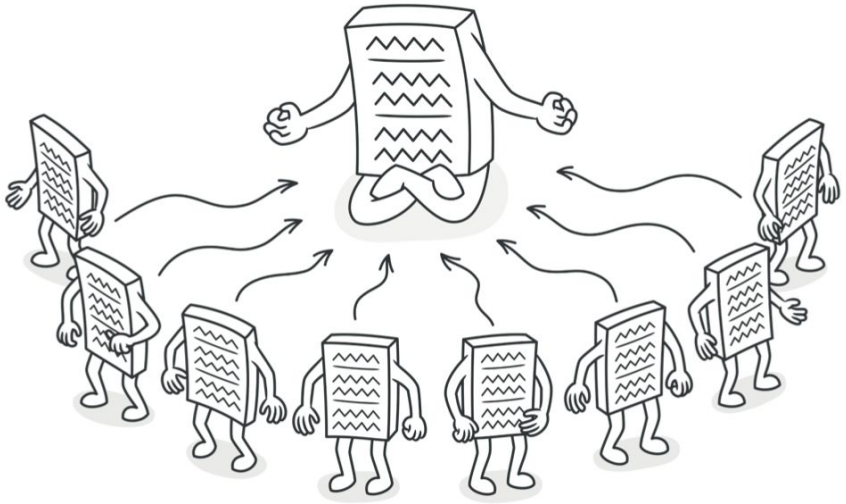
Lecture 6: Design Patterns
Singleton
...

Singleton Design Pattern

Singleton is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance.



Singleton

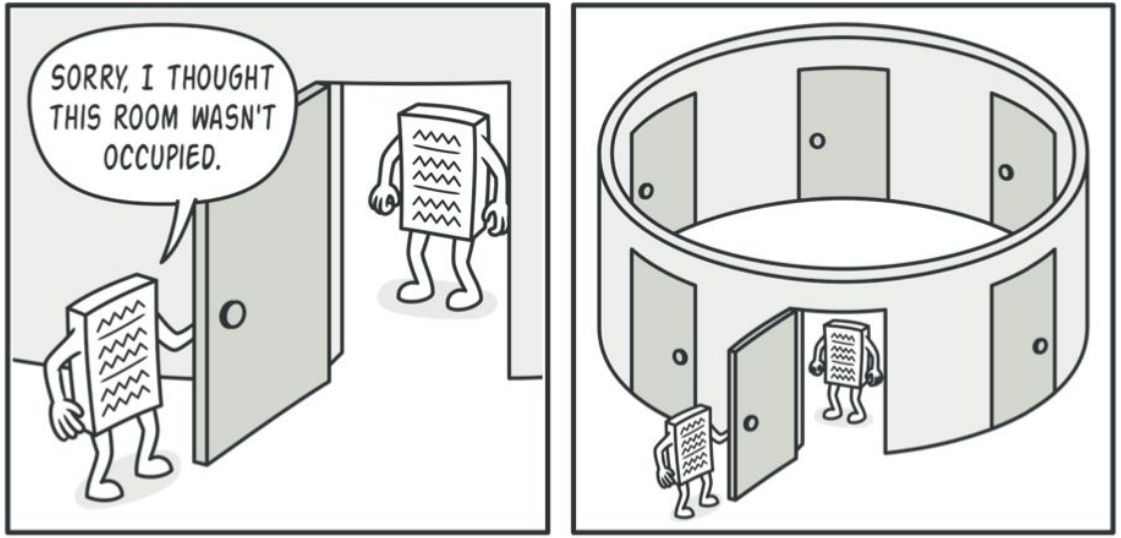
The Singleton pattern ensure that a class has just a single instance.

The most common reason for this is to control access to some shared resource—for example, a database or a file.

Here's how it works: imagine that you created an object, but after a while decided to create a new one. Instead of receiving a fresh object, you'll get the one you already created.

Note that this behavior is impossible to implement with a regular constructor since a constructor call must always return a new object by design.

Singleton



Clients may not even realize that they're working with the same object all the time.

Singleton

Provides a global access point to that instance. Remember those global variables that you (all right, me) used to store some essential objects? While they're very handy, they're also very unsafe since any code can potentially overwrite the contents of those variables and crash the app.

Just like a global variable, the Singleton pattern lets you access some object from anywhere in the program. However, it also protects that instance from being overwritten by other code.

Singleton

The singleton design pattern is useful when you need to create only one object or you need some sort of object capable of maintaining a global state for your program.

Other possible use cases are:

Controlling concurrent access to a shared resource. For example, the class managing the connection to a database.

A service or resource that is transversal in the sense that it can be accessed from different parts of the application or by different users and do its work. For example, the class at the core of the logging system or utility.

Singleton: Real-World Analogy

The government is an excellent example of the Singleton pat- tern. A country can have only one official government. Regard- less of the personal identities of the individuals who form governments, the title, “The Government of X”, is a global point of access that identifies the group of people in charge.

Singleton

The Singleton class declares the static method getInstance that returns the same instance of its own class.

Calling the getInstance method should be the only way of getting the Singleton object.

Singleton: Example

Let's implement a program to fetch content from web pages

We will use the urllib module to connect to web pages using their URLs; the core of the program would be the URLFetcher class that takes care of doing the work via a fetch() method.

We want to be able to track the list of web pages that were tracked, hence the use of the singleton pattern: we need a single object to maintain that global state.

Singleton: Example

First, our naive version, to help us track the list of URLs that were fetched, would be:

```
class URLFetcher:
    def __init__(self):
        self.urls = []

    def fetch(self, url):
        req = urllib.request.Request(url)
        with urllib.request.urlopen(req) as response:
            if response.code == 200:
                the_page = response.read()
                print(the_page)
                urls = self.urls
                urls.append(url)
                self.urls = urls
```

Does that example create a singleton?

What will happen if we run

```
print(URLFetcher() is URLFetcher())
```

Singleton in Python3

We first implement a metaclass for the singleton, meaning the class (or type) of the classes that implement the singleton pattern, as follows:

```
class SingletonType(type):
    _instances = {}

    def __call__(cls, *args, **kwargs):
        if cls not in cls._instances:
            cls._instances[cls] = super(SingletonType,
                                        cls).__call__(*args, **kwargs)
        return cls._instances[cls]
```

Singleton in Python3

Now, we will rewrite our URLFetcher class to use that metaclass. We also add a dump_url_registry() method, which is useful to get the current list of URLs tracked:

```
class URLFetcher(metaclass=SingletonType):
    def fetch(self, url):
        req = urllib.request.Request(url)
        with urllib.request.urlopen(req) as response:
            if response.code == 200:
                the_page = response.read()
                print(the_page)
                urls = self.urls
                urls.append(url)
                self.urls = urls
    def dump_url_registry(self):
        return ', '.join(self.urls)
if __name__ == '__main__':
    print(URLFetcher() is URLFetcher())
```

Singleton in Python2 or without control of source

In Python2, the metaclass doesn't exist yet. Instead, we add a global var to hold the singleton and refer to it via a method to get the singleton.

```
_url_fetcher_instance = None

def get_url_fetcher():
    global _url_fetcher_instance
    if not _url_fetcher_instance:
        _url_fetcher_instance = URLFetcher()
    return _url_fetcher_instance
```

Singleton in Python2

```
class URLFetcher:
    """Note: this class should ONLY be used via get_url_fetcher method!"""
    def __init__(self):
        self.urls = []

    def fetch(self, url):
        req = urllib.request.Request(url)
        with urllib.request.urlopen(req) as response:
            if response.code == 200:
                the_page = response.read()
                print(the_page)
                urls = self.urls
                urls.append(url)
                self.urls = urls
```

Does this work?

```
print(get_url_fetcher() is get_url_fetcher())
```

Singleton: Side note

You can also do this by overriding the __new__ operator on the class in python

But I wouldn’t recommend it!! It’s confusing for other readers of code

Singleton: Pros

You can be sure that a class has only a single instance.

You gain a global access point to that instance.

The singleton object is initialized only when it’s requested for the first time.

Singleton: Cons

TODO talk about state changes and basically the object cannot have state that affects runtime behavior

Prototype Design Pattern

A prototype design pattern is used for creating exact copies of objects

This used to be hard/manual in Java and an entire pattern was set for it.

Python does it automatically with the copy library!

Prototype and python object copying

Definitions:

In the general case of creating a copy of an object, what happens is that you make a new reference to the same object, a method called a shallow copy.

But, if you need to duplicate the object, which is the case with a prototype, you make a deep copy.