Parameteroverføring 'by Name'.

Grunnprinsippet er å etablere et object i det kallende scopet. Dette objectet vil ha to attributt metoder; get og put for henholdsvis å lese av verdien til den aktuelle parameteren eller å skrive tilbake en ny verdi. Klassen som brukes ved slike parameteroverføringer ser slik ut:

```
abstract class ValueTypeByName_<T> {
      abstract T get();
      void put(T x) { error("Illegal ...."); }
}
```

Her benytter vi både abstrakte Java klasser og såkalt 'generics' dvs. at den aktuelle typen er en parameter. Vi antar at denne klassen er deklarert i omgivelsen til hele programmet. Merk at put-metoden har en default error slik at navne-parametre som det ikke kan skrives til slipper å redefinere denne.

Anta en Simula Procedure:

***procedure** P(k); **name** k; **integer** k; k:=k+1;*

Denne blir oversatt til denne Java metoden:

```
void P(ValueTypeByName_<Integer> k) {
      k.put(k.get() + 1); // E.g: k=k+1
}
```

På kallstedet, i praksis i selve aktuell parameter listen, lager vi et objekt av en konkret subklasse av `ValueTypeByName_<T>` ved å angi typen Integer og definerer metodene get og put. F.eks. Hvis den aktuelle parameteren er en variabel q, da vil den aktuelle parameteren bli kodet slik:

```
 new ValueTypeByName_<Integer>() {
      Integer get() { return (q); }
      void put(Integer x) { q = (int) x; }
}
```

Hvis derimot den aktuelle parameteren er et uttrykk som (j+m*n) da vil den bli kodet slik:

```
new ValueTypeByName_<Integer>() {
      Integer get() { return (j + m * n);       }
}
```

Her ser vi at put-metoden ikke er redefinert slik at et eventuellt forsøk på å tilordne en ny verdi til denne navne-parameteren vil resultere i en feilmelding.

Et klassisk eksempel på bruk av navne-parametere henter vi fra Wikipedia's artikkel om Jensen's Device. Vi skriver det om litt fordi Simula ikke tillater at kontroll-variabelen i et for-statement er en formell parameter overført by name.

```
long real procedure Sum(k, lower, upper, ak);
    value lower, upper; name k, ak;
integer k, lower, upper; long real ak;
        begin  long real s;
               s := 0.0;
               k := lower;
               while k <= upper do
               begin
                      s := s + ak;
                      k := k + 1;
               end while;
               Sum := s
        end Sum;
```

Oversatt til Java blir dette:

```
public double Sum(ValueTypeByName_<Integer> k,
              int lower, int upper,
              ValueTypeByName_<Double> ak) {
       public double s;
       s = ((double) (0.0));
       k.put(lower);
       while (k.get() <= upper) {
              s = s + ak.get();
              k.put(k.get() + 1);
       }
       return (s);
}
```

Et procedurekall som dette:

```
integer i;
long real array A[0:99];
long real result;

resultat=Sum(i,10,60,A[i]);
```

Blir d oversatt til:

```
public int control;
public double[] A=new double[100];
public double result;

result = Sum(new ValueTypeByName_<Integer>() {
       Integer get() {   return(i); }
       void put(Integer x) { i = (int)x; }
}, 10, 60, new ValueTypeByName_<Double>() {
       Double get() { return (A[i]); }
       void put(Double x) { A[i] = (double) x; }
});
```

Appendix 1: Hele Simula Koden.

```
class JensensDevice;
begin
% Jensen's device exploits call by name and side-effects.
% Call by name is an argument passing convention that delays the
% evaluation of an argument until it is actually used in the
% procedure (a consequence of the copy rule for procedures).
% Algol introduced call by name and it was kept by Simula.
% See: https://en.wikipedia.org/wiki/Jensen's_Device


long real procedure Sum(k, lower, upper, ak);
value lower, upper; name k,ak;
integer k, lower, upper; long real ak;
  begin
        long real s;
        s := 0.0;
        k := lower;
        while k <= upper do
        begin
              s := s + ak;
              k := k + 1;
        end while;
        Sum := s
  end Sum;

  integer i;
  long real array A[0:99];
  long real result;

  Sum(i,10,60,A[i]);

end JensensDevice;
```

Appendix 2: Hele den genererte Java Koden (med definisjonen av `ValueTypeByName_<T>` )

```
package examples;

public class JensensDevice {
	// ************************************************
	// *** FRAMEWORK for Name-Parameters in Java Coding
	// *** The Abstract Generic Class ValueTypeByName_
	// *** is supposed to be defined in the Environment
	// ************************************************
	abstract class ValueTypeByName_<T> {
		abstract T get();
		void put(T x) { error("Illegal ..."); }
	}

	// ************************************************
	// *** COMPILER Generated Code
	// ************************************************
	public double Sum(ValueTypeByName_<Integer> k, int lower, int upper,
			ValueTypeByName_<Double> ak) {

			double s;
			s = ((double) (0.0));
			k.put(lower);
			while (k.get() <= upper) {
				s = s + ak.get();
				k.put(k.get() + 1);
			}
			return (s);
	}

	public int i;
	public double[] A[100];
	public double result;

	// Constructor
	public JensensDevice() {

		result = Sum(new ValueTypeByName_<Integer>() {
			Integer get() {   return(i); }
			void put(Integer x) { i = (int)x; }
		}, 10, 60, new ValueTypeByName_<Double>() {
			Double get() { return (A[i]); }
			void put(Double x) { A[i] = (double) x; }
		});

	}
}
```