Lecture 4

Sequences of characters, such as ``Hello'', ``Bye!'', ``The answer is 42'' and ``A'', are objects. They belong to the String class for which there is a predefined class definition in the Java library. But this is such a useful class that there are a few optimisations that make it different from all other classes.

We will write a program that repeatedly redisplays a name with an ever growing prefix in uppercase.

The String class

- To create new String objects, e.g.:

```
String s1;
s1 = new String("Hello");

String s2 = new String("Bye!");
```

- Character positions within a String are indexed starting from 0:

	0	1	2	3	4		
“	H	e	l	l	o	”	(length is 5)

- Contrast s3, which is the **empty** String (length 0) with s4, which is not empty:

```
String s3 = new String("");
String s4 = new String(" ");
```


String interface

- `public int length()`
- `public String concat(String str)`
- `public int indexOf(String str)`
- `public String toLowerCase()`
- `public String toUpperCase()`
- `public String substring(int beginIndex)`
- `public String substring(int beginIndex, int endIndex)`
- And many more

Example messages

- What do these produce?

```
String s5 = new String("testing");
String s6 = new String(" and ");
String s7 = new String("debugging");

String s8 = s7.substring(
                s7.indexOf("bug"));

String s9 = s7.concat(s6).concat(s5);
```

- String objects are immutable: they aren't changed by methods. Instead, new objects get created:

```
String s10 = s5.toUpperCase();
System.out.println(s5);
System.out.println(s10);
```


Strings are special

- String objects are so common that their creation has a short-cut.

- E.g. instead of

```
String s11 = new String("development");
```

we can just write

```
String s11 = "development";
```


Concatenation: an optimisation

- There is a concatenation method:

```
String s16 = "ha";

String s17 = s16.concat(s16);
```

- But this is so common that there is an operator we can use instead:

```
String s17 = s16 + s16;
```

Concatenation using +

- If you mix `String`s and other values in a concatenation expression, Java converts the other values to `String`s.

```
String s18 = "This is CS" + 2000;
```

— `s18` refers to a `String` object whose contents are `"This is CS2000"`

- Very useful in messages that invoke `println`:

```
System.out.println("Orange's redness: " +
    Color.orange.getRed());
```

— displays `Orange's redness:  255`

- But, be careful! Java evaluates multiple + operators from left-to-right unless parentheses dictate otherwise. . .

. . . So what do these produce?

```
System.out.println(
   "You also study CS" + 20 + 10);

System.out.println(
   "You might also like CS" + (20 + 20));


System.out.println(
   9 + 9 + "% of students like both");


System.out.println("" + 1 + 0 +
   "% of students like everything");
```


The `JLabel` class

- One way to add text to a window is to use the `JLabel` class definition (`javax.swing.JLabel`).

- One way to create:

```
JLabel lb = new JLabel("A string");
```

- One method in its interface:

```
lb.setText("A new string");
```

This changes the label from ``A string'' to ``A new string''.

Example program

- Suppose `f` is a `JFrame`. The part of the window where we display content is called the content pane. To get a reference to `f`'s content pane:

```
Container contentPane =
         f.getContentPane();
```

(Must include `import java.awt.Container`)

- To add a `JLabel` (`lb`) to the content pane:

```
contentPane.add(lb);
```

- We will write a program that displays a name and repeatedly redisplays it with successively bigger prefixes in uppercase.