)
  - No explicit initializers for Manager
  - Execute body: department="Sales"

1. Bind parameters.
2. If explicit this(), goto 5.
3. super(),
4. explicit var. init.
5. Execute body

(String) Name	"Joe Smith"
(double) Salary	15000.0
(Date) Birth Date	null
(String) Department	"Sales"

```
public class Employee extends Object {
  private String name;
  private double salary = 15000.00;
  private Date birthDate;

  public Employee(String n, Date DoB) {
    // implicit super();
    name = n;
    birthDate = DoB;
  }
  public Employee(String n) {
    this(n, null);
  }
}
```

```
public class Manager extends Employee {
  private String department;
  public Manager(String n, String d) {
    super(n);
    department = d;
  }
}
```