1. **面向对象（object orientend）特性**： **封装** 就是使用访问修饰符隐藏字段 属性 方法，方法的实现；**继承** 类的继承，子类可以拿到父类的字段 属性 方法；**多态** 子类可以替换父类，并可以重写父类的方法，是设计模式的基础
2. **面向对象思考方式**：继承（inherited）；组合（composition 强调生命周期，一个依赖于另一个，强关联的如人和心脏）/ 聚合（aggregations: one object is a part of another，弱关联如人和学生）；接口和实现；抽象
3. **设计原则**：1-5统称为SOLID设计原则
4. **单一职责**：拆分功能，责任分离
5. **开闭原则**：关闭修改，开放扩展
6. **里氏替换原则**：子类可以替换父类对象，但是父类对象不能替换子类对象
7. **接口分离原则**：应该依赖需要的接口，不依赖用不到的
8. **依赖倒置原则**：依赖于抽象，而不是依赖于具体，依赖注入模式
9. **迪米特法则**：减少直接耦合，横向纵向耦合，多采用中介的方式，如facade和mediator模式
10. **合成（组合/聚合）复用原则**：尽量使用组合而不是继承，使用原有的对象作为字段来复用已有的功能

**软件设计原则：**

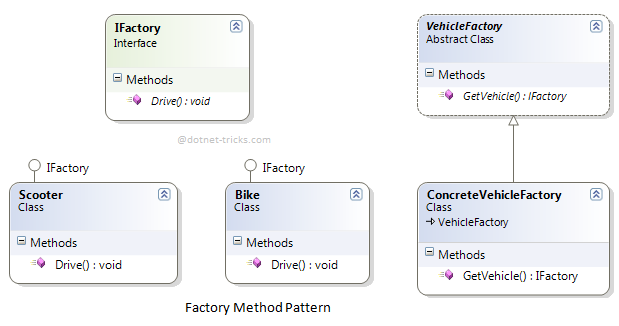
1. SOLID（1-5的简写）
2. DRY（don't repeat yourself）
3. KISS（keep it simple ，stupid）
4. YAGNI（you ain't gonna need it）
5. **设计模式**：

**优点**：1、可以提供软件设计中常见问题的解决方案2、可以在多种平台多种语言中使用3、提供特殊场景的标准术语

1. **创建型 用来创建对象**
2. 工厂方法（factory method）：创建派生类的对象，只是针对一个基类派生出的对象，将物体创建的过程，隐藏到具体工厂内，根据类型动态创建需要的对象

* 工厂接口
* 具体工厂类
* 产品接口
* 具体产品类

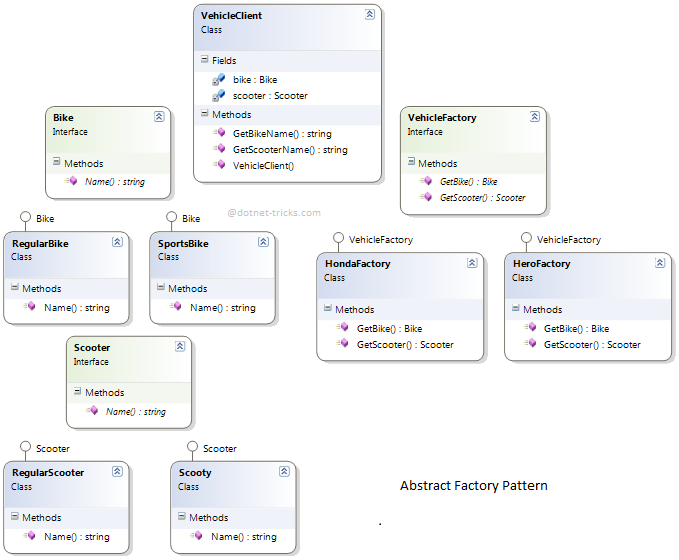
示例图：



1. 抽象工厂（abstract factory）：也叫做工厂的工厂，创建一系列类的对象，其实就是多个工厂方法集中到一个工厂类中，对工厂的创建同时采用工厂方法

* 抽象工厂接口（包含具体产品的工厂方法）
* 具体工厂
* 抽象物品接口
* 具体物品
* 客户端（使用抽象工厂和抽象产品接口来创建一系列相关物品）

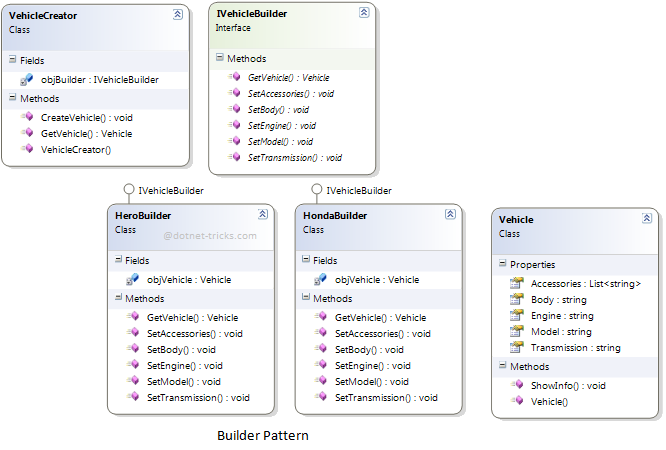
示例图：



1. 生成器（builder）：将物体按照其表现进行拆分成更多物体，按照一定流程来生成一个复杂的对象

* Builder接口，定义了生产具体产品所有步骤
* 具体builder接口
* 具体产品
* Director使用不同builder构建具体产品，可控制所有步骤的调用

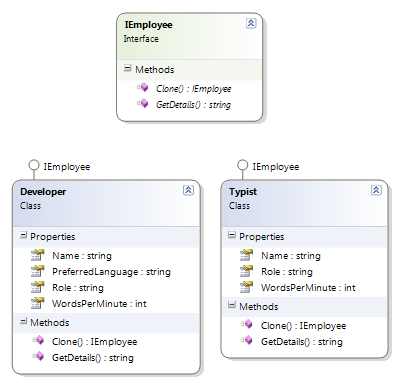
示例图：



1. 原型模式（Prototype）：由一个物体clone或者复制来创建更多对象，如果一个物体的创建过程消耗很大，就可以使用这个模式来大量的复制

* 原型接口 提供clone方法，浅拷贝MemberwiseClone，共享引用类型字段地址，深拷贝Clone全都是新的地址，对元数据引用类型的修改，不会影响拷贝的数据
* 具体的原型

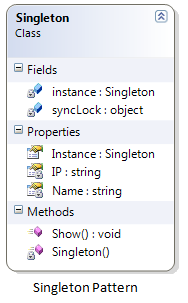
示例图：

·

1. 单例模式（single）：确保类只有一个实例

* Eager 初始化
* Lazy 初始化
* 线程安全方式
* 隐藏构造函数

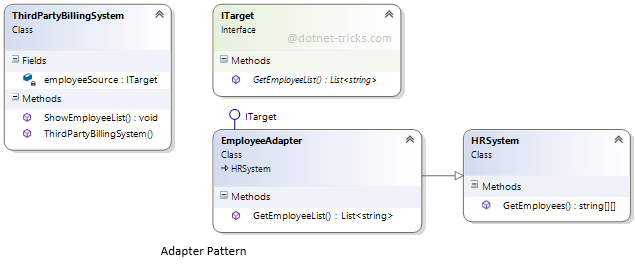
示例图：



1. **结构型 用来处理对象的组合关系**
2. 适配器（adapter）：将不同类的接口统一到一个接口

* ITarget client调用的接口
* Adapter 继承接口，来实现适配，持有被适配对象
* Adaptee 被适配者，提供的接口和客户端不适配
* Client 客户端和适配器交互

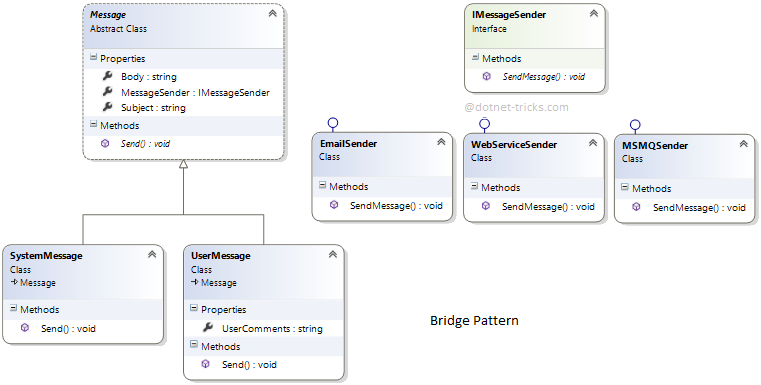
示例图：



1. 桥接（bridge）：将对象的抽象和其实现分离开，将多个维度的变化抽象出来，作为其字段进行组合，可以避免只对单一的类进行继承，只对变化点进行抽象实现

* 抽象类 包含多个可变的桥接接口
* 具体类 继承抽象
* 桥接接口
* 具体具体桥接实现、

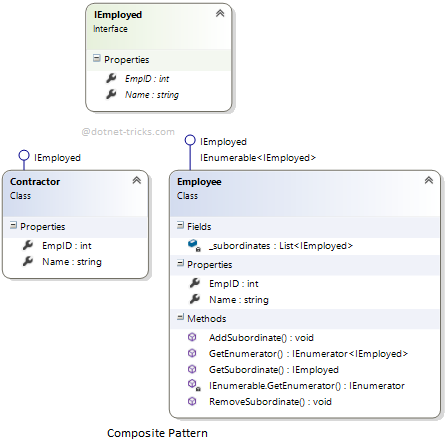
示例图：



1. 组合模式（composite）：单个或组合对象的树形结构，对单个对象的处理同样适用于一组对象

* 组件抽象类 包含统一的操作方法
* 组合类 中间节点，继承抽象，包含一组集合，提供add，remove，find，get等方法，可以理解为领导，下面有好多员工
* 单个节点类 叶子节点，继承抽象，可以理解为最底层员工，

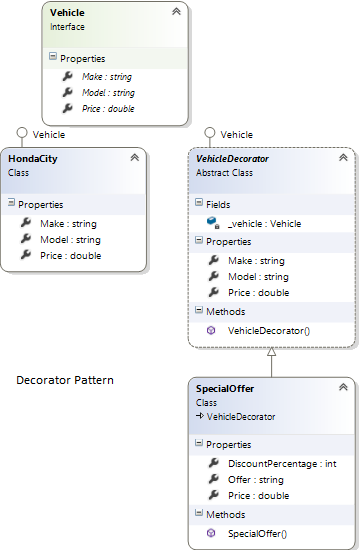
UML图：



1. 装饰模式（decorator）：动态给已经存在的对象添加功能，并且不改变其结构，通过继承的方式扩展属于静态，组合属于动态，创建一个装饰器类，包裹一个现有对象，并且为其添加新的功能

* 抽象类
* 具体类
* 装饰器类 继承抽象，持有抽象对象
* 具体装饰器

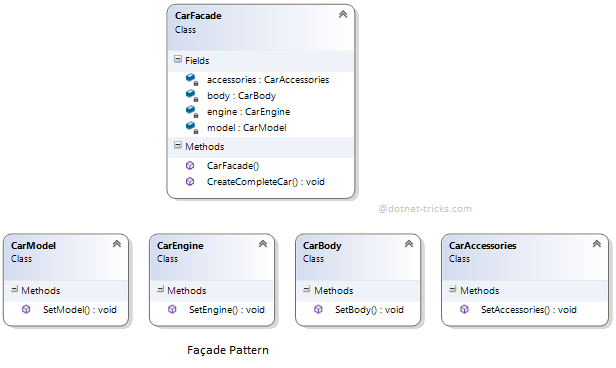
示例图：



1. 外观模式（facade）：使用一个单独类来代表一个复杂的系统，将外部接口和系统内部接口隔离

* 复杂系统
* 复杂系统的各个组成
* 外观类 持有复杂系统各个组件的对象
* 客户端

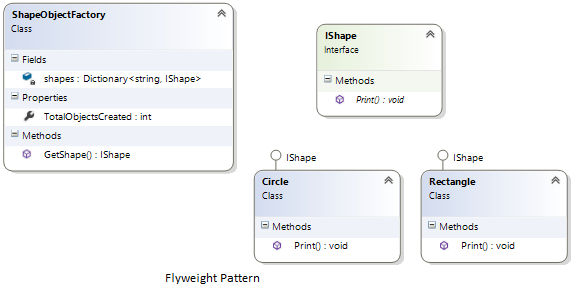
示例图：



1. 享元模式（flyweight）：多个物体内存共享，最小化内存占用，常用一种数据结构存储需要的对象，避免反复创建，对象池算是一种

* 享元工厂类 持有享元的对象列表
* 享元接口
* 具体的享元

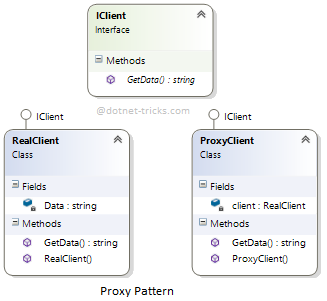
示例图：



1. 代理模式（proxy）：使用代理对象来处理原来对象不好处理的问题

* 对象接口
* 具体的对象类
* 代理类 继承接口，持有需要代理的具体对象

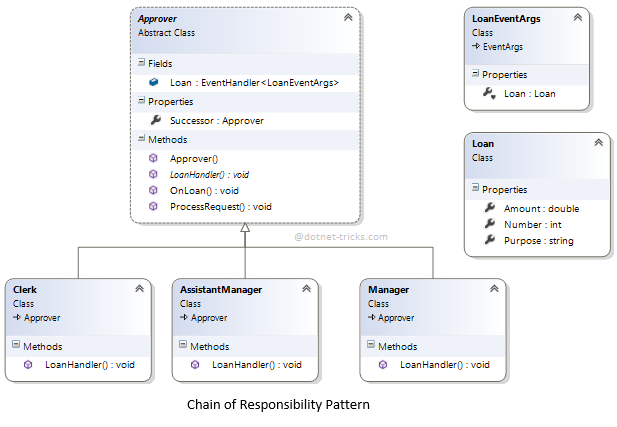
示例图：



1. **行为性 处理对象间的行为关系，如交流，管理，职责**
2. 责任链（chain of responsibility）:在一系列对象间传递请求 ，请求发送者不知道最后谁来处理，链式结构消息传递

* 处理类 抽象类，持有下个处理类的对象
* 具体处理类 实现抽象

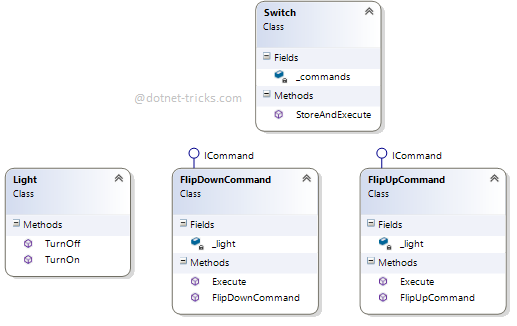
示例图：



1. 命令模式（command）：将请求包装成命令，传递给执行对象

* 客户端 创建和处理命令
* 请求器 请求命令执行操作，顺序存储执行的命令，方便redo和undo
* 命令接口
* 具体命令
* 接受器 持有命令执行对象

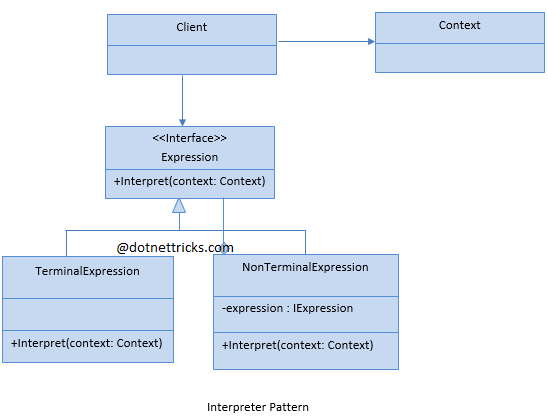
示例图：



1. 解释器（interpreter）：提供一个解释接口，来解释特定的内容

* 客户端 持有解释器和具体内容
* 解释器接口
* 具体解释器

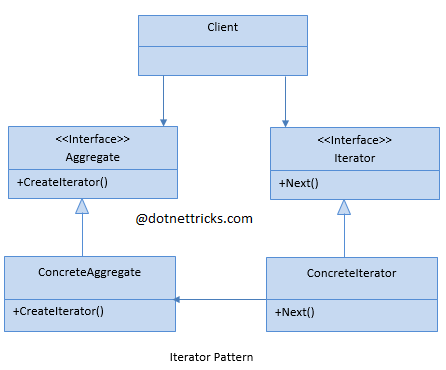
示例图：



1. 迭代器（iterator）：提供以顺序方式访问集合对象元素的方法，而不需要知道其底层结构

* 迭代器接口
* 具体的迭代器
* 集合接口
* 具体的集合

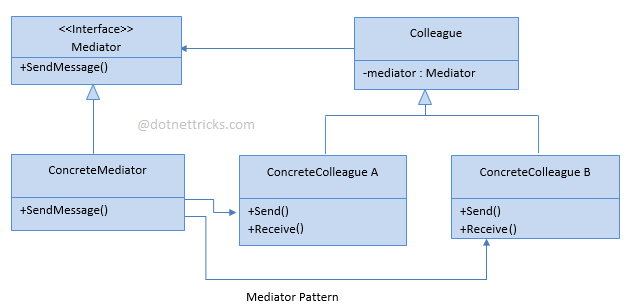
示例图：



1. 中介模式（mediator）：允许多个对象相互交流，并不需要知道对象的具体结构，将个对象的耦合关系放到一个类中处理，解耦，可以参考房产中介处理房东和租客

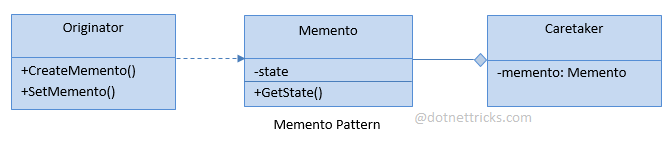
* 中介接口 持有需要相互交流的对象
* 具体中介实现接口
* 对象接口 持有中介对象
* 具体对象

示例图：



1. 备忘录（memento）：捕获对象的状态并存储，可以以后重建不改变封装结构

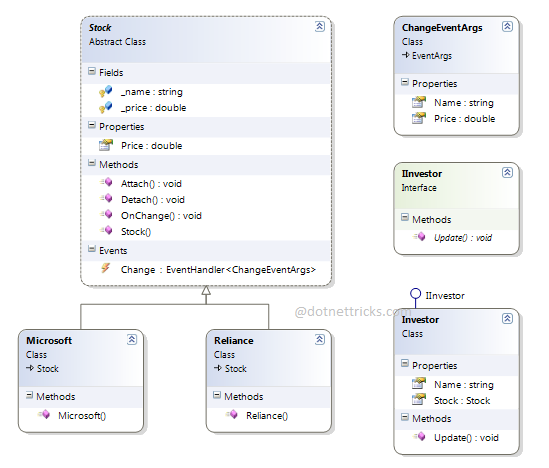
* Originator 持有memento的对象
* Memento 对state的封装
* Caretaker 持有memento

示例图：

1. 观察者（observer）：订阅者发布消息给所有的观察者

* 订阅者接口 持有所有的观察者
* 具体订阅者
* 观察者接口 持有订阅者对象
* 具体观察者

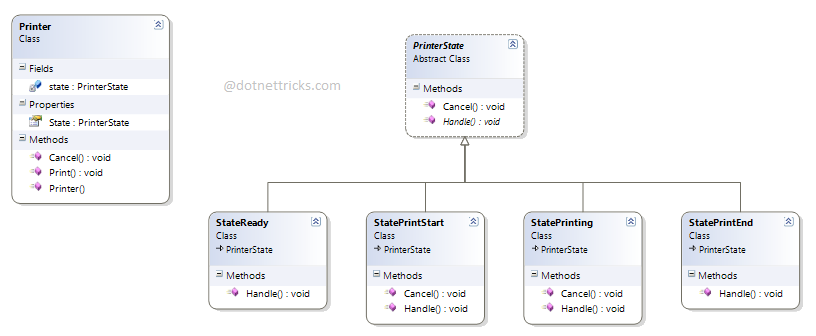
示例图：



1. 状态模式（state）：内部状态改变，来改变对象的行为

* Context 持有状态对象
* 状态接口处理context的表现
* 具体状态

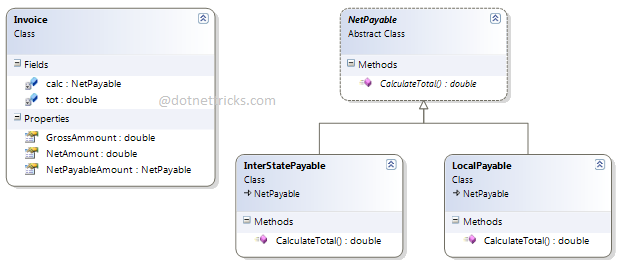
示例图：



1. 策略模式（strategy）：在运行时，从一系列算法中，动态选取特定的算法

* Context 持有策略的对象
* 策略接口 处理具体的算法
* 具体的策略

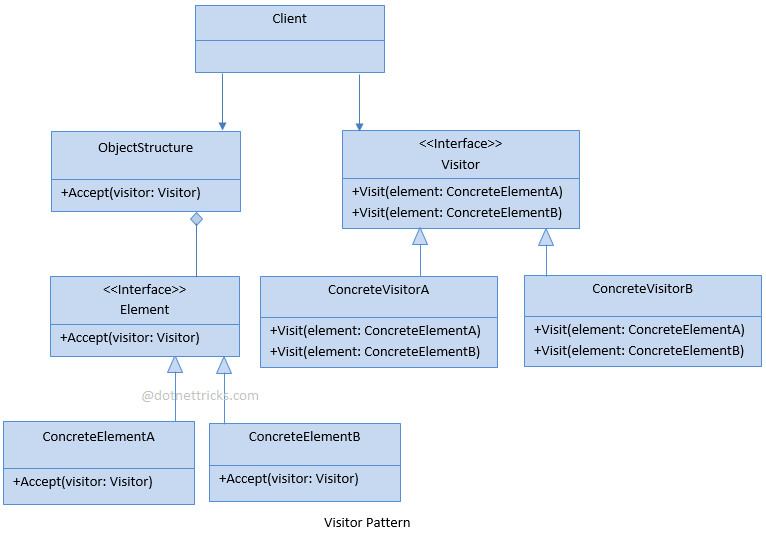
示例图：



1. 访问器（visitor）：在不改变对象结构或类的情况下，对一组对象创建并执行新的操作

* Client 持有所有的访问器，和objectstructure
* ObjectStructure 持有所有的可以被访问器使用的对象
* Element 对象抽象类
* 具体的element
* Visitor 访问器接口
* 具体的访问器

示例图：



1. 模板方法（template method）:定义一个具体算法的各个基本步骤，并允许修改单个步骤的实现，也就是说步骤的实现可以重载或者重写，和策略模式类似，但是不是直接替换算法

* 抽象算法 有许多抽象步骤
* 具体的算法 步骤进行重写

示例图：

