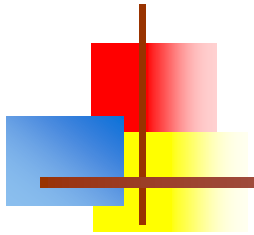
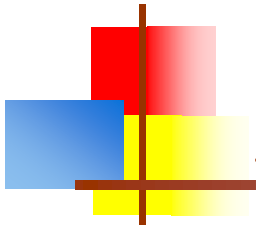


String and Char

Part I: String



About Strings

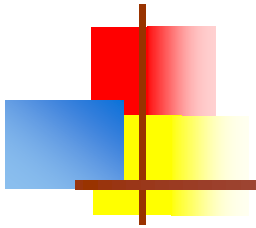
- Strings are objects, but there is a special syntax for writing String literals:

  "Hello"

- Strings, unlike most other objects, have a defined *operation* (as opposed to a *method*):

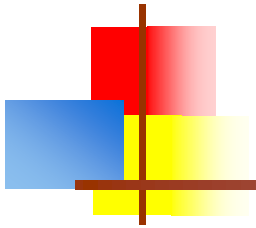
  " This " + "is String " + "concatenation""

- Strings can contain any character, but some of them must be "escaped" in order to write them in a literal
  - \" stands for the double-quote (") character
  - \n stands for the newline character
  - \\ stands for the backslash (\)character
  - Each of these is *written as* a two-character sequence, but represents a *single* character in the string



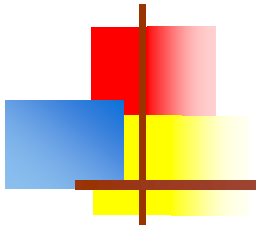
Useful String methods I

- 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



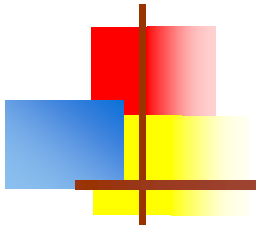
Useful String methods II

- 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



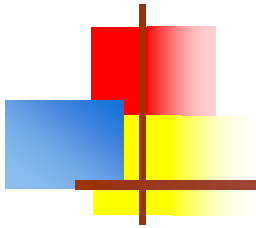
Useful String methods III

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



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

Understanding "index"

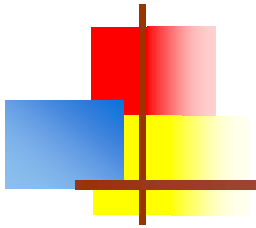
- With charAt(index), indexOf(x), and lastIndexOf(x), just count characters (starting from zero)

```
"She said, \"Hi\""
 0  1 2 3 4  5 6 7  8 9  10  11 12  13
```

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

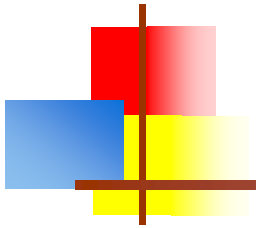
```
"She said, \"Hi\""
 0  1 2 3 4  5 6 7  8 9 10  11 12 13  14
```

- So, for example, substring(4, 8) is "said", and substring(8, 12) is ", \"H"
- If indexOf(',') is 8, then substring(0, indexOf(',')) is "She said" and substring(indexOf(',') + 1) is " \"Hi\""



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

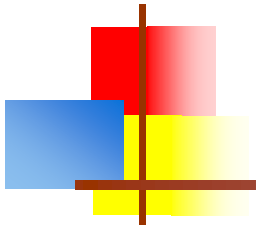


Useful String methods VII

- String[] split(String regex)
  - Breaks the string up into an array of strings
  - The parameter is a regular expression that defines what separates the strings
  - For example,

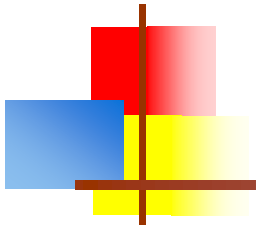
```
String s = "one, two, three";
String[] ss = s.split(", ");
```

    - This assigns the array {"one", "two", "three"} to ss
  - Regular expressions are complex expressions that assign meanings to many common punctuation marks, such as +, *, period, and [
    - Hence, regular expressions are powerful, but can be treacherous if you aren't very familiar with them



Finally, a *useless* String method

- String toString()
  - Returns this String
- Why do we have this method?
  - Consistency--*Every* Object has a toString() method

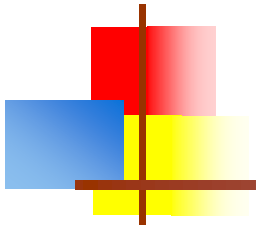


Strings are immutable

- 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:

```
myString = myString + anotherCharacter;
```

- Creating a few extra Strings in a program is no big deal
- Creating a *lot* of Strings can be very costly



More about equals

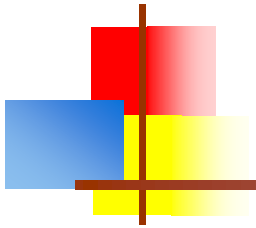
- If you write

  ```
  String s = "abc";
  String t = "abc";
  ```

  the compiler only creates the string "abc" once, and makes s and t both refer to this *one* string
  - It can do this because strings are immutable
  - Hence, the test s == t will be true
- However, if you now write

  ```
  String u = "a" + "bc";
  ```

  the test s == u will be false
  - This is because they are different strings
- Moral: Use equals for strings, not ==



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:

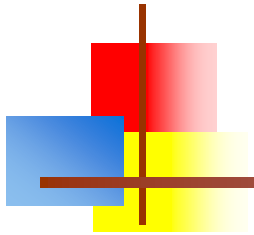
    ```
    if (name.equals("Dave")) { … }
    ```

  - But you *could* also do it this way:

    ```
    if ("Dave".equals(name)) { … }
    ```

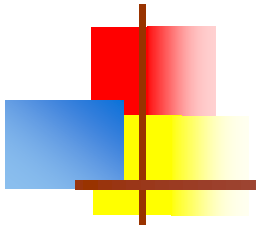
- It turns out that the *second* way is usually better
- **Why?**
  - If name == null,

    the first way will cause a NullPointerException, but the second way will just return false



Strings, etc.

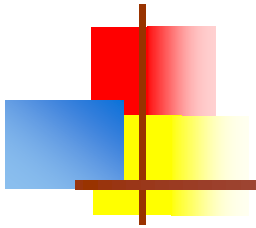
Part II: Characters



The Character class

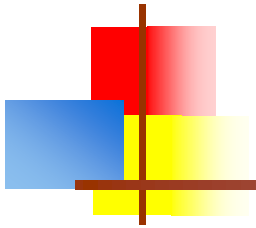
- char is a primitive type, not an object, therefore…
- …there are no methods you can call on a char
- This is why we need a Character class!

- There are a lot of methods in the Character class
  - They are all static
  - This means we talk to the *class*, not to an individual char
  - ch2 = Character.toUpperCase(ch1);



Some Character methods

- static boolean isDigit(char ch)
- static boolean isLetter(char ch)
- static boolean isLetterOrDigit(char ch)
- static boolean isLowerCase(char ch)
- static boolean isUpperCase(char ch)
- static boolean isWhitespace(char ch)
- static char toLowerCase(char ch)
- static char toUpperCase(char ch)
- For more methods, see java.lang.Character



The End