

Lektion 4 – Strängar och Exceptions

Java för C++-programmerare

Syfte: Stränghantering och hantering exceptionella händelser.

Att läsa: Java direkt, kap 5, 11 samt andra relevanta avsnitt.

Relevant [API-Dokumentation](#)



Stränghantering

java.lang.String



- En sträng består av ett antal tecken
- Strängar i Java hanteras som objekt av klassen String
- En sträng kan inte modifieras efter att den har skapats!

String
- value : char[] - count : int
+ length(): int + charAt(int): char + indexOf(char): int ...

En sträng har:

← Ett värde och en längd

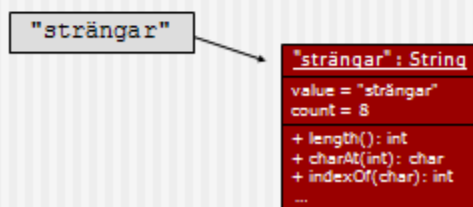
← Ett antal metoder

String-klassen finns i paketet `java.lang` vilket alltid är tillgängligt utan import. En speciell egenskap med String-objekt är det är *immutable* - innehållet i en sträng kan inte förändras. Som bilden ovan visar så representeras en sträng internt av en s.k. teckenarray (en array av `char`) samt en räknare.

Strängkonstanter



- "tecken omslutna av dubbel-fnuttar"
- En konstant, t.ex "strängar" ses som en referens till ett String-objekt.
- Alla förekomster av "strängar" i programmet refererar till samma objekt





Addera Strängar

- Ett sätt att skapa en *ny* sträng är att addera två andra.

- + operatoren används för detta

```
String efterNamn = "Karlsson";  
String namn = "Kalle " + efternamn;  
System.out.println(namn);    // Kalle Karlsson
```

- Primitiva typer konverteras om automatiskt vid strängaddition

```
System.out.println("5 + 5 = " + 5 + 5);    // 5 + 5 = 55  
System.out.println("5 + 5 = " + (5 + 5));  // 5 + 5 = 10  
      ("5 + 5 = " + "5" + "5")              ("5 + 5 = " + "10")
```

Operator + är överlagrad för String. Resultatet av en strängaddition är en ett String-objekt eftersom String är immutable. Det innebär att det är mycket ineffektivt att addera flera strängar eftersom nya objekt hela tiden måste skapas. För att bygga upp en sträng av flera andra strängar ska istället klassen StringBuffer användas. Mer om den senare.

I exemplet med println ser vi att ett numeriskt värde implicit konverteras till en sträng med motsvarande innehåll.. Detta sker eftersom i flera steg

- println förväntar sig en String efter ett +-tecken
- litteralen 5 är av typen int men kan konverteras implicit till ett Integer-objekt med värdet 5
- klassen Integer har omdefinierat metoden toString(), vilken returnerar den förväntade strängen.

En String är indexerad med första tecken vid index 0, precis som en C++-string.

Klassen String har många metoder, en del finns i flera överlagrade former, några av dem är

- length() - returnerar antalet tecken i strängen
- charAt(int) - returnerar tecknet för ett visst index, motsvarar index-operatorn [int] i C++ vilken saknas i Java.
- indexOf(...) - returnerar första index för ett visst tecken eller -1 om tecknet saknas
- lastIndexOf(...) - som indexOf fast söker från höger till vänster
- substring(...) - returnerar en delsträng av en sträng



Fler Metoder

- Klassen String innehåller även dessa metoder (används inte lika ofta)

```
boolean endsWith(String suffix)
boolean startsWith(String prefix)
String toUpperCase()
String toLowerCase()
String trim()
String replace(char oldChar, char newChar)
```

```
String s1 = "Javakursen";
boolean end = s1.endsWith("sen");           // true
boolean start = s1.startsWith("Java");      // true
s1 = s1.toUpperCase();                      // "JAVAKURSEN"
s1 = s1.toLowerCase();                     // "javakursen"
s1 = " Javakursen ".trim();                 // "Javakursen"
s1.replace('a', 'o');                       // "Jovokursen"
```



Jämföra strängar

- Tre metoder för att jämföra strängar

```
public boolean equals(Object anObject) // Överlagring
public boolean equalsIgnoreCase(String anotherString)
public int compareTo(String anotherString)
```

- Två strängar är lika om de innehåller samma tecken i rätt ordning

```
String s1 = "Java";
String s2 = "java";
s1.equals(s2);                // false
s1.equalsIgnoreCase(s2);     // true
s1 == s2;                    // false (fel sätt)
s1.compareTo("C++");          // returnerar 1
s1.compareTo("Pascal");       // returnerar -1
```


- == kollar om det är samma referens



Java erbjuder tre olika metoder för att jämföra strängar med varandra

- `equals(String)` – som kontrollerar om två strängar innehåller exakt samma tecken i exakt samma ordning. Denna metod tar hänsyn till stora och små bokstäver. Metoden `equals()` överlagrar alltså metoden i klassen `Object` som jämför om två objekt är lika.
- `equalsIgnoreCase(String)` – Här anses strängarna "Java" och "java" vara samma strängar.
- `compareTo()` jämför två strängar för att avgör vilken som kommer först i alfabetisk ordning. Metoden returnerar en `int` som är 0 (noll) om de båda strängarna är lika. Mindre än noll returneras om `s1` kommer före `s2`, och större än 0 returneras om `s2` kommer före `s1`. Observera att denna metod inte tar hänsyn till de svenska tecknen å, ä och ö vid jämförelsen. Denna metod kan vara bra att använda om man vill sortera strängar i bokstavsordning.

Observera att **vi inte kan använda likhetsoperatoren `==` för att jämföra om innehållet i två objekt är lika**. Eftersom `String` är en klass kommer `==` tillämpad på två `String`-objekt att returnera `true` om de båda objektreferenserna refererar till samma objekt (annars `false`).



Fler Metoder

- Klassen `String` innehåller även dessa metoder (används inte lika ofta)

```
boolean endsWith(String suffix)
boolean startsWith(String prefix)
String toUpperCase()
String toLowerCase()
String trim()
String replace(char oldChar, char newChar)

String s1 = "Javakursen";
boolean end = s1.endsWith("sen");           // true
boolean start = s1.startsWith("Java");      // true
s1 = s1.toUpperCase();                      // "JAVAKURSEN"
s1 = s1.toLowerCase();                      // "javakursen"
s1 = " Javakursen ".trim();                  // "Javakursen"
s1.replace('a', 'o');                        // "Jovokursen"
```



Formatering



- Metoden `format()` kan användas för att formatera en sträng eller utskrift

```
public static String format(String format, Object ... args)
// formatspecificerare ==> %[flagga][bredd]typ
```

- Kan göra väldigt avancerade formateringar, mest för siffror

```
int tim = 4; int min = 6; int sek = 9;
String f = String.format("%02d:%02d:%02d", tim, min, sek);
// f ==> 04:06:09

// eller för att skriva ut direkt
System.out.format("%02d:%02d:%02d", tim, min, sek);
// ger: 04:06:09
```

Modellen för den här typen av formatering är hämtad från standard C-bibliotekets funktioner `printf`, `fprintf` m.fl. och ger stora möjligheter till detaljerad formatering. Formatsträngen innehåller både den text som ska skrivas ut och olika formatspecificerare som anger hur innehållet ska formateras. De övriga parametrarna (en eller flera) innehåller de värden som ska formateras. I formatsträngen börjar varje ny formatspecificerare med tecknet `%` därefter följer eventuellt en flagga som t.ex. kan ange om vänsterjustering ska användas, om talen ska grupperas, om talets tecken alltid ska visas (+ eller -) om utfyllnad ska ske med blanksteg eller siffran 0 etc.

I exemplet i bilden formateras finns tre formatspecificerare i formatsträngen. `%02d` anger att det är heltal som ska formateras och att utskriften minst ska uppta två positioner. Eventuell utfyllnad ska göras med siffran 0. De tre övriga argumenten är de heltal som ska formateras (formatsträngen innehåller tre formatspecificerare).

Kursboken sida 147 – 151 för mer läsning om metoden `format`.



java.lang.StringBuffer



- För ändring/addition av strängar
- Några konstruktorer:

```
public StringBuffer()  
public StringBuffer(String en_existerande_sträng)
```

- Några metoder:

```
public StringBuffer append(Type t)           // lägger till sist  
public StringBuffer insert(int offset, Type t) // sätter in vid offset  
public StringBuffer delete(int start, int end) // tar bort tecknen mellan  
public StringBuffer reverse()                 // vänder - java → avaj  
public String toString()                      // StringBuffer som String  
  
public char charAt(int index)                 // returnerar tecknet vid index  
public int indexOf(String str)                // index där str finns (vänster→höger)  
public int lastIndexOf(String str)            // index där str finns (höger→vänster)  
public int length()                           // antal tecken i aktuell StringBuffer  
public String substring(int start)            // en substräng från start till slutet
```

Om man vet att en sträng kommer att behöva modifieras efter att den är skapad ska man klassen `StringBuffer` i stället för `String`. `StringBuffer` är i stort sett uppbyggd på samma sätt som en sträng (har ett värde och längd) men har även en kapacitet som anger hur många tecken en `StringBuffer` kan innehålla. Kapaciteten ökas automatiskt vid behov.

För att skapa en `StringBuffer` kan man antingen använda den tomma konstruktorn som skapar en `StringBuffer` med längden 0, innehållet "" och kapaciteten 16 tecken. Det finns även en konstruktor som tar en sträng (`String`) som argument vilket innebär att det är lätt att skapa en `StringBuffer` från en redan existerande sträng (ett sätt att konvertera en sträng till en `StringBuffer`).

När vi har skapat en `StringBuffer` kan vi anropa metoder för att lägga till strängar, sätta in strängar, tecken, tal m.m. sist eller vid ett visst index i `StringBuffer`.

Konstruktorn `StringBuffer(String)` konverterar en sträng till en `StringBuffer`. Genom metoden `toString()` får vi hela innehållet i en `StringBuffer` som en `String`.



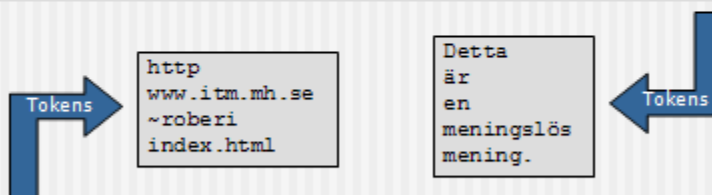
java.util.StringTokenizer

- Används för att dela upp en sträng i delar (tokens), t.ex ta fram orden ur en mening
- Normalt delas strängen vid mellanslag, tabb och radbyte
- Man kan också ange vilka tecken strängen ska delas vid.

StringTokenizer (forts)

- Default-avgränsare är *whitespace*

```
String s = "Detta är en meningslös mening.";
StringTokenizer st = new StringTokenizer(s);
```



```
String s = "http://www.itm.mh.se/~roberi/index.html";
StringTokenizer st = new StringTokenizer(s, ":/");
```

- Kan ange egna avgränsare

Med en StringTokenizer kan man dela upp en sträng i olika delar, s.k. tokens. Default avgränsare (delimiter) är något "white space"-tecken (mellanslag, tab, newline etc). Annan avgränsare kan anges i en överlagrad konstruktör.



StringTokenizer (forts)



■ Exempel

```
import java.util.StringTokenizer;
public class RaknaOrd {
    public static void main(String[] args)
    {
        String mening = "Detta är en meningslös mening";
        StringTokenizer st = new StringTokenizer(mening);
        int antalOrd = st.countTokens();
        System.out.println("Antal ord: " + antalOrd);

        // Skriver ut orden
        while (st.hasMoreTokens())
        {
            System.out.println(st.nextToken());
        }
    }
}
```

I while-loopen skriver vi ut alla tokens, dvs. ord i strängen. Default avgränsare "white space" gäller. Metoden `hasMoreTokens()` kommer att returnera true så länge som det finns fler tokens att hämta ur `st`. Med andra ord kommer while-loopen att snurra så länge som det finns fler tokens.



Exceptions/Undantag

Javas undantagshantering är ursprungligen en kopia av den i C++ men med vissa tillägg.

- Ett exception är en händelse som inträffar under exekveringen av ett program och som avbryter det normala programflödet.
- Att skapa ett exception-objekt och skicka det till runtime-systemet kallas för att ”*kasta ett undantag*” (*throw an exception*).
- Den hanterare som tar hand om undantaget sägs ”*fånga undantaget*” (*catch the exception*).

Fördelen med att använda undantag är att man på ett bra sätt kan separera kod för felhantering från ”vanlig” kod.

Att fånga och hantera undantag

- **try-blocket**
 - Satser som kan generera undantag placeras inom ett try-block.
- **catch-blocket(-en)**
 - Direkt efter try-blocket placeras ett eller flera catch-block.
- **finally-block**
 - Direkt efter det sista catch-blocket erbjuder Java möjligheten att placera ett **finally**-block. Om ett sådant block finns så kommer *satserna i det alltid att exekveras* oavsett om ett undantag hanterats i ett catch-block eller inte. Använd detta block till att stänga filer och frigöra andra systemresurser.
 - Om ett finally-block saknas så fortsätter exekveringen på första satsen efter det sista catch-blocket, precis som i C++.
 - Någon motsvarighet till finally finns inte i C++ i nuvarande standard.
- **throw**
 - Kastar ett skapat exception-objekt till nästa nivå av catch-block eller ut ur den aktuella metoden bakåt/uppåt till föregående (anropande) metod i anropskedjan.
- **throws**
 - Deklarerar de typer av undantag som den aktuella metoden kan kasta.



- **catch**
 - En metod kan fånga ett undantag genom att tillhandahålla en hanterare för den aktuella typen av undantag.
- - **Om en metod väljer att inte innesluta kod som kan generera undantag i ett try-block så måste metoden deklarera att den kastar undantaget vidare med throws i metodhuvudet.** Detta är en mycket strängare policy än i C++ där man i princip kan ignorera att funktioner kan generera undantag.

Java tillåter alltså inte att man struntar i att undantag kan kastas!

Man har som tidigare nämnts två sätt att hantera detta:

1. "Struts-modellen" där man sticker huvudet i sanden och gör det så enkelt som möjligt genom att inte skriva try/catch-block utan istället deklarera alla metoder med throws exceptiontyp1, exceptiontyp2... och på så vis låta anropande metod ta hand om problemet. Om man konsekvent gör det så kommer ett exception att "bubbla upp" till main-metoden. Om den också är deklarerad med throws så kommer programmet att avslutas pga. ett "unhandled exception", annars måste main, som sista utpost, ha try/catch-block för de aktuella undantagen. Detta är en "lat kodning" och ingen modell som rekommenderas i seriös programmering men den är snabb och kan användas i testprogram under utvecklingen av en applikation.
2. Alla anrop som kan generera undantag är inneslutna i try-block och eventuella undantag hanteras i de följande catch-blocken. Eventuellt kastas ett undantag vidare uppåt/bakåt till anropande metod.

De båda metoderna kan även kombineras för att dirigera undantagen till vissa delar av koden och sköta felhanteringen där.

"Lat" kodning:

```
Class MyClass
{
    public MyClass() throws Exception
    {
        // Ett undantag här kommer att avsluta programmet
    }
    public static void main(String[] args) throws Exception
    {
        new MyClass();
    }
}
```