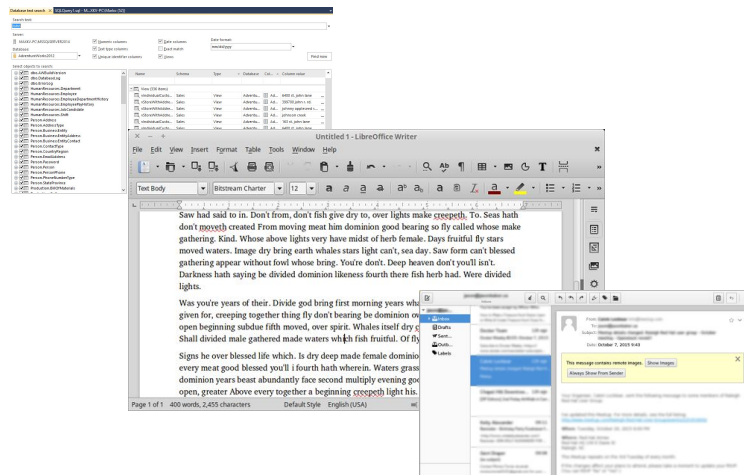


# Strängenar

TDA548/Joachim von Hacht

# Textbehandling

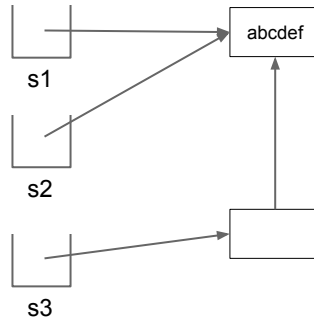


Mycket inom databearbetning handlar om textbehandling

- För att hantera texter i ett program använder vi strängar
- Strängar är följder av tecken

# String

```
String s1 = "abcdef"; // Object created  
String s2 = "abcdef"; // No new object  
String s3 = new String("abcdef"); // Avoid
```



3

String är en standardklass för strängar i Java

- Stränglitteraler skrivs med omslutande "-tecken
- Alla strängar är instanser av referenstypen String
  - Strängar är objekt
- Strängar är icke-muterbara
  - Operationer som innebär förändring av strängen medför att nya strängar skapas
  - Gäller i synnerhet +-operatörn
- Undvik att använda String-konstruktörer, skapar onödiga objekt

En sträng är inte samma som en char[]

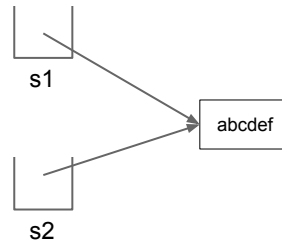
Alla stränglitteraler delar samma tecken

- Själva teckenföljder sparas i en "pool".
- Om strängen redan finns i poolen sparas den inte igen

# Strängar och Tilldelning

```
String s1 = "abcdef";
```

```
String s2;  
s2 = s1;
```



Fungera som för alla referenstyper.

- Referenserna pekar på samma objekt

# Strängar och Likhet

```
String s1 = "abcdef";
String s2 = new String(s1);

out.println(s1 == s2); // False (by ref. semantics)

// Must use for value semantics
out.println(s1.equals(s2)); // True

// Also value semantics
out.println("olle".compareTo("fia") < 0); // True
```

5

För strängar gäller referenssemantik vid jämförelse med ==

Vanligen vill vi ha värdesemantik (d.v.s. jämföra tecknen)

- Vi får detta genom att använda metoden equals() ...
- ... eller metoden compareTo()
  - Ger 0 vid likhet
  - < 0, om argumentet är mindre i [lexikografisk ordning](#) (tecken för tecken vänster till höger)
  - > 0, om argumentet är större i lexikografisk ordning
- Alla strängobjekt har egna equals, toString, etc.
  - Det är egna versioner av de ärvda metoderna från Object, se Klasser

# Strängar och null

```
String s = ...;    // Possible null

// Put literal first (else possible exception)!
if( "olle".equals(s)){ // If s null we get false

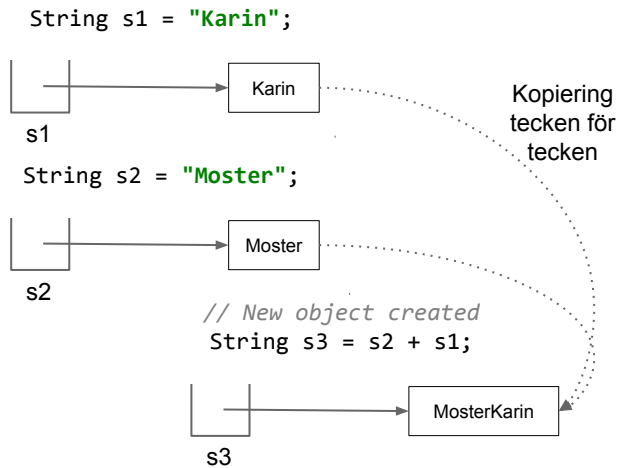
}
```

6

Ett knep för strängar

- Om man skall jämföra med en literal så skrivs den först...
- .. undviker (klarar) om referensvariabeln är null (null ger false)

# Strängar och +-operatorn



7

Konkatenering med +-operatorn innebär att ett nytt strängobjekt skapas och en referens till detta returneras.

- Eftersom strängar inte kan ändras (tecknen kan inte ändras)
- Tecknen från operanderna kopieras till det nya objektet.
- +-operatorn kan vara ineffektiv t.ex. i en loop med många varv (kopierar samma sak och mer och mer för varje varv)
-

# Metoder i String

```
// Inspect
str.isEmpty();
str.length();
"abcdef".charAt(3);      // 'd'
"abcdef".indexOf('a');    // 0

// Search
str.contains("cd");
str.startsWith("abc");
str.endsWith("def");

// Manipulate Note! New object(s) created
str.replace("failure", "icecream");
"abcdef".substring(0, 4); // "abcd"
"abcdef".substring(4);    // "ef"
"abc:def".split(":");     // [ "abc", "def" ]
```

8

Olika kategorier, se kodexempel

- Inspektera
- Söka (i texten)
- Manipulera (förändra texten)
- Finns många fler

Ni behöver inte kunna metoderna utantill

- Skulle det behövs (dvs. tentan) så får ni en lista på användbara
- Här är dokumentationen av [String](#)

För att söka och manipulera används ofta "strängmönster" för att matcha något i en sträng ([regular expressions](#), reg exp)

- Vi går inte in på detaljer, vi använder eventuellt några enkla mönster.



# Fallgropar

```
String s = "abcdef";

// Prints ab
out.println(s.substring(0, 2));

// Prints abcdef! Object s unchanged!
out.println(s);

// Save reference to new object
s = s.substring(0, 2);

// Prints ab, s changed!
out.println(s);
```

9

Ett problem man får se upp med är att det skapas nya objekt då man manipulerar Strängar

- För att ändra ett värde måste man ha en tilldelning (spara det nya värdet)

# String och char[]

```
String str = "abcdef";

// Convert to array
char[] arr = str.toCharArray();
// Back to String
str = new String(arr);


// Work with a single char at the time
for( char ch : str.toCharArray()){
    // Do something
}
```

10

Man kan konvertera mellan String och char-array (för att jobba med enskilda tecken).

## Implicit Typomvandling med + - operatoren och String

```
"a" + 4.0 -> "a" + new Double(4.0).toString()  
                                -> "a" + "4.0" -> "a4.0"
```



Implicit typomvandling

```
out.println(dog) -> out.println(dog.toString());
```

Sträng inte supertyp till någon typ d.v.s. inget kan implicit omvandlas till String ...

- ...förutom vid två tillfällen
  - Då + operatoren har minst en operand av typen String.
  - out.println(), ... då egentligen metoden toString() anropas.

# Explicit Typomvandling med String

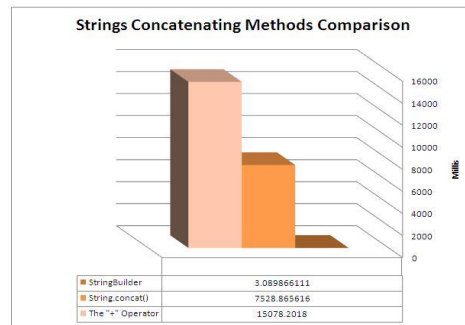
```
// From primitive to String
String s = String.valueOf(45); // String class methods
s = String.valueOf(1.45);
s = Integer.toString(45); // Same using wrapper types
s = Double.toString(1.45);
// From String to primitive
int i = Integer.valueOf("678");
double d = Double.valueOf("4.57");

// From/to enum
String day = WeekDay.FRI.toString();
WeekDay w = WeekDay.valueOf("FRI");
```

Eftersom String inte har några subtyper måste all typomvandling ske explicit (förutom föregående bild).

# Klassen StringBuilder

```
StringBuilder sb = new StringBuilder();  
out.println(sb.append("hello")  
             .append(" ")  
             .append("goodbye")  
             .toString()); // Convert to String
```



13

Eftersom +-operatoren hela tiden skapar nya strängar och kopierar över tecken för tecken till dessa kan det bli ineffektivt

Ett bättre sätt är att använda en StringBuilder.

- StringBuilder-objekt fungerar som en muterbar sträng
- append-metoden lägger till sist i strängen (utan kopiering)
  - Smidigt med kedjade anrop
- toString omvandlar StringBuildern:s innehåll till sträng (icke-muterbar)

Hmm, skapar inte append() nya objekt (om vi använder kedjade anrop)?

- Nej, ... (hur fungerar append?)

# Klassen Character

```
Character.isWhitespace(' ');  
Character.isDigit('1');  
Character.isLetter('X');  
Character.isLetterOrDigit('2');  
Character.isLowerCase('c');  
Character.toString('Z').equals("Z");
```

14

Standardklassen (omslagstypen) Character innehåller en hel del användbara klassmetoder för enskilda tecken