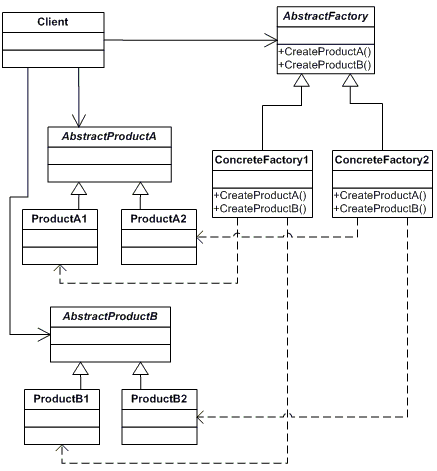
**Abstract Factory Design Pattern**

|  |  |
| --- | --- |
| http://www.dofactory.com/Images/redarrow.gif [definition](http://www.dofactory.com/Patterns/PatternAbstract.aspx#intent) http://www.dofactory.com/Images/redarrow.gif [UML diagram](http://www.dofactory.com/Patterns/PatternAbstract.aspx#UML) http://www.dofactory.com/Images/redarrow.gif [participants](http://www.dofactory.com/Patterns/PatternAbstract.aspx#participants) | http://www.dofactory.com/Images/redarrow.gif [sample code in C#](http://www.dofactory.com/Patterns/PatternAbstract.aspx#csharp) |

|  |  |  |  |
| --- | --- | --- | --- |
| **definition**   |  |  | | --- | --- | | http://www.dofactory.com/Images/pixel.gif | Provide an interface for creating families of related or dependent objects without specifying their concrete classes.   Frequency of use: http://www.dofactory.com/Images/use_high.gif  high | | http://www.dofactory.com/Images/552092_4.jpg |

[http://www.dofactory.com/Images/up.gifreturn to top](http://www.dofactory.com/Patterns/PatternAbstract.aspx)

**UML class diagram**



[http://www.dofactory.com/Images/up.gifreturn to top](http://www.dofactory.com/Patterns/PatternAbstract.aspx)

**participants**

    The classes and/or objects participating in this pattern are:

* **AbstractFactory**  **(ContinentFactory)**
  + declares an interface for operations that create abstract products
* **ConcreteFactory**   **(AfricaFactory, AmericaFactory)**
  + implements the operations to create concrete product objects
* **AbstractProduct**   **(Herbivore, Carnivore)**
  + declares an interface for a type of product object
* **Product**  **(Wildebeest, Lion, Bison, Wolf)**
  + defines a product object to be created by the corresponding concrete factory
  + implements the AbstractProduct interface
* **Client**  **(AnimalWorld)**
  + uses interfaces declared by AbstractFactory and AbstractProduct classes

[http://www.dofactory.com/Images/up.gifreturn to top](http://www.dofactory.com/Patterns/PatternAbstract.aspx)

**sample code in C#**

This structural code demonstrates the Abstract Factory pattern creating parallel hierarchies of objects. Object creation has been abstracted and there is no need for hard-coded class names in the client code.

[**Show code**](http://www.dofactory.com/Patterns/PatternAbstract.aspx#_self1)

|  |
| --- |
| // Abstract Factory pattern -- Structural example |

This real-world code demonstrates the creation of different animal worlds for a computer game using different factories. Although the animals created by the Continent factories are different, the interactions among the animals remain the same.

[**Show code**](http://www.dofactory.com/Patterns/PatternAbstract.aspx#_self2)

|  |
| --- |
| // Abstract Factory pattern -- Real World example |

This .NET optimized code demonstrates the same real-world situation as above but uses modern, built-in .NET features, such as, generics, reflection, object initializers, automatic properties, etc.

[**Show code**](http://www.dofactory.com/Patterns/PatternAbstract.aspx#_self1)

|  |
| --- |
| // Abstract Factory pattern -- .NET optimized |