Implementing registries using Metaclasses

Andreas Madsack (AX Semantics)

TOC:

Problem

For example, we have an implementation for a surface realizer in **a few** languages.
All of them have the **same interfaces**.
We don't want to load them via imports, but get them from a **registry**:

In [2]:
```
german = Registry.get_language_instance("de-DE")
```

Solution - Decorators

Used for example in **AllenNLP** or **Thinc** to register a part of a neural network.
Or **Flask** to register url routes.
Could look like:

```
In [4]: @registry_decorator("de-DE")
        class VerbRenderer_DE_DE():
            def render(self, lemma):
                return f"rendered: {lemma}"
```

```
In [5]: german = registry.get("de-DE")()
```

```
In [6]: german.render("foo")
```

```
Out[6]: 'rendered: foo'
```

Show me the code

In [7]:
```
registry = {}

def registry_decorator(language):
    def registry_func(func):
        @functools.wraps(func)
        def wrapper():
            registry[language] = func
        return wrapper()
    return registry_func
```

In [8]:
```
@registry_decorator("de-DE")
class VerbRenderer_DE_DE():
    def render(self, lemma):
        return f"rendered: {lemma}"
```

Solution - Metaclasses

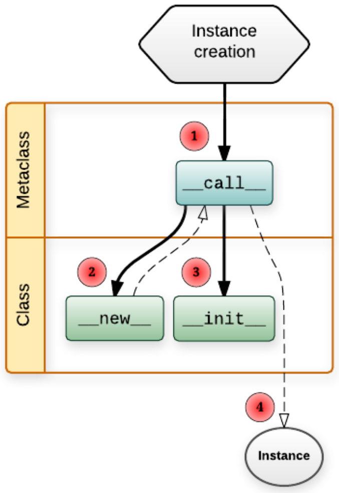
This solution is not per se better than decorators.
It just fitted our needs better.

Example:

In [10]:
```
class VerbRenderer_DE_DE(VerbRendererBase, language="de-DE"):
    def render(self, lemma):
        return f"rendered: {lemma}"
```

First a few words about metaclasses

Metaclasses allow to change class instance creation. Creation is happening at import time.



src: https://blog.ionelmc.ro/2015/02/09/understanding-python-metaclasses/ (https://blog.ionelmc.ro/2015/02/09/understanding-python-metaclasses/)

Show me the code

In [11]:
```
class RegisterMetaClass(ABCMeta):
    def __new__(cls, class_name, bases, namespace, language):
        # insert variables into class namespace
        namespace["language"] = language
        namespace["lang_code"] = language.split("-")[0].lower() if language else 
None

        new = super().__new__(cls, class_name, bases, namespace)
        if not language:
            return new

        registry[language] = new
        return new

class VerbRendererBase(metaclass=RegisterMetaClass, language=None):
    # this class is not in the registry, because language is not set!
    pass
```

Differences between 2 solutions?

- decorators seem like less magic for the user
- if you already use a baseclass, this baseclass can do the metaclass thingy

Registry

Both solutions need to be imported. Our solution:

```
In [12]: registry = {}

         class Registry:
             @classmethod
             def get_language_class(cls, name):
                 if not registry:
                     cls.load_modules()
                 return registry[name]

             @classmethod
             def get_language_instance(cls, name):
                 return cls.get_language_class(name)()

             @staticmethod
             def load_modules():
                 root = Path(__file__).resolve().parent

                 for module in root.glob("languages/*/*.py"):
                     module = str(module)
                     if not module.lower().endswith("__init__.py"):
                         module = module[module.index("example/languages") : -len(".py")]
         .replace(
                             "/", "."
                         )
                         import_module(module)
```

Live demo

In [13]:
```
!ls -l
```

```
total 28
drwxr-xr-x 4 mfa users  4096 Mar 26 11:13 example
-rw-r--r-- 1 mfa users 12636 Mar 25 20:46 Implementing-registries-using-Metacl
asses.ipynb
-rw-r--r-- 1 mfa users   612 Mar 26 11:15 README.md
drwxr-xr-x 3 mfa users  4096 Mar 26 11:13 tests
```

In [14]:
```
!ls -l example
```

```
total 12
-rw-r--r-- 1 mfa users    0 Feb 17 17:49 __init__.py
drwxr-xr-x 5 mfa users 4096 Mar 22 19:38 languages
drwxr-xr-x 2 mfa users 4096 Mar 26 11:13 __pycache__
-rw-r--r-- 1 mfa users 1668 Mar 26 11:13 registry.py
```

```
In [15]: from example.registry import Registry
```

```
In [16]: Registry.supported_languages()
```

```
Out[16]: {'de-DE', 'en-US'}
```

```
In [17]: Registry.get_language_instance("de-DE").render("foo")
```

```
Out[17]: 'rendered (DE): foo'
```

```
In [18]: Registry.get_language_instance("en-US").render("foo")
```

```
Out[18]: 'rendered: foo'
```

```
In [19]: !cat example/languages/en/en_us.py
```

```
from ..base import ExampleBase


class Example_EN_US(ExampleBase, language="en-US"):
    pass
```

```
In [20]: !python -m pytest
```

```
============================= test session starts ==============================
===
platform linux -- Python 3.8.2, pytest-5.4.1, py-1.8.1, pluggy-0.13.1
rootdir: /home/mfa/git/registry-metaclasses
collected 3 items

tests/test_registry.py ...                                                [10
0%]

============================== 3 passed in 0.02s ===============================
===
```

Thanks!

code will be released here: https://github.com/mfa/registry-metaclasses (https://github.com/mfa/registry-metaclasses)