

---

# Searching and Strings

IST 256

Application Programming for Information Systems

# Searching for Strings

---

- In an array, we do a simple linear search for an item by going through the array in order from the first and comparing each item to the search string.

```
String [ ] names = new String [20];
String searchname = ... ; // get a value to search for
for (int i = 0; i < names.length; i ++)
{
    if ( names[i].equals(searchname))
    {
        //found one!
    }
}
```

# Searching for Strings

---

- If we want to find only one (the first one), we can save it and exit from the loop:

```
String [ ] names = new String [20];
String searchname = ... ; // get a value to search for
String resultname = "";
for (int i = 0; i < names.length; i ++ ) {
    if ( names[i].equals(searchname))
    {
        resultname = names[i];
        break;
    }
}
// display the result
```

“break” causes the program to break out of the loop; immediately jump to the end

# Searching for Strings

---

- But if we don't find one, we can display that, too:

```
String [ ] names = new String [20];
String searchname = ... ; // get a value to search for
String resultname = "";
for (int i = 0; i < names.length; i ++) {
    if ( names[i].equals(searchname))
    {
        resultname = names[i];
        break;
    }
}
// if still empty result, then it was not found
if (resultname.equals("")) {
    resultname = "No match found."
}
// display resultname
```

# Searching for Strings

---

- If there's more than one match, we can find them all:

```
String [ ] names = new String [20];
String searchname = ... ; // get a value to search for
String resulthead = "<html>Result: <p>:";
String resultnames = "";
for (int i = 0; i < names.length; i++) {
    if ( names[i].equals(searchname))
    {
        resultnames = resultnames + "<p>" + names[i];
    }
}
// if still empty result, then none was found
if (resultnames.equals("")) {
    resultnames = "No match found."
}
// display resulthead + resultnames
```

# Search Criteria

---

- So far, we have used an exact string match with the string function equals

```
names[i].equals(searchname)
```

- Suppose that we want to find names where the search name is the start of the name (e.g. “Alex Brown” and we search for “Al” or “Alex”)

```
names[i].startsWith(searchname)
```

uses the String function startsWith

# Search Criteria

---

- Or suppose that we want the strings to match regardless of capitalization. This is called “ignoring case”, and there is another equals method that will ignore case in doing the comparison:

```
names[i].equalsIgnoreCase(searchname)
```

- For other comparisons, such as startsWith(), use the .toLowerCase() method to convert both strings to lower case and then compare them.

```
names[i].toLowerCase().startsWith(  
    searchname.toLowerCase())
```

# Search Criteria

---

- Or suppose that we want to find the search string as a substring of the name, anywhere that it occurs. There is a String function contains that tests if a substring occurs anywhere in the string

```
names[i].contains(searchname)
```



# Search Criteria

---

- Here is an alternate way to find if the search name is a substring of the name. There is a String function `indexOf` that finds a substring and returns an integer that is the number of the character where the substring occurs (starting from 0)

if searchname is “Alex”,  
    `searchname.indexOf(“ex”)`  
returns the integer 2, the position where “ex” is in “Alex”

- But if the substring is not found, `indexOf` returns -1, so our search criteria becomes  
    `names[i].indexOf(searchname) != -1`

# Other String Methods

---

- Other useful functions include:
  - `endsWith()`
  - `lastIndexOf()`
  - `toUpperCase()`
  - `substring(startIndex, stopIndex)` returns the substring that starts at the character at `startIndex` and ends one character before the `stopIndex`.
  - `compareTo(compareString)` is `True` if the `String` is less than the `compareString` and `false` otherwise. This is useful for sorting arrays in alphabetic order by name.

# Finding more String methods

---

- A complete list is found in the API under the String class:
  - <http://docs.oracle.com/javase/7/docs/api/>
  - Where you scroll down to find the String class in the lower left pane
  - And note that String “implements” the CharSequence class, so you are allowed to pass a String to any method that asks for a CharSequence parameter.

# Digression on primitive types vs. classes

- In Java, primitive types (int, double, boolean) use operators, while classes have fields and methods. But arrays and Strings have a mix:

|                                       | Operators              | Use new constructor | Use fields and methods         |
|---------------------------------------|------------------------|---------------------|--------------------------------|
| Primitive types: int, double, boolean | Yes<br>+, -, *, /, &&, | No                  | No                             |
| Arrays                                | Indexing: [ ]          | Mostly: new int [ ] | .length                        |
| Strings                               | Catenation: +          | No                  | Mostly: .equals, indexOf, etc. |
| Everything else                       | No                     | Yes                 | Yes                            |