JUG ISTANBUL

JAVA 9 & JAVA 10 WORKSHOP

EĞİTMENLER

TANER DİLER

GÖKALP GÜRBÜZER

ALTUĞ B. ALTINTAŞ

TAYLAN KURT

RESOURCES

https://www.sitepoint.com/ultimate-guide-to-java-9/

https://static.rainfocus.com/oracle/oraclecode17/sess/1485992822413001Yd6N/PF/Cool%20in%20Java%208,%20and%20new%20in%20Java%209.pdf

https://github.com/ozlerhakan/java9-module-examples

https://trishagee.github.io/presentation/real_world_java_9/

https://github.com/CodeFX-org/demo-java-9

https://github.com/java9-modularity/examples

https://guides.gradle.org/building-java-9-modules/

https://www.youtube.com/watch?v=Txsl-K83ygI

https://www.youtube.com/watch?v=Yacu1yUktjY

https://blog.codefx.org/tools/jdeps-tutorial-analyze-java-project-dependencies/

http://tutorials.jenkov.com/java/modules.html

QUICK INFO

JAR FILE : Derlenmiş sınıfların paketlenmesini ve taşınabilmesini artıran sıkıştırılmış dosyalardır.

- Jar dosyası isimlendirmesinde bir kural yoktur
- Jar dosyası oluşturulduktan sonra ismi değiştirilebilir
- Jar dosyasındaki tüm sınıflara erişim yapılabilmektedir
- Her ne kadar bir sınıf, özellik ve method private tanımlansa da Java Reflection ile erişimi değiştirilebilir

MODULE : Java 9 ile gelen yeni paketleme türüdür.

- Tekil ismi vardır.
- Versiyonu vardır.
- Sadece izin verilen paketlerdeki sınıflara erişim yapılabilmektedir.
- Bağımlı olduğu modüllerin neler olduğu belirtilmektedir.
- Java Reflection ile izin verilmemiş sınıflara erişim yapılamaz.

MODULE TYPES

UNNAMED MODULES : Module Path içerisinde olmayan JAR paketidir. Klasik yöntemle oluşturulan paketlerdir.

AUTO MODULES: Module Path'de bulunup içerisinde module-info.java dosyası olmayan JAR paketleridir.

NAMED MODULES: Module Path'de bulunan içerisinde module-info.java dosyası olan JAR veya JMOD paketleridir.

MODULE DESCRIPTOR FILE

Oluşturulacak modülün root path'inde module-info.java isminde oluşturulan dosyadır. Modüllerin birbiri ile olan bağımlılık seviyeleri bu dosyada tanımlanır.

Nedir bu seviyeler:

1. Paket erişimleri
2. Reflection erişimleri
3. Bağımlı olunan modüller
4. Servis tanımlamaları

CHEAT SHEET : https://zeroturnaround.com/rebellabs/java-9-modules-cheat-sheet/

```
module [MODULE_NAME] {

       exports [PACKAGE];
       exports [PACKAGE] to [SPECIFIC_MODULE_NAME];

       requires [DEPENDED_MODULE_NAME];
       requires transitive  [DEPENDED_MODULE_NAME_TO_OPEN_UP];

       uses [SERVICE_CLASS_NAME];
       provides [SERVICE_CLASS_NAME] with [CLASS_NAME_TO_IMPL] ;

       opens [PACKAGE_NAME_FOR_REFLECTION_ACCESS];
       opens [PACKAGE_NAME_FOR_REFLECTION_ACCESS] to [SPECIFIC_MODULE_NAME];
}
```

JAVA SİSTEM MODÜLLERİ

JAVA_HOME/jmods klasörü altında bulunur. Tüm modüller java.base modülünü default kullanır. Bu yüzden module-info.java dosyasında ayrıca tanımlanmasına gerek yoktur.

Bu dizin altında bulunan modüllerde -incubator- olarak işaretlenmiş modüller bulunmaktadır. Bunlardan biri de HTTP Client modülüdür. incubator olarak işaretlenmiş modüllerin geliştirilmesi devam etmektedir.

WORKSHOPS

HAZIRLIK :

1. https://github.com/JUGIstanbul/java_9_10_workshop
2. Java 9 veya üst versiyonu kurulmalı
3. Proje loglama, kütüphane ve çalıştırılabilir uygulamalar için JUG Istanbul dizin yapısı oluşturulmalı
    a. Windows'da c:/jug_istanbul, Linux'de /jug_istanbul dizini oluşturulmalı
    b. /jug_istanbul/bin, /jug_istanbul/modules, /jug_istanbul/conf ve /jug_istanbul/logs dizinleri oluşturulmalı

WORKSHOP-1 : INTRODUCTION TO MODULES

SÜRE : 30 min.
AMAÇ : İlk Modülümüzü Oluşturalım
SENARYO :

JUG İstanbul olarak open source bir e-commerce projesi gerçekleştiriyoruz. Proje gereksinimleri incelendiğinde girilen ürünün başlıklarının SEO için hem ürün sayfası hem de resim url'lerinde kullanılması gerektiği ortaya çıkmıştır. Bunun için slugify yapılması gerekmektedir.

Ürün Başlığı : Lenovo Y520 i7-7700HQ 16GB 1TB+256SSD GTX1050 Win10 80WK004JTX
Slugified Text : lenovo-y520-i7-7700hq-16gb-1tb256ssd-gtx1050-win10-80wk004jtx

Yazılacak kodun başka bir projelerde de kullanılması için JAVA 9 Modülü olarak paketlenmesi istenmektedir.

GÖREVLER :

1. Aşağıdaki bilgilere göre modül oluşturulmalı
   Modül Name : jugistanbul.slugifier
   Modül Versiyon : 1.0.0
   Erişime Açılacak Paket : jug.istanbul.slugify
2. Öncelikle Java 9 ile gelen sistem modüllerinin neler olduğunu listeleyelim.

```
java --list-modules
```

3. Sınıflarda kullanılan sistem modül bağımlılıklarını tespit edelim.
4. Tespit edilen modüller module-info.java içerisinde tanımlayalım.
5. CMD/Terminal'den aşağıdaki komutu kullanarak uygulamayı derleyelim.

```
javac -d out -s src $(find src/main/java -name "*.java")
```

6. Slugifier modülünde main methodunu barındıran Slugifier sınıfı çalıştırılmalı ve gönderilen ifadenin slugified yapıldığı görülmeli

```
java --module-path out --module jugistanbul.slugifier/jug.istanbul.slugifier.Slugifier
```

WORKSHOP-2 : MODULE BY MODULE COMPILING AND PACKAGING

SÜRE : 30 min.
AMAÇ : modül bazlı derleme yapıp paketlemek
SENARYO :

Java 9'u yeni başlayan bir yazılımcı olarak her gün yeni şeyler keşfediyorum. Bunlardan bir tanesi de kaynak kodları modül ismindeki dizinlerde barındırıp modül bazlı derleme yapabilmek…

Bu özelliği öğrendikten sonra elimdeki modüllere uygulamak istedim. Bunun için workshop-2 altında bulunan projelerde aşağıdaki değişiklikleri yapalım:

1. workshop-2 dizini altına src klasörü oluşturalım
2. altına jugistanbul.workshop2.module1 ve jugistanbul.workshop2.module2 klasörlerini oluşturalım
3. jug-istanbul dizini altında bulunan module-1/src/main/java dizini altındakileri jugistanbul.workshop2.module1 klasörü altına kopyalayalım
4. jug-istanbul dizini altında bulunan module-2/src/main/java dizini altındakileri jugistanbul.workshop2.module2 klasörü altına kopyalayalım
5. Terminal açalım ve src dizini altına geçelim ve aşağıdaki komutu çalıştıralım (copy/paste lütfen yapmayınız)

```
javac -d out --module-source-path out
```

```
--module jugistanbul.workshop2.module1, jugistanbul.workshop2.module2
```

6. Derleme işlemi gerçekleştikten sonra out dizini altında jugistanbul.workshop2.module1, jugistanbul.workshop2.module2 modül dizinlerinin oluştuğu görülür
7. Derlediğimiz modülleri paketlemek için aşağıdaki komutu kullanalım

```
jar --create --file /jugistanbul_fs/modules/jugistanbul.workshop2.module1.jar -C out/jugistanbul.workshop2.module1

jar --create --file /jugistanbul_fs/modules/jugistanbul.workshop2.module2.jar -C out/jugistanbul.workshop2.module2
```

8. /jugistanbul_fs/modules altında jar dosyalarının oluştuğunu kontrol edelim
9. Aşağıdaki komut ile oluşan modül paketlerinin içeriğine bir göz atalım

```
jar --describe-module --file /jugistanbul_fs/modules/jugistanbul.workshop2.module1.jar

jar --describe-module --file /jugistanbul_fs/modules/jugistanbul.workshop2.module2.jar
```

10. /jugistanbul_fs/modules dizini altında hangi modüllerin olduğunu listeleyelim

```
java --module-path /jugistanbul_fs/modules --list-modules
```

```
jdk.xml.dom@10.0.2
jdk.xml.ws@10.0.2
jdk.zipfs@10.0.2
oracle.desktop@10.0.2
oracle.net@10.0.2
jugistanbul.workshop2.module1 file:///jugistanbul_fs/modules/jugistanbul.workshop2.module1.jar
jugistanbul.workshop2.module2 file:///jugistanbul_fs/modules/jugistanbul.workshop2.module2.jar
```

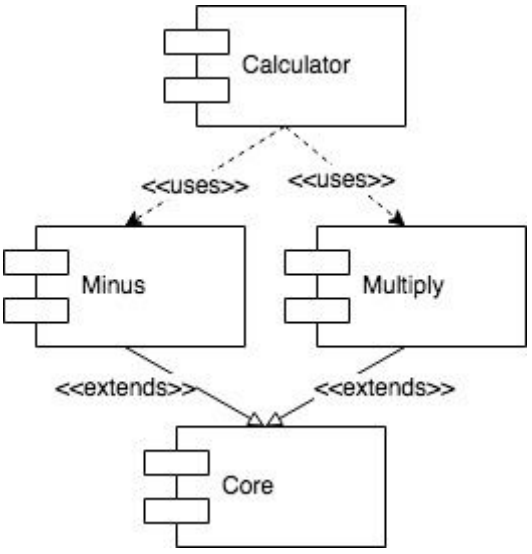
11. Oluşturduğumuz modüllerle birlikte Sistem Modüllerinin de listelendiğini göreceksiniz. Listelemeyi belirttiğimiz modül ve onun bağımlılıkları ile sınırlamak istersek --limit-modules parametresini kullanalım.

```
java --list-modules --limit-modules jugistanbul.workshop2.module1  --module-path /jugistanbul_fs/modules

java --list-modules --limit-modules jugistanbul.workshop2.module2  --module-path /jugistanbul_fs/modules
```

WORKSHOP-3 :

Karmaşık hesaplamaları yapabilen hesap makinası üreticisi için Java 8 ile yazılım gerçekleştirmiştik. Java 9 Modül'ün çıktığını öğrendiğimizde zaten Maven üzerinde modüler çalıştığımız için bu yeni özelliğe geçmenin tam bize göre olduğuna karar verdik.



Yaptığımız araştırmalarda Java 9 öncesinde yazılmış projelerin Java 9 ile çalışabilmesi classpath üzerinden kütüphanelerin yüklenmesi özelliğinin devam ettiğini öğrendik. Bunun için maven pom.xml'lerinde compiler plugin'inin source ve target değerlerini 9'a çekmemiz yeterli oldu.

GÖREV - 1:

SÜRE : 30 min.

AMAÇ : Tüm maven modülleri, Java 9 Modülü haline getirelim

1. Workshop-3 içerisindeki hesap makinası yazılımını indirelim.
2. module-info.java dosyalarını ve exports/requires tanımlamalarını yapalım.
3. modülleri yukarıdaki öğrendiğimiz komutları kullanarak Named Modules yapalım.
4. /jugistanbul_fs/modules dizini altına kopyalayalım.
5. jugistanbul-calculator maven modülü içerisindeki main class'ı aşağıdaki komutu kullanarak çalıştıralım.

```
java --module-path [MODULE_PATH] --module [MODULE_NAME]/[MAIN_CLASS]
```

- [...] içerisini uygun değerler ile değiştirelim

6. Değerler düzyün verildiğinde aşağıdaki sonuç alınmalıdır.

```
Minus Operation : 10-(-5)-2-20 => -7.0
Multiply Operation : 10*(-5)*2*20 => -2000.0
```

7. Uygulamayı çalıştırdığımızda JVM hangi modülleri yüklediğini görelim. Bunun için yukarıda çalıştırdığımız komuta --verbose:module parametresini ekleyelim.

```
java --module-path [MODULE_PATH] --verbose:module --module [MODULE_NAME]/[MAIN_CLASS]
```

8. Aşağıdaki gibi bir çıktı alacağız. Bu komutu ardarda çalıştırdığımızda modüllerin yüklenme sırasının değişeceğini göreceksiniz.

```
[0.089s][info][module,load] java.base location: jrt:/java.base
[0.226s][info][module,load] jugistanbul.calc.minus location: file:///jugistanbul_fs/lib/jugistanbul.workshop3.minus.jar
[0.226s][info][module,load] jdk.internal.vm.ci location: jrt:/jdk.internal.vm.ci
[0.226s][info][module,load] jugistanbul.calc.multiply location: file:///jugistanbul_fs/lib/jugistanbul.workshop3.multiply.jar
[0.226s][info][module,load] java.datatransfer location: jrt:/java.datatransfer
[0.227s][info][module,load] java.desktop location: jrt:/java.desktop
[0.227s][info][module,load] jdk.internal.vm.compiler.management location: jrt:/jdk.internal.vm.compiler.management
[0.227s][info][module,load] jdk.jartool location: jrt:/jdk.jartool
[0.227s][info][module,load] jdk.javadoc location: jrt:/jdk.javadoc
```

9. Komut çıktısını incelediğimizde ihtiyaç duymadığımız birçok sistem modülünün de yüklendiğini göreceksiniz. Bunu çalıştıracağımız modül ve bağımlılıkları seviyesinde limitlemek istediğimizde --limit-modules parametresini ekleyelim.

```
java --module-path [MODULE_PATH] --verbose:module  --limit-modules jugistanbul.calc.app --module [MODULE_NAME]/[MAIN_CLASS]
```

10. Komutu çalıştırdığımızda aşağıdaki gibi bir çıktı almalıyız.

```
[0.090s][info][module,load] java.base location: jrt:/java.base
[0.206s][info][module,load] jugistanbul.calc.core location: file:///jugistanbul_fs/lib/jugistanbul.workshop3.core.jar
[0.206s][info][module,load] jugistanbul.calc.multiply location: file:///jugistanbul_fs/lib/jugistanbul.workshop3.multiply.jar
[0.206s][info][module,load] jugistanbul.calc.minus location: file:///jugistanbul_fs/lib/jugistanbul.workshop3.minus.jar
[0.206s][info][module,load] jugistanbul.calc.app location: file:///jugistanbul_fs/lib/jugistanbul.workshop3.calculator.jar
Minus Operation : 10-(-5)-2-20 => -7.0
Multiply Operation : 10*(-5)*2*20 => -2000.0
```

11. Bir modülün bağımlılıklarını onu kullanmadan önce tespit etmek için jdeps komutu kullanılmaktadır. Oluşturduğumuz jugistanbul.calc.app modülünün bağımlılıklarını gelin inceleyelim.

```
jdeps --module-path [MODULE_PATH] [FULL_PATH_TO_MODULE_JAR]
```

JDeps'in farklı kullanımları için https://blog.codefx.org/tools/jdeps-tutorial-analyze-java-project-dependencies/ incelenebilir.

GÖREV - 2 :

SÜRE : 15 min.

AMAÇ : Müşteriden yazdığımız kodların çok esnek olmadığı geri dönüşünü aldık. Oluşturdukları bir işlem nesnesini o tipteki bir referansda tutmak zorunda kaldıklarını belirttiler. Bundan dolayı hesap makinasındaki tuş takımlarında değişiklik yapmak istediklerinde o butonların kodlarında sürekli değişiklik yapmak zorundaymışlar. Bizden Operation gibi bir üst sınıf tanımlayıp tanımlayamayacağımızı sordular.

Halbu ki biz zaten öyle yapmıştık. Peki niye göremiyorlar?

1. Jugistanbul-calculator maven modülü içerisindeki Calculator sınıfını açalım.
2. new MinusOperation(10, -5, 2, 20) ve new MultiplyOperation(10,-5, 2,20) ek oluşturduğumuz nesneleri Operation minus = … ve Operation multiply = … referanslarına atayalım.
3. jugistanbul.calc.core.Operation import'unu yapalım.
4. jugistanbul.calculator modülünü command line / terminal kullanarak derleyelim
    1. Jugistanbul-calculator maven modülü dizinine geçelim
    2. Aşağıdaki komutu çalıştıralım
    3. javac --module-path /jugistanbul_fs/lib -d out -s src $(find src/main/java -name "*.java")
5. Derleme sırasında aşağıdaki gibi bir hata almalıyız
6. [Ekran Çıktısı Kaydedildi]

İlgili hatada jugistanbul.calc.core modülünün yüklendiğini ama jugistanbul.calc.app tarafından okunamadığını anlıyoruz. Okunabilir yapabilmek için iki yöntem bulunmaktadır:

1. jugistanbul.calc.app modülünün module-info.java içerisinde import ile direkt erişebilir hale getirmek
2. jugistanbul.calc.multiply ve jugistanbul.calc.minus modülleri üzerinden erişilebilir hale getirmek

Zor ve kısıtlı olan ikinci yöntemi seçelim.

1. multiply ve minus modüllerinin module-info.java dosyalarında "requires jugistanbul.calc.core" yerine "requires transitive jugistanbul.calc.core" yazalım.
2. Her iki modülü de derleyip Java 9 Modülü oluşturalım.
3. "javac --module-path /jugistanbul_fs/lib -d out -s src $(find src/main/java -name "*.java")" komutunu tekrar çalıştıralım.
4. Derlemenin gerçekleştini görelim.

GÖREV - 3:

SÜRE : 15 min.

AMAÇ : Katıldığımız bir etkinlikde Java 9 ile uygulamaya özel çalıştırılabilir JRE imajının oluşturulabileceğini öğrendik. Bu imajın çalıştırılacağı bilgisayarlarda JRE'nin ayrıca kurulmasına gerek olmamaktaymış. Ayrıca daha performanslı çalıştığını öğrendik. Çünkü Runtime'da yapacağı modül ve sınıf linkleme işlemini imaj oluştururken yapmaktadır.

Hadi hesap makinası yazılımını JRE imajı haline getirelim:

1. Bir önceki workshop'larda oluşturduğumuz hesap makinası modüllerinin /jugistanbul/lib dizini altında olduğuna emin olalım
2. JRE imaj oluşturma aşağıdaki komutu çalıştıralım

```
jlink --module-path $JAVA_HOME/jmods:/jugistanbul/modules --add-modules
jugistanbul.calc.app --output calculator --launcher
calc=jugistanbul.calc.app/jugistanbul.calc.app.Calculator
```

3. --output parametresi ile belirttiğimiz calculator isminde bir dizin oluşmuş olmalı
4. Oluşturduğumuz imajı çalıştıralım

```
./calculator/bin/calc
```

5. Bu dizin altına geçip içerisinde bulunan dizinleri inceleyelim
6. Şimdi de JDK 9'un kurulu olduğu dizin içerisini inceleyelim
7. Diğer parametreleri kullanım şekilleri için https://docs.oracle.com/javase/9/tools/jlink.htm#JSWOR-GUID-CECAC52B-CFEE-46CB-8166-F17A8E9280E9 inceleyebilirsiniz.

WORKSHOP-4 :

SÜRE : 30 min.

AMAÇ : Servis Modülleri

SENARYO:

Endüstriyel IOT yapan bir yazılım firmasında çalışıyoruz. Fabrikalardan veri çekip bunları depolayan ve karar destek mekanizması olarak çalışan bir uygulamamız bulunmaktadır. Fabrikalarda sensör veri kaynağı genelde PLC ve OPC'lerdir. Bu yüzden veri toplama modülünü servis haline getirelim ve bu servisi PLC ve OPC'ler gerçekleştirsin. İleride başka bir veri kaynağından da mesela ModBus okuma

yapacağımız zaman kolay bir şekilde ekleyebilelim, tıpkı JDBC driver'i olarak MySql veya Oracle driverlarını yüklemek gibi...

Artık module-info.java oluşturmayı ve modülleri birbirleri ile bağlamayı öğrendiğimizde göre maven modülü olan dataservice projesindeki modülleri java 9 modüllerine çevirebiliriz.

GÖREV :

1. Java 9 modüllerine dönüştürme işlemini yapalım.
2. Modülleri birbiri ile bağladıktan sonra PLC ve OPC modüllerini service provider olarak ayarlayalım.
3. PLC ve OPC modülleri içindeki module-info.java dosyasına aşağıdaki tanımlamayı yapalım.

```
provides [SERVICE_INTERFACE] with [SERVICE_IMPLEMENTER_CLASS];
```

4. Modüllerimizi /jugistanbul_fs/modules altına oluşturalım
5. iotapp projesinin kaynak kodlarında DataServiceProvider sınıfını bulalım ve kodlarını inceleyelim
6. ServiceLoader sınıfının Java 1.6'dan veri var olduğunu biliyor muydunuz? Java 1.6'daki kullanım şeklini https://dzone.com/articles/simple-dependency-injection-wi yazısından inceleyebilirsiniz.
7. Java 9 ile ServiceLoader kullanımı module-info.java üzerinde yapılan provides...with... ve uses… direktifleriyle daha esnek hale geldiğini bilmeliyiz.
8. iotapp projesini dataservice'ini kullanması için module-info.java doyasında aşağıdaki tanımlamayı yapalım

```
uses [SERVICE_INTERFACE];
```

9. /jugistanbul_fs/modules dizini altına iotapp modülünü oluşturalım.
10. iotapp modülü içerisindeki main sınıfını çağırarak uygulamamız çalıştıralım
11. Zamanınız kaldıysa iotapp'ı uygulamasını da imaj haline getirebilirsiniz. Bunun için isterseniz öncelikle aşağıdaki yönlendirici ve detay bilgiyi bir okuyalım.

Jlink, modül çözümlemesi yaparken requires yönlendirmesini takip eder. ServiceProviders için kullanılan uses ve provides yönlendirmelerini takip etmez. ServiceProvider modüllerini --add-module yönlendirmesi ile teker teker eklemek gerekir. Yada --bind-services yönlendirmesi kullanılarak otomatik servis çözümlemesi yaptırılabilir. Bu opsiyon gereksiz servislerin de

yüklenmesine neden olabilir. Bu yüzden --suggest-providers yönlendirmesi ile belirtilen bir servisi gerçekleştiren modüllerin neler olduğu listelenebilir.

JLink, oluşturulacak imaja eklenecek paketlerin modül olarak oluşturulmasını bekler. Bu yüzden Otomatik Modülleri ve İsimsiz Modülleri kabul etmez.

JAVA 9 ile gelen diğer özelliklerin kullanımını incelemek için https://github.com/tanerdiler/java9 örnekleri inceleyebilirsiniz.

WORKSHOP 5: Maven + Spring Boot

AMAÇ: Gerçek hayatta kullanabileceğimiz bir modül yapısı ile Spring Boot projesi ayağa kaldırmak
Kod (spoiler içerir):
https://github.com/JUGIstanbul/java_9_10_workshop/tree/master/spring-boot-workshop

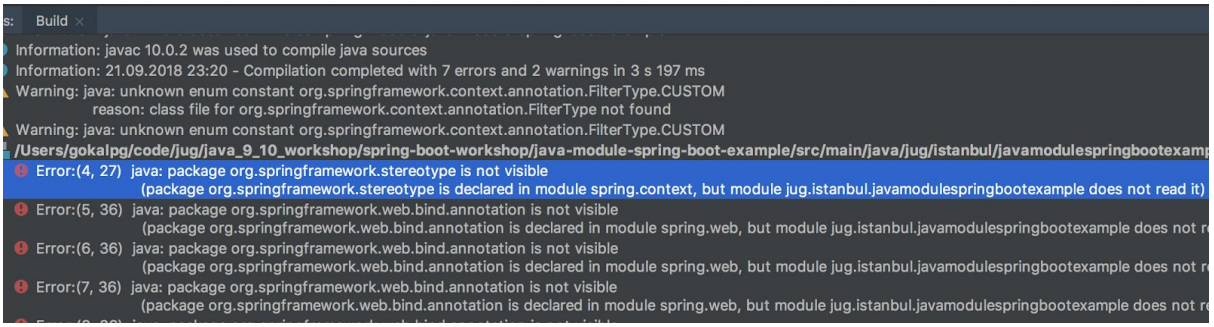
(README.md dosyasında da aşağıdakine benzer bir içerik bulunabilir)

Kurgu: 2 Modül, biri Slugifier'ın az modifiye edilmiş hali - "TurkishCharacterConverter" adındaki sınıfın paketi expose edilmiyor.
Diğer modül, http://start.spring.io'dan sadece web bağımlılığı seçimiş bir Spring Boot uygulaması (Java 10 seçmek için "advanced options" açılması gerekiyor.

Spring Boot uygulaması ayağa kalkıp, "slugify/v1" endpointinden Slugifier'a işlettiği metni döndürüyor; veri tabanı yok, session yok, hiç bir şey yok. Bunun için "SlugifyController" adında bir @Controller yarattık, bir de uygulamayı ayağa kaldıran main sınıfı var, başka hiç bir şey yok.

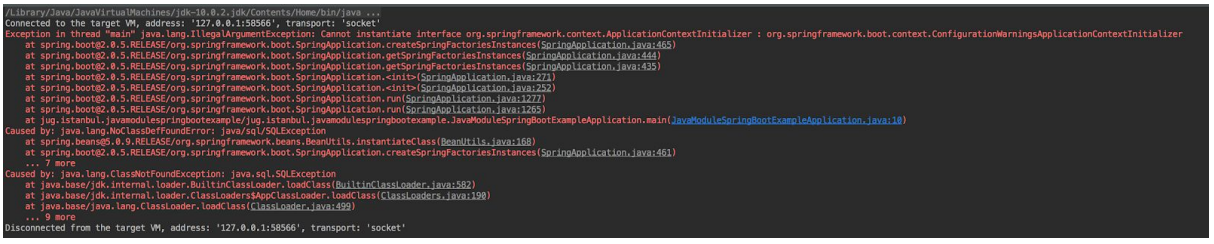
Meydan Okuma: Buradaki asıl olay iki tane: ilki, maven ile jigsaw'un birbirleriyle olan ilişkisi - hiç ilişkileri yok! maven derleme zamanında kütüphaneleri bir araya getirip bizim gösterdiğimiz şekilde derleyip build etmekten başka bir iş yapmıyor. Jigsaw ise çalışma anında hangi modülün, hangisiyle hangi münasebette bulunabileceğini düzenliyor.

İkinci olay ise doğru module-info.java'yı yazmak - koddaki halini hemen göstermek istemeyebiliriz; biraz deneye yanıla bulmaları daha iyi olur. deneme yanılma şöyle oluyor:

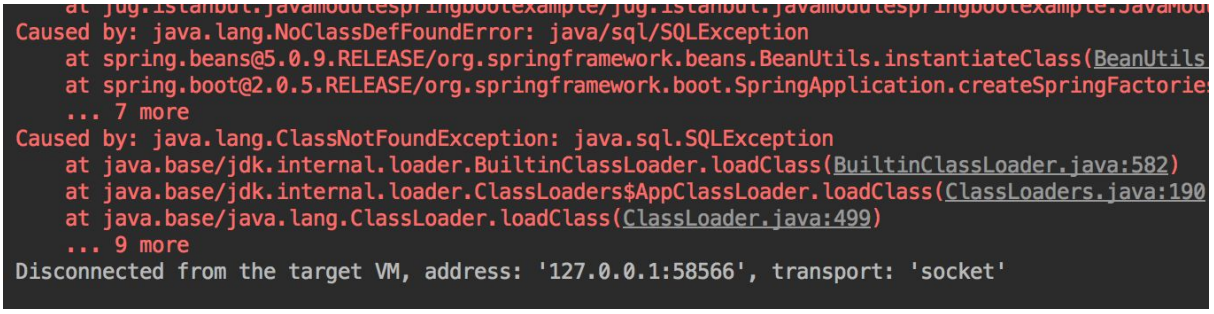


yani IntelliJ diyor ki, git spring.context ve spring.web'e requires ekle.

Tabii bazı bağımlılıklar çalışma anında ortaya çıkıyor - neden? çünkü Spring, reflection'a dayalı bir çatı.

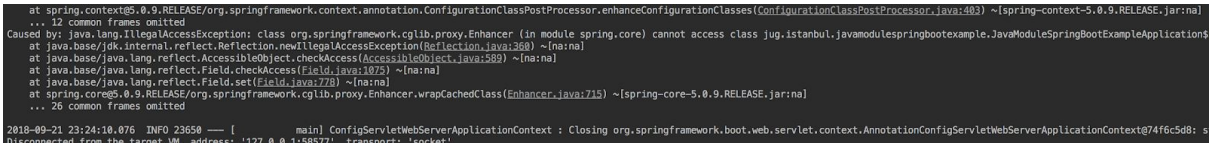


(zooming)



hadi, requires java.sql!

Hadi çalıştır! BOM!



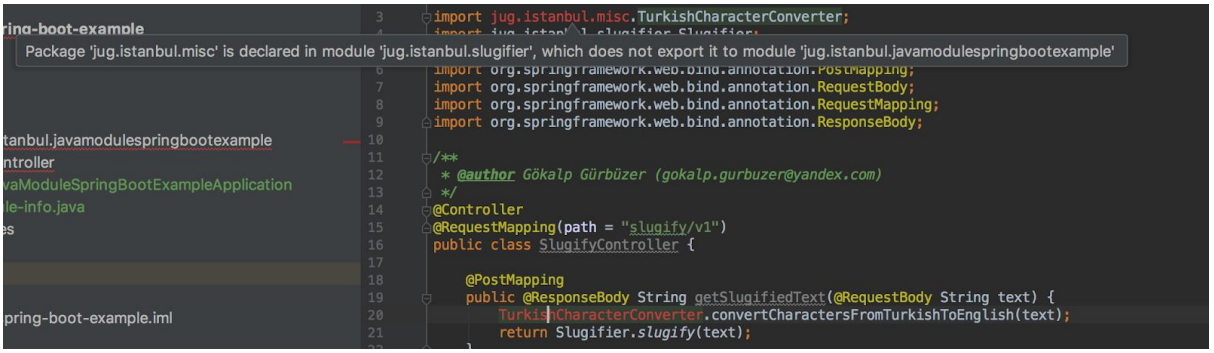
(zoom)

```
ingframework.context.annotation.ConfigurationClassPostProcessor.enhanceConfigurationClasses(ConfigurationClassPostProcessor.java:403) ~[sprin

: class org.springframework.cglib.proxy.Enhancer (in module spring.core) cannot access class jug.istanbul.javamodulespringbootexample.JavaMod
ction.newIllegalAccessException(Reflection.java:360) ~[na:na]
leObject.checkAccess(AccessibleObject.java:589) ~[na:na]
eckAccess(Field.java:1075) ~[na:na]
t(Field.java:778) ~[na:na]
framework.cglib.proxy.Enhancer.wrapCachedClass(Enhancer.java:715) ~[spring-core-5.0.9.RELEASE.jar:na]
```

Demek ki neymiş? module-info.java'ya open yazacakmışız ki spring kankamız bizim bean'lerimizi okuyabilsin :D

Son olarak da Spring Boot uygulaması içinden, kullandığımız kütüphanenin dışarıya açmadığı sınıfı kullanmaya çalışalım:



Sonuç: Bir legacy Spring Boot uygulamasını Java 9+'da ayağa kaldırmak zor olacaktır; ancak ilkini yaptıktan sonra diğerlerini daha hızlı adapte edebiliriz. Özellikle Spring Boot 2.0 öncesinin biraz sancılı olduğunu okudum.