Java Kodunuzun Nesne-Merkezli Olmadığının 10 İşareti

Akın Kaldıroğlu
akin@javaturk.org

5 Ekim 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)*

Neden?

- Neden “Java Kodunuzun Nesne-Merkezli Olmadığının 10 İşareti”?
  - CDI, fonksiyonel programming vb. yapılara sahip Java dili varken,
  - JVM üzerinde Scala, Groovy vb. bir sürü dilciğe sahipken,
  - EJB, JPA, JSF, Spring gibi yetkin frameworkler kullanıyorken,
  - Son derece gelişmiş JVM, AppServer, IDE vb. yapılar elimizin altındayken,
- Hala mı o *eski nesne-merkezli olup-olmama* tartışmaları?

Kadim Sorular?

- Evet tüm bunlara sahibiz ama şu soruları hala sorabiliriz?

- Tüm bu araçlar yazılım geliştirmemizi kolaylaştırıyor mu?

- Tüm bu araçlar geliştirdiğimiz yazılımın kalitesini arttırıyor mu?

Hız ve Kalite

- Tüm bu gelişmeler yazılımı geliştirme hızımızı arttırıyor.
- Ama geliştirdiğimiz yazılımın kalitesi hala bizim sorumluluğumuzda,
  - Tüm bu yapılar bize daha kaliteli yazılım geliştirmemiz için imkan sağlıyorlar ama bu imkanları kullanmak bizim sorumluluğumuzda.

Problem Nedir?

- Java projelerinde envai çeşit dilcikler, frameworkler, araçlar vs. kullanılmasına rağmen hala en temel nesne-merkezli prensipler atlanıyor.

- Çok sıklıkla karşılaştığım ve nesne-merkezli yaklaşıma ters olan bu durumları Java bağlamında listeledim.

- Bu maddelerin ciddi bir kısmı **cohesion** ve **coupling** ile ilgili.

10 İşaret

- "Java kodunuzun nesne-merkezli olmadığının 10 işareti"
- Listedeki maddeler geneldirler, özel durumlarda farklı şeyler söylenebilir.
- Maddeler doğrudan kod kalitesinden ziyade kodu nesne-merkezli olmasının önündeki engellere odaklanmaktadır.

10 Numara

- Sıra dışı durumları ifade etmek için Exception nesneleri oluşturmamak.
  - No Exception objects to represent exceptional situations.
  - Her şeyi nesnelerle ifade ediyorsak, sıra dışı durumları da Exception nesneleriyle ifade etmeliyiz.
- Çünkü sıra dışı durumlar iş mantığının bir parçasıdır.
- Sıra dışı durumları görmezden gelmek ya da *int* ya da *String* vb. tiplerle, return değeri olarak ifade etmek nesneden uzaklaşmaktır, anlam problemine ve karmaşık programlara yol açar.

10 Numara

- Fırlattığınız sıra dışı durum nesneleri, metotların arayüzlerinin yani anlamlarının bir parçasıdır,
  - Yeterliyse APIlerdeki Exception nesneleri
  - Aksi taktirde iş modelimizi sıra dışı durumlara özgü kendi Exception nesnelerimizle zenginleştirmeliyiz.
  - Kullanıcıya bilgi verme, loglama vb. ihtiyaçlar yanında aslen sistemin çalışmasını devam ettirme amaçlı kullanılmalıdır.

10 Numara

```
public interface UserDao{

   public User getUser(String tckn) throw Exception;

   public User saveUser(User user) throw Exception;

}
```

```
public interface UserDao{

   public User getUser(String tckn) throw NoSuchUserException;

   public User saveUser(User user) throw DuplicatedUserException;

}
```

9 Numara

- Aşırı miktarda karar ifadelerine (“if/switch” vb.) sahip metotlar.

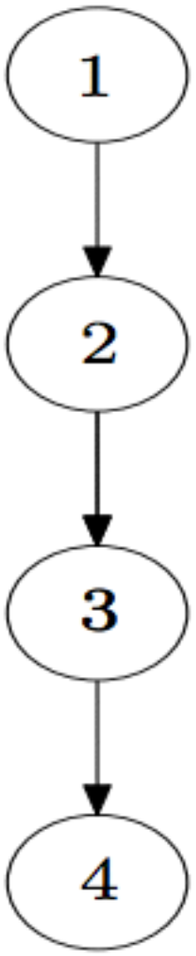
  - Methods that include excessive number of decision (if/switch etc.) statements.

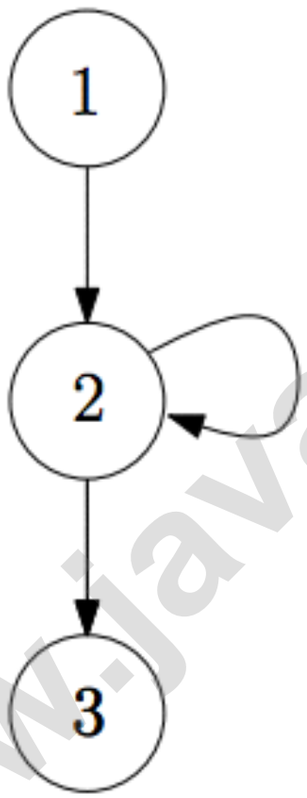
- Karar yapılarının oluşturduğu karmaşıklık, *cyclomatic complexity*, metodun anlaşılmasını ve testini zorlaştırır.

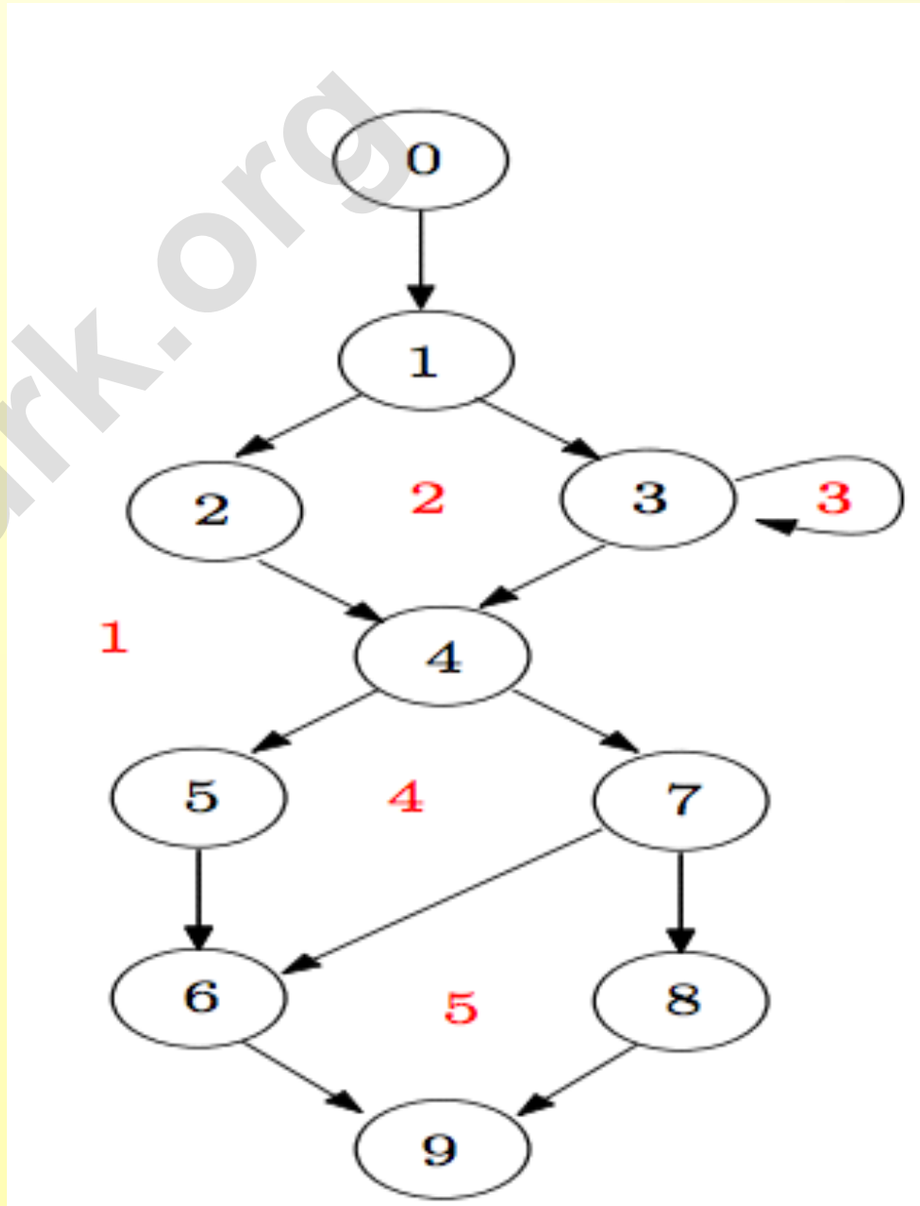
- Aşırı karar, muhtemelen pek çok farklı konuyu bir arada yönetmek anlamına geldiğinden, ekseriyetle cohesionu düşürür.

- Böyle metotlar muhtemelen çok sayıda parametre alırlar.

9 Numara







9 Numara

- Yüksek CC'nin sebebi, çoğu zaman düzgün soyutlamalar yaptığımızda farklı metot ve sınıflara koyulacak yapıları, bir arada halletmeye çalışmaktır.
  - Strategy, Command, Proxy, State gibi tasarım kalıpları bu duruma çözümdür.
- Bazen de bir "is-a" hiyerarşisi kurmamak yüksek CC'ye sebep olur.
  - Bir sınıftaki "tip" gösteren int ya da String bir değişken metotlarınızda pek çok "if/switch"e sebep olur.
  - Bir tip hiyerarşisi oluşturup, polymorphic davranışla kararı JVM'e bırakmak olması gereken davranıştır.

9 Numara

```
public class JavaturkUser{

   private String username;
   private String password;
   private String name;
   private String lastname;
   private int type;

   public void register(){
      if(type == 1){
         …
      }
      else if(type == 2){
         …
      }
      …
   }
}
```

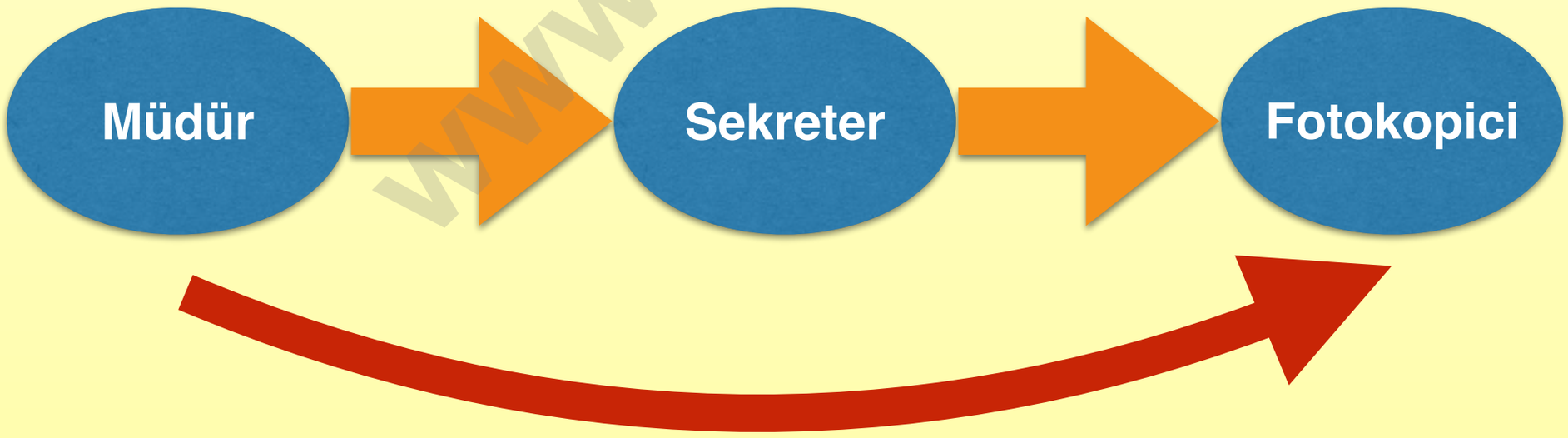
8 Numara

- Demeter prensibini ihlal eden metotlar.
  - Methods that break the Demeter law.
  - Demeter kanunu, her yazılım biriminin sadece ve sadece çok yakınındakilerle haberleşebileceğini, uzaktakileri bilmemesi gerektiğini söyler.
- Yabancılarla konuşma ya da az bilgi prensibi (principle of least information)!
- Bu prensip yapılarımızı yüksek couplingten korur!

8 Numara

- Buna göre bir nesnenin metotlarında, kendisi üzerinde metot çağrısı yapabileceği nesneler ancak şunlar olabilir:
  - O nesnenin instance variableları,
  - O nesnenin metotlarına geçilen parametre nesneler,
  - O nesnenin o metotunda oluşturulan nesneler.
- Yani bir nesne ancak arkadaşlarıyla konuşur, yabancılarla konuşmaz.

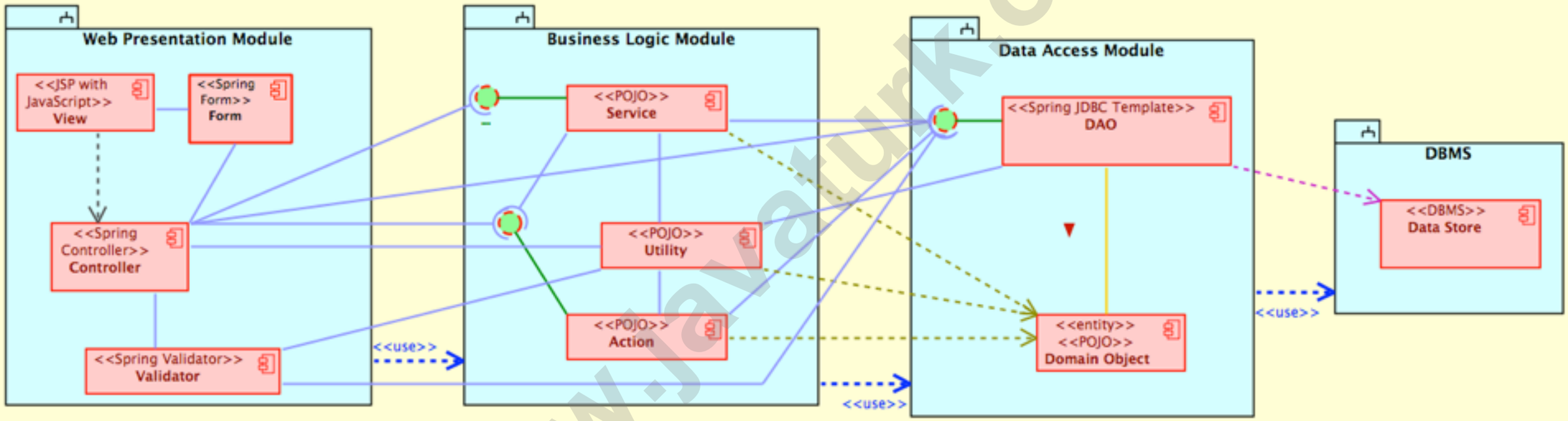
```
public class A{
   private B b;

   public void f(C c){
      b.g();            // 1- Yapılabilir
      c.u();            // 2- Yapılabilir
      D d = new D()
      d.v();            // 3- Yapılabilir
      E e = c.w(); // Yapma bunu!!!
      e.z();           // E'den iş isteme, C'den, E'den iş
                       // istemesini talep et.
   }
}
```



8 Numara

- Developerlar, bu prensibi göz önüne almadan, gelişi güzel coupling yaratma eğilimdedirler,
  - Çünkü onlar amaçlarını en kısa yoldan gerçekleştirme isterler.
- Bu da bağımlılığı yüksek dolayısıyla da kaotik yazılımlara sebep olur.

8 Numara



7 Numara

- Karmaşık metotlar.
  - Complex methods.
  - Metotların karmaşıklığına etki eden temelde 3 nokta vardır:
    - Satır sayısı
    - Parametre sayısı
    - Karar karmaşıklığı (cyclomatik complexity)

7 Numara

- Metotların satır sayısı ortalama olarak 10'u geçmemelidir.

- Her metot, tekrar kullanılabilir bir tek işi yerine getirmelidir.

- Koordine eden metotlar, koordine edilen metotları çağırarak daha büyük bir bağlamda ama tek bir işi yerine getirmelidir.

7 Numara

- Argüman sayısı arttıkça hem metotların hem de sınıflarının cohesionları düşer, couplingleri artar.
  - Makul sayıda (<5) argümanlı metotlar yazmak ve geçilen argüman sayısını ortalama olarak 1 civarında tutmak sağlıklıdır.
  - Constructorlar, factory metotları, façade ve remote arayüzler bunun istisnaları olabilir.
- Çok parametreli metotları ya bölün ya da parametreleri nesne olarak soyutlayıp geçin.

```
public JVTUser(String username, String password, boolean enabled, boolean accountNonExpired,
               boolean credentialsNonExpired, boolean accountNonLocked,
               Collection<? extends GrantedAuthority> authorities,
               String uid, String email, String salt, String name, String surname,
               String msisdn, String accountId, boolean verified, boolean newUser,
               String welcomeMessage, boolean verifyReminder,
               Date registrationDate, Date verificationDate)
```

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

        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
        );
```

```
public static JVTUser createNonlockedUnverifiedUser(RegistrationForm registrationForm)
```

```
JVTUser user = JVTUser.createNonlockedUnverifiedUser(registrationForm);
```

6 Numara

- Çok sayıda metoda sahip olan sınıflar.
  - Classes with lots of methods.
  - Fazla sayıda metoda sahip olan sınıflar fazla sorumluluğa sahiptir.
    - Sınıflar sadece bir kavramla ilgili sorumlulukları bir araya getirmeli.
- Bu tür sınıflar çok şeyi bir araya getirir bu yüzden de cohesionu düşük, couplingi yüksek olur.

```
public class SuperDashboard extends JFrame implements MetaDataUser{
 public String getCustomizerLanguagePath()
 public void setSystemConfigPath(String systemConfigPath)
 public String getSystemConfigDocument()
 public void setSystemConfigDocument(String
                                     systemConfigDocument)
 public boolean getGuruState()
 public boolean getNoviceState()
 public boolean getOpenSourceState()
 public void showObject(MetaObject object)
 public void showProgress(String s)
 public boolean isMetadataDirty()
 public void setIsMetadataDirty(boolean isMetadataDirty)
 public Component getLastFocusedComponent()
 public void setLastFocused(Component lastFocused)
 public void setMouseSelectState(boolean isMouseSelected)
 public boolean isMouseSelected()
 public LanguageManager getLanguageManager()
 public Project getProject()
 public Project getFirstProject()
 public Project getLastProject()
 public String getNewProjectName()
 public void setComponentSizes(Dimension dim)
 public String getCurrentDir()
 public void setCurrentDir(String newDir)
 public void updateStatus(int dotPos, int markPos)
 public Class[] getDataBaseClasses()
 public MetadataFeeder getMetadataFeeder()
 public void addProject(Project project)
 public boolean setCurrentProject(Project project)
 public boolean removeProject(Project project)
 public MetaProjectHeader getProgramMetadata()
 public void resetDashboard()
 public Project loadProject(String fileName,
                            String projectName)
 public void setCanSaveMetadata(boolean canSave)
 public MetaObject getSelectedObject()
 public void deselectObjects()
 public void setProject(Project project)
 public void editorAction(String actionName,
                          ActionEvent event)
 public void setMode(int mode)
 public FileManager getFileManager()
 public void setFileManager(FileManager fileManager)
 public ConfigManager getConfigManager()
 public void setConfigManager(ConfigManager configManager)
 public ClassLoader getClassLoader()
 public void setClassLoader(ClassLoader classLoader)
 public Properties getProps()
 public String getUserHome()
 public String getBaseDir()
 public int getMajorVersionNumber()
 public int getMinorVersionNumber()
 public int getBuildNumber()
 public MetaObject pasting(MetaObject target,
           MetaObject pasted, MetaProject project)
 public void processMenuItems(MetaObject metaObject)
 public void processMenuSeparators(MetaObject metaObject)
 public void processTabPages(MetaObject metaObject)
 public void processPlacement(MetaObject object)
 public void processCreateLayout(MetaObject object)
 public void updateDisplayLayer(MetaObject object,
                                int layerIndex)
 public void propertyEditedRepaint(MetaObject object)
 public void processDeleteObject(MetaObject object)
 public boolean getAttachedToDesigner()
 public void processProjectChangedState(boolean
                                        hasProjectChanged)
 public void processObjectNameChanged(MetaObject object)
 public void runProject()
 public void setAçowDragging(boolean allowDragging)
 public boolean allowDragging()
 public boolean isCustomizing()
 public void setTitle(String title)
 public IdeMenuBar getIdeMenuBar()
 public void showHelper(MetaObject metaObject,
                        String propertyName)
 // ... many non-public methods follow ...
}
```

6 Numara

- Servis nesneleri ise couplingi yüksek olma eğilimindedirler, daha ufak servis nesnelerine bölünmelidir.

- İş alanı nesneleri ise doğrudan veri tabanındaki tablolara karşılık gelmemeli, uygun bir şekilde modellenmelidir.

- Devasa utility nesneleri ise alt nesnelere bölünmelidir,
  - SystemUtil yerine StringUtil, SecurityUtil, HttpUtil gibi.

6 Numara

- Çok fazla sayıda metoda sahip sınıfların çok fazla değişkeni de olabilir.

- Çok sayıda değişken düşük cohesion, yüksek coupling demektir.

6 Numara

```
public class MyService{
    private UserDao userDao;
    private RegisteredEmailDao registeredEmailDao;
    private AuthorityDao authorityDao;
    private Utils utils;
    private MailSender mailSender;
    private PasswordEncoder passwordEncoder;
    private SaltSource saltSource;
    private AuditLogDao auditLogDao;
    private EraseUserAction eraseUserAction;
    private AuthenticationUtils authenticationUtils;
    private BankDao bankDao;
    private BankParamDao bankParamDao;
    private KeyAliasDao keyAliasDao;
    private JavaTurkCipher javaturkCipher;
    private BankServiceFactory bankServiceFactory;
    private OtpDao otpDao;
    private BankUtils bankUtils;
    private FormUtils formUtils;
    private EventLogAction eventLogAction;
    private BankServiceLogAction logAction;
…
}
```

6 Numara

```
public class JavaturkUser{
    private String username;
    private String password;
    private String name;
    private String lastname;
    private int type;
     private Set<GrantedAuthority> authorities;
     private boolean accountNonExpired;
     private boolean accountNonLocked;
     private boolean credentialsNonExpired;
     private boolean enabled;
     private String uid;
     private String email;
     private String salt;
     private String name;
     private String lastname;
     private String msisdn;
     private String accountId;
     private boolean verified;
     private boolean newUser;
     private String welcomeMessage;
     private boolean verifyReminder;
     private Date registrationDate;
     private Date verificationDate;
     private String registrationType;
...
}
```

5 Numara

- Aralarında benzerlik ilişkisi olan nesneleri modellemek için kalıtım kullanmamak.
  - No inheritance hierarchy to model is-a relationships between objects.
- Eğer kalıtım/inheritance kullanmıyorsanız, polymorphismden yararlanamazsınız
- Bu da nesneleriniz üzerindeki iş mantığını karmaşıklaştırır, “if/switch” sayısını arttırır.

5 Numara

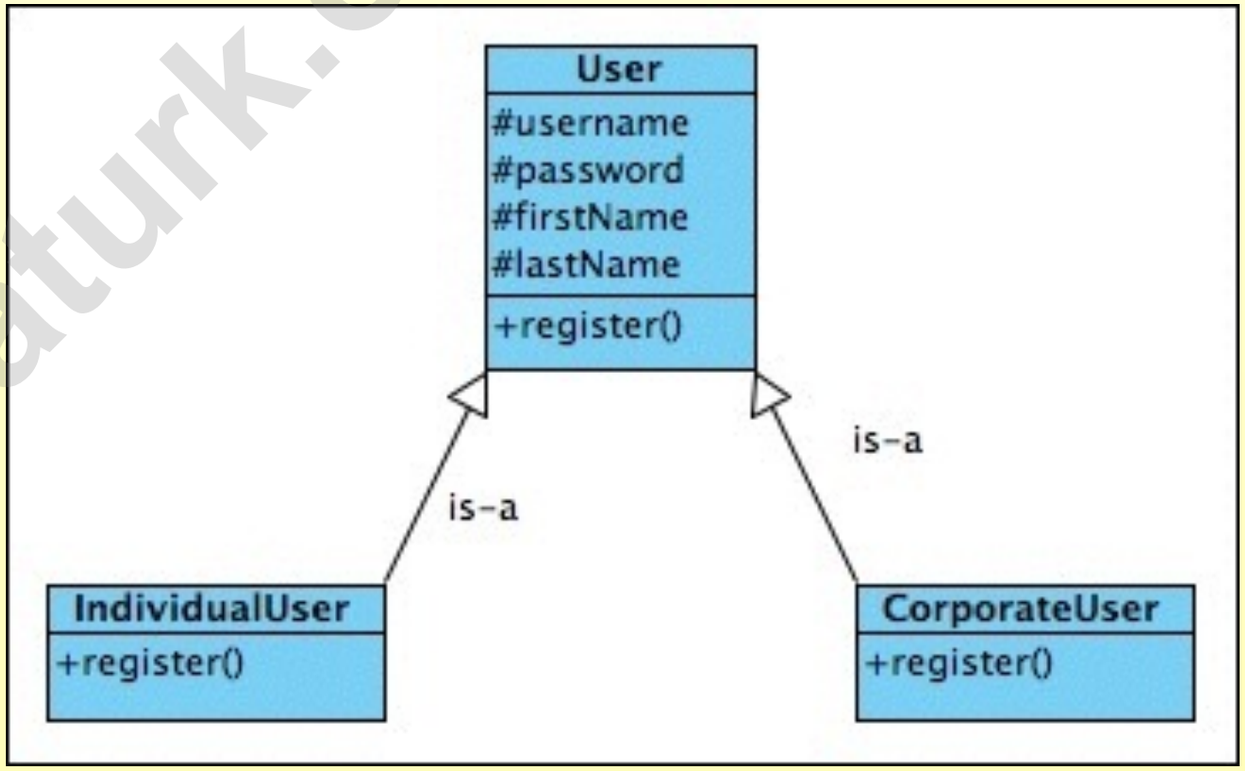
```
public class User{

   private String username;
   private String password;
   private String firstName;
   private String lastName;
   private int type;

   public void register(){
      if(type == 1){
         doThis();
      }
      else if(type == 2){
         doThat();
      }
      …
   }
}
```

5 Numara

```
public abstract class User{

    protected String username;
    …

    public abstract void register();
}
```

```
public class IndividualUser extends User{

    …

    public void register(){
        doThis();
    }
}
```

```
public class CorporateUser extends User{

    …

    public void register(){
        doThat();
    }
}
```



4 Numara

- Kodunuzda hiç ya da çok az sayıda interface olması.
  - No or very few interfaces in the code.
  - Nesne-merkezli programlama, “responsibility-driven”dır.
  - Interface kullanmamak, sorumlulukları-davranışları soyutlamamak demektir.
  - Hiç ya da az interface, muhtemelen data-driven düşünme anlamına gelir.

4 Numara

- Bir yazılım projesinde tasarımın en temel belirtisi interfacelerdir.

- Tasarıma interface ile başlanıldığında elde etdilen sınıfların cohesionunu yüksek, couplingini az olma eğilimindedir.

- Örneğin, interfacelerdeki metotlar, sınıfların üzerine koyuverilenlerden daha az argüman alma eğilimindedirler.

3 Numara

- Kodunuzda hiç alan nesnesinin olmaması ya da olsa bile çok yetersiz olmaları.
  - No or very thin domain objects in the code.
  - İş mantığınızı alan nesneleri ile ifade etmiyorsanız, sadece primitivler ve collectionları kullanıyorsunuz demektir.
  - Bu durum anlam ve anlaşılma problemine sebep olur.
  - İş mantığınız ise servis, utility vb. nesnelere dağılmış demektir.

3 Numara

```
public class Date {
   private int day;
   private String month;
   private int year;

   public int getDay() {…}

   public void setDay(int day) {…}

   …

   public String toString() {
      return "Date: " + day +
             " - " +
             month + " - " + year;
   }
}
```

- Date sınıfını, cohesion ve coupling açısından ele alalım.
  - Nedir sizce bu nesnenin cohesionu?
  - Ve diğer nesnelerle olan ya da olacak olan irtibatı, couplingi nedir?

2 Numara

- Yüksek içerik bağımlılığı yani bir başka sınıfın nesne değişkenlerine aşırı sayıda doğrudan erişim ya da aşırı *set* ve *get* metodu çağrısı.

  - High content coupling, i.e. direct access to the instance variables or excessive number of calls to setters and getters of another class.

- Aşırı sayıda bir başka sınıfın nesne değişkenlerine doğrudan erişim ya da aşırı *set* ve *get* metodu çağrısı, yüksek bir bağımlılık oluşturur.

2 Numara

- İçerik bağımlılığı, hizmet yerine bilgi alış-verişi demektir.

- Nesnelerin varlık sebebi bilgi alıp-vermek değildir.

- Nesneler hizmet için vardırlar, veri yerine getirilecek hizmet için vardır.

1 Numara

- Sınıflarda aşırı miktarda statik metot kullanımı.
  - Excessive use of static methods in classes.
- Statik, nesne kullanımının alternatifidir.
  - Fakat statik kullanım standart dışı kullanımdır, genel değildir, bazı durumlara mahsus olmalıdır.
- Ne kadar çok statik kullanırsanız o kadar az nesne oluşturursunuz.

1 Numara

```
public class MyClass {
   private int i;

   public static void main(String[] args){

      System.out.println(i);

      print();

}

   public void print() {
      System.out.println(i);
   }
}
```

1 Numara

- Statik yapılar, nesne oluşturmanın anlamsız olduğu haler içindir.
- Sınıfın durumuna bağlı olmayan metotlar,
  - Utility metotları örneğin
- Singleton kullanımına alternatif durumlar

10 İşaret

1. **Sınıflarda aşırı miktarda statik metot kullanımı.**
2. **Yüksek içerik bağımlılığı.**
3. **Kodunuzda hiç alan nesnesinin olmaması ya da olsa bile yetersiz olmaları.**
4. **Aralarında benzerlik ilişkisi olan nesneleri modellemek için kalıtım kullanmamak.**
5. **Kodunuzda hiç ya da çok az sayıda arayüz olması.**
6. **Çok sayıda metoda sahip olan sınıflar.**
7. **Karmaşık metotlar.**
8. **Demeter prensibini ihlal eden metotlar.**
9. **Aşırı sayıda karar ifadelerine sahip olan metotlar.**
10. **Sıra dışı durumları ifade etmek için Exception nesneleri oluşturmamak.**

Bazı Noktalar

- Değerleri hard-coded yapmaktan hiç doküman yazmamaya kadar pek çok farklı **kötü kod işareti** mevcuttur.

- Ama ben sadece nesne kullanımıyla ilgili olanları listeledim.

- Listeyi Java bağlamında yaptım ama hepsi diğer nesne-merkezli diller için de geçerlidir.

Dinlediğiniz için teşekkür ederim.

Bu sunuma www.javaturk.org adresinden ulaşabilirsiniz.