# Clean Code

Akın Kaldıroğlu
akin@javaturk.org

30 Eylül 2014

# Akın Kaldıroğlu Kimdir?

- Akın Kaldıroğlu, Ayvalık'lıdır.
- 1990 İTÜ mezunudur.
- 1993-2001 yılları arasında ABD'de Bilgisayar ve Yazılım Mühendisliği yüksek lisans eğitimleri almış ve çalışmıştır.
- Analist-programcı olarak başladığı kariyerine Yazılım Mühendisliği ve Java danışmanı ve eğitmeni olarak devam etmektedir.
- **www.javaturk.org**'da blog yazmaktadır.
- Müzik, felsefe ve çocukları en çok sevdiği hobileridir.
- **akin@javaturk.org** ve sosyal medyadan rahatlıkla ulaşılır.

# Seminerler

- Bu tür seminerleri vermedeki amacım, tanışmak, paylaşmak, öğrenmek ve tanıtımdır.
  - Clean Code
  - Tasarım Şablonları (Design Patterns)
  - Java Kodunuzun Nesne-Merkezli Olmadığının 10 İşareti
  - *Java 8 ve Fonksiyonel Programlama*
  - *JVM ve Tuningi (JVM and Its Tuning)*

Hanlon's razor der ki "Never attribute to malice that which is adequately explained by stupidity."

"Aptallıkla açıklanabilecek şeyi kötü niyete yormayın."

Johann Wolfgang von Goethe de der ki:

"..misunderstandings and neglect create more confusion in this world than trickery and malice. At any rate, the last two are certainly much less frequent."

".. yanlış anlamalar ve ihmal, kurnazlık ve kötü niyetten daha fazla karışıklık çıkarır. Ne olursa olsun, son ikisi daima çok daha seyrektir."

# Clean Code Nedir? - I

- Bjarne Stroustrup: şık ve etkin (elegant and efficient)
- Grady Booch: basit ve doğrudan (simple and direct)
- Michael Feathers: dikkatli, önem veren birisi tarafından yazılmış görünür (it looks like it was written by someone who cares)
- Ron Jeffries: küçük, ifade gücü yüksek, basit ve tekrarsız (small, expressive, simple, and no duplication)

# Clean Code Nedir? - II

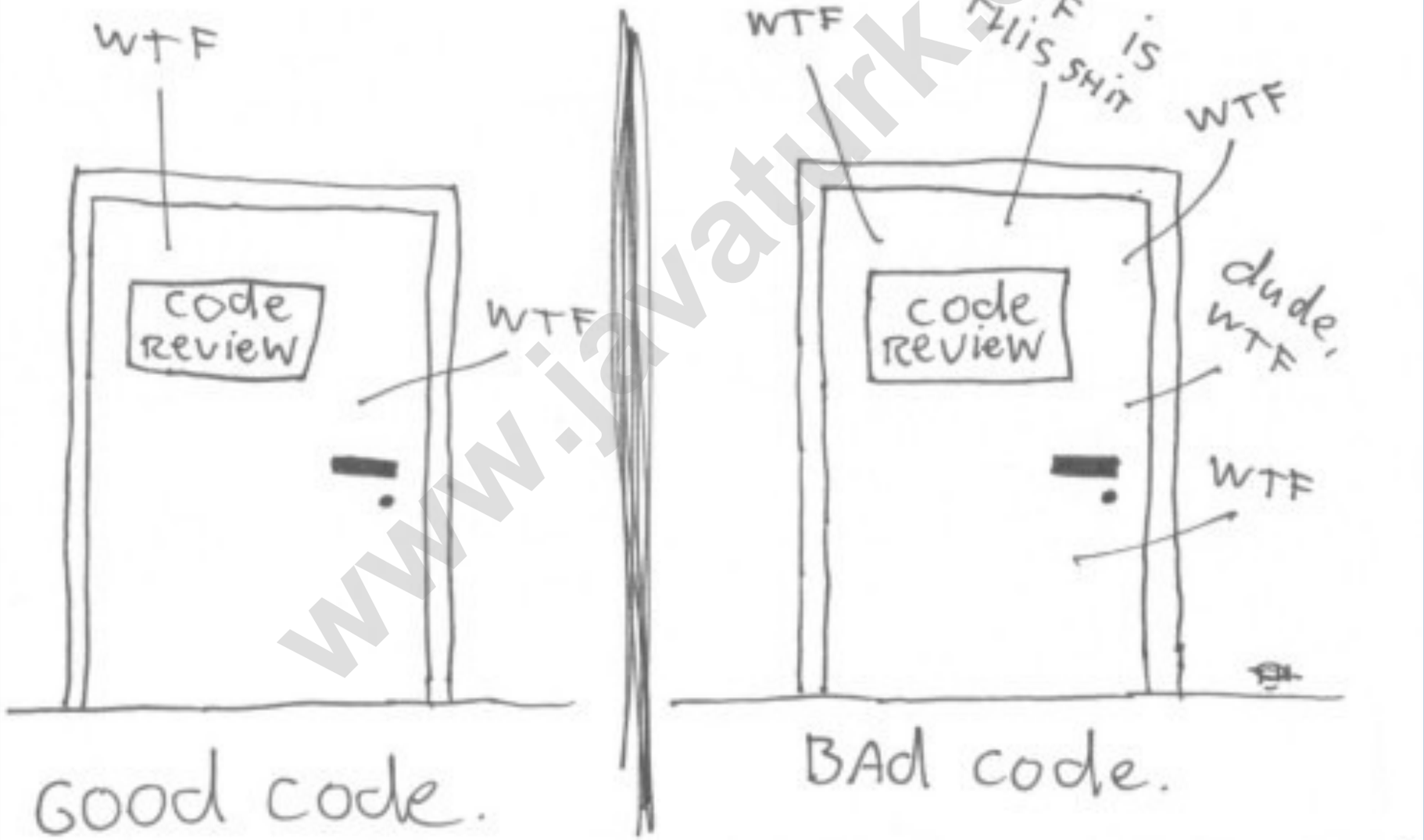
**Temiz kod, orijinal yazarından başka developer tarafından okunabilir ve geliştirilebilir.** Birim ve kabul testlerine sahiptir. Anlamlı isimleri vardır. Bir şeyi yapmanın pek cok yolundan ziyade tek bir yolunu saglar. Açık-seçik olarak tanımlanmış minimal bağımlılıklara sahiptir ve **temiz ve minimal bir API** sunar. Kod okunabilir olmalıdır, çünkü sadece programlama diline bağlı kalınırsa, her türlü gerekli bilgi kodda açık bir şekilde ifade edilemez.

Dave Thomas

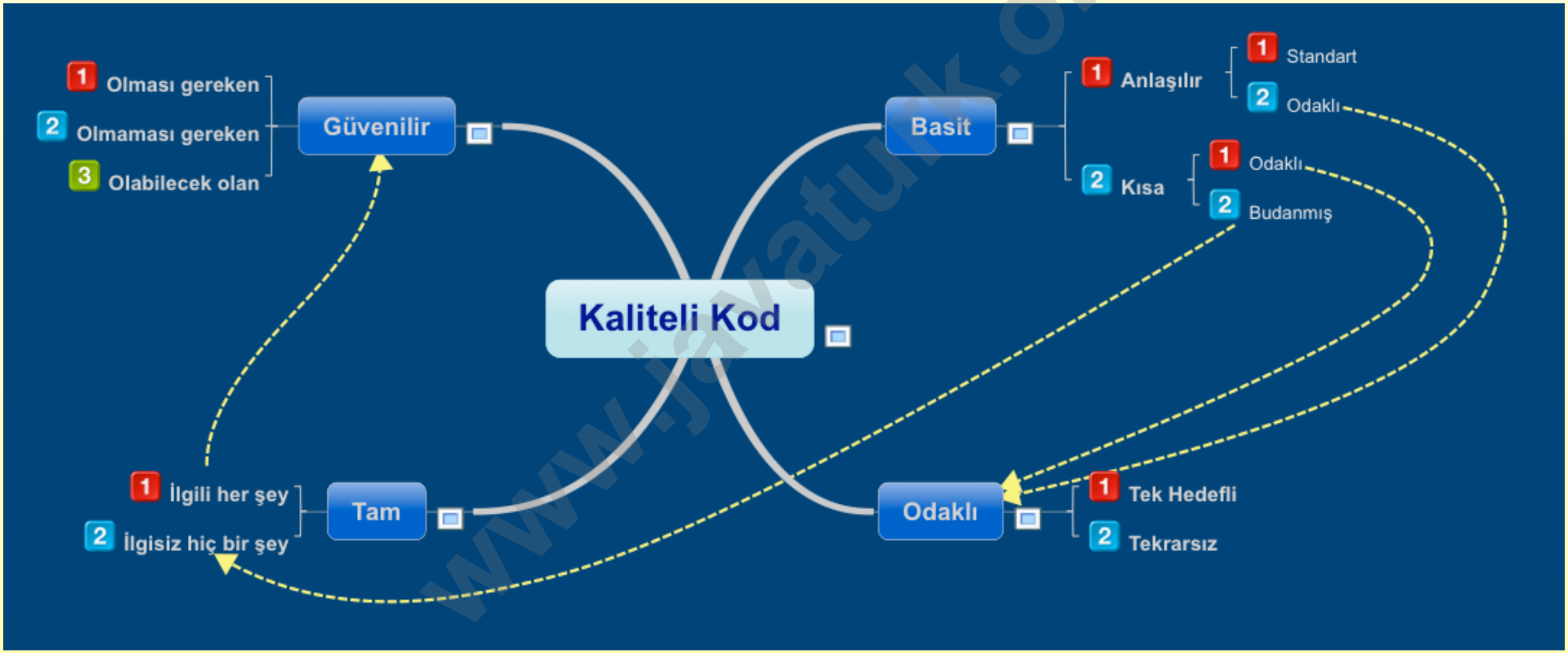
# Bence “Clean Code”

- Yazılım aslen karmaşıktır, geliştirmek ise zordur.
  - Aslolan geliştirmek değil değiştirmektir (bakım), dolayısıyla değişebilen yazılım geliştirmek daha da zordur.
- **Clean Code, zaten zor olan programlama sürecini, insan ürünü zorluklarla daha da zorlaştırmamaktır.**
  - Clean code, teknolojiden, sektörden bağımsız olarak, temiz, anlaşılır (kaliteli) kod yazmaktır.

# The ONLY VALID MEASUREMENT OF CODE QUALITY: WTFs/MINUTE



# Temiz Kod


Kaliteli Kod
Güvenilir
1 Olması gereken
2 Olmaması gereken
3 Olabilecek olan
Basit
1 Anlaşılır
1 Standart
2 Odaklı
2 Kısa
1 Odaklı
2 Budanmış
Tam
1 İlgili her şey
2 İlgisiz hiç bir şey
Odaklı
1 Tek Hedefli
2 Tekrarsız

# Basitlik - I

- Basitlik meziyettir, karmaşıklık gelişi güzelliktir.
  - Basitlik, sanılanın aksine erişilmesi karmaşıklıktan daha zor olandır.
  - Bir şeyi ilk defa yaparken muhtemelen karmaşık yaparız.
  - Basit olan rahat anlaşılır, tutarlı, tek düze ve umulandır; şaşırtmayandır.
- Basitleştirmek, bir şeye tümüyle, her yönüyle hakim olup, asıl ile teferruatı ayırmaktır, gelişi güzel indirgeme değildir.

# Basitlik - II

Einstein'e atfedilen

Her şey olabilidiğince basit olmalı, ama daha da basit değil.
(Everything should be made as simple as possible, but no simpler.)

ya da

Her şeyi olabildiğince basitleştir, ama daha da basit yapma.
(Make things as simple as possible, but not simpler.)

cümleleri bunu ifade eder.

# Basitlik - III

- Fransız yazar Antoine de Saint-Exupery'ın şu sözü mükemmel olmanın en temel özelliği olan basitliği çok güzel bir şekilde ifade etmektedir:

Mükemmellik, eklenecek bir şey olmadığında değil, çıkarılacak bir şey olmadığında başarılır.

(Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away.)

# Basit Kod Nasıldır? I

- Kaliteyi tanımlamak zordur ama en azından kötü örneklere bakarak kalitenin ne olduğuyla alakalı bir şeyler söyleyebiliriz.
- En azından böyle değildir!

```
cardInfo.setIdNo(cardInfo.getTcCitizen() ?
 (formUtils.isNullOrEmptyString(verifyOtpWSResponse.getTckn()) ?
  registrationForm.getTckn() : verifyOtpWSResponse.getTckn()):
(formUtils.isNullOrEmptyString(verifyOtpWSResponse.getCustno())) ?
  registrationForm.getTckn(): verifyOtpWSResponse.getCustno());
```

# Basit Kod Nasıldır? II

Bu çok daha basittir.

```
// Burada CardInfo'nun Idsi atanır. Id olarak eğer musteri TC vatandasi ise
// ve ya WS'den ya da fromdan gelen tckn atanır. Eğer müşteri TC vatandaşı
// değilse Id olarak customerNo alınır.
// Burada da yine ya WS'den ya da formdan gelen customer no atanır.

String idNo = null;
Boolean isTcCitizen =  cardInfo.getTcCitizen();
String tcknFromForm = registrationForm.getTckn();
String tcknFromWS = verifyOtpWSResponse.getTckn();
boolean nullOrEmptyTcknFromWS  = formUtils.isNullOrEmptyString(tcknFromWS);
String custNoFromWS = verifyOtpWSResponse.getCustNo();
boolean nullOrEmptyCustNoFromWS  = formUtils.isNullOrEmptyString(custNoFromWS);

 if(isTcCitizen){
      if(nullOrEmptyTcknFromWS)
          idNo = tcknFromForm;
      else
          idNo = tcknFromWS;
 }
 else{
       if(nullOrEmptyCustNoFromWS)
          idNo = tcknFromForm;
      else
          idNo = custNoFromWS
 }
 cardInfo.setId(idNo);
```

```
cardInfo.setIdNo(cardInfo.getTcCitizen() ?
 (formUtils.isNullOrEmptyString(verifyOtpWSResponse.getTckn()) ?
  registrationForm.getTckn() : verifyOtpWSResponse.getTckn()):
(formUtils.isNullOrEmptyString(verifyOtpWSResponse.getCustno())) ?
  registrationForm.getTckn(): verifyOtpWSResponse.getCustno());
```

```
// Burada CardInfo'nun Idsi atanır. Id olarak eğer musteri TC vatandasi ise
// ve ya WS'den ya da fromdan gelen tckn atanır. Eğer müşteri TC vatandaşı
// değilse Id olarak customerNo alınır.
// Burada da yine ya WS'den ya da formdan gelen customer no atanır.
String idNo = null;
Boolean isTcCitizen =  cardInfo.getTcCitizen();
String tcknFromForm = registrationForm.getTckn();
String tcknFromWS = verifyOtpWSResponse.getTckn();
boolean nullOrEmptyTcknFromWS  = formUtils.isNullOrEmptyString(tcknFromWS);
String custNoFromWS = verifyOtpWSResponse.getCustNo();
boolean nullOrEmptyCustNoFromWS  =
formUtils.isNullOrEmptyString(custNoFromWS);

 if(isTcCitizen){
      if(nullOrEmptyTcknFromWS)
          idNo = tcknFromForm;
      else
          idNo = tcknFromWS;
 }
 else{
       if(nullOrEmptyCustNoFromWS)
          idNo = tcknFromForm;
      else
          idNo = custNoFromWS
 }
 cardInfo.setId(idNo);
```

# Basit Kod

- Bu iki kod örneği arasında en temel fark anlaşılırlıktır.
  - Bir metotta yapılabilecek bir işi bir cümlede yapıyor.
    - Soyutlama seviyesi yanlış!
  - Birden fazla iş, gereksiz bir kısaltmayla tek bir işe indirgenmiş.
    - Karmaşıklığı arttırdığı gibi tekrar kullanımı da önlüyor.
  - Mekanı rahat kullanmıyor, görüntü açısından sıkıntılı.
  - Açıklanmış, dokümante edilmiş değil.

# Basit Kod

# Basitlik İlkesi

- Basit olan kodun temelde iki özelliği vardır:
  - rahat anlaşılır,
  - ve olabildiğince kısadır.
- Rahat anlaşılan kod hem **standartlara uygun** olarak yazılmıştır hem de **odaklı**dır.
- Basit kod **kısa**dır, küçüktür, çünkü hem odaklıdır hem de **budanmış**tır yani sadece gerekli olanı yapar, gereksiz olana dokunmaz.
- **KISS: Keep it simple, stupid - keep it simple and short**

**The difference between a bad programmer and a good programmer is understanding . That is, bad programmers don't understand what they are doing, and good programmers do. Believe it or not, it really is that simple.**

**İyi programcıyla kötü programcı arasındaki fark anlamadır. Yani kötü programcılar ne yaptıklarını anlamazlar, iyi programcılar ise anlarlar. İster inanın, ister inanmayın ama bu gerçekten bu kadar basittir.**

**Max Kanat-Alexander in Code Simplicity**

# Anlaşılır Kod - Şekil Şartları

- Anlaşılır kod öncelikle kod standartlarına uyar, yani:
  - anlaşılır isimler içerir,
  - ferahtır, mekanı etkin kullanır dolayısıyla rahat okunur,
  - tutarlıdır yani şaşırtmaz, beklendiği gibidir, aşina olunandır,
  - açıklanmış - dokümante edilmiş (commented - documented) koddur.
- Herhangi bir sebepten standartlara uyulmadıysa, muhakkak bu durum sebebiyle birlikte belirtilir.

# Anlaşılır İsimlendirme - I

- Anlaşılır isimlendirmenin en temel engeli kısaltmadır.
- Her kısaltma bir kodlamadır ve bir zihinsel durumu yansıtır.
- Zihinsel kodlamalardan kaçının, açıkça anlaşılır olun.

```
private int nofInst;
```

```
private int noflnst;
```

- "*I*" mı "*L*" mi "*1*" mi? "*L*" ise "*fl*" "*float*" mı demek?
- "*number of*" mu "*no*" mu?
- Ya da "*Institution*" mı, "*Installation*" mı yoksa "*Installments*" mı?

# Anlaşılır İsimlendirme - II

- “executeInner“ ya da “executeUser” ne demek?

```
public boolean executeInner(…){
      ...
}

public boolean executeUser(…){
      ...
}
```

- Ne yapıyorsanız ona isim verin.

```
public List<RegisteredEmail> findPrimaryUserListByUid(String uid);
```

# Anlaşılırlık ve Kısalık

- Anlaşılır olma ile kısa olma “?:” örneğindeki gibi zaman zaman çelişebilir,

- Bu durumda aslolan anlaşılır olmadır, kısa olmak adına anlaşılır olmaktan fedakarlıkta bulunamayız.

```
double rs = a + ++b * c/a * b;

yerine

double rs = a + (++b)* ((c / a)* b);

hatta

b++;
double d = c / a;
double rs = a + b * d * b;
```

# Standartlara Uyum

- Muhakkak isimlendirme ve şekil (format) standartlarınız olsun:
  - Her dil için ayrı ayrı
  - Veri tabanı için
  - XML, properties dosyaları, web servisleri vs. için
- Ve bu standartlara uyum kabul edilebilir kodun olmazsa olmaz bir özelliği olsun.
- Araçlar, pair-programming ve gözden geçirmeler yardımıyla standartların uyulmasını kontrol edin.

# İsimlendirme ve Şekil Standartları

- Java, kültür olarak muhtemelen isim ve şekil standartlarının en fazla uyulduğu ortamdır:
  - Java Code Conventions September 12, 1997 (Oracle Java Code Conventions http://www.oracle.com/technetwork/java/codeconv-138413.html)
  - http://www.ambysoft.com/downloads/javaCodingStandards.pdf
  - Google Style of Java http://google-styleguide.googlecode.com/svn/trunk/javaguide.html

# JavaTürk Standardı

- JavaTurk'ün de Türkçe olarak oluşturulmuş bir isimlendirme ve şekil standardı vardır.

- http://www.javaturk.org/?p=3180 adresinden HTML ve PDF yayınlanmıştır.

- Bu standart, Java dünyasındaki yaygın standartlara uygun olarak hazırlanmıştır.

- Serbestçe kullanabilirsiniz.

# Dokümantasyon

- Kod dokümantasyonunun iki türünden bahsedilebilir:
  - API dokümantasyonu: Kodun arayüzünün dokümantasyonu.
    - JavaDoc nefis bir araçtır.
  - Kod içi dokümantasyon ise, ihtiyaca bağlı olarak kısa "//" notları halinde yapılabilir.
- Debugger kullanmaya harcanan vakti API dokümantasyonuna harcamak çok daha faydalıdır.

# Anlaşılır Kod - İçerik Şartları - I

- İçerik açısından bakıldığında anlaşılır kod, okuyanın çok zeki ya da konuya çok hakim olmasına gerek bırakmadan kavranan koddur.

- Martin Fowler, Refactoring isimli kitabında şöyle der:

Herhangi bir insan bilgisayarın anlayabileceği kod yazabilir. İyi programcılar ise insanların anlayabileceği kod yazarlar.

Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

- Dolayısıyla öncelikle programcı, kendisi için anlaşılır kod yazmalıdır.
  - Kodunu yazdıktan sonra dön bir bak, 3 ay ya da 1 yıl sonra bu kodu anlaman ne kadar sürer?

# Anlaşılır Kod - İçerik Şartları - II

- Anlaşılır kodun en temel içerik özelliği, odaklı olmasıdır.

- Odaklı olan kod, çatı (framework)-bileşen-paket-arayüz-sınıf-metot-blok-değişken, ne olursa olsun, sadece ve sadece bir şeyi halleder.

  - Odaklı kod, bir yerde birden fazla şeyi bir araya getirmez.

- Odaklı kod aynı zamanda kısadır, küçüktür.

  - Yazılımda hiç bir soyutlamanın büyüğü makbul değildir.

- Kodlama, anlama, anlatma, test, değişim vs. için odaklı kod!

# Odaklılık İlkesi

- Odaklı kodun iki özelliği vardır:
  - **Tek hedefli**
  - **Tekrarsız**
- Tek hedeflilik, bir seferde sadece bir şeyi hedeflemek, onu yapmak, halletmek demektir.
  - Bir şeyi değiştirmenin sadece bir sebebi olmalıdır.
- Yapılan işi sadece ve sadece bir yerde ve bir kere yapmak, tekrarsız kodun gereğidir.

# Odaklı Kod

# Prensipler

**Separation of concerns**

**Single-responsibility principle**

**High-cohesion**

**Low-coupling**

# Doğru Soyutlama Seviyesi

- Bu prensipleri uygularken doğru soyutlama seviyesini belirlemek çok önemlidir:
  - Bir pakette bir araya getirilecek sorumluluklar grubunu bir sınıfa sığdırmak,
  - Bir sınıfta ya da bir kaç metotta yapılacak sorumlulukları birleştirip bir metotta yapmak,
  - Bir metotta yapılacakları bir bloka hatta satıra sığdırmak,
  - Bir blokta yapılacak adımları, tek bir satırda, bir adım olarak yapmak
- çok sık rastlanan türden yanlış soyutlamalardır.

# Cümle - Blok

- Aşağıdaki örnekte aslında bir blokla belki de bir metotta ifade edilmesi gereken yapı ile onun tek satırda ifade edilen tek cümlelik karşılığı bulunmaktadır.

```
double rs = a + ++b * c/a * b;



…
double rs = 0.0;
{
   b++;
   double d = c / a;
   rs = a + b * d * b;
}
…
```

```
cardInfo.setIdNo(cardInfo.getTcCitizen() ?
 (formUtils.isNullOrEmptyString(verifyOtpWSResponse.getTckn()) ?
  registrationForm.getTckn() : verifyOtpWSResponse.getTckn()):
(formUtils.isNullOrEmptyString(verifyOtpWSResponse.getCustno())) ?
  registrationForm.getTckn(): verifyOtpWSResponse.getCustno());
```

```
// Burada CardInfo'nun Idsi atanır. Id olarak eğer musteri TC vatandasi ise
// ve ya WS'den ya da fromdan gelen tckn atanır. Eğer müşteri TC vatandaşı
// değilse Id olarak customerNo alınır.
// Burada da yine ya WS'den ya da formdan gelen customer no atanır.
String idNo = null;
Boolean isTcCitizen =  cardInfo.getTcCitizen();
String tcknFromForm = registrationForm.getTckn();
String tcknFromWS = verifyOtpWSResponse.getTckn();
boolean nullOrEmptyTcknFromWS  = formUtils.isNullOrEmptyString(tcknFromWS);
String custNoFromWS = verifyOtpWSResponse.getCustNo();
boolean nullOrEmptyCustNoFromWS  = formUtils.isNullOrEmptyString(custNoFromWS);

 if(isTcCitizen){
      if(nullOrEmptyTcknFromWS)
          idNo = tcknFromForm;
      else
          idNo = tcknFromWS;
 }
 else{
       if(nullOrEmptyCustNoFromWS)
          idNo = tcknFromForm;
      else
          idNo = custNoFromWS
 }
 cardInfo.setId(idNo);
```

# Tek Hedefli Kod: Metot

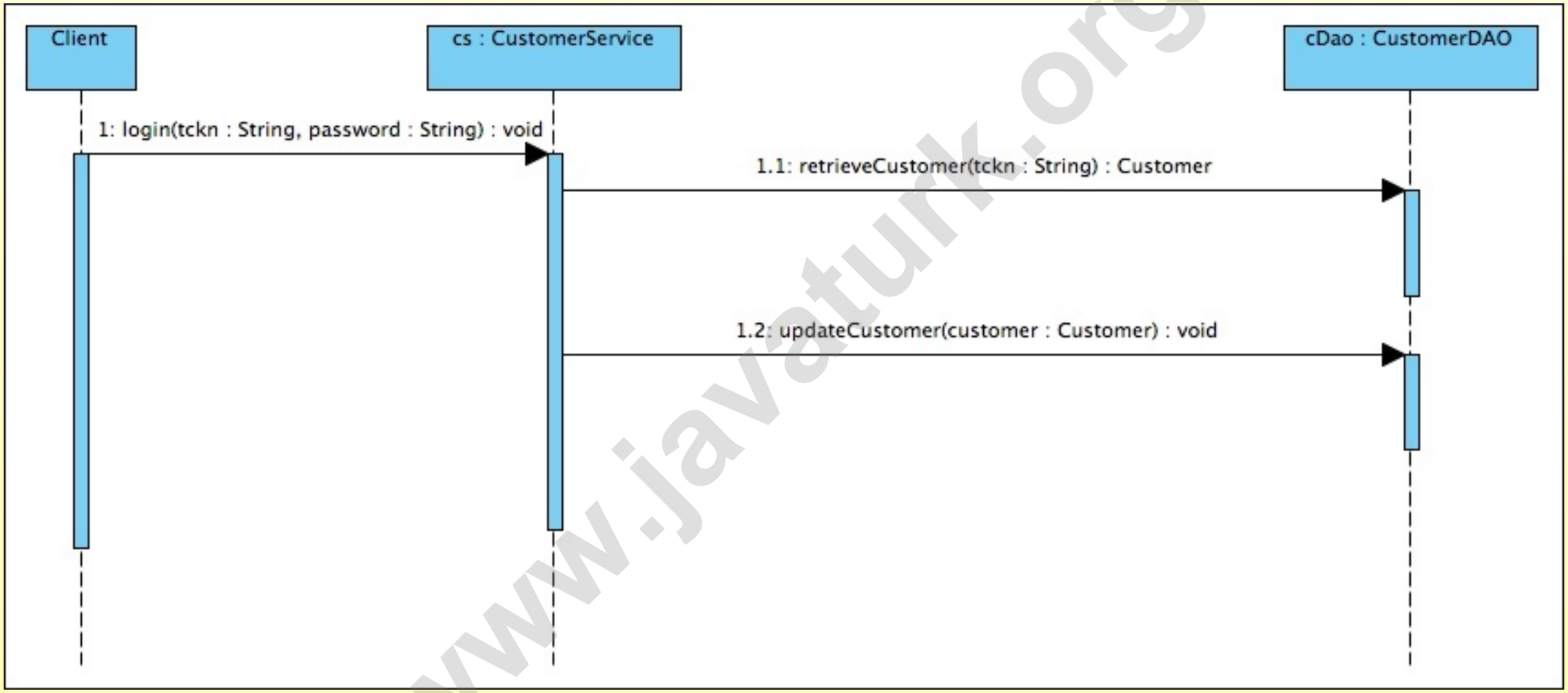
- Bir metotta sadece bir sorumluluk, bir iş ya da bir sürecin bir adımı yerine getirilmeli.
  - Metot, tekrar kullanımın en temel öğesidir.
  - Tekrar kullanılma ihtimali olan her bloğu metot yapın.
- Metotlar ortalama olarak 10 satırı geçmemeli,
- Metotlar olabildiğince az parametre almalı,
- Metotlar olabildiğince az karar mekanizmasına sahip olmalı (cyclomatic complexity).

# Odaklı Metotlar

- Aşağıdaki metodu nasıl yazarız?

```
public void login(String tckn, String password) throws
    NoSuchCustomerException,CustomerAlreadyLoggedException,
    WrongCustomerCredentialsException,
    MaxNumberOfFailedLoggingAttemptExceededException,
    CustomerLockedException;
```

```
public void login(String tckn, String password) throws NoSuchCustomerException,
                  CustomerLockedException, CustomerAlreadyLoggedException,
                  WrongCustomerCredentialsException,
                  MaxNumberOfFailedLoggingAttemptExceededException {
    Customer customer = customerDao.retrieveCustomer(tckn);

    // If passwords match, customer hasn't already been locked nor logged in
    // Customer loggs in and it is now currentCustomer
    if (customer.getPassword().equals(password) &
                          !customer.isLocked() & !customer.isLoggedIn()) {
        // Database is updated when a customer logs in.
        customer.setLoggedIn(true);
        if (customerDao.updateCustomer(customer))
        currentCustomer = customer;
        loginAttemptCount = 0;
    } else if (customer.isLoggedIn()) {
        throw new CustomerAlreadyLoggedException("Customer is already logged in.
                                                  Please first log out.");
    } else if (customer.isLocked()) {
        throw new CustomerLockedException("Customer is locked.
                                           Please consult your admin.");
    } else if (!customer.getPassword().equals(password)) {
        loginAttemptCount++;
        if (loginAttemptCount == Integer.parseInt(ATMProperties.getProperty(
                                                  "customer.maxFailedLoginAttempt"))) {
            customer.setLocked(true);
            customerDao.updateCustomer(customer);
            throw new MaxNumberOfFailedLoggingAttemptExceededException("Max number of
                                                        login attempt reached: "
                                                        + loginAttemptCount);
        }
        throw new WrongCustomerCredentialsException("TCKN/password is wrong.");
    }
}
```



```
private void checkIfCustomerAlreadyLoggedIn(Customer customer) throws CustomerAlreadyLoggedException {
    if (customer.isLoggedIn()) {
        throw new CustomerAlreadyLoggedException("Customer is already logged in.
                                                  Please first log out.");
    }
}
```

```
private void checkIfCustomerLocked(Customer customer) throws CustomerLockedException {
    if (customer.isLocked()) {
        throw new CustomerLockedException("Customer is locked. Please consult your admin.");
    }
}
```

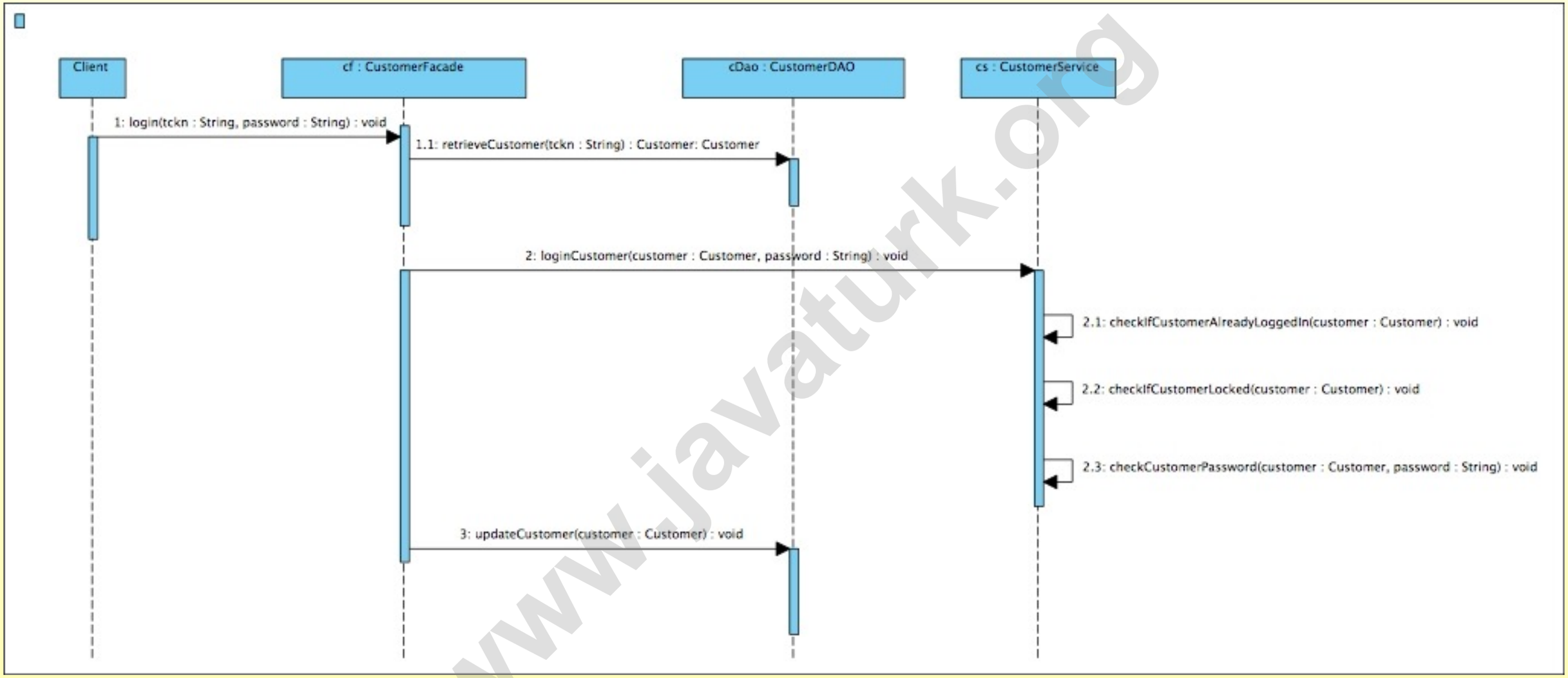
```
private void checkCustomerPassword(Customer customer, String password) throws
        MaxNumberOfFailedLoggingAttemptExceededException, WrongCustomerCredentialsException {
    if (!customer.getPassword().equals(password)) {
        loginAttemptCount++;
        checkLoginAttempCount(customer);
        throw new WrongCustomerCredentialsException("Wrong password!");
    }
}
```

```
private void checkLoginAttempCount(Customer customer) throws
                                                MaxNumberOfFailedLoggingAttemptExceededException{
    if (loginAttemptCount == Integer.parseInt(ATMProperties.getProperty(
                                                    "customer.maxFailedLoginAttempt"))) {
        lockCustomer(customer);
    }
}
```

```
private void lockCustomer(Customer customer) throws MaxNumberOfFailedLoggingAttemptExceededException{
    customer.setLocked(true);
    throw new MaxNumberOfFailedLoggingAttemptExceededException("Max number of login
                                                        attempt reached: "
                                                        + loginAttemptCount);
}
```

```
public void login(String tckn, String password) throws
   NoSuchCustomerException, CustomerAlreadyLoggedException,
   WrongCustomerCredentialsException,
   MaxNumberOfFailedLoggingAttemptExceededException,
   CustomerLockedException{

      Customer customer = customerDao.retrieveCustomer(tckn);
      loginCustomer(customer, password);

}
```

```
private void loginCustomer(Customer customer, String password) throws
   CustomerAlreadyLoggedException,
   WrongCustomerCredentialsException,
   MaxNumberOfFailedLoggingAttemptExceededException,
   CustomerLockedException{

       boolean login = false;
      checkIfCustomerAlreadyLoggedIn(customer);
      checkIfCustomerLocked(customer);
      checkCustomerPassword(customer, password);
      customerDao.updateCustomer(customer);

}
```



# Sınıf

- Sınıflar sadece bir kavramla ilgili sorumlulukları bir araya getirmeli.
- Sınıfların karmaşıklığı sahip olduğu sorumluluklarla ölçülür.
  - Çok metot, çok sorumluluk, karmaşık arayüz, karmaşık sınıf.
- Durumunun (state) karmaşıklığı da buna bağlıdır.
  - Çok nesne değişkeni, karmaşık durum, karmaşık sınıf.

# Sorumluluk Kategorizasyonu

- Sorumlulukların da farklı çeşitleri olduğu, ve bu çeşitliliğe göre gruplanması gerektiğini unutmayın:
  - Kullanan sınıflar (client) açısından: Farklı kullanıcı sınıflar farklı arayüzlere erişmeli.
  - Sorumluluğun mahiyeti açısından: Farklı tabiattaki sorumlulukla, aynı kavramla ilgili olsalar bile bir arada olmayabilirler.
- Tasarım şablonları, aynı kavramla ilgili nasıl farklı dikey sorumluluk grupları olabileceğini öğretir.

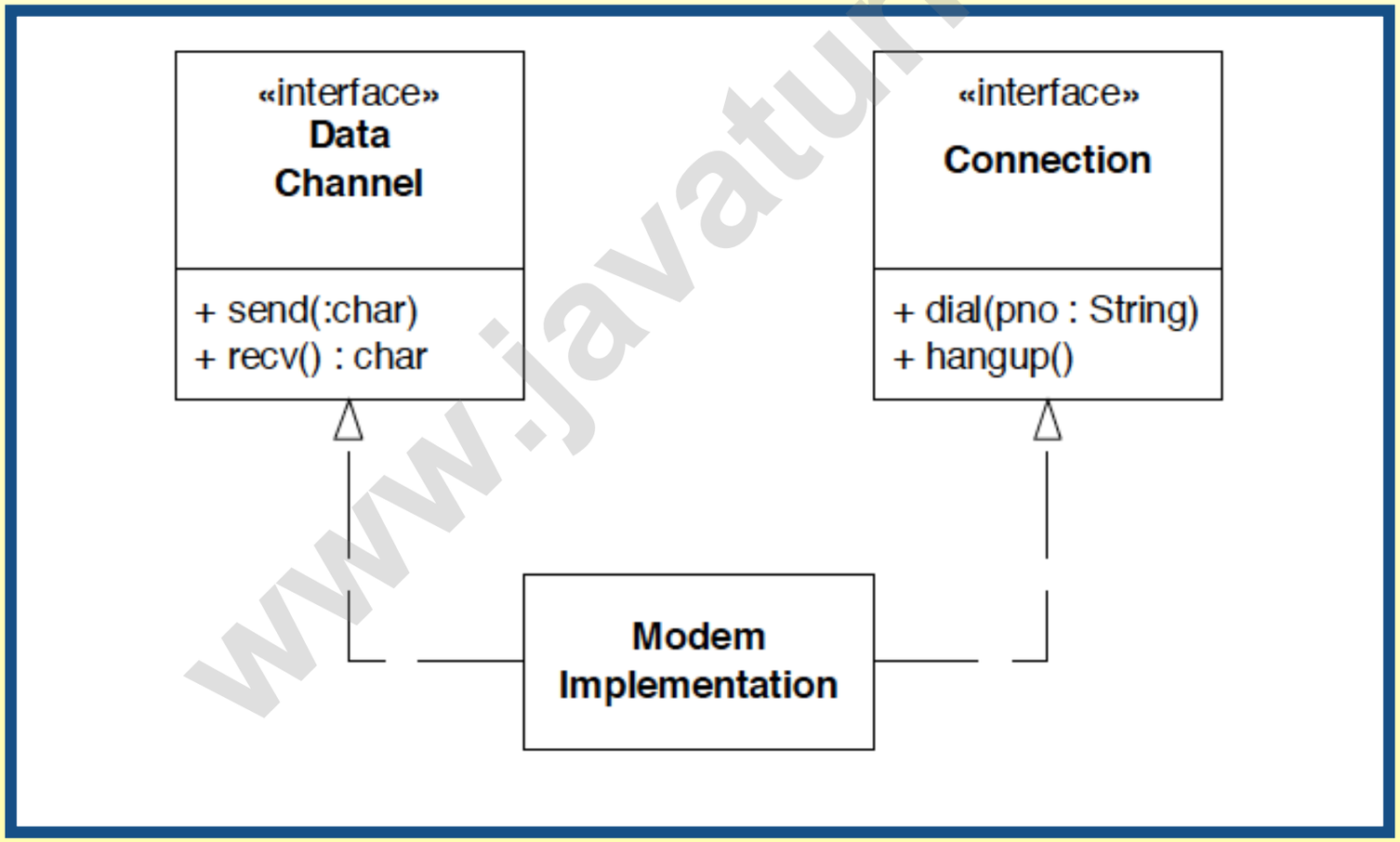
# Modem - I

- Modem arayüzünü tek sorumluluk prensibi açısından düşünelim.
- Modemin hizmet verdiği farklı clientlar olabilir mi?

```
public interface Modem{
      public void dial(String pno);
      public void hangup();
      public void send(char c);
      public char receive();
}
```

# Modem - II

- Veri almak ve göndermek ile bağlantı kurmak ve bağlantıya son vermek iki ayrı sorumluluk alanı, dolayısıyla ayrılmalı.



# Account - I

- Account arayüzünü tek sorumluluk prensibi açısından düşünelim.
  - Ne tür sorumluluk gruplarından bahsedebiliriz?

```
public interface Account{
     public Date getCreationDate()
     public double getBalance();
     public double getInterestRate();
     public Customer getOwner();
     public List<Transaction> getHistory(Date from, Date to);
     public void print();
     public void save();
}
```

# Account - II

```
public class Account implements Printable, Persistent, Transactional{
     public Date getCreationDate()
     public double getBalance();
     public double getInterestRate();
     public Customer getOwner();
}
```

```
public interface Printer{
     public void print(Printable printable);
}
```

```
public interface Dao{
     public void save(Persistent persistent);
}
```

```
public interface TransactionService{
      public List<Transaction> getHistory(
                    Transactional transactional,Date from, Date to);
}
```

# Kısa Kod İçin Budama

- Odaklı dolayısıyla da küçük soyutlamalar bir seferde elde edilemez.
- Soyutlamalarınızı devamlı surette budayarak kısa ve küçük soyutlamalara ulaşabilirsiniz.
  - Refactoring!
- Unutmayın, mükemmelliğe ekleyerek değil, çıkararak ulaşabilirsiniz.

# Kısa Kod İçin Budama

- Pascal “The Provincial Letters” isimli eserinin 16. mektubunda şöyle der:

Bu mektubu böyle uzun yapabildim çünkü kısa tutmak için zamanım yoktu.

(“I have only made this letter longer because I have not had the time to make it shorter.”)

# Tekrarsız Kod

- Odaklanmak aynı zamanda, bir şeyi sadece bir yerde yapmak, birden fazla yerde yapmamak, tekrardan kaçınmak, tekrarsız kod yazmak demektir.

- Tekrarsız kod, ancak **tasarım**la ve sonrasında gerektiğinde **refactoring** ile elde edilir.

- Merkezi ve ortak bir tasarımla şekillenmeyen, sonrasında ciddi bir koordinasyon ve iletişime ortamında yazılmayan kodların pek çok yerde tekrara sahip olması kaçınılmazdır.

# Don’t Repeat Yourself - I

- Don’t Repeat Yourself, DRY, tekrarsız kod yazmanın sloganıdır:

Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.

Her bilgi parçasının sistemde, tek, açık, güvenilir bir ifadesi olmak zorundadır.

- Davranış, bilgi, kısıt, vs. ne olursa olsun sadece bir yerde olsun.

# Don’t Repeat Yourself - II

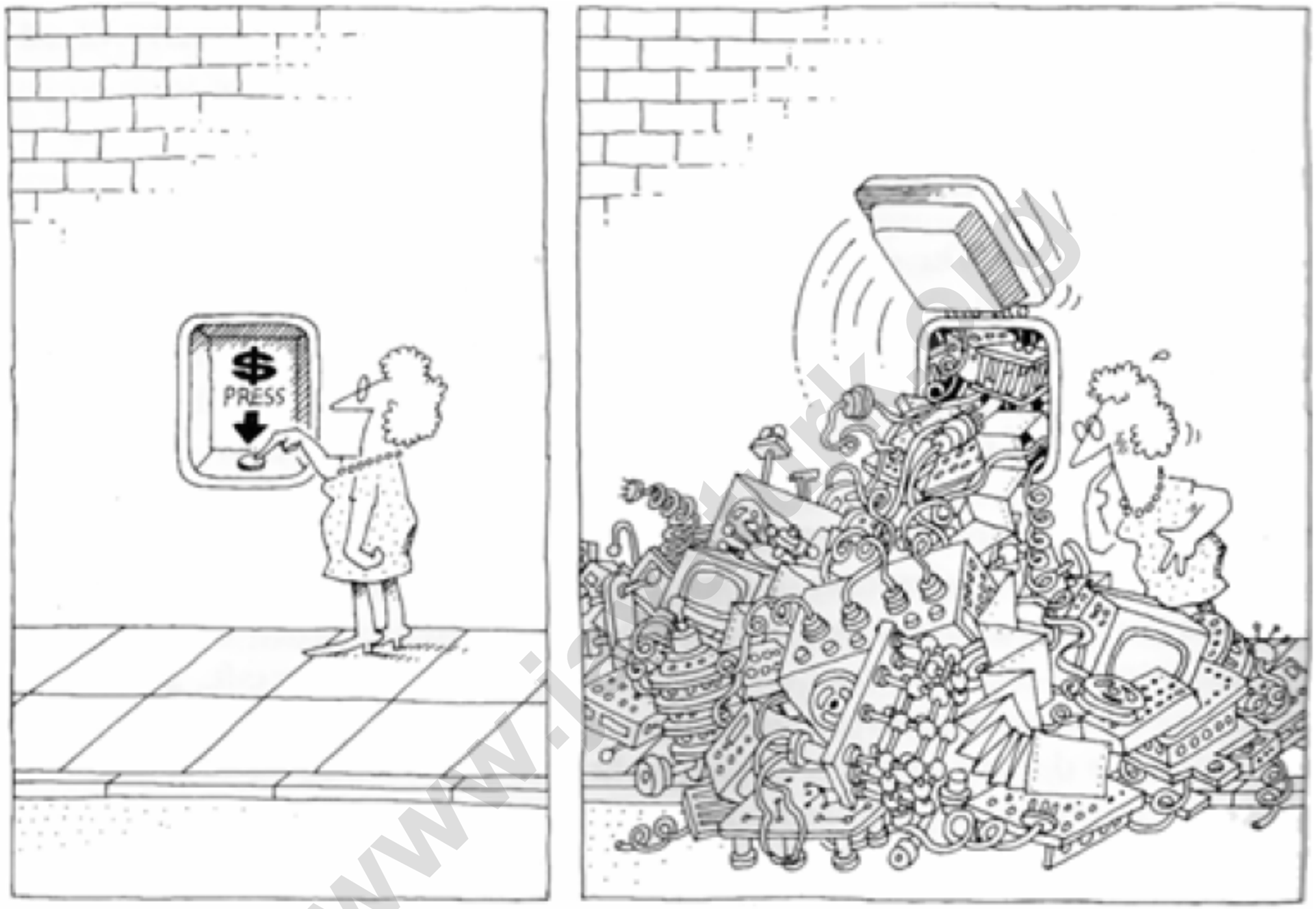
- DRY prensibini sadece tasarım ile gerçekleştirmek imkansızdır.
- Kodlarken refactoring ile tekleştirmeye gidilir.
- Tekleştirme, cut-paste ile sağlanır.
  - Birbirine benzeyen kodlar, tek bir yapı altında toplanır.
  - Tekrar kullanım (reusability) artar.
- Copy-paste kullanmayın, cut-paste kullanın!

```
public void authenticateUser(RegistrationForm registrationForm,
                             HttpServletRequest request,
                             Locale locale) throws JVTException {

        String uid = utils.generateUUID();
        String salt = utils.generateSalt(32);
        String accountId = utils.generateUUID();

        JVTUser user = new JVTUser(
                registrationForm.getEmail().trim(), null,
                true, true, true, true, AuthorityUtils.NO_AUTHORITIES,
                uid, registrationForm.getEmail().trim(), salt,
                registrationForm.getName().trim(),
                registrationForm.getSurname().trim(),null, accountId, true,
                true, null, true, new Date(),null
        );

        if ( executeInner(registrationForm, locale, user) ){

            authenticateFastRegisterUser(registrationForm,request);

            mailSender.sendMail(user.getEmail(), user.getUsername(),
                                MailType.VERIFICATION,
            utils.getPlatformPropertyFromFile("email_baseUrl"), locale);

            auditLogDao.insert(new AuditLog(user.getUid(), user.getUsername(),
                               utils.getRemoteIp(), utils.getSid(),
                               AuditType.REGISTRATION,
                               AuditSubtype.MAIL_SENT, ResponseCode.SUCCESS,
                               new Date(), null)
            );
        }
}
```

```
public void authenticateUser(RegistrationForm registrationForm,
                             HttpServletRequest request,
                             Locale locale) throws JVTException {

   JVTUser user = JVTUser.createUnauthenticatedUser(registrationForm);

   if(executeInner(registrationForm, locale, user)){

      authenticateFastRegisterUser(registrationForm, request);

      mailSender.sendMail(user, locale);

      auditLogDao.insert(
          AuditLog.createAuditLogForUserAuthentication(user));
   }
}
```

# Temiz Kod Nasıl Elde Edilir?

- Öncelikle kurumsal ve bireysel farkındalık ve bilgi,
  - Kurumsal süreç, bireysel işler,
- Azıcık düşünme ve tasarım,
- Bol iletişim, paylaşma ve tartışma,
- Refactoring için cesaret ve zaman,
- PDM, SonarCube gibi kod kalitesi araçları,
- Kod review, formal ya da peer review süreci.




The task of the software development team
is to engineer the illusion of simplicity.

# Dinlediğiniz için teşekkür ederim.

Bu sunuma www.javaturk.org adresinden ulaşabilirsiniz.