Smalltalk ile Yazılım Geliştirme: Squeak(TASLAK)

Zekeriya KOÇ

09-07-2007

Özet

Bu belge, Squeak bütünleşik geliştirme ortamını kullanarak Smalltalk programlama dili ile yazılım geliştirme süreçlerine giriş niteliğinde kaynak sağlaması amacı ile yazılmıştır.

İçindekiler

Kısım I
Giriş

Smalltalk programlama dili ilk programlama dillerinden biridir. Tam olarak Modula programlama dilinden sonra nesneye yönelimli olarak ifade edilebilecek ilk programlama dilidir ve bu kavramı bilgisayar dünyasına yerleştiren dil olmuştur. II. Kısımda açık kaynak kodlu bir Smalltalk gerçekleştirimi olan Squeak ortamı tanıtılacaktır. III. Kısımda Smalltalk programlama dili sözdizimine ve genel prensiplerine temel bir giriş yapılacaktır.

Kısım II
Squeak Geliştirme Ortamı

1 Neler Lazım?

Smalltalk geliştirme ortamı squeak sanal makinesi üzerinde çalışan bir imajdır. Bu imajı kaydettiğiniz zaman her şeyi ile içinde bulunduğunuz geliştirme "Dünya"sını kaydetmiş olursunuz ve imajı tekrar açtığınızda tam olarak kaydettiğiniz andaki duruma geri dönersiniz.

2 Kurulum

Smalltalk ile geliştirmeye başlamak için en az iki bileşene ihtiyaç duyacaksınız: 1-Sanal Makine 2- İmaj. Çeşitli işletim sistemleri için sanal makine ve imaj dosyalarını bu adresten[1] indirebilirsiniz. Kurulum oldukça basit. Linux sistemler için sanal makine kurulumu yapılması gerekiyor ve bunun için indirdiğiniz arşivden çıkan INSTALL betiğini kullanabilirsiniz. Sonrası bir imaj indirip kaydetmek ve imaj ile aynı klasördeyken "squeak" komutunu çalıştırmaktan ibaret. Windows ortamı içinse Sqeuak ana sayfasındaki "Windows" linkini kullanarak hem sanal makineyi hemde imaj dosyasını içeren arşivi indirebilirsiniz. Arşiv içindeki "squeak.exe" programını çalıştırdığınızda sqeuak ortamı karşınıza gelecektir.

3 Temel Araçlar

Squeak çalıştığında bazı pencereler içeren boş sayılabilecek bir ekran karşılar bizi. Bu ekran Smalltalk ile geliştirme yaparken içinde yaşayacağımız dünyamızdır. Geliştirme için kullanabileceğimiz bazı araçları içeren bir çekmeceyi sağ tarafta "Tools" başlığı ile göreceksiniz. Bu çekmece elimizdeki geliştirme araçlarınının sadece küçük bir kısmına hızlı erişim sağlamaktadır. Dünyamızda etkileşime geçerken en sık kullanacağımız menü olan "World"(dünya) manüsünü açmak için ekranın boş bir yerine farenin sol tuşuyla tıklamamız gerekir. Karşımıza gelen menü bir çok başka menüye geçiş sağlamakla birlikte "do it", "print it", "incpect it" gibi temel bazı komutları vermemizi de sağlar. Bunların yanında önemli bir nokta "World Menu" aracılığı ile imajımız kaydedebiliriz. Squeak ortamında yaptığımız her değişiklik (buna yazdığımız kodlarda dahil çünkü yazdığımız kodlar bu dünyada "yaşıyorlar") imajı kaydetmezsek kaybolacaktır.

3.1 Workspace

Temel bir kaç geliştirme aracını tanıyalım kısaca. Bu esnada squeak ortamının genel prensiplerini de görmüş olacağız. Öncelikle nereye kod yazacağız diye sorabilirsiniz. Haklısınız tabii ki. Küçük kod parçaları yazıp hemen sonuçlarını görmek istediğimizde kullanacağımız "Workspace" isimli bir aracımız var. Denemeler yapmak için çok kullanışlı olan bu aracı isterseniz Tools çekmecesinden

[1] http://www.squeak.org/Download/

dünyaya sürkleyip bırakabilir, isterseniz "Dünya" menüsünden "Open..." seçeneğini seçtiğinizde çıkan menüden seçerek açabilirsiniz. Karşımıza gelen pencere içine biraz kod yazalım ve bakalım neler yapabileceğiz. Workspace içine 1 + 2 yazın ve yazdığınız ifadeyi fare ile seçerek sağ tıklayın (Linux için orta fare tuşu). Gelen menüde "print it" seçeneğini seçerseniz yazdığınız ifadenin hemen yanında işlemin sonucu olan "3" sayısını göreceksiniz. "print it" seçeneği yanında kısayolunu da göreceksiniz (p). Bunun anlamı ifade seçiliyken Alt-p tuşlarına birlikte basarsanız "print it" işlemini çağırmış olacaksınız demektir. Diğer işlemlerin kısayollarını da tabii ki aynı şekilde kullanabilirsiniz.

3.2 Transcript

Bir başka aracı görelim şimdi. Bu esnada kod çalıştırmak içinde bir fırsatımız olacak. Yine ister Tools çekmecesinden ister World->Open...->Transcript yolu ile bir "Transcript" penceresi açın. Bu pencere bir çıktı görüntüleme penceresidir. Ve evet klişeleşmiş "Merhaba Dünya!!" söz öbeğini bu pencere içinde görüntüleyeceğiz. Workspace penceresine geri dönelim ve daha önce yazdığımız işlemin altından devam ederek şunu yazalım:

```
Transcript show: 'Merhaba Dunya'.
```

Şimdi bu kod parçasını fare ile seçip "print it" işlemini uygularsak bize "Merhaba Dünya" yazısını değil "a Transcript stream" mesajını verecektir. Bu komutun sonucu Transcript penceresinde görüntülenmek üzere hazırlanmış bir akıştır. Mesajımızı çıktı penceremizde görüntülemek için yazdığımız kod parçasını "print it" değil "do it" komutu ile çalıştırmamız gerekir. Bunu kod seçiliyken sağ fare tuşu menüsünden yapabileceğiniz gibi Alt-d kısayolu ile de yapabilirsiniz. İşlem sonunda Transcript penceresinde Merhaba Dunya yazısını göreceksiniz. Tebrikler!!! Squeak ile "Merhaba Dunya" işlemini yapmış bulunuyoruz. :)

Smalltalk, gerek ilk nesneye yönelimli programlama dillerinden biri olması gerekse nesne yönelimli programlama kavramını bilgisayar literatürüne yerleştiren dil olması sebebiyle, geliştirme süreci sınıflar, sınıfların tanımları, sınıf özellikleri ve sınıf metodları (nesneye gönderilen mesajlar) inşa edilerek ilerler.

3.3 Class Browser

Peki bu sınıfları nasıl tanımlayacağız? Bunun için "System Bowser" denilen bir aracımız var ("Browser", "Class Browser" şeklinde de ifade edilir). Bu aracıda diğerleri gibi Tools çekmecesinde bulabilir ve ayrıca World->Open...->Class Browser yoluyla açabilirsiniz. Karşınıza gelen pencere Smalltalk ile çalışırken en çok vakit geçireceğiniz penceredir.

Class Browser penceresi 5 ana bölüme ayrılmıştır. Üst kısımda kalan ilk bölüm sistem kategorilerini içerir. Halihazırda Squeak sisteminde fazlasıyla sınıf mevcuttur. Bunlara, gerek yeni kuracağınız paketlerin gerekse kendi yazacağınız uygulama sınıflarının ekleneceğini düşünürsek bütün bu sınıfların içinde düz bir liste halinde gezinmek ve aradığınızı bulmak çok ciddi bir sorun oluşturacaktır. Bu olası sorun sınıfları kategorilere ayırarak çözülmüştür. Sınıf kategorileri yazdığınız uygulamanın çalışmasında herhangi etkiye sahip değildir. Sadece sisteminizin düzenli olmasını ve aradığınızı daha rahat bulmanızı sağlar (tabii ki Squeak'in çok daha gelişmiş sınıf ve kod bulma mekanizmaları vardır).

Kategori bölmesinin yanında sınıfların bulunduğu bölme vardır. Seçtiğiniz kategori altındaki sınıflar burada listelenecektir. Mevcut sınıflardan birini seçerseniz, aşağıdaki büyük bölmede sınıfın tanımını görebilirsiniz. Bu tanımın detaylarına dil özelliklerin bahsederken ineceğiz. Sınıf isimlerinin yanında ise sınıf kategorilerine benzer bir şekilde metot kategorileri bulunur. Uygulamanızın iş yapan kısmı olan metotlarınızın yaptıkları işlere göre sınıflandırıldığı yer burasıdır. Squeak bazı sık kullanılan metot kategorilerini size otomatik olarak sunar. Örnek bir sınıf tanımı yaparken bu işleme de değineceğiz. Bir metot kategorisi seçtiğinizde üst taraf en sağdaki metot listesi aktif hale gelir ve seçtiğiniz kategorideki metotları listeler. Son olarak metot listesinden bir metot seçerseniz aşağıdaki büyük bölmede ilgili metodun içerdiği Smalltalk kodunu görebilirsiniz.

3.3.1 Örnek Bir Sınıf Tanımı

Şimdi Class Browser aracını kullanarak örnek bir sınıf tanımlayalım. Yalnız bu işlem esnasında yazacağımız küçük kod parçalarının detaylarına girmeyeceğiz ve işlemin mekanik kısmına odaklanacağız. Dil konusundaki temel açıklamalar takip eden kısımda ele alınacaktır.

Öncelikle örnek çalışmalarımız için bir sınıf kategorisi tanımlayalım. Bunub için Class Browser penceresinin üst taraf en soldaki bölmesinde sağ tıklayarak (Linux için öntanımlı orta fare tuşu), gelen menüden "Add Item..." işlemini seçelim. Karşımıza gelecek olan pencereye oluşturmak istediğimiz kategorinin adını yazıp "Accept" düğmesine tıkladığımızda, altında örnek sınıfımızı oluşturacağımız kategorimiz oluşacaktır.

Yeni kategorimiz seçili hale getirdiğimizde, alt kısımdaki kod bölmesinde yeni bir sınıf tanımı için hazırlanmış bir şablon göreceksiniz. Şöyle ki;

```
Object subclass: #NameOfSubclass
instanceVariableNames: ''
classVariableNames: ''
poolDictionaries: ''
category: 'Zek-Deneme'
```

Sınıf tanımımızı bu şablon üzerinden yapacağız. Sözdizimi detaylarına giriyoruz ama "#NameOfSubClass" yazan yere yeni sınıfımızın adını yazmamız gerekiyor. Kod bölmesinin son hali şöyle olmalı;

```
Object subclass: #DenemeSinifi
instanceVariableNames: ''
classVariableNames: ''
poolDictionaries: ''
category: 'Zek-Deneme'
```

Kod bölmesinde bir değişiklik yaptığınızda bölmenin çerçevesi kırmızı renk alır. Bu, o bölmedeki değişikliklerin henüz kaydedilmediğini gösterir. Bu durumdayken kod bölmesinden çıkmak isterseniz, kaydetmediğiniz değişikler ile ilgili bir uyarı alırsınız ve Squeak size çıkış işleminden vazgeçme şansı tanır. Sınıf tanımımızı şimdilik tamamladık. Sınıfımızın oluşturularak Squeak sisteminde yerini alması için, tanımı kaydetmemiz, bir diğer değişle bölmenin çerçevesindeki kırmızı rengi yok etmemiz gerekir. Bu işi kod bölmesinde sağ tıklayarak (Linux için hangi tuş olduğu yukarıda belirtilmişti) gelen menüden "Accept(s)" işlemini seçerek yapıyoruz. İşlemin yanındaki kısayolun nasıl kullanılacağı belgenin önceki

bölümlerinde belirtilmişti.

Kayıt işlemi yapıldığında sınıfımız, kategori bölmesinin hemen yanındaki sınıf bölmesinde yerini almış olmalıdır. Şimdi çiçeği burnunda sınıfımıza bir metod tanımlayabiliriz. Metod tanımlayabilmek için bir ön şart olmasa da, her zaman metodlarınızı metod kategorileri altında tutmanız tavsiye edilir. Yani öncelikle bir metod kategorisi oluşturacağız. Bunun için sınıf bölmesinin sağındaki bölmede sağ tıklayın ve menüden "New Category..." seçeneğini seçin. Squeak öntanımlı kategorileri gösteren bir menü getirecektir karşınıza. Buradan bir kategori seçebileceğiniz gibi, en üstteki "New..." seçeneği ile tamamen yeni bir kategori de oluşturabilirsiniz. Biz de böyle yapalım ve "actions" adında bir kategori oluşturalım.

Şimdi oluşturduğunuz mesaj (Smalltalk jargonunda metodlar mesaj olarak ifade edildiğinden artık biz de bu ismi kullanacağız) kategorisini seçerseniz, daha önce sınıf oluştururken olduğu gibi aşağıdaki kod bölmesinde bir mesaj şablonu göreceksiniz ve yeni mesajınızın gövdesini buraya gireceksiniz. Bu arada mesajları herhangi bir kategori altında oluşturmayıp, daha sonra istediğiniz gibi kategorize etmeniz de mümkün. Mesaj şablonu şu şekilde olacaktır;

```
messageSelectorAndArgumentNames
  "comment stating purpose of message"

  | temporary variable names | statements
```

Yine sözdizimine fazla odaklanmadan mesaj şablonunu aşağıdaki şekilde değiştirelim;

```
konus
  ^ 'Merhaba!!!'
```

Mesajı, Alt-s tuş kombinasyonu ile kaydettiğimizde bir kereye mahsus olmak üzere bir başlangıç değeri girmeniz gerekecek buraya meslea "0" vererek geçebilirsiniz. Kayıt işlemi tamamlandığında, mesaj bölmesinde mesajımızın yerini aldığını göreceğiz. Evet şu an da kendisine gönderilen bir mesaja cevap verebilen bir sınıfımız var. Bu sınıfımızı kısaca Workspace penceresi içinde kullandıktan sonra Smalltalk programlama dilinin temellerine başlangıç (giriş seviyesinde) yapabiliriz.

İlk örneklerimizi yaptığımız Workspace penceresine aşağıdaki kod parçasını yazalım;

```
| a |
a := DenemeSinifi new.
a konus.
```

Burada kısaca, sınıfımızın yeni bir örneğini oluşturup bir değişkende depoluyoruz ve bu örneğe "konus" mesajını gönderiyoruz. Bize vermesi gereken cevabın "Merhaba!!!" olduğunu biliyoruz. Kodun hepsini seçip Alt-p tuşlarına basarsanız sınıfın yapması gerekeni yaptığını ve "Merhaba!!!" karakter katarını yazdırdığını göreceksiniz.

Kısım III
Smalltalk Programlama Dili

4 Önceki Bölüm Örneklerinin Açıklanması

4.1 Nesne - Mesaj

Önceki bölümde kullandığımız ve detaylarına inmekten kaçındığımız kod parçalarına şimdi sözdizimini incelemek için biraz daha yakından bakacağız.

```
Transcript show: 'Merhaba Dunya'.
```

Bu kod parçasında, Smalltalk programlama dilinin nesneye yönelimli yaklaşımının temel yapısını görebiliriz. Transcript isimli nesneye, üzerinde tanımlanmış olan "show" mesajını "Merhaba Dunya" parametresi ile göndermiş oluyoruz ve o da tanımlanan işini yaparak yolladığımız parametreyi kendi üzerine yazıyor. Buradan çıkan bir kaç sonuç;

- Mesajlara parametre gönderirken iki nokta üstüste (:) ile bunu yapıyoruz.
- Karakter katarları (string) tek tırnak (') sembolü ile çevreleniyor.
- Smalltalk programlama dilinde ifade ayracı bildiğimiz nokta (.).

4.2 Sınıf Tanımlama

```
Object subclass: #DenemeSinifi
instanceVariableNames: ''
classVariableNames: ''
poolDictionaries: ''
category: 'Zek-Deneme'
```

Sınıf Tanımlama şablonu. Aslında burada da aynen bir önceki örnekte yapılan işlemler yapılıyor. "Object" nesnesine çeşitli mesajlar gönderiliyor. "subclass" mesajı "Object" sınıfından bir alt sınıf türetmek istediğimizi söylüyor ve parametre olarak yeni sınıfın adını sembol olarak mesaja ekliyor. Ve diğer mesajlarıda çeşitli parametreler ile gönderiyoruz. İşlem sonucuda yeni sınıfımız oluyor ve Class Browser penceresinde yerini alıyor. Yukarıdaki kod parçasını Class Browser içinde yazarak kaydettiğimizi ve bu işlem sonucunda yeni sınıfımızın oluştuğunu hatırlayacaksınız. Bu kod parçasını kopyalayarak bir Workspace içinde çalıştırırsanız, Class Browser içinde aldığınız sonucun aynısını aldığınızı göreceksiniz. Verdiğiniz isimde bir sınıf oluşacak ve anında Class Browser içindeki yerini alacaktır.

4.3 Çalışan Kod

Şimdi önceki bölümün son kod örneğini biraz değiştirelim;

```
"Yeni sinifimizdan bir ornek olusturup ona bir mesaj gonderen kod."
| a |
a := DenemeSinifi new.
a konus.
```

Burada yine bir kaç Smalltalk sözdizimi kuralını görüyoruz;

- Yorumlar/Açıklamalar çift tırnak karakterleri (")arasında bulunmalıdır.
- Değişkenler kullanılmadan önce tanımlanmalıdırlar. Bu boru karakterleri (|)içinde yapılır. Birden fazla değişken tanılama isterseniz aralarında birer boşluk bırakarak bunu yapabilirsiniz. |a b c d| gibi.
- Atama operatörü iki noka üstüste ve eşittir sembollerinden oluşur (:=).

4.4 Sınıf Tanımları (Biraz Daha Detay)

Java, Python gibi nesneye yönelimli programlama dillerinden aşina olabileceğiniz bazı kavramlar vardır.

- Sınıf Özellikleri (Alanları).
- Sınıf Metotları.
- Statik Özellikler (Alanlar) ve metotlar.

Bu kavramlar tabii ki Smalltalk progamlama dilinde mevcutlar. Aslında bugünkü nesneye yönelimli dillerin atası sayılabilecek olan bu dilde, bu kavramların yer almaması oldukça garip olurdu.

4.4.1 Sınıf Özellikleri

Sınıf özellikleri ile başlayalım. Aşağıdaki Python kodunu inceleyelim;

```
class Deneme:
  alan1 = None
  alan2 = 1
```

Burada Deneme sınıfına alan1 ve alan2 adında iki tane sınıf özelliği (ya da sınıf alanı) tanımlamış oluyoruz. Aynı işlemi bir Smalltalk sınıfında yapmak istediğimizde, daha önce oluşturduğumuz sınıfımızın koduna aşağıdaki gibi ekleme yapmamız gerekir;

```
Object subclass: #DenemeSinifi
instanceVariableNames: 'alan1 alan2'
classVariableNames: ''
poolDictionaries: ''
category: 'Zek-Deneme'
```

Bu eklemeleri yapıp sınıfı kaydedersek sınıfımızın ve bu sınıfın örneklerinin (nesne) alan1 ve alan2 adında iki tane alanı olacaktır.

Erişim Metotları Sınıfınızı bu şekilde tanımladınız diyelim. Bu alanların kullanılabilir olabilmesi için alanlara erişim metotları tanımlamamız gerekir. Bu, nesneye yönelimli programlama dillerinde kapsülleme ya da sarmalama (encapsulation) olarak bilinen prensibi uygulamanızı sağlar. Alanlarınıza direkt erişimi engeller ve kontrolünüz altında tutabilirsiniz. Bunun için aslında her bir alan için bir tane okuma bir tanede yazma mesajı tanımlamanız gerekir. Ama Squeak bu işi sizin için otomatik olarak yapabilir. Sınıfınızın adının yazdığı bölmeye sağ tıklayıp menüden "more..." seçeneğini seçin. Yine gelen menüden "create inst var

accessors" seçeneğini seçin. Mevcut alanlarınız listelenecektir. İstediğiniz alanı seçin. Mesaj kategorileri bölmesinde "accessing" isimli bir kategori ve mesaj bölmesinde alan adları ile oluşmuş iki mesaj göreceksiniz. Yeni oluşan mesajları inceleyelim. Oldukça basitler.

```
alan1
  ^ alan1
```

```
alan1: anObject
  alan1 := anObject
```

İlk mesaj okuma için kullanılır. "a" DenemeSinifi tipinde bir nesne ise "a alan1." komutu alan1 özelliğinin o anki değerini dönecektir. "^" karakteri "return" anlamına gelir yani değer döndürür. İkinci mesaj yazma mesajıdır. "a alan1: 'selamm'." komutu verirseniz alan1 özelliğinin değeri 'selamm' karakter katarı olarak atanır. Tabii ki mevcut erişim metotlarının alanlara direkt erişimden bir farkı yoktur. Siz isterseniz bunların içine çeşitli kontrol kodları yazarak, sınıf alanlarınızı sarmalayabilirsiniz.