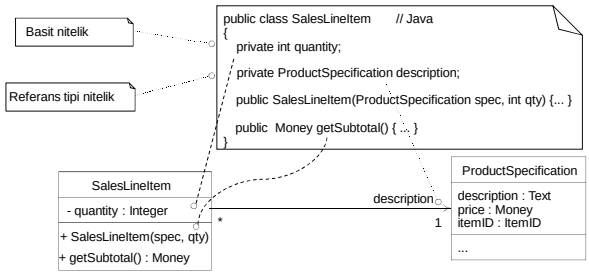
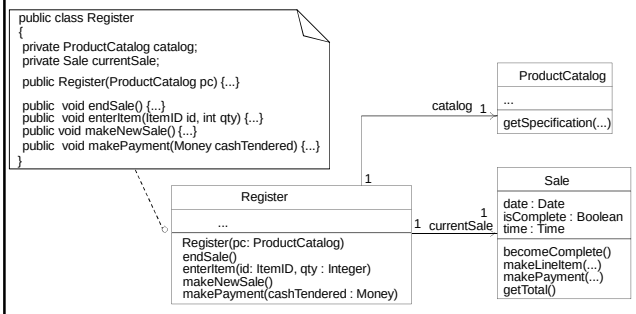
Kodlama

Sınıf tanımları yazılım sınıflarının diyagramlarından yararlanılarak oluşturulur.

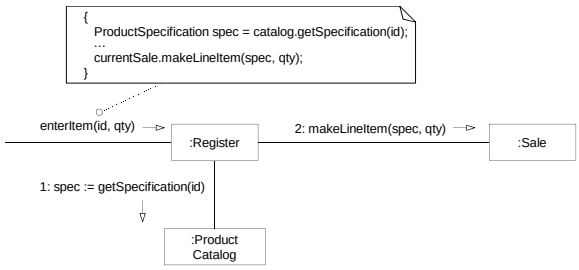
Karmaşık veri tiplerine (örneğin sınıf) sahip üyeler referans ya da işaretçi olarak yaratılmalılar.



Sınıflarda yer alan metotların imzaları sınıf diyagramlarından belirlenir.
Metotlar arası etkileşim ise etkileşim diyagramlarından yaralanılarak oluşturulur.

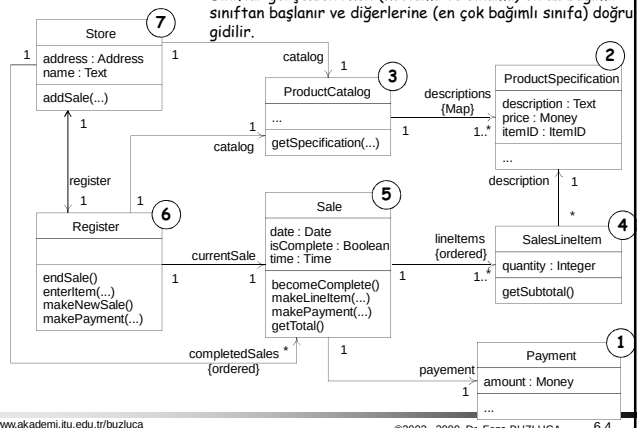


Şekildeki 1 ve 2 numaralı mesajlar Register sınıfının enterItem metodunda birer satır oluştururlar.



Sınıfların Gerçeklenme Sıraları

Sınıflar gerçeklenirken (kodlama ve sınama) en az bağımlı sınıftan başlanır ve diğerlerine (en çok bağımlı sınıfa) doğru gidilir.



Sınama Güdümlü Gerçekleme (*Test-Driven Development*)

Birim Sınama (*Unit Test*): Her sınıf bir bütün oluşturduğundan ve tek başına bir anlam taşıdığından ayrı bir birim olarak test edilir.

Sınıfların kodlarını yazmadan önce bu sınıfların sınamasını yapacak program parçalarının yazılması uygun bir yöntemdir (*Test-first programming, test-first development, **test-driven development***).

Sınamayı yapan program parçası (sınama sınıfı) aşağıdaki işleri yapar:

- Sınaması yapılacak olan sınıftan nesneler yaratır,
- Bu nesnelere mesajlar gönderip sonuçlar alır ve sınıfın gönderilen mesajlara doğru yanıtlar verip vermediği kontrol edilir.
- Metotların parametreleri zorlanarak (sınır dışı değerler verilerek) yazılımın sağlamlığı sınanır.

Tüm sınama sınıfı bir defada yazılmaz. Önce sadece bir metodu sınayan program parçası yazılır, ardından sınıfın metodu yazılarak test edilir.

Bir metot sınamadan geçtikten sonra diğer metodun sınama programına başlanır.

Sınama platformları:
Junit: Java, http://www.junit.org
Nunit: .NET, http://www.nunit.org

CruiseControl: Open Source
http://cruisecontrol.sourceforge.net/

Sınama programını önce yazmanın yararları:

- Sınama programlarının göz ardı edilmesi önlenir.
- Programcılar motive olur. Sınamadan geçmenin başarısı
- Sınama programı yazılırken kodlanacak olan sınıfın arayüzü hakkında daha ayrıntılı düşünülmüş olur.
- Kodlarda değişiklik yapıldığında yapılan değişikliğin bir hataya neden olup olmadığını belirleyecek hazır sınama programları el altında bulunur.

Örnek: Sale sınıfının makeLineItem metodunun test edilmesi

```
public class SaleTest extends TestCase
{
   public void testMakeLineItem(){
      Sale fixture = new Sale();            // Sınamada kullanılacak olan nesne
      Money total = new Money(7.5);   // Yardımcı nesneler
      Money price = new Money(2.5);
      ItemId id = new ItemId(1);
      ProductDescription desc = new ProductDescription(id, price, "product 1");
         // Metot sınanıyor ...
      fixture.makeLineItem(desc, 1);
      fixture.makeLineItem(desc, 2);
      assertTrue( fixture.getTotal().equals(total));   // Toplam doğru hesaplanıyor mu?
      :::
```

Örnek Kodlama

Bu bölümde örnek POS sistemine ilişkin yazılım sınıflarının bir kısmı örnek olarak Java dilinde kodlanmıştır.

```
public class Register
{
   private ProductCatalog catalog;
   private Sale currentSale;

  public Register( ProductCatalog catalog )
   {
      this.catalog = catalog;
   }

   public void makeNewSale()
    {
      currentSale = new Sale();
    }
  public void enterItem( ItemID id, int quantity )
   {
      if (currentSale !=NULL){
          ProductSpecification spec = catalog.getSpecification(id);
          currentSale.makeLineItem(spec, quantity);
      }
       else ...                // exception handler
    }
```

```
// Aşağıdaki metodlarda da
//   if (currentSale !=NULL) kontrolü yapılmalı

public void endSale()
  {
    currentSale.becomeComplete();
  }
public void makePayment( Money cashTendered )
  {
    currentSale.makePayment (cashTendered);
  }
}       // Register sınıfının sonu
```



```
public class ProductSpecification
{
   private ItemID id;
   private Money price;
   private String description;

   public ProductSpecification( ItemID id, Money price, String description )
   {
      this.id = id;
      this.price = price;
      this.description = description;
   }

   public ItemID getItemID() { return id; }

   public Money getPrice() { return price; }

   public String getDescription() { return description; }

}
```



```
public class Sale
{
   private List <SalesLineItem> lineItems = new ArrayList() <SalesLineItem> ;
   private Date date = new Date();
   private boolean isComplete = false;
   private Payment payment;

   public Money getBalance()
   {
      return payment.getAmount().minus(getTotal());
   }
   public void becomeComplete() { isComplete = true; }
   public boolean isComplete() { return isComplete; }
   public void makeLineItem( ProductSpecification spec, int quantity )
   {
      lineItems.add ( new SalesLineItem(spec, quantity) );
   }
```



```
// Sale sınıfının devamı
public Money getTotal()
 {
   Money total = new Money();
   Money subtotal = null;
   Iterator i = lineItems.iterator();
   while( i.hasNext() )
   {
     SalesLineItem sli = i.next();
     subtotal = sli.getSubtotal()
     total.add( subtotal );
   }
   return total;
 }

  public void MakePayment ( Money cashTedered )
  {
    payment = new Payment (cachTendered);
  }
}         // Sale sınıfının sonu
```


C++ ile Örnek Kodlama

Bu bölümde örnek POS sistemine ilişkin yazılım sınıflarından üçünün kodu örnek olarak C++ dilinde verilmiştir.

a) UrunTanimlayici (ProductSpeification) Sınıfı:

```
class UrunTanimlayici
{
  private:
     UrunKod kod;
     Para fiyat;
     string tanim;
  public:
    UrunTanimlayici( const UrunKod &kd, const Para &ft, const string &tnm )  // Kurucu
   {
      kod = kd;
      fiyat = ft;
      tanim = tnm;
   }

   const UrunKod & urunKoduVer() { return kod; }

   const Para & fiyatVer() { return fiyat; }

   const string & tanimVer() { return tanim; }
};
```


b) Satis (Sale) Sınıfı:

```
class Satis
{
 private:
    vector <SatisKalemi*> kalemler;
    Tarih tarih;
    bool tamam_mi;
    Odeme *odeme;

public:
   Satis(){
     tamam_mi = false;
   }
   Para & paraUstuVer()
   {
     return odeme->miktariVer().eksi(toplamiVer());
   }
   void bitir() { tamam_mi = true; }
   bool bitti_mi() { return tamam_mi; }
   void kalemOlustur( const UrunTanimlayici *tnm, int miktar )
   {
      SatisKalemi *sk = new SatisKalemi(tnm, miktar);  // Yeni bir Satış kalemi yaratıldı
      kalemler.push_back ( sk );                       // Satış kalemi diziye (vector) eklendi
   }
```

Satis (Sale) Sınıfı devamı:

```
Para toplamiVer()
{
    Para toplam;
    for(unsigned int j=0; j<kalemler.size(); j++){    // Bütün kalemler taranıyor
      SatisKalemi *sk = kalemler[j];                  // Diziden bir kalem alınıyor
      toplam.arti( sk->altToplamiVer() );             // Alttoplam toplama ekleniyor
    }
   return toplam;
}

void odemeYap ( Para & nakit )
{
    odeme = new Odeme (nakit);                  // Yeni bir ödeme yaratıldı
}

~Satis ()
{
   delete odeme;                                // Satış yok olurken Ödeme yok edildi
}
};          // Satis sınıfının sonu
```


c) Terminal (Register) Sınıfı:

```
class Terminal
{
  private:
    UrunKatalog * katalog;
    Satis * gecerliSatis;
 public:
  Terminal( UrunKatalog * katalog )
  {
    this->katalog = katalog;                    // Terminalin kullanacağı katalog
  }

  void satisBaslat()
  {
    gecerliSatis = new Satis();                 // Yeni bir satış yaratıldı
  }
```


Terminal (Register) Sınıfı devamı:

```
void urunGir( UrunKod & kod, int miktar )
  {
     if (gecerliSatis !=NULL) {
       UrunTanimlayici *tnm = katalog->tanimlayiciVer(kod);   // katalogtan tanımlayıcı alınıyor
       gecerliSatis->kalemOlustur(tnm, miktar);               // girilen ürün satışa aktarılıyor
     }
     else   ....  // exception
  }

void satisBitir()              // if (gecerliSatis !=NULL) kontrolü yapılmalı
  {
    gecerliSatis->bitir();
  }

void odemeYap( Para & nakit )
  {
    gecerliSatis->odemeYap(nakit);
  }
};      // Terminal sınıfının sonu
```