Enhetstesting med AsUnit

Innledning

Læreboka IT-2 Programmering i ActionScript 3.0 omtaler enhetstesting på side 169. Her skal vi lære om systemet AsUnit for å vise hvordan vi kan gjennomføre slike enhetstester. Enhetstesting inngår blant annet i Test drevet utvikling (*Test Driven Development - TDD*) som er en smidig (*agile*) utviklingsmodell hvor vi skriver testprogrammet før vi skriver selve programmet. I vår sammenheng er en enhet en funksjon i en klasse. Enhetstester er altså testprogram som avgjør om funksjonene i det programmet vi har laget (skal lage) oppfører seg korrekt i henhold til spesifikasjonene. Dersom du ønsker å vite mer om enhetstesting, vil du finne mye informasjon om du googler *unit testing*.

AsUnit byr på flere metoder og klasser som forenkler arbeidet med å skrive testprogrammet. På det laveste nivået finner vi Assert metodene hvor vi sjekker om en påstand er sann eller usann. Dersom den er usann, sier vi at testen feiler. For å teste en funksjon lager vi en testfunksjon som inneholder flere kall til Assert metoder. Når en test feiler, avbrytes testfunksjonen og de gjenværende Assert metodene vil ikke bli utført. Feilen fanges opp av testprogrammet som fortsetter med neste testfunksjon. La oss si vi har en funksjon som legger sammen to tall:

```
function leggSammen(a:int, b:int):int {
   return a+b;
}
```

I testprogrammet har vi testfunksjonen

```
function testLeggSammen():void {
   assertEquals(”tre pluss to er fem”, 5, leggSammen(2,3);
   assertEquals(”fire pluss null er fire”, 4, leggSammen(4,0);
}
```

Testfunksjonene begynner med ordet `test` og så kommer navnet på funksjonen som skal testes. Den første parameteren i Assert metoden er en frivillig melding som vises dersom testen feiler. Den andre parameteren er hva vi forventer at funksjonen `leggSammen()` skal returnere. Den tredje parameteren er hva funksjonen `leggSammen()` virkelig returnerer. Dersom forventet verdi og virkelig verdi ikke er like, feiler testen. AsUnit byr på en rekke Assert metoder:

- http://asunit.org/docs/asunit3/asunit/framework/Assert.html

Testfunksjonene legger vi inne i en testklasse som har det samme navnet som klassen vi skal teste, med ordet `Test` lagt til på slutten. I eksempelet vi skal gjennomføre, tester vi klassen `Konto` med en testklasse som AsUnit oppretter og gir navnet `KontoTest`. AsUnit gjør dessuten `KontoTest` til subklasse av TestCase som gir tilgang til alle Assert metodene.

AsUnit oppretter også klassen `AllTests` som er en samling av alle testklassene. `AllTests` er subklasse av TestSuite, som således er en samling av TestCase'er.

For å starte det hele, må vi skrive en liten klasse som vi har kalt `BankTestRunner`, og som må være subklasse av TestRunner. I konstruktøren til klassen vi lager, angir vi at det er `AllTests` som skal kjøres. Vi ender altså opp med å kjøre en TestRunner (`BankTestRunner`) som starter en TestSuite (`AllTests`) som består av en eller flere TestCase'er (`KontoTest`).

Hver testfunksjon skal kjøres uavhengig alle andre testfunkjsoner. Dette oppnås med funksjonen `setUp()` som kalles før testfunksjonen kjøres, og funksjonen `tearDown()` som kalles etterpå. AsUnit oppretter funksjonene for oss. I eksempelet vi skal gjennomføre opprettes en ny instans av `Konto` klassen i `setUp()` for så å fjernes igjen i `tearDown()`.

Installasjon

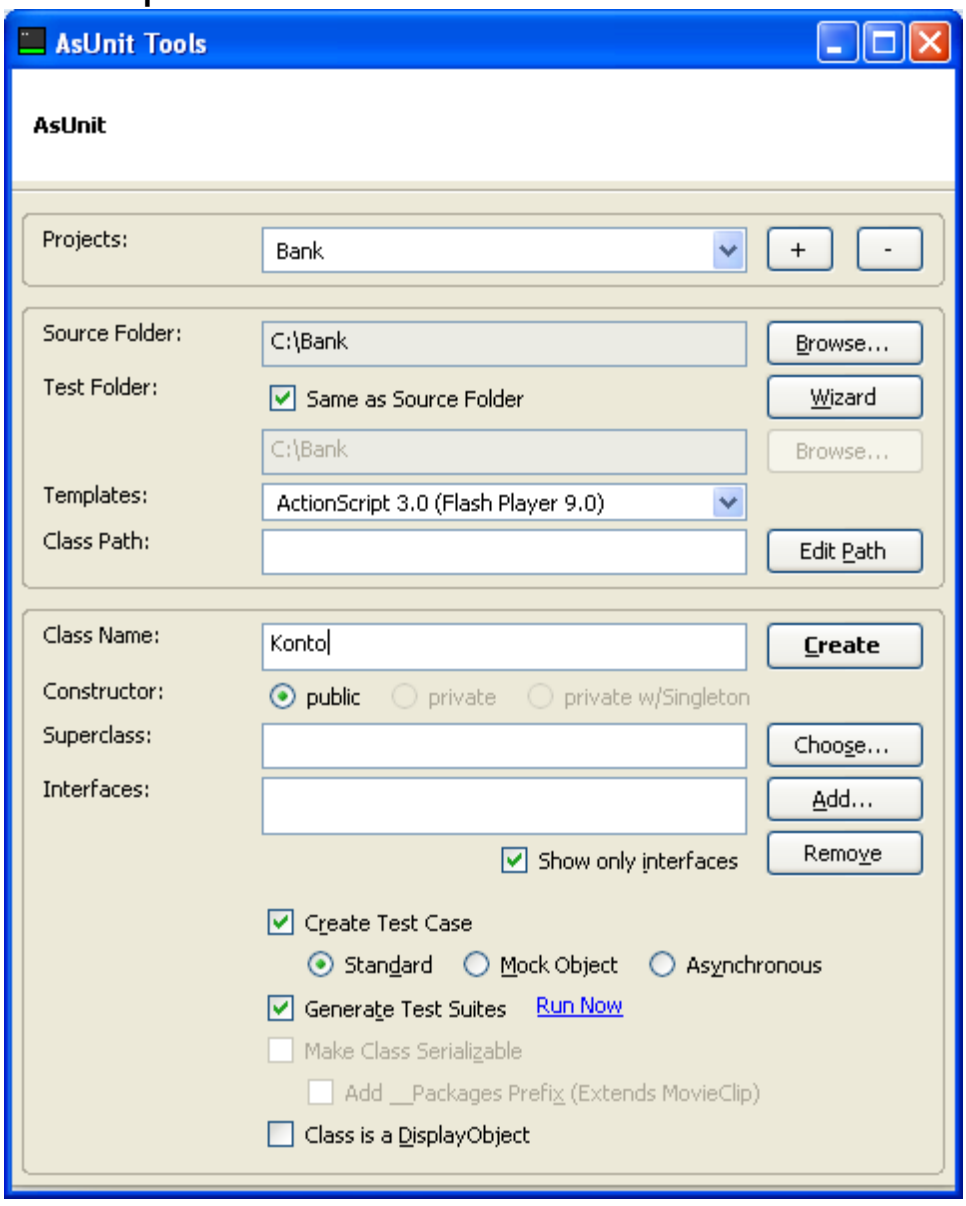
AsUnit finnes som en kjørbar .msi fil på

- http://sourceforge.net/projects/asunit/files/

som foreslår å bruke C:\Programfiler\AsUnit mappen. Det er ikke så mye dokumentasjon om AsUnit, men det har mye felles med JUnit for java. Se for øvrig:

- http://asunit.org/

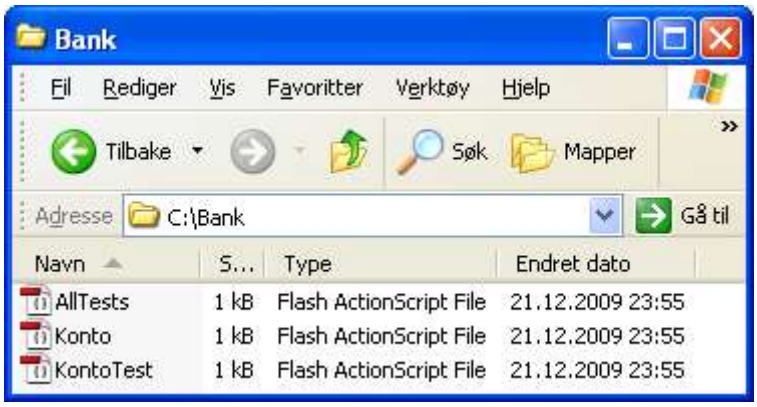
Eksempel



Vi skal lage og teste en `Konto` klasse med en variabel (attributt), `saldo`, og tre funksjoner (metoder): `settInn()`, `taUt()` og `getSaldo()`.

Begynn med å starte AsUnit.exe fra Start menyen eller skrivebordet. Se figuren til venstre og trykk på pluss knappen til høyre for *Project*. I veiviseren som følger, velg *Project Name* Bank, *Templates* ActionScript 3.0, *Source Folder* C:\bank, *Test Folder* Same as Source Folder, *Choose the Class Path* Next og Finish.

Class Name Konto, huk av for *Create Test Case*, *Standard* og *Generate Test Suites*. Klikk *Create*

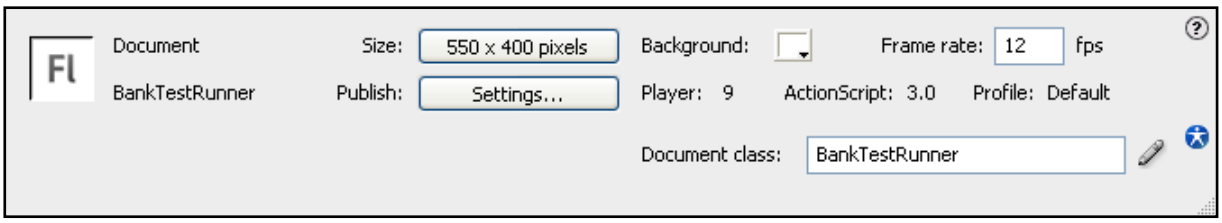


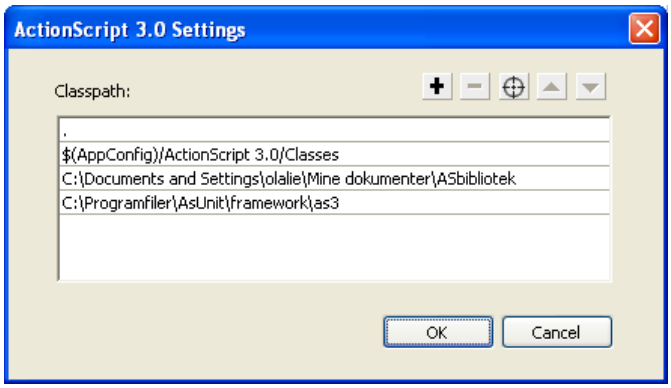
Tre filer blir opprettet: `AllTest.as`, `Konto.as` og `KontoTest.as`

Start Flash C3 og lag en ActionScript fil som du kaller `BankTestRunner.as` og legger i mappen C:\bank. Innholdet av filen (programmet) ser du nedenfor.

```
package {
    import asunit.textui.TestRunner;

    public class BankTestRunner extends TestRunner {
        public function BankTestRunner() {
            start(AllTests, null, TestRunner.SHOW_TRACE);
        }
    }
}
```

Lag en Flash fil (ActionScript 3.0) som lagrer i mappen C:\bank og kaller `BankTestRunner.fla`. Dokumentklassen til flash filen setter du til `BankTestRunner`:





Ha med C:\Programfiler\AsUnit\framework\as3 i Classpath ved å velge *Edit* → *Preferences…* → *ActionScript* → *ActionScript 3.0 Settings*. Se figuren til høyre.

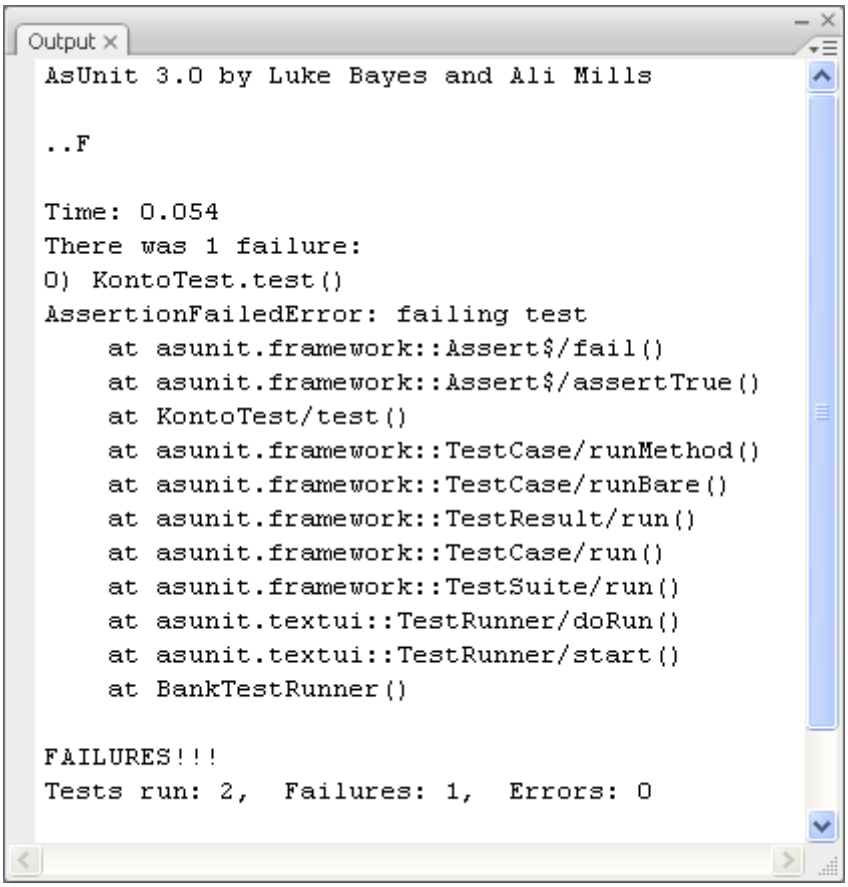
`KontoTest` klassen inneholder to testfunksjoner som er opprettet av AsUnit, `testInstantiated()` og `test()`:

```
public function testInstantiated():void {
      assertTrue("Konto instantiated", instance is Konto);
}

public function test():void {
      assertTrue("failing test", false);
}
```

Når vi tester `BankTestRunner.fla` med Ctrl-Enter, kjøres de to testene.

Den første testen, som sjekker at vi har fått opprettet et objekt av klassen `Konto`, går bra. Instansen blir opprettet av `setUp()` og variabelen som peker på instansen heter `instance`.

Den andre testen er en falsk test som alltid vil feile.



La oss fjerne den siste testen og lage `testSettInn()` og `testTaUt()` i `KontoTest`:

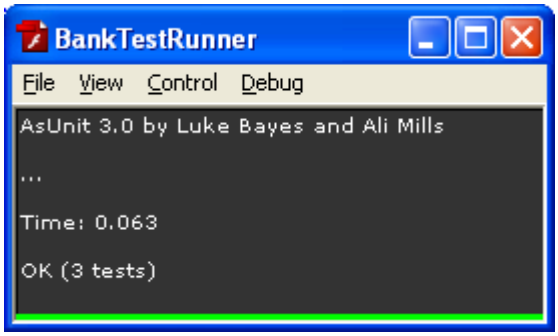
```
public function testSettInn():void {
   instance.settInn(800);
   assertTrue("Etter å ha satt inn 800 på en nyopprettet konto "
     + "skal saldo være 800", instance.getSaldo() == 800);
   instance.settInn(400);
   assertEquals("Etter å ha satt inn enda 400 skal saldo være 1200",
     1200,instance.getSaldo());
}

public function testTaUt():void {
   instance.settInn(5000);
   instance.taUt(1500);
   assertTrue("Etter innskudd på 5000 og uttak på 1500 skal saldo på "
     + "en nyopprettet konto være 3500", instance.getSaldo() == 3500);
}

/*
public function test():void {
      assertTrue("failing test", false);
}
*/
```

`testSettInn()`, setter først inn 800 kroner på en ny konto. Så henter den saldoen og forsikrer seg om at den også er 800 kroner. Så settes det inn ytterligere 400 kroner, og deretter sammenlignes saldoen med 1200.

`testTaUt()`, setter først inn 5000 på en ny konto. Deretter tas det ut 1500. Til slutt hentes saldoen og sammenlignes med 3500.

Når vi tester BankTestRunner.fla med Ctrl-Enter, får vi syv kompilator feil fordi vi ikke har skrevet programmet ennå! Bare testene. Så vi skriver programmet (funksjonene i `Konto` klassen) som vist nedenfor:

```
package  {

   public class Konto {
     private var saldo:Number = 0;

     public function settInn(belop:Number) {
      saldo += belop;
     } // slutt metode settInn

     public function taUt(belop:Number) {
      saldo -= belop;
     } // slutt metode taUt

     public function getSaldo():Number {
      return saldo;
     } // slutt metode getSaldo

   } // slutt klasse konto
} // slutt pakke
```



Når vi nå kjører BankTestRunner.fla, får vi ingen feil.

For å se hvordan det tar seg ut når en test feiler, kan du sammenlignes saldoen med 3000 i stedet for 3500 i `testTaUt()`.

Til slutt

Med nok erfaring forutser man hva som sannsynligvis kan gå galt og tester disse områdene først. Men det kan være vanskelig å tenke på alt som kan få en funksjon til å feile. Et hjelpemiddel til å skrive gode tester kan være retningslinjene som vi finner på

- http://media.pragprog.com/titles/utj/StandaloneSummary.pdf

Filene som er brukt i eksempelet ovenfor finner du på

- http://tip.no/skole/res/AsUnitEksempel.zip