python design patterns Cheat Sheet

by mutanclan (mutanclan) via cheatography.com/79625/cs/19395/

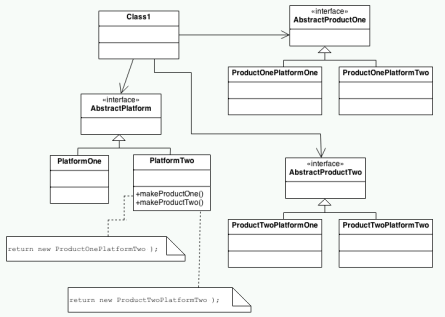
Abstract Factory Intention

Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

A hierarchy that encapsulates: many possible
"platforms", and the construction of a suite of
"products".

The new operator considered harmful.

Abstract Factory UML



Abstract Factory

```
import abc
class Abstra ctF act ory (me -
tac las s=a bc.A BC Meta):
        @ab c.a bst rac tmethod
       def create _pr odu ct_ -
a(s elf):
               pass
        @ab c.a bst rac tmethod
       def create _pr odu ct_ -
b(s elf):
               pass
class Concre teF act ory 1(A -
bst rac tFa ctory):
       def create _pr odu ct_ -
a(s elf):
                return Concre -
teP rod uctA1()
       def create _pr odu ct_ -
b(s elf):
                return Concre -
teP rod uctB1()
```

Abstract Factory (cont)

```
> class ConcreteFactory2(AbstractFactory):
    def create_product_a(self):
        return ConcreteProductA2()
    def create_product_b(self):
        return ConcreteProductB2()
class AbstractProductA(metaclass=abc.AB-
CMeta):
    @abc.abstractmethod
    def interface_a(self):
        pass
class ConcreteProductA1(AbstractProd-
uctA):
    def interface_a(self):
        pass
class ConcreteProductA2(AbstractProd-
uctA):
    def interface_a(self):
        pass
class AbstractProductB(metaclass=abc.AB-
CMeta):
    @abc.abstractmethod
    def interface_b(self):
        pass
class ConcreteProductB1(AbstractProd-
uctB):
    def interface_b(self):
        pass
class ConcreteProductB2(AbstractProd-
uctB):
    def interface_b(self):
        pass
def main():
    for factory in (ConcreteFactory1(),
                    ConcreteFactory2()):
        product_a = factory.create_product_a()
        product_b = factory.create_product_b()
```

Abstract Factory (cont)

```
>         product_a.interface_a()
        product_b.interface_b()
if __name__ == "__main__":
    main()
```