Yazılım Geliştirme Ortam ve Aracları

4. Hafta

Maven, Junit ve Git Kullanimi

Kaynaklar:
1-) https://kurukod.wordpress.com/2015/06/26/maven-nedir-nasil-kullanilir-2/
2-) http://www.ugurkizmaz.com/YazilimMakale-609-Java-Unit-Test-ve-JUnit-Kullanimi.aspx
3-) https://www.gitbook.com/book/aliozgur/git101/details
4-) https://www.gitbook.com/book/vigo/git-puf-noktalari/details
5-) https://kurukod.wordpress.com/2015/06/26/maven-nedir-nasil-kullanilir-2/

Maven

- Maven nedir?

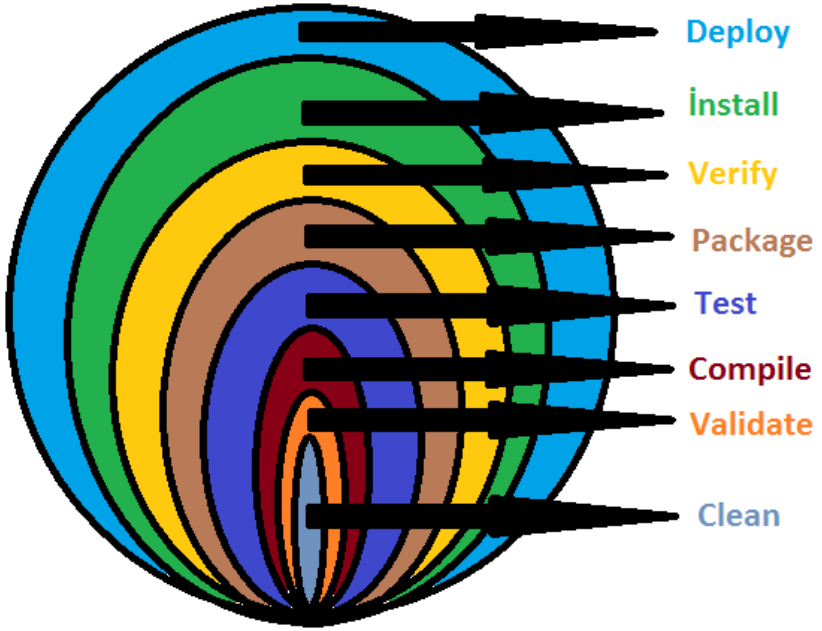
Maven Nedir?

Maven, proje geliştirirken proje içerisinde bir standart oluşturmamızı, geliştirme sürecini basitleştirmemizi, dokümantasyonumuzu etkili bir şekilde oluşturmamızı, projemizdeki kütüphane bağımlılığını ve IDE bağımlılığını ortadan kaldırmamızı sağlayan bir araçtır.

- Yazılım Proje Yönetim Aracı
  - Modül yönetimi
  - Bağımlılık (dependency) yönetimi
  - Yapılandırma (build)
  - Raporlama
  - Test
  - ....
- Kurulum?
  - Eclipse, Netbeans, IntelliJ gibi IDE'lere entegre gelir
  - JDK kurulumu gerektirir
  - http://maven.apache.org/download.cgi

Maven Komutları

- **clean:** projenin derlenmesi sırasında oluşan target klasörünün silinmesini sağlar.
- **validate:** projenin target dosyasını siler ve daha sonra hatalı kısımları tarar.
- **compile:** projeyi clean ve validate eder, daha sonra derler.
- **test:** projeyi derler ve test sınıflarını çalıştırır.
- **package:** proje testlerini yapar ve eğer hata yoksa projeyi paketler.
- **verify:** projeyi paketler ve daha sonra bu paketlerin geçerliliğini kontrol eder.
- **install:** projeyi doğruladıktan sonra repository sunucusuna yükler.
- **deploy:** projeyi uygulama sunucusuna gönderir.



Maven POM Dosyası

Proje yapılandırma yönetimi, projenizin build ya da deploy yapılandırmalarını **POM**dosyasından yönetebilirsiniz. Sadece birkaç satır kodla bu yapılandırmalar arasında geçişler yapabilirsiniz. Mesela büyük çaplı bir proje, farklı sunucu sistemlerinde ya da farklı veri tabanlarında eş zamanlı olarak çalışması gerekebilir. Bunun için her güncelleme sırasında farklı yapılandırma ayarlarıyla bu sistemleri güncellememiz gerekir. Her sistem için yapılandırma dosyalarını baştan düzenlemek oldukça yorucu bir iş.
Ancak **POM** dosyasında tanımlanacak yapılandırma ayarları işimizi görecektir. Sadece yapılandırma adını değiştirerek proje çıktısını farklı sistemlere uygun hale getirebilmekteyiz.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

        <modelVersion>4.0.0</modelVersion>

        <groupId>tr.edu.medipol</groupId>
        <artifactId>MavenProjeOrnekParent</artifactId>
        <version>2.1.4</version>
        <packaging>pom</packaging>

        <name>Maven Proje Ornek</name>
        <description>Hafta 4 mAVEN juNİT pROJESİ</description>

        <modules></modules>
</project>
```

Maven POM Dosyası

Bağımlı kaynaklar, projede kullanılacak tüm kütüphaneler ve eklentiler **POM**(Project Object Model) dosyasından kolayca yönetilebilmektedir. **Maven**, kütüphane dosyalarını kendi repository sunucularında barındırır. Projede kullanmak istediğiniz kütüphane dosyalarını ilk olarak sizin local repository klasörünüzde arar, eğer bulamazsa kendi sunucularında arama yapar, eğer kendi sunularında da bulamazsa sizin tanımlayacağınız bir sunucu adresinden dosyayı sizin local klasörünüze indirir ve projeniz içerisinde kullanabilmenizi sağlar. Ayrıca bir kütüphane başka kütüphanelere bağımlıysa bu bağımlı olduğu kütüphaneleri de indirir ve projenize ekler.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

        <modelVersion>4.0.0</modelVersion>

        <groupId>tr.edu.medipol</groupId>
        <artifactId>MavenProjeOrnekParent</artifactId>
        <version>2.1.4</version>
        <packaging>pom</packaging>

        <name>Maven Proje Ornek</name>
        <description>Hafta 4 mAVEN juNİT pROJESİ</description>

        <modules></modules>
</project>
```

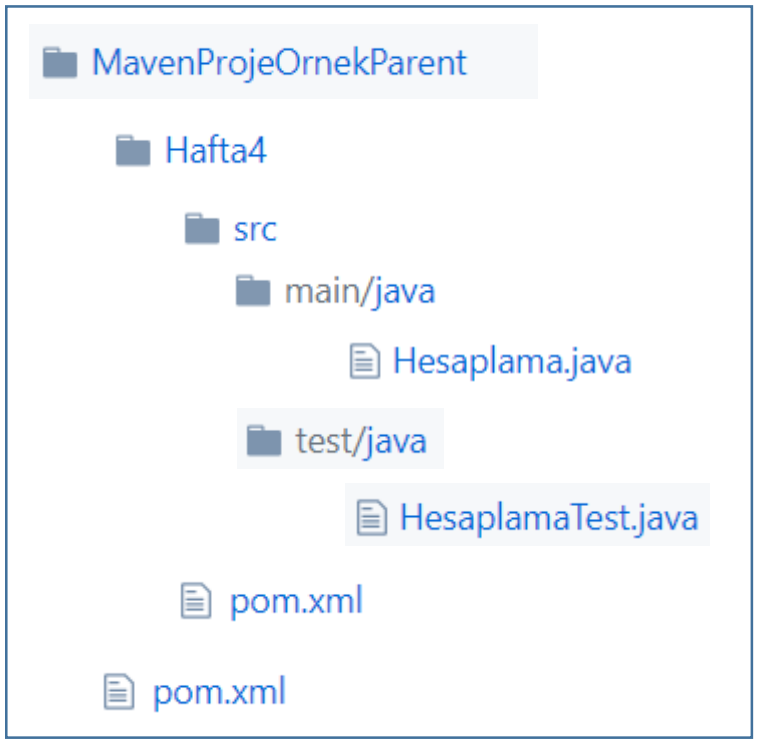
Sürüm yönetimi, her **Maven** projesinin bir **grup id** ‘si, bir **yapı id** ‘si ve bir de **sürüm numarası**vardır. Projenin farklı sürümlerini saklayabilir ve bunları daha sonra yeni projelerde kullanabiliriz.

Örnek: Parent

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

        <modelVersion>4.0.0</modelVersion>

        <groupId>tr.edu.medipol</groupId>
        <artifactId>MavenProjeOrnekParent</artifactId>
        <version>2.1.4</version>
        <packaging>pom</packaging>

        <name>Maven Proje Ornek</name>
        <description>Hafta 4 mAVEN juNİT pROJESİ</description>

        <modules></modules>
</project>
```

Örnek: Modül

Maven Proje Yapısı




```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"
>
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>tr.edu.medipol</groupId>
    <artifactId>MavenProjeOrnekParent</artifactId>
    <version>2.1.4</version>
  </parent>

  <artifactId>Hafta4</artifactId>

  <dependencies>

        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.2</version>
            <scope>test</scope>
        </dependency>

  </dependencies>

</project>
```

Önceki slaytta tanımlanan parent

Parent'tan hiyerarşik olarak miras alan alt modül = artifact
- Parent ile aynı grup ve versiyona sahip olur

JUnit

- Birim Testi nedir?
- Junit nedir, ne için kullanılır?

Birim Testler

- Birim (Unit) test, yazılımda en temel test çeşitidir. Yazılan fonksiyonların çalışıp çalışmadığının testlerini yapmak için kullanılır.
- Buradaki en temel noktalardan biri, oluşturulan bütün testler birbirinden bağımsız çalışmaktadır ve her testin tek bir sonucu olmalıdır (true / false)
- JUnit , Java da en sık kullanılan Test Framework'leridnen biridir.

Örnek

```java
public class Hesaplama {

	public static void main(String[] args) {
		int sonuc = topla(19, 25);
		System.out.println(sonuc);
	}

	public static int topla(int sayi1, int sayi2) {
		return sayi1 + sayi2;
	}

	public static int carpma(int sayi1, int sayi2) {
		return sayi1 * sayi2;
	}

	public static int bolme(int sayi1, int sayi2) {
		return sayi1 / sayi2;
	}
}
```

```java
import static org.junit.Assert.*;

import org.junit.Test;

public class HesaplamaTest {

	@Test
	public void testToplaHepsi() {

		for (int sayi1=-10000; sayi1<10001; sayi1++) {
			for (int sayi2=-10000;sayi2<10001; sayi2++) {
				int gercekSonuc = Hesaplama.topla(sayi1, sayi2);
				assertEquals(sayi1+sayi2, gercekSonuc);
			}
		}

	}


	@Test
	public void testTopla25ve14() {
		int gercekSonuc = Hesaplama.topla(25, 14);
		assertEquals(39, gercekSonuc);
	}

	@Test
	public void testToplaNegatif() {
		int gercekSonuc = Hesaplama.topla(-25, -14);
		assertEquals(-39, gercekSonuc);
	}

}
```

Git

- Git nedir?

Git Nedir?

- GIT, dağıtık çalışan sürüm kontrol sistemi (*DVCS*) ve kaynak kod yönetim (*SCM*) aracıdır.
  - DVCS: **D**istributed **V**ersion **C**ontrol **S**ystem,
  - SCM: **S**ource **C**ode **M**anagement anlamına gelir.

- Herhangi bir merkez sunucuya ihtiyaç duymadan, offline olarak çalışabilmesi, kolay merge ve branch alma gibi özellikleriyle diğer eş değer araçlara göre öne çıkar.

Repository ya da Repo Nedir?

- Revizyon kontrolü altındaki dizin bir Repository’dir. Kısaca içinde dosyaların (*daha doğrusu kaynak kodların*) bulunduğu bir depodur.

- "git init« komutuyla boş bir repo oluşturulduğunda ön tanımlı olarak "master« isimli branch’de bulunulur.

Branch Nedir?

- İstediğiniz bir anda, elinizdeki kod’dan hızlıca bir ya da N tane kopya çıkartma işlemidir. Yerel operasyondur. Yani yaptığınız her branch, siz paylaşmadıkça sadece sizde bulunur.
- Her branch kodun anlık kopyasını içerir.
- Branch’ler birbirlerine "merge« edilebilir, yani değişiklikler birleştirilebilir.
- Daha teknik bir anlatımla branch aslında içinde commit-id yazan bir işaretçiden başka bir şey değildir.

Commit Nedir?

- Git versiyonlama standardında bir commit komple anın fotoğrafını (snapshot) içerir.

- Mesela,
  - Örneğin bir branch içinde çeşitli kodlarınız var.
  - Bir OKUBENI.txt dosyası oluşturdunuz ve içerisine ad ve soyadınızı yazdınız.
  - Değişiklikleri commit yaptığınızda, kodlarınızın o anki (yani OKUBENI.txt içeren) hali bir ID ile kaydedilir. ID = 0 diyelim.
  - Daha sonra OKUBENI.txt dosyasının içine doğum tarihinizi de yazdınız ve commitlediniz. Dosyanın bu halini içeren anlık durum kaydedilir. Bu haline ID=1 diyelim.
  - Daha sonra OKUBENI.txt'yi sildiniz ve commit'lediniz. Şimdi kodun OKUBENI.txt içermeyen haline sahipsiniz. Bu haline ID = 2 diyelim.
  - İstediğiniz an ID=0 ki hale dönüp OKUBENI.txt dosyasının sadece ad ve soyadınızı içeren halini geri elde edebilirsiniz veya değişik ID'lere sahip bu commitler arasındaki değişiklikleri karşılaştırabilirsiniz.

DeğişiklikleriGeçici Kaydetmek (Stash)

git stash ile üzerinde çalıştığınız ancak henüz commit etmediğiniz değişikliklerin geçici olarak Git tarafından kayıt altına alınmasını sağlarsınız.

Eclipse gibi arayüzlerde commitlemeden önce değişikliklerinizi stash alanına almanız gerekir.

Branch'lerle Çalışmak

- GIT'in en önemli özelliği branch mekanizmasıdır.
- Her bir branch kodun değişik bir andaki kopyasını içerir.
- Kodun üzerinde yeni bir ekleme ya da değişiklik yapacağımızda genelde yeni bir branch alırız ve değişikliklerimizi bu branch üzerinde yaparız.
- Değişikliklerimiz tamamlanınca, bu branch'i ana branch ile birleştirebilir yani değişiklikleri ana branch'a birleştirebiliriz.
- Biz değişiklik yaparken, ana branch'da da değişiklikler yapılmış olabilir. Bu durumda eğer bir «çakışma» yoksa bizim değişikliklerimizle, ana branch'daki değişiklikler birleşir.

https://vigo.gitbooks.io/git-puf-noktalari/content/bolum-02/branch-ler-ile-calismak.html

Checkout ve merge Kavramı

- **git checkout** komutu ile değişikliklerin aktarılacağı hedef branch'inizi aktif (HEAD) hale getirirsiniz.

- **git checkout** komutu ile bir başka branchdekiş değişikliği aktif branch ile birleştirebilir yani başka branch’daki değişiklikleri aktif branch’a çekersiniz.

Branch’lerin Çakışması: Conflict

- Aynı dosyalar üzerinde çalışınca **conflict** yanı çakışma yaşamak kaçınılmazdır.

- Bu durumda çakışmayı çözmek yazılımcının sorumluluğundadır.

```
<!DOCTYPE html>
    <html lang="en">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>Demo site</title>
        <link rel="stylesheet" href="public/css/application.css" type="text/css" />
    </head>
    <body>
        <header>
            <h1>Demo Site 2017</h1>
        </header>
<<<<<<< HEAD
        <script src="public/js/global-1.js"></script>
=======

        <footer>Address information</footer>
        <script src="public/js/global.js"></script>
>>>>>>> feature
    </body>
</html>
```

https://vigo.gitbooks.io/git-puf-noktalari/content/bolum-02/branch-ler-ile-calismak.html

Uzak Repo (Remote) Kavramı

- Günlük çalışmamız sırasında staging ve commit gibi versiyon kontrolü ile ilgili işlemlerin çoğunu yerel diskimizde yer alan local repository üzerinde yaparız.
- Ancak takım çalışması söz konusu olduğunda, takımdaki geliştiricilerin birlikte çalışabilmesi için herkesin değişikliklerini ortak bir alanda yayınlaması ve diğerlerinin de bu ortak alan üzerinden bu değişiklikleri kendi branch'lerine entegre etmesi gerekecektir.
- Uzak (remote) repository'leri en basit anlamda tüm ekibin erişimi olan dosya sunucusu olarak düşünebilirsiniz.

Uzak Repo (Remote) Kavramı

- Yerel (local) repository'ler geliştiricilerin kendi bilgisayarlarında yer alırken Remote repository'ler, çoğunlukla internet olmak üzere, ekipteki herkesin erişebileceği bir sunucuda yer alırlar.
- Teknik olarak uzak (remote) repository'ler ile yerel repositoryler arasında bir fark yoktur. Yerelrepository'ler için önceki bölümlerde ele aldığımız commit işlemi, branch oluşturma gibi işlemlerin tamamı remote repository'ler için de yapılabiliyor.
- Uzak bir repository'yi **git clone** komutu ile yerel diskinizde indirebilirsiniz.

Uzak Repo (Remote) Kavramı

- **git push** komutu ile yerel branch'inizdeki değişiklikleri uzaktaki branch'a gönderebilirsiniz.

- **git fetch** komutu ile de uzaktaki branch'inizdeki değişiklikleri yerel branch'a alabilirsiniz.

Git Ek Kaynaklar

- Kaynaklar
  - https://www.gitbook.com/book/aliozgur/git101/details
  - https://www.gitbook.com/book/vigo/git-puf-noktalari/details

Demo

- Maven Projesi Olusturma
- Hesaplama.java
- HesaplamaTest.java
- Git Commit/Push/Fetch