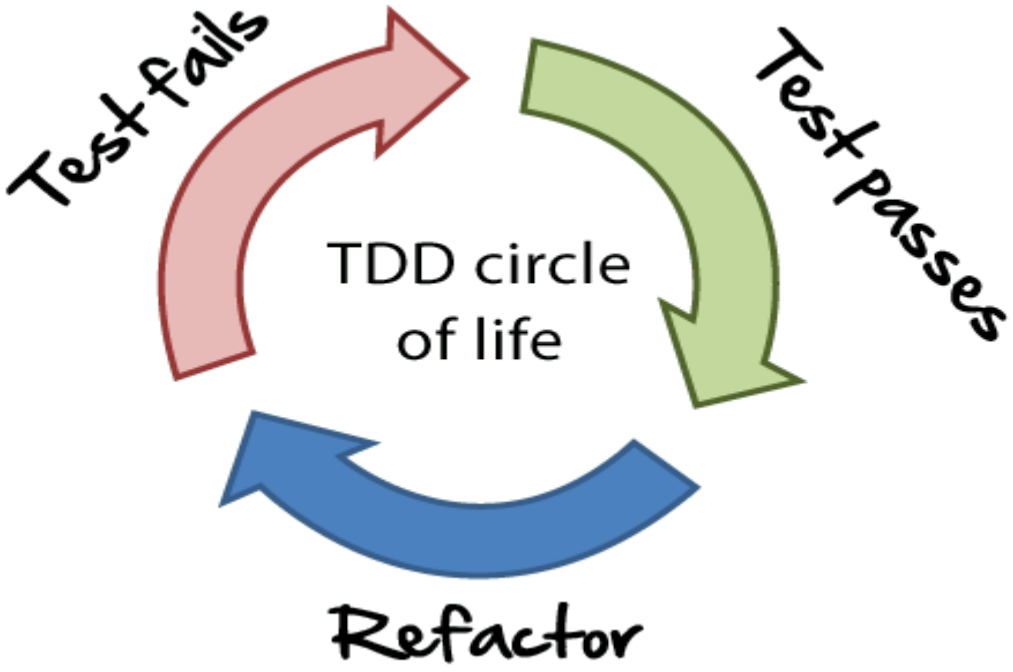
TDD Nedir?

İlk önce testi yazıp sonra kodu geliştirme yöntemidir. Çoğunlukla, programcı testinin (ya da birim testinin) uygulama geliştirme sürecini uzatacağı düşünülür. Bu yanlış bir düşüncedir Yazılımcı testlerinin tasarıma başlarken yazılması tasarımı kolaylaştırır. Kolay olan şey genelde kısadır. TDD, tasarımı basitçe yapmamızı, basit ve yalın kodlama yapmamızı, düzenli refactoring yapmamızın kolaylaşmasını ve gevşek bağlılığı sağlayan bir yaklaşımdır.

1. Bir test yazılır.
2. Test başarısız olur.
3. Test başarılı hale getirilir.
4. Mevcut bütün testlerin başarılı olması sağlanır.
5. Kod refactor edilir. Yani kodda iyileştirme ve(ya) temizleme yapılır.



Obje Mock'lama

Mock kavramı istediğimiz bir objenin yerine geçebilen sahte objelerdir. Bu objeler bizim istediğimiz gibi davranmasını sağlarız.

Mock’lamak ne işe yarar?
-Unit test bir birimi test ettiği için, oradaki akışı test ederken bu akışa bağlı olan dependency‘lerin test akışını bozmamasını sağlar.

-Unit test işlemini yaparken, test’i istediğimiz senaryoda yönlendirebilmemizi sağlar.
-Complex objelerin yavaşlıklarından kurtulabilmemizi sağlar.

Spring Profile

-Profile anotasyonu uygulamamızdaki farklı çalışma isterlerine göre programımızın hangi işlevinin çalışacağını çalışacağını isteğimize göre seçmemizi sağlamaktadır.
Mesela pdf ve html formatlarında cıktılar üretmek istiyoruz.

```
import org.springframework.context.annotation.Profile;

import org.springframework.stereotype.Component;

@Component

@Profile("pdf")

public class PrintServicePdf implements PrintService{

    @Override

   public String printHello(String name) {

      return "Pdf Printed : " + name ;  }

}
```

```
import org.springframework.context.annotation.Profile;

import org.springframework.stereotype.Component;
```

```
@Component

@Profile("html")

public class PrintServiceHtml implements PrintService{



   @Override

   public String printHello(String name) {

       return "Html Printed : " +name;

   }

}
```

Properties dosyasında bunu belirtiriz.

spring.profiles.active=pdf/html

```
@RestController

@RequestMapping(path = "/")

public class PrintController {

   @Autowired

   private PrintService printService;



   @RequestMapping(path = "hello/{name}", method =
RequestMethod.GET)

   public String hello(@PathVariable(value = "name") String name){
```

```
        return printService.printHello(name);

    }

}
```

Burada çıktıyı properties dosyamızda hangi formatı verdiysek o formatta bize bir çıktı üretir.

Unit test
-Bir yazılımın en küçük test edilebilir bölümlerinin,tek tek ve bağımsız olarak doğru çalışması için incelendiği bir yazılım geliştirme sürecidir.Unit Test yazmak kodda yeniden düzenleme(Refactor) işlemini yapmayı kolaylaştırır. Kodda değişiklik yaptığımızda, Unit Testi çalıştırıp oluşturduğumuz algoritmaya uygun bir şekilde çalışıp çalışmadığını kolaylıkla test edebiliriz.

Bazı Unit Test Framework' leri

JUnit

Spock

NUnit

TestNG

Jasmin

Mocha

Unit test nasıl yazılmalıdır?

1. **Sadece bir senaryo test edilir.**
2. **Kullanılan adımlar belirlenir.**
3. **Test method ismi test edilen senaryonun yansıması olmalıdır.**
4. **Test edilen kısım diğer kısımlardan bağımsız olmalıdır.**

5. **Testlerimiz tam otomatik şekilde çalışmalıdır.**
6. **Hızlı çalışabilmeli ve çabuk sonuçlar vermelidir.**
7. **Okunaklı, anlaşılabilir ve sürdürebilir olmalıdır.**
8. **Test başarısız olduğunda durmalı ve iyi bir hata raporu döndürmelidir. Bu hata raporunda neyi test ettin ? ne yapmalı ? beklenen çıktı neydi ve gerçekte ne yaptığıdır ?**

JUnit4

Java'da birim testi yazmamıza olanak sağlayan test frameworktür.

En çok kullanılan anatosyanları

@Before(): Test durumundan önce oluşturmak istediğimiz kaşulları bu annnotationu yazıp oluşturuyoruz.

@BeforeClass():Tüm testler için önceden yapılması gereken işlemleri yaparız.

@After(): Değişkenleri sıfırlamak, geçici dosyaları silmek, değişkenleri silmek gibi her Test Durumundan sonra bazı ifadeleri çalıştırmak istiyorsanız bu açıklama kullanılabilir .

@AfterClass():Tüm test durumarından sonra çalıştırmak istediklerimizi yazarız.

@Test(): Public methodlar için test oluşturacaksak bu annotationu kullanırız ama private methodlar için oluşturacaksak **@org.junit.Test** annotationunu ekleriz.

@Test(timeout=500)eğer süreli test edeceksek ve testin exceptionlardan etkilenmesini istemiyorsak **@Test(expected=IllegalArgumentException.class)**bu annotationu kullanırız.

@Ignore():Üzerine koyduğumuz testi çalıştırmaz.

JUnit Assertions

assertEquals(): Beklenen ve gerçekleşen methodun eşit olup olmadığını döner.

AssertNotNull() and *assertNull():Beklediğimiz veye gerçekleşen değerin null dönüp dönmediğini control ederiz.*

AssertNotSame() ve *assertSame():Oluşturulan 2 referansın aynı nesneden üretilip üretilmediğini control ederiz.*

AssertTrue() ve *assertFalse():Belli bir şartı sağlamak istediğimizde kullanırız.*

assertThat():Mesela oluşturduğumuz bir dizinin sahip olduğu değerlerin içinde var olup olmadığına bakarız.

JUnit 5

JUnit 5 ‘ e JUnit 4’ten sonra yenilikler katılmıştır.

· @Test – Bunu bir test method olarak işaretler ve test plugini ile çalışmasını sağlar

· @TestFactory – Private veya static methodlar için kullanılır, Stream, collection, iterable’a ait dinamik testler yazılır.

· @DisplayName – Test isimlerinin yazılması için kullanılan bir annotationdur. @DisplayName(“MyClass”)

· @BeforeAll/@BeforeEach – lifecycle method içinde yer alır. Her @Test, @RepeatedTest, @ParameterizedTest, @TestFactory annotationu gerçekleşmeden önce bu methodların içinde belirtilen classlar yeniden yaratılır.

· @AfterAll/@AfterEach – lifecycle methods içinde yer alır. Test yöntemleri uygulanıktan sonra yapılacaklar bu methodlar altında yer alır.

· @Tag -Testi bir özellik ile belirtmek istiyorsak bu kullanılır. @Tag(“hız”)

· @Disabled –Kullanılmayan testlerde testi geçmesi için kullanılır.

· @Nested- Testlerin sırasını control etmek için kullanılır.

JUnit5 Assertions

assertEquals() and assertNotEquals() : Bu methodta beklenen ve gerçekleşen değerlerin aynı olup olmadığını control etmek için kullanılır. assertEquals(“Selan”, new String(“Selam”));

assertArrayEquals(): Beklenen ve gerçekleşen dizilerin aynı olup olmadığını control ederken kullanılır.

assertThrows(): Eğer exception atıyor ise bu yapı kullanılır.