Enhetstesting – Test Doubles

Agenda

- Test Doubles: Hva og hvorfor
- Dummy, Fake, Stub, Spy og Mock
- Mocking med Mockito i Java

Når enheter er avhengig av andre enheter...

- En gitt enhet kan ofte være avhengig av en eller flere andre enheter
  - Kalles noen ganger «collaborators»
- Dette kan fort bli et problem når vi tester enheten
  - Vi ønsker å teste enhetens funksjonalitet i isolasjon
    - Feil er i så fall begrenset til enhetens kode og er lett å feilsøke
  - Collaborators kan i større eller mindre grad påvirke enhetens funksjonalitet
    - Funksjonaliteten varierer basert på noe utenfor enheten selv
    - Dette er uforutsigbart og vanskelig å feilsøke!
- Vi må på et vis sørge for at slike collaborators ikke forvirrer testresultatene

Test Doubles

- Løsningen er å isolere enheten under testing ved å erstatte collaborators med noe vi faktisk har kontroll over

- Teknikken kalles å lage test doubles
  - Falske versjoner av collaborators
    - Har de samme egenskapene, men uten funksjonalitet
    - Testen bestemmer resten
  - Fordelen: Alt styres av testen og enheten som testes

- Typer: Dummy, Fake, Stub, Spy og Mock

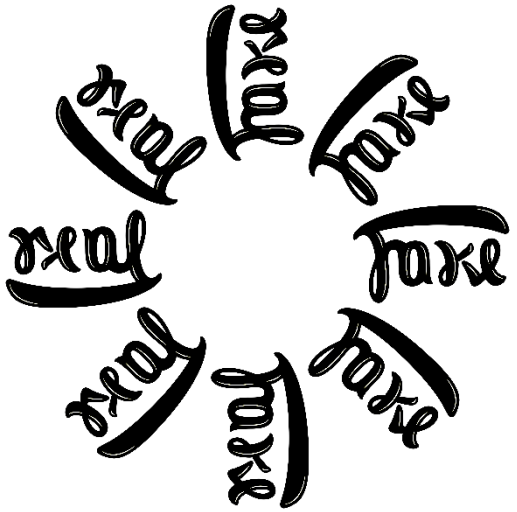
Dummy

- Et objekt som påkreves ved kjøringen av enheten som testes, men som ikke egentlig brukes til noe i testen (direkte eller indirekte)
- Typisk en påkrevd parameter for et annet objekt som faktisk benyttes
  - Altså bare et syntaktisk krav. Innholdet av objektet spiller ingen rolle.
  - Ofte et tegn på dårlig kode i objektet som benyttes...
- Vi kan løse dette ved å
  - Opprette et vanlig objekt av den påkrevde klassen
  - Lage en barneklasse av den påkrevde og sende med et objekt av den
  - Hvis det gjelder et interface kan vi opprette en ny klasse basert på dette

https://www.beaulieu.co.uk/news/visitors-suggest-names-for-beaulieus-crash-test-dummy/

Fake

- Typisk en forfalsket, statisk respons ment å komme fra slikt som
  - Databaser
    - Byttes ut med midlertidige lister i minnet
    - Byttes ut med statiske resultater av spørring
  - Filsystemer
    - Byttes ut med resultat-strings eller byte-buffer i minnet
  - Webservere
    - Byttes ut med en lettviktig webserver som ikke benyttes i produksjon
  - Andre typer services...



Stub

- En Stub er et objekt som funksjonelt benyttes i testen, men som vi gjør forutsigbar
  - Typisk sette faste retur-verdier for benyttede metoder
  - Meget vanlig å benytte stubbing i testing
- En Stub kan defines ved å
  - lage en barneklasse av den orginale og override relevante metoder
  - Hvis interface: lage en ny klasse som implementerer interfacet
    - Eller anonym indre klasse
  - Kan også oppnås via Mocking (senere slides)

https://networklessons.com/wp-content/uploads/2013/02/stub-tree.jpg

Spy

- En Spy er et type objekt som erstatter et orginalt objekt og sporer bruk under testing

- En Spy kan gi innsikt i slikt som
  - Antall ganger metoder blir kalt
  - Argumenter som blir gitt til metoder
  - Rekkefølge av metode-kall
  - Objektet tilstand

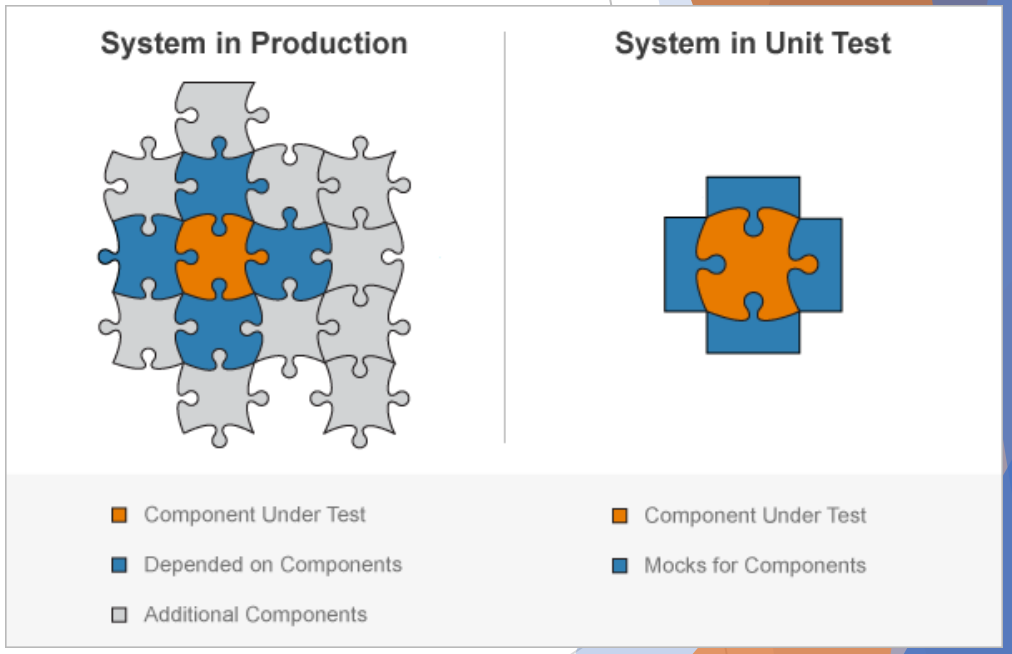
- Eksempel på bruksområder:
  - Kontrollere interaksjon mellom relevante objekter (integrasjonstesting)
  - Kontrollere kodeflyt for effektiv kjøring (kodeoptimalisering)
  - Hvis resultatet er et metodekall (Altså ikke en retur-verdi)

Mock

- En Mock kan tenkes på som en blanding av Stub og Spy

- Kan lage et falsk objekt av et originalt - Stub
  - Kraftigere enn en Stub
  - Vi kan definere betingelser og nøyaktig hva som skal skje ved disse
    - F.eks. Når en metode kalles med X argument, returner Y verdi
  - Ofte mer intuitiv å benytte enn en Stub (er avhengig av rammeverk)

- Vi kan observere hvordan Mock-objektet benyttes - Spy
  - Ulempen er at Mock-objektet ikke er ekte, mens en Spy er et ekte objekt med ekte funksjonalitet
    - Dermed litt forskjellige bruksområder



https://www.mathworks.com/help/matlab/matlab_prog/create-mock-object.html

Bruk av Test Doubles i Java - Mockito

- Jeg kan anbefale å benytte rammeverket Mockito i Java
  - Kan lage alle former for Test Doubles med relativt enkel og konsis syntaks
  - Det kan likevel være tilfeller hvor det er enklere å gjøre det «manuelt»
- Må legges til som en dependency i pom.xml
  - I tillegg til JUnit

```
<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter</artifactId>
        <version>5.10.2</version>
    </dependency>
    <dependency>
        <groupId>org.mockito</groupId>
        <artifactId>mockito-junit-jupiter</artifactId>
        <version>5.11.0</version>
        <scope>compile</scope>
    </dependency>
    <dependency>
        <groupId>org.mockito</groupId>
        <artifactId>mockito-core</artifactId>
        <version>5.11.0</version>
    </dependency>
</dependencies>
```

Mockito – Lage en Mock(1)

- Si at vi ønsker å opprette en Mock av en klasse SomeClass
  - Har et par metoder

```
public class SomeClass {

    private int variable;

    public int complicatedMethod() {
        // Something complicated takes place
    }

    public void doSomething() {
        // Does something without return
    }
}
```

- Når vi lager en test-klasse må vi spesifisere at denne skal kunne lage Mocks
  - Legg til @ExtendWith(MockitoExtension.class) over klassedefinisjonen

```
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.junit.jupiter.MockitoExtension;

@ExtendWith(MockitoExtension.class)
public class SomeTests {

}
```

Mockito – Lage en Mock (2)

- Det er flere måter å opprette et Mock-objekt i Mockito
- En måte er ved bruk av den statiske metoden Mockito.mock(<klasse>)
  - Kan forkortes til mock() ved statisk import statement av Mockito

```
import static org.mockito.Mockito.*;

@ExtendWith(MockitoExtension.class)
public class SomeTests {

   @Test
   public void aTest() {
      SomeClass mockSomeClass = mock(SomeClass.class);
      // do something with mockSomeClass
   }
}
```

Mockito – Lage en Mock (3)

- En annen måte er å deklarere Mock-en som en global variabel i test-klassen
  - Benytter @Mock
  - Vi slipper å deklarere på nytt i alle tester som Mock-er denne klassen

```
@ExtendWith(MockitoExtension.class)
public class SomeTests {

   @Mock
   SomeClass mockSomeClass;

   @Test
   public void aTest() {
      // do something with mockSomeClass
   }
}
```

Mockito – Bruke en Mock – Stubbing (1)

- Vi kan stubbe resultater fra mockens metoder
  - Altså sette statiske resultater for definerte betingelser
- I koden under sier vi at mockSomeClass.complicatedMethod(1) skal returnere 42:
  - Syntax: Mockito.when(<metodekall på mock>).thenReturn(<returverdi>);
  - Alle andre (udefinerte) parametere vil returnere int sin standardverdi (altså 0)

```
@Mock
SomeClass mockSomeClass;

@Test
public void testStubbing() {
   Mockito.when(mockSomeClass.complicatedMethod(1)).thenReturn(42);

   int result = mockSomeClass.complicatedMethod(1);

   Assertions.assertEquals(42, result);
}
```

Mockito – Bruke en Mock – Stubbing (2)

- En alternativ syntax for stubbing er å benytte
  - Mockito.doReturn(<returverdi>).when(<mockingobjekt>).<metodekall>;

```
Mockito.doReturn(42).when(mockSomeClass).complicatedMethod(1);
```

- Dette er nyttig i to tilfeller
  - Mockito.when() gjør et faktisk metodekall. Hvis dette er problematisk (f.eks. medfører exception) kan vi benytte Mockito.doReturn(), som ikke kaller metoden
  - Mockito.when() kan ikke benyttes på void metoder mens Mockito.doReturn() kan. I slike tilfeller kan Mockito.doNothing() også benyttes:

```
Mockito.doNothing().when(mockSomeClass).doSomething();
```

  - Ellers er Mockito.when() ofte foretrukket på grunn av lesbarhet

Mockito – Bruke en Mock – Stubbing (3)

- Vi kan også benytte enkelte ferdigdefinerte metoder for å tilpasse betingelser/parametere

- F.eks. at betingelsen gjelder alle mulige verdier av en gitt datatype
  - Mockito.anyString()
  - Mockito.anyInt()
  - Mockito.anyDouble()
  - ....

```
Mockito.when(mockSomeClass.complicatedMethod( Mockito.anyInt() )).thenReturn(1337);
```

- Mockito kan også matche String-parametere med Regex
  - Mockito.matches(<regex>)

Mockito – Bruke en Mock – Verifisering (1)

- Vi kan verifisere forskjellige typer bruk av Mock-objekter (Spy-funksjonalitet)
  - Om en metode blir kalt
  - Hvor mange ganger en metode blir kalt
  - Om en metode IKKE blir kalt
  - Rekkefølge av metode-kall
  - Osv. [Guide]

Mockito – Bruke en Mock – Verifisering (2)

- Synax for å verifisere en metode:
  - Mockito.verify(<mock-objekt>, <verifikasjons-type>).<metode>;
- I koden under verifiserer vi at doSomething() blir ...
  - Kallt minst én gang - Mockito.atLeastOnce()
  - kalt nøyaktig 2 ganger - Mockito.times()
  - Samt at vi verifiserer at metoden complicatedMethod() ikke blir kalt – Mockito.never()

```
@Test
public void testVerify() {
   Mockito.doNothing().when( mockSomeClass ).doSomething();

   mockSomeClass.doSomething();
   mockSomeClass.doSomething();

   Mockito.verify( mockSomeClass, Mockito.atLeastOnce() ).doSomething();
   Mockito.verify( mockSomeClass, Mockito.times(2) ).doSomething();
   Mockito.verify( mockSomeClass, Mockito.never() ).complicatedMethod( Mockito.anyInt() );
}
```

Kodeeksempel - Mocking

- Se på Mock-funksjonalitet for seg self
  - Opprette, stubbe, verifisere...
- Se på et mer praktisk eksempel
  - Enhet for å beskrive filmer