Escape Sequence

- Strings can contain any character, but some of them must be “escaped” in order to write them in a literal.
- Each of these is written as a two-character sequence, but represents a single character in the string:
  - `\"` stands for the double-quote (" ) character
  - `\n` stands for the newline character
  - `\\` stands for the backslash (\ ) character
  - `\t` stands for the tab spacing
  - `\0` stands for null character indicating nothing
  - Some terminals incorrectly display null as space character

  E.g. System.out.println("\"This is in quotes\"");

  "This is in quotes"

Useful String methods I

- `boolean equals(Object obj)`
  - Tests if this String is the same as the obj (which may be any type; false if it’s not a String)

- `boolean equalsIgnoreCase(String other)`
  - Tests if this String is equal to the other String, where case does not matter

- `int length()`
  - Returns the length of this string; note that this is a method, not an instance variable

Still more about equals

- Suppose you want to test whether a variable name has the value "Dave"
  - Here’s the obvious way to do it:
    ```java
    if (name.equals("Dave")) { ... }
    ```
  - But you could also do it this way:
    ```java
    if ("Dave".equals(name)) { ... }
    ```

- It turns out that the second way is usually better

- Why?
  - If name == null, then first way will cause NullPointerException
  - But the second way will just return false
Useful String methods II

- **char charAt(int index)**
  - Returns the character at the given index position (0-based)

- **boolean startsWith(String prefix)**
  - Tests if this String starts with the prefix String

- **boolean endsWith(String suffix)**
  - Tests if this String ends with the suffix String

Useful String methods III

- **int indexOf(char ch)**
  - Returns the position of the first occurrence of ch in this String, or -1 if it does not occur

- **int indexOf(char ch, int fromIndex)**
  - Returns the position of the first occurrence of ch, starting at (not after) the position fromIndex

  With `charAt(index)`, `indexOf(x)`, just count characters (starting from zero)

```
"She said, \"Hi\\""
```

Useful String methods IV

- **int lastIndexOf(char ch)**
  - Returns the position of the last occurrence of ch in this String, or -1 if it does not occur

- **int lastIndexOf(char ch, int fromIndex)**
  - Returns the position of the last occurrence of ch, searching backward starting at position fromIndex

  There are two similar methods that take a String instead of a char as their first argument

Useful String methods V

- **String substring(int beginIndex)**
  - Returns a new string that is a substring of this string, beginning with the character at the specified index and extending to the end of this string.

- **String substring(int beginIndex, int endIndex)**
  - Returns a new string that is a substring of this string, beginning at the specified `beginIndex` and extending to the character at index `endIndex - 1`.
  - Thus the length of the substring is `endIndex - beginIndex`
Substring Example

- With `substring(from, to)`, it works better to count positions between characters

  "She said, "Hi"

- So, for example, `substring(4, 8)` is "said", and `substring(8, 12)` is ", "Hi"

- If `indexOf(\',\')` is 8, then `substring(0, indexOf(\',\'))` is "She said" and `substring(indexOf(\',\') + 1)` is "Hi"

Strings are immutable

- A String, once created, cannot be changed
- *None* of the preceding methods modify the String, although several create a new String
- Statements like this create new Strings:
  ```java
  myString = myString + anotherCharacter;
  ```
- Creating a few extra Strings in a program is no big deal
- Creating a *lot* of Strings can be very costly

Useful String methods VI

- **String toUpperCase()**
  - Returns a new String similar to this String, in which all letters are uppercase

- **String toLowerCase()**
  - Returns a new String similar to this String, in which all letters are lowercase

- **String trim()**
  - Returns a new String similar to this String, but with whitespace removed from both ends
    - “whitespace” means any spaces or tab characters