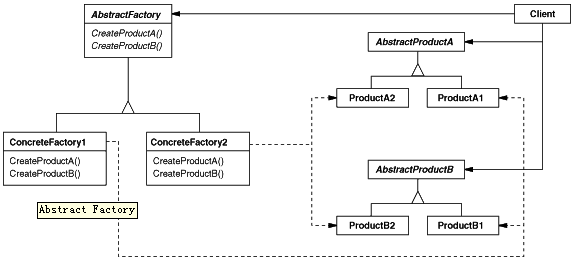
***2. Abstract Factory*（抽象工厂）**



**意图：**

提供一个创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类。   
**适用性：**

 一个系统要独立于它的产品的创建、组合和表示时。

 一个系统要由多个产品系列中的一个来配置时。

 当你要强调一系列相关的产品对象的设计以便进行联合使用时。

 当你提供一个产品类库，而只想显示它们的接口而不是实现时。

1. *#!/usr/bin/python*
2. *#coding:utf8*
3. '''
4. Abstract Factory
5. '''
7. import random
9. class PetShop:
10. """A pet shop"""
12. def\_\_init\_\_(self, animal\_factory=None):
13. """pet\_factory is our abstract factory.
14. We can set it at will."""
16. self.pet\_factory= animal\_factory
18. defshow\_pet(self):
19. """Creates and shows a pet using the
20. abstract factory"""
22. pet= self.pet\_factory.get\_pet()
23. print("This is a lovely",str(pet))
24. print("It says", pet.speak())
25. print("It eats",self.pet\_factory.get\_food())

28. *# Stuff that our factory makes*
30. classDog:
31. defspeak(self):
32. return "woof"
34. def\_\_str\_\_(self):
35. return "Dog"

38. classCat:
39. defspeak(self):
40. return "meow"
42. def\_\_str\_\_(self):
43. return "Cat"

46. *# Factory classes*
48. classDogFactory:
49. defget\_pet(self):
50. return Dog()
52. defget\_food(self):
53. return "dog food"

56. classCatFactory:
57. defget\_pet(self):
58. return Cat()
60. defget\_food(self):
61. return "cat food"

64. *# Create the proper family*
65. def get\_factory():
66. """Let's be dynamic!"""
67. return random.choice([DogFactory, CatFactory])()

70. *# Show pets with various factories*
71. if \_\_name\_\_ =="\_\_main\_\_":
72. shop= PetShop()
73. fori in range(3):
74. shop.pet\_factory= get\_factory()
75. shop.show\_pet()
76. print("="\* 20)