2012 – 2a

Vi tenker oss i denne oppgaven at vi har et Java-lignende språk hvor metoder kan ha lokalt definerte metoder. Dessuten kan man deklarere variable og metoder også på ytterste programnivå. Dette skal fungere som vanlig i språk som er statisk skopet.

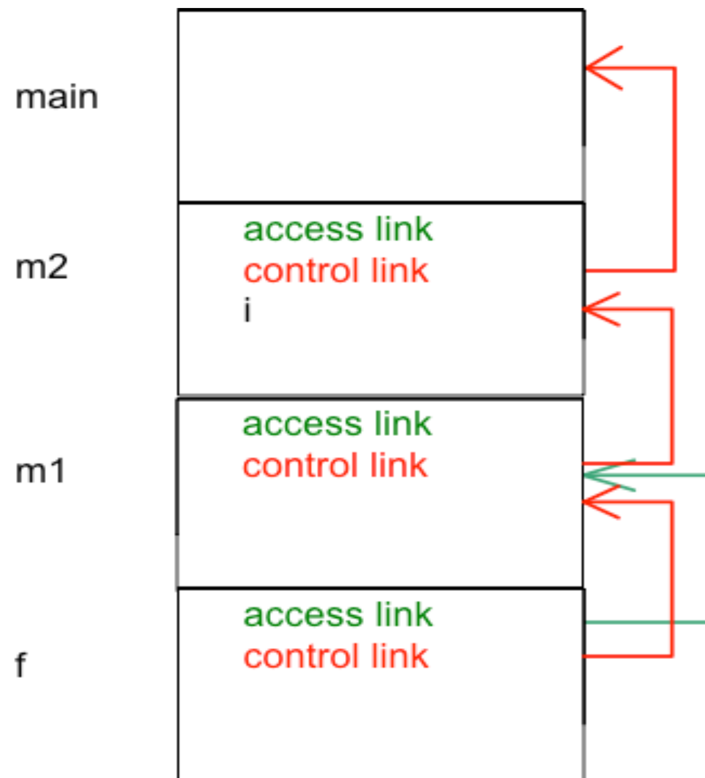
Det følgende er et program i dette språket. Oppstart av programmet skjer ved å kalle main-metoden.

```
{
  class C {
    void m1(){
      void f() {};
      f();
    }
    void m2() {
      int i;
      void g() {
        int j; j=i;
      };
      i=1;rC.m1();
    };
  };
  C rC;
   void main() {
     rC = new C (); rC.m2();
   }
}
```

INF 5110 - 2013

2012 – 2a

Tegn kall-stakken som den ser ut når aktiveringsblokken ('activation record') for f er på toppen av stakken for første gang, inklusive variable, access-linker og control-linker, bortsett fra access-linker for metoder som er direkte deklarert i en klasse (kan da anta at access-linken peker til C-objektet, men dette er ikke viktig for oppgaven).



2012 – 2b

I resten av oppgave 2 innfører vi metoder som parametere. Det er en regel at aktuelle parametere til slike må være metoder som er direkte synlige fra kallstedet. Eksemplet over blir så endret slik at f blir en metodeparameter, og kallet på f i m1 blir da et kall på en metodeparameter:

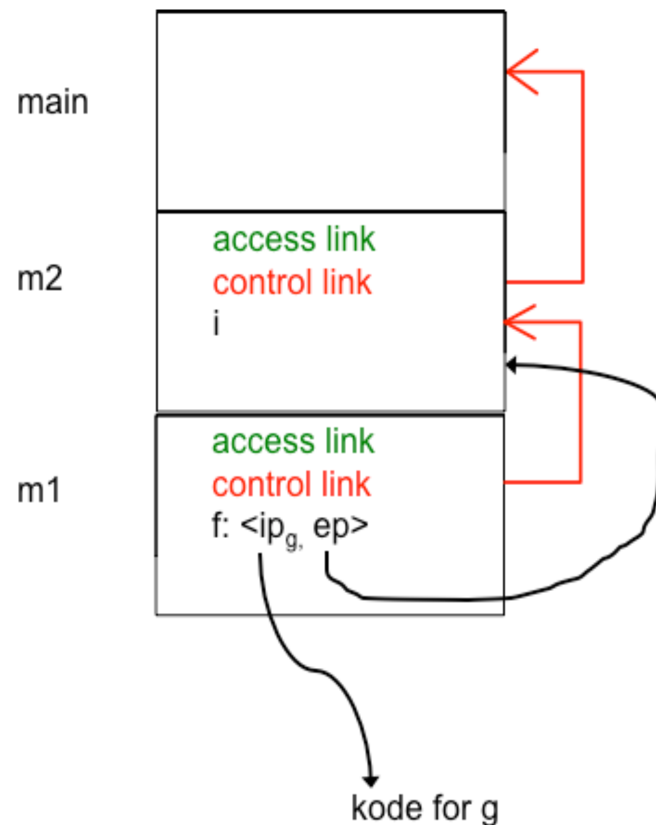
```
{
  class C {
    void m1(void f()){
      f();
    }
    void m2() {
      int i;
      void g() {
        int j; j=i;
      };
      i=1; rC.m1(g);
    };

  };
  C rC;
   void main() {
     rC = new C (); rC.m2();
   }
}
```

INF 5110 - 2013

2012 – 2b

Tegn kall-stakken som den ser ut når aktiveringsblokken for kallet rC.m1(g) er på toppen av stakken. Hvordan representeres metoden g i denne aktiveringsblokken slik at kallet på parameteren f (dvs kallet f() i m1) kan utføres?



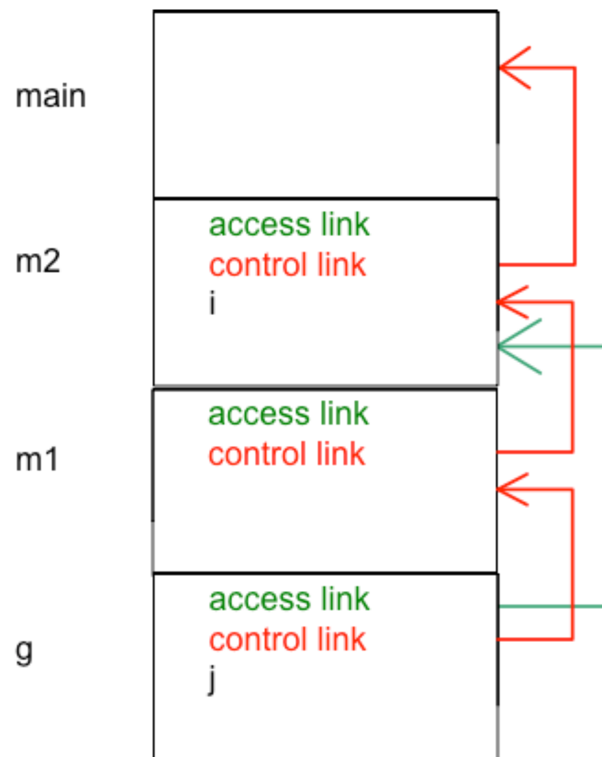
Parameteren f er representert ved et par <ip, ep>,
hvor ip peker ut koden for den aktuelle parameteren og
hvor ep peker ut aktiveringsblokken for den blokken som tekstlig omslutter definisjonen av den aktuelle parameteren.

I det aktuelle kall peker ip ut koden for g og ep peker ut aktiveringsblokken for m2, hvor g er definert.

2012 – 2c

Tegn kall-stakken som den ser ut når aktiveringsblokken for kallet av f i m1 er på toppen av stakken.

Forklar hvordan access-link for aktiveringsblokken på toppen av stakken settes i dette spesielle tilfellet.



Access link for g settes vha ep-delen av den closure som f er representert ved, og denne peker på aktiveringsblokken for m2

INF 5110 - 2013

2012 – 3a

Det følgende er en del av en grammatikk for et språk med klasser. Det er bare tatt med de produksjoner som har betydning. Klasser har for eksempel også variable, men de er ikke viktige her.

```
class → class name { methodDecls }
methodDecls → methodDecls ; methodDecl
methodDecls → methodDecl

methodDecl →
  type? name ( parameters ) body

parameters → parameters , parameter
parameters → parameter
parameter → type name

type → int
type → bool
```

INF 5110 - 2013

2012 – 3a

Den regel som skal spesifiseres ved hjelp av en attributtgrammatikk er at en klasse kan ha flere enn én constructor, men de må ha forskjellige signaturer i form av antall og/ eller typer av parametere.

Lag attributtgrammatikken basert på ideen om at methodDecl har et attributt constructorName som er satt sammen av navnet på metoden og strenger som tilsvarer typene på parameterne. En constructor C med parametertypene (int, int) vi således få constructorName 'C_i_i', mens en constructor C med parametertyper (int, bool) vil få constructorName 'C_i_b'. Testen vil derfor være, som antydet i første rekke i tabellen under, at alle navne i mengden av constructornavne er forskjellige. Vi innrømmer at dette ikke er det mest optimale, men det er ikke poenget her.

Definer de regler som gjør denne testen mulig. Anta at du har funksjoner og operatorer for å innsette navne i en menge og for å konkatenere strenge og tegn.

2012 – 3a

Grammar Rule	Semantic Rule
class → **class** ***name*** **{** *methodDecls* **}**	*class*.OK = all names in *methodDecls*.SetOfConstructornames are different
$methodDecls_1$ → $methodDecls_2$ **;** *methodDecl*	
methodDecls → *methodDecl*	
methodDecl → *type* ***name*** **(** *parameters* **)** *body*	
$parameters_1$ → $parameters_2$ **,** *parameter*	
parameters → *parameter*	
parameter → *type* ***name***	
type → **int**	*type*.typeString = 'i'
type → **bool**	*type*.typeString = 'b'

2012 – 3a.1

Grammar Rule	Semantic Rule
class → **class** ***name*** **{** *methodDecls* **}**	*class*.OK = all names in *methodDecls*.SetOfConstructornames are different *methodDecls*.className= ***name***.name
$methodDecls_1$ → $methodDecls_2$; *methodDecl*	$methodDecls_1$.constructorNameSet = $methodDecls_2$.constructorNameSet + *methodDecl*.constructorNameSet $methodDecls_2$.className= $methodDecls_1$.className *methodDecls*.className= $methodDecls_1$.className
methodDecls → *methodDecl*	*methodDecls*.constructorNameSet = *methodDecl*.constructorName *methodDecl*.className= *methodDecls*.className

INF 5110 - 2013

2012 – 3a.2

Production	Semantic rule
methodDecl → *type* ***name*** **(** *parameters* **)** *body*	*if methodDecl*.className = ***name***.name *then* *methodDecl*.constructorName = ***name***.name + '_' + *parameters*.paramTypes
$parameters_1$ → $parameters_2$ **,** *parameter*	$parameters_1$.paramTypes = $parameters_2$.paramTypes + '_' + *parameter*.typeName
parameters → *parameter*	*parameters*.paramTypes = *parameter*.typeName
parameter → *type* ***name***	*parameter*.typeName = *type*.typeString
type → **int**	*type*.typeString = 'i'
type → **bool**	*type*.typeString = 'b'

2011 - 2

Anta at vi har et objekt-orientert språk, hvor en virtuell metode i en klasse kan redefineres (”overriding”) i en subklasse av denne klassen. En virtuell metode deklareres med en virtual modifier, mens en redefinisjon deklareres med modifieren redef. Metoder uten virtual er vanlige metoder og kan altså ikke redefineres. Merk at dette ikke er helt som i Java. I Java er alle metoder virtuelle, mens her gjelder det bare de som har modifieren virtual.

```
class A {
  virtual void m(int x,y){...}
  void p(){...}
  virtual void q(){...}
}
class B extends A{
  redef void m(int x,y){...}
  void r(){...}
}
class C extends A{
  redef void q(){...}
}
```

```
class D extends B{
  redef void m(int x,y){...}
}
class E extends B{
  redef void q(){...}

class F extends C{
  redef void m(int x,y){...}
```

2011 - 2a

Vi antar nå først at klassen for et gitt objekt bestemmer, på vanlig måte, hvilken versjon av en virtuell metode som kalles.

Lag virtuell-tabellene for klassene A, B, C, D,E og F. For hvert element i tabellene skal du bruke notasjonen A::m for å angi hvilken metode som gjelder. Indeksen i disse tabeller starter på 1.

	A
1	A::m
2	A::q

	B
1	B::m
2	A::q

	C
1	A::m
2	C::q

	D
1	D::m
2	A::q

	E
1	B::m
2	E::q

	F
1	F::m
2	C::q

2011 - 2b

I resten av oppgaven skal vi for virtuelle metoder ha den semantikk at en redefinert virtuell metode, for eksempel m, skal, som det første den gjør, kalle den tilsvarende virtuelle eller redefinerte metode (dvs m) i den nærmeste superklasse som har en slik, før den utfører sin egen body. Dette vil i sin tur føre til at redefinerte eller virtuelle metoder m i videre superklasser utføres.

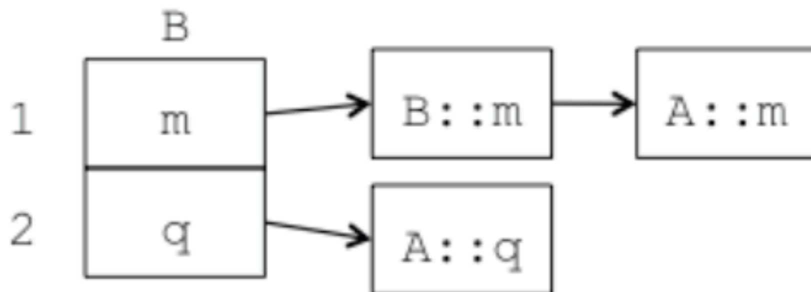
Dette kunne man implementere ved å sette inn det riktige kall som første statement i bodies på redefinerte metoder. Men semantikken rundt parameteroverføring skal her være litt spesiell slik at denne enkle måten ikke vil fungere. Parametrene som gis med i det opprinnelige kallet skal nemlig gå direkte som parametre til den metoden som utføres først, altså den som er merket virtual i programmet. Når denne er ferdig utført skal de verdiene som da står i dens parametervariable overføres som aktuelle parametre til den neste dypere redefinerte metode, osv. Dette gjør at stakken av kall må settes opp først, og de aktuelle parametre må gis til den første virtuelle metode som skal utføres.

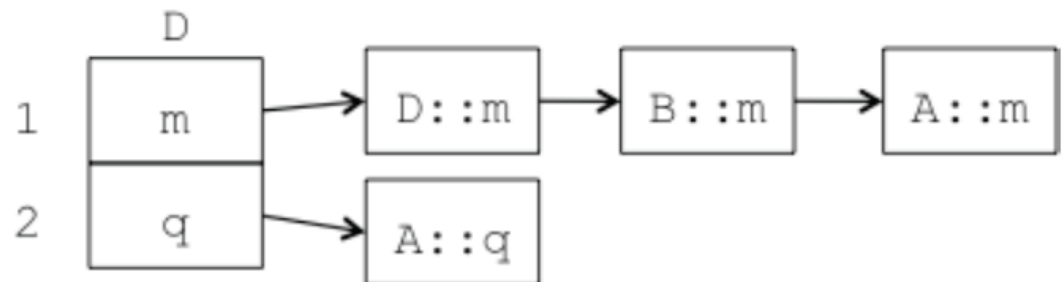
Hvis for eksempel m kalles med m(1,2) på et D-objekt, så skal stakken settes opp og de aktuelle parametre gis til aktiveringsblokken tilsvarende A::m, og utførelsen skal starte med utførelsen av A::m. Ved exit av A::m skal verdiene av parametrene x og y gis som aktuelle til den versjon av m som da skal utføres.

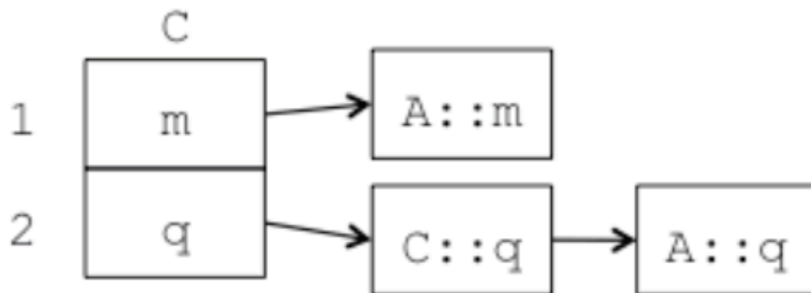
2011 - 2b

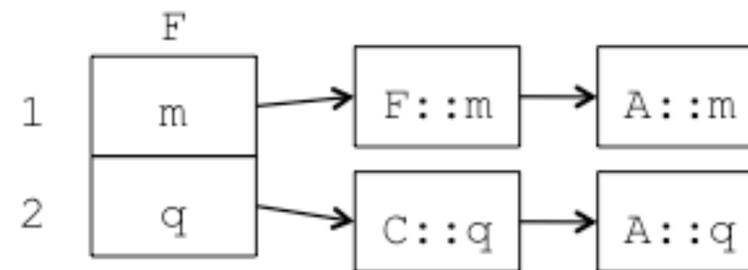
For å implementere denne nye semantikk utvides virtuell-tabellen, slik at det for hvert indeks blir en liste av metode-angivelser. Denne listen vil dermed angi sekvensen av de metoder som skal kalles.

Tegn de nye virtuell-tabeller for klassene D og F. Tabellene for B og C er gitt under. For metode-angivelser brukes samme notasjon som før.









2011 - 2c

I denne del av oppgaven skal du skissere hvordan stakken i **2b** kan lages ved hjelp av de nye virtuell-tabeller. Du kan anta at du har en run-time rutine makeActivationRecord(metode). Denne tar som parameter en metode-angivelse (for eksempel A::m) med nok informasjon til å lage en aktiveringsblokk med riktig størrelse, men du skal skissere hvordan control-link og retur-adresse settes i aktiveringsblokken.
Anta at den aktuelle virtuell-tabell holdes i en variabel med navn vt, den aktuelle metoden holdes i en variabel method, og funksjonen index(method) gir deg index i vt. Anta videre at inngangen i tabellen gir en peker til første metode-angivelse, og at hver av disse har en next peker. For metode-angivelsen som svarer til den virtuelle metode hvor den defineres for første gang (i eksemplet her m i A) er denne peker none.

Du kan gjerne illustrere resultatet for et kall m(1,2) på et D-objekt, men om resten er riktig er det OK uten.

2011 - 2c

```
caller = fp;
method = vt(index(method));
while method =/= none do {
  ar = makeActivationRecord(method);
  ar.conrolLink = caller;
  ar.returAdresse = første statement i metoden som tilsvarer fp
  method = method.next;
  caller = ar;
}
```

2011 - 2d

Overføring av parametre mellom de enkelte utførelser av en virtuell metode kan åpenbart ikke gjøres som en del av det å sette opp stakken i **2c**, men må gjøres ved bl.a. å sette inn ekstra kode i metoder merket virtual eller redef. Hvilken kode skal settes inn og hvor? Koden kan gjøre bruk av alle deler av den aktuelle aktiveringsblokk.

Utfør følgende kode, som det siste før exit, i alle aktiveringer bortsett fra den dybeste:

```
controlLink.x = x
controlLink.y = y
```