# Class StringBuilder

**Catenating a character to a String can be expensive**

The operation of adding one character to a `String s`, as done using this assignment statement:

```
s= s + 'c';
```

can take a lot of time — time proportional to the length of `s`. That's because `Strings` are immutable, so the operation is carried out like this:

1. Evaluate the expression:
    a. Create a new `String` object, call it `s1`, with space for all the chars —those in `s` and character `'c'`;
    b. Copy the chars from `s` to `s1`;
    c. Copy char `'c'` into `s1`;
2. Store into `s` a pointer to the new object `s1`.

Consider writing a loop to add 1,000 characters to an empty `String s`:

```
for (int k= 0; k < 1000; k= k+1) {
   s= s + (some char depending on k);
}
```

Let's add up the number of characters that will be copied into a new `String` object:

When k = 0: 1 char will be copied
When k = 1: 2 chars will be copied
…
When k = 999: 1000 chars will be copied

That's a total of 1 + 2 + 3 + … + 1000 = 1000 * 999 / 2 = 499,500 characters! Very inefficient.

For every day programming, when it is known that strings will not be long, creating a string as shown above is OK. But in a program that will be put into production and used often, creating long strings, the above method of creating a string by appending chars over and over is not OK. Below, we show a better way.

**Class StringBuilder**

Use class `StringBuilder`, in package `java.lang`, whose objects are mutable. Here is how we would write the loop given above along with a statement to store the result in `String` variable `s`. We give some explanation below:

```
StringBuilder sb= new StringBuilder("");

for (int k= 0; k < 1000; k= k+1) {
   sb.append(some char depending on k);

}

String s= sb.toString();
```

Method `sb.append` appends its argument to `sb`. No new `StringBuilder` object is created. There will be a need to create a new array of chars from time to time, but this is done in a way that ensures that the number of chars copied is proportional to 1000 in the above case. Perhaps 2000 chars are copied in total, or 3000, but certainly not 499,500! We study this issue when studying complexity and the concept of "amortized time", but you need not be concerned with that here.

Method `append` can have an argument of any type, and that argument is turned into a `String` of chars in the usual fashion and appended.

Class `StringBuilder`'s method `insert` can be used to insert its argument at *any* place with the string of characters that are in the object. We suggest that you read the API documentation for class `StringBuilder`.