UML anvendt i IT-2

Introduksjon

Vi skal først se på ulike anvendelser av UML. Deretter skal vi installere en versjon Violet UML editor som lar oss lage ActionScript 3 kode av klassediagram. Til slutt skal vi anvende verktøyet i et eksempel: Terningsspillet Syv.

Kompetansemålene under Planlegging og dokumentasjon i IT-2 omfatter blant annet:

- *velge og bruke relevante teknikker og verktøy for planlegging og utvikling av IT-løsninger*
- *forklare hensikten med teknisk dokumentasjon og lage slik dokumentasjon for IT-løsninger, med spesiell vekt på å dokumentere grensesnitt mellom ulike delsystemer*

Læreboka *IT-2 Programmering i ActionScript 3.0* har med en side om Modellering i UML (side 167). Dette dokumentet er ment som tilleggsstoff og forutsetter at du er kjent med Klasser (side 180) og Dokumentklassen (side 217).

Litt mer om UML

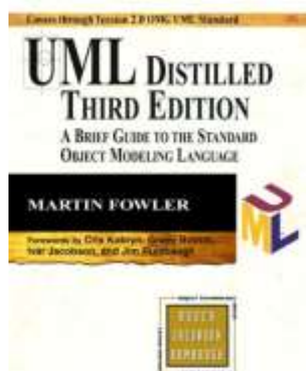
UML er et modelleringsspråk, et system av grafiske framstillinger (standardiserte diagrammer) for å beskrive og designe programvare, spesielt objekt orienterte informasjonssystemer (IT-løsninger/multimedieapplikasjoner).

Den tradisjonelle prosessen med å lage kode med utgangspunkt i modeller (f. eks UML diagrammer) kalles "*forward engineering*". Det motsatte, å lage modeller med utgangspunkt i kode, kalles "*reverse engineering*". Hensikten med *reverse engineering* er å hjelpe oss til å forstå koden (f. eks når du blir bedt om å endre på et program noen andre har laget).

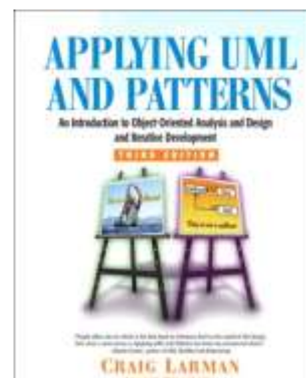
Det er vanligst å benytte UML som en *skisse* til å diskutere alternative løsninger med andre utviklere.

Hvis man bruker UML som en *blåkopi*, har designeren laget så fullstendige UML diagrammer at det er ganske enkelt for en programmerer å skrive koden uten altfor mye tankevirksomhet.

Når programmeringen blir mer mekanisk, kan den automatiseres. Noen UML verktøy, f. eks Violet UML editor (åpen kildekode) har "*forward engineering*" egenskaper og kan lage deler av koden basert på UML diagrammer (f. eks definere klasser, variabler, funksjoner, parametere, osv.). Til slutt når vi det punktet hvor hele systemet kan spesifiseres i UML. UML kan da benyttes som et *programmeringsspråk*. Når UML diagrammene kompileres direkte til kjørbar kode, er UML blitt kildekoden. Da blir det meningsløst å snakke om *forward*- og *reverse engineering* fordi UML og kildekoden er det samme.



Ønsker du å lære mer om UML og objektorientert modellering, kan du lese bøkene til Martin Fowler (*UML Distilled*) og Craig Larman (*Applying UML and Patterns*). Nivået er imidlertid mer tilpasset universitet og høgskole enn videregående skole.



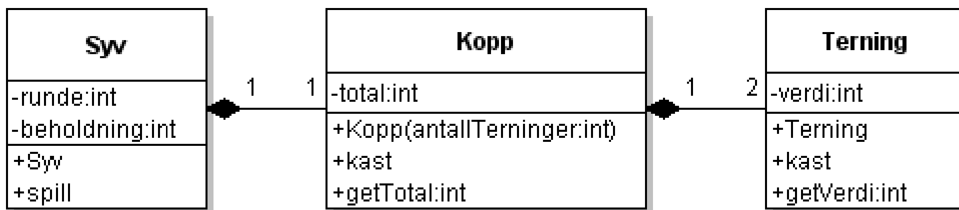
Violet UML editor er veldig lett å lære og lar oss raskt lage enkle UML diagrammer. Det finnes andre gratis UML verktøy som er mer avansert, f.eks ArgoUML, men de lager ikke ActionScript kode. Brukerveiledningen i punkt 3 nedenfor er litt mer detaljert enn den offisielle utgaven i punkt 2. Versjonen som kan lage ActionScript heter VASGen, og kan lastes ned fra punkt 1 nedenfor

1. http://selfmummy.appspot.com/vasgen/Vasgen-0.2.1.jar Vasgen-0.2.1 er en kjørbar JAR fil
2. http://alexdp.free.fr/violetumleditor/page.php?id=en:userguide offisiell brukerveiledning
3. http://www.computing.surrey.ac.uk/courses/csm03/labs/COMM005_Lab_Tutorial_0.9.4.htm

Eksempel: Terningsspillet "Syv"

Når vi skal utvikle et system, undersøker vi først hva systemet skal gjøre. I denne analysefasen fokuserer vi på kravspesifikasjonen og ikke på løsningen. Spillet kan spesifiseres ved følgende bruksmønster: Spilleren får 60 kr og en kopp med to terninger. Det koster 10 kroner og kaste terningene. Dersom summen av terningene blir 7, så vinner spilleren 50 kr. Spilleren får spille 25 ganger, men bare så lenge han har penger.

Når vi vet hva som skal gjøres, er neste skritt å finne ut hvordan vi kan gjøre det. I denne design fasen forsøker vi å få fram en løsningsmodell som kan tilfredsstille kravspesifikasjonen. Mye av utviklingsarbeidet er å dele systemet inn i hensiktsmessige klasser og forstå sammenhengen mellom dem. Dessuten må vi finne ut hvilke attributter (variabler) og metoder (funksjoner) som bør inngå i hver klasse. Det finnes mange prinsipper og kjent mønstre som hjelper oss i dette arbeidet. Nedenfor vises et UML klassediagram for terningsspillet Syv.

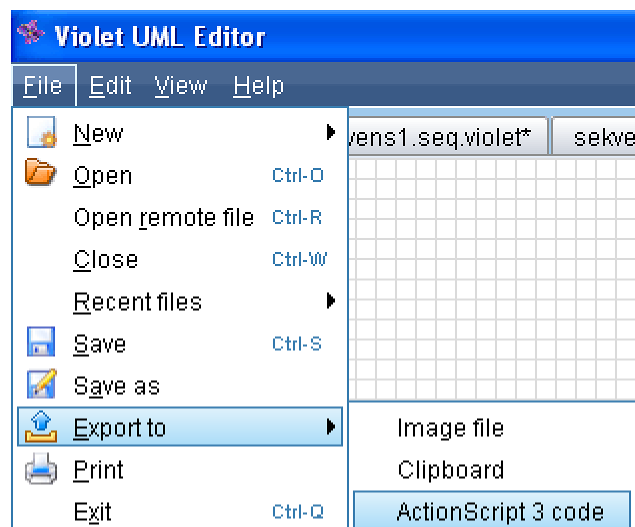


Syv klassen spiller med Kopp klassen som kaster med Terning klassen. Når vi gjør det på denne måten, kan vi blant annet bruke Kopp og Terning klassen i andre spill også.

Når vi har klassediagrammet aktivt i Violet UML Editor, kan vi generere skallet til koden.

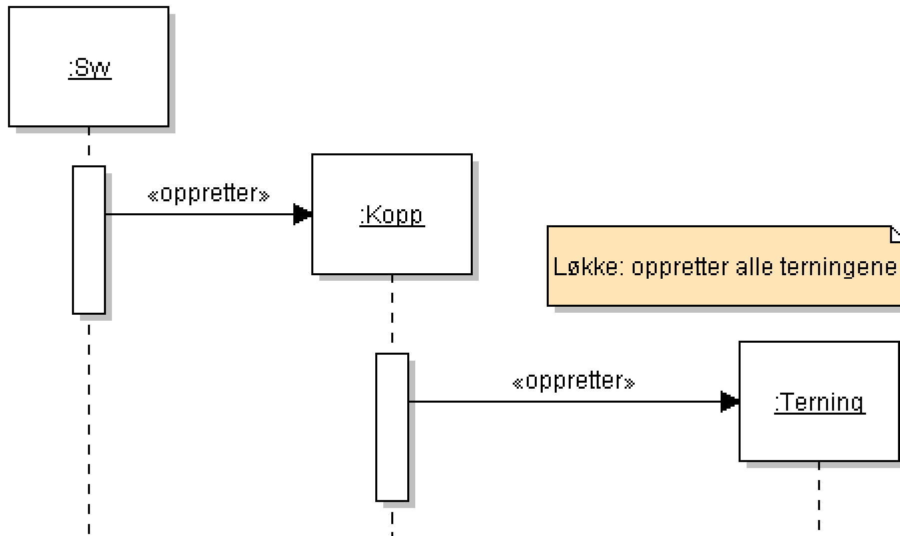
Følgende ActionScript 3.0 (.as) filer blir generert:

Siden vi ikke har opprettet en pakke, havner filene i undermappen VASGenDefault til mappen vi åpner. Du får ingen bekreftelse på at filene har blitt laget, men må sjekke mappen selv.



Versjon 1: Vi oppretter objektene.

Vi velger klassen Syv til å være dokumentklassen (rot). UML sekvensdiagrammet nedenfor viser hvordan objektene lages ved oppstart.



Kopier Syv.as, Kopp.as og Terning.as over til en annen mappe og åpne dem i Flash. Deretter oppretter du en Flash fil i samme mappe som du kaller Syv.fla. Husk å angi dokument klassen Syv i *Properties* panelet til dokumentet. Prøver du å kjøre Syv.fla (*Ctrl-Enter*), får du tre feilmeldinger. Først må du fjerne VASGenDefault etter package i alle .as filene. Deretter må du legge til import flash.display.MovieClip og endre klasse definisjonen til public class Syv extends MovieClip i Syv.as filen.Til sist oppretter du en ny kopp i konstruktøren:

```
package  {

    import flash.display.MovieClip;

    public class Syv extends MovieClip {
        private var runde:int;
        private var beholdning:int;
        private var kopp:Kopp;

        public function Syv():void {
            trace('executing method Syv.Syv');
            kopp = new Kopp(2);
        }

        public function spill():void {
            trace('executing method Syv.spill');
        }
    }
}
```

I kopp konstruktøren oppretter du 2 terninger:

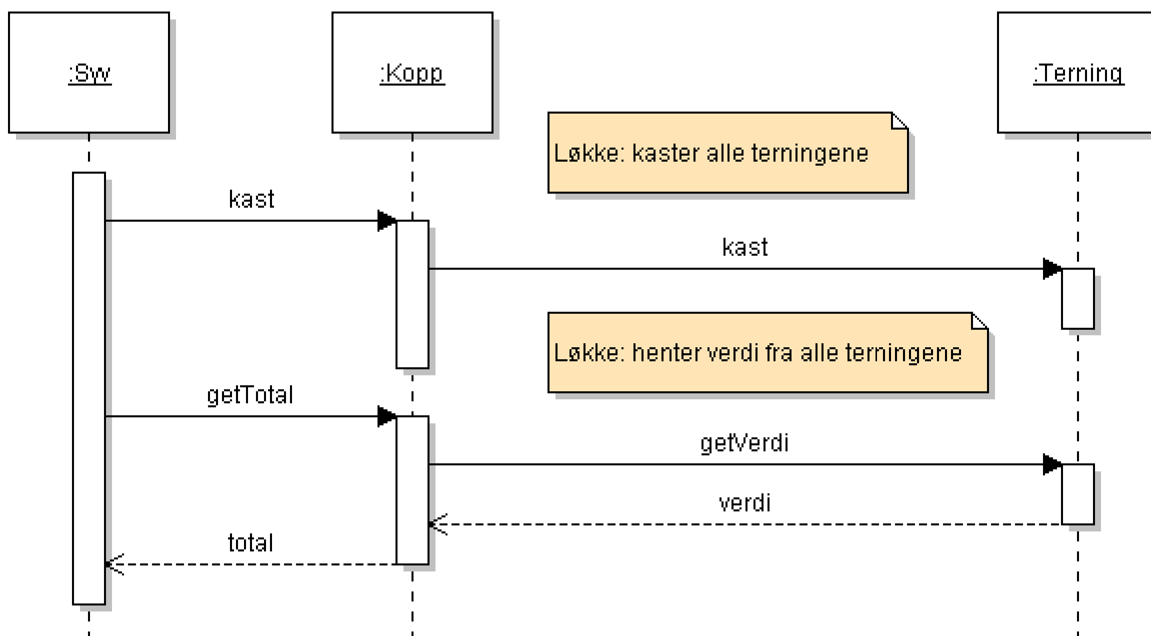
```
package  {
    public class Kopp {
        private var total:int;
        private var terninger:Array = new Array();;

        public function Kopp(antallTerninger:int):void {
            trace('executing method Kopp.Kopp');
            for (var i:int=0;i<antallTerninger;i++) terninger[i] = new Terning();
        }

        public function kast():void {
            trace('executing method Kopp.kast');
        }

        public function getTotal():int {
            trace('executing method Kopp.getTotal');
            return null;
        }
    }
}
```

Dette gir følgende utskrift:

```
Output × | Compiler Errors
executing method Syv.Syv
executing method Kopp.Kopp
executing method Terning.Terning
executing method Terning.Terning
```

Versjon 2: Vi koder metodene.

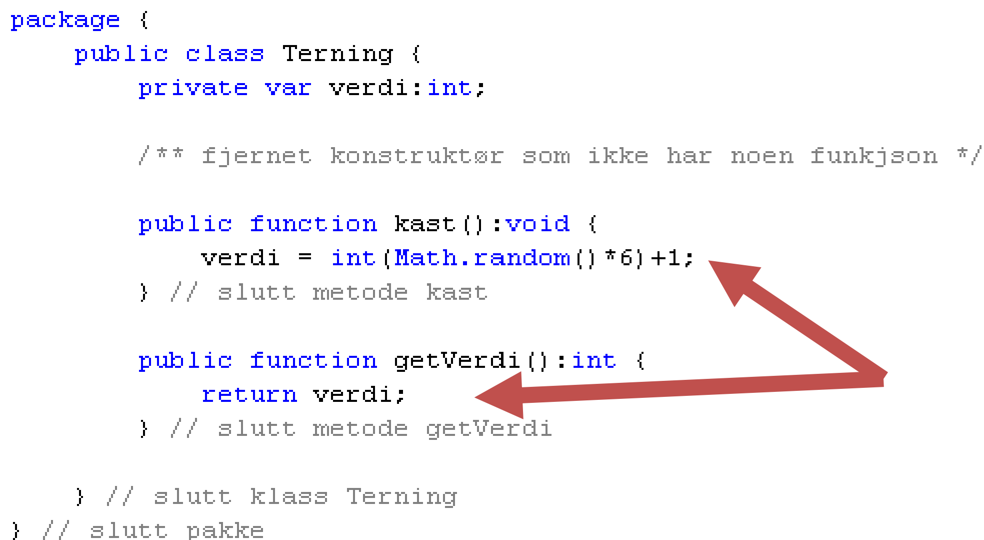
UML sekvensdiagrammet nedenfor viser hvordan en runde foregår.



Dokumentklassen sender en melding til koppen om at terningene skal kastes. Koppen sender meldingen videre til hver av terningene som endrer sin verdi. Deretter spør dokumentklassen koppen om resultatet. Koppen spør så alle terningene om deres verdi og sender totalen tilbake til dokumentklassen. Dette er for øvrig et eksempel på *Command-query separation*. Prinsippet

går ut på at metoder enten skal være en kommando eller en spørring, men ikke begge deler. Metoden kast endrer på terningens tilstand (verdi), og metoden getVerdi returner verdien.

Når en terning kastes, settes en vilkårlig verdi mellom 1 og 6. Metoden getVerdi returnerer denne verdien. Konstruktøren trenger vi ikke lenger:



```
package {
    public class Terning {
        private var verdi:int;

        /** fjernet konstruktør som ikke har noen funkjson */

        public function kast():void {
            verdi = int(Math.random()*6)+1;
        } // slutt metode kast

        public function getVerdi():int {
            return verdi;
        } // slutt metode getVerdi

    } // slutt klass Terning
} // slutt pakke
```

Koppen må vi kode metodene for å kaste alle terningene og svare hva totalen blir. Dessuten må vi opprette en ny variabel som gjør antallTerninger tilgjengelig for metodene:

```
package  {
    public class Kopp {
        private var total:int;
        private var antallTerninger:int;
        private var terninger:Array = new Array();;

        public function Kopp(antallTerninger:int):void {
            this.antallTerninger = antallTerninger;
            for (var i:int=0;i<antallTerninger;i++)
                terninger[i] = new Terning();
        } // slutt konstruktør

        public function kast():void {
            for (var i:int=0;i<antallTerninger;i++)
                terninger[i].kast();
        } // slutt metode kast

        public function getTotal():int {
            total=0;

            for (var i:int=0;i<antallTerninger;i++)
                total += terninger[i].getVerdi();

            return total;
        } // slutt metode getTotal

    } // slutt klasse Kopp
} // slutt pakke
```

Hovedprogrammet kjøres i metoden spill som holder det gående i 25 runder så fremt det er penger igjen:

Syv

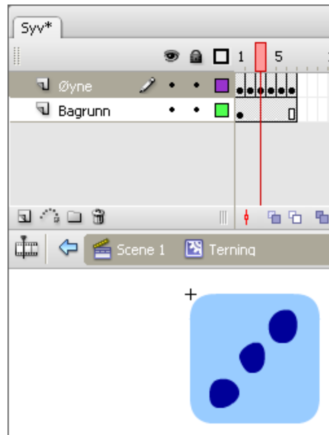
```
package  {

    import flash.display.MovieClip;

    public class Syv extends MovieClip {
        const ANTALL_RUNDER:int = 25;
        private var runde:int = 1;
        private var beholdning:int=60;
        private var kopp:Kopp;

        public function Syv():void {
            kopp = new Kopp(2);
            spill();
        } // slutt konstruktør

        public function spill():void {

            while (runde <= ANTALL_RUNDER && beholdning > 0) {
                kopp.kast();

                if (kopp.getTotal()==7) {
                    beholdning += 50;
                }
                else {
                    beholdning -= 10;
                } // slutt if

                trace ("Runde: " + runde + ", Resultat: " +  kopp.getTotal()
                    + ", Beholdning: " + beholdning);
                runde++;
            } // slutt while

        } // slutt metoden spill
    } // slutt klasse Syv
} // slutt pakke
```

Utskriften vil variere hver gang vi kjører programmet:

Output × | Compiler Errors

```
Runde: 1, Resultat: 11, Beholdning: 50
Runde: 2, Resultat: 8, Beholdning: 40
Runde: 3, Resultat: 9, Beholdning: 30
Runde: 4, Resultat: 8, Beholdning: 20
Runde: 5, Resultat: 7, Beholdning: 70
Runde: 6, Resultat: 9, Beholdning: 60
Runde: 7, Resultat: 5, Beholdning: 50
Runde: 8, Resultat: 10, Beholdning: 40
Runde: 9, Resultat: 3, Beholdning: 30
Runde: 10, Resultat: 3, Beholdning: 20
Runde: 11, Resultat: 3, Beholdning: 10
Runde: 12, Resultat: 2, Beholdning: 0
```

Versjon 3: Vi legger til GUI



Først lager vi et MovieClip symbol av terningen som forklart på side 92 i IT-2 Programmering i ActionScript 3.0. Deretter lager vi et MovieClip symbol som vi kaller Utsyn. Her legger vi ut statiske og dynamiske tekstfelt, en knapp og 2 instanser av terningen.

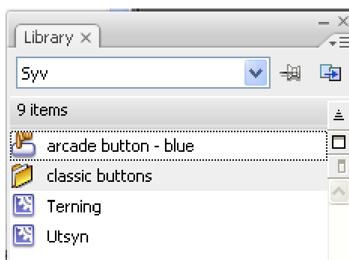
Slå 7 til sammen og vinn kr 50!

Din beholdning

Antall runder

Spill!

Max 25 forsøk

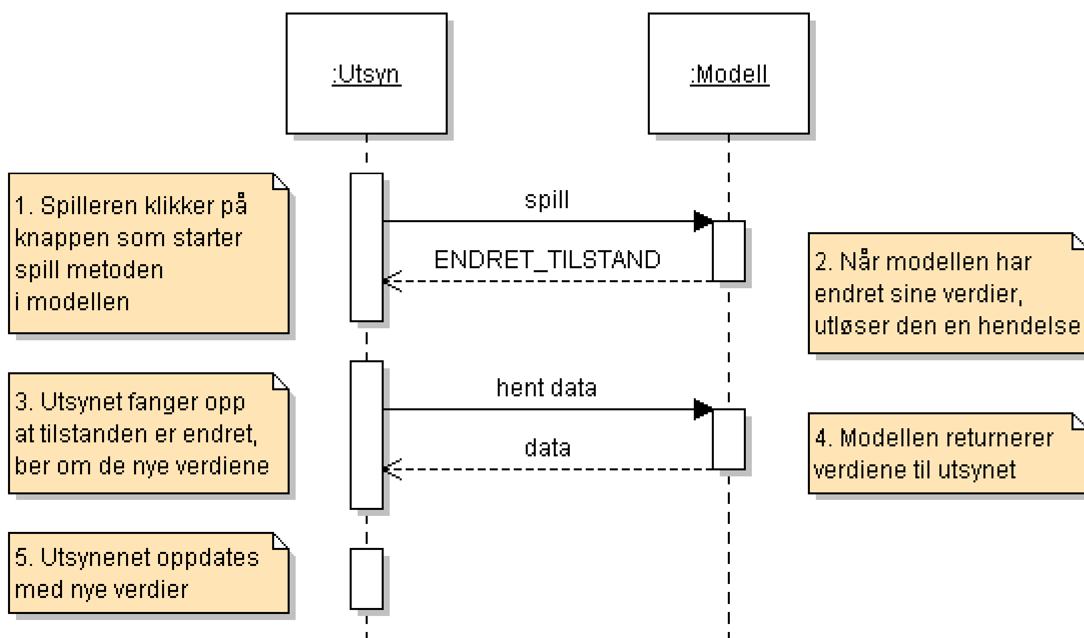


Biblioteket inneholder Terning, Utsyn og en knapp.

Prøv hvordan det skal bli på http://tip.no/skole/res/syv.swf
Vi skiller utsynet fra modellen som forklart på side 216 i læreboka. Først legger vi utsynet til dokumentklassens (Syv) hovedtidslinje.

```
_utsyn = new Utsyn(this,beholdning,runde);
addChild(_utsyn);
```

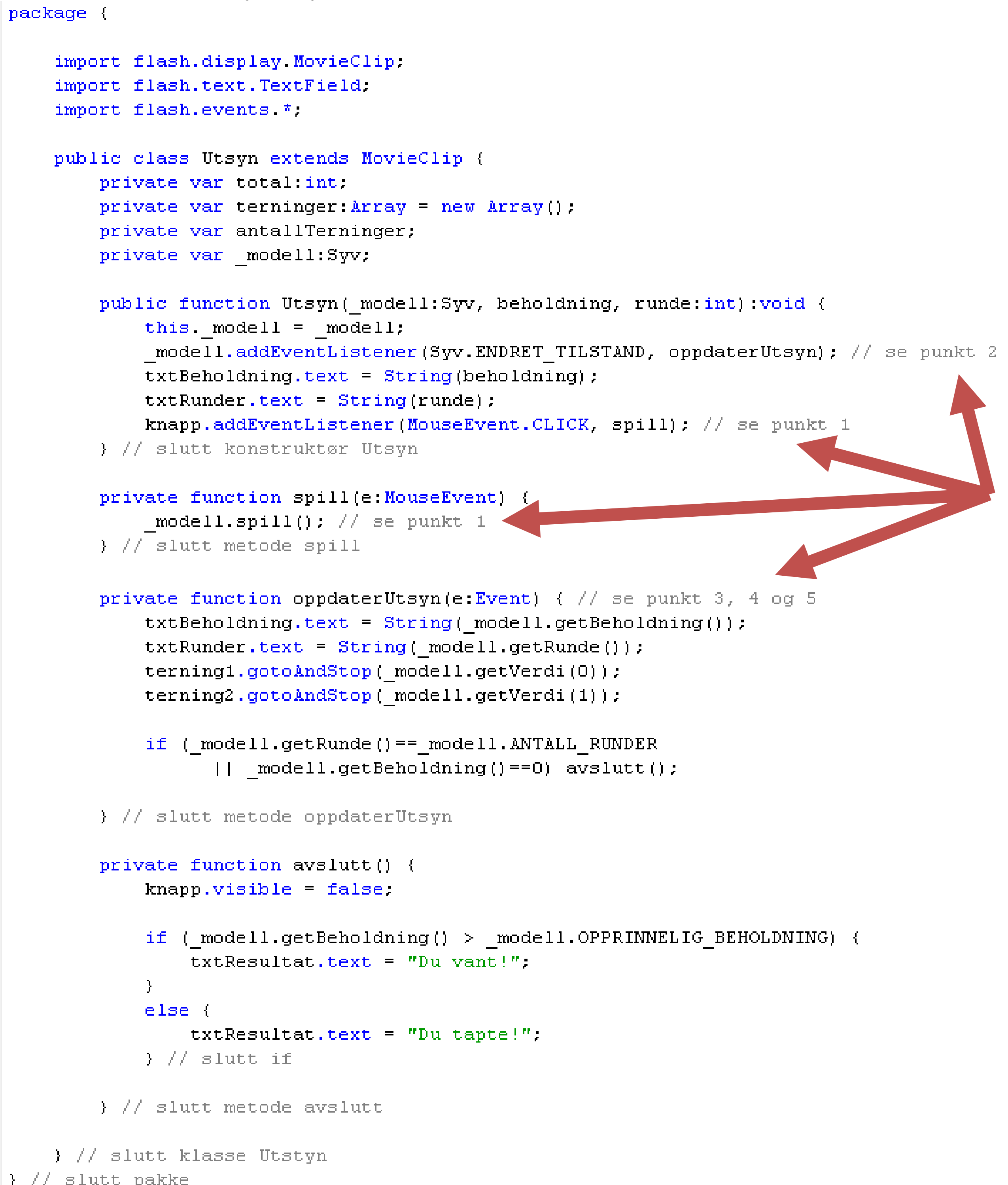
Konstruktøren til utsynet får med seg en referanse til modellen (this) samt oppstart verdiene for beholdning og runde. Den videre programflyten er vist nedenfor. Modellen i denne sammenheng er Syv klassen (med tilhørende Kopp og Terning klasser). De nummererte forklaringene finner du igjen som kommentarer i koden.



Vi trenger ikke å endre Terning klassen, men husk Linkage og Export for ActionScript av Knapp, Terning og Utsyn symbolene i biblioteket. I Kopp klassen må vi legge til en metode for å returnere en bestemt terning sin verdi (hittil kunne modellen kun se totalen):

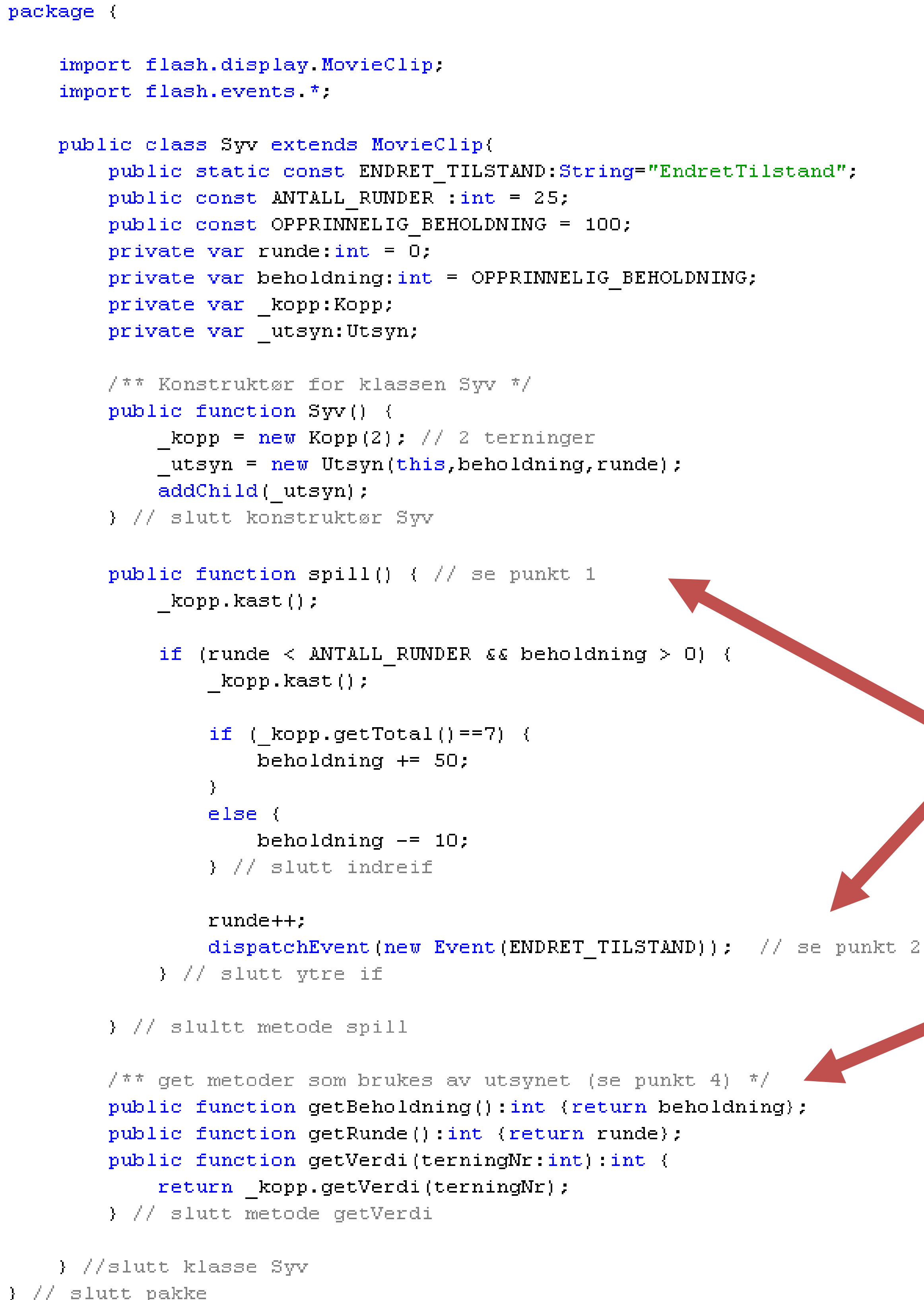
```
public function getVerdi(terningNr:int) {
    return terninger[terningNr].getVerdi();
} // slutt metode getVerdi
```

Her er den ser du den nye Utsyn klassen:



```
package {

    import flash.display.MovieClip;
    import flash.text.TextField;
    import flash.events.*;

    public class Utsyn extends MovieClip {
        private var total:int;
        private var terninger:Array = new Array();
        private var antallTerninger;
        private var _modell:Syv;

        public function Utsyn(_modell:Syv, beholdning, runde:int):void {
            this._modell = _modell;
            _modell.addEventListener(Syv.ENDRET_TILSTAND, oppdaterUtsyn); // se punkt 2
            txtBeholdning.text = String(beholdning);
            txtRunder.text = String(runde);
            knapp.addEventListener(MouseEvent.CLICK, spill); // se punkt 1
        } // slutt konstruktør Utsyn

        private function spill(e:MouseEvent) {
            _modell.spill(); // se punkt 1
        } // slutt metode spill

        private function oppdaterUtsyn(e:Event) { // se punkt 3, 4 og 5
            txtBeholdning.text = String(_modell.getBeholdning());
            txtRunder.text = String(_modell.getRunde());
            terning1.gotoAndStop(_modell.getVerdi(0));
            terning2.gotoAndStop(_modell.getVerdi(1));

            if (_modell.getRunde()==_modell.ANTALL_RUNDER
                  || _modell.getBeholdning()==0) avslutt();

        } // slutt metode oppdaterUtsyn

        private function avslutt() {
            knapp.visible = false;

            if (_modell.getBeholdning() > _modell.OPPRINNELIG_BEHOLDNING) {
                txtResultat.text = "Du vant!";
            }
            else {
                txtResultat.text = "Du tapte!";
            } // slutt if

        } // slutt metode avslutt

    } // slutt klasse Utsyn
} // slutt pakke
```

Og her kommer den modifiserte Syv klassen:



```
package {

    import flash.display.MovieClip;
    import flash.events.*;

    public class Syv extends MovieClip{
        public static const ENDRET_TILSTAND:String="EndretTilstand";
        public const ANTALL_RUNDER :int = 25;
        public const OPPRINNELIG_BEHOLDNING = 100;
        private var runde:int = 0;
        private var beholdning:int = OPPRINNELIG_BEHOLDNING;
        private var _kopp:Kopp;
        private var _utsyn:Utsyn;

        /** Konstruktør for klassen Syv */
        public function Syv() {
            _kopp = new Kopp(2); // 2 terninger
            _utsyn = new Utsyn(this,beholdning,runde);
            addChild(_utsyn);
        } // slutt konstruktør Syv

        public function spill() { // se punkt 1
            _kopp.kast();

            if (runde < ANTALL_RUNDER && beholdning > 0) {
                _kopp.kast();

                if (_kopp.getTotal()==7) {
                    beholdning += 50;
                }
                else {
                    beholdning -= 10;
                } // slutt indreif

                runde++;
                dispatchEvent(new Event(ENDRET_TILSTAND));  // se punkt 2
            } // slutt ytre if

        } // slultt metode spill

        /** get metoder som brukes av utsynet (se punkt 4) */
        public function getBeholdning():int {return beholdning};
        public function getRunde():int {return runde};
        public function getVerdi(terningNr:int):int {
            return _kopp.getVerdi(terningNr);
        } // slutt metode getVerdi

    } //slutt klasse Syv
} // slutt pakke
```

Her ligger kildefilene http://tip.no/skole/res/syv.zip.