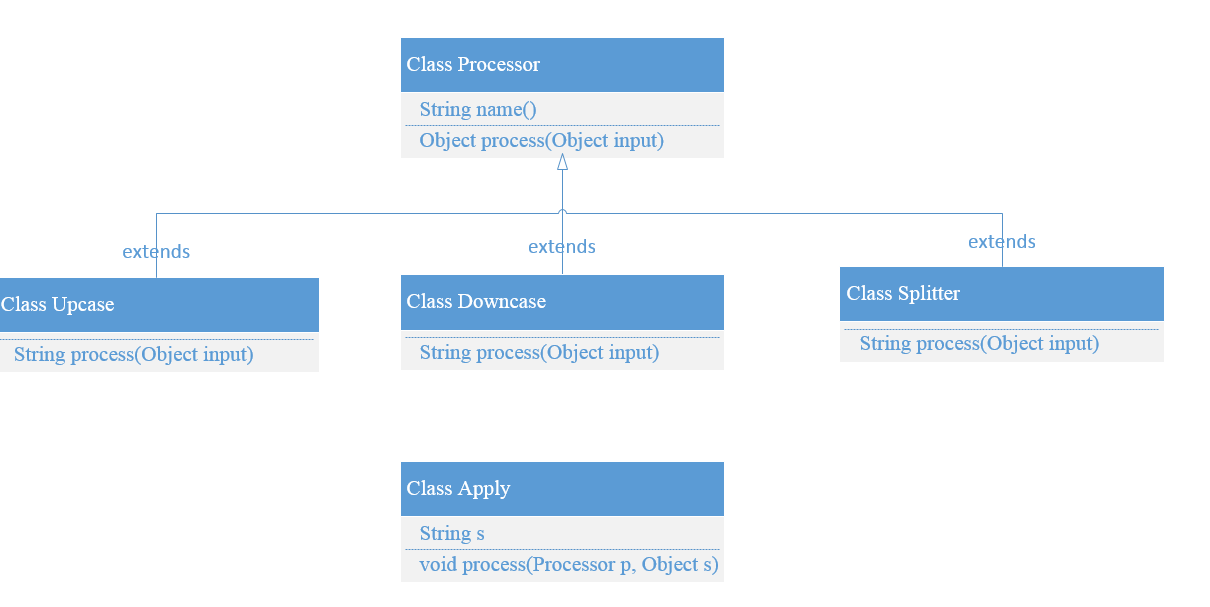
几种设计模式：

1. 策略设计模式

何时用到: 创建一个能够根据所传递的参数对象的不同而具有不同行为的方法，被称为策略设计模式。



代码example:

Apply{

Process(Process p, Object s){…}

}

process(new Upcase(), s);

process(new Downcase(), s);

process(new Splitter(), s);

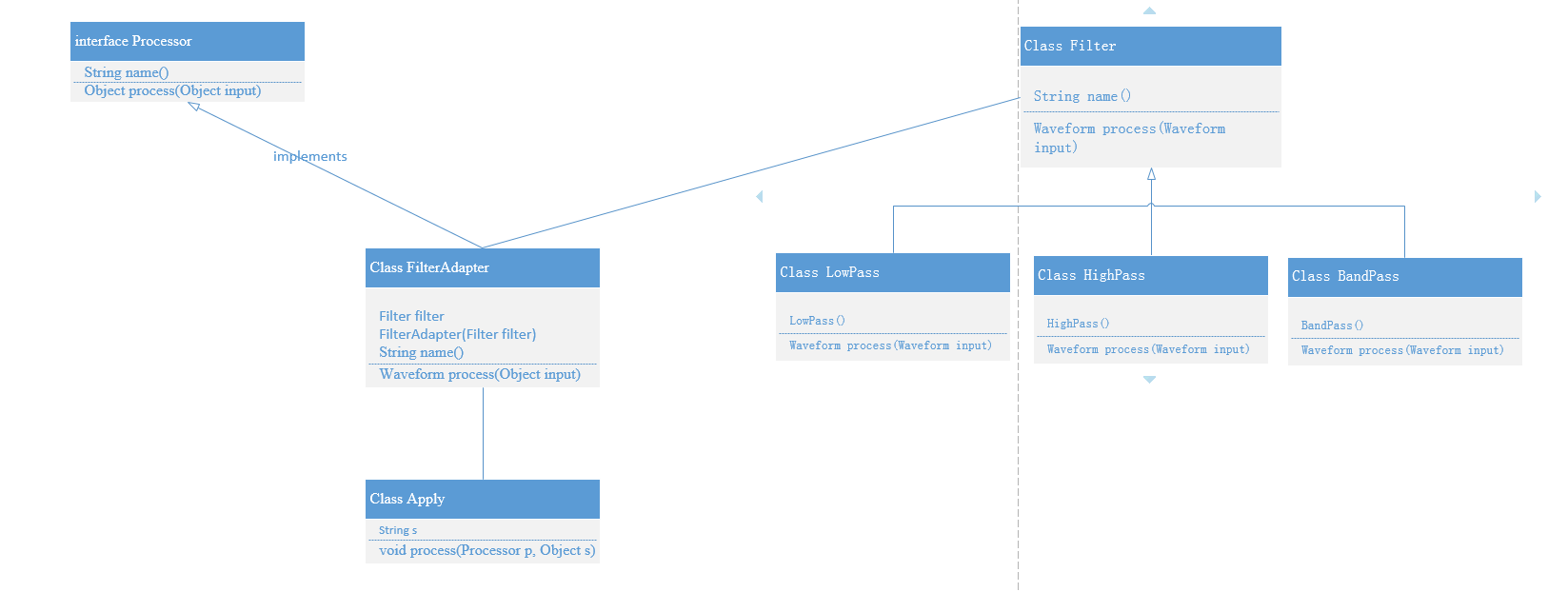
2. 适配器设计模式

何时用到；解决类之间耦合过紧的问题。碰到无法修改客户端想要使用的类，如类库是被发现而非被创建的。适配器中的代码将接受你所拥有的接口，并产生你所需要的接口。

适配器模式是将一个东西，适配到另一个东西上，但是我不想动这两个东西，加一个转换器来做。比如我现在有个插头，一个插座，但是插头插不进去插座，因为接口不一致，我又不想换插头和插座，所以就是需要一个适配器或者说转换器就行了。适配器模式就是这种用途，我已经实现的不希望修改，通过添加新的类适用到别的环境中。

优点概括而言：更好的复用性以及更好的扩展性。

将Processor设为interface，耦合限制就会变得松动，让Filter复用Apply.process



代码example:

FilterAdapter(适配器) implements Processor(适配之二){

Filter filter; (适配之一)

Public FilterAdapter(Filter filter){

this.filter = filter;

}

Public Waveform process(Object input) (实现Process接口的方法){

…}

}

Apply.process(new FilterAdapter(new LowPass()), w); 这回Filter能被Apply用了，解决了耦合过紧的问题。

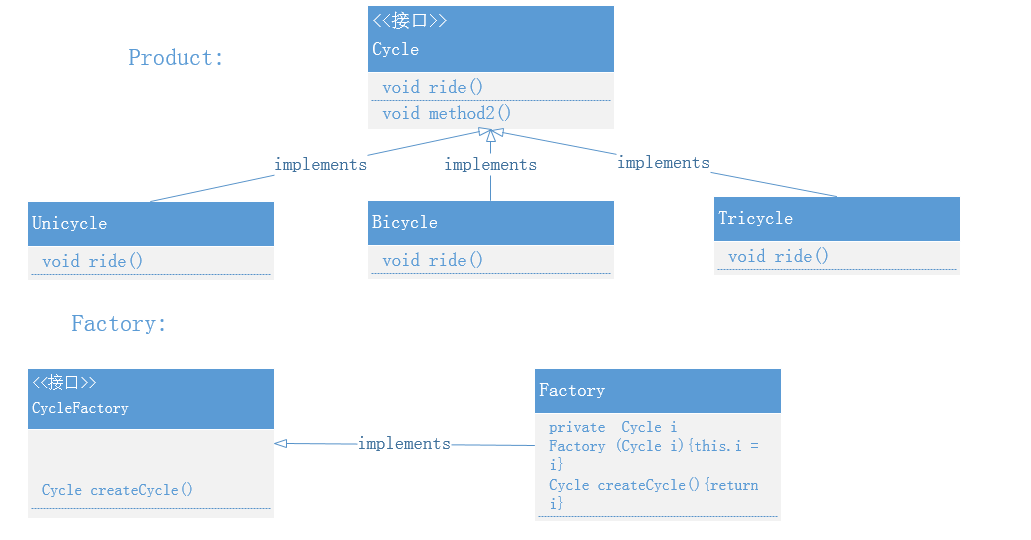
3. 缺省适配器设计模式

何时用到：多重继承，单继承+接口的形式替代多继承的功能。

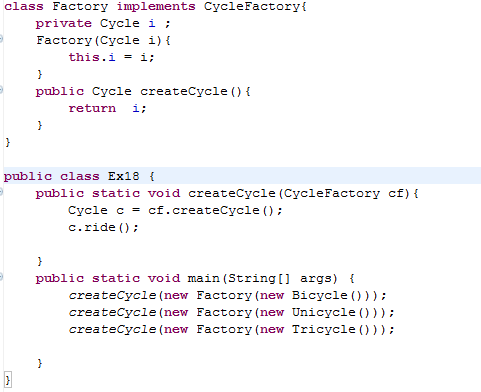
4. 工厂设计模式

何时用到；生成遵循某个接口的对象工作由工厂类来做。这么做完全是为了方便简洁，为生成对象提供一个统一的出口，不用说我想用到的时候随意创建，而是统一通过工厂来做。

好处是：通过这种方式，我们的代码将完全与接口的实现分离，这就使我们可以透明地将某个实现，替换成另一个实现。



代码example:



辅助理解：http://www.cnblogs.com/java-my-life/archive/2012/04/13/2442795.html