

FUNCTION OBJECTS

Instructors: Crista Lopes
Copyright © Instructors.

How to model functions as data in OOP?

```
7 def read_file(path_to_file, func):
8     with open(path_to_file) as f:
9         data = f.read()
10        func(data, normalize)
11
12 def filter_chars(str_data, func):
13     pattern = re.compile('[\W_]+')
14     func(pattern.sub(' ', str_data), scan)
15
16 def normalize(str_data, func):
17     func(str_data.lower(), remove_stop_words)
18
19 def scan(str_data, func):
20     func(str_data.split(), frequencies)
21
22 def remove_stop_words(word_list, func):
23     with open('../stop_words.txt') as f:
24         stop_words = f.read().split(',')
25         # add single-letter words
26         stop_words.extend(list(string.ascii_lowercase))
27     func([w for w in word_list if not w in stop_words], sort)
28
29 def frequencies(word_list, func):
30     wf = {}
31     for w in word_list:
32         if w in wf:
33             wf[w] += 1
34         else:
35             wf[w] = 1
36     func(wf, print_text)
37
38 def sort(wf, func):
39     func(sorted(wf.iteritems(), key=operator.itemgetter(1),
40                reverse=True), no_op)
41
42 def print_text(word_freqs, func):
43     for (w, c) in word_freqs[0:25]:
44         print w, "-", c
45     func(None)
46
47 def no_op(func):
48     return
49
50 #
51 # The main function
52 #
53 read_file(sys.argv[1], filter_chars)
```

```
64 TFTheOne(sys.argv[1]) \
65     .bind(read_file) \
66     .bind(filter_chars) \
67     .bind(normalize) \
68     .bind(scan) \
69     .bind(remove_stop_words) \
70     .bind(frequencies) \
71     .bind(sort) \
72     .bind(top25_freqs) \
73     .printme()
```

First attempt

```
class ReadFile {  
    String readFile(string arg)  
    {  
        string contents = // Code for reading a file  
        return contents;  
    }  
}
```

and then

```
new ReadFile().readFile(...);
```

or

```
ReadFile rf = new ReadFile();  
someObj.meth(rf);
```

First attempt

```
class FilterChars
{
    String filter(String arg)
    {
        string fcontents = // Code for filtering
        return fcontents;
    }
}
```

same

Problem: no commonality between function classes

```
class TFTheOne
{
    private Object value;
    TFTheOne(Object v) { value = v; }

    public TFTheOne bind(??? func) {
        value = func.???(value);
        return this;
    }

    public printme() {
        System.out.println(value);
    }
}
```

Object?

→ Giant switch statement

Problem: no commonality between function classes

```
class TFTheOne
{
    private Object value;
    TFTheOne(Object v) { value = v; }

    public TFTheOne bind(Object func) {
        if (func instanceof ReadFile)
            value = func.ReadFile(value);
        else if (func instanceof FilterChars)
            value = func.FilterChars(value);
        else if ...

        return this;
    }
}
```

→ Giant switch statement

A Function Object Interface

```
interface IFunction
{
    Object call(Object arg);
}
```

More variations are possible...

The One – Function examples

```
class ReadFile implements IFunction
{
    Object call(Object arg)
    {
        string fileName = (String)arg;
        string contents = // Code for reading a file
        return contents;
    }
}
```

```
class FilterChars implements IFunction
{
    Object call(Object arg)
    {
        string data = (String)arg;
        string fcontents = // Code for filtering
        return fcontents;
    }
}
```


The One

```
class TFTheOne
{
    private Object value;
    TFTheOne(Object v) { value = v; }
```

```
    public TFTheOne bind(IFunction func) {
        value = func.call(value);
        return this;
    }
```

```
    public printme() {
        System.out.println(value);
    }
}
```

Beautifully
generic!

The One – Main

```
public static void main(String[] args) {  
    TFTheOne one = new TFTheOne(args[...]);  
  
    one.bind(new ReadFile()).bind(new FilterChars()).bind ...  
}
```

Kick Forward (CPS)

```
interface IFunction
{
    void call(Object arg, IFunction func);
}
```

and then

```
class ReadFile implements IFunction
{
    void call(Object arg, IFunction func)
    {
        string fileName = (String)arg;
        string contents = // Code for reading a file
        func(contents, new FilterChars());
    }
}
```

Functions as Objects

- ❑ Define a clean “function” interface
- ❑ Define one class per function, each implementing the interface
- ❑ When you need to pass the function, instantiate the class and pass that object

More generally

- Objects are capsules of procedures/functions
 - ▣ with or without state
- Functions as data in pure OOP → function objects
- If you need to manipulate them independent of their type → use interface / super