

Klassen String

- Representerar en teckensträng – t.ex. "abc"
- Objekt av klassen är **Immutable** – oföränderlig
- Enda klass det går att instatiera objekt av på två sätt

String x = "abc";

eller

String x = new String("abc");

- Varför? – göra det enklare för programmeraren

Hur fungerar det egentligen?

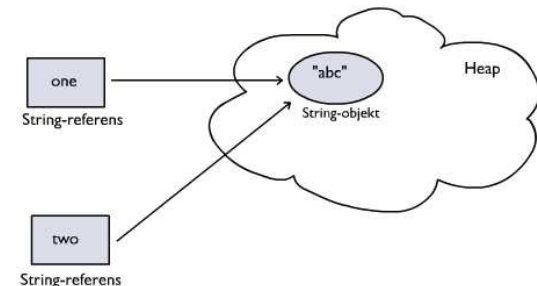
- **String one = "abc";**
Skapar ett nytt objekt med värdet "abc" och låter **one** referera till det objektet
- **String two = one;**
Skapar en till referens till samma objekt

----- Vad händer nu? -----

- **two = two.concat("def");**
Ett helt nytt objekt skapas. Det gamla ändras **inte**

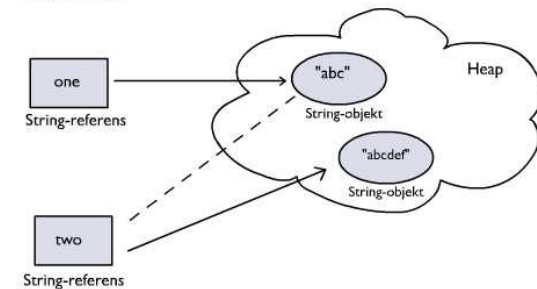
String one = "abc";

String two = one;



String one = "abc";

String two = two.concat("def");



String – lagring i minnet

- Java Virtual Machine har en speciell **String constant pool** som lagrar String-konstanter (t.ex. abc)

Exempel:

String one = "abc"; - Kompilatorn ser efter om String-konstanten "abc" finns i "poolen"

Om det finns:

Låt **one** referera till den String-konstanten som finns i "poolen" – skapa inget nytt String-objekt

Om det inte finns:

Skapa ett **nytt** String-objekt med värdet "abc"

Lagra det objektet i "poolen" och låt **one** referera till det

Skapa en String med *new*

- **String one = new String("abc");**
Skapar ett nytt String-objekt med värdet "abc" **utanför** "poolen" och låter one referera till det

Men...

Lägger **också** till ett String-objekt i "poolen" med värdet "abc" om det inte redan finns ett.
Är **inte** samma objekt som **one** refererar till.

- Kort sammanfattning:

new String – Skapar ett String-objekt utanför "poolen" **och** ett String-objekt i "poolen" om det inte redan finns ett med samma värde.

Skapar **två** objekt men bara **en** referensvariabel dvs. till det utanför "poolen"

Konstruktörer i String

- String har 15 konstruktörer – två är *deprecated* dvs. bör inte användas

Exempel:

new String() - ny tom String

new String(char[] value) - ny String som innehåller tecknen i *value-array*

new String(String value) - ny String med värdet av *value* - gör en kopia

Viktiga metoder i String

- **public char charAt(int index)** - returnerar tecknet på angivet index

Exempel:

```
String x = "OOP - Rules!";  
char c = x.charAt(6);
```

c kommer att innehålla "R"

- **public String concat(String s)** – slår ihop två String

Exempel:

```
String x = "OOP";  
String z = x.concat(" - Rules!");
```

z kommer att innehålla värdet "OOP -Rules!"

- **public boolean equalsIgnoreCase(String s)** - kontrollerar om två teckensträngar är lika utan att ta hänsyn till stor eller liten bokstav

Exempel:

```
String x = "ABC";  
String z = "abc";
```

x.equalsIgnoreCase(z) - kommer att returnera true

Viktiga metoder i String forts.

- **public int length()** - returnerar längden på en String

Exempel:

```
String x = "abc";
```

x.length() - returnerar 3

- **public String replace(char old, char new)** - ersätter alla förekomster av tecknet "old" med tecknet "new"

Exempel:

```
String x = "Jxvx";
```

```
x.replace('x', 'a');
```

x kommer att returnera "Java"

- **public String trim()** - tar bort alla blanksteg i början eller slutet på en String

Exempel:

```
String x = "    OOP - Rules!    ";
```

```
x.trim();
```

x kommer att returnera "OOP - Rules!"

String - fallgropar

- **equals** - kontrollerar om teckensträngen är lika
- **operatorn ==** kontrollerar om referensen är till samma objekt

Exempel:

```
String one = "abc";
```

```
String two = "abc";
```

one == two - returnerar **true**, de refererar till samma objekt i "poolen"

```
String three = "abc";
```

```
String four = new String("abc");
```

three == four - returnerar **false**, de refererar inte till samma objekt

Däremot:

one.equals(two) - returnerar **true**, deras teckensträngar är lika

three.equals(four) - returnerar också **true**, deras teckensträngar är lika

Klassen StringBuilder

- Används när man behöver göra många modifieringar av en teckensträng
- Objekt av klassen är **inte** immutable och skiljer sig därför från String
- Går bara att skapa med operatör new

Exempel:

```
StringBuilder one = new StringBuilder("OOP ");  
one.append("- Rules!");
```

Detta ändrar innehållet i one från "OOP " till "OOP - Rules!"

Viktiga metoder i StringBuilder

- **public StringBuilder append(String s)** - lägger till s till en StringBuffer

Exempel:

```
StringBuilder one = new StringBuilder("OOP ");  
one.append("- Rules!");
```

Ändrar innehållet i one från "OOP " till "OOP - Rules!".
Returnerar en referens till det modifierade StringBuilder-objektet.

- **public StringBuilder insert(int offset, String s)** - lägger in s på angivet index

Exempel:

```
StringBuilder x = new StringBuilder("OO!");  
x.insert(2, "P - Rules");
```

Sätter in "P - Rules" mellan "OO" och "!".
Ändra StringBuilder-objektet till att hålla värdet "OOP - Rules!".

Viktiga metoder i StringBuilder forts.

- **public StringBuilder reverse()** - Kastar om tecknen så att de blir omvända

```
StringBuilder x = new StringBuilder("OOP - Rules!");  
x.reverse();
```

Ändrar x värde till "!seluR - POO" samt returnera en referens till x

public String toString() - returnerar en teckensträng

```
StringBuilder x = new StringBuilder("OOP - Rules!");  
x.toString()
```

Returnerar "OOP - Rules!"