Eiffel nedir?

Aslında, bu durumda yazılım geliştirmeye kapsamlı bir yaklaşım olan “Eiffel Development Framework ™” kısaltmasıdır. Sağlam, yeniden kullanılabilir yazılımların baştan sona oluşturulması için Eiffel metodolojisinden oluşur; metodolojiyi destekleyen Eiffel dili; ve Eiffel derleyicisini içeren ortam olan EiffelStudio ™ ve geliştirme ortamını oluşturan verimlilik araçlarının tamamı. Tek tek parçalar, mümkün olan en iyi yazılımı sağlama arayışında birbirlerinin işlevleriyle uyumludur ve onları destekler.

Bu kadar basit ve güçlü bir şekilde çalışmak için başka hiçbir yazılım geliştirme sistemi tasarlanmamıştır. Bu yaklaşımın ve tasarımın sonuçları şaşırtıcı. Verimlilik artar. Sahip olma maliyeti düşer. Kalite seviyeleri yükselir.

Eiffel, şirketlerin yazılımlarını diğer dillerin ve geliştirme araçlarının yeteneklerinin çok üzerinde bir verimlilik ve güvenilirlik seviyesine taşır.

(Object-Oriented Programming’i temel,baz edinmiş bir dildir.)

İsmi nereden geliyor?

Eiffel adını ünlü kuleyi yaratan mühendis Gustave Eiffel'den almıştır. 1889'da 1889 Dünya Fuarı için inşa edilen Eyfel Kulesi, Eiffel 'de yazılmış yazılım projeleri gibi zamanında ve bütçe dahilinde tamamlandı. Bu harika yapıya bakarsanız, tıpkı Eiffel Yazılımının yeniden kullanılabilir kitaplıklarından oluşan bir sistem gibi, son derece güçlü, aşağıdan yukarıya bir yapı elde etmek için tekrar tekrar birleştirilen ve değişen az sayıda sağlam tasarım deseni göreceksiniz. Günümüzde birçok yazılım sistemi gibi, Eyfel Kulesi başlangıçta geçici bir yapı olarak düşünülmüştü; ve Eiffel ile inşa edilen birçok sistem gibi, orijinal hedeflerinin çok ötesine dayanabildi.

Eiffel ve EiffelStudio arasındaki fark nedir?

Eiffel, geliştiricinin harika bir yazılım yazmak için kullandığı dildir. EiffelStudio, Eiffel dilini çevreleyen ortam ve araç setidir.(Derleyici IDE)

Eiffel herhangi bir özel uygulama alanı için mi tasarlandı?

Özellikle değil. Eiffel, değişen pazar ve kullanıcı taleplerine uyum sağlaması gereken iddialı sistemlerde parlıyor. Ciddi uygulama geliştirmeye birçok fayda sağlar ve finansal uygulamalardan üretime, ürün konfigürasyon kontrolüne, sağlık hizmetlerine, telekomünikasyon sistemlerine, savunmaya kadar çeşitli endüstrilerde kullanılır.

Eiffel, soyutlama ve yapılandırma olanakları sayesinde ölçekleniyor. Projenizin (veya şirketinizin) boyutları ve kapsamları büyüdüğünde sizi hayal kırıklığına uğratmayacak birkaç ortamdan biridir.

Peki ya işletim sistemleri? Eiffel nerelerde çalışır?

Eiffel, geliştiricilerin sevdiği bir özellik olan çok taşınabilirdir. Windows (klasik ve .NET), Unix, Linux, VMS ve Mac OS X dahil olmak üzere hemen hemen her yerde çalışır. Bu, geliştiricilere tamamen farklı bir işletim sisteminde yeni kod geliştirirken eski kodlarını koruma esnekliği sağlar.

Eiffel nasıl ortaya çıktı ve tarihi nedir?

Eiffel, 1985 yılında Eiffel Software'de (daha sonra ISE olarak bilinir), başlangıçta çeşitli dahili uygulamalar geliştirmek için dahili bir araç olarak tasarlandı. Eiffel Yazılım mühendisleri, modern yazılım mühendisliği kavramlarını entegre eden güçlü, nesneye yönelik bir ortama ihtiyaç duyuyordu ve mevcut hiçbir şey yoktu. Bu nedenle, Eiffel Yazılımının kurucusu Dr. Bertrand Meyer, onlar için The Eiffel-1 derleyicisini tasarladı. Ekim 1986'daki ilk OOPSLA konferansında kamuoyuna tanıtıldı ve burada dikkat çekti ve bizi 1986 sonunda ticari bir ürün olarak piyasaya sürdü. Sonraki yıllarda hızla yayılan teknoloji, ABD, Kanada, Avrupa ve Uzak Doğu'da endüstriyel projelerde başarılı bir şekilde kullanıldı. En başından beri, Eiffel her seviyeden programlamayı öğretmek için ideal bir araç olarak akademik camianın dikkatini çekti. Dünyanın dört bir yanındaki çok sayıda üniversite bu dili ilk öğretim dili olarak kullanmıştır.

Çevrenin ardışık versiyonları yılda yaklaşık bir kez ortaya çıktı. Hızlı bir şekilde en çok satan unvan haline gelen ve sekiz dile çevrilen Dr. Bertrand Meyer'ın Nesne Odaklı Yazılım İnşaatı kitabının 1988'de yayımlanmasıyla Eiffel tanıma daha geniş görünürlük kazandı; kitap, Eiffel'i nesne teknolojisi ve Designt Contract ™ ile ilgili temel kavramları açıklamak ve göstermek için kullandı. O zamandan beri, bu kitabın genişletilmiş yeni bir baskısı yayınlandı: Nesneye Dayalı Yazılım İnşaatı, 2. Baskı.

Orijinal teknolojinin son tekrarı, 1990 yazında yayınlanan 2.3 sürümüdür. Bir sonraki sürüm olan Eiffel 3, tamamen Eiffel'de yazılmıştır; 2.3'ten önyüklendi. Eiffel 3, hızlı yeniden derleme için Erime Buz Teknolojisini, yenilikçi kullanıcı arayüzü konseptlerine dayanan tamamen grafik bir ortamı ve kütüphanelerdeki önemli grafikleri (grafik, ağ…) ve üretilen kodun optimizasyonunu tanıttı. İlk sürümler Unix'te yayınlandı ve ardından Linux, VMS, OS / 2, Windows (Windows 3.1, Windows 95, Windows NT), .NET ve Mac OS X izlendi.

Bugün, Eiffel teknolojisi yazılım geliştirmenin sınırlarını zorlamaya devam ediyor. EiffelStudio'nun tanıtımı ile programcılar, tüm büyük platformlarda en yüksek kalitede, sağlam, ölçeklendirilebilir, tekrar kullanılabilir yazılım uygulamalarına ulaşmak için verimli bir IDE'den yararlanabilir.(Son versiyonu 2018 yılında güncellenmiş olmaktadır.)

EiffelStudio nedir?

EiffelStudio, yalnızca Eiffel Nesnesi Odaklı dil için tasarlanmış Entegre Geliştirme Ortamıdır (IDE). Yazılım geliştirme sürecinin tüm yaşam döngüsünü sorunsuz bir şekilde ele alan EiffelStudio, ekibinizin ilk tasarım zamanından kurulum, test ve bakıma kadar geliştirmesine yardımcı olacak olanaklar sunar. İş Nesnesi Gösterimi (BON) yöntemi için Bilgisayar Destekli Yazılım Mühendisliği (CASE) aracı gibi yerleşik işlevler, geliştirme sırasında sisteminizin tasarımını görmenize ve etkileşimde bulunmanıza olanak tanır ve her şey aynı anda yapıldığından tersine mühendislik gerekmez. . EiffelStudio ayrıca kodunuz hakkındaki bilgileri görüntülemek için mükemmel tarama mekanizmalarına sahiptir ve yürütme sırasında çalışır ve davranır. Tam özellikli göz atmadan, metriklere ve profil oluşturmaya kadar, EiffelStudio size sisteminizle ilgili neredeyse her şey hakkında bilgi verebilir.

Design by Contract ™ metodolojisi etrafında merkezlenen EiffelStudio'nun hata ayıklayıcısı, yazılımınızın sizin için hataları bulmasına izin vererek diğer dillerle tasarlanmış sistemlerin dayanması gereken bakım maliyetlerini en aza indirir. Göz atılabilir bir düzenleyiciyle birleştiğinde, EiffelStudio 'zor bulunan' hataları bulmak ve düzeltmek için sisteminizin herhangi bir yerine gitmenizi sağlar ve böylece proje maliyetlerini daha da azaltır.

Mevcut yazılımımı bırakmak zorunda kalmadan Eiffel'e Geçiş yapabilir miyim?

Kesinlikle evet. Eiffel, çeşitli dillerde yazılmış yazılım bileşenlerini yeniden kullanmak için bir kombinasyon teknolojisi olarak en iyi şekilde kullanılabilen, açık bir sistemdir. Özellikle Eiffel, aşağıdakileri destekleyen sofistike bir C ve C ++ arayüzü içerir:

- Eiffel'den C işlevlerini çağırmak.

- C ++ sınıflarına ve tüm bileşenlerine (işlevler veya "yöntemler", veri üyeleri, yapıcılar, yıkıcılar vb.) Eiffel'den erişme.

- Cecil kütüphanesi (C-Eiffel Call-In Kütüphanesi) üzerinden C veya C ++ 'dan Eiffel mekanizmalarına erişim.

- Otomatik olarak bir C ++ sınıfından bir "sarıcı" Eiffel sınıfı üretiyoruz.

Eiffel, önceki uygulamaların en iyi sonuçlarını yeniden kullanırken modern yazılım teknolojisine geçmeyi mümkün kılar.

Eiffel'in çalışma zamanı performansı ne kadar hızlı?

HIZLI. Eiffel Software, çalışma zamanı performansından ödün vermeden modern nesne teknolojisinin tüm gücünü kullanmanın mümkün olduğunu göstermiştir. Çeşitli ölçütler, C ve Fortran'a benzer çalışma zamanı verimliliği gösterir ve çoğu durumda daha iyidir.

Eriyen Buz Teknolojisi Nedir?(Melting Ice Technology)

Bir değişiklikten sonra hızlı bir geri dönüş için, Eiffel Software'in derlemeyi birleştiren, optimum verimli kod üreten, bayt kodu yorumlamasıyla birleştiren benzersiz artımlı derleme teknolojisi.

Eiffelde Infix ve Prefix Notations

```
i.plus (j)
```

```
plus alias "+" (other: INTEGER):
```

```
INTEGER do
```

```
... end
```

```
i + j
```

`i.plus (j)` or `i + j Eiffel dilinde bu iki gösterimde doğrudur(toplama işlemi)`

Eiffelde Operatorler

Operator ::= Unary | Binary

Unary ::= `not` | "+" | "-" | Free_unary
Binary ::= "+" | "-" | "*" | "/" | "//" | "\\" | "^" | ".." | "<" | ">" | "<=" | ">="
| `and` | `or` | `xor` | `and then` | `or else` | `implies` | Free_binary

Eiffelde Grafikler nasıl işler?

Eiffel, çeşitli uygulama senaryoları için çeşitli grafik kitaplıkları sunar.

Taşınabilir projeler için, EiffelVision, kullanıcı arabirimi nesnelerini (pencereler, iletişim kutuları, menüler, düğmeler, iletişim kutuları vb.) Ve desteklenen tüm öğeler üzerinde çalışacak geometrik figürleri (çokgenler, daireler ve benzeri) kapsayan üst düzey bir grafik kütüphanesidir. platformları, her durumda yerel görünüm ve izlenime uyum sağlar.

Platforma özgü projeler için, belirli bir pencere sisteminde bulunan tüm “kontroller” veya “widget'lar” setinden yararlanmak için platforma özgü kütüphaneler mevcuttur:

Windows'da WEL (Windows Eiffel Kütüphanesi), en son kontroller de dahil olmak üzere esasen tüm Windows grafik API'lerine erişim sağlar. Ayrı bir sayfada, WEL'in Windows ve Eiffel'in avantajlarını nasıl birleştirdiği açıklanmaktadır. WEL tasarımının genel bir sunumunun yanı sıra çevrimiçi olarak kapsamlı bir WEL öğreticisi mevcuttur.

Unix, Linux ve VMS'de GTK Eiffel Kütüphanesi olan GEL'i kullanabilirisiniz.

EiffelVision'ı veya platforma özgü kitaplıklardan birini kullanmak özel bir öneri değildir: soyut yetenekleri için EiffelVision'ı ve örneğin belirli Windows denetimlerinden yararlanmak için WEL'i kullanarak iki düzeyi karıştırabilir ve eşleştirebilirsiniz. Aslında, EiffelVision dahili olarak, her platformda uygulanması için ilgili platforma özgü kütüphaneye güvenir, bu nedenle Windows'ta EiffelVision kullanıyorsanız zaten WEL'e sahipsiniz.

İlişkisel veritabanları nasıl işler?

Eiffel Software, nesne ilişkisel arayüzler için EiffelStore kütüphanesini sağlar, ODBC (Windows'ta düzinelerce veritabanı sistemine erişim sağlar), Oracle, Sybase ve Ingres için eşlemeler bulunur.Eiffel ayrıca Matisse, Versant ve O2 gibi nesneye yönelik veritabanlarıyla da arayüz oluşturmuştur.

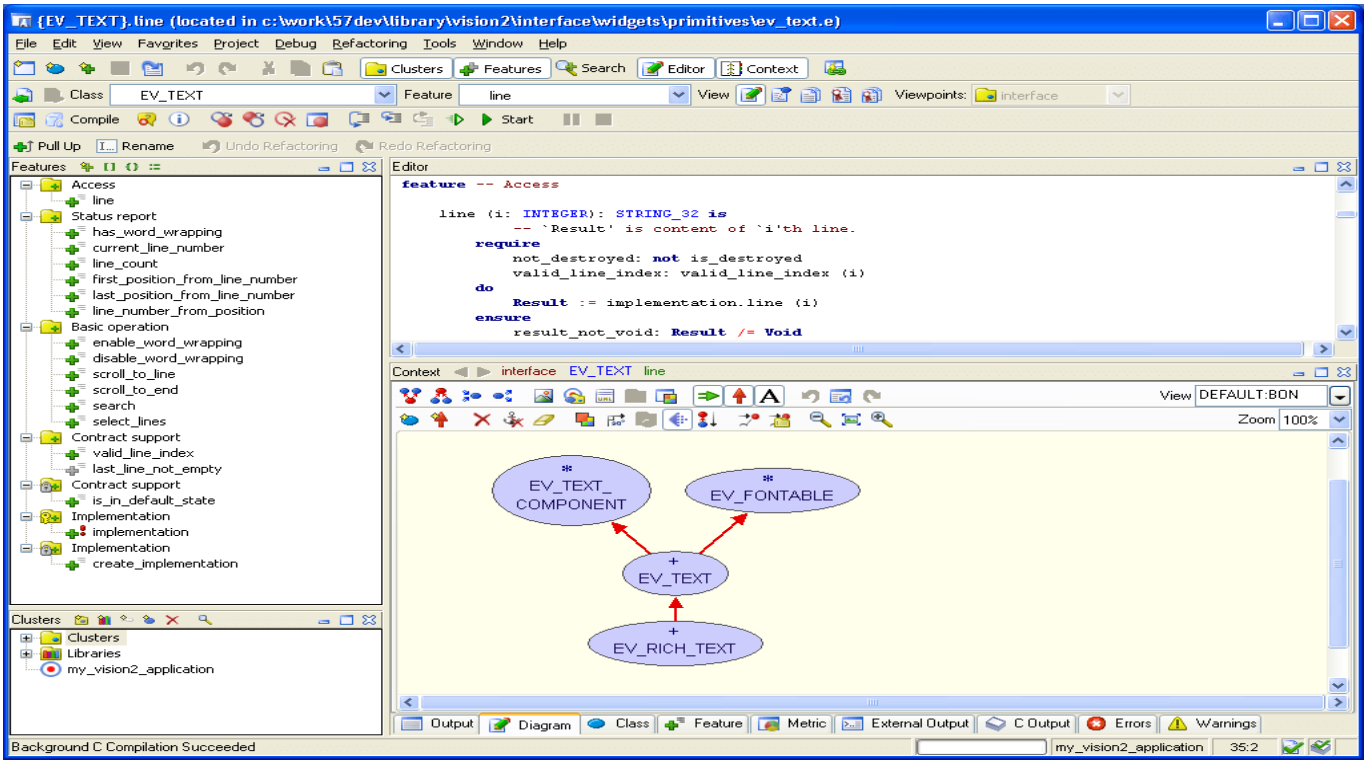
Eiffel programlama neye benziyor?

Aslında, her biri bir “veri soyutlaması” olan “sistemler” - bir sınıflar kümesi - dış dünyadan ya da uygulamadan belirli bir nesne kümesi hakkında konuşmak istiyoruz. Örneğin, uçuş kontrol sisteminde AIRPORT, RADAR ve RUNWAY sınıfları, bankacılık sisteminde MÜŞTERİ ve HESAP sınıfları olabilir. Herhangi bir sistemde LINKED_LIST ve HASH_TABLE gibi genel amaçlı sınıflara da sahip olabilirsiniz, ancak normalde bunları yazmaz, ancak EiffelBase gibi bir kütüphaneden tekrar kullanırsınız

IDE neye benziyor?

EiffelStudio beklediğiniz her şeyi ve daha fazlasını içeren tamamen grafiksel bir ortamdır: gidiş-dönüş (veya geri dönüşümlü) mühendislik, hızlı yeniden derleme, düzenleme, gelişmiş tarama olanakları, otomatik dokümantasyon ile analiz ve tasarım tezgahı (örneğin “kısa form ”), gelişmiş bir hata ayıklama mekanizması vb. sunar.

Eiffel Studiodan bir ekran görüntüsü:



Öznitellikler

Her değişken statik bir türü vardır. Nitelikler ve bağımsız değişkenler için tür, bildirimin bir parçasıdır. Sonuç türü, tanımlandığı işlevin imzasıyla ima edilir.

Bir değişken yalnızca kendi türündeki değerleri tutabilir. Çalışma zamanında ela_edcondses gibi bir tamsayı değişkenine bir dize değeri atarsanız, atama başarısız olur. Buna tip hatası denir: üretim yazılımında felakettir.

Eiffel, statik olarak yazılmış bir dildir. Derleyicinin bir tür arızanın meydana gelemeyeceğini doğrulaması her zaman mümkün olacak şekilde tasarlanmıştır.

Şimdiye kadar gördüğümüz varlıkların, INTEGER, REAL ve BOOLEAN türleri, karşılık gelen basit sayısal değerleri içerecek kadar büyüktür. Bu türlere genişletilmiş türler denir.

Giren özellik STRING türünde. Karakter dizelerinin uzunluğu değişkendir ve bir varlıkta tutulamayacak kadar karmaşıktır. Bu nedenle STRING nesneleri başka bir yerde saklanır ve varlık yalnızca nesnenin depolandığı yere bir başvuru içerir. STRING, başvuru türüne bir örnektir.

Eiffel'de Yeniden kullanılabilirlik nasıl oluyor?

Roland Racko'nun Yazılım Geliştirmede yazdığı gibi:

"Eiffel ile ilgili her şey, tek başına, açık bir şekilde, görkemli bir şekilde yeniden kullanılabilirliğe odaklanmıştır - doğrudan ayrılmış kelimeler ve noktalama işaretlerinin seçimine ve derleme zamanı ortamına kadar".

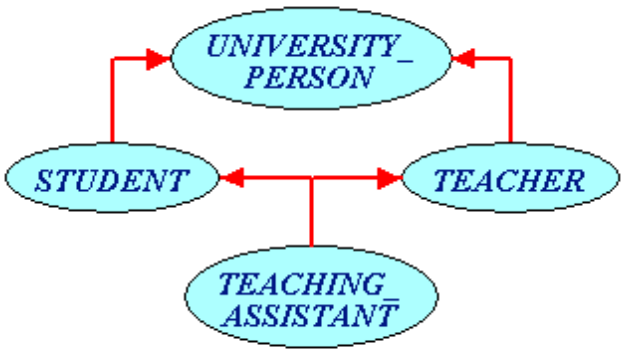
Daha iyi söyleyemezdik. Eiffel, ilk günden itibaren, her zaman tekerleği yeniden icat eden herkes yerine, yüksek kaliteli bileşenlerin yeniden kullanımına dayanan yeni yazılım endüstrisinin aracı olacak şekilde tasarlandı.

Bu da organizasyonlara sayısız saat ve binlerce dolar tasarruf geliştirme süresi kazandırır.

Eiffel, zengin bir dizi profesyonel yeniden kullanılabilir kitaplık seti (dikkatlice hazırlanmış binlerce bin) sağlayarak bu fikirleri hayata geçirmiştir: EiffelBase, EiffelVision, EiffelNet, EiffelWeb, EiffelParse, EiffelLex, WEL, MEL, PEL vb.

Eiffelde Tekrarlanan miras nedir(repeated inheritance)?

Tekrarlanan miras, bir sınıfın diğerinden iki veya daha fazla yoldan miras aldığı durumudur. Yalnızca Eiffel'de, ortak atanın her bir özelliği için, örneğin yinelenen torunda bir veya iki özellik verip vermediği gibi 'doğum tarihi' veya 'kütüphane privileleri' gibi ayrı ayrı karar verebilirsiniz. Şekilde, bir öğretim görevlisi, ister eğitmen olarak ister öğrenci olarak görüntüleniyorsa aynı doğum tarihine sahiptir (bir özellik), ancak bu kapasitelerin her biri altında farklı kütüphane ayrıcalıklarına sahiptir (iki özellik). Bu esneklik, neyi paylaşmak ve neyi çoğaltmak istediğinizi belirtmek için vazgeçilmezdir.



Eşzamanlı mühendislik nedir?

Eşzamanlı mühendislik, EiffelStudio'nun, sisteminizi tasarlarken genel tasarımını görmek ve etkileşimde bulunmak için otomatik olarak BON diyagramları oluşturmasına izin verir.

Eiffelde type ve binding kavramları:

Eiffel, hataların çalışma zamanında değil derleme zamanında yakalanmasını sağlamak için statik olarak yazılmıştır. Örneğin, sisteminiz yanlışlıkla üçgen şeklinde olan bir grafik nesnesinin köşegenini hesaplamak için bir istek yürütmeye çalışabilirse, Eiffel derleyicisi herhangi bir hasara yol açmaya başlamadan önce hatayı yakalar. Diğer nesne yönelimli dillerin çoğu, bu tür hataların derleyiciden çıkabileceği bir dereceye kadar “dinamik yazma” kullanır. (Özellikle statik olarak yazılmış olarak bildirilen ancak yine de neredeyse rastgele tür dönüşümlerine izin veren C uzantılarına dikkat edin.)

Eiffel, hedef nesneye bağlı olarak bir işlemin doğru sürümünün her zaman uygulanacağını garanti etmek için dinamik olarak bağlıdır. Örneğin, bir tür düzlemi temsil eden bir nesneye “kalkış” özelliğini uygularsanız, her biri kendi `` kalkış '' özelliğine sahip birkaç düzlem türü varsa, uygun olanın her zaman otomatik olarak seçileceği garantisine sahip olursunuz. (Aksine, bazı yaklaşımlar varsayılan olarak “statik bağlama” kullanır ve bu da felaketle yanlış davranışlara neden olabilir.)

BON nedir?

BON, Eiffel’deki yakın kavramlara (kesintisizlik, tersine çevrilebilirlik, sözleşme yapma) dayanan ve basit, sezgisel grafiksel konvansiyonları tanımlayan bir analiz ve tasarım yöntemi olan Business Object Notation'un kısaltmasıdır. BON, büyük ve karmaşık sistemleri tanımlamanız gerektiğinde ölçeklendirme kabiliyeti, özellikle de çeşitli soyutlama seviyelerindeki bileşenlerin ayrıntılarını yakınlaştırırken bütünün bir görünümünü koruduğu için dikkat çekicidir. EiffelStudio'nun Diyagram Aracı olan EiffelCase, BON'u destekler. Yöntem, çevrimiçi olarak sunulan Kim Walden ve Jean-Marc Nerson'un “Dikişsiz Nesneye Dayalı Yazılım Mimarisi” kitabında açıklanmaktadır.

Design by Contract(Programlama dil paradigması) mekanizması Eiffelde nasıl işler?

Sözleşme ile Tasarım kaliteli yazılım üretimini destekleyen eşsiz bir mekanizmadır. Kodunuzun, geliştirmenin "kurallarını" takip ettiği için önemli ölçüde daha az hata olmasını sağlar. Eiffel’de DBC(Design by Contract)sayesinde hata sayısını azalttığını, test maliyetlerini azalttığını ve pazara sunma süresini hızlandırdığı bir gerçektir.

Eiffel, C ++, C # ve Java gibi diğer dillerin yapamayacağını bana ne getirebilir?

Temiz açık sözdizimi, yeniden kullanılabilirlik, Sözleşmeye göre yerel Tasarım, tam yazılım yaşam döngüsü, taşınabilirlik, açıklık için destek… Kısacası, projeleri programa göre bitirebilme ve geliştirmenin tüm aşamalarında artan verimlilik.

Çoklu kalıtım nedir?

Çoklu kalıtım, Eiffel'de bir sınıf olarak bilinen bir yazılım biriminin diğer birçok sınıfın özelliklerini miras alabileceği bir mekanizmadır. Çoğu modern programlama dili, bir sınıfın yalnızca bir başka sınıftan miras alabileceği tek mirası destekler, ancak çoklu miras, geliştiriciye istendiği kadar miras alma yeteneği sayesinde sınırsız kalıtım sunar. Birden fazla miras kullanmak aşağıdaki avantajları sağlar: gelişmiş yeniden kullanım, daha iyi genel sistem tasarımı ve mimarisi, daha fazla esneklik, daha kolay bakım ve hata ayıklama sağlar.

Jeneriklik nedir?

Jeneriklik, yazılım metninde tip parametreli sınıf modülleri için destektir. Nesneye yönelik çevrelerde bunlar genel sınıflar olarak bilinir. Bu tür sınıflar yazılım metninde genel parametreler kullanırlar; bunlar daha sonra sınıf aslında bir istemci sınıfı tarafından kullanıldığında resmi parametreler ile değiştirilir. Böyle bir yöntemin faydaları, diziler ve listeler gibi kapsayıcı nesnelerini düşündüğünüzde sınıflarda tam olarak gerçekleşir. Bu türler bir dizi keyfi öğeyi tutacak şekilde tasarlanmıştır. İdeal olarak, içerilen bu elemanlar, kitaplardan müşterilere, sisteminizde bulunan herhangi bir türe kadar her türlü eleman olabilmelidir. Aksi takdirde, saklamak istediğiniz her öğe türü için ayrı bir sınıf tanımı, acı verici ve zaman alıcı bir etkinlik yazmanız gerekir. Jeneriklik, çalışma zamanında gerçek tür ile değiştirilecek genel bir tür olduğunu varsayarak bu sorunu çözer.

Eiffel .NET?

Evet. .NET için Eiffel, klasik Eiffel ile tamamen aynı şekilde kullanılabilir ve uygulama üretmek için ekstra dil yapısına gerek yoktur.

Eiffel, .NET'e birkaç benzersiz ve güçlü özellik getiriyor. En belirgin şekilde, .NET Framework'ün ilk sürümünden bu yana birden fazla miras ve jeneriklik sunan tek .NET dilidir. Bu iki mekanizma, tamamen yeniden kullanılabilir bir yazılım oluşturmak için gerçekten nesne yönelimli bir vazgeçilmez yardımcıdır.

.NET için Eiffel ayrıca yazılım metninin doğruluğunu, güvenilirliğini ve sağlamlığını sağlayan Design By Contract'ın tüm avantajlarını da beraberinde getirir. Diğer bazı .NET dilleri sözleşme mekanizmalarını desteklerken, .NET için Eiffel gerçek bir dil yapısı olarak yerel olarak destekleyen tek dildir.

Eiffel ayrıca varolan bazı kitaplıkları .NET'e getirir, böylece Windows gibi diğer .NET kitaplıklarını kullanabilmenin yanı sıra grafik öğeler için Eiffel'in WEL veya EiffelVision2 kitaplıklarını kullanabilirsiniz. EiffelVision2, çok platformlu olduğu için özellikle benzersiz bir grafik kütüphanesidir, böylece derlenen sistemin desteklenen tüm platformlarda aynı ekranı ve davranışı üretmesini sağlar.

.NET için Eiffel yeniden kullanılabilir bileşenler nasıl üretir?

.NET için Eiffel, yalnızca Ortak Aracı Dil (CIL) kodunu derleyerek yeniden kullanılabilir bileşenler üretir ve bu da .NET destekli başka herhangi bir dil tarafından kullanılmasını sağlar. Ancak, bu yeniden kullanım yöntemi, yalnızca bir tür yeniden kullanım eylemidir, yani yazılım bileşenlerinin yeniden kullanımıdır. Potansiyel yeniden kullanılabilirliğin başka birçok seviyesi ve alanı vardır.

Eiffel derleyici, daha sonra başka bir Eiffel projesinden yeniden kullanılabilen 'önceden derlenmiş kütüphaneler' de üretebilir. Bir derleme referans vermek için önceden derlenmiş bir kitaplık kullanmanın avantajları arasında, yerel Eiffel yapılarını (çoklu kalıtım ve jeneriklik gibi) tüm kaynağı yeniden derlemek zorunda kalmadan yeniden kullanma yeteneği bulunur.

Eiffel dili, en başından beri yeniden kullanım seviyeleri göz önünde bulundurularak geliştirildi ve bunlar geliştiriciye birçok açıdan yarar sağlayabilir. Birden fazla kalıtım ve jeneriklik, gerçek yazılım uygulaması içinde yeniden kullanım sağlar, aynı kodun farklı sınıflarda tekrarlanması gereğini ortadan kaldırır - eğer bir sınıfın sağladığı işlevselliğe ihtiyacınız varsa, o zaman bir istemci olun veya miras alın. Aynı şekilde, kodunuzun büyük kısmı sadece küçük varyasyonlarla benzer şeyler yapıyorsa, onu bir sınıfa veya kütüphaneye soyutlayabilirsiniz. Bu uygulama kullanılır ve EiffelStudio veya Eiffel ENViSioN! İle sağlanan yeniden kullanılabilir kütüphanelerde görülebilir.

VB.NET ve C # birkaç sınıf var. Onları Eiffel'de yeniden yazmam gerekir mi?

Hayır. VB.NET, C # veya .NET için Eiffel gibi .NET destekli bir dilde yazılmış kodlar otomatik olarak Ortak Aracı Dil (CIL) kodunu derler. Bu standart aracılığıyla, tamamen birlikte çalışabilen bağımsız modüller (.dll'ler veya .exe'ler) oluşturulur, böylece bir geliştirici kodu yeniden yazmak veya başka bir dile aktarmak zorunda kalmadan herhangi bir kodu yeniden kullanabilir.

Eiffel Hangi UML araçlarını destekler?

EiffelStudio, herhangi bir Eiffel sistemi için XMI (XML Meta Veri Değişimi) bilgisi üretebilir. XMI notasyonu, bu standardı destekleyen herhangi bir ürün (örneğin, IBM'in Rational Rose) arasında sistem bilgisi alışverişini mümkün kılar.

Eiffel'de nasıl web yazarız?

.NET için Eiffel kullanarak bir web hizmeti oluşturmak, diğer herhangi bir .NET destekli dilde bir tane oluşturmak kadar basittir. Internet'te göstermek istediğiniz yöntemleri içeren sınıf, System.Web.Services.WebService .NET türünden devralınabilir. Sınıf özelliklerini Internet'te göstermek için, onlara web yöntemleri olarak bildiren özel öznitelikler verilmelidir.

Eiffel Atik Programlama(Agile Programming) için iyi midir?

Modern yazılım projeleri atikliğe ihtiyaç duyar: piyasa taleplerine hızlı tepki verme, başarılı sistemleri aşamalı olarak geliştirme ve ekipleri güçlendirme yeteneği. Eiffel mükemmel bir çözümdür: dil ve çevre, kısa iterasyonlar yoluyla düzenli sürüm üretme teknolojisi sağlar; yöntem, değiştirilmesi kolay mimarileri, değişimin hoş karşılanması için çevik bir hayali yerine getirir; ve tüm Eiffel araçları, ekip üyelerinin yaratıcılıklarından en iyi şekilde faydalanmalarına ve eski teknolojilerin engellerinden kurtulmalarına odaklanmıştır.

Çöp toplama(Garbage collection)

Eiffel, artık herhangi bir yere başvurulmadığında nesneleri otomatik olarak bellekten kaldıran bir çöp toplayıcı kullanır. Bu nedenle, örneğin C veya C ++ 'dan farklı olarak, belleği yönetmek gerekli değildir.

HELLO WORLD! ÖRNEĞİ:

```
class
HELLO_WORLD
Create
    say_it
feature
    say_it    do
              Io.put_string ("Hello World!")

              end

end
Hello World!
```

Hafıza yönetimi(Memory Management)

Nesneler oluşturma talimatları ile oluşturulur, ancak nasıl yok edildikleri aşağıdaki gibidir:

Açık çözüm, programcının onları yok etmesi için bir araç sağlamaktır. Bu, dil uygulayıcısı için kolaydır, ancak uygulama programcısı için ciddi zorluklara yol açar.

Sistemin bir bölümünde oluşturulan nesneler başka bir yerde kullanılmak üzere hedeflenebilir. Bir nesne oluşturmaktan sorumlu olan kod, artık gerekli olmadığında farkında olmayabilir. Sistemin bir kısmı bir nesneyle bittiğinde, bunun başka bir yerde kullanılmadığından emin olamaz.

Nesneler, onları kullanan uygulama tarafından atıldıktan sonra bile birbirlerine karşılıklı referanslar içerebilir.

Bu hususlar bellek yönetimini programcı için çok ağır bir yük haline getirir ve Eiffel, programcının nesnelerin ölmesine neden olabileceği 'sil' veya 'serbest' ifadesi sunmaz. Bunun yerine, Eiffel çalışma zamanı sistemi nesnelere yapılan başvuruları otomatik olarak izler ve güvenli olduğunda alanı geri kazanır. Bu işleme 'çöp toplama' denir.

Artık ihtiyacınız olmayan referansları onlara Boşluk atayarak açıkça atayarak çöp toplayıcısına yardımcı olabilirsiniz, ancak çoğu başvuru (nesneler değil) rutin çıkışta yine de yok olur.

Çöp toplayıcının dikkatlice kontrol edilen kesintisinin bile tolere edilemediği durumlar vardır ve bu tür uygulamalar için temel kütüphanelerdeki sınıflar, onu kontrol etmenizi sağlayan özellikler içerir.

KAYNAKÇA

https://www.eiffel.org/doc/eiffel/ET-_The_Dynamic_Structure-_Execution_Model

https://eiffel-guide.com/

https://www.maths.tcd.ie/~odunlain/eiffel/eiffel_course/eforb.htm

https://www.eiffel.org/

https://en.wikipedia.org/wiki/Eiffel_(programming_language)

https://www.eiffel.org/doc/eiffel/Eiffel_programming_language_syntax

HAZIRLAYAN ÖĞRENCİ: 18253007 - BERK BARIŞ KARA (Eiffel Programming langueage)

EIFFEL DİLİNDE IS_PALINDROME METHODU:

```
1  is_palindrome (a_string: STRING): BOOLEAN
2                          -- Is `a_string' a palindrome?
3                  require
4                          string_attached: a_string /= Void
5                  local
6                          l_index, l_count: INTEGER
7                  do
8                          from
9                                  Result := True
10                                 l_index := 1
11                                 l_count := a_string.count
12                         until
13                                 l_index >= l_count - l_index + 1 or not Result
14                         loop
15                                 Result := (Result and a_string [l_index] = a_string [l_count - l_index + 1])
16                                 l_index := l_index + 1
17                         end
18                 end
19
```

EIFFEL DİLİNDE PERFECT_NUMBER METHODU:

```
class
	APPLICATION

create
	make

feature

	make
		do
			io.put_string ("  6 is perfect...%T")
			io.put_boolean (is_perfect_number (6))
			io.new_line
			io.put_string (" 77 is perfect...%T")
			io.put_boolean (is_perfect_number (77))
			io.new_line
			io.put_string ("496 is perfect...%T")
			io.put_boolean (is_perfect_number (496))
		end

	is_perfect_number (n: INTEGER): BOOLEAN
			-- Is 'n' a perfect number?
		require
			n_positive: n > 0
		local
			sum: INTEGER
		do
			across
				1 |..| (n - 1) as c
			loop
				if n \\ c.item = 0 then
					sum := sum + c.item
				end
			end
			Result := sum = n
		end

end
```

EİFFEL DİLİNDE FİBONACCİ METHODU:

```
class
        APPLICATION

create
        make

feature

        fibonacci (n: INTEGER): INTEGER
                require
                        non_negative: n >= 0
                local
                        i, n2, n1, tmp: INTEGER
                do
                        n2 := 0
                        n1 := 1
                        from
                                i := 1
                        until
                                i >= n
                        loop
                                tmp := n1
                                n1 := n2 + n1
                                n2 := tmp
                                i := i + 1
                        end
                        Result := n1
                        if n = 0 then
                                Result := 0
                        end
```

DEVAMI AŞAĞIDAKİ TAKİP EDEN SAYFADADIR!!

```
31                  end
32
33  feature {NONE} -- Initialization
34
35          make
36                          -- Run application.
37                  do
38                          print (fibonacci (0))
39                          print (" ")
40                          print (fibonacci (1))
41                          print (" ")
42                          print (fibonacci (2))
43                          print (" ")
44                          print (fibonacci (3))
45                          print (" ")
46                          print (fibonacci (4))
47                          print ("%N")
48                  end
49
50  end
```

EİFFEL DİLİNDE SEZAR ŞİFRELEME:

```
class
	APPLICATION

inherit
	ARGUMENTS

create
	make

feature {NONE} -- Initialization

	make
			-- Run application.
		local
			s: STRING_32
		do
			s := "The tiny tiger totally taunted the tall Till."
			print ("%NString to encode: " + s)
			print ("%NEncoded string: " + encode (s, 12))
			print ("%NDecoded string (after encoding and decoding): " + decode (encode (s, 12), 12))
		end

feature		-- Basic operations

	decode (to_be_decoded: STRING_32; offset: INTEGER): STRING_32
				-- Decode `to be decoded' according to `offset'.
		do
			Result := encode (to_be_decoded, 26 - offset)
		end

	encode (to_be_encoded: STRING_32; offset: INTEGER): STRING_32
				-- Encode `to be encoded' according to `offset'.
		local
			l_offset: INTEGER
			l_char_code: INTEGER
```

DEVAMI TAKİP EDEN DİĞER SAYFADADIR!!

```
36                  do
37                          create Result.make_empty
38                          l_offset := (offset \\ 26) + 26
39                          across to_be_encoded as tbe loop
40                                  if tbe.item.is_alpha then
41                                          if tbe.item.is_upper then
42                                                  l_char_code := ('A').code + (tbe.item.code - ('A').code + l_offset) \\ 26
43                                                  Result.append_character (l_char_code.to_character_32)
44                                          else
45                                                  l_char_code := ('a').code + (tbe.item.code - ('a').code + l_offset) \\ 26
46                                                  Result.append_character (l_char_code.to_character_32)
47                                          end
48                                  else
49                                          Result.append_character (tbe.item)
50                                  end
51                          end
52                  end
53  end
```

Fibonacci-anagram-asal sayı-mükemmel sayı-sezar şifreleme-palindrom kodlarının detaylı şekli aşağıdaki linktedir:

https://github.com/bariss48/Eiffel-programming-language