03. Catalog Pattern

CheolSeong Park
tjd4987@naver.com

03. Catalog Pattern

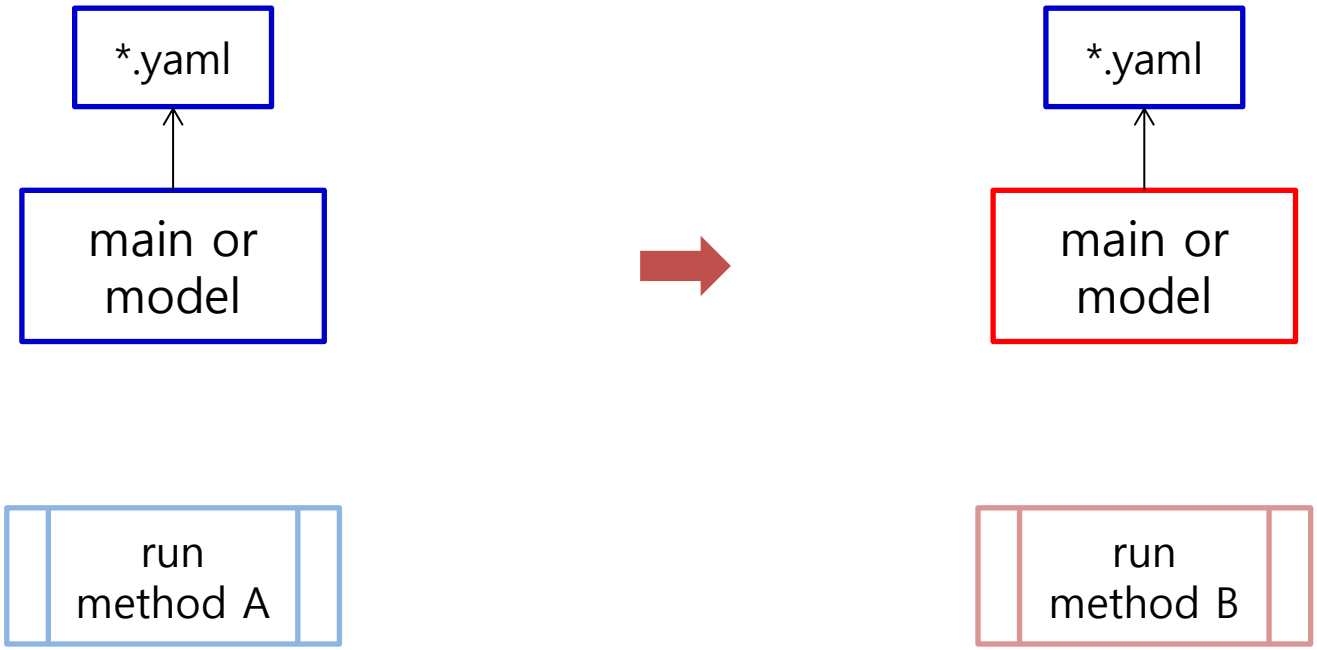
- 생성자 파라미터에 의해 지정된 method가 호출되도록 하는 디자인 패턴

03. Catalog Pattern

- 무슨 장점이 있나요?
  - 클래스 파라미터에 따라, 코드 수정 없이 클래스 내 특정 함수를 수행시킬 수 있습니다.
  - 클래스 내, 함수 호출 순서를 바꾸고 싶을 때, 코드 수정을 **덜** 할 수 있습니다.

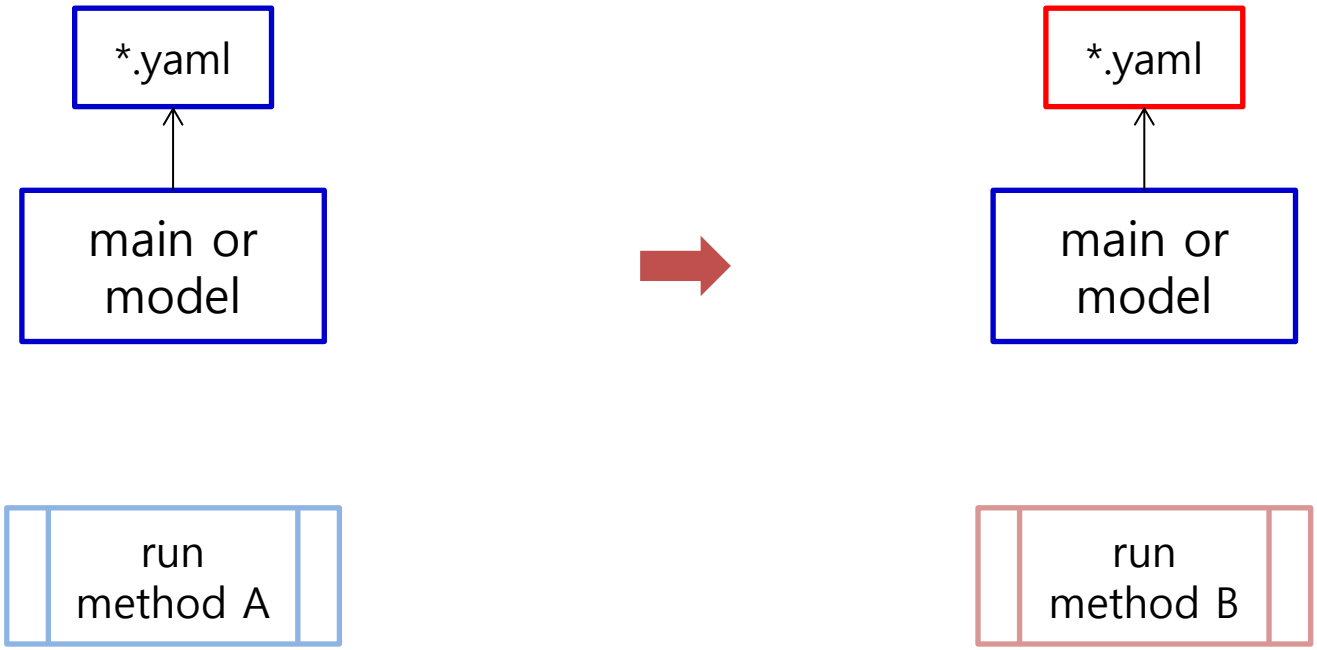
03. Catalog Pattern

- 기존
  - method A를 호출하는 모듈에서 method B를 호출하도록 바꾸기 위해 코드 수정이 불가피



03. Catalog Pattern

- Catalog 패턴 적용 시
  - YAML 변경만으로 method B를 수행하는 모듈로 변경 가능



03. Catalog Pattern

- ./configs/cnn.yaml과 resnet.yaml
  - 각각 vanilla forward와 residual forward를 수행하도록 함

```yaml
hyper_parameters:
  batch_size : 32
  epochs : 5
  learning_rate : 0.001

network_parameters:
  activation : 'relu'
  layer1 : "ConvLayer"
  layer2 : "ConvLayer"
  layer3 : "ConvLayer"
  forward_method : "vanilla"

data_scale_factor : 10

dataset_root : '../../00_data'
```

cnn.yaml

```yaml
hyper_parameters:
  batch_size : 32
  epochs : 5
  learning_rate : 0.001

network_parameters:
  activation : 'relu'
  layer1 : "ConvLayer"
  layer2 : "ConvLayer"
  layer3 : "ConvLayer"
  forward_method : "residual"

data_scale_factor : 10

dataset_root : '../../00_data'
```

resnet.yaml

03. Catalog Pattern

- MyModel_생성자
  - Decorator를 이용하여, yaml의 지정된 layer를 가져옵니다.

```tensorflow
class MyModel(Model):
    def __init__(self, cfg):
        super(MyModel, self).__init__()
        self.layer1 = get_layer(cfg['layer1'], cfg)
        self.layer2 = get_layer(cfg['layer2'], cfg)
        self.layer3 = get_layer(cfg['layer3'], {'activation': None})
        self.relu = tf.keras.layers.ReLU()
        self.flatten = Flatten()

        self.d1 = Dense(128, activation=cfg['activation'])
        self.d2 = Dense(10)

        self.fwd_method = cfg['forward_method']
```

tensorflow

```pytorch
class MyModel(nn.Module):
    def __init__(self, cfg):
        super(MyModel, self).__init__()
        _cfg = {k:val for k,val in cfg.items()}
        _cfg['in_channels'] = 1
        self.layer1 = get_layer(cfg['layer1'], _cfg)
        _cfg['in_channels'] = 32
        self.layer2 = get_layer(cfg['layer2'], _cfg)
        _cfg['activation'] = 'none'
        self.layer3 = get_layer(cfg['layer3'], _cfg)
        self.relu = nn.ReLU()
        self.flatten = nn.Flatten()

        self.d1 = nn.Linear(self.layer3.outdim,128)
        self.d2 = nn.Linear(128,10)

        self.fwd_method = cfg['forward_method']
```

pytorch

03. Catalog Pattern

- MyModel_forward
  - 생성자에서 정의된 self.fwd_method에 따라, fwd 함수 중 지정된 함수를 수행합니다.

```tensorflow
def vanilla_fwd(self, x):
    y1 = self.layer1(x)
    y2 = self.layer2(y1)
    y3 = self.layer3(y2)
    y3 = self.relu(y3)
    y3 = self.flatten(y3)
    d1 = self.d1(y3)
    return self.d2(d1)
def residual_fwd(self,x):
    y1 = self.layer1(x)
    y2 = self.layer2(y1)
    y3 = self.layer3(y2)
    y3 = self.relu(y1+y3)
    y3 = self.flatten(y3)
    d1 = self.d1(y3)
    return self.d2(d1)

_forward_catalog = {
    "vanilla": vanilla_fwd,
    "residual": residual_fwd,
}

def call(self, x):
    return self._forward_catalog[self.fwd_method].__get__(self)(x)
```

tensorflow

```pytorch
def vanilla_fwd(self, x):
    y1 = self.layer1(x)
    y2 = self.layer2(y1)
    y3 = self.layer3(y2)
    y3 = self.relu(y3)
    y3 = self.flatten(y3)
    d1 = self.relu(self.d1(y3))
    return self.d2(d1)
def residual_fwd(self,x):
    y1 = self.layer1(x)
    y2 = self.layer2(y1)
    y3 = self.layer3(y2)
    y3 = self.relu(y1+y3)
    y3 = self.flatten(y3)
    d1 = self.relu(self.d1(y3))
    return self.d2(d1)

_forward_catalog = {
    "vanilla": vanilla_fwd,
    "residual": residual_fwd,
}

def forward(self, x):
    return self._forward_catalog[self.fwd_method].__get__(self)(x)
```

pytorch

03. Catalog Pattern

- 나머지 코드는 지난번 02. Decorator와 거의 변화가 없습니다.

03. Catalog Pattern

- 실행 예시
  - python main.py --config ./configs/cnn.yaml
    - vanilla forward 수행
  - python main.py --config ./configs/resnet.yaml
    - residual forward 수행

03. Catalog Pattern

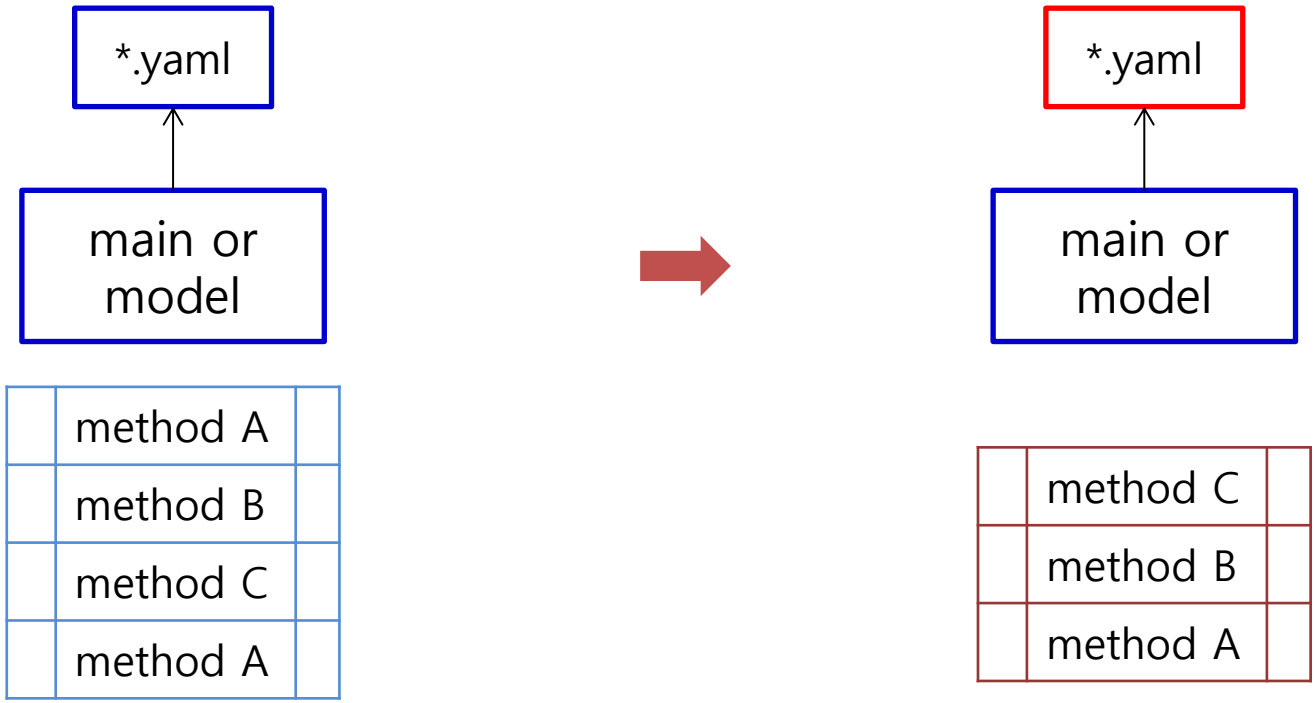
- if-else 코드와 다른점

```
if fwd_method == 'vanilla'
       vanilla_fwd()
else if fwd_method == 'residual'
       residual_fwd()
```

  - method 호출 순서를 yaml 변경만으로 조정할 수 있습니다.
    - Ex)
      - method A → method B → method C → method A
      - method C → method B → method A
    - **상기 if-else 코드는 수정없이 불가능**

03. Catalog Pattern

- Catalog 패턴 적용 시
  - YAML 변경만으로 method 순서를 조정할 수 있다.



03. Catalog Pattern

- 04_catalog_advanced에 catalog 패턴 응용 실습 코드가 하나 더 있습니다.
  - activation 후 batch_norm
    vs batch_norm 후 activation 실험 코드

```yaml
hyper_parameters:
  batch_size : 32
  epochs : 5
  learning_rate : 0.001

network_parameters:
  after_layer :
    - "bn"
    - "relu"

data_scale_factor : 10
```

relu_bn.yaml

```yaml
hyper_parameters:
  batch_size : 32
  epochs : 5
  learning_rate : 0.001

network_parameters:
  after_layer :
    - "relu"
    - "bn"

data_scale_factor : 10
```

bn_relu.yaml