BMB214 Programlama Dilleri Prensipleri

Ders 14. İstisna İşleme ve Olay İşleme (Exception Handling and Event Handling)

Erdinç Uzun
Tekirdağ Namık Kemal Üniversitesi
Bilgisayar Mühendisliği Bölümü

Konular

- İstisna İşlemeye Giriş
- C++'da Özel Durum İşleme
- Java'da Özel Durum İşleme
- Python ve Ruby'de Özel Durum İşleme
- Olay İşlemeye Giriş
- Java ile Olay İşleme
- C#'ta Olay İşleme

İstisna İşlemeye (Exception Handling) Giriş

- İstisna İşlemesi Olmayan Bir Dilde
  - Bir istisna meydana geldiğinde, kontrol, bir mesajın görüntülendiği ve programın sonlandırıldığı işletim sistemine bildirilir.
- İstisna İşleme Olan Bir Dilde
  - Programların bazı istisnaları yakalamasına izin verilir, böylece sorunu çözme ve devam etme imkanı sağlanır.

Temel Kavramlar

- Birçok dil, programların giriş/çıkış (I/O) hatalarını yakalamasına izin verir (EOF dahil)
- Bir istisna (exception), donanım veya yazılım tarafından tespit edilebilen ve özel işlem gerektirebilecek herhangi bir olağandışı olaydır.
- Bir istisna tespit edildikten sonra gerekebilecek özel işleme, istisna işleme (exception handling) denir
- İstisna işleme kod birimi, istisna işleyici (exception handler) olarak adlandırılır

İstisna İşleme Alternatifleri

- İlişkili olay gerçekleştiğinde bir istisna ortaya çıkar
- İstisna işleme yeteneklerine sahip olmayan bir dil yine de istisnaları tanımlayabilir, tespit edilebilir, tetiklenebilir ve işleyebilir (kullanıcı tanımlı, yazılım tespitli)
- Alternatifler:
  - Yardımcı bir parametre (auxiliary parameter) gönderin veya bir alt programın dönüş durumunu belirtmek için dönüş değerini kullanın
  - Tüm alt programlara bir etiket parametresi (label parameter) geçirin (hata dönüşü geçilen etikete aittir)
  - Tüm alt programlara bir istisna işleme alt programı geçirin

Yerleşik İstisna İşlemenin (Built-in Exception Handling) Avantajları

- Hata tespit (error detection) kodu yazmak zahmetlidir ve programı karıştırır
- İstisna işleme, programcıları birçok farklı olası hatayı dikkate almaya teşvik eder
- İstisna yayma, istisna işleme kodunun yüksek düzeyde yeniden kullanımına izin verir

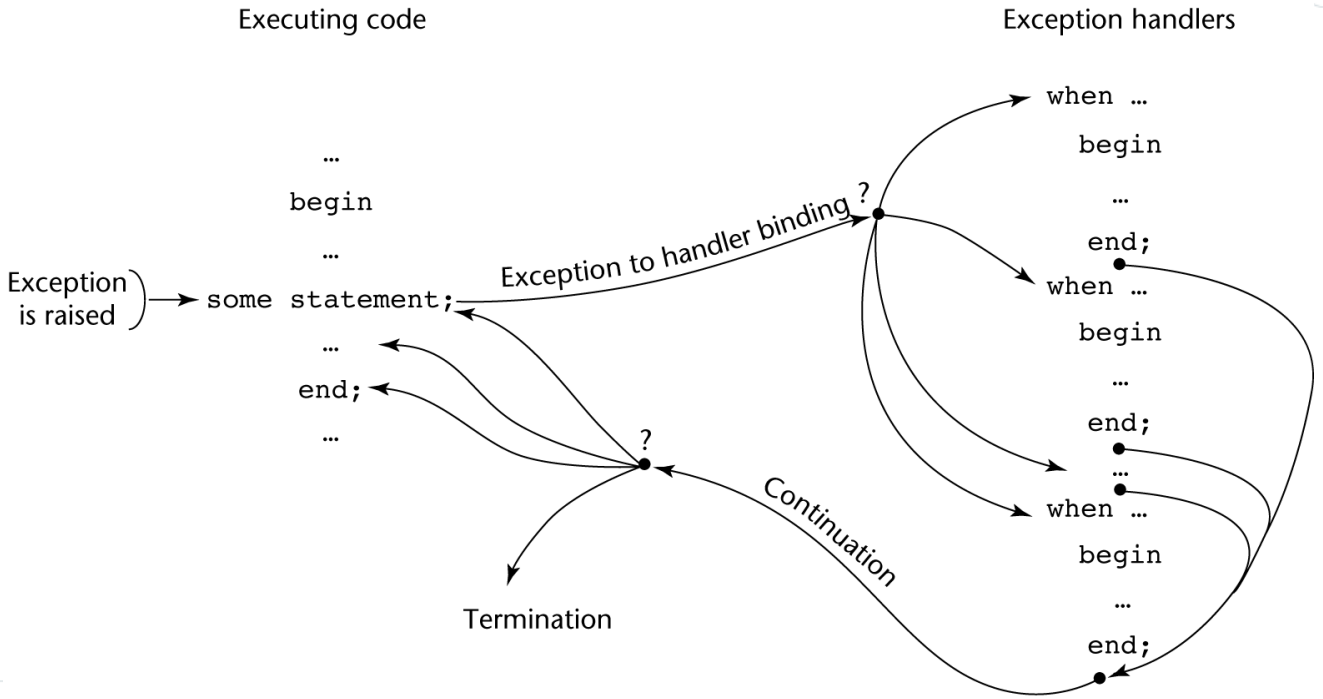
Tasarım Zorlukları

- İstisna işleyiciler (exception handlers) nasıl ve nerede belirtilir ve kapsamları (scope) nedir?
- Bir istisna oluşumu bir istisna işleyiciye nasıl bağlanır?
- İşleyiciye istisna hakkında bilgi aktarılabilir mi?
- Bir istisna işleyicisi yürütmeyi tamamladıktan sonra, eğer varsa, yürütme nerede devam eder? (devam - devam ettirme)
- Bir çeşit sonuçlandırma sağlandı mı?

Tasarım Zorlukları...

- Kullanıcı tanımlı istisnalar nasıl belirtilir?
- Kendilerininkini sağlamayan programlar için varsayılan istisna işleyicileri olmalı mı?
- Önceden tanımlanmış istisnalar açıkça tetiklenebilir mi?
- Donanım tarafından algılanabilen hatalar, ele alınabilecek istisnalar olarak değerlendiriliyor mu?
- Önceden tanımlanmış istisnalar var mı?
- Varsa istisnalar nasıl devre dışı bırakılabilir?

İstisna İşleme Kontrol Akışı



İstisna işleme
C++

- 1990'da C++'ya eklendi
- Tasarım CLU, Ada ve ML tasarımına dayanmaktadır

İstisna İşleyiciler (Exception Handlers)
C++

```
try {
-- bir istisna oluşturması beklenen kod
}
catch (formal parameter) {
-- işleyici kodu
}
...
catch (formal parameter) {
-- handler code
}
```

Catch fonksiyonu

- catch, tüm işleyicilerin adıdır - aşırı yüklenmiş (overloaded name) bir addır, bu nedenle her birinin biçimsel parametresi formal parameter) benzersiz olmalıdır
- Biçimsel parametrenin bir değişkeni olması gerekmez
  - İçinde bulunduğu işleyiciyi diğerlerinden ayırmak için basitçe bir tür adı olabilir
- Biçimsel parametre, bilgileri işleyiciye aktarmak için kullanılabilir

İstisna Oluşturma

- İstisnaların tümü aşağıdaki ifadeyle açıkça ortaya konmaktadır:

  throw [expression];

- Parantezler metasemboldür
- İşlenen olmayan bir throw yalnızca bir işleyicide görünebilir; göründüğünde, istisnayı yeniden tetiklenir ve daha sonra başka bir yerde ele alınır.
- İfadenin türü, amaçlanan işleyicinin belirsizliğini ortadan kaldırır

İşlenmeyen İstisnalar (Unhandled Exceptions)

- İşlenmeyen bir istisna, çağrıldığı fonksiyonu arayana (caller) iletilir
- Bu yayılma ana fonksiyonu devam ediyor
- İşleyici bulunmazsa, varsayılan (default) işleyici çağrılır

Devam ettirme (Continuation)

- Bir işleyici yürütmeyi tamamladıktan sonra, kontrol, bir öğesi olduğu işleyiciler sırasındaki son işleyiciden sonraki ilk statement'a akar.
- Diğer tasarım seçenekleri
  - Tüm istisnalar kullanıcı tanımlıdır
  - İstisnalar ne belirtilmiş ne de beyan edilmiş
  - Varsayılan işleyici, unexpected (beklenmedik) şekilde programı sonlandırır; unexpected kullanıcı tarafından yeniden tanımlanabilir
  - Fonksiyonlar, tetiklenen istisnaları listeleyebilir
  - Spesifikasyon olmadan, bir fonksiyon herhangi bir istisna yaratabilir (throw clause)

Değerlendirme

- Önceden tanımlanmış (predefined exceptions) istisna yoktur
- İstisnaların adlandırılmaması ve donanım ve sistem yazılımı tarafından algılanabilen istisnaların ele alınamaması gariptir
- İşleyicilere istisnaları parametre türü aracılığıyla bağlamak kesinlikle okunabilirliği desteklemez

İstisna İşleme
Java

- C++'ya dayalıdır, ancak daha çok OOP felsefesiyle uyumludur
- Tüm istisnalar, Throwable sınıfının torunları olan sınıf nesneleridir.

Expection Sınıfları
Java

- Java kitaplığı Throwable'ın iki alt sınıfını içerir:
  - Error
    - Heap taşması (overflow) gibi olaylar için Java yorumlayıcısı tarafından atılır
    - Kullanıcı programları tarafından asla ele alınmaz
  - Exception
    - Kullanıcı tanımlı istisnalar genellikle bunun alt sınıflarıdır
    - Önceden tanımlanmış iki alt sınıfa sahiptir
      - IOException ve RuntimeException (örneğin, ArrayIndexOutOfBoundsException ve NullPointerException)

İstisna İşleyiciler (Exception Handlers) Java

- C++ 'da olduğu gibi, her catch için adlandırılmış bir parametre gerektirmesi ve tüm parametrelerin Throwable'ın soyundan olması gerekir.
- try clause sözdizimi tam olarak C++ ile aynıdır
- İstisnalar, C++ 'da olduğu gibi, throw ile tetiklenir, ancak genellikle aşağıdaki gibi, throw, nesneyi oluşturmak için new operatörü içerir

```
throw new MyException ();
```

İstisnaları İşleyicilere Bağlama (Binding)
Java

- Bir istisnayı bir işleyiciye bağlamak Java'da C++'da olduğundan daha basittir
  - İlk işleyiciye bir parametreyle bağlı bir istisna, thrown nesneyle veya onun atasıyla aynı sınıftır.
- İşleyiciye bir throw dahil edilerek bir istisna ele alınabilir ve yeniden tetiklenebilir (bir işleyici ayrıca farklı bir istisna atabilir)

Devam ettirme (Continuation)
Java

- Try yapısında işleyici bulunamazsa, arama en yakın çevreleyen try yapısında vb. devam eder.
- Metotta işleyici bulunmazsa, istisna yöntemin çağırıcısına yayılır.
- İşleyici bulunmazsa (ana yolun sonuna kadar), program sonlandırılır
- Tüm istisnaların yakalanmasını sağlamak için, tüm istisnaları yakalayan herhangi bir try yapısına bir işleyici dahil edilebilir.
  - Sadece bir Exception sınıfı parametresi kullanın
  - Tabii ki, try yapısında son olmalı

Kontrol edilmiş ve Kontrol Edilmemiş İstisnalar Java

- Java throws clause, C++'ın throw clause'ndan oldukça farklıdır
- Error ve RunTimeException sınıfının özel durumları ve bunların tüm alt öğeleri denetlenmemiş istisnalar (unchecked exceptions) olarak adlandırılır; diğer tüm istisnalara kontrol edilmiş istisnalar (checked exceptions) denir
- Bir metot tarafından atılabilecek kontrol edilmiş istisnalar şunlardan biri olmalıdır:
  - throw clause listelenir veya
  - Metotta ele alındı

Diğer Tasarım Seçenekleri
Java

- Bir metot, kendi throws clause geçersiz kıldığı metottan daha fazla istisna bildiremez
- Throws clause belirli bir kontrol edilmiş istisnayı listeleyen bir metodu çağıran bir metot, bu istisnayı ele almak için üç alternatife sahiptir:
  - İstisnayı yakalayın ve işleyin
  - İstisnayı yakalayın ve kendi throws clause listelenen bir istisna atın
  - Onu throws clause beyan edin ve onu ele almayın

finally Clause
Java

- Bir try yapısının sonunda görünebilir
- Form:

```
finally{
...
}
```

- Amaç: try yapısında ne olduğuna bakılmaksızın yürütülecek kodu belirtmek

Örnek
Java

```
try {
   for (index = 0; index < 100; index++)
{
          ...
          if (...) {
                return;
          } //** end of if
} //** end of try clause
finally {
   ...
} //** end of try construct
```

Assertions

- Hesaplamanın mevcut durumuna ilişkin bir mantıksal expression bildiren programdaki statement'lardır.
- Doğru (true) olarak değerlendirildiğinde hiçbir şey olmaz
- Yanlış (false) olarak değerlendirildiğinde bir AssertionError istisnası atılır
- Program değişikliği veya yeniden derleme olmadan çalışma sırasında (runtime) devre dışı bırakılabilir
- İki form
  - `assert` *condition*`;`
  - `assert` *condition*`:` *expression*`;`

Değerlendirme

- İstisna türleri, C++ durumunda olduğundan daha mantıklıdır
- throw clause C++ 'dan daha iyidir (C ++' daki throw clause programcıya çok az şey söyler)
- Finally clause genellikle kullanışlıdır
- Java yorumlayıcısı, kullanıcı programları tarafından ele alınabilecek çeşitli istisnalar tetikler.

İstisna İşleme
Python

- İstisnalar nesnelerdir; temel sınıf BaseException'dır
- Önceden tanımlanmış ve kullanıcı tanımlı tüm istisnalar, Exception'dan türetilmiştir.
- Exception'ın önceden tanımlanmış alt sınıfları, ArithmeticError (alt sınıflar OverflowError, ZeroDivisionError ve FloatingPointError'dır) ve LookupError'dır (alt sınıflar IndexError ve KeyError'dır)

İstisna İşleme...
Python

```
try:
   - The try block
except Exception1:
   - Handler for Exception1
except Exception2:
   - Handler for Exception2
...
else:
   - The else block (no exception is raised)
finally:
   - the finally block (do it no matter what)
```

İstisna İşleme...
Python

- İşleyiciler, adlandırılmış istisnayı ve bu istisnanın tüm alt sınıflarını ele alır, bu nedenle adlandırılmış istisna bir Exception ise, tüm önceden tanımlanmış ve kullanıcı tanımlı istisnaları işler.
- İşlenmeyen istisnalar en yakın çevreleyen try bloğuna yayılır; işleyici bulunmazsa, varsayılan işleyici çağrılır
  - Raise IndexError bir örnek oluşturur
- Yükseltilmiş istisna nesnesi şu şekilde elde edilebilir:
  - Exception as ex_obj:

İstisna İşleme...
Python

- Assert statement, Boole ifadesini (ilk parametre) test eder ve ikinci parametresini, oluşturulacak istisna nesnesi için yapıcıya (constructor) gönderir.

```
assert test, data
```

İstisna İşleme...
Ruby

- İstisnalar nesnelerdir
- Önceden tanımlanmış birçok istisna vardır
- Kullanıcı tarafından işlenen tüm istisnalar ya StandardError sınıfı ya da onun bir alt sınıfıdır.
- StandardError, mesaj ve geri izleme olmak üzere iki yöntemi olan Exception'dan türetilmiştir.
- Genellikle bir biçime sahip olan raise ile istisnalar gündeme getirilebilir:

```
raise "bad parameter" if count == 0
```

İstisna İşleme...
Ruby

- İşleyiciler, bir başlangıç-bitiş kod bloğunun sonuna yerleştirilir; rescue tarafından tanıtıldı

```
begin
- Statements in the block
rescue
- Handler
end
```

- Blok, else ve finally Java'da benzer şekilde else ve / veya `ensure` clauses içerebilir
- Tartıştığımız diğer dillerden farklı olarak, Ruby'de bir istisna oluşturan kod, işleyicinin sonuna bir yeniden retry statement'ına yerleştirilerek yeniden çalıştırılabilir.

Olay İşleme (Event Handling)

- Bir olay (event), bir grafik uygulamasında fareyle tıklanması gibi belirli bir şeyin gerçekleştiğine dair bir bildirimdir.
- Olay işleyici (event handler), bir olaya yanıt olarak yürütülen bir kod segmentidir.

Java Swing GUI Bileşenleri

- Metin kutusu, JTextField sınıfının bir nesnesidir
- Radyo düğmesi, JRadioButton sınıfının bir nesnesidir
- Applet'in ekranı bir çerçevedir, çok katmanlı bir yapıdır
- İçerik bölmesi (Content pane), uygulamaların çıktı (output) koyduğu bir katmandır
- GUI bileşenleri bir çerçeveye yerleştirilebilir
- Düzen yöneticisi nesneleri (Layout manager objects), bileşenlerin yerleşimini kontrol etmek için kullanılır

Java Olay Modeli

- GUI bileşenleriyle kullanıcı etkileşimleri, olay dinleyicileri (event listeners) adı verilen olay işleyicileri tarafından yakalanabilen olaylar oluşturur
- Bir olay üreteci (event generator), bir mesaj göndererek bir olayı dinleyiciye söyler
- Olay işleme yöntemlerini standart bir protokole uygun hale getirmek için bir arayüz (interface) kullanılır
- Bir dinleyici uygulayan bir sınıf, dinleyici için bir arabirim uygulamalıdır

Java Olay Modeli...

- Bir olay sınıfı, bir onay kutusunu, radyo düğmesini veya bir liste öğesini tıklama olayı ile ilişkilendirilen ItemEvent'tir.
- ItemListener arabirimi, ItemEvent olayları için bir işleyici olan itemStateChanged yöntemini tanımlar.
- Dinleyici, addItemListener ile oluşturulur.

Olay İşleme
C#

- C# (ve diğer .NET dillerinde) olay işleme Java'dakine benzer
- .NET'in iki yaklaşımı vardır:
  - Windows Forms ve Windows Presentation Foundation - yalnızca ilkini ele alıyoruz (orijinal yaklaşım budur)
- Bir uygulama, önceden tanımlanmış Form sınıfını alt sınıflar (System.Windows.Forms'da tanımlanmıştır)
- GUI bileşenlerinin yerleştirileceği bir çerçeve veya panel oluşturmaya gerek yoktur.
- Pencereye metin yerleştirmek için label nesneleri kullanılır
- Radyo düğmeleri RadioButton sınıfının nesneleridir

Olay İşleme...
C#

- Bileşenler, bileşenin Location özelliğine yeni bir Point nesnesi atanarak konumlandırılır.

```
private RadioButton plain = new RadioButton();
plain.Location = new Point(100, 300);
plain.Text = "Plain";
controls.Add(plain);
```

- Tüm C# olay işleyicileri aynı protokole sahiptir, dönüş türü void'tir ve iki parametre de object ve EventArgs türlerindedir.

Olay İşleme...
C#

- Bir olay işleyicinin herhangi bir adı olabilir
- Bir radyo düğmesi, düğmenin Boolean Checked özelliğiyle test edilir

```
private void rb_CheckedChanged (object o,
                                EventArgs e) {
  if (plain.Checked) …
  ...
}
```

- Bir olayı kaydetmek için, yeni bir EventHandler nesnesi oluşturulmalı ve olay için önceden tanımlanmış delegeye (delegate) eklenmelidir.

Olay İşleme...
C#

- Bir radyo düğmesi işaretlenmemiş durumdan işaretli duruma geçtiğinde, CheckedChanged olayı başlatılır
- İlişkili delege (delegate) olayın adıyla başvurulur
- İşleyicinin adı rb_CheckedChanged ise, onu Plain adlı radyo düğmesine şu şekilde kaydedebiliriz:

```
plain.CheckedChanged +=
    new EventHandler (rb_CheckedChanged);
```