Programmeringsspråket

Simula

Hva — hvorfor — hvordan

Stein Gjessing og *Dag Langmyhr*

Programmeringsspråket *Simula*

Simula er et programmeringsspråk oppfunnet av *Ole-Johan Dahl* og *Kristen Nygaard* på 1960-tallet. Det var dette språket som introduserte **objektorientert programmering**.

- Hvorfor ble dette programmeringsspråket laget?
- Hvordan var det å programmere i?
- Hvilke nyvinninger kom ut av prosjektet?
- Hvordan spredde disse ideene seg?
- Hva er språkets status i dag?



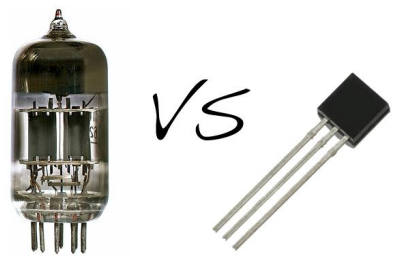
Status i 1962

La oss rekapitulere fra foreles-ningen om datahistorie i IN1020.

UiO : 2. generasjon: Transistorer

År	Teknologi	Størrelse	Instr/sek	Pris (2019kr)
1945-60	Radiorør	10m^3	2000	60 mill
1960-68	Transistorer	500dm^3	500 000	50 mill





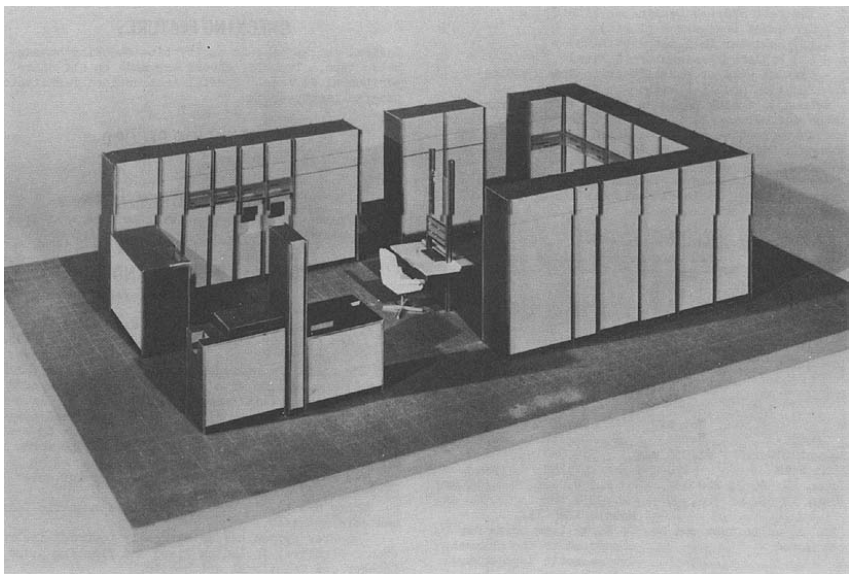
Maskinene ble mindre, raskere og mye mer pålitelige

15 5

Situasjonen i Norge i 1962

Det fantes færre enn 20 datamaskiner i Norge. Universitetet i Oslo hadde tre. Den største (ved Meteorologisk institutt) kostet 75 millioner kroner (i dagens penger).

Industrien og statlige institusjoner så nytten av datamaskiner. *Norsk regnesentral* (NR) var opprettet for å tilby bruk av datamaskin og hjelp til å utvikle programmer. De hadde en Univac 1107 verdt 300 mill kr (i dagens penger).



Programmering i 1962

Det fantes ikke så mange programmeringsspråk:

COBOL («COmmom Business-Oriented Language») var for regnskap og personalsystemer.

FORTRAN («FORmula TRANslation») gjorde det enkelt å skrive matematiske formler.

LISP («LISt Processing») var bygget rundt håndtering av lister.

ALGOL («ALGOrithmic Language») var laget for å skrive algoritmer.

Typisk var at disse språkene ikke hadde andre datastrukturer enn arrayer (lister i LISP).

Paradigmer

Det finnes veldig mange programmeringsspråk, og de kan ha svært ulike egenskaper.

Alle problemer kan (i prinsippet) programmeres i omtrent alle språk, men ofte var det uforholdsmessig mye arbeid. Derfor er det vanlig å dele dem inn i **paradigmer** utifra hva de har spesielt god støtte for.

Noen vanlige programmeringsparadigmer er:

Paradigme	Asm	C	Java	Python	Simula
Imperativt	✔	✔	✔	✔	✔
Objekt-orientert	✖	✖	✔	✔	✔
Parallellitet (tråder)	(✔)	(✔)	✔	(✔)	✖
Hendelsesdrevet (GUI)	(✔)	(✔)	✔	(✔)	✖
Diskret simulering	✖	✖	✖	(✔)	✔

(✔) angir at det ikke er innebygget støtte, men den finnes i et standardbibliotek.

NB! Denne oversikten gjelder språkene i sin opprinnelige form. Mange har siden kommet med større revisjoner for å støtte oo-programmering og annet.

Simulering

Simulering er å utføre et program som implementerer en **modell** av noe, oftest av noe i den virkelige verden. Det brukes når det er for farlig, for vanskelig eller for dyrt å gjøre det i virkeligheten.

Eksempler

- flysimulator
- kjernefysisk eksplosjon
- hvordan en smittsom sykdom sprer seg
- ventetid og reisetid om man endrer en bussrute
- klimaet

NB! «Diskret» betyr «stegvis, ikke kontinuerlig» mens «diskré» betyr å kunne holde på hemmeligheter.

Diskret simulering

Undertypen **diskret simulering** brukes når man har distinkte aktører som gjør ting på visse tidspunkter.

Eksempel: Sikkerhetskontrollen på Gardermoen

Passasjerer ankommer på tilfeldige tider.
Sperrer sjekker ombordstigningskortet.
Køer holder orden på passasjerene.
Vektere veileder passasjerene og sjekker noen.
Skannerne skanner bagasjen og passasjerene.

Typisk spørsmål

Hva blir ventetiden om man åpner/stenger en skanner?

NB! «Diskret» betyr «stegvis, ikke kontinuerlig» mens «diskré» betyr å kunne holde på hemmeligheter.

Hvordan programmere diskret simulering

I diskret simulering i Simula oppretter man **prosesser** som er dataobjekter med handling (og noe annet enn OS-prosesser).

- Objektene «later som» de er aktører i handlingen: de venter til de skal gjøre noe, de endrer sine data og de influerer på andre prosessers handlinger.
- Noe vil gjerne variere, for eksempel tidspunkter for når noe skjer, og da trekker man dette tilfeldig i henhold til gitte statistiske fordelinger.
- Underveis samler objektene data om ventetider og annet for å lage statistikk.
- Ved å kjøre simuleringen mange ganger får man sannsynligvis et ganske godt estimat på hva som virkelig ville skje.

Et eksempel: Heisene i tårnet i OJDs hus

- Det er tre heiser, og de går fra 0. etasje (garasjen) til 10. etasje.
- Passasjerene kommer til ulike tider, men
  - Vi vet i gjennomsnitt hvor mange som kommer hver time.
  - Det kommer ekstra mange i morgentimene, og de skal oftest reise opp i tårnet.
  - Det er også mange flere reisende i ettermiddagsrushet, og de skal typisk reise ned.

Spørsmål

Hva er vanlig ventetid?
Hva blir ventetiden om man stenger én av heisene?

Datastruktur

Vår modell bruker disse klassene:

class Lift definerer heisene. De er Simula-**prosesser** som flytter seg mellom etasjene og henter og avleverer passasjerer.

class Passenger beskriver passasjerene. De er også prosesser:

1. De ankommer heisen.
2. De trykker på opp- eller nedknappen og venter.
3. Når heisen kommer i riktig retning, går passasjerene inn og trykker på knappen for rett etasje.
4. Når heisen er kommet til riktig etasje, går de ut.

class Floor brukes for etasjene. De er passive objekter.

Heisene i Ole-Johan Dahls hus

Simula

```
Process CLASS Lift;
BEGIN
   REF(Head) passengers;
   BOOLEAN ARRAY
         selected_floors (0:10);
   INTEGER floor;
END;

Process CLASS Passenger (
      start_floor, end_floor);
INTEGER start_floor, end_floor;
BEGIN

   REF(Lift) selected_lift;
END;

CLASS Floor;
BEGIN
   REF(Head) up_queue, down_queue;
END;
```

«Kvasi-Java»

```
class Lift extends Process {

   ArrayList<Passenger> passengers;
   boolean[] selected_floors =
         new boolean[11];
   int floor;
}

class Passenger extends Process {
   int start_floor, end_floor;
   Passenger (int f1, int f2) {
      start_floor = f1;  end_floor = f2;
   }
   Lift selected_lift;
}

class Floor {
   ArrayList<Passenger> up_queue,
         down_queue;
}
```

Simuleringsverktøy

Disse Simula-metodene og -setningene brukes i dette eksemplet:

ACTIVATE *p* AT *t* starter prosess *p* i tidspunkt *t*.

ACTIVATE *p* AFTER Current starter prosess *p* når den nåværende passiveres.

Hold(*t*) passiverer prosessen *t* tidsenheter.

Passivate passiverer denne prosessen inntil videre.

Time angir simuleringstidspunktet.

Wait(*q*) legger denne prosessen inn i køen *q* og passiverer den.

Heisene i Ole-Johan Dahls hus

Simula

```
Process CLASS Lift;
BEGIN
   REF(Head) passengers;
   BOOLEAN ARRAY
         selected_floors (0:10);
   INTEGER floor;

   WHILE TRUE DO BEGIN
      IF nothing_to_do THEN Passivate;
      let_passengers_enter;
      Hold(lift_start_time);
      move_to_next_floor;
      WHILE NOT stop_on_this_floor DO
      BEGIN
         Hold(lift_pass_time);
         move_to_next_floor;
      END WHILE;
      Hold(lift_stop_time);
      let_passengers_leave;
      selected_floors(floor) := FALSE;
   END WHILE;
END;
```

«Kvasi-Java»

```
class Lift extends Process {

   ArrayList<Passenger> passengers;
   boolean[] selected_floors =
         new boolean[11];
   int floor;

   while (true) {
      if (nothing_to_do()) passivate();
      let_passengers_enter();
      hold(lift_start_time);
      move_to_next_floor();
      while (!stop_on_this_floor()) {

         hold(lift_pass_time);
         move_to_next_floor();
      }
      hold(lift_stop_time);
      let_passengers_leave();
      selected_floors[floor] = false;
   }
}
```

Heisene i Ole-Johan Dahls hus

Simula

```
Process CLASS Passenger (
        start_floor, end_floor);
INTEGER start_floor, end_floor;
BEGIN

  REF(Lift) selected_lift;

  ACTIVATE find_closest_lift
           AFTER Current;
  IF start_floor < end_floor THEN
    Wait(floors(start_floor).up_queue)
      ELSE
    Wait(floors(start_floor).down_queue);

    selected_lift :- find_a_lift;
    selected_lift.
        selected_floors(end_floor) := TRUE;
    Wait(selected_lift.passengers);
END;
```

«Kvasi-Java»

```
class Passenger extends Process {
  int start_floor, end_floor;
  Passenger (int f1, int f2) {
    start_floor = f1;  end_floor = f2;
  }
  Lift selected_lift;

  activate( find_closest_lift(),
        AFTER, current());
  if (start_floor < end_floor)
    wait(floors[start_floor].up_queue)
  else
    wait(floors[start_floor].down_queue);

  selected_lift = find_a_lift();
  selected_lift.
    selected_floors[end_floor] = true;
  wait(selected_lift.passengers);
}
```

Loggfil

En loggfil er lett å lage og gir nyttig informasjon under uttestingen av programmet.

```
0:00:00: Lift 1 is passivated on floor 2.
0:00:00: Lift 2 is passivated on floor 2.
0:00:00: Lift 3 is passivated on floor 2.
8:00:00: Simulation started.
8:00:10: Passenger #1 arrives on floor 2 and wants to go up to floor 3.
8:00:10: Passenger #1 presses the up button on floor 2.
8:00:10: Lift 1 is activated.
8:00:10: Lift 1 has selected to go up.
8:00:10: Passenger #1 enters lift 1 and presses button 3.
8:00:14: Lift 1 starts from floor 2 going up; #1 inside.
8:00:14: Passenger #2 arrives on floor 2 and wants to go up to floor 3.
8:00:14: Passenger #2 presses the up button on floor 2.
8:00:14: Lift 2 is activated.
8:00:14: Lift 2 has selected to go up.
8:00:14: Passenger #2 enters lift 2 and presses button 3.
8:00:17: Lift 2 starts from floor 2 going up; #2 inside.
```

Resultatene

100 kjøringer for ulike antall heiser ga dette resultatet for reisetid med heisene:

Antall	Gjennomsnitt	Max
1	124 s	1130 s
2	42 s	170 s
3	34 s	127 s
4	31 s	118 s
6	27 s	105 s
10	24 s	87 s

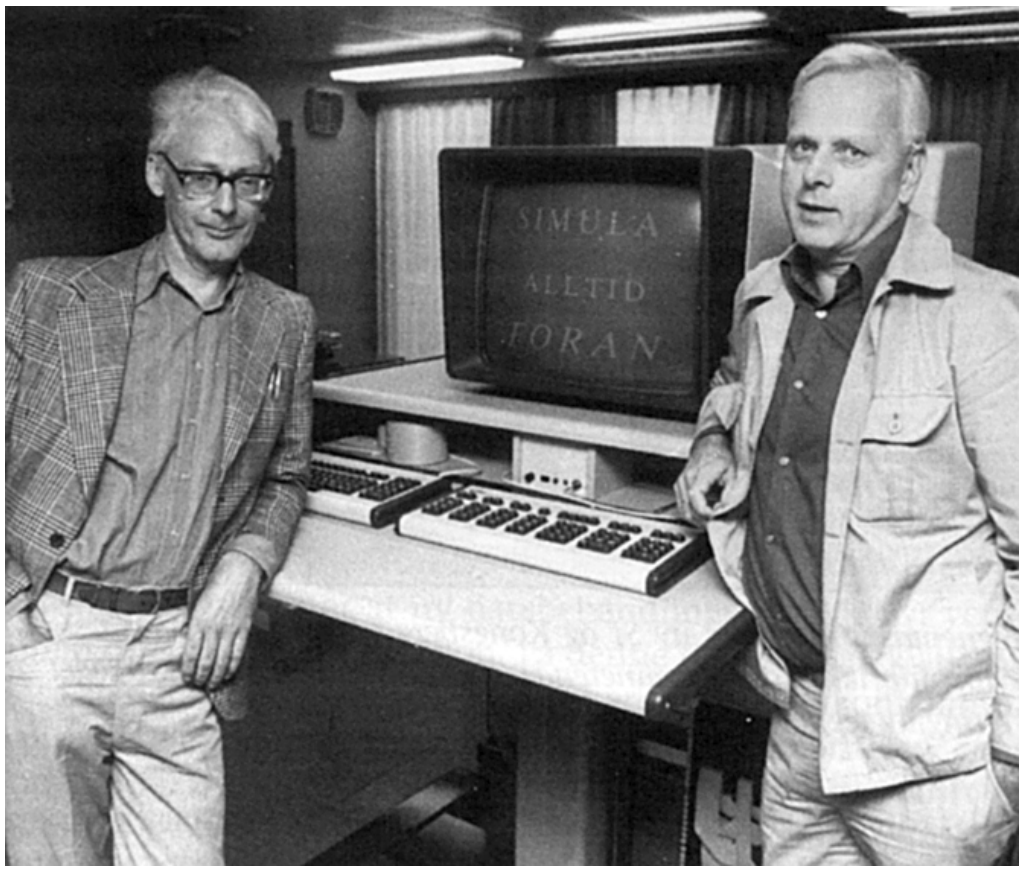
Kunne vi ikke brukt tråder?

Denne diskrete simuleringen minner mye om trådprogrammering. Kunne vi ikke brukt tråder i stedet?

- På 1960-tallet fantes det ennå ikke tråder!
- Det er mye enklere å programmere med Simulas måte. Da er bare én prosess aktiv av gangen, og vi styrer hvilken. Det skjer ingenting overraskende.
- I Simula kan man lettere finne feil i koden fordi man kan gjenta en kjøring.
- Det er lettere å forholde seg til tiden i en Simula-simulering. (Hva er tiden i et trådprogram?)
- En kjøring i Simula går raskere, for det er tidkrevende å opprette og synkronisere tråder.

Den første løsningen

Ole-Johan Dahl traff *Kristen Nygaard* i operasjonsanalysemiljøet ved *Forsvarets forskningsinstitutt* (FFI) på 1950-tallet. I 1960 flyttet de til NR. Basert på erfaringene med diskret simulering ved FFI og NR laget de første versjon av Simula.



Simula I

De valgte å gjøre det ved å ta utgangspunkt i ALGOL og utvide det til et nytt programmeringsspråk. De la til:

- **Klasser** (kalt «activity») og **objekter** (kalt «process») med handlinger og lokale data.
- Setninger for å styre simuleringen.
- **Lister** (kalt «set»).
- Metoder for ulike former for tilfeldig trekking.

Resultatet kalte de **SIMULA** (i dag kjent som «Simula I»).

Erfaringer og nye ideer

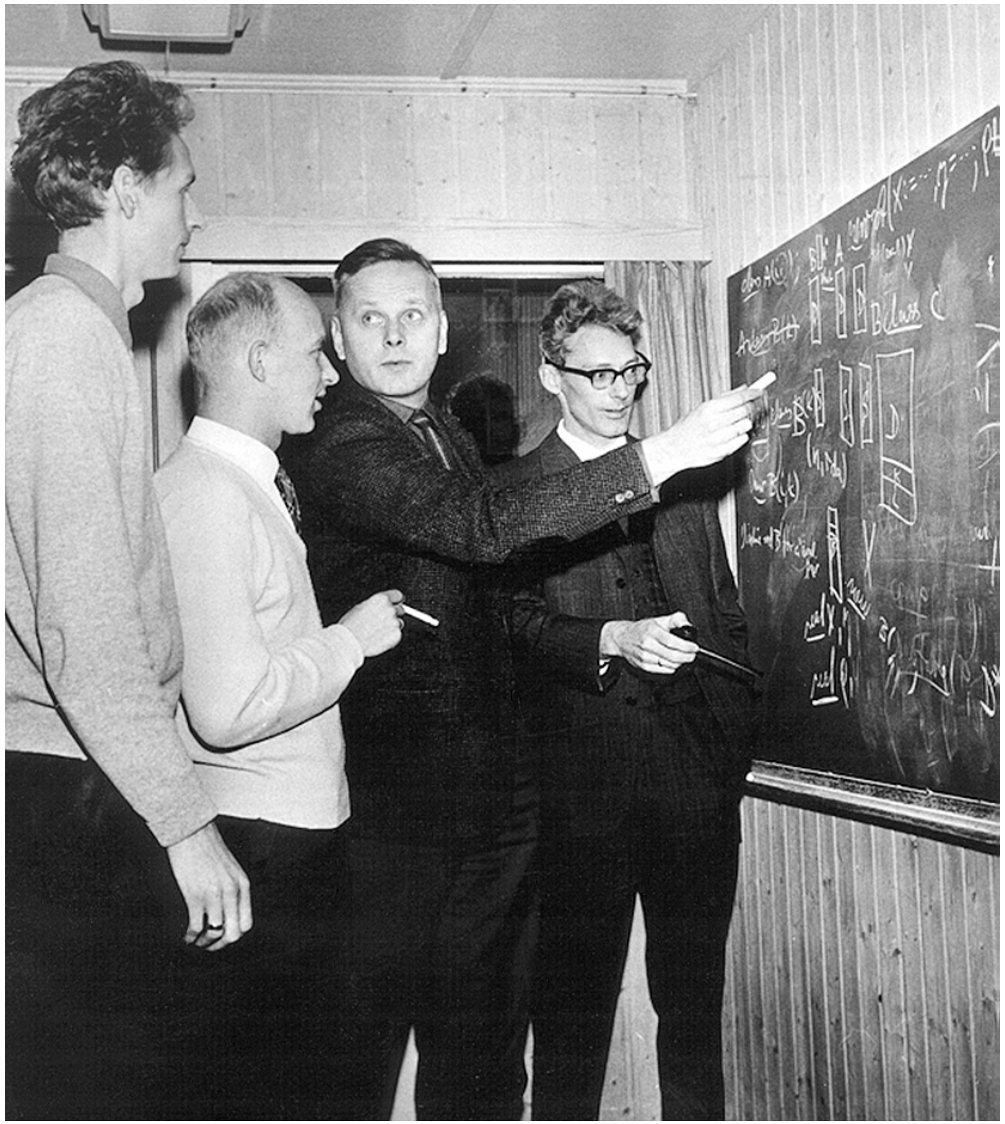
- Klasser og objekter trenger ikke være begrenset til simulering – de kan brukes som generelle begreper.
- Subklasser er en nyttig utvidelse, både til simulering og annet.
- Polymorfi er en kraftig mekanisme.

Basert på disse (og andre) erfaringene laget Ole-Johan og Kristen et nytt (og generelt) programmeringsspråk: **Simula 67** (i dag bare kalt **Simula**).

Samarbeid

Selv om Simula er Ole-Johans og Kristens verk, jobbet de ikke i et vakuum:

- Det var medarbeidere ved NR å diskutere med.
- De kommuniserte med internasjonalt kjente forskere: Edsger Dijkstra, Tony Hoare, Niklaus Wirth, . . .



Simula 67

Hva var nytt i Simula 67 i forhold til andre av datidens språk?

Klasser

Det viktigste var selvfølgelig *klasser* og *objekter*. De ga

- **innkapsling** der man kan skjule variabler og metoder
- **nye datatyper** når det er behov for det
- **modelleringsmekanisme** som samsvarer bedre med det man skal modellere

Subklasser og arv

- Det er enkelt å lage varianter av klasser.
- Man slipper ofte mye kopiering av kode.
- Det samsvarer godt med hvordan vi mennesker strukturerer mangfold.

Polymorfi

er en kraftig mekanisme:

- Den er uvurderlig for å lage nye datatyper.

Objektorintert programmering

Disse tre (klasser, subklasser og polymorfi) utgjør essensen av **objektorientering**.

Simula 67

Effekten av Simula 67 var mye større enn bare de nevnte språkmekanismene. Simula 67 ble en revolusjon for mange programmerere. Nå kunne man tenke helt anderledes om det å programmere!

Se bare på https://www.youtube.com/watch?v=yrvozM_ldAQ.

Et enkelt eksempel

En **brøk** har en *teller* og en *nevner*. Før programmerte man dette med to variabler og diverse metoder.
Nå kunne man definere en **class Brøk** som et nytt element i språket. Man kunne programmere på et høyere nivå.

Alan Kay er kjent for sitt arbeid for personlige datamaskiner. I 1966 fikk han i oppdrag å se på en Simula-kompilator:

> *I get shivers now thinking of it. It rotated my point of view through a different dimension and nothing has been the same since. I suddenly understood the purpose of higher-level languages.*

Basert på ideene fra Simula laget han sitt eget programmeringsspråk: **Smalltalk**.

Alan Kay er også mannen som fant på betegnelsen «objektorientert».

Bjarne Stroustrup lærte Simula som masterstudent i Århus.

> *The features of Simula were almost ideal for the purpose and I was particularly impressed by the way the concepts of the language helped me think about the problems in my application. The class concept allowed me to map my application concepts into the language constructs in a direct way that made my code more readable than I had seen in any other language.*

Da han senere kom til et arbeidssted som ikke hadde Simula, laget han **C++** for å kunne programmere objektorientert.

James Gosling møtte Simula ved å prøve å forstå en Simula-kompilator da han var student:

> *It was really, really hard to understand. Eventually I did. It was kind of like an explosion in my mind. It was really wonderful.*

Siden kom han til datafirmaet SUN og utviklet programmeringsspråket **Java** som var gjennomsyret av Simulas ideer.

Ut i verden

Simula ble lagt merke til, og mange tok det i bruk. I 1973 ble det anslått at Simula var i bruk på 250 datamaskiner rundt i verden.

Noen brukte det til diskret simulering:

IBM tok det i bruk i sin systemutvikling.

Intel brukte det til å utvikle sine prosessorer.

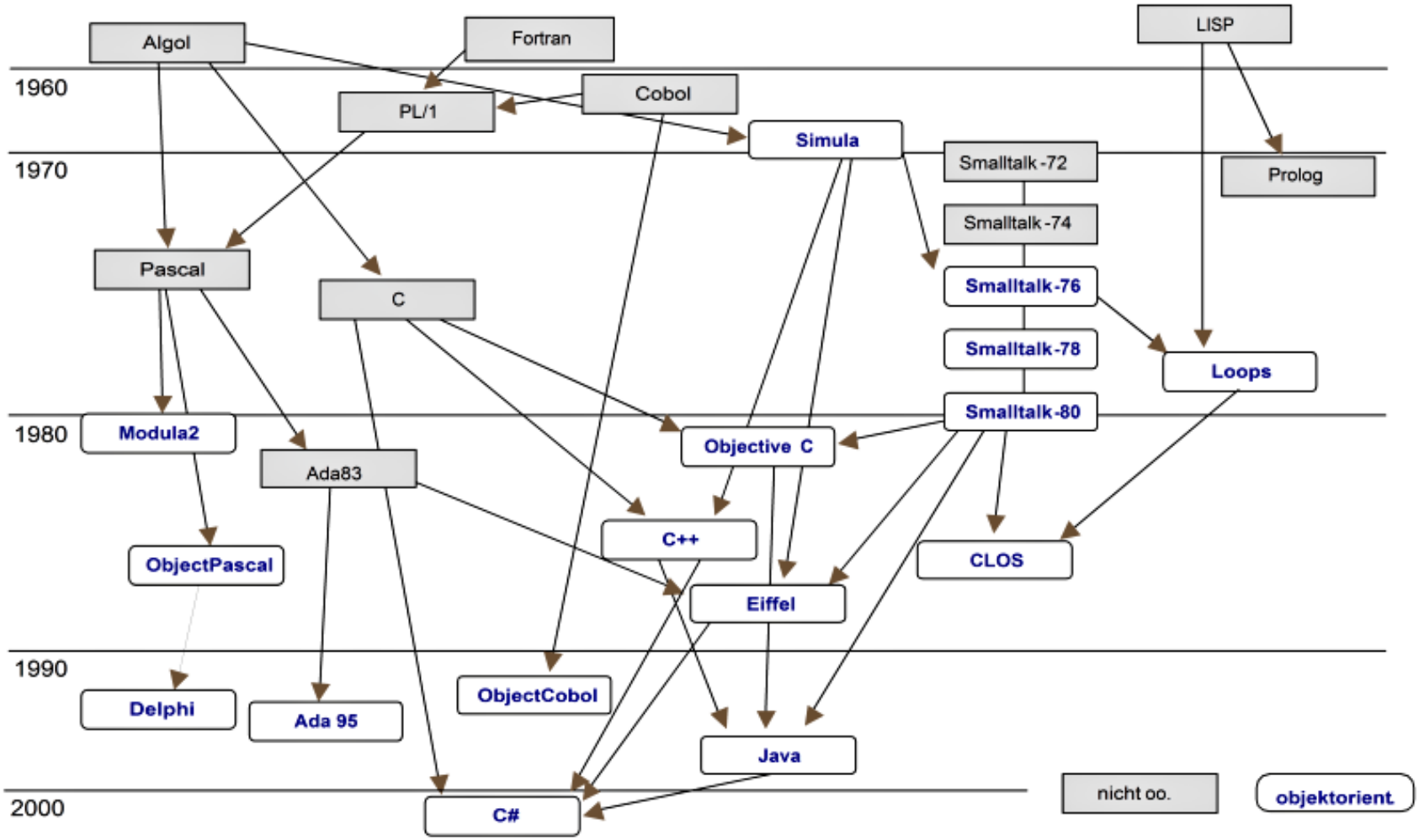
Andre ble (på tross av navnet!) interessert i Simula som et generelt programmeringsspråk.

Hva skjedde videre?

1970-årene var Simulas glansdager. Etter hvert avtok bruken:

- Det var kostbart å utvikle en Simula-kompilator for hver ny datamaskin som kom.
  NR startet arbeidet med en portabel Simula, men den kom nok for sent.
- Mange utviklere valgte å lage sin egen vri – i stedet for å implementere Simula valgte de å lage sitt eget OO-språk.

Dette slo an!



(Alle de hvite rutene er objektorienterte språk og bygger på Simula.)

Heder og ære

Association for Computing Machinery er en av de aller største foreningene for datafolk. Hvert år deler de ut **A M Turing award** som gjerne kalles «Informatikkens nobelpris».



I 2001 ble den tildelt Ole-Johan og Kristen.

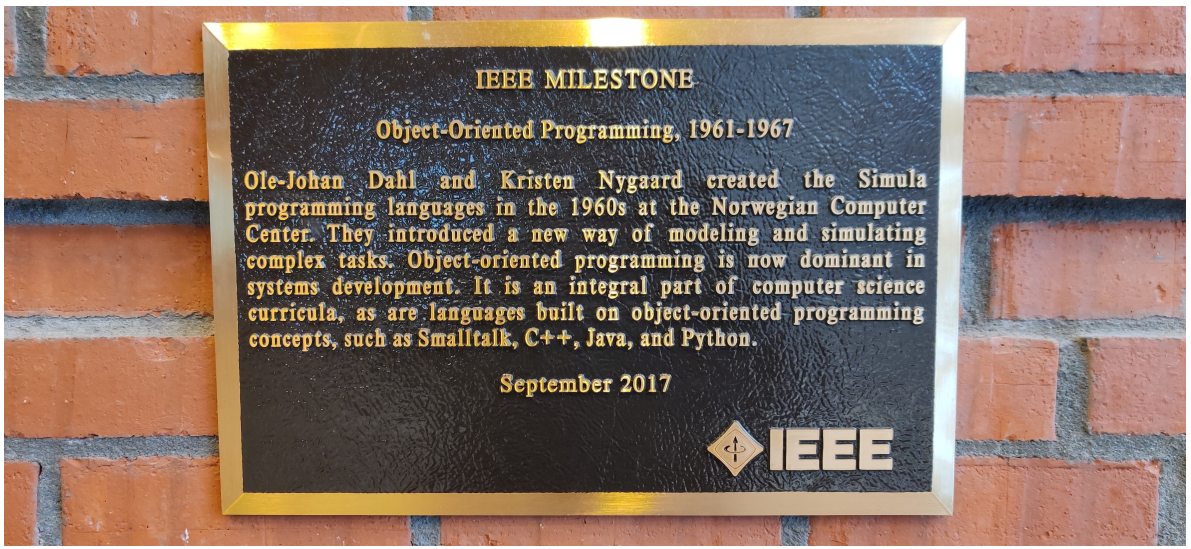
Institute of Electrical and Electronics Engineers er en annen av de største foreningene for datafolk (og andre ingeniører). De deler hvert år ut **John von Neumann-medaljen** som er omtrent like prestisjefull som ACMs Turing-pris.



I 2002 ble den tildelt Ole-Johan og Kristen.

IEEE Milestones

I tillegg setter IEEE opp plaketter som markerer de viktigste milesteinene i verdens teknologiske utvikling. I 2017 kom en slik til Ifi:





I 1998 fikk Ole-Johan og Kristen den første **Rosing-prisen**, Den norske dataforenings hederspris.

St. Olav

I 2000 ble Ole-Johan og Kristen utnevnt til *kommandør av St Olavs orden*.



Den største æren?

I 2012 fikk de to hvert sitt hus på Blindern oppkalt etter seg:



Ole-Johan Dahls hus



Kristen Nygaards hus

samt to rom i 5. etasje i dette huset og dette auditoriet.

Hvorfor ble Simula så vellykket?

Det skyldtes nok samarbeidet mellom Ole-Johan og Kristen. Et kraftig forenklet bilde er dette:

- Begge var meget dyktige fagfolk.
- Kristen var et oppkomme av ideer, mange gode men også en del ikke fullt så gode.
- Ole-Johan var programmereren som kunne vurdere hva som lot seg gjennomføre og hva som måtte forkastes.
- De to kunne være rykende uenige, men de respekterte hverandre.



Hva skjedde siden?

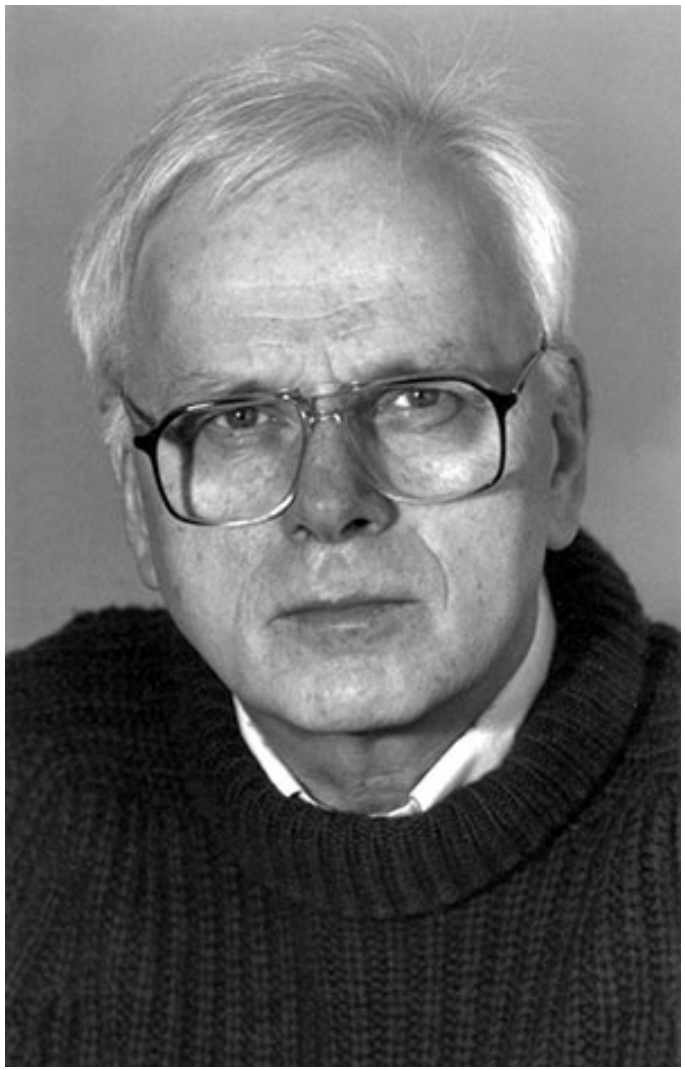
- **Ole-Johan Dahl** kom til Universitetet i Oslo i 1968.
- Han var den sentrale personen i å bygge opp Ifi.
- Han skrev bøker om oo og datastrukturer.
- Han forsket etter hvert på **formelle metoder**:

  Spesifikasjon sier matematisk hva et program skal gjøre.

  Verifikasjon sjekker program-koden mot spesifikasjonen.



- **Kristen Nygaard** laget en «etterfølger» til Simula: Beta.
- I 1984 kom han til Ifi hvor han bygget opp faggruppen *Design av informasjonssystemer*.
- Han forsket særlig på brukermedvirkning (spesielt fagforeningene) av programvare.
- Han var politisk engasjert og leder for *Nei til EU* frem til valget i 1994.



Undervisning

Simula viste seg å være et meget velegnet undervisningsspråk. Det ble brukt i Norge, Sverige, Danmark, Nederland, Tyskland, Frankrike, Storbritannia, USA, Canada, ...

Slutten

I 2000 gikk Ifi over til å satse på Java i stedet for Simula. Det var pragmatiske grunner:

- Mangel på kompilator for våre maskiner
- Få lærebøker
- Java støttet datanett, GUI og tråder
- Java var mer relevant for jobbmarkedet

Dessuten følte vi at Java tok vare på Simulas ideer.

Status i dag

Status er:

- Simula brukes ikke lenger som programmeringsspråk.

Men

- ideene fra Simula (dvs objektorientering) lever videre i de fleste vanlig brukte programmeringsspråk.

Har du lyst til å prøve Simula selv?

Det lages nå en nettside om Simula: https://www.mn.uio.no/tjenester/it/hjelp/programvare/simula/.

Der finnes link til en nyutviklet Simula-kompilator skrevet i Java.