Detachable Classes

Stein tar opp denne problemstillingen: (han sier)

Ellers så jeg litt nærmere på det vi snakket om med detach. Jeg tror ikke jeg fikk fram ordentlig det eksempelet jeg hadde tenkt på, for når jeg nå ser på det igjen oppstår den samme dumme situasjonen jeg forsøkte å få fram sist.

Jeg tenkte altså på noe sånt (og jeg bruker noen kjente Simula-navn):

class Link; begin .... end;

Link class Process; begin ... end;

Link class Storage; begin ... end;

Her er det altså meningen at Process-objekter skal gå som korutiner (gjerne i en simuleringssammenheng), men det samme skal IKKE gjelde for Link-objekter generelt. Spesielt skal Storage-objekter ikke gå som korutiner.

Så kan det hende at man en gang sitter med en ref(Link)-peker rL til et Process-objekt man ønsker å detache, og man gjør dette ved å si rL.detach; Dette er jo helt greit i Simula, men så vidt jeg kan se vil det i Java-versjonen føre til at man må starte alle objekter av subkklasser av Link i en egen tråd.

Stein har helt rett, men spørsmålet er

Gjør det noe ?

Jeg vil gå gjennom i detalj på de neste sidene.

Tar utgangspunkt i dette Simula Programmet:

```
begin
  ref(Link) x;
  ref(Process) y;
  ref(Storage) z;

  class Link;          begin     ! Ingen detach her ;  end;

  Link class Process;  begin     detach;               end;

  Link class Storage;  begin     ! Ingen detach her ;  end;

  x:-new Process();
  x.detach;

  z:-new Storage();

end;
```

Og Java-koden til hovedblokka og klassen Link blir slik:
– og det er helt greit:

```
public class adHoc00 extends RTObject$ {
   public boolean isQPSystemBlock() { return(true); }
   adHoc00$Link x=null;
   adHoc00$Process y=null;
   adHoc00$Storage z=null;
   // Constructor
   public adHoc00(RTObject$ staticLink) {
      super(staticLink);
   }
   // Statements
   public RTObject$ STM() {
      BPRG("adHoc00");
      x=new adHoc00$Process(((adHoc00)PRG$)).START();
      x.detach();
      z=new adHoc00$Storage(((adHoc00)PRG$)).START();
      EBLK();
      return(null);
}  }

public class adHoc00$Link extends CLASS$ {
   public boolean isDetachUsed() { return(true); }
   // Constructor
   public adHoc00$Link(RTObject$ staticLink) {
      super(staticLink);
      // Create Class Body
      CODE$=new ClassBody(CODE$,this) {
         public void STM() {
            BBLK(); // Iff no prefix
            if(inner!=null) inner.STM();
            EBLK(); // Iff no prefix
      }};
   }
   // Class Statements
   public adHoc00$Link STM() { return((adHoc00$Link)CODE$.EXEC$()); }
   public adHoc00$Link START() { START(this); return(this); }
}
```

Det ser ikke ut til at vi kan avgjøre at detach blir utført på objekter av en gitt klasse. Men vi kan avgjøre at detach ikke blir utført på visse objekter.

Det kunne da være på sin plass å endre navn på member funksjonen

IsDetachUsed til ***isDetachable***.

Vi får altså en unødvendig kostnad ved genereringen av visse objekter ved at koden startes i en egen tråd. Denne tråden er riktignok kortlivet men overheaden er der. Dette kan vi leve med siden dagens hardware er så mye raskere enn på 70'tallet.

En bedre løsning får vi kanskje når prosjekt Loom konkluderer. Jeg håper fortsatt at Java VM får 'detachable classes' altså Java-klasser med egen stack.

Og så; jeg legger merke til at Stein bruker klassene Link og Process. De er jo Standard-klasser og merket i Link om den er ***detachable*** eller ikke er jo ikke satt. Jeg prøver derfor følgende Simula Program:

```
Simulation begin
  ref(Link) x;
  ref(Storage) z;

  Link class Storage;
  begin ! Ingen detach her ;  end;

  x:-new Process();
  x.detach;

  z:-new Storage();
end;
```

Dette gir samme resultat; nemlig at Storage Objektet startes i en egen tråd.

Årsaken er at ***detachable*** flagget settes i den lokale kopien av deskriptoren til Standard klassene. (Slik vil det også bli for separat kompilerte klasser).