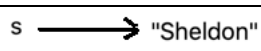
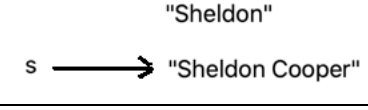
Class 2
Strings are immutable

Once a String is created, it cannot be changed.
Example

String s = “Sheldon”;	
s = s + “ Cooper”	

Concatenation creates a new String. Concatenation copies “Sheldon” into a new String and adds “ Cooper” to that.

Example:

String s = ‘a’	s → “a”
s = s + “b”;	“a” s → “ab”
s = s + “c”;	“a” “ab” s →”abc”
s = s + “d”;	“a” “ab” “abc” s →”abcd”
s = s + “e”;	“a” “ab” “abc” “abcd” s →”abcde”

Each time a letter was added to the string a new string was created.
So if s is currently “abcd” then
S = s+ “e”

- Makes a copy of “abcd”
- Adds “e” making the string “abcde”
- Assigns the new string to s

Strings are immutable. You cannot alter a string. If you want to add “e” Java makes a new string.

The original string (“abcd”) is now longer accessible and is called an orphan.

Here is a program that just keeps 'X' onto a string (X XX XXX XXXX XXXXX etc.). But every time it does that, it makes a brand new string. I ran the program making strings of size 1000, 10000, 100000, and 20000 and timed each program.
Here is the program with the output

```
public class Concatenation
{
  public static void main(String[] args)
  {
    String s = "X";
    long start = System.currentTimeMillis();

    for (int i = 0; i < 1000; i++)
      s = s + "X";

    long end = System.currentTimeMillis();
    System.out.println("Time: "+ (end-start)+ "
mls");
  }
}
```

Output: Time: 4 mls

```
public class Concatenation
{
  public static void main(String[] args)
  {
    String s = "X";
    long start = System.currentTimeMillis();

    for (int i = 0; i < 10000; i++)
      s = s + "X";

    long end = System.currentTimeMillis();
    System.out.println("Time: "+ (end-
start)+ " mls");
  }
}
```

Output: Time: 59 mls

```
public class Concatenation
{
  public static void main(String[] args)
  {
    String s = "X";
    long start = System.currentTimeMillis();

    for (int i = 0; i < 100000; i++)
        s = s + "X";

    long end = System.currentTimeMillis();
    System.out.println("Time: "+ (end-start)+ "
mls");
  }
}
```

Output: Time: 5454 mls

```
public class Concatenation
{
  public static void main(String[] args)
  {
    String s = "X";
    long start = System.currentTimeMillis();

    for (int i = 0; i < 200000; i++)
        s = s + "X";

    long end = System.currentTimeMillis();
    System.out.println("Time: "+ (end-
start)+ " mls");
  }
}
```

Output: Time: 22155 mls

Notice when I doubled the work (100,000 to 200,000) the time (roughly) quadrupled.

Another String method is toUpperCase()

Example

String s = “abc”	s → “abc”
s = s.toUpperCase()	“abc” s → “ABC”

The method toUpperCase() does NOT change the original string “abc” to “ABC”
STRINGS ARE IMMUTABLE
toUppercase() creates a new String (“ABC”) and assigns “ABC” to s.
The original string (“abc”) is no longer referenced by any variable. It is inaccessible.
It is called an *orphan.*

How to test equality of strings.

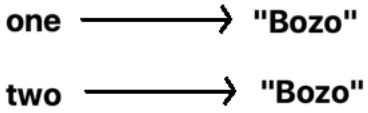
Assume that you create two String objects:

```
String one = new String(“Bozo”);
String two = new String(“Bozo”);
```

What is the value of the expression

one == two

Is it true or false???
one is holding the address of the first String and two is holding the address of the second String



Even though the characters are the same one and two store two different addresses.
The expression one == two compares the ADDRESSES stored in one and two. They are different so the expression one == two is false.
With Strings == compares references (addresses in memory)

To determine whether two strings are identical character by character use the equals(…) method

one.equals(two)

This will return true. Both strings have identical characters (in the same order).
Basic Rule:

USE equals(…) WHEN TESTING EQUALITY OF STRINGS

There are very few cases when you want to compare addresses and use ==

Example

```
public  class TestEquals
{
    public static void main(String[] args)
    {
        String a = new String(“Rex”);
        String b = new String(“Rex”);
        String c = new String(‘rex”);
        System.out.println(a ==b);
        System.out.println(a.equals(b);
        System.out.println(a.equals(c));
        c = a;
        System.out.println(a ==c);
        System.out.println(a.equals(c));
    }
}
```

What is the output?

Here are a few methods of the String class – there are many more

Some String methods

Method	Explanation	Example
char charAt(int index)	s.charAt(i) returns the character at index *i*. All Strings are indexed from 0.	String s = "Titanic"; s.charAt(3) returns 'a' (indexing starts at 0)
int compareTo(String t) **DO NOT USE > or < TO COMPARE TWO STRINGS**	compares two Strings, character by character, using the ASCII values of the characters. s.compareTo(t) returns a negative number if s < t. s.compareTo(s) returns 0 if s == t. s.compareTo(t) returns a positive number if s > t.	String s = "Shrek"; String t = "Star Wars"; s.compareTo(t) returns a negative number. s.compareTo(s) returns 0. t.compareTo(s) returns a positive number
int compareToIgnoreCase(String t)	similar to compareTo(...) but ignores differences in case.	String s = "E.T."; String t = "e.t."; s.compareToIgnorecase(t) returns 0.
boolean equals(Object t) (The strange parameter will make sense later. For now, think of the parameter as String)	s.equals(t) returns true if s and t are identical character by character.	String s = "Star Trek"; String t = "STAR TREK"; s.equals(t) returns false s.equals("Star Trek") returns true
boolean equalsIgnoreCase(String t)	s.equalsIgnoreCase(t) returns true if s and t are identical, ignoring case.	String s = "STAR TREK"; String t = "Star Trek"; s.equalsIgnorecase(t) returns true
int indexOf(String t)	s.indexOf(t) returns the index in s of the first occurrence of t and returns -1 if t is not a substring of s.	String s = "**The** Lord Of The Rings"; s.indexOf("The") returns 0; s.indexOf("Bilbo") returns –1.
int indexOf(String t, int from)	s.indexOf(t, from) returns the index in s of the first occurrence of t beginning at index from; an unsuccessful search returns –1.	String s = "The Lord Of **The** Rings"; s.indexOf("The", 6) returns 12;
int length( )	s.length() returns the number of characters in s.	String s = "Jaws"; s.length() returns 4
String replace(char oldChar, char newChar)	s.replace(oldCh, newCh) returns a String obtained by replacing every occurrance of oldCh with newCh.	String s = "Harry Potter"; s.replace ('r','m') returns "Hammy Pottem"
String substring(int index)	s.substring(index) returns the substring of s consisting of all characters with index greater than or equal to index.	String s = "The Sixth Sense"; s.substring(7) returns "th Sense"

String substring(int start, int end)	s.substring(start, end) returns the substring of s consisting of all characters with index greater than or equal to start and strictly less than end.	String s = "The Sixth Sense"; s.substring(7, 12) returns "th Se"
String toLowerCase()	s.toLowerCase() returns a String formed from s by replacing all upper case characters with lower case characters.	String s = "The Lion King"; s.toLowerCase() returns "the lion king"
String toUpperCase()	s.toUpperCase() returns a String formed from s by replacing all lower case characters with upper case characters.	String s = "The Lion King"; s. toUpperCase() returns "THE LION KING"
String trim()	s.trim() returns the String with all leading and trailing white space removed.	String s = " Attack of the Killer Tomatoes "; s.trim() returns "Attack of the Killer Tomatoes"

One method that may be a little strange is compareTo(…)
With integers or even single characters you can use < and >.
40 < 50 returns true and 'a' > 'b' returns false. You cannot use the > and < operators with strings.

Suppose that a is the String "abc" and b is the String "xyz"
a.compareTo(b) returns a negative number because a precedes b alphbetically (a is less than b)
b.compareTo(a) returns a positive number because b follows a alphabetically
a.compareTo(a) returns 0 because a is the same as a

For example:

```
if (a.compareTo(b)) < 0)
    System.out.print("Hello")
else if (a.compareTo(b) > 0)
    System.out.print("Bye")
else
   System.out.print("They're the same");
```

Here is the fine print:
Suppose s1 → "abc" and s2 → "ABC". What does s1.compareTo(s2) return?
You might think they are the same but they are not. The compareTo(..) method uses the ACSII values of the characters. Since 'a' has ASCII value 97 and 'A' has value '65
S1.compareTo(s2) returns a positive number since 97 is greater than 65.

There is also a method int compareTolgnoreCase(String t)

Here are some examples of small programs that illustrate the String methods.
Example: Write a program with a method that accepts a person's name in the form
Firstname space lastName
And returns a string with the name in the form
Lastname,Firstname
For example, Bart Simpson will be reformatted as Simpson, Bart

Here is a shell for the program with an ALGORITHM for creating the new name

```
public class Names
{
    public static String reverseName(String name)
    {
        //find the position of the space (use indexOf(…)
        // get the first name, all characters up to the space (use substring(..))
        // get the last name, all characters AFTER the spce (use substring(..))
        // make a new string by concatenating lastname+”,”+firstname
        // return the new string
    }
   public static void main(String[] args)
  {
    Scanner input = new Scanner (System.in);
    System.out.print("Enter a name: ");
    String name = input.nextLine();
    String newName = reverseName(name);
    System .out.println(newName);
  }
}
```

Here is the program:

```
import java.util.*;
public class Names
{

  public  static String reverseName(String name)
  {
    //takes a name in the form "first last" and
    //returns it in the form "last,first"
    // for example Homer Simpson is returned as Simpson,Homer

    int space = name.indexOf(" ");                    // find the  position of the space
    String firstName = name.substring(0,space);       // get characters up to the space
    String lastName = name.substring(space+1);        // get all characters after the space
    return lastName+","+firstName;                    //makes and return the reformatted name
  }
```

```
  public static void main(String[] args)
  {
    Scanner input = new Scanner (System.in);
    System.out.print("Enter a name: ");
    String name = input.nextLine();
    String newName = reverseName(name);
    System .out.println(newName);
  }
}
```

Output:

```
Enter a name:  Homer Simpson
Simpson,Homer
```

The next example reads a string of words, each separated by a single space, and returns the words in reverse order.

For example

my dog has fleas

is returned as

fleas has dog my

```
import java.util.*;
public class ReverseWords
{

  public  static String reverse(String s)
  {
    //takes a line of words separated by a space
    //returns returns them in reverse order

    String reverse = ""; // the new string with words reversed
    while (s.indexOf(" ") >=0)   //indexOf() returns -1 if it cannot find the character
    {
       int space = s.indexOf(" ");                 // find a space
       String word = s.substring(0,space);   // get the word up to the space
       reverse = word + " " +reverse;         // adds to front of reverse
       s = s.substring(space+1);                 // remove the word from the “sentence”
    }
    reverse = s+" " + reverse;                  // add the last word
    return reverse;
  }

  public static void main(String[] args)
  {
    Scanner input = new Scanner (System.in);
    System.out.print("Enter a line of words separated by a space: ");
    String line = input.nextLine();
    String newLine = reverse(line);
    System.out.println(newLine);
  }
}
```

We can trace the program to see how it works

Example:
A Palindrome is a String that reads the same forwards and backwards
For example
Solos
Civic
Madam
Racecar
Are all palindromes.

Here is a program with a method that accepts a string and determines whether or not it is a palindrome

```
import java.util.*;
public class Palindrome
{
 public  static boolean isPalindrome(String s)
 {
  //determines whether or not a string is a Palindrome
  // assumes all characters are aphabetical

  String reverse = "";

  for (int i = 0; i < s.length(); i++)       // iterate through each character
   reverse = s.charAt(i) + reverse;      // keeps adding to the front

  return reverse.equalsIgnoreCase(s);  // notice the equality ignored
 }

 public static void main(String[] args)
 {
  Scanner input = new Scanner (System.in);
  System.out.print("Enter a string: ");
  String str = input.nextLine();
  if (isPalindrome(str))
        System.out.println(str+ " is a palindrome");
  else
        System.out.println(str+ " is  NOT a palindrome");
 }
}
```

Output

Enter a string: Racecar Racecar is a palindrome	Enter a string: Snowball Snowball is NOT a palindrome >	Enter a string: kayak kayak is a palindrome	

String Builder class (This should be new to everyone)

Because strings are immutable, programs with heavy concatenation can be pretty inefficient because a new string must be created for each instance of concatenation.

Java also provides the StringBuilder class. Unlike String objects, StringBuilder objects can be changes. In other words, a StringBuilder object is like a String but it is NOT immutable.

Many of the methods are the same as those of the String class. For example, the StringBuilder class has methods such as length() or charAt(i)

Unlike String objects StringBuilder objects can be altered. Strings are immutatable; StringBuilders are not. A StringBuilder is like a String that you can change.

Here is how you instantiate or create StringBuilder objects.

When you instantiate or create a StringBuilder object a *buffer* (section of memory) is created for the characters of the StringBuilder object. A program can change the characters in the buffer. In other words, a StringBuilder object, unlike a String, is NOT immutable.

Example:

StringBuilder sb = new StringBuilder()	Default buffer holds 16 characters
StringBuilder sb = new StringBuilder(100)	Initially buffer holds 100 characters
StringBuilder sb = new StringBuilder("Hello")	Initially buffer is size 6 and contains the characters of "Hello" (sb→ "Hello")

The buffer can expand as needed and the characters in the buffer can be changed.

StringBuilderMethods

Method	**Explanation (sb refers to a StringBuilder)**	**Example**
StringBuilder append(String s) StringBuilder append(char c) StringBuilder append(StringBuilder s)	sb.append(x) appends *x* to the end of sb and returns a reference to the **altered** StringBuilder object.	StringBuilder s = new StringBuilder("New"); s.append(" York"); returns StringBuilder("New York") **and** alters s
char charAt(int i)	sb.charAt(i) returns the character at position i. StringBuilder character sequences are indexed from 0	StringBuilder s = new StringBuilder("Iowa"); char ch = sb.charAt(3); ch has the value 'a'

StringBuilder delete(int start, int end)	sb.delete(start, end) removes the characters from position start to position end -1 and returns a reference to the altered StringBuilder object.	StringBuilder s= new StringBuilder("Delaware"); s.delete(2,6); returns StringBuilder("Dere") and alters s
StringBuilder deleteCharAt(int i)	sb.deleteCharAt(i) removes the character at index i and returns a reference to the altered StringBuilder object.	StringBuilder s = new StringBuilder("Maine"); s.deleteCharAt(1); returns StringBuilder("Mine") and alters s
int indexOf(String s)	sb.indexOf(s) returns the index of the first occurrence of s in sb. If s is not a substring of sb, returns –1.	StringBuilder s = new StringBuilder("Florida"); int x = s.indexOf("or"); x has the value 2
int indexOf(String s, int from)	sb.indexOf(s, from) returns the index of the first occurrence of s in sb starting at index from. If s is not a substring of sb, returns –1.	StringBuilder s = new StringBuilder("Mississippi"); int x = s.indexOf("is",2); x has the value 4
StringBuilder insert(int index, String s) StringBuilder insert(int index, char ch)	sb.insert(index, s) inserts s into sb at position index.	StringBuilder s = new StringBuilder("Oo"); s.insert(1, "hi); returns StringBuilder("Ohio") and alters s
int length()	sb.length() returns the number of characters in sb.	StringBuilder s = new StringBuilder("Vermont"); s.length returns 7
StringBuilder replace(int start, int end, String s)	sb.replace(start, end,s) replaces all characters from start to end-1 with s and returns a reference to the altered StringBuilder object.	StringBuilder s = new StringBuilder("Texas"); s.replace(1,4,"axe") returns "Taxes" and alters s
StringBuilder reverse()	sb.reverse() reverses the order of the characters of sb and returns a reference to the altered StringBuilder object.	StringBuilder s = new StringBuilder("Utah"); s.reverse() returns StringBuilder("hatU") and alters s
String substring(int index)	s.substring(index) returns the substring of s consisting of all characters with index greater than or equal to index. Notice that the method returns a String reference.	StringBuilder sb = new StringBuilder("New Jersey"); sb.substring(4) returns "Jersey"
String substring(int start, int end) Notice that the method returns a String reference.	s.substring(start, end) returns the substring of s consisting of all characters with index greater than or equal to start and strictly less than end.	StringBuilder sb = new StringBuilder("New Jersey"); sb.substring(0,3) returns "New"

String toString()	sb.toString() returns a String representation of the characters of sb.	StringBuilder s = new StringBuilder("Illinois"); String str =s.toString(); str refers to the **String** object "Illinois"

The Equals(..) Method of StringBuilder

Note: **Unlike the String class the equals of the StringBuilder class compares REFERENCES not characters. A reference is a memory address**

StringBuilder a = new StringBuilder("Sheldon"); StringBuilder b = new StringBuider("Sheldon");	**a ⟶ "Sheldon"** **b ⟶ "Sheldon"**

StringBuilder a = new StringBuilder("Sheldon");
StringBuilder b = new StringBuilder("Sheldon");

a.equals(b) is false because the equals of StringBuilder compares references
How can we compare characters of a StringBuilder object?

String s = a.toString(); // returns a String version of a
String t = toString();

s.equals(t) returns true --> String equals compares characters
Here we are using the equals of the String class, which compars characters

Example:
Two of the methods of the StringBuilder class are
append(String s) and append(char c) // adds a String or character to the end

StringBuilder sb = new StringBuilder("Hello"); // sb →"Hello"
Sb.append(" Newman"); // sb → "Hello Newman"

This code does NOT create a new StringBuilder object but adds " Newman" to the end of the StringBuilder "Hello"

Example:

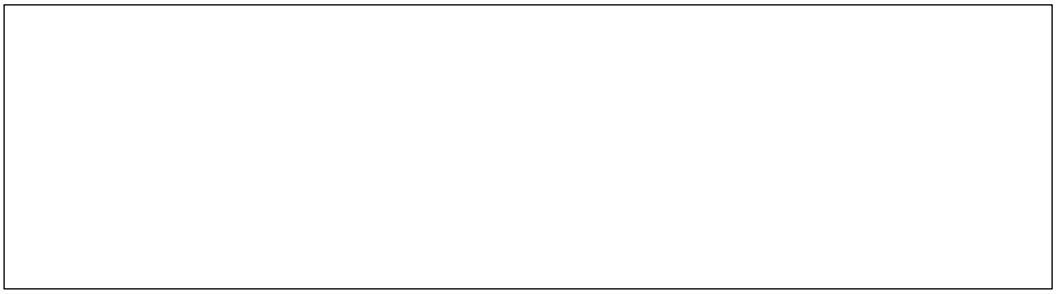
StringBuilder sb = new StringBuilder("Kramer");	**sb ⟶ "Kramer"**
sb.delete(1,4); // remove characters (1) through (3)	**sb ⟶ "Ker"**
sb.delete(sb.append("mit Frog");	**sb ⟶ "Kermit Frog"**
sb.insert(8,"The ");//insert at position 8	**sb ⟶ "Kermit The Frog"**
Sb.reverse();	**sb ⟶ gorF ehT timreK**

In each case, the StringBuilder object is changed. A new one is not created.

Here is an exercise that compares Strin and StringBuilder methods

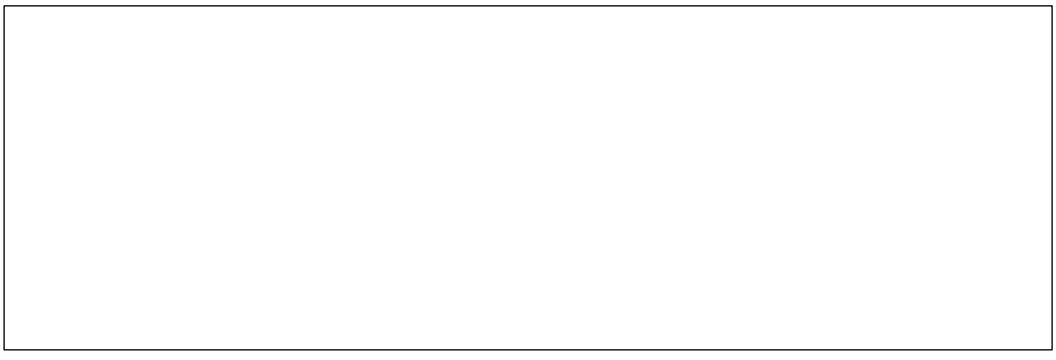
```
public class Strings
{
  public static void main(String[] args)
  {
    String alphabet         = new String("abcdefghijklmnopqustuvwxyz");
    StringBuilder alphabet1 = new StringBuilder("abcdefghijklmnopqustuvwxyz");
```

// removes "efghi" from the alphabet using String – fill in the code



// removes "efghi" from the alphabet1 using StringBuilder

//reverse a sequence of characters on alphabet using String



//reverse a sequence of characters in alphabet1 using StringBuilder

alphabet1 = new StringBuilder("abcdefghijklmnopqustuvwxyz"); **// alphabet1 had been changed**

// What is the output?

```
String a = new String("Homer");
String b = new String ("Homer");
```

// == vs equals for String and StringBuilder

```
System.out.println("Using equals with String : "+ a.equals(b));
System.out.println("Using == with String : "+ a == b);
```

```
StringBuilder c = new StringBuilder("Homer");
StringBuilder d = new StringBuilder ("Homer");

System.out.println("Using equals with StringBuilder : "+ c.equals(d));
System.out.println("Using == with StringBuilder : "+ (c == d));
System.out.println("Using equals with StringBuilder and toString() : "+
                                                    c.toString().equals(d.toString()));
```

```
}
}
```