selsoft
build better, deliver faster

Temiz Kod (Clean Code)

5. Bölüm - Savunmacı Programlama

Eğitmen:

Akın Kaldıroğlu

Çevik Yazılım Geliştirme ve Java Uzmanı

Konular

- **Savunmacı Programlamaya Giriş**
- **Assertions**
  - Dillerde Assertion Kullanımı
- **Sıra Dışı Durum Yönetimi**
  - Geleneksel Sıra Dışı Durum Yönetimi
- **Savunmacı Programlama Teknikleri**

Savunmacı Programlamaya Giriş

Savunmacı Sürüş - I

- **Savunmacı programlama** (**defensive programming**) ismini **savunmacı sürüş**ten (**defensive driving**) alır.
- **Savunmacı sürüş**: Driving to save lives, time, and money, in spite of the conditions around you and the actions of others.
- Savunmacı sürüş, sürücülere, trafikte başkalarının hatalarını tolere edecek ve hiçbir zarar vermeyecek şekilde teyakkuzda olmalarını ve buna göre davranmaları öğütler.

Savunmacı Sürüş - II

- Çünkü trafikte her şey olabilir, herhangi bir sürücü hiç umulmadık bir hareket yapabilir, riskli bir davranışta bulunabilir.
- Dolayısıyla sürücülere düşen, etrafındaki diğer sürücülerin ne gibi hatalar yapabileceğini düşünmek ve buna göre davranmaktır.
  - Örneğin sinyal vermeden aniden dönebileceğini düşünüp öndeki araçla aradaki mesafeyi korumak bir savunmacı sürüş tekniğidir.

Savunmacı Programlama - I

- **Savunmacı programlama** programcıya pek çok şeyin hatalı olabileceğini akılda tutarak kodunu geliştirmesini tavsiye eder.
- **Savunmacı programlamanın temel amacı, programın güvenirliğini (reliability) sağlamaktır.**
- Güvenilirlik, program/yazılımın olabildiğince hatadan arınmış olması ve bu sayede çalışmaya devam etmesidir.
- Bu sebeple savunmacı programlama, çalışma sırasında oluşabilecek ve programın çalışmasına engel olacak herhangi istenmeyen bir duruma karşı önleyici tedbirler almayı amaçlar.

Savunmacı Programlama - II

- Bu tedbirler hem programın kullanıcılarına hem de programın diğer geliştiricilerine karşı alınmalıdır:
  - Örneğin kullanıcılar sıklıkla beklenmedik türde veri girerler, bu durumda programın nasıl davranacağı önemlidir,
  - Benzer şekilde her programcı kodunu öyle geliştirmelidir ki diğer programcılar bu kodu rahatlıkla, yanlışa düşmeden kullanıp, hatalı çıkarımda bulunmadan kendi kodlarını geliştirebilsinler.
    - Hatta programcı, ileride kendisi için bile kodunu savunmacı bir şekilde yazmalıdır.

Önleme ve Yönetme - I

- Savunmacı programlamanın tedbirleri iki türlü olabilir:
  - Geliştirme sırasında alınan önlemler ve tabi olarak yapılan testler ile istenmeyen durumun oluşmayacağından emin olmak,
    - Bu amaçla **assertion** mekanizmaları kullanılır,
  - Çalışma zamanında olabilecek durumları öngörmek
    - Bu amaçla **sıra dışı durum yönetimi** (**exception handling**) yapılır.
- Tekniklerin ilk türü önlemeye ikincisi ise yönetmeye odaklanır.

Önleme ve Yönetme - II

- Dolayısıyla bazı tedbirler geliştirme sürecinde alınır ve istenmeyen durumun gerçekleşmeyeceğinden emin olunduktan sonra geliştirilen kod canlı ortama aktarılır.
  - Örneğin
    - sıfıra bölme ya da negatif bir sayının kare kökünün alınmasından kaçınmak,
    - açılan bir dosyanın kapandığından emin olmak,
    - başa ve ortaya ekleme ve çıkarmalar olduğunda bağlı liste (linked list) kullanmak bu türden önlemlerdendir.

Önleme ve Yönetme - III

- Diğer türlü tedbirler ise canlı ortamda bazı durumların olabileceğini düşünür ve çalışan sistem içinde bunları yönetmeye çalışır.
  - Bu tür durumların geliştirme sırasında, canlı ortamda olmayacak şekilde tedbir alın(a)mamasının en temel sebebi, durumun doğrudan bir hata olmaması ve yönetme şartlarının, kullanıcıya bilgi vermek, ondan girdi istemek gibi canlı ortama bağlı olmasıdır.
  - Örneğin, açılması istenilen dosyanın bulunamaması, açılamaması ya da ödeme yapılacak kredi kartının limitinin yetmemesi bu türden önlemlerdendir.

Önleme ve Yönetme - IV

- Bu anlamda önleme genelde geliştirme sırasında yanlış programlama tekniklerinin kullanılmasının ve iş mantığındaki hataların (bug) önüne geçmeyi hedeflerken yönetme, çalışma zamanında iş süreçlerindeki normalin dışındaki durumları öngörmeyi amaçlamaktadır.

Sistemli Önlemler

- Savunmacı programlama gelişi güzel kontroller yapmak, her noktada gerekli-gereksiz ayrımı yapmadan değişkenlerin değerlerini, nesneleri durumlarını vs. kontrol etmek değildir.
- Böye bir yaklaşım hem geliştirmeyi ve bakımı güçleştirir hem de çalışma zamanını hantallaştırır.
- Savunmacı programlama en temelde olabilecek belli başlı durumlara karşı önlem almayı hedefler.

Önlemler - I

- Örneğin şu konularda önleyici tedbirleri, önlemleri almak, ihtiyatlı olmak gereklidir:
  - Tutarlı isimlendirme vb. kod geleneği (code convention) kullanımı,
  - Sisteme dışarıdan yapılan yanlış girdiler (wrong input),
  - Sistem içerisindeki modüllerin birbirlerine geçtikleri geçersiz ya da yanlış girdiler (invalid or wrong parameters),
  - İlkel veri problemleri, değer aralığı (range), değer kesinliği (precision) vs.,
  - `null` referanslar,

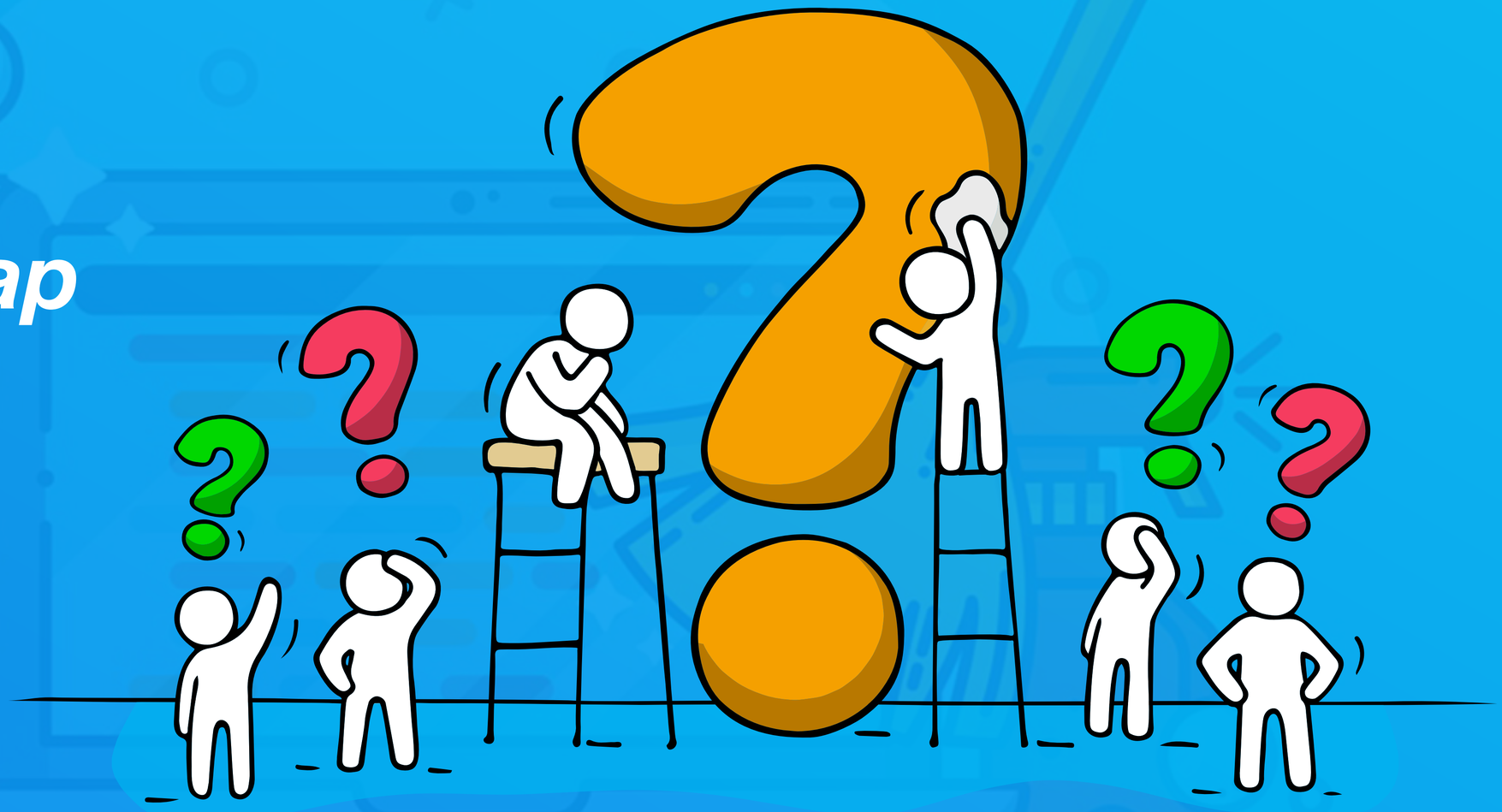
Önlemler - II

- İstenmeyen değişken değerleri ve nesne durumları,
- Değişen nesneler (mutable objects), değişmemesi gereken alanlar ve tutarsız durumlar,
- İlkel/karmaşık (primitive/complex) tip dönüşümleri (type conversions),
- Doğru algoritma, uygun veri yapısı (data structure), torba (collection, container) vs. seçimi.

Önlemler - III

- Etkin CPU, bellek vb. kaynak kullanımları,
- Dosya, veri tabanı vb. dış kaynaklara etkin erişim ve kullanma,
- Güvenliği zedeleyici durumlar,
- İş süreçleri ve kurallarındaki sıra dışı durumlar (exceptions),
- vs.

Soru ve Cevap Zamanı!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

selsoft

build better, deliver faster

Tam Yazılım

Tam Yazılım - I

- **Sağlıklı bir yazılım tamdır (complete)**.
  - Yazılımın tam olması, yazılımın olması gerekenleri yaparken olmaması gerekenleri önlemesi ve olabilecek olanları da öngörürüp gerekli davranışlara sahip olması demektir.

Tam Yazılım - II

- Olmaması gerekenlerin önlemesi savunmacı programlama teknikleri ve assertion yapılarını kullanarak olur,
- Olabilecek olanları öngörmek ve yazılımı ona göre tasarlamak da sıra dışı durum yönetimiyle mümkündür.

Assertions

selsoft

build better, deliver faster

Assertion - I

- **Assertion** kelime olarak öne sürme, iddia etme, tastikleme gibi anlamlara gelir.
- Programlamada ise savunmacı programlamanın bir tekniğidir.
- Assertion, her zaman doğru olması gereken bir durumdur.
  - Dillerde çoğu zaman bir **`boolean`** değer üreten bir mekanzima ile gerçekleştirilir.
- Aksi taktirde hata (assertion error) oluşur.

Assertion - II

- Assertion program akışında olmaması gereken bir durumun gerçekleşmeyeceğinden emin olmak için kullanılır.
- Örneğin
  - Burada `i < 0` olmamalı,
  - Burada `open == false` olmamalı
  - Burada `totalPrice < price` olmamalı
  - Burada `reference == null` olmamalı, vs.

Assertion - III

- Assertion ifadeleri çoğunlukla açılıp kapatılır (enabled/disabled) yapıdadır,
- Geliştirme (development) sırasında assertionlar çalıştırılır,
- Assertion hatası üretecek durumlar tespit edilir ve assertion hatası üretmeyecek şekilde önlemleri alınır, kodda değişiklikler yapılır.

```
public double calculateSquareRoot(double d)
return Math.sqrt(d); // d < 0 ?
}
```

Assertion - IV

- Hiç bir assertion hatası fırlatmayacak hale getirilen kod canlı ortamda assertion yapıları kapatılmış (disabled) olarak çalıştırılır.
  - Assertion ifadeleri koddan çıkarılmaz, sadece kapatılıp açılır.
- Geliştirme ortamında etkin olan assertionlar, yeni sürümler ve geliştirilen kodlar vb. için çalışmaya devam eder.

Assertion - V

- Assertion yapılarının `if-else`'den farkı, açılıp-kapatılma özelliğine sahip olmalarıdır.
  - Koddaki `if-else` ifadeleri kapatılıp açılan yapıda değildir, daima çalışır ama asseretionlar etkin değilse çalışmaz.
- Assertion yapılarının sıradışı durumlardan farkı benzer şekilde, assertionların canlı ortamda etkin olmamalarıdır halbuki sıra dışı durumlar canlı ortamda etkindir.

Dillerde Assertion - I

- Assertion, aralarında bazı kullanım farklılıklarıyla, C/C++, Java, C#, Python, Golang, Rust vs. modern dillerde vardır:
- Örneğin
  - C/C++'da derleme zamanında ön işlemci (preprocessor) ile koddan kaldırılır.
  - Java'da **`assert`** anahtar kelimesi vardır ve **`-enableassertions`** ya da **`-ea`** ile aktif hale getirilir.

Dillerde Assertion - II

- C#'da **`System.Diagnostics.Debug.assert()`** kullanılır ve sadece debug modunda aktiftir.
- Python'da **`assert`** anahtar kelimesi vardır, varsayılan halde çalışır durumdadır ve `-O` seçeneğiyle kaldırılır.

selsoft

build better, deliver faster

Java'da Assertion

assert - I

- **`assert`** anahtar kelimesi Java'ya 1.5 sürümüyle birlikte katılmıştır.
- **`assert`** kullanımının iki şekli vardır:
  - İlkinde **`assert`** kendisini takip eden ifadenin doğruluğunu test etmek için kullanılır.
  - İfade doğruysa çalışma devam eder, yanlışsa **`java.lang.AssertionError`** fırlatılır.

```
assert expression;
```

assert - II

- İkinci şekilde ise **`assert`**'ten sonra iki ifade vardır:

```
assert expression1 : expression2;
```

- Çalışma ilki gibidir.
- Tek fark ilk ifadenin yanlış olması halinde bir değer üreten ikinci ifadenin sonucunun **`AssertionError`**'in uygun bir kurucusuna geçilmesidir.
- İkinci ifade durum ile ilgili mesajdır, beklenmeyen, hatalı durum olabilir.

AssertionError

- `java.lang.AssertionError` bir `Error` sınıfıdır.
- `assert`'ten sonra gelen ifade yanlış olduğunda fırlatılır.
  - Bu durumda varsayılan kurucusu çağrılır ve herhangi bir mesaj almaz.

```
assert expression;
```

  - Eğer bir mesaj geçilirse bunu alan 7 kurucusundan birisi çağrılır.

```
assert expression1 : expression2;
```

assert - III

- **`assert`** kullanımı, maliyetli oluşundan dolayı çalışma zamanı için açılıp kapatılabilen bir özelliktir.
- Varsayılan durumda **`assert`** kullanımı kapalıdır.
- Açmak için JVM'e **`-enableassertions`** ya da **`-ea`** seçeneklerini geçmek gereklidir, aksi taktirde **`assert`** ifadeleri çalışmaz.
- Bu yüzden **`assert`** ifadeleri kod geliştirme aşamasında, kodun doğru çalıştığından emin olmak için kullanılır ve canlı (live) ortamda kapatılır.

AssertExample

- `org.javaturk.cc.ch05.assertion.AssertExample`

Test Amaçlı assert Kullanımı

- `assert` kullanımının açılır-kapatılır bir özellik olması, bu yapının programcı tarafından test amacıyla serbestçe kullanılabilmesine imkan tanır.
- Bu yüzden `assert` genel olarak bir verinin geçerliliğini (invariant) test etmek için kullanılır.
  - Bu veri bir metottaki yerel değişken olabileceği gibi bir nesnenin durumu da olabilir.
  - Ön ve son şartları (pre & post conditions) test etmek için de kullanılır.

SqrtCalculator.java

- `org.javaturk.cc.ch05.assertion.SqrtCalculator`

PrePostConditionsExample

- `org.javaturk.cc.ch05.assertion.PrePostConditionsExample`

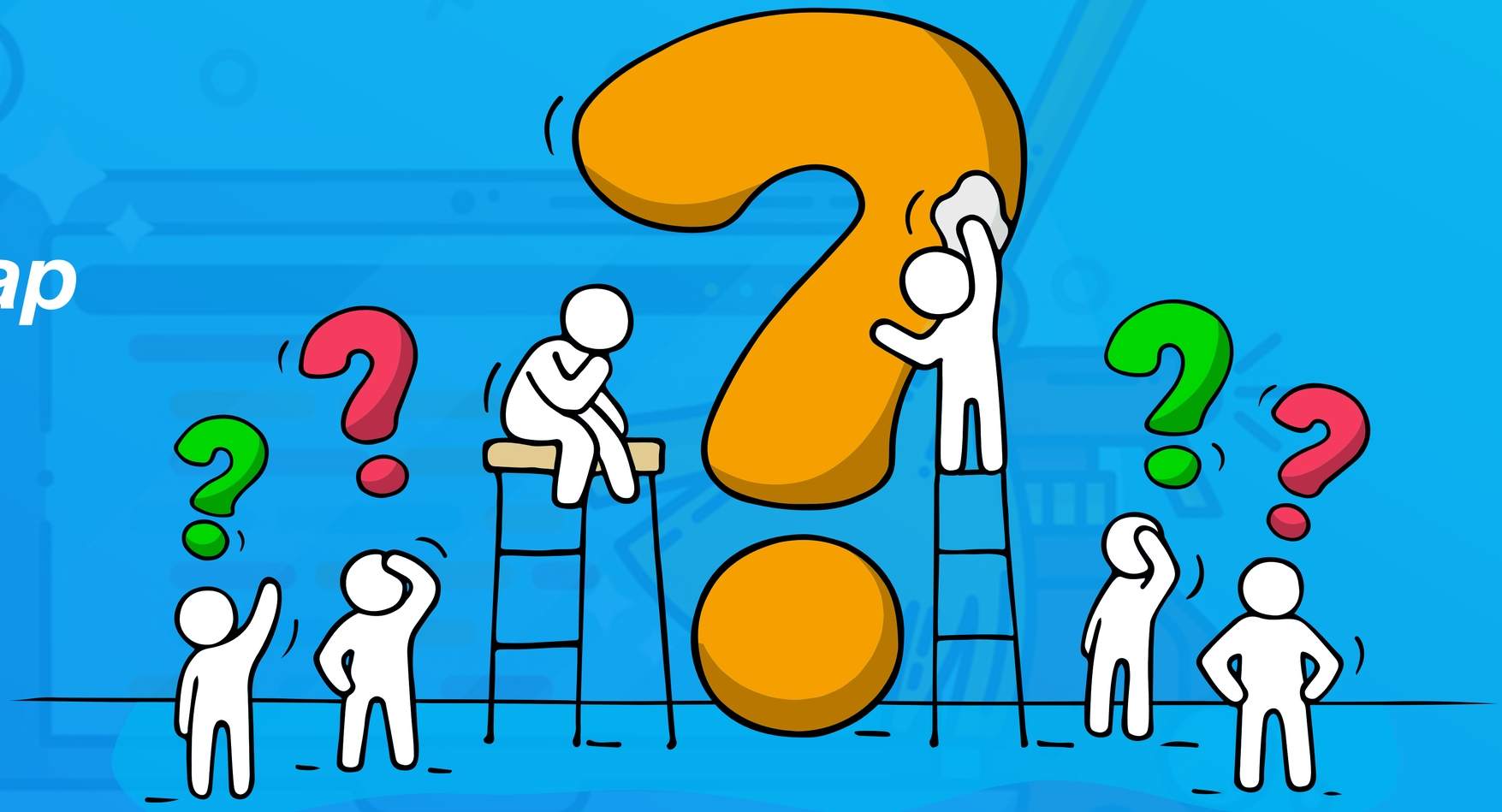
SwitchWithAssertion

- `org.javaturk.cc.ch05.assertion.SwitchWithAssertion`

Assertion ve Sıra Dışı Durumlar

- **`assert`** kullanımı açılır-kapatılırdır ama sıra dışı durum yönetimi kalıcıdır.
- Dolayısıyla **`assert`** canlı ortamda olması muhtemel durumları yakalamak için kullanılmaz.
  - Zaten sıra dışı durum oluşunca yakalanır ve çalışma devam eder.
  - Ama **`AssertionError`** oluşunca JVM çalışmasını durdurur.
- **`assert`** kodun doğru çalıştığından emin olmak amacıyla, bir geliştirme zamanı yapısı olarak, bir durumun olmadığını test etmek için kullanılır ve canlı ortamda çalıştırılmaz.

Soru ve Cevap Zamanı!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

Sıradışı Durum Yönetimi

selsoft

build better, deliver faster

Sıra Dışı Durum Yönetimi Nedir? - I

- Yazılımda **sıra dışı durum** (**exception**), yazılımın normal çalışmasından bir sapmadır.
- Sıra dışı durumlar, yazılımların çalışması sırasında icra edilen süreçlerde beklenen ve olması gerekenler dışında meydana gelen anormal hallerdir.
- Programlama dillerinde bu durumları yönetmek için **sıra dışı durum yönetimi** (**exception handling/management**) isimli yapılar bulunur.

Sıra Dışı Durum Yönetimi Nedir? - II

- Önemli olan sıra dışı durumlardan kaçınmak değildir, çünkü sıra dışı durumlar aslında o kadar da sıra dışı değildir, işin bir parçasıdır:
  - Kredi kartıyla ödeme yaparken kartın limitinin yetmemesi ya da kullanım tarihinin geçmiş olması,
  - Bir dosyayı açmaya çalışırken dosyanın bulunamaması ya da açma yetkisine sahip olunmadığının ortaya çıkması,
  - Başvuru yaparken bir bilginin eksik olması dolayısıyla sürecin ilerleyememesi
- gibi durumlar, iş süreçlerinin tabi parçası olan ama anormal hallerdir.

Sıra Dışı Durum Yönetimi Nedir? - III

- Sıra dışı durumlardan kaçınmak mümkün değildir çünkü sıra dışı durumlar iş mantığının parçasıdır.
- Bu gibi durumların olabileceğini öngörmek ve yazılımı ona göre tasarlamak yazılım kalitesi açısından önemlidir.
- Yanlış olan, iş ve ihtiyaç analizi sırasında süreçler detaylandırılırken bu durumların hiç düşünülmemesidir.
- Yani her şeyin yolunda gideceğini, süreçler ve kurallardan hiç bir sapmanın olmayacağını düşünmek safdilliktir.

Sıra Dışı Durumlar Nasıl Bulunur? - I

- Sıra dışı durumları bulmak için iş süreçleri ve kurallarıyla ilgili
  - Neler yanlış gidebilir?,
  - Hangi başka durumlar olabilir?,
  - Varsayımlarımızda neler doğru olmayabilir?,
  - Ya şöyle olursa ne olur?
- gibi şeytanın avukatlığını yapmak nevinden sorular sormak gereklidir.

Sıra Dışı Durumlar Nasıl Bulunur? - II

- İş süreçleri ve kuralları **`if-else if`** olarak modellenebilecekse **`else`** kısımı veya **`switch-case`** olarak modellenebilecekse **`default`** hali de düşünülmelidir.
- Bu anlamda sıra dışı durum alternatif durum değildir:
  - Ödemenin kredi kartıyla yapılması asıl akışı oluştururken nakit ya da hediye çekiyle yapılması alternatif akış olabilir.
  - Kredi kartının limitinin yetmemesi ya da hediye çekinin süresinin bitmiş olması sıra dışı durumdur.

Tam Yazılım - I

- **Sağlıklı bir yazılım tamdır (complete).**
  - Yazılımın tam olması, yazılımın olması gerekenleri yaparken olmaması gerekenleri önlemesi ve olabilecek olanları da öngörürüp gerekli davranışlara sahip olması demektir.

Tam Yazılım - II

- Olmaması gerekenlerin önlemesi savunmacı programlama teknikleri ve assertion yapılarını kullanarak olur,
- Olabilecek olanları öngörmek ve yazılımı ona göre tasarlamak da sıra dışı durum yönetimiyle mümkündür.

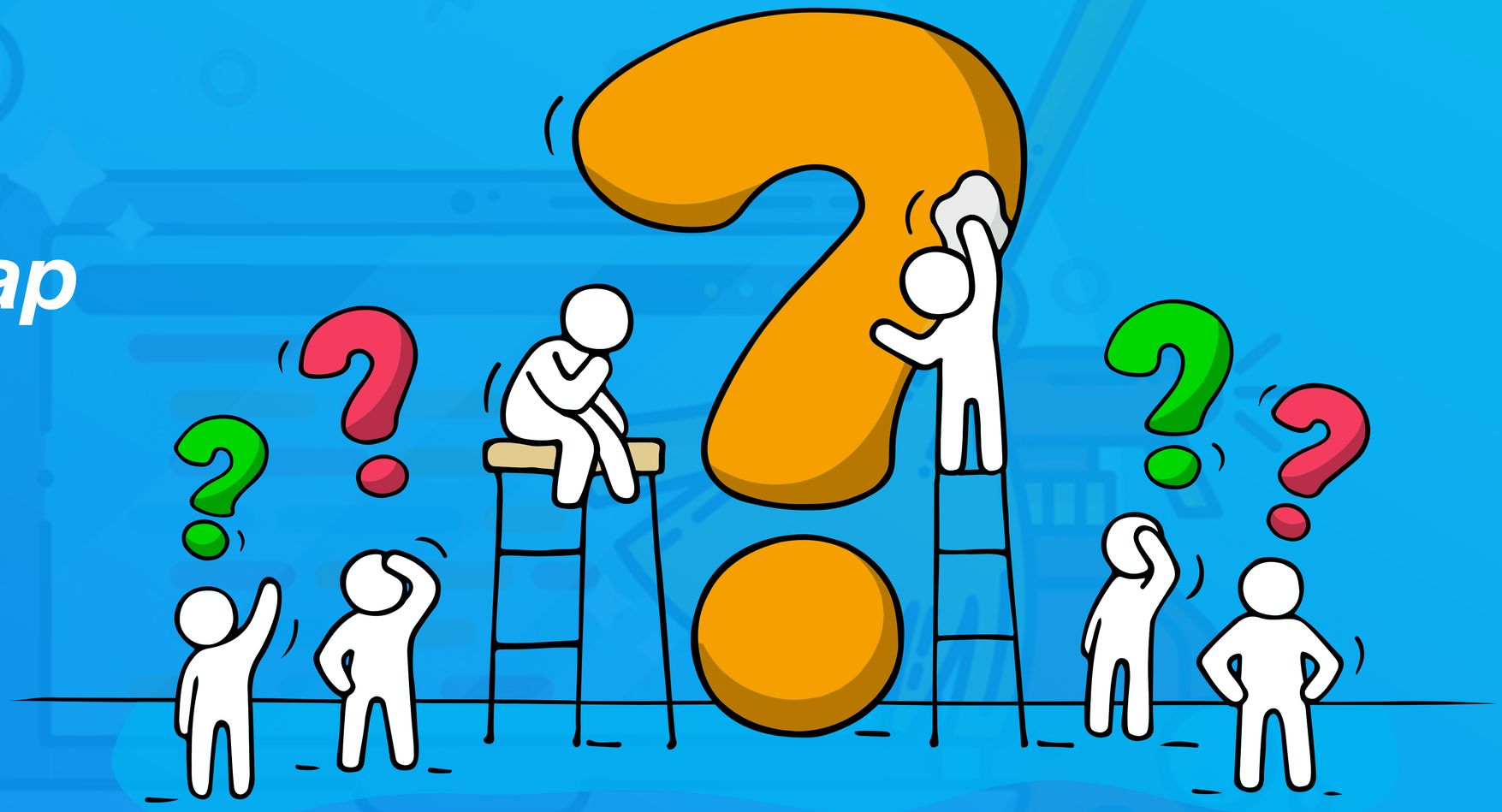
Sıra Dışı Durum Olduğunda - I

- Önemli olan sıra dışı durum oluştuğunda yazılımın çalışmasına nasıl devam edeceğidir.
- Yazılımlarda sıra dışı durum oluştuğunda temelde iki seçenek söz konusu olur:
  - Durumu kullanıcıya bildirmek ve onun inisiyatif almasını istemek ve kararına göre devam etmek,
    - Dosya bulunamadığında yeni bir dosya ismi sormak ya da kredi kartı limiti yetmediğinde başka bir kredi kartı ya da ödeme yöntemi sormak gibi.

Sıra Dışı Durum Olduğunda - II

- Kullanıcıya geri dönmeden dolayısıyla ondan bir inisiyatif gelmesini beklemeden ve hatta ona hissettirmeden, durumu telafi edici karar alıp çalışmaya devam etmek,
  - Dosya bulunamadığında yeni bir dosya oluşturup devam etmek.
- Bu ikinci durum kullanılabilirlik (usability) açısından daha iyi bir seçenek olsa da her zaman uygulamak mümkün değildir.
  - Alış verişte ödeme sırasında kredi kartı limiti yetmediğinde başka bir kredi kartı ya da ödeme yöntemi sormak gibi.

Soru ve Cevap Zamanı!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

Sıra Dışı Durum ve Hata

Sıra Dışı Durum ve Hata - I

- Yazılım açısından sıra dışı durum, hata (error) değildir.
- Hata hangi sebeple olursa olsun, genelde geri dönüşü olmayan durumdur.
- Yazılımlarda farklı tipte hatalar söz konusudur:
  - **Derleme hataları (compiler errors)**: Söz dizimi (syntax) hataları bu tiptendir.

Sıra Dışı Durum ve Hata - II

- **Çalışma zamanı hataları** (**run-time errors**): Programın çalışması sırasında olan hatalardır:
  - Çalışma zamanı yapısındaki bir durumdan kaynaklanan hata, “out of memory” gibi bellek hataları, vs bu türdendir.
- **Mantık hataları** (**Logical errors**): Yazılım süreçlerindeki hatalardır ve **bug** olarak adlandırılır.
  - Bu hatalar tamamen yazılım takımının sorumluluğundadır.

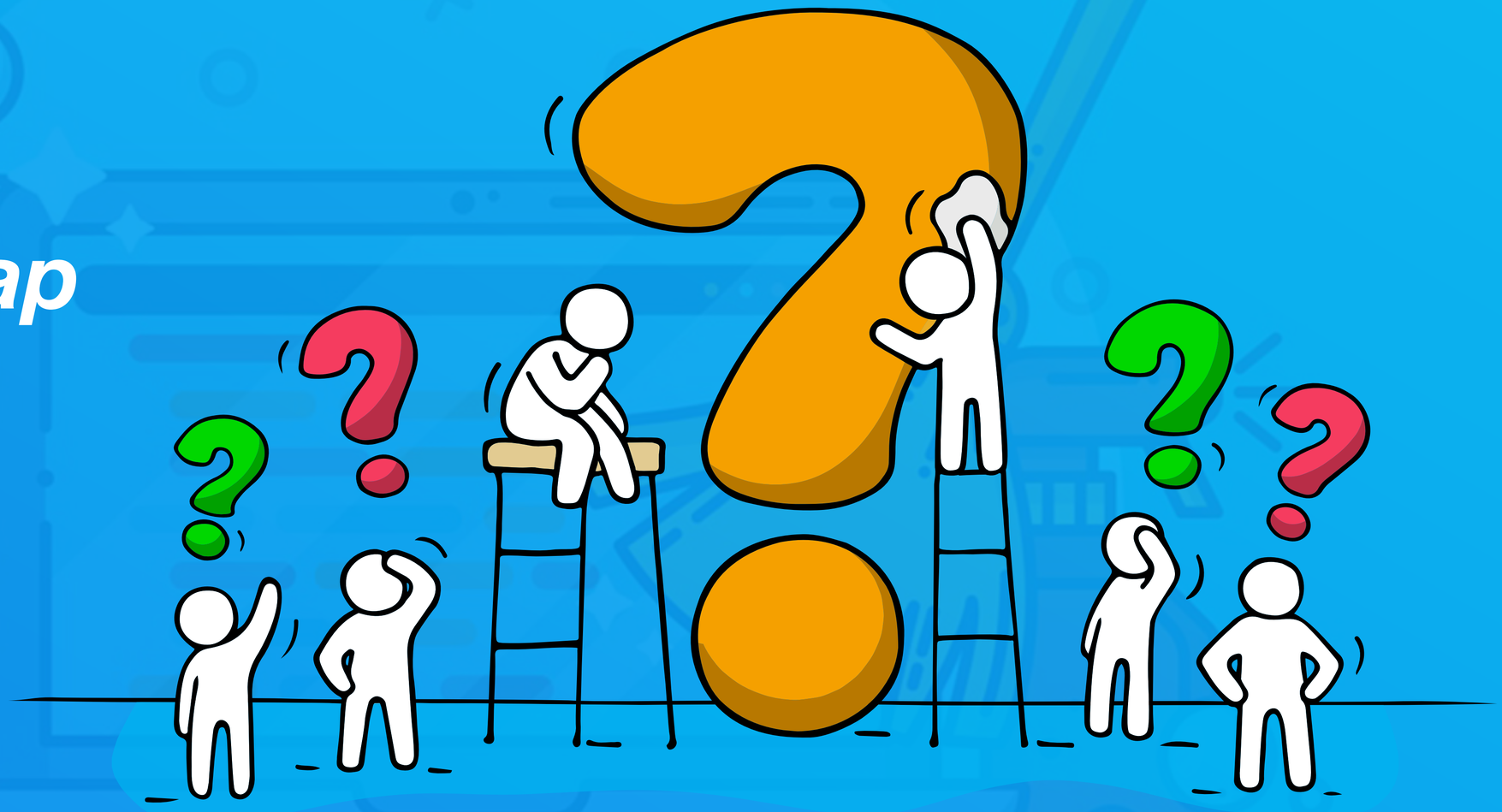
Mantık Hataları - I

- Mantık hatalarının bir kısmı iş mantığıyla ilgilidir.
- İş süreçleri veya iş kurallarının geliştirilmesinde yapılan hatalardır.
  - Satın alınan malın, kupon, indirim, vergi vs.den sonraki fiyatını yanlış hesaplamak bu cinsten, **bug** denince genelde akla gelen hatalardır.
- Sıra dışı durumları atlamak, onları bulup sistemin davranışının parçası haline getirmemek de bugdır, bir ya da daha fazla sayıda buga sebep olabilir.

Mantık Hataları - II

- Bazı mantık hataları ise iş mantığından ziyade, programlama dilinin kullanımıyla ilgilidir:
  - `null` olan bir referansın üzerinde metot çağrısı yapmak,
  - `n` odaya sahip bir dizide `n`. odaya ulaşmaya çalışmak,
  - Bir nesneyi mümkün olmayan bir başka tipe çevirmeye (cast) çalışmak, vb. hatalar.
- Bunlar doğrudan programcının hatasıdır ve değişik savunmacı programlama teknikleri ve testler ile önlenmelidir.

Soru ve Cevap Zamanı!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

Temel Sıra Dışı Durum Kavramları

Sıra Dışı Durum Yönetimi Kavramları - I

- Sıra dışı durum yönetiminin temel kavramları şunlardır:
  - **Sıra dışı durum** (**exception**): Sıra dışı durumun kendisidir, durum ile ilgili bilgileri de taşır.
  - **Fırlatma** (**throw**): Sıra dışı durumu oluşturup çalışma zamanı yapısına bildirmektir.
  - **Yükseltme** (**raise/propagate**): Sıra dışı durumu bir üst bağlama göndermektir.
    - Sıra dışı durum bir metotta fırlatılmışsa o metodun sıra dışı durumu yönetilmek üzere kendisini çağıran metoda göndermesidir.

Sıra Dışı Durum Yönetimi Kavramları - II

- **Yakalama** (**catch**, **handle**): Fırlatılan sıra dışı durumun, yönetilmek üzere özel bir kod parçasına girmesidir.
- **Çağrı zinciri /yığını** (**call chain/stack**): Metotların birbirlerini çağırmalarından doğan zincir ya da yığındır.
- **Yığın izi** (**stack trace**): Herhangi bir anda aktif olan metot pencerelerinin (method frame) yığındaki durumudur.

Geleneksel Sıra Dışı Durum Yönetimi

Geleneksel Sıra Dışı Durum Yönetimi

- Her tür yazılımda sıra dışı durumlar söz konusudur ve yönetilmelidir.
- Dillerde bu tür durumlar için özel yapılar olmasa bile programatik olarak sıra dışı durumlar tespit edilip yönetilir.
- Şöyle gibi bir **`readFile()`** metodu olsun:

```
// Pseudo code
readFile (fileName) {
   …
}
```

- **`readFile()`** metodunda olabilecek sıra dışı durumların belirlenip geleneksel şekilde de olsa yönetilmesi gereklidir.

Muhtemel Sıra Dışı Durumlar

- Verilen örnekte aşağıdaki sıra dışı durumlar söz konusu olabilir:
  - Dosya açılamazsa?
  - Dosyanın boyutu belirlenemezse?
  - Dosyayı açmak için yeterince bellek yoksa?
  - Dosya belleğe okunurken problem çıkarsa?
  - Dosya edit/save edilemezse?
  - Dosya kapatılamazsa?

```
// Pseudo code
readFile (fileName) {
   open the file;
   determine its size;
   allocate that much memory;
   read the file into memory;
   edit the file;
   save the file;
   close the file;
}
```

Geleneksel Sıra Dışı Durum Yönetimi - I

- Geleneksel sıra dışı durum yönetimi, `int` ya da `String` hata kodları üzerinden ve bu hata kodlarını sistemde gezdirme mekanizmalarıyla yapılır.

```
// Pseudo code
errorCodeType readFile(file) {
   int errorCode = 0;
   open the file;
   if (fileIsOpen) {
     determine the length of the file;
     if (gotTheFileLength) {
       allocate that much memory;
       if (gotEnoughMemory) {
         read the file into memory;
         if (readFailed) { errorCode = -1; } // Can't read
       } else { errorCode = -2; } // Not enough memory
     } else { errorCode = -3; }   // File length unavailable
     close the file;
     if (fileDidntClose && errorCode == 0){ errorCode = -4; }
     else { errorCode = errorCode and -4; }
   } else { errorCode = -5; } // File can't open
   return errorCode;
}
```

Geleneksel Sıra Dışı Durum Yönetimi - II

- `readFile()` metodunun aşağıdaki gibi bir metot zincirde çağrıldığını düşünün.
- `readFile()` metodunun fırlattığı sıra dışı durum kodunun `method3()` ve `method2()` üzerinden `method1()`'e yükseltildiği durumda yandaki gibi bir kod yapısı söz konusu olacaktır.

```
// Pseudo code
method1 {
   call method2;
}
method2 {
   call method3;
}
method3 {
   call readFile;
}
```

```
// Pseudo code
method1 {
  errorCodeType error;
  error = call method2;
  if (error)
    doErrorProcessing;
  else proceed;
}

errorCodeType method2 {
  errorCodeType error;
  error = call method3;
  if (error) return error;
  else proceed;
}

errorCodeType method3 {
  errorCodeType error;
  error = call readFile;
  if (error) return error;
  else proceed;
}
```

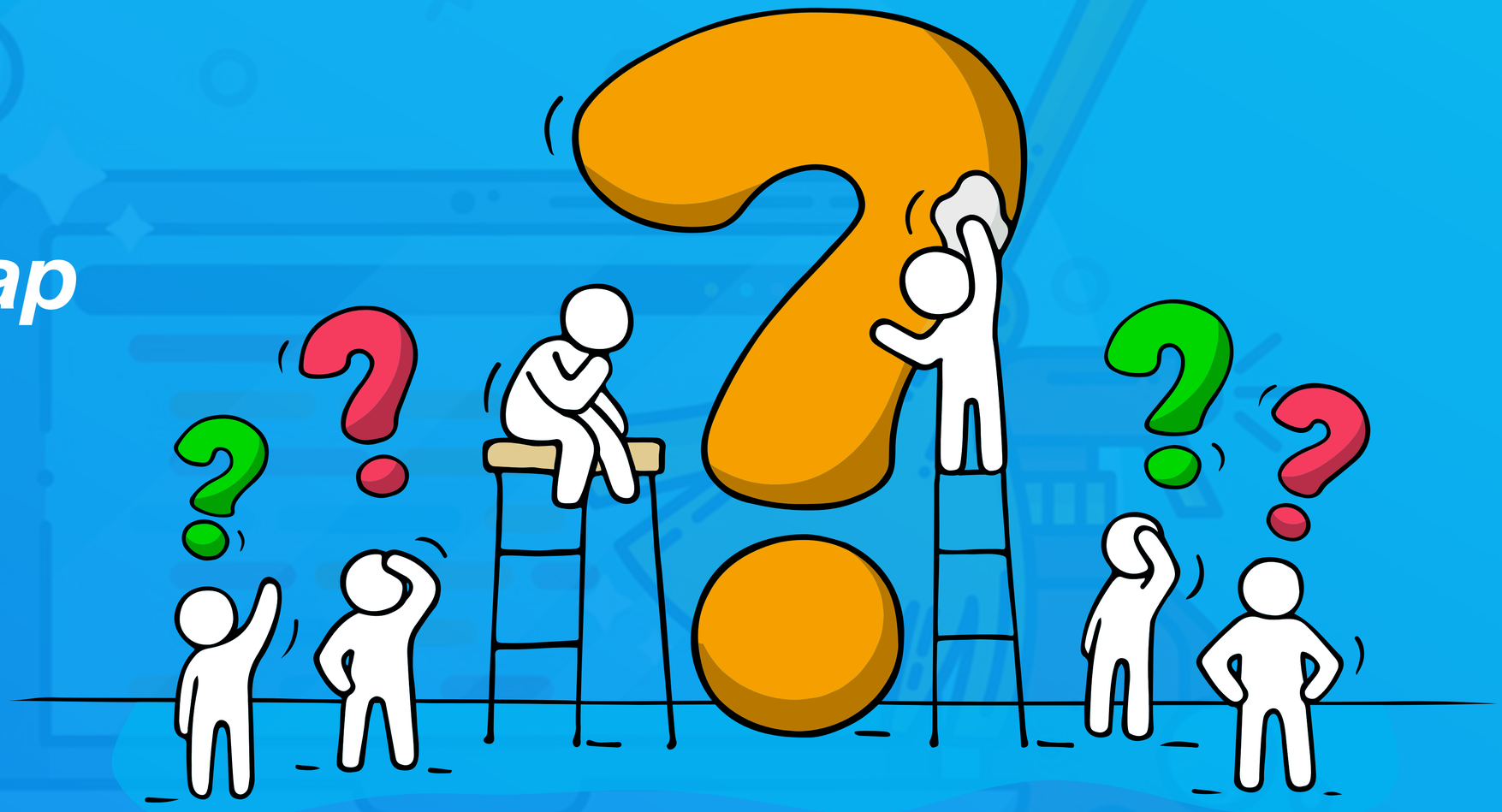
Geleneksel Yaklaşımın Problemleri - I

- Geleneksel sıra dışı durum yönetiminin en problemli iki özelliği şunlardır:
  - Dilde sıra dışı durum yönetim mekanizmaları yoktur, yönetim tamamen programı yazanlar tarafından kurulur.
    - Sıra dışı durumun fark edilmesi, oluşturulması, yığındaki metotlara ulaştırılması ve yakalanması, tamamen dildeki genel amaçlı yapılarla halledilir.
    - Bu durum programları daha karmaşık ve anlaşılmaz kılar.

Geleneksel Yaklaşımın Problemleri - II

- Sıra dışı durumlar ancak `int` ya da `String` vb. tiplerde değişkenlerle ifade edilir.
  - Bu da prosedürel yapıların en temel problemi olan “anlam” problemini tekrar gündeme getirir.

Soru ve Cevap Zamanı!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

selsoft

build better, deliver faster

Modern Sıra Dışı Durum Yönetimi

Modern Dillerde Sıra Dışı Durum Yönetimi - I

- Modern dillerde sıra dışı durum yönetimi için **`try-catch`** ya da benzeri bloklar kullanılır.
  - Aralarında söz dizimsel ve kullanımsal farklılıklar olmakla beraber **C++**, **Java** ve **C#**, **`try-catch`** geleneğine uyar.
    - **Java** ve **C#**, **`try-catch`**'e **`finally`**'yi de ekler.
  - **Python**, **`try-except-else—finally`** yapısını kullanır.
  - **Go**, geleneksel **`try-catch—finally`**'yi kullanmaz, sıra dışı durumları için, hata kodlarını fonksiyonlardan geri döndürmeyi tercih eder.

Modern Dillerde Sıra Dışı Durum Yönetimi - II

- Modern dillerde sıra dışı durumlara has kontrol yapısı yanında sıra dışı durumlar için oluşturduğu nesne hiyerarşisi de söz konusudur.
- Dillerin kütüphanelerinde hazır sıra dışı durumlar olduğu gibi programcıların yenilerini oluşpturmalarına da izin verir.

Modern Sıra Dışı Durum Yönetimi - I

- Örnek olarak ele alınan **`readFile()`** metodunun Java'nın sıra dışı durum yönetimiyle nasıl yönetileceğini kavramsal olarak modelleyelim.

```
// Pseudo code
method1 {
   call method2;
}
method2 {
   call method3;
}
method3 {
   call readFile;
}
```

```
// Pseudo code
readFile (fileName) {
   open the file;
   determine its size;
   allocate that much memory;
   read the file into memory;
   edit the file;
   save the file;
   close the file;
}
```

Modern Sıra Dışı Durum Yönetimi - III

- Sıra dışı durum fırlatma ihtimali olan kod **`try`** bloğunda, fırlatılabilecek sıra dışı durumları yakalayacak kod ise **`catch`** bloğuna konur.
- Örnekte 5 tane sıra dışı durum fırlatılma ihtimali olduğundan 5 tane **`catch`** cümleciği (clause) vardır.

```
// Pseudo code
readFile {
  try {
    open the file;
    determine its size;
    allocate that much memory;
    read the file into memory;
    close the file;
  }catch (fileOpenFailed){
      doSomething;
  }catch (sizeDeterminationFailed){
      doSomething;
  }catch (memoryAllocationFailed){
      doSomething;
  }catch (readFailed){
      doSomething;
  }catch (fileCloseFailed){
      doSomething;
  }
   ...
}
```

Modern Sıra Dışı Durum Yönetimi - III

- Alternatif olarak sıra dışı durumu yakalamayıp-yönetmeyip, bir üst bağlama yükseltmek de mümkündür.
- Bu durumda **`try-catch`** bloğu kullanılmaz, oluşabilecek sıra dışı durumlar metodun arayüzünde listelenir öyle ki bu metodu çağıran metot da yakalamak ya da yükseltmekten birini seçebilsin.
- Bunun için **`throws`** anahtar kelimesi kullanılır.

```
// Pseudo code
readFile() throws fileOpenFailed,
  sizeDeterminationFailed,
  memoryAllocationFailed,
  readFailed,
  fileCloseFailed  {

    open the file;
    determine its size;
    allocate that much memory;
    read the file into memory;
    close the file;
}
```

Modern Sıra Dışı Durum Yönetimi - IV

- `readFile()` metodunun fırlattığı sıra dışı durumların bir kısmının `method2()` yakalandığını ama diğer bir kısmının ise `metot1()`'e yükseltilip burada yönetildiğini varsayalım.
- Bu durumda yakalanan sıra dışı durumlar `try-catch` bloklarında tanımlanırken yakalanmayan durumlar bir üst metoda `throws` cümlesi ile yükseltilecektir.

```
// Pseudo code
readFile() throws
  fileOpenFailed,
  sizeDeterminationFailed,
  memoryAllocationFailed,
  readFailed,
  fileCloseFailed  {
    …
}
```

```
// Pseudo code
method1() {
  try {
    call method2();
  } catch (fileOpenFailed){
      doExpProcessing;
  } catch (sizeDeterminationFailed){
      doExpProcessing;
  }
}

method2() throws fileOpenFailed, sizeDeterminationFailed {
  try {
    call method3();
  } catch (memoryAllocationFailed){
      doExpProcessing;
  } catch (readFailed){
      doExpProcessing;
  } catch (fileCloseFailed){
      doExpProcessing;
  }
}

method3() throws fileOpenFailed, sizeDeterminationFailed,
                 memoryAllocationFailed, readFailed, fileCloseFailed {
  readFile();
}
```

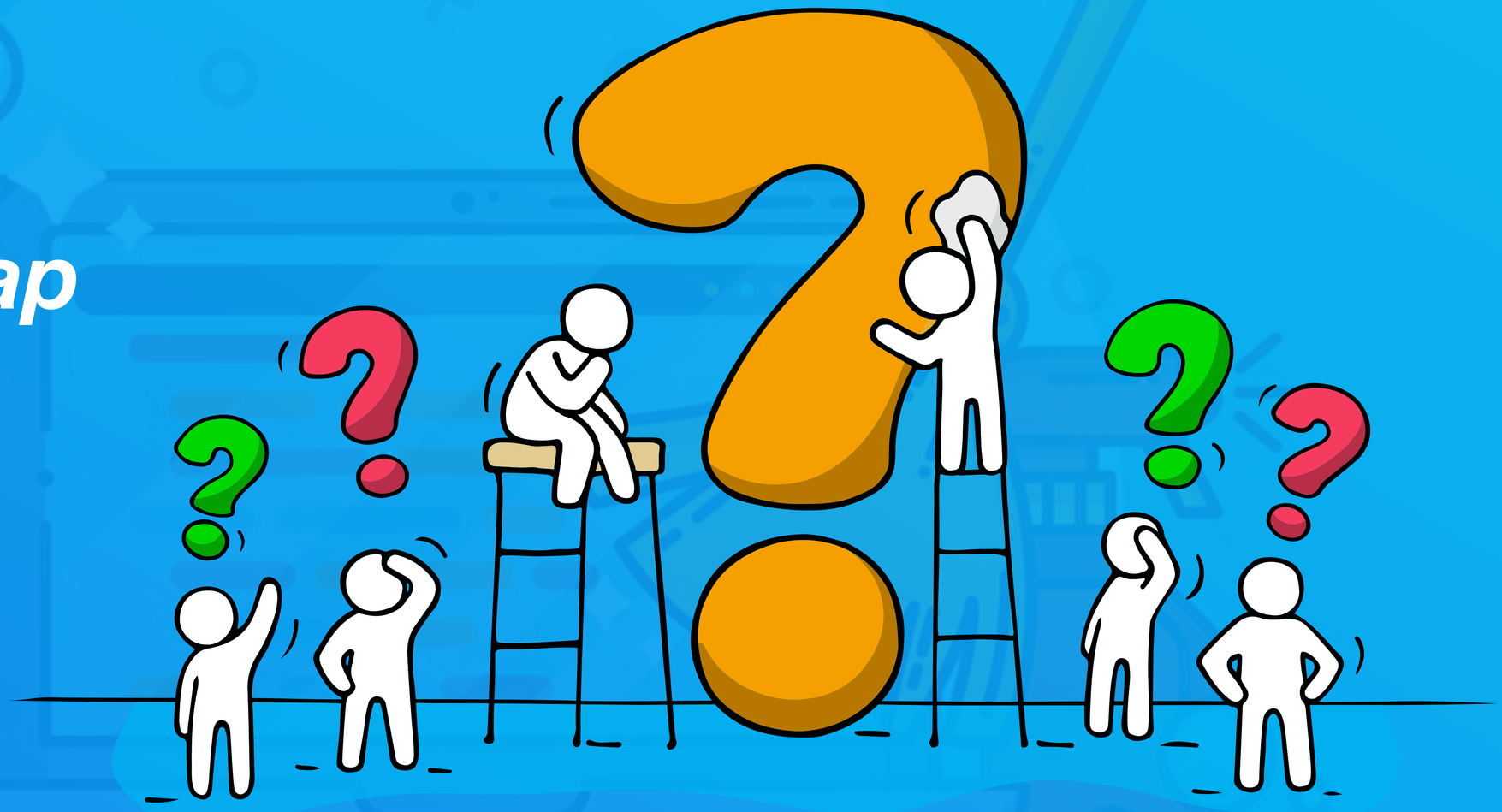
- **`readFile()`** metodunun fırlattığı 5 sıra dışı durumlun tamamının **`method2()`** içinde yönetildiğini varsayalım.
- **`method2()`**'den **`method1()`**'e yükseltilen hiç bir sıra dışı durum yoktur dolayısıyla **`method1()`** sıra dışı durumlardan habersizdir.
- Bu durumda yandaki gibi bir kod yapısı söz konusu olacaktır.

```
// Pseudo code
method1() {
    call method2();
}

method2(){
  try {
    call method3();
  } catch (memoryAllocationFailed){
      doExpProcessing;
  } catch (readFailed){
      doExpProcessing;
  } catch (fileCloseFailed){
      doExpProcessing;
  } catch (fileOpenFailed){
      doExpProcessing;
  } catch (sizeDeterminationFailed){
      doExpProcessing;
  }
}

method3() throws fileOpenFailed,
            sizeDeterminationFailed,
            memoryAllocationFailed, readFailed,
            fileCloseFailed {
  readFile();
}
```

Modern Sıra Dışı Durum Yönetimi - V

- Dolayısıyla soru şudur: Sıra dışı durumu kim yakalar?
- Sıra dışı durumu yakalayan, o durumu düzeltmek için bilgiye sahip olandır.
  - Ya kendisi düzeltir ya da düzeltecek nesnelere yönlendirir.
- Sıra dışı durumu kimler yükseltir (raise)?
  - Sıra dışı durumu düzeltecek bilgiye sahip olmayanlar ise sıra dışı durumu yakalayacak olana yükseltirler.

Soru ve Cevap Zamanı!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

Checked ve Unchecked Sıra Dışı Durumlar

Checked ve Unchecked - I

- `try-catch` ile yakalanması gereken sıra dışı durumlara **checked exception** denir.
- Sıra dışı durum nesnesi olup, fırlatılmasına rağmen `try-catch` ile yakalanması gerekmeyen sıra dışı durumlara da **unchecked exception** denir.
- Checked exceptionlar metot arayüzünde tanımlanması gerekirken unchecked olanların tanımlanmasına gerek yoktur.

Checked ve Unchecked - II

- Checked ve unchecked exception, programlama dünyasında bir tartışma konusudur.
- C++ gibi bazı dillerde tüm sıra dışı durumlar unchecked exceptiondır, checked exception yoktur.
- Java'da ve C#'da bazı sıra dışı durumlar checked diğerleri unchecked exceptiondır.
- Java dünyasında pek çok framework sadece unchecked exception kullanır, onları fırlatır.

Checked ve Unchecked - III

- Checked exceptionlar, fırlatan metodun arayüzünde belirtildiğinden, çağıran metot tarafından **try-catch** ile yönetilmelidir.
- Unchecked exceptionlar, fırlatan metodun arayüzünde belirtilmesi gerekmediğinden ve bundan dolayı belirtilmediğinden, çağıran metot tarafından **try-catch** ile yönetilmelmez ve fırlatıldığında çalışma zamanı çöker, çalışması durur.

Checked ve Unchecked - IV

- Dolayısıyla, çalışma zamanında yakalanıp, ister kullanıcıya geri dönüp yeni girdi isteyerek ister varsayılan bir çözümle ilerleyerek yönetilecek durumlar checked exception olmalıdır.
    - Çalışma sırasında fırlatılılan sıra dışı durumdan çıkılmak isteniyorsa (recovering from an exception), uncheck exception kullanılmalıdır.
- Checked exceptionlar çoğunlukla iş mantığıyla ilgili sıra dışı durumlardır.

Checked ve Unchecked - V

- Eğer bir sıra dışı durumun çalışma zamanında fırlatıldığında yakalanarak yönetilmesi dolayısıyla o sıra dışı durumdan sistemi çökertmeden çıkılmak istenmiyorsa bu durum da sıra dışı durum unchecked exception olmalıdır.
- Unchecked exceptionlar çoğunlukla programlama mantığıyla ilgili sıra dışı hatta hata durumlarıdır.

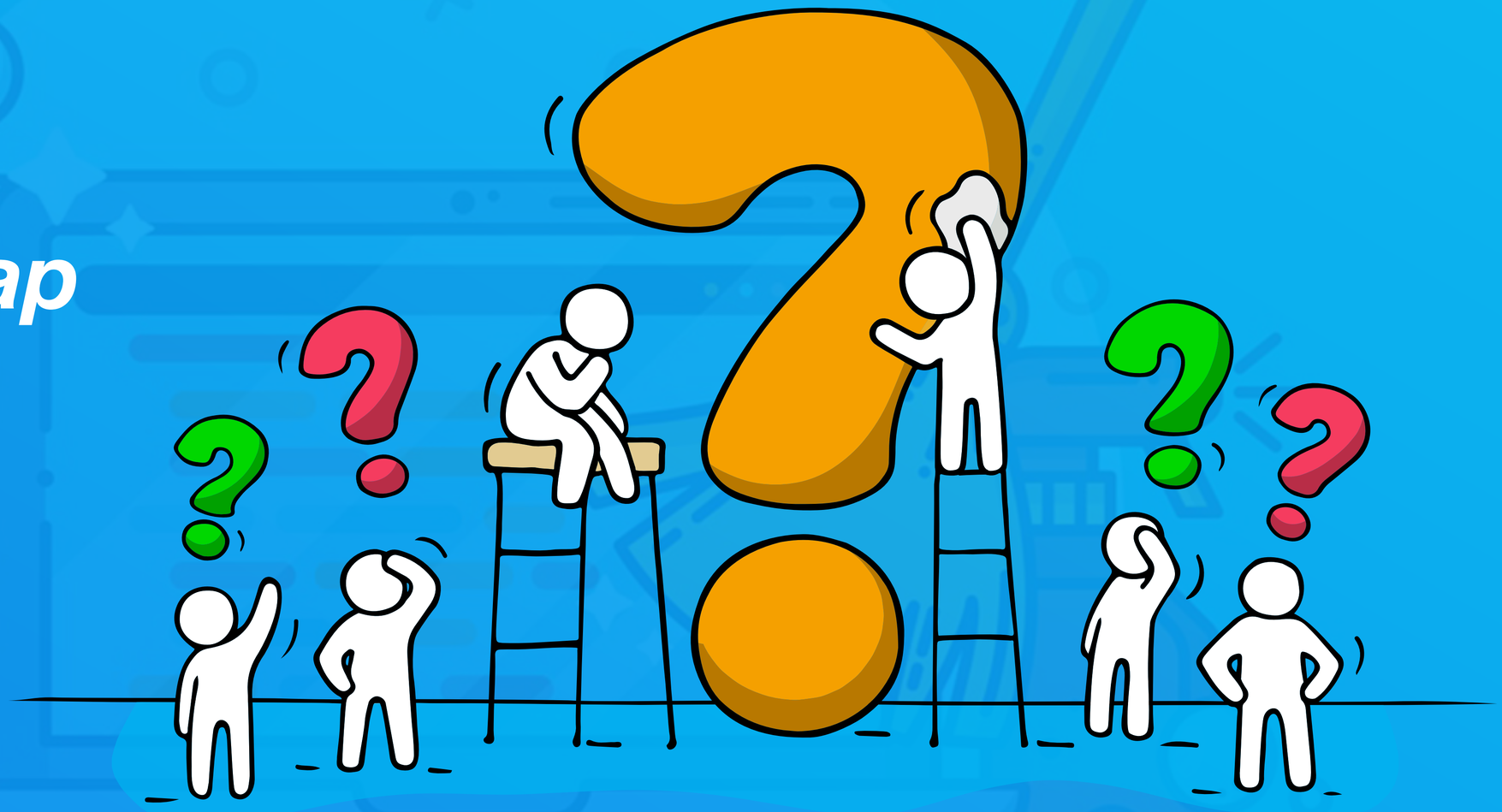
Checked ve Unchecked - VI

- Örneğin Java'da **`IOException`** ve **`FileNotFoundException`**, checked exception iken **`NullPointerException`** ve **`IndexOutOfBoundException`** ve alt sınıfları birer unchecked exceptiondır.
- Dolayısıyla sağlıklı bir sistemin çalışma zamanında hiç bir unchecked exception fırlatılmamalıdır.
- Bu amaçla savunmacı programlama teknikleri ve assertionlar kullanılmalı ve sistemin hiç bir unchecked exception fırlatmadığından emin olunmadan release yapılmamalı ve canlı ortama çıkılmamalıdır.

Checked ve Unchecked - VII

- Modelinizin parçası olarak tasarladığınız sıra dışı durumlar için de checked-unchecked kararını vermeniz gereklidir.
  - Örneğin, karmaşık bir fiyat hesaplamasında, her türlü indirim söz konusu olsa bile sonuç fiyatın minimum bir değerin altına inmemesi gerekliyse, bu minimumun altında çıkan fiyatı sıra dışı durum olarak modellemek istersek hangi tür sıra dışı durum kullanılmalıdır?
  - Bu durumu assertion ile yönetmeyi düşünür müsünüz?

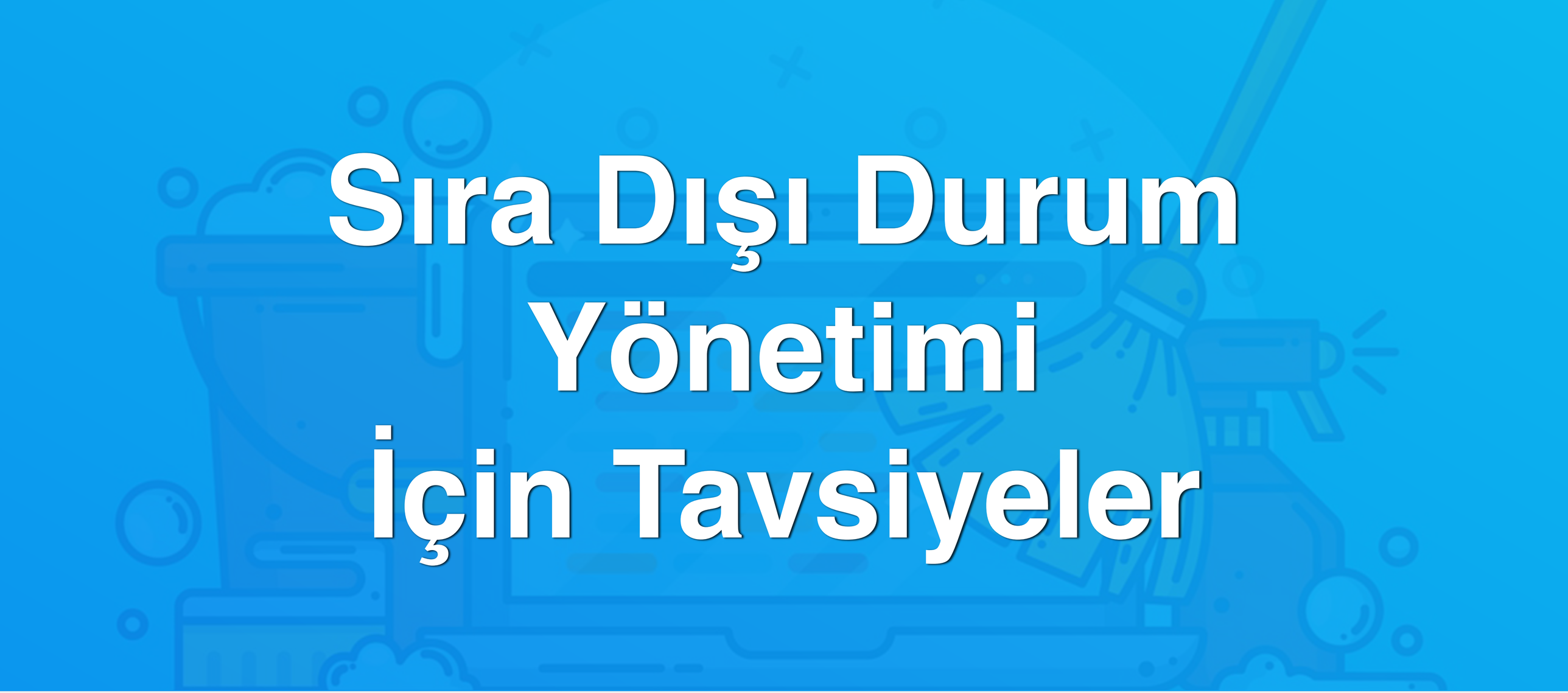
Soru ve Cevap Zamanı!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

Sıra Dışı Durum Yönetimi İçin Tavsiyeler



selsoft
build better, deliver faster

Tavsiyeler - I

- Sıra dışı durumları `int` ve `String` gibi kodlarla yönetmeyin, dilin sıra dışı durumlara has yapılarını kullanın.
- Sıra dışı durumları iş modelinizin parçası haline getirin.
  - Bu amaçla kendi sıra dışı durum nesne hiyerarşisi oluşturun.
  - Sıra dışı durum nesnelerine iş bağlamıya ilgili detayları koyun.
  - Sıra dışı durumların checked-unchecked olacağına karar verin.

Tavsiyeler - II

- Sıra dışı durum nesnelerini kullanırken olabildiğince özel olun, genel olmayın.
  - Örneğin Java'daki **`Exception`** nesnesini hiç bir zaman fırlatmayın, daima daha özel, **`IOException`**, hatta **`FileNotFoundException`** gibi alt nesnelerini fırlatın.
- Fırlatılan sıra dışı durumları API'de belirtin ve hangi durumlarda fırlatılacağını açıklayın.

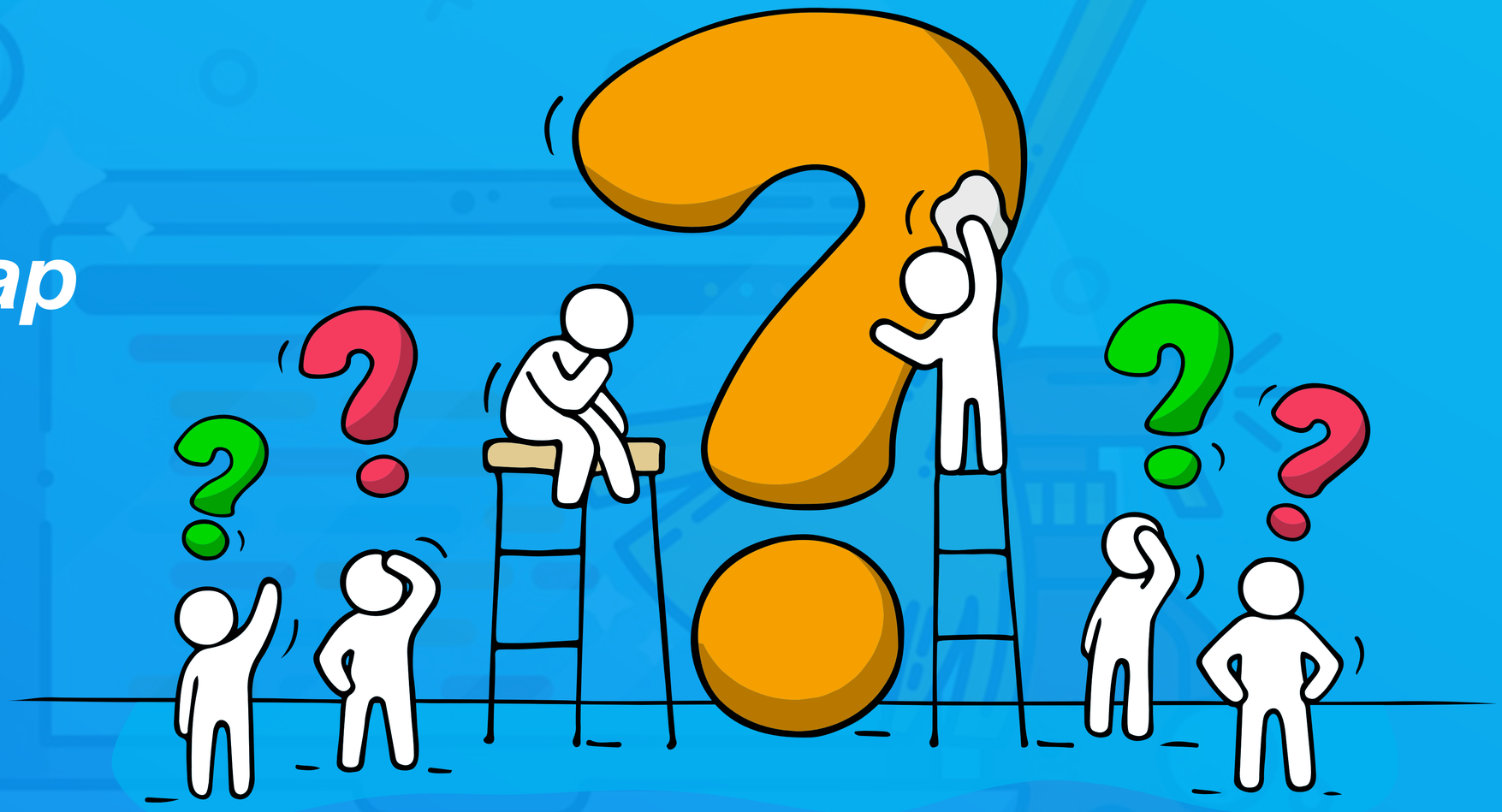
Tavsiyeler - III

- Fırlattığınız sıra dışı durum nesnelerine gerekli detayda veriyi koyun.
  - Diller genelde fırlatılan sıra dışı durumla ilgili hangi sınıfın hangi satırında fırlatıldığı, yığın izin (stack trace) vb. bilgileri sıra dışı durum nesnesinde belirtirler.
  - Sıra dışpı durum nesnelerinde bunun dışındaki iş durumları, sıra dışı duruma sebep olan geçersiz veri ya da duru belirtmelidir.

Tavsiyeler - IV

- Sıra dışı durumlarda kaynak yönetimi önemlidir.
  - `try` bloğunda açılan kaynaklar sıra dışı durum fırlatılırsa kapanmayabilir.
- Bu amaçla kaynaklı `try` kullanın ya da `finally` bloğunda kaynaklarını kapatın.
- Sıra dışı durumları loglayın.
  - Bu amaçla bir loglama yapısı kurun.

Soru ve Cevap Zamanı!

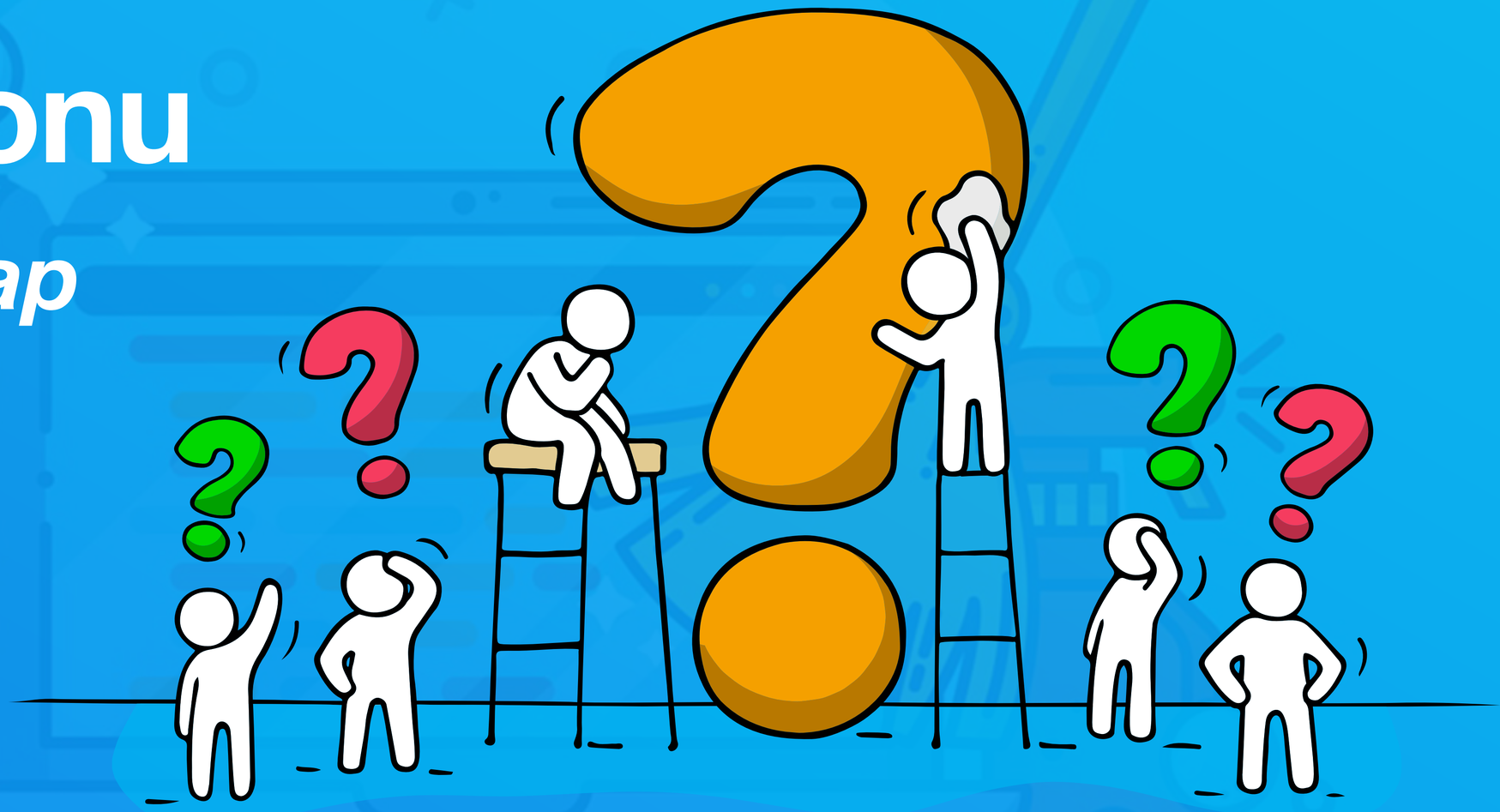


selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

Bölüm Sonu

Soru ve Cevap Zamanı!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr