

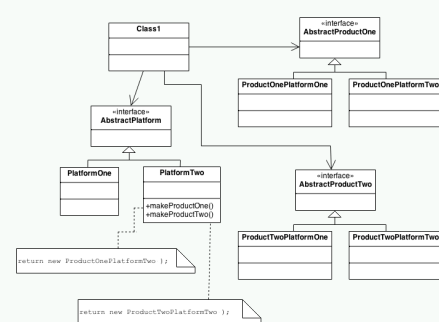
Abstract Factory Intention

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 AbstractFactory(metaclass=abc.ABCMeta):
    @abc.abstractmethod
    def create_product_a(self):
        pass

    @abc.abstractmethod
    def create_product_b(self):
        pass

class ConcreteFactory1(AbstractFactory):
    def create_product_a(self):
        return ConcreteProductA1()

    def create_product_b(self):
        return ConcreteProductB1()
```

Abstract Factory (cont)

```
> class ConcreteFactory2(AbstractFactory):
    def create_product_a(self):
        return ConcreteProductA2()
    def create_product_b(self):
        return ConcreteProductB2()

class AbstractProductA(metaclass=abc.ABCMeta):
    @abc.abstractmethod
    def interface_a(self):
        pass

class ConcreteProductA1(AbstractProductA):
    def interface_a(self):
        pass

class ConcreteProductA2(AbstractProductA):
    def interface_a(self):
        pass

class AbstractProductB(metaclass=abc.ABCMeta):
    @abc.abstractmethod
    def interface_b(self):
        pass

class ConcreteProductB1(AbstractProductB):
    def interface_b(self):
        pass

class ConcreteProductB2(AbstractProductB):
    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()
```