16Organizing classes into inheritance hierarchies

Birkaç sınıfın ortak özellikleri, associations veya işlemleri varsa, bu ortak yönleri içeren ayrı bir üst sınıf oluşturarak çoğaltmayı önlemek en iyisidir. Tersine, eğer karmaşık bir sınıfınız varsa, onun işlevselliğini birkaç özel subclasses bölmek iyi olabilir. 17Örneğin, çeşitli hesap türlerinin bulunduğu bir bankacılık uygulaması oluşturduğunuzu düşünün. Bakiyenin ve sahibinin bulunması, hesaba para yatırılabilmesi, açılıp kapatılabilmesi gibi bazı şeyler tüm hesaplarda ortaktır. Diğer şeyler hesapları farklılaştırır; örneğin, bir ipotek hesabının negatif bakiyesinin yanı sıra teminat olarak bir mülk (örneğin bir ev) vardır; bir tasarruf hesabı, içinde yüksek bir bakiye tutmak için daha yüksek faiz gibi belirli ayrıcalıklara sahip olabilir. Bu örnekte Account sınıfının SavingsAccount, CheckingAccount ve MortgageAccount alt sınıflarının üst sınıfı olması gerektiğini söyleyebiliriz.

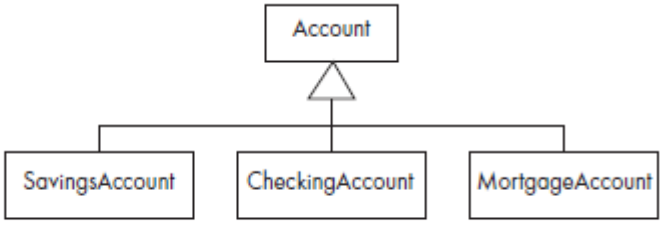

Figure 2.4 Basic inheritance hierarchy of bank accounts

Bir subclass ile doğrudan superclass arasındaki ilişkiye generalization denir. Alt sınıfa specialization adı verilir. Bir veya daha fazla generalization içeren hiyerarşiye kalıtım hiyerarşisi, generalization hiyerarşisi veya isa hiyerarşisi denir. Kalıtım hiyerarşilerini Şekil 2.4'te gösterildiği gibi grafiksel olarak çizebilirsiniz. Küçük üçgen aynı üst sınıfı paylaşan bir veya daha fazla genellemeyi simgeliyor ve superclass işaret ediyor. Bu tür diyagramların superclass üstte ve subclasses aşağıda olacak şekilde çizilmesi en açık şekilde görülür; ancak başka düzenlemelere de izin verilir.

Tanım: Kalıtım, bir üst sınıfta tanımlanan özelliklerin bir alt sınıfı tarafından örtülü olarak sahip olunmasıdır. Özellikler değişkenleri ve yöntemleri içerir.

Üst sınıf superclasses

Alt sınıf subclasses

Nitelikler attributes

Miras hiyerarşisini oluşturarak inheritancei kontrol edersiniz. Hangi sınıfların üst sınıf, hangi sınıfların alt sınıf olduğunu tanımladıktan sonra miras otomatik olarak gerçekleşir. Örneğin Hesap'ın tüm özellikleri SavingsAccount, CheckingAccount ve MortgageAccount'ta da mevcuttur.

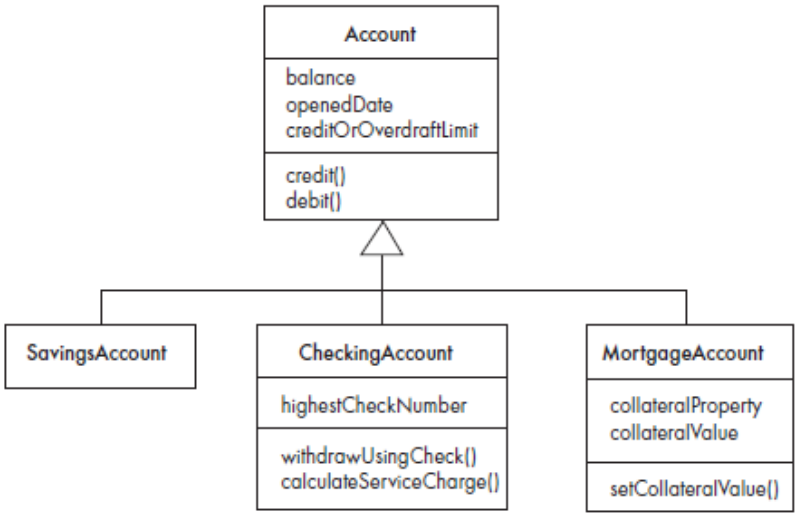

Figure 2.5 Inheritance hierarchy of bank accounts showing some attributes and operations

Şekil 2.5, Şekil 2.4'ü genişleterek Account'un sahip olduğu ve aynı zamanda üç alt sınıf tarafından da miras alınan çeşitli nitelikleri ve işlemleri göstermektedir. Nitelikler sınıf kutusunun ortasında gösterilir; işlemler altta gösterilmektedir. Diyagramın daha anlaşılır olması için devralınan özellikler alt sınıflarda açıkça gösterilmemiştir; ancak her alt sınıfa özel yeni özellikler gösterilir. Sınıfları miras hiyerarşileri halinde düzenlemek, nesne yönelimli tasarım ve programlamada önemli bir beceridir. Hata yapmak ve geçersiz generalizations yaratmak kolaydır.

18Uyulması gereken en önemli kurallardan biri isa kuraldır. İsa kuralı, A sınıfının, yalnızca İngilizce'de 'A, B'dir' demek anlamlıysa, B sınıfının geçerli bir alt sınıfı olabileceğini söylüyor. Örneğin 'Bir Tasarruf Hesabı bir Hesaptır' demek mantıklıdır; tersini söylemek, 'Hesap Tasarruf Hesabıdır' demek mantıklı değil. Tüm üst sınıf-alt sınıf çiftlerini (genellemeleri) isa kuralına göre test etmelisiniz. Bu nedenle miras hiyerarşilerine sıklıkla isa hiyerarşileri adı verilir. İsa kuralının ihlal edildiğini tespit ettiğinizde bu, geçersiz bir genelleme yaptığınızın açık bir göstergesidir. Ancak isa kuralının geçerli olduğu tüm durumlar iyi genellemeler değildir. Kontrol etmeniz gereken diğer önemli noktalar şunlardır:

1.Alt sınıfa veya üst sınıfa belirsiz isimler verdiyseniz (daha önce açıklandığı gibi 'Otobüs' gibi), genellikle kötü genellemeler yaratacaksınız.

2. Bir alt sınıf, ayırt edici özelliğini yaşamı boyunca korumalıdır. Örneğin, OverdrawnAccount alt sınıfını oluşturmaya karar verdiyseniz, isa kuralının geçerli olduğu görünür: ' OverdrawnAccount isa, account. Ancak, fazla para çekilen bir hesap, yeterli miktarda para yatırıldıktan sonra ayrı bir hesap türü olarak kalmayacaktır. Dolayısıyla bu iyi bir genelleme değil; aslında OverdrawnAccount sınıfı ayrı bir sınıf olmamalıdır.

3. Miras alınan özelliklerin tümü her alt sınıfta anlamlı olmalıdır. Şekil 2.5'te, üç alt sınıfın her birinin bir balance, bir openDate'e ve bir CreditOrOverdraftLimit'e sahip olduğundan emin olmalısınız. Ayrıca her bir alt sınıfta alacak ve borç işlemlerini gerçekleştirmenin mantıklı olduğundan ve bu işlemlere ilişkin tüm yöntemlerin tutarlı bir şekilde davranacağından da emin olmalısınız.

Tasarımcıların bu üç kontrolü gözden kaçırması yaygın bir hatadır. Kontrollerin gözden kaçırılması durumunda ortaya çıkan kodun istenmeyen kalıtımla başa çıkmak için birçok özel koşula ihtiyacı olur ve anlaşılması zorlaşır. Yukarıdakilerden çıkarabileceğimiz temel sonuçlar şunlardır: genellemeler ve bunların sonucunda ortaya çıkan kalıtım, tekrarların önlenmesine ve yeniden kullanımın

iyileştirilmesine yardımcı olur; ancak kötü tasarlanmış genellemeler aslında çözdüklerinden daha fazla soruna neden olabilir.

19Daire, Nokta, Dikdörtgen, Matris, Elips, Doğru, Düzlem.

Şekil 2.6 olası bir çözümü göstermektedir; bu tür bir sorunun genellikle birden fazla kabul edilebilir yanıtı olabilir.

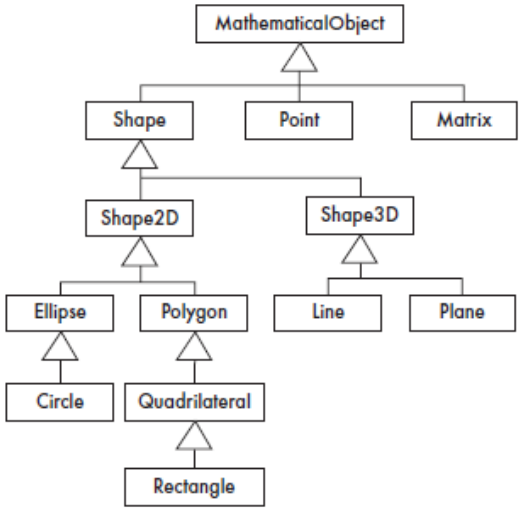

Figure 2.6 A possible inheritance hierarchy of mathematical objects

Bir Point yozlaşmış bir Şekil olduğunu düşünebilirsiniz. Ama kaç boyutu var? Bir noktanın herhangi bir sayıda boyutu olabilir. Belki de ihtiyaç duyulan şey ayrı Point2D ve Point3D sınıflarına sahip olmaktır.

Bir Line de benzer şekilde 2 boyutlu veya 3 boyutlu olabilir.

Circle'ın Ellipse'in bir alt sınıfı olarak gösterilmesi ilginçtir.

Matematiksel olarak bir daire, bir elipsin tüm özelliklerine sahiptir. Bir elipsin iki odağı vardır; Bir daire içinde bu iki odak merkezde birbirine eşit olacak şekilde sınırlandırılmıştır. Nesneye yönelik bir sistemde alt sınıflar, üst sınıflarının tüm özelliklerine sahip olmalıdır; elips durumunda, geçerli bir işlem odaklardan birini değiştirmektir. Bu, Çemberin bir odağını değiştirebilmemiz gerektiği anlamına geliyor ki bu biraz tuhaf. Buna, diğer odağın otomatik olarak değişmesi ve dairenin tek merkezli bir daire olarak kalması koşuluyla izin verebiliriz. ama, bu

Çözüm tamamen tatmin edici değil çünkü Circle'ın her örneğinin odakları depolamak için hala iki özelliğe sahip olması gerekiyor. Dairelerin ve elipslerin niteliklerini düzenlemenin farklı bir yolunu gösteren alternatif bir alt hiyerarşi Şekil 2.7'de gösterilmektedir. Diğer bir seçenek de Circle sınıfından tamamen kurtulmak ve sadece Ellipse sınıfını kullanmaktır; daha sonra belirli elipslerin her zaman daire olarak kalmasını istiyorsanız Ellipse'e constrainAsCircle Boolean niteliğini ekleyebilirsiniz.

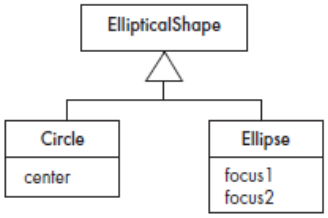

Figure 2.7 An alternative approach to defining ellipses and circles that avoids difficulties that would occur if `Circle` were a subclass of `Ellipse`