**Pattern Name:**

Abstract Factory Pattern

**Short Description:**

Create instances of classes belonging to different families

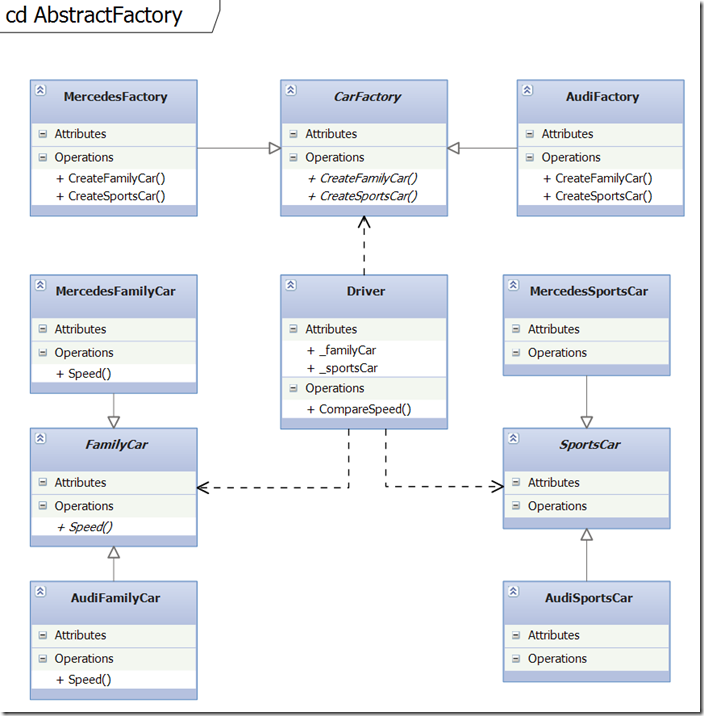
**Usage:**

Very frequently used and very useful

**Complexity:**

1 / 5

**UML Class Diagram:**

[](http://download.codeplex.com/download?ProjectName=csharpdesignpatterns&DownloadId=236763)

**Explanation:**

* The abstract factory class defines the abstract methods that have to be implemented by concrete factory classes. It serves as interface and contract definition.
* The concrete factory classes contain the real implementation that define which classes are created during run-time.
* Note that the methods return values are also defined by abstract classes, this allows a high flexibility and independence, leading to methods that must only be implemented once.
* The returned classes are however specific to each concrete factory class (you will see their implementation below).

    public abstract class CarFactory  
    {  
        public abstract SportsCar CreateSportsCar();  
        public abstract FamilyCar CreateFamilyCar();  
    }  
   
    public class AudiFactory : CarFactory  
    {  
        public override SportsCar CreateSportsCar()  
        {  
            return new AudiSportsCar();  
        }  
   
        public override FamilyCar CreateFamilyCar()  
        {  
            return new AudiFamilyCar();  
        }  
    }  
   
    public class MercedesFactory : CarFactory  
    {  
        public override SportsCar CreateSportsCar()  
        {  
            return new MercedesSportsCar();  
        }  
   
        public override FamilyCar CreateFamilyCar()  
        {  
            return new MercedesFamilyCar();  
        }  
    }

* Here you see the abstract classes that are used during the creation process (a method was added that serves to prove the validity of the design).
* Based on the abstract classes some real example implementation are created, those will be instantiated during run-time, depending on the concrete factory that creates them.

    public abstract class SportsCar  
    {  
    }  
   
    public abstract class FamilyCar  
    {  
        public abstract void Speed(SportsCar abstractFamilyCar);  
    }  
   
    class MercedesSportsCar : SportsCar  
    {  
    }  
      
    class MercedesFamilyCar : FamilyCar  
    {  
        public override void Speed(SportsCar abstractSportsCar)  
        {  
            Console.WriteLine(GetType().Name + " is slower than "  
                + abstractSportsCar.GetType().Name);  
        }  
    }  
   
    class AudiSportsCar : SportsCar  
    {  
    }  
   
    class AudiFamilyCar : FamilyCar  
    {  
        public override void Speed(SportsCar abstractSportsCar)  
        {  
            Console.WriteLine(GetType().Name + " is slower than "  
                + abstractSportsCar.GetType().Name);  
        }  
    }

* When implementing the associations between the Driver class, the abstract factory class and the abstract classes you may either use the common language agnostic approach using only private members (which is the most memory efficient one).

    public class Driver  
    {  
        private SportsCar \_sportsCar;  
        private FamilyCar \_familyCar;  
   
        public Driver(CarFactory carFactory)  
        {  
            \_sportsCar = carFactory.CreateSportsCar();  
            \_familyCar = carFactory.CreateFamilyCar();  
        }  
   
        public void CompareSpeed()  
        {  
            \_familyCar.Speed(\_sportsCar);  
        }  
    }

* Or you may use the C# language specific approach where everything is wrapped using private properties. This allows adding logic when accessing or changing the private members but might be a little overkill (this is also what gets auto-generated when modeling using the class diagram).

    public class Driver  
    {  
        private CarFactory \_carFactory;  
        private SportsCar \_sportsCar;  
        private FamilyCar \_familyCar;  
   
        public Driver(CarFactory carFactory)  
        {  
            CarFactory = carFactory;  
            SportsCar = CarFactory.CreateSportsCar();  
            FamilyCar = CarFactory.CreateFamilyCar();  
        }  
   
        private CarFactory CarFactory  
        {  
            get { return \_carFactory; }  
            set { \_carFactory = value; }   
        }  
   
        private SportsCar SportsCar  
        {  
            get { return \_sportsCar; }  
            set { \_sportsCar = value; }   
        }  
   
        private FamilyCar FamilyCar  
        {  
            get { return \_familyCar; }  
            set { \_familyCar = value; }   
        }  
   
        public void CompareSpeed()  
        {  
            FamilyCar.Speed(SportsCar);  
        }  
    }

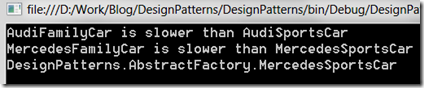
* You may also use another C# language specific solution that uses generic classes to create objects and that is also a valid implementation for the abstract factory pattern.

    public class GenericFactory<T>   
        where T : new()  
    {  
        public T CreateObject()  
        {  
            return new T();  
        }  
    }

* In the last step we add some code to test the software design and the Abstract Factory implementation.

    public static void AbstractFactory()  
    {  
        // Language agnostic version  
        CarFactory audiFactory = new AudiFactory();  
        Driver driver1 = new Driver(audiFactory);  
        driver1.CompareSpeed(); ;  
   
        CarFactory mercedesFactory = new MercedesFactory();  
        Driver driver2 = new Driver(mercedesFactory);  
        driver2.CompareSpeed();  
   
        // C# specific version using generics  
        var factory = new GenericFactory<MercedesSportsCar>();  
        var mercedesSportsCar = factory.CreateObject();  
        Console.WriteLine(mercedesSportsCar.GetType().ToString());  
   
        Console.ReadKey();  
    }

* When running the example you can see that everything is working as expected and that the correct classes are instantiated during runtime.

[](http://download.codeplex.com/download?ProjectName=csharpdesignpatterns&DownloadId=236740)