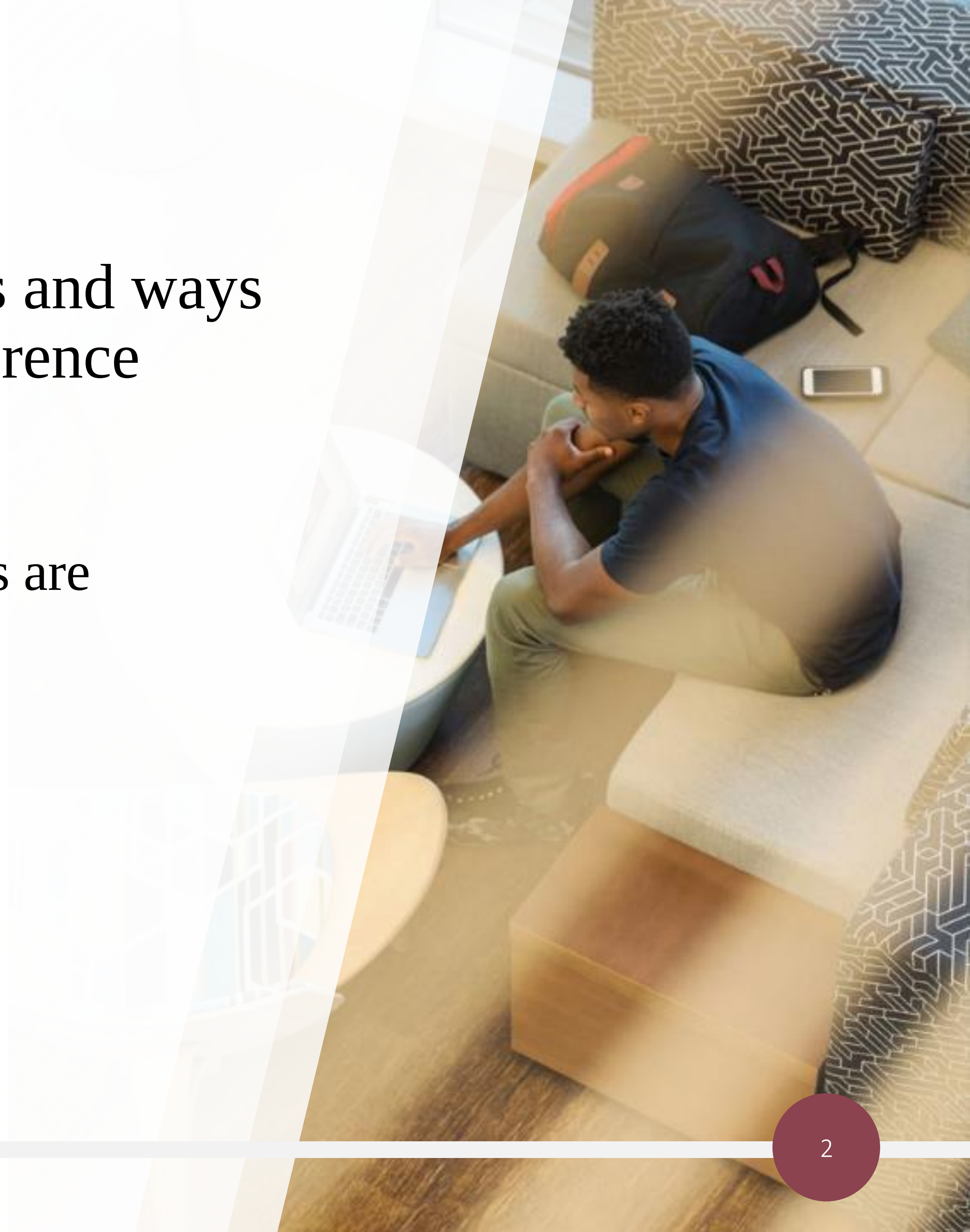
Working with Java Data Types

StringBuilder

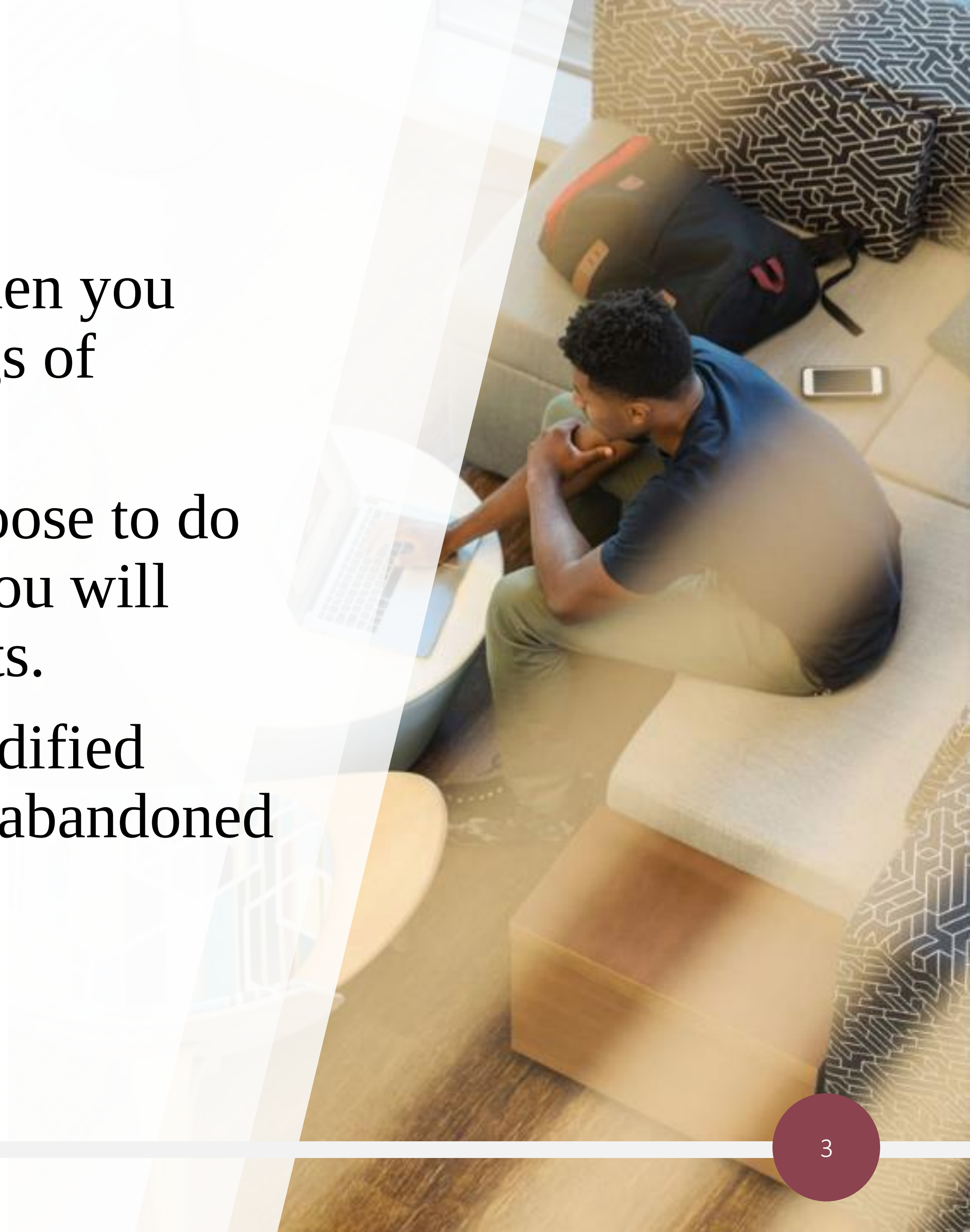
StringBuilder

- *StringBuilder* gives you *String*-like objects and ways to manipulate them, with the important difference being that these objects are mutable.
  - *StringBuilder* <u>objects</u> are mutable
  - *String* <u>objects</u> are immutable (*String* references are mutable)



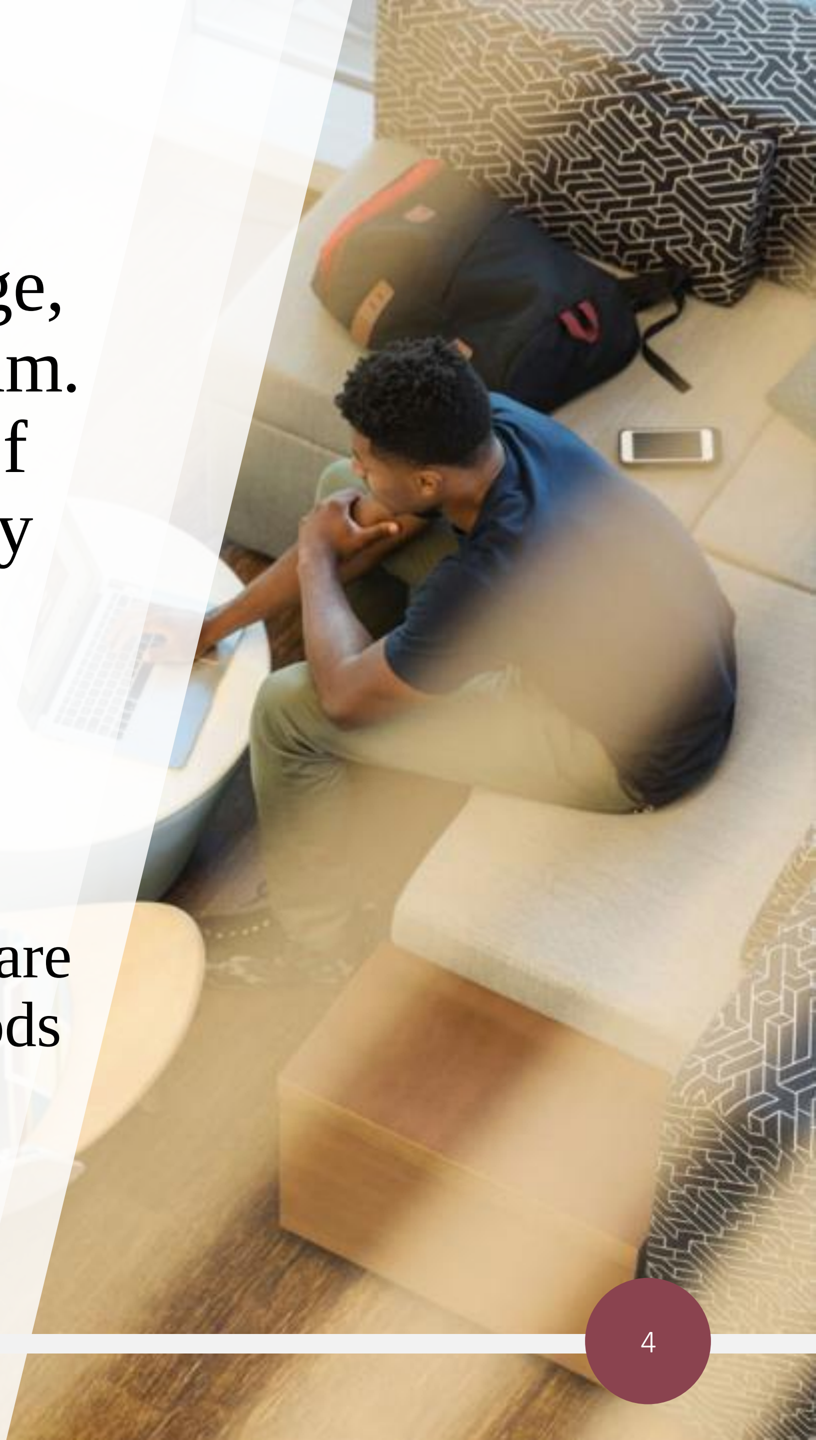
StringBuilder

- The *StringBuilder* class should be used when you have to make a lot of modifications to strings of characters.
- *String* objects are immutable, so if you choose to do a lot of manipulations with *String* objects, you will end up with a lot of abandoned *String* objects.
- However, *StringBuilder* objects can be modified over and over again without leaving behind abandoned objects.

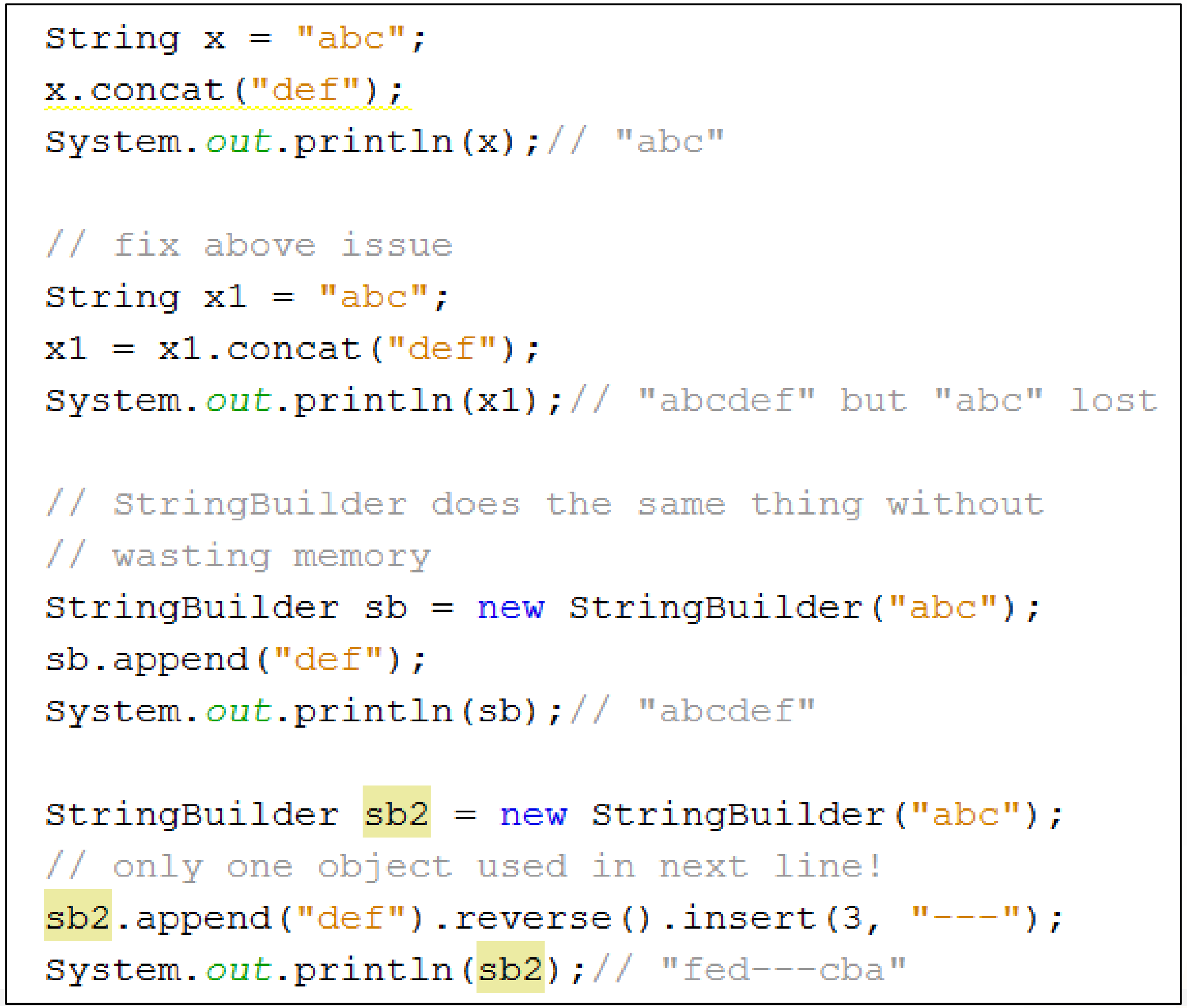


StringBuilder

- A common use for *StringBuilder* is file I/O, where large, ever-changing streams of data are handled by the program. *StringBuilder* objects are ideal for handling the blocks of data, passing them on and then reusing the same memory to handle the next block of data.
- Prefer *StringBuilder* to *StringBuffer*
  - ➢Both *StringBuilder* and *StringBuffer* API's are the same.
  - ➢*StringBuilder* is not thread-safe as its methods are not *synchronized*; use *StringBuffer* for threading as its methods are *synchronized*. However, the overhead of *synchronized* methods means that *StringBuilder* runs faster (and its use is recommended by Oracle).



String versus StringBuilder



```
String x = "abc";
x.concat("def");
System.out.println(x);// "abc"

// fix above issue
String x1 = "abc";
x1 = x1.concat("def");
System.out.println(x1);// "abcdef" but "abc" lost

// StringBuilder does the same thing without
// wasting memory
StringBuilder sb = new StringBuilder("abc");
sb.append("def");
System.out.println(sb);// "abcdef"

StringBuilder sb2 = new StringBuilder("abc");
// only one object used in next line!
sb2.append("def").reverse().insert(3, "---");
System.out.println(sb2);// "fed---cba"
```

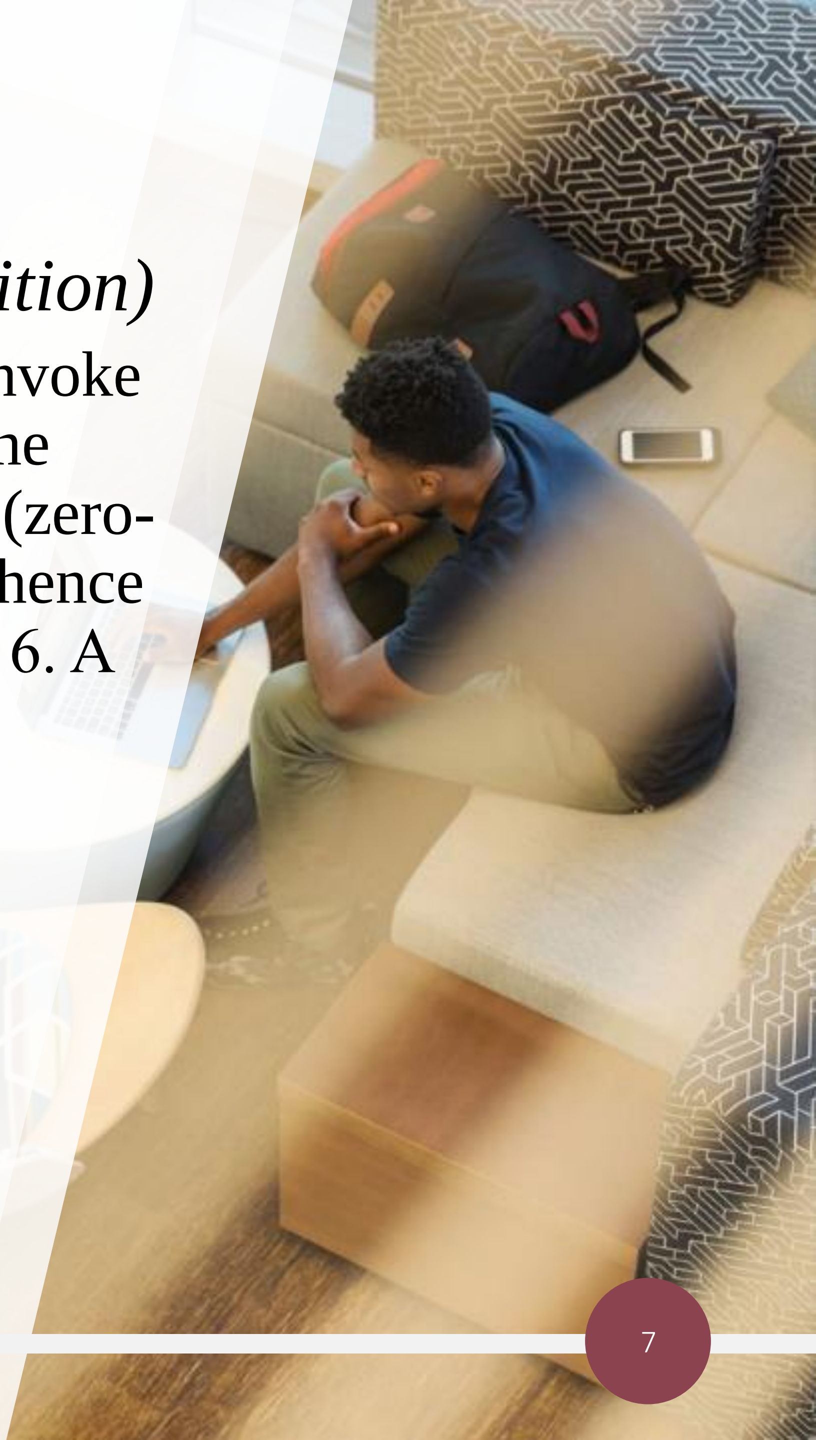
Important StringBuilder Methods

- *public StringBuilder append(String s)*
  - updates the value of the object that invoked the method, **whether or not the returned value is assigned to a variable** (the opposite to *String*). Overloaded for many different argument types.
    - ➢StringBuilder sb = new StringBuilder(“set ”);
      sb.append(“point”);
      System.out.println(sb); // “set point”

    - ➢StringBuilder sb2 = new StringBuilder(“pi = ”);
      sb2.append(3.142f);
      System.out.println(sb2); // “pi = 3.142” (note: no “f”)

Important StringBuilder Methods

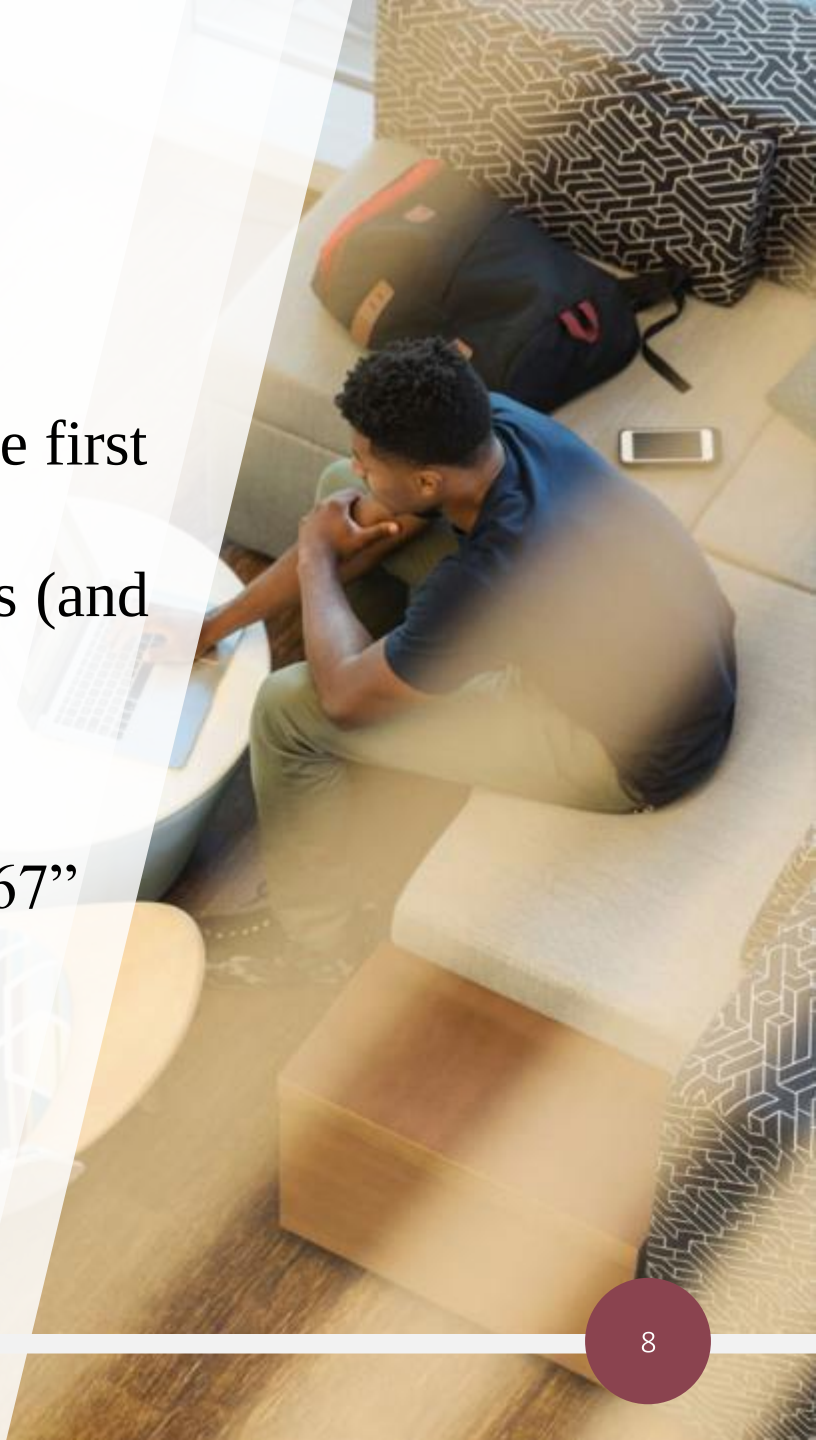
- *public StringBuilder delete(int beginIndex, int endPosition)*
  - Removes a substring from the *StringBuilder* object used to invoke the method. The arguments are similar to the *substring()* in the *String* class. The first argument represents the starting index (zero-based). Unfortunately, the 2nd argument is NOT zero-based (hence the term “position”). For example, position 7 is in fact index 6. A useful rule-of-thumb is to remove *endPosition-beginIndex* characters starting at *beginIndex*.
    - ➢StringBuilder sb = new StringBuilder(“0123**45**6789”);
      // remove 6-4 (2) chars beginning at index 4
      System.out.println(*sb.delete(4, 6)*); // “01236789”



Important StringBuilder Methods

- *public StringBuilder insert(int offset, String s)*
  - the *String* passed in as the 2nd argument is inserted into the *StringBuilder* starting at the offset location represented by the first argument (the offset is zero-based).
  - As with *append()*, *insert()* is overloaded to accept other types (and not just *String*).

```
➢StringBuilder sb = new StringBuilder("01234567");
 System.out.println(sb.insert(4,  "---"));       // "0123---4567"
```



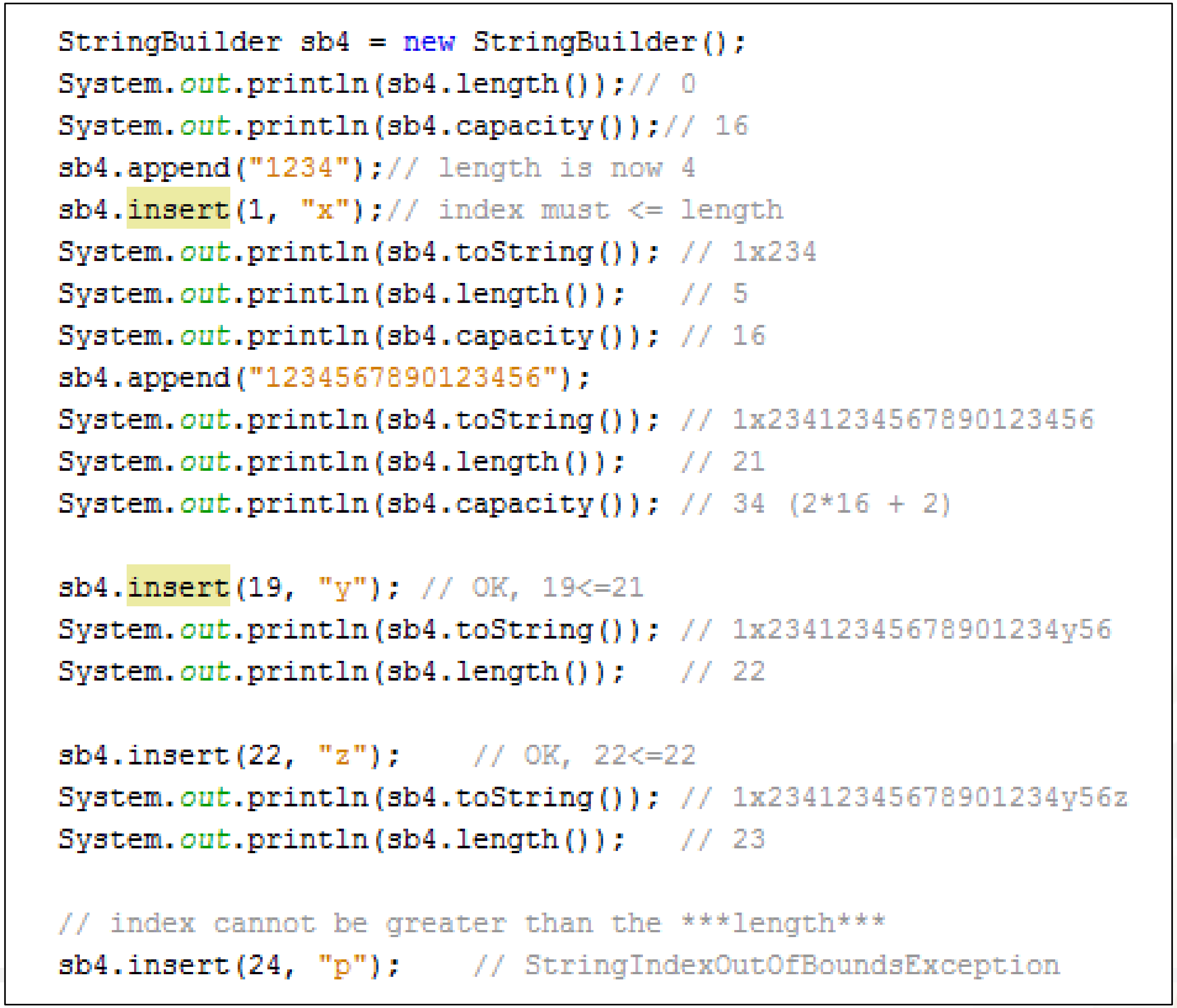
Important StringBuilder Methods

- *public StringBuilder reverse()*
  - the characters in the *StringBuilder* are reversed

    ➢StringBuilder sb = new StringBuilder(“abcd”);
    System.out.println(*sb.reverse()*);   // “dcba”

- *public* ***<u>String</u>*** *toString()*
  - returns the value of the *StringBuilder* object that invoked the method, as a *String*.

    ➢StringBuilder sb = new StringBuilder(“it is raining”);
    System.out.println(*sb.toString()*); // “it is raining”



```java
StringBuilder sb4 = new StringBuilder();
System.out.println(sb4.length());// 0
System.out.println(sb4.capacity());// 16
sb4.append("1234");// length is now 4
sb4.insert(1, "x");// index must <= length
System.out.println(sb4.toString()); // 1x234
System.out.println(sb4.length());   // 5
System.out.println(sb4.capacity()); // 16
sb4.append("1234567890123456");
System.out.println(sb4.toString()); // 1x2341234567890123456
System.out.println(sb4.length());   // 21
System.out.println(sb4.capacity()); // 34 (2*16 + 2)

sb4.insert(19, "y"); // OK, 19<=21
System.out.println(sb4.toString()); // 1x23412345678901234y56
System.out.println(sb4.length());   // 22

sb4.insert(22, "z");    // OK, 22<=22
System.out.println(sb4.toString()); // 1x23412345678901234y56z
System.out.println(sb4.length());   // 23

// index cannot be greater than the ***length***
sb4.insert(24, "p");    // StringIndexOutOfBoundsException
```