به نام خدا

Composite patterns

استاد: دکتر فرهاد مردوخی

ترجمه: پژمان محمدی

اصلاح و مرور: همایون ملکی

درس: مهندسی نرم افزار پیشرفته

رشته: مهندسی کامپیوتر، گرایش نرم افزار

دانشگاه رازی

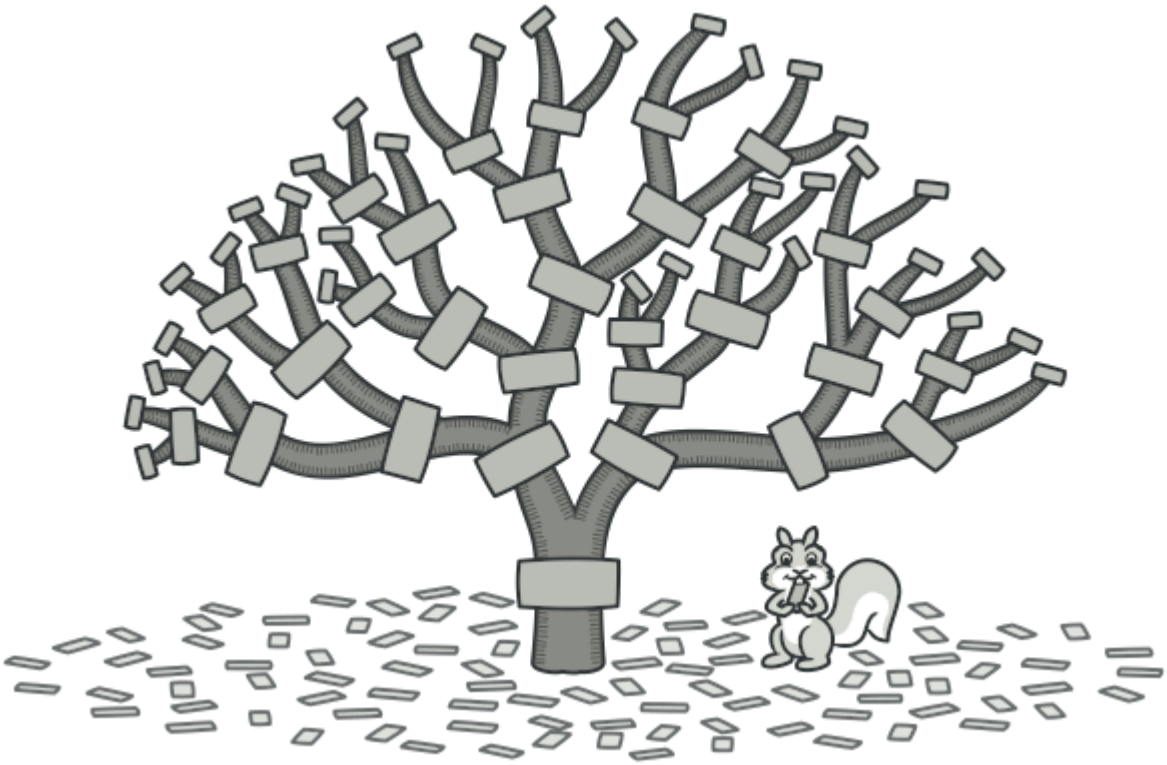
سال تحصیلی ۱۴۰۰-۱۳۹۹

Composite

همچنین به عنوان درخت شی شناخته می شود

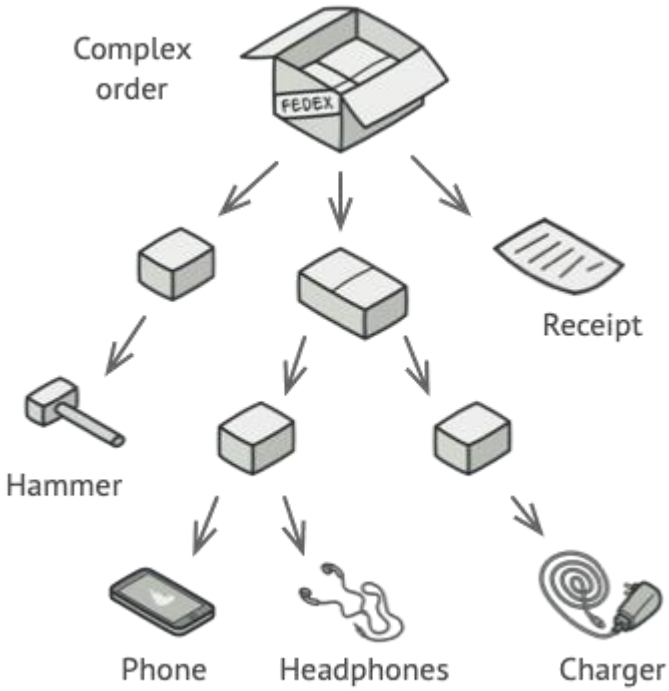
قصد

کامپوزیت یک الگوی طراحی ساختاری است که به شما امکان می دهد اشیا را در ساختارهای درختی ترکیب کنید و سپس با این ساختارها کار کنید گویی که اشیا جداگانه هستند.



مسئله

استفاده از الگوی کامپوزیت فقط زمانی منطقی است که مدل اصلی برنامه شما بتواند به عنوان یک درخت نشان داده شود.
به عنوان مثال ، تصور کنید که دو نوع شی دارید: محصولات و جعبه ها. یک جعبه می تواند شامل چندین محصول و همچنین تعدادی جعبه کوچکتر باشد. این جعبه های کوچک همچنین می توانند برخی از محصولات یا حتی جعبه های کوچکتر و غیره را در خود جای دهند.
بگویید تصمیم دارید یک سیستم سفارش ایجاد کنید که از این کلاسها استفاده کند. سفارشات می توانند حاوی محصولات ساده و بدون بسته بندی و همچنین جعبه های پر از محصولات ... و جعبه های دیگر باشند. قیمت کل چنین سفارشی را چگونه تعیین می کنید؟

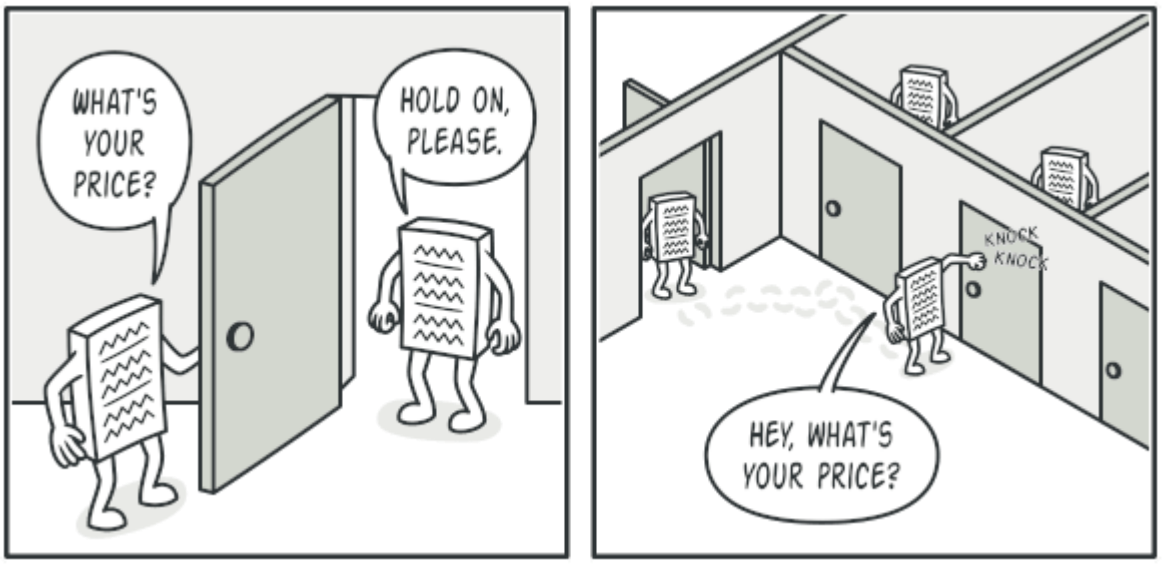


یک سفارش ممکن است شامل محصولات مختلف ، بسته بندی شده در جعبه ها باشد که در جعبه های بزرگتر و غیره بسته بندی می شوند. کل ساختار مانند یک درخت وارونه به نظر می رسد.
می توانید روش مستقیم را امتحان کنید: همه جعبه ها را باز کنید ، تمام محصولات را مرور کنید و سپس کل را محاسبه کنید. این در دنیای واقعی قابل انجام است. اما در یک برنامه ، به سادگی اجرای یک حلقه نیست. پیش از این باید از کلاسهای محصولات و جعبه هایی که طی می کنید ، سطح لانه سازی جعبه ها و سایر جزئیات ناخوشایند مطلع شوید. همه اینها رویکرد مستقیم را بیش از حد ناجور یا حتی غیرممکن می کند.

راه حل

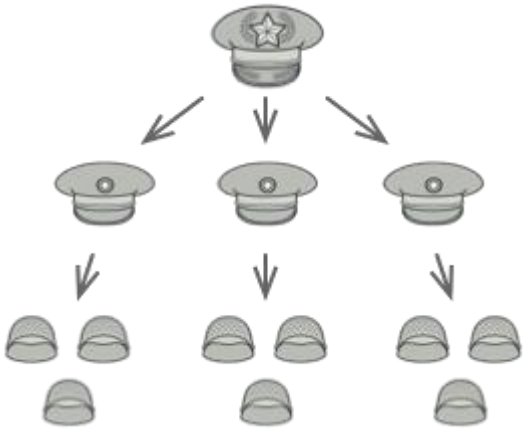
الگوی کامپوزیت پیشنهاد می کند که شما با محصولات و جعبه ها از طریق یک رابط مشترک کار کنید و روشی را برای محاسبه قیمت کل بدست بیاریم.
این روش چگونه کار می کند؟ برای یک محصول ، به راحتی قیمت محصول را بدست آورده. برای یک جعبه ، یا بیشتر، از جعبه استفاده می کند ، قیمت جعبه ها را بپرسید و سپس مبلغی را برای جعبه های استفاده شده بازگرداند. اگر یکی از این موارد شامل جعبه کوچکتری بود ، آن جعبه نیز شروع به مرور محتوای درون خود کرده ، تا زمانی که قیمت تمام اجزای داخلی محاسبه شود. یک جعبه حتی می تواند هزینه اضافی مانند هزینه بسته بندی را به قیمت نهایی اضافه کند.

قیاس در دنیای واقعی



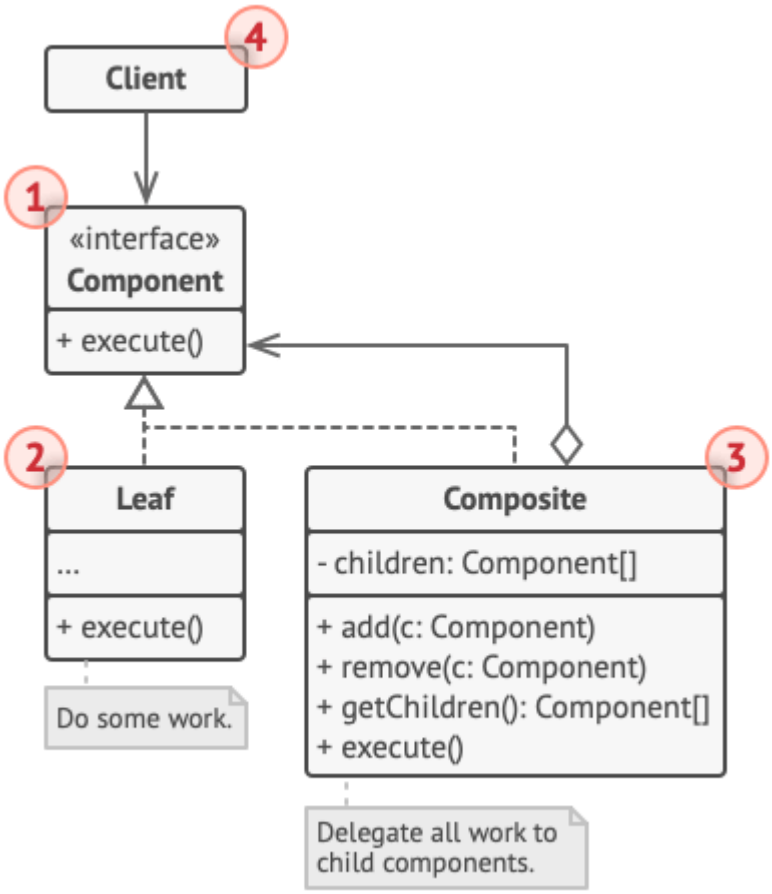
الگوی کامپوزیت به شما امکان می دهد رفتاری را به صورت بازگشتی روی تمام اجزای یک درخت اجرا کنید.
بزرگترین مزیت این روش این است که شما نیازی به مراقبت از کلاسهای واقعی اشیا که درخت را تشکیل می دهند نیست.
نیازی نیست که بدانید یک شی یک محصول ساده است یا یک جعبه پیچیده. از طریق رابط مشترک می توانید با همه آنها یکسان رفتار کنید. هنگامی که شما یک روش را فرا می خوانید ، اشیا خود، درخواست را به پایین درخت منتقل می کنند.

نمونه ای از یک ساختار نظامی



ارتش اکثر کشورها به عنوان سلسله مراتب ساختار یافته است. ارتش متشکل از چند لشکر است. یک لشکر مجموعه ای از تیپ ها است و یک تیپ متشکل از جوخه ها هستند. سرانجام ، یک جوخه گروه کوچکی از سربازان واقعی است. دستورات در بالای سلسله مراتب داده می شوند و به هر سطح منتقل می شوند تا زمانی که هر سربازی بداند چه کاری باید انجام شود.

ساختار



۱- Component رابط کامپوننت عملیاتی را توصیف می کند که برای عناصر ساده و پیچیده درخت مشترک هستند.

۲- Leaf برگ یک عنصر اساسی درخت است که عناصر فرعی ندارد.
معمولاً اجزای برگ بیشتر کارهای واقعی را انجام می دهند ، زیرا آنها کسی را ندارند که کار را به آنها واگذار کنند.

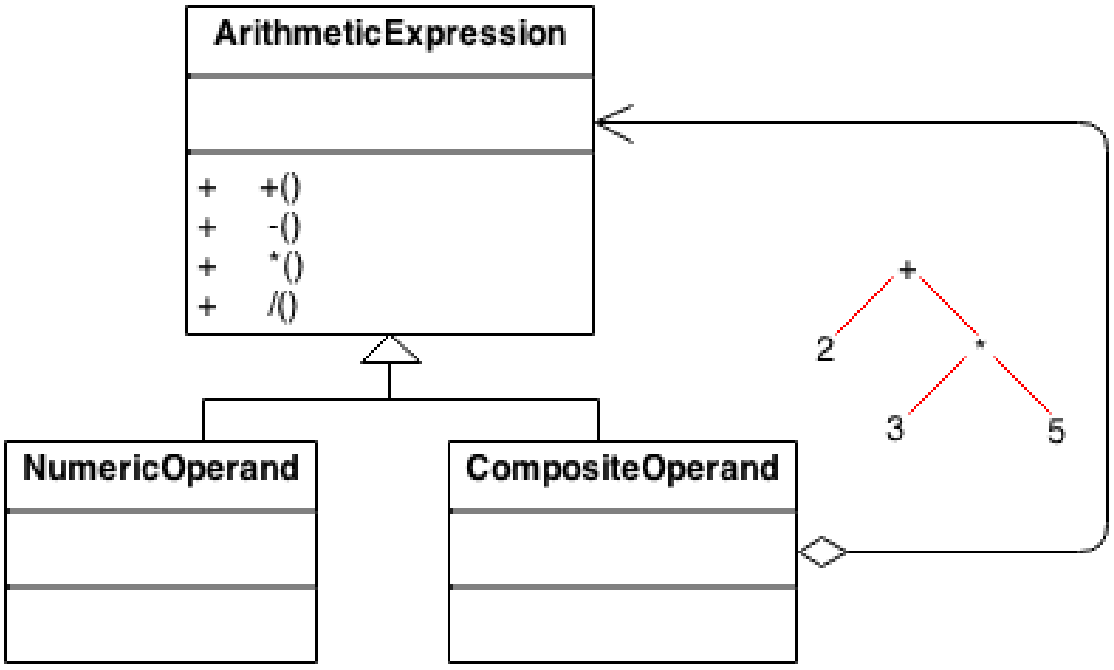
۳- Container (aka *composite*) ظرف (با نام مستعار کامپوزیت) عنصری است که دارای عناصر فرعی است: برگها یا ظروف دیگر. یک کانتینر کلاسهای واقعی فرزندانش را نمی داند. فقط رابط مولفه با تمام عناصر فرعی کار می کند.
با دریافت یک درخواست ، یک کانتینر کار را به عناصر فرعی خود واگذارمی کند ، نتایج متوسط را پردازش می کند و نتیجه نهایی را به مشتری برمی گرداند.

۴- Client مشتری از طریق رابط مولفه با تمام عناصر کار می کند. در نتیجه ، مشتری می تواند با هر دو عنصر ساده یا پیچیده درخت به همان شیوه کار کند.

مثال دیگر

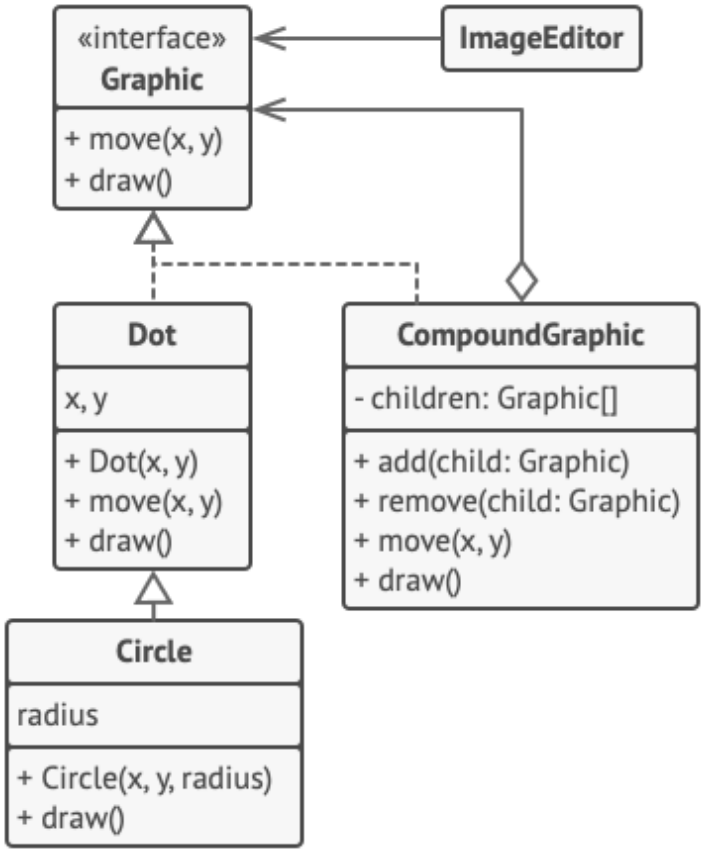
اگرچه مثال انتزاعی است ، اما عبارات حسابی مرکب هستند .یک عبارت حسابی از یک عملوند ، یک عملگر (+ - * /) و یک عملوند دیگر تشکیل شده است .عملوند می تواند یک عدد باشد یا یک عبارت حسابی دیگر .بنابراین

۲ + ۳ و (۲ + ۳) + (۴ * ۶) هر دو عبارت معتبری هستند.



شبه کد

در این مثال ، الگوی کامپوزیت به شما اجازه می‌دهد تا دسته‌ای از اشکال هندسی را در یک ویرایشگر گرافیکی اجرا کنید



مثال ویرایشگر اشکال هندسی

کلاس CompoundGraphic یک ظرفی است که می‌تواند شامل هر تعداد از اشکال فرعی دیگرو شکل‌های مرکب باشد .
یک شکل مرکب از همان روش‌ها یک شکل ساده استفاده می‌کند . با این حال ، یک شکل مرکب به جای انجام کاری به تنهایی ، درخواست را به صورت بازگشتی به همه فرزندانش منتقل می کند و نتیجه را "جمع بندی" می کند.کد مشتری با تمام اشکال از طریق یک رابط واحد مشترک برای همه کلاس‌های شکل کار می‌کند . بنابراین ، مشتری نمی‌داند که آیا با یک شکل ساده کار می‌کند یا یک شکل ترکیبی . مشتری می‌تواند با ساختارهای بسیار پیچیده شی کار کند ، بدون همراهی با کلاس‌های واقعی که ساختار را شکل می‌دهد

واسط اجزا عملکرد مشترک هر دو را اعلام می‌کند
اشیا ساده و پیچیده یک ترکیب‌بندی

```
interface Graphic is
    method move(x, y)
    method draw()
```

کلاس برگ نماینده اهداف نهایی یک ترکیب است یک شی برگ هیچ چیز فرعی ندارد . معمولا , برگ است

اشیایی که کار واقعی را انجام می‌دهند در حالی که فقط اشیا مرکب انتقال به زیر اجزای آن‌هارا انجام میدهد

```
class Dot implements Graphic is
    field x, y

    constructor Dot(x, y) { ... }

    method move(x, y) is
        this.x += x, this.y += y

    method draw() is
```

یک نقطه در x و y رسم کنید . تمام کلاس‌های جز می‌توانند اجزای دیگر را گسترش دهند .

```
class Circle extends Dot is
    field radius

    constructor Circle(x, y, radius) { ... }

    method draw() is
```

یک دایره در x و y با شعاع R رسم کنید .

کلاس مرکب ، اجزای پیچیده را نشان می‌دهد که ممکن است بچه‌دار شوند . اشیا مرکب معمولاً واقعی را انتخاب می‌کنند برای فرزندان کار کنید و سپس نتیجه را جمع‌بندی کنید.

```
class CompoundGraphic implements Graphic is
    field children: array of Graphic
```

یک شی مرکب می‌تواند اجزای دیگر (اعم از ساده یا پیچیده) را اضافه یا حذف کند .

```
    method add(child: Graphic) is
```

یک فرزندرا به ردیف فرزندان اضافه کنید .

```
        method remove(child: Graphic) is
```

یک فرزند را از ردیف فرزندان دور کنید .

```
method move(x, y) is
foreach (child in children) do
   child.move(x, y)
```

یک کامپوزیت ، منطق اولیه خود را به طور خاص اجرا می‌کند . این روش از طریق تمام فرزندان خود ، جمع‌آوری و جمع‌بندی نتایج خود ، از صورت بازگشتی استفاده می‌کند . از آنجا که فرزندان composite's این تماس‌ها را به فرزندان خود منتقل می‌کنند ، کل درخت شی در نتیجه از بین می‌رود .

```
method draw() is
```

برای هر مولفه فرزند : - مولفه را رسم کنید . به‌هنگام‌سازی مستطیل تکی۲ . یک مستطیل با استفاده از مختصات boundingرسم کنید . کد مشتری با تمام مولفه‌ها از طریق رابط پایه آن‌ها کار می‌کند . به این ترتیب کد مشتری می‌تواند از مولفه‌های ساده برگ و نیز کامپوزیت های پیچیده پشتیبانی کند .

```
class ImageEditor is
   field all: array of Graphic

   method load() is
      all = new CompoundGraphic()
      all.add(new Dot(1, 2))
      all.add(new Circle(5, 3, 10))
```

اجزای انتخاب‌شده را به یک ترکیب مرکب پیچیده ترکیب کنید

```
      method groupSelected(components: array of Graphic) is
      group = new CompoundGraphic()
      foreach (component in components) do
         group.add(component)
         all.remove(component)
      all.add(group)
```

همه اجزا محدود خواهند شد

```
      all.draw()
```

قابل اجرا بودن

هنگامی که مجبورید یک ساختار شی مانند درخت را پیاده سازی کنید ، از الگوی Composite استفاده کنید.
الگوی Composite دو نوع عنصر اساسی را برای شما فراهم می کند که دارای یک رابط مشترک هستند: برگ های ساده و ظروف پیچیده. یک ظرف را می توان از هر دو برگ و سایر ظروف تشکیل داد. با این کار می توانید ساختار جسم بازگشتی تودرتو را که شبیه درخت است ، بسازید.
وقتی می خواهید کد مشتری با عناصر ساده و پیچیده به طور یکنواخت رفتار کند از این الگو استفاده کنید.

همه عناصر تعریف شده توسط الگوی کامپوزیت از یک رابط مشترک استفاده می کنند. با استفاده از این رابط ، مشتری نگران کلاس واقعی اشیایی نیست که با آنها کار می کند.

نحوه اجرا

۱. اطمینان حاصل کنید که مدل اصلی برنامه شما می تواند به عنوان یک ساختار درختی نشان داده شود. سعی کنید آن را به عناصر ساده تقسیم کنید. به یاد داشته باشید که ظروف باید بتوانند حاوی عناصر ساده و ظروف دیگر باشند.
۲. رابط مولفه را با لیستی از روشها که هم برای مولفه های ساده و هم برای مولفه های پیچیده منطقی است ، اعلام کنید.
۳. برای نمایش عناصر ساده کلاس برگ ایجاد کنید. یک برنامه ممکن است چندین کلاس برگ مختلف داشته باشد.
۴- یک کلاس کانتینر برای نمایش عناصر پیچیده ایجاد کنید. در این کلاس ، یک قسمت آرایه برای ذخیره منابع به عناصر فرعی ارائه دهید. این آرایه باید بتواند برگها و ظروف را ذخیره کند ، بنابراین مطمئن شوید که با نوع رابط مو لفه اعلام شده باشد.
هنگام اجرای روش های رابط مو لفه ، به یاد داشته باشید که یک کانتینر قرار است بیشتر کارها را به عناصر فرعی جدا کند.
۵- در آخر ، روشهای افزودن و حذف عناصر فرزند را در ظرف تعریف کنید.

به خاطر داشته باشید که این عملیات را می توان در رابط مو لفه اعلام کرد. این امر اصل تفکیک رابط را نقض می کند زیرا این متدها در کلاس برگ خالی خواهند بود. با این حال ، مشتری می تواند همه عناصر را به طور مساوی درمان کند ، حتی هنگام ساختن درخت.

مزایا و معایب :

شما می‌توانید با ساختارهای درختی پیچیده به راحتی کار کنید : پلی مورفیسم و بازگشت به نفع خود استفاده کنید .
اصل باز / بسته . شما می‌توانید انواع مختلف عنصر را بدون شکستن کد موجود ، که در حال حاضر با درخت شی کار می‌کند ، وارد برنامه کنید .
تهیه یک رابط مشترک برای کلاسهایی که کارایی آنها بیش از حد متفاوت است ممکن است دشوار باشد. در برخی از سناریوها ، باید بیش از حد کلی رابط مولفه را ایجاد کنید و درک آن دشوارتر شود.

روابط با الگوهای دیگر

شما می‌توانید از سازنده برای ایجاد درخت‌های مرکب پیچیده استفاده کنید زیرا می‌توانید مراحل ساخت آن را به صورت بازگشتی برنامه‌ریزی کنید .
زنجیره مسئولیت اغلب در ترکیب با کامپوزیت استفاده می‌شود . در این مورد , وقتی یک مولفه برگ درخواست می‌کند , ممکن است آن را از طریق زنجیره تمام اجزای والد پایین به ریشه درخت هدف منتقل کند .
می‌توانید از هرس کردن درختان با کامپوزیت استفاده کنید .
شما می‌توانید از بازدیدکنندگان برای اجرای یک عملیات روی کل درخت مرکب استفاده کنید .
شما می‌توانید گره‌های مشترک برگ درخت کامپوزیت را برای ذخیره برخی ram پیاده‌سازی کنید .

کامپوزیت و دیکوتر نمودارهای ساختاری مشابهی دارند , زیرا هر دو بر ترکیب بازگشتی تکیه دارند تا تعداد ثابتی از اشیا را سازماندهی کنند.
شبیه کامپوزیت است اما فقط یک مولفه فرزند دارد . یکی دیگر از این تفاوت‌ها عبارت است از اضافه کردن مسئولیت‌های اضافی به شی مورد نظر .
با این حال , الگوها می‌توانند با هم هم‌کاری کنند: شما می‌توانید از دیکتور برای گسترش رفتار یک شی خاص در درخت مرکب استفاده کنید .
طراحی‌ها برای استفاده سنگین از کامپوزیت استفاده می‌شوند . استفاده از این الگو به شما اجازه می‌دهد تا ساختارهای پیچیده را به جای ساخت دوباره آن‌ها از ابتدا کلون کنید .