# MAVEN NOTLARI

Maven,projemize eklemek istediğimiz jarları,frameworkleri(mysql connector,spring, hibernate) gibi projeye sonradan ekleyeceğimiz özellikleri tek bir xml dosyasında tutarak bize kolaylık sağlar.Aşağıdaki 2 sitede maven repository sitesidir. İkincisi daha iyi ve tercih edilendir.

\*Maven Repository = Maven konfigurasyon kodlarının olduğu depolar.

1. <https://search.maven.org/>

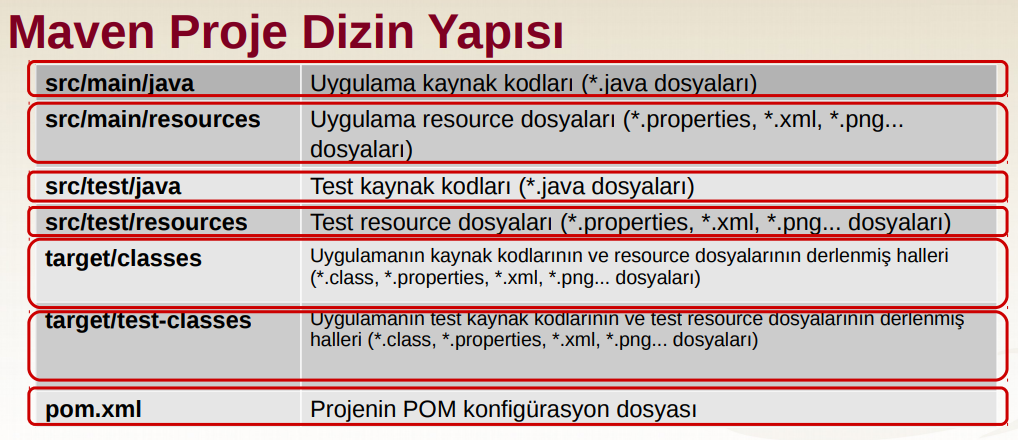
2. <https://mvnrepository.com/>

ÖNEMLİ OLAYLAR 1=Maven projesi oluştururken CTRL+N oradan maven seçiyoruz.Başlangıçta groupID’si org.apache.maven.archetypes olan artifactID’si quickstart olanlara başlıyoruz.Yeni bir maven projesi oluştururken groupID’ye genelde com.şirketin ismi,artifactID’ye ise projenin ismi verilir.

ÖNEMLİ OLAYLAR 2=Jar projesi demek desktop uygulamaları için falan,war projesi ise web uygulamarı için,ear ise enterprise geliştirmek içindir.

ÖNEMLİ OLAYLAR 3=Eğer war oluşturursak proje başlangıcında bi hata olur.Bunun için soldan projenin üstüne gelip sağ tıklıyoruz.Java EE Tools’a tıklıyoruz ve yandan Generate Deployment Descriptor Stub’a basıyoruz.Artık hatanın gitmiş olması lazım.

ÖNEMLİ OLAYLAR 4=Maven projemize soldan sağ tıklayıp run as > maven install basınca consolede build failure diye bir hata gelebilir.Eğer gelirse window > preferences > java > installed jres geliyoruz sağda add’e tıklayıp pcde kurulu olan jdk nın konumunu veriyoruz.Bu yolla eclipse jdk mızı ekledik ama aktifleştirmedik.Aktifleştirmek için installed jresin içinde Execution Environmente girip oradan kurulu olan eclipse versiyonuna tıklıyoruz.Ve sağdan jdkyı seçiyoruz.



# LOG4J2 NOTLARI

# Log4j vs Log4j2

# Log4j loglama işlemlerinin log4j.properties dosyasıyla yapılmasını sağlar,Log4j2’da ise artık log4j2.xml dosyasını kullanılıyoruz.Log4j eskide kaldı artık kullanılmıyor.Log4j2 ise yeni ortaya çıkmış log4j ve Logback’in eksiklerini kapatmaya çalışıyor.Bu yüzden biz güncel olanı yani Log4j2’yu öğrenicez.

# Loglama

# Loglama son kullanıcılar ve sistem yöneticileri için, uygulamanın devamlılığı açısından çok önemlidir. Herhangi bir projede karşılaşılan hataları, performans sıkıntılarını ve bugları anlık olarak yakalayıp düzeltmek için kullanılır.Log4j Core’yi mvnrepositoryden bulup pom.xml(maven) yoluyla projemize ekliyoruz.Projemizin resources klasörünün içinde log4j2.xml diye bir file oluşması lazım eğer oluşmadıysa bunuda google’dan bulup resources dosyasının içine koyuyoruz.Bu xml dosyasının resources klasöründe olmasın çok önemli.

1. .<Console name="Console" target="SYSTEM\_OUT"> = Bu kod Loglama işleminin sonuçlarının console’da olucağını belirtiyor.
2. <PatternLayout pattern="%d{ISO8601} [%t] %-5level %logger{%36} - %msg%n"/> = Bu kod loglama işleminin console’da nasıl gözükeceğini belirtiyor

* "%d{ISO8601} = Consolede ilk başta tarih saat vs gözüksün istiyoruz.
* [%t] = Logumuz hangi metodta çalıştı
* %-5level = bu hata payı gibi bişi
* %logger{%36} = tutulan log hangi classta çalıştı
* %msg%n = tutulan logtaki mesaj
* (%F:%L) = tutulan logun hangi classta hangi satırda olduğunu gösterir
* <Root level="INFO"> = Bunu içine WARN yada ERROR yazarak sadece o levele ait logları consolda gösteririz.

--------------------------------------------------------------------------------------------------------------------------

<Configuration>

<Appenders>

<Console name="Console" target="SYSTEM\_OUT">

<PatternLayout

pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" />

</Console>

</Appenders>

<Loggers>

<Logger name="com.project" level="WARN" additivity ="false" >

<AppenderRef ref="Console" />

</Logger>

<Logger name="com.project" level="INFO" additivity ="false" >

<AppenderRef ref="Console" />

</Logger>

</Loggers>

</Configuration>

1. Kırmızı olanlar bu Loggerın sadece tırnak için yazılmış olan paketlerde çalışacağını söylüyor.

2. Mavi olanlar ise hangi tür Loggerların çalışacağını söylüyor.

3. Yeşil olanlar ise Loggerın console’da bir kez çalışacağını söylüyor.(Şayet additivity yazmasaydık.Ve programı her çalıştırdığımızda önceki loggerlarıda yazacaktı.

---------------------------------------------------------------------------------------------------------------------------

<Logger name="com.project" level="WARN" additivity ="false">

<AppenderRef ref="Console" />

</Logger>

<Logger name="com.project.Logging" level="INFO" additivity ="false">

<AppenderRef ref="Console" />

</Logger>

Yukarıdaki log4j2.xml kodunda WARN türünün bütün com.project nin alt paketlerin arayacağını ve bulunca logger oluşacağını belirtiyor.Ama INFO türünün sadece com.project.Logging paketindeki classlarda arayacağını söylüyor.

# File içine Loglama

<?xml version="1.0" encoding="UTF-8"?>

<Configuration>

<Appenders>

<Console name="Console" target="SYSTEM\_OUT">

<PatternLayout

pattern="%d{ISO8601} [%t] %-5level %logger{%36} - %msg%n" />

</Console>

<file name="file" fileName="C:\Users\Hp\Desktop\Logging.log">

<PatternLayout

pattern="%d{ISO8601} [%t] %-5level %logger{%36} - (%F:%L) - %msg%n" />

</file>

# //Filenin ID’si

# //Filenin oluşturulacağı yerin konumu

# //Logların hangi satır hangi metodta olduğu belirten kod

</Appenders>

<Loggers>

<Root level="ALL"> //ALL yazarak info,error,fatal tüm türleri dahil ettik.

<AppenderRef ref="Console" /> //Burada 2 Tane appender oluşturduk çünkü logları bir console birde filenin içine yazdırmak istiyoruz.

<AppenderRef ref="file" />

</Root>

</Loggers>

# </Configuration>

# MAİN CLASS

Logger **log** = LogManager.*getLogger*();

# log.info("Logging Messages");

# RollingFile Kullanımı

# Log4j2’de rollingfile’nin amacı şudur.Diyelim ki bi file dosyasına loglama yapacağız.Bu rolling file ile örneğin loglama yapacağımız filenin boyutunu belirliyoruz,Mesela 100 mb,Bu file 100 mb boyutuna ulaşana kadar loglamaları bu filenin içine yapacak ve 100 mb teye ulaşınca otomatik yeni bir file oluşturucak ve sonraki 100 mblik logları bunun içine yazacak.

<?xml version="1.0" encoding="UTF-8"?>

<Configuration>

<Properties>

<Property name ="path">C:\Users\Hp\Desktop\</Property>

</Properties>

<Appenders>

<RollingFile name="rollingFile"

fileName="${path}RollingFile.log"

filePattern="${path}RollingFile-%d{yyyy-MM-dd}-%i.log">

<PatternLayout pattern="%-5level %logger{%36} (%F:%L) - %msg%n"></PatternLayout>

<Policies>

<SizeBasedTriggeringPolicy size="3 KB" /> //Boyut 3 KB’yi geçince yeni bir file oluştur komutu

</Policies>

<DefaultRolloverStrategy max="25" ></DefaultRolloverStrategy> //Aynı anda max 25 Tane file açabilirsin komutu.

Uyarı:Diyelim ki rolling file’nin boyutunu 3 KB tuttuk ve 30 filelik loglama yaptık.Ama burada aynı anda açılabilir 20 tane file yazdık.Böyle olursa En son 20 logu tutar.ilk 20 logluk alan olmadığı için kayıt etmez.

</RollingFile>

</Appenders>

<Loggers>

<Logger name="com.project" level="INFO" additivity="false">

<appender-ref ref="rollingFile"></appender-ref>

</Logger>

</Loggers>

</Configuration>

MAİN CLASS

Logger **log** = LogManager.*getLogger*();

**for** (**int** **i** = 1; i <= 800; i++)

{

log.info(i + ". Logging Message");

}