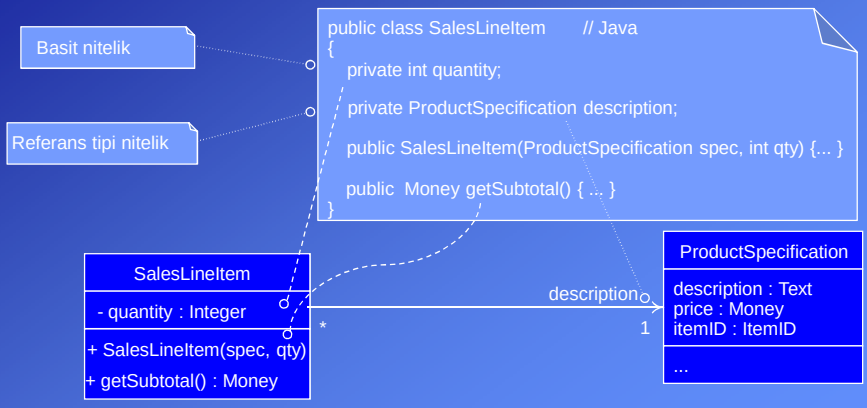
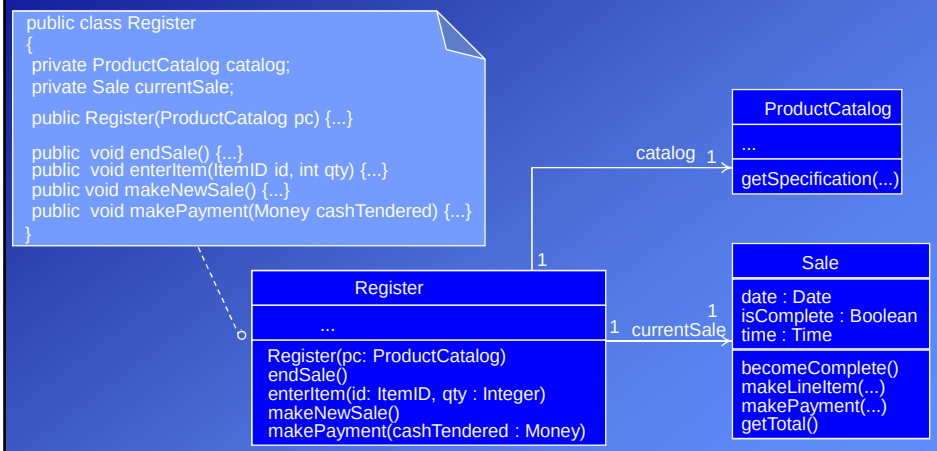
Kodlama

Sınıf tanımları yazılım sınıflarının diyagramlarından yararlanılarak oluşturulur.

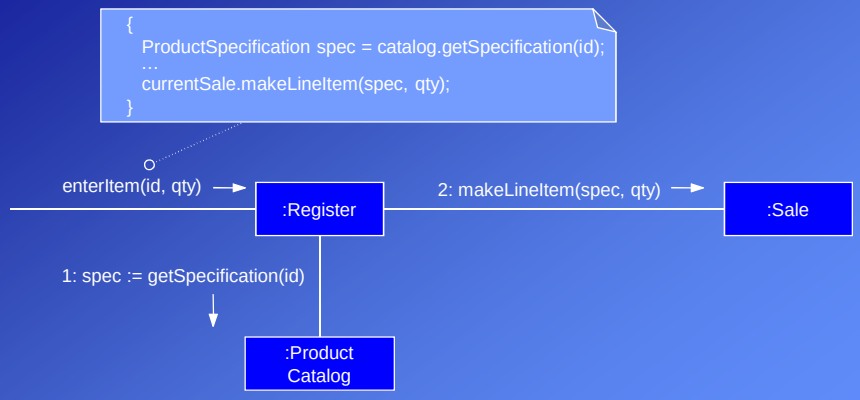
Karmaşık veri tiplerine (örneğin sınıf) sahip üyeler referans ya da işaretçi olarak yaratılmalılar.



Metotlar etkileşim diyagramlarından yaralanılarak oluşturulur.



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



Sınıfları Gerçekleme Sırası

Sınıflar gerçeklenirken (kodlama ve sınama) en az bağımlı sınıftan başlanır ve diğerlerine (en çok bağımlı sınıfa) doğru gidilir. Yansı 6.5'te örnek sınıfların olası gerçeklenme sıraları gösterilmiştir.

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*).

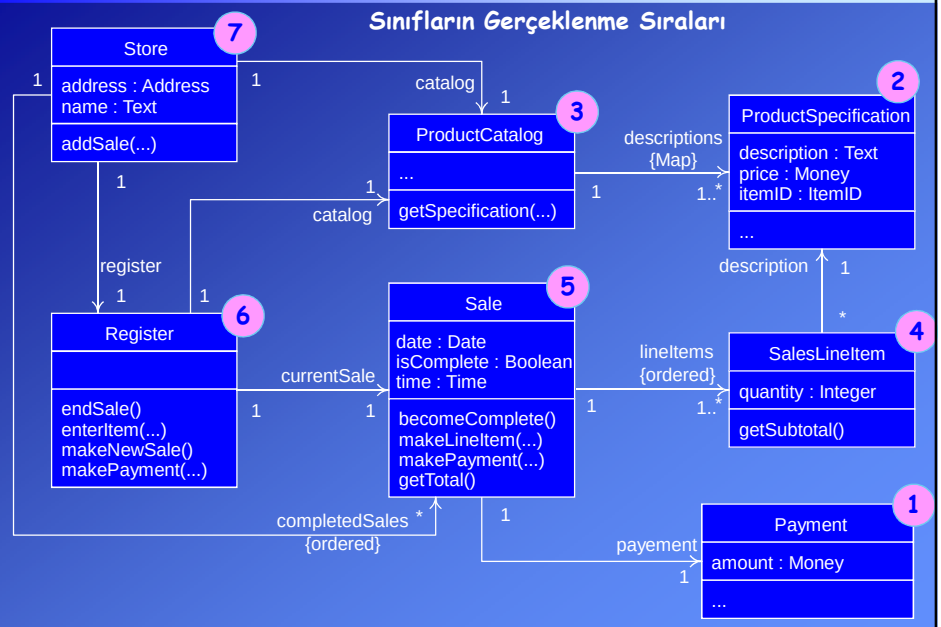
Sınamayı yapan program parçası, sınaması yapılacak olan sınıftan nesneler yaratır,

bu nesnelere mesajlar gönderip sonuçlar alır ve sınıfın gönderilen mesajlara doğru yanıtlar verip vermediği kontrol edilir.

Sınama programını önce yazmanın yararları:

- Sınama programlarının göz ardı edilmesi önlenir.
- 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.

Sınıfların Gerçeklenme Sıraları



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 lineItems = new ArrayList();
   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();
   Iterator i = lineItems.iterator();
   while( i.hasNext() )
   {
     SalesLineItem sli = (SalesLineItem) i.next();
     total.add( sli.getSubtotal() );
   }
   return total;
 }

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