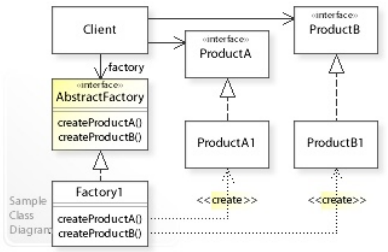
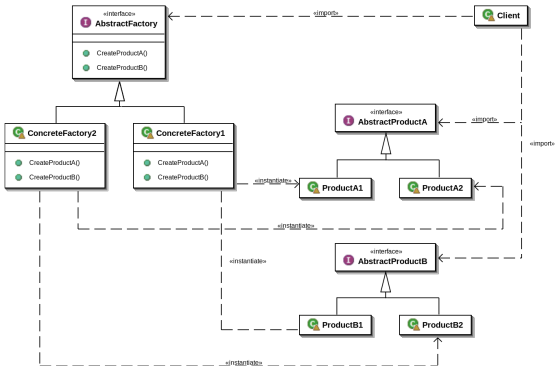
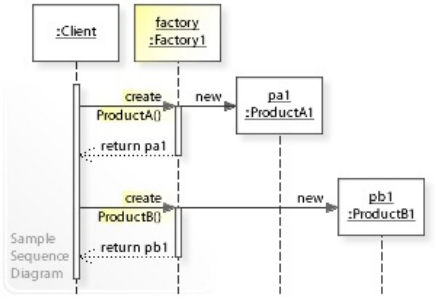
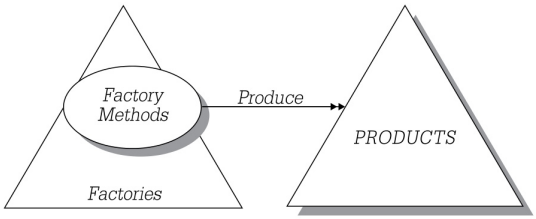
① Design pattern description

Abstract Factory c Creational Pattern





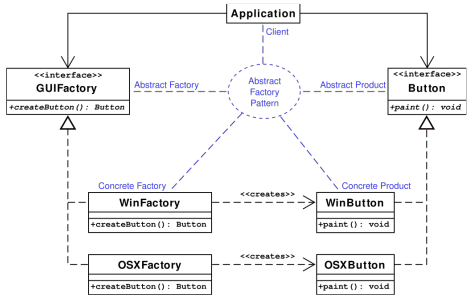




(2) Design pattern example

```python
appearance = "linux"

if appearance == "linux":
    factory = LinuxFactory()
elif appearance == "osx":
    factory = MacOSFactory()
elif appearance == "win":
    factory = WindowsFactory()
else:
    raise NotImplementedError(
        "Not implemented for your platform: {}".format(appearance)
    )

if factory:
    button = factory.create_button()
    result = button.paint()
    print(result)
```

```python
class GUIFactory:
    __metaclass__ = ABCMeta

    @abstractmethod
    def create_button(self):
        pass
```




```python
from abc import ABCMeta, abstractmethod

class Button:
    __metaclass__ = ABCMeta

    @abstractmethod
    def paint(self):
        pass
```

```python
class LinuxFactory(GUIFactory):
    def create_button(self):
        return LinuxButton()

class WindowsFactory(GUIFactory):
    def create_button(self):
        return WindowsButton()

class MacOSFactory(GUIFactory):
    def create_button(self):
        return MacOSButton()
```

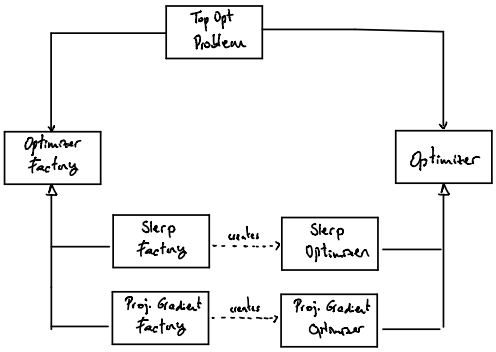
```python
class LinuxButton(Button):
    def paint(self):
        return "Render a button in a Linux style"

class WindowsButton(Button):
    def paint(self):
        return "Render a button in a Windows style"

class MacOSButton(Button):
    def paint(self):
        return "Render a button in a MacOS style"
```

③ Existing Example in FEM-MAT-OO ?

Not explicitly, but almost

④ Design pattern proposal in FEM-MAT-OO ?

Ex 1




```
class TopOptProblem
  properties (Access = private)
    optimizer
  end

  function create Optimizer (obj, Settings)
    switch Settings.algorithm
      case 'SLERP'
        factory = SlerpFactory (Settings)
      case 'PROJECTED-GRADIENT'
        factory = ProjectedGradientFactory (Settings)
    end
    obj.optimizer = factory.create
    obj.optimizer.solve()
  end
end
```

Ex 2