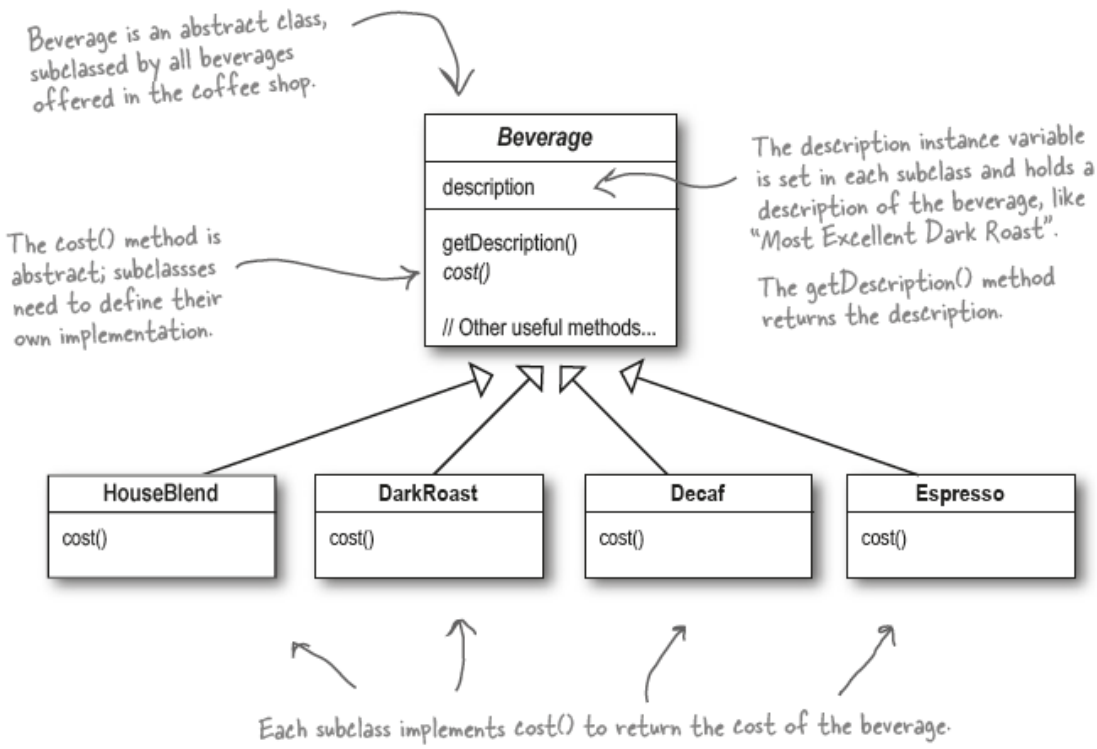
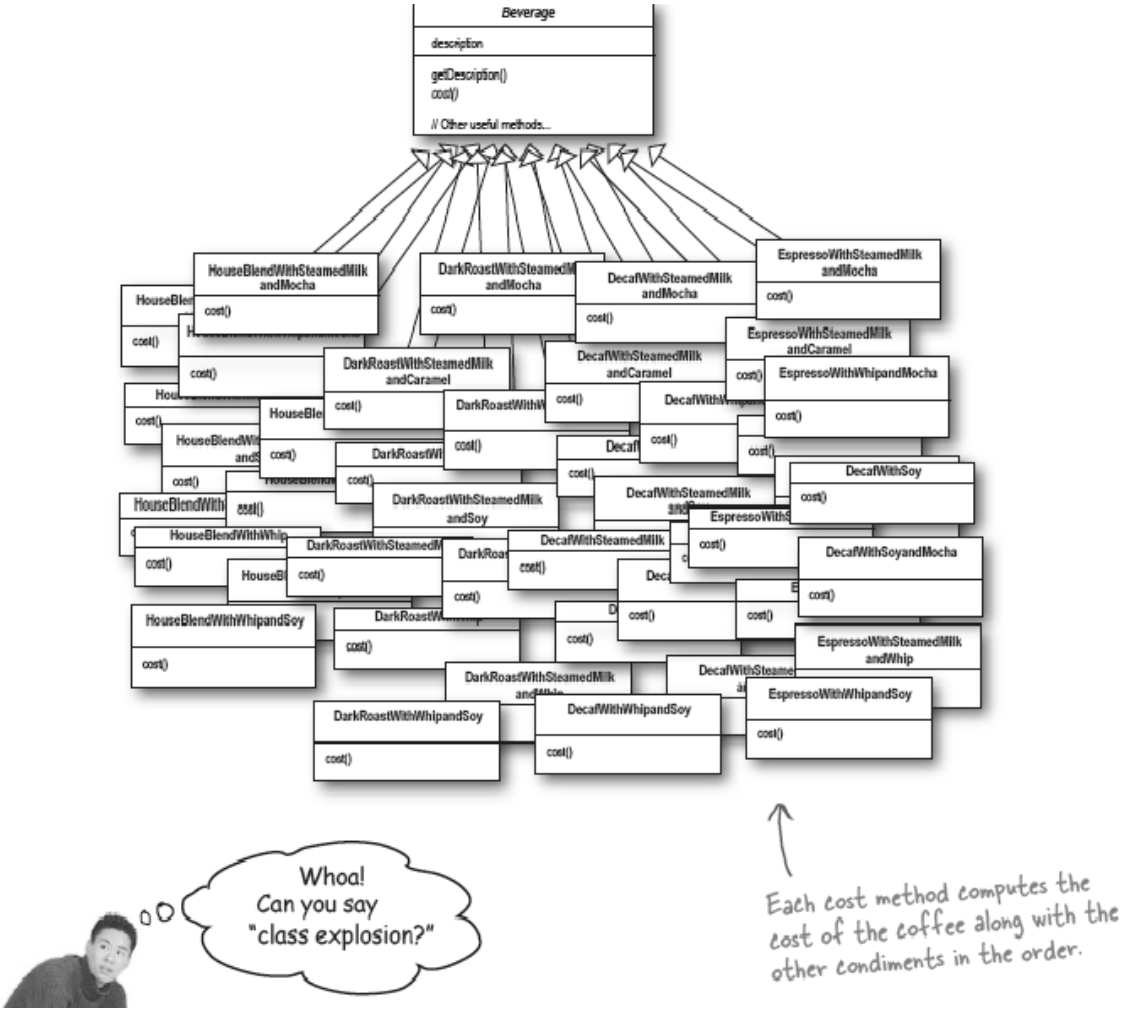
عنوان: **معرفی Decorator Design Pattern**
نویسنده: حامد صمدی
تاریخ: ۱۳۹۲/۰۱/۲۳ ۸:۳۰
آدرس: www.dotnettips.info
برچسب‌ها: Design patterns

فرض کنید که می‌خواهیم یک برنامه برای یک فروشگاه نوشیدنی (مانند coffee shop) بنویسیم ، این فروشگاه در ابتدای کار ممکن است ، منوی ساده‌ای جهت ارائه به مشتری داشته باشد. برای مثال ممکن است که فقط 3 یا 4 محصول داشته باشد.
بنابراین ممکن است ما برنامه‌ای را که می‌خواهیم برای این مشتری بنویسیم به صورت زیر طراحی کنیم:

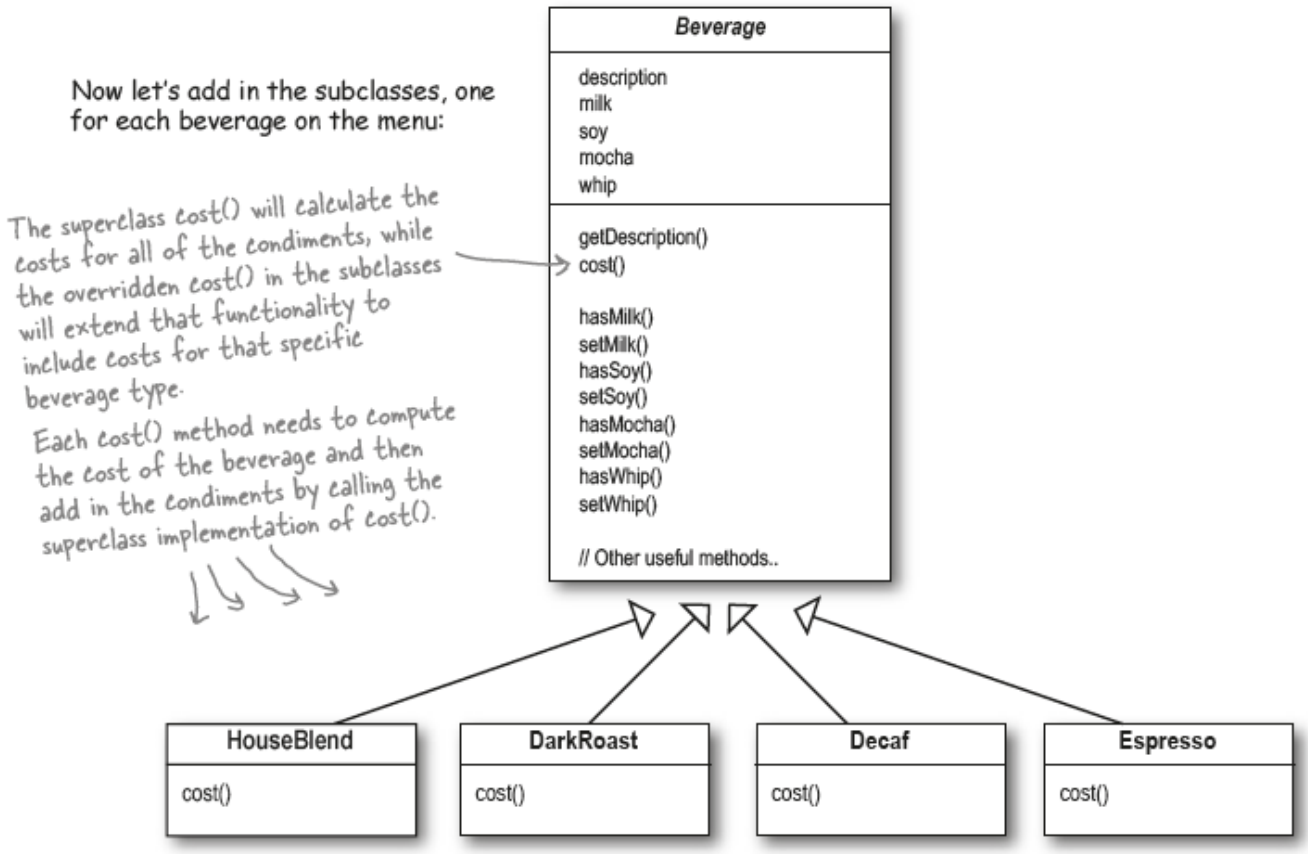


که بسیار طبیعی و درست می‌باشد. اما حالا در نظر بگیرید که این فروشگاه در آینده ممکن است محصولات خود را افزایش بدهد و یا حالاتی که ممکن است این محصولات با هم ادغام شوند را در نظر بگیرید. برای مثال ممکن است شما بخواهید که قهوه‌تان را با شیر نوش جان کنید و یا ....

بنابراین تعداد این حالات را در نظر بگیرید که در آینده ممکن است چقدر زیاد بشود:



خوب پس چه‌کاری ما می‌توانیم برای نگهداری این برنامه انجام بدهیم؟ یکی از راه‌هایی که ممکن است به فکر ما برسد این است که روش بالا روش احمقانه ای است. چرا ما باید به همه‌ی این کلاس‌ها نیاز داشته باشیم. ما می‌توانیم که چاشنی‌ها را در کلاس اصلی نگه‌داری کنیم و کلاس محصولاتمان را از کلاس اصلی به ارث ببریم اجازه دهید تا این کار را با هم انجام بدهیم



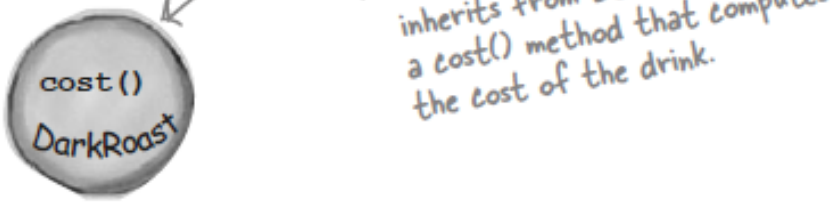
خوب با این روش ما n کلاس تشکیل شده در رویکرد اول را فقط به 5 کلاس تبدیل کردیم. خوب این روشی بسیار ایده‌ال به نظر می‌رسد. اما ممکن است در آینده که تعداد چاشنی‌های ما بالا می‌رود و همچنین تعداد محصولاتمان نیز ممکن است بیشتر شود مجبور شویم که تعداد این کلاس‌ها را بیشتر کنیم، و یا فکر کنید که ما می‌خواهیم هریک از چاشنی‌هایمان، یک قیمت را نسبت بدهیم. بنابراین مجبوریم که تمامی این‌ها را در کلاس پایه اضافه کنیم؛ بله درست است، ما با کلاس پایه‌ی حجیمی روبرو می‌شویم که بیشتر خواص و یا متدهای آن برای زیر کلاس‌های دیگر مناسبت نیستند. خوب آیا روش بهتری برای جلوگیری از این مشکل داریم؟ بلی.

خوب ما به این مسئله به این صورت نگاه می‌کنیم که شروع می‌کنیم با نوشیدنی‌ها و آن‌ها را با چاشنی‌ها در زمان اجرا تزیین (Decorate) می‌کنیم؛ نه کامپایل.

برای مثال اگر مشتری ما یک نوشیدنی DarkRoast با Mocha و Whip خواست، سپس ما :
-1 یک شی از DarkRoast ایجاد می‌کنیم .
-2 آن را با یک شی از Mocha تزئین می‌کنیم.
-3 آن را با یک شی از Whip - تزیین می‌کنیم.
-4 متد Cost() را صدا می‌زنیم و یک Delegation را برای اضافه کردن قیمت چاشنی‌ها در نظر می‌گیریم.

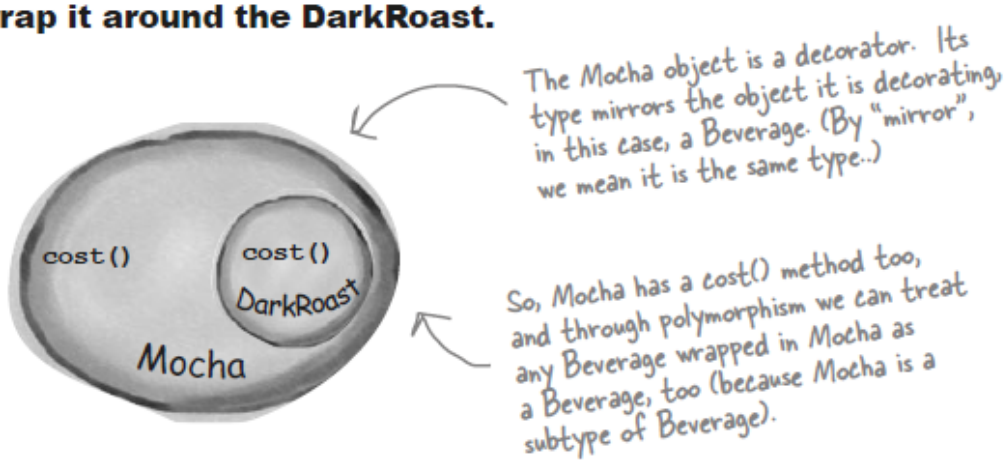
بسیار خوب؛ اما ما عملیات تزئین یک شی را چگونه انجام می‌دهیم و delegation ما چگونه عمل می‌کند .
یک اشاره : به شیء تزئین کننده، مانند یک Wrappers فکر کنید. اجاز بدهید ببینم که چه طور این کار را انجام می‌دهیم.
-1 یک شی از DarkRoast ایجاد می‌کنیم.

1 We start with our DarkRoast object.

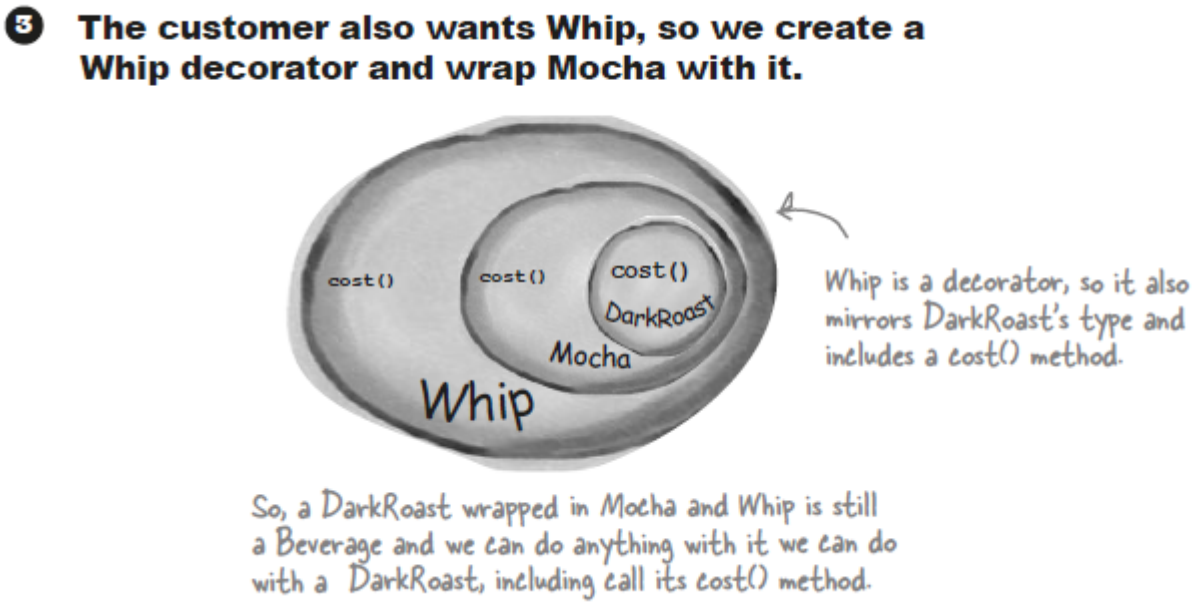


-2 آن را با یک شی از Mocha تزئین می‌کنیم.

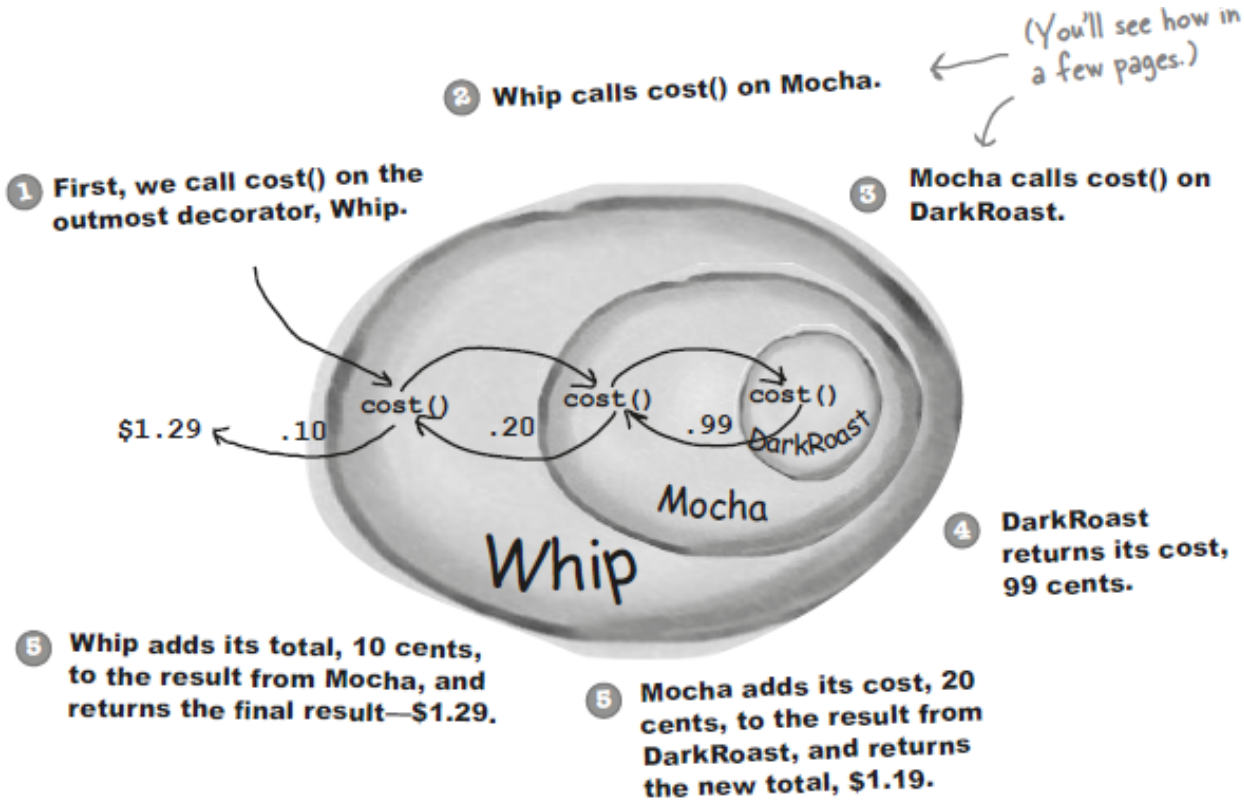
2 The customer wants Mocha, so we create a Mocha object and wrap it around the DarkRoast.

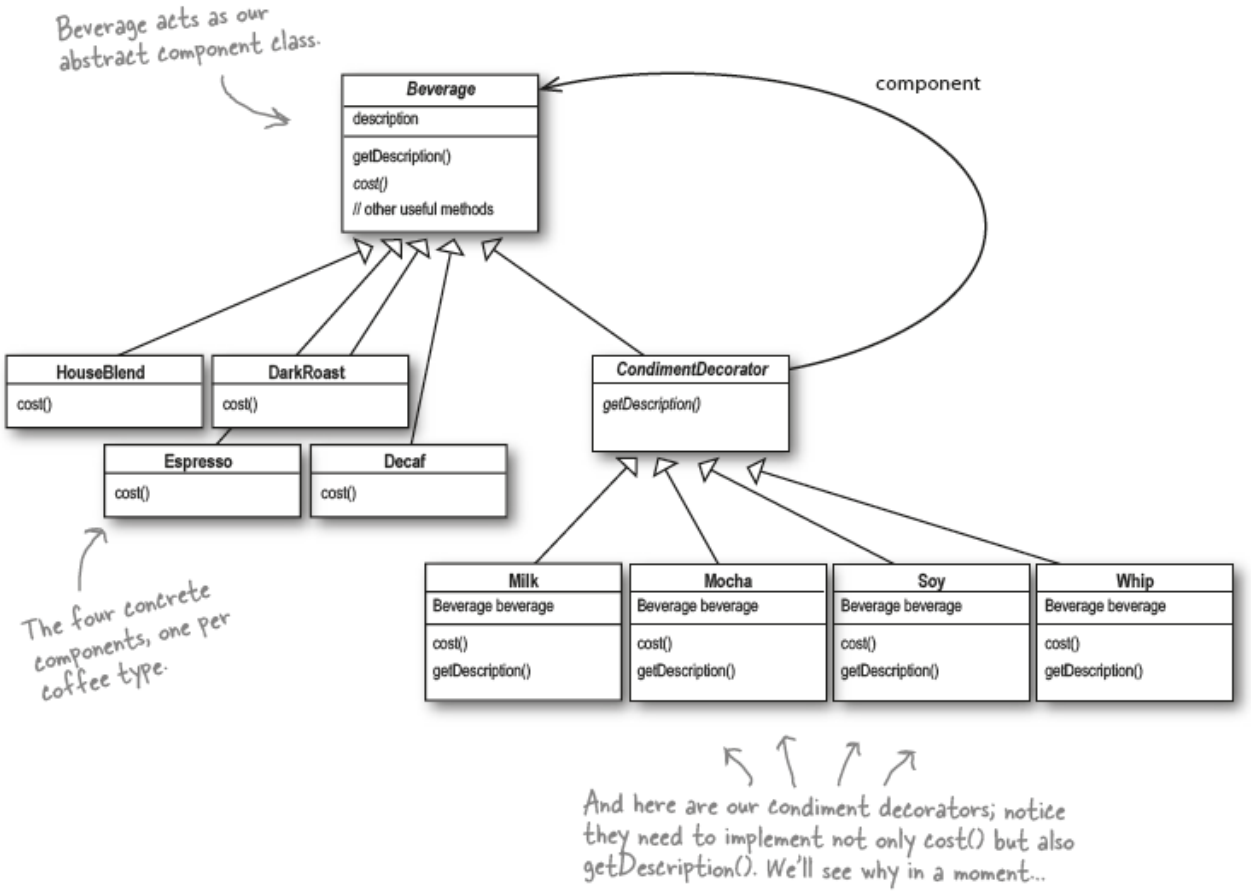


-3 آن را با یک شی از Whip تزیین می‌کنیم



4- حالا زمان محاسبه قیمت محصول برای مشتری فرا رسیده است. ما این کار را را با صدا زدن بیرونی‌ترین Decorator(Whip) انجام می‌دهیم و شی whip به کمک Delegate مابقی توابع cost را صدا می‌زند.





در آخر شما می‌توانید پیاده سازی این برنامه را به زبان جاوا در زیر مشاهده نمایید.

```
public abstract class Beverage
{
     string description ="unknow beverage";
     public String getDescription(){
         return description;
     }
     public abstract double cost();
}

public abstract class CondimentDecorator extends Beverage {
    public abstract string getDescription();
}

public class Espersso extends Beverage{
   public Espersso()
   {
      description="Espersso";
   }
   public double cost(){
     return 1.99;
   }
}

public class HouseBelend extends Beverage {
   public HouseBelend()
   {
     description="HouseBelend";
   }
   public double cost()
   {
     return .89;
   }
}
```

```
public class mocha extends condimateDecorator {
  Beverage beverage;
  public mocha(Beverage beverage)
  {
     this.beverage=beverage;
  }
   public string getDescription(){
      return beverage.getdescription() + "Mocha";
   }
   public double cost(){
     return .20 +beverage.cost
   }
}


// Now Use These classes in Final form
Beverage beverage=new Espersso();
//Customers want a coffe with double milk and whip
beverage=new mocha(beverage);
beverage=new mocha(meverage);
beverage=new whip(beverage);

system.out.println(beverage.getDescription() + "$" +beverage.cost());
```

نظرات خوانندگان

نویسنده: سید ایوب کوکبی
تاریخ: ۱۳۹۲/۰۱/۲۳ ۹:۵۳

به نظرم ترجمه بخشی از کتاب Head First Design Pattern باشه. کتاب خوبیه.

نویسنده: حامد صمدی
تاریخ: ۱۳۹۲/۰۱/۲۳ ۱۲:۳۹

بله آقای کوکبی ترجمه ای از کتاب Head First Design Pattern است

نویسنده: توحید عزیزی
تاریخ: ۱۳۹۲/۰۱/۲۴ ۱:۱۲

ضمن تشکر از مقاله بسیار مفید شما، در مثال آخر مقاله، نوشته شده که مشتری اسپرسو را با دو شات «شیر» و [یک] شات «ویپ» می‌خواهد، اما کد نوشته شده، ۲ شات «موکا» اضافه کرده است.
از دقت خودم در قضایای شکمی، شرمنده ام :)

```
//Customers want a coffe with double milk and whip
beverage=new mocha(beverage);
beverage=new mocha(meverage);
```