TÖL304G
Forritunarmál
Vikublað 9

Snorri Agnarsson

17. október 2022

Efnisyfirlit

1 Upplýsingahuld og hönnunarskjöl — Information Hiding And Design Documents

D.L. Parnas setti fyrir löngu síðan fram þá meginreglu í hugbúnaðarsmíð að sérhverja veigamikla hönnunarákvörðun skyldi fela í einingu. Til dæmis, ef nota skal forgangsbiðraðir í kerfinu skal fela í einingu hvernig forgangsbiðraðirnar eru útfærðar. Tilgangurinn með þessu er að sjálfsögðu sá að unnt verði að breyta ákvörðuninni seinna án þess að það kosti stóran uppskurð á kerfinu.

D.L. Parnas, long ago, stated the fundamental principle of software development that every important design decision should be hidden in a module. For example, if you use priority queues in the system we should hide how the priority queues are implemented. The purpose is, of course, that we will be able to change the implementation without needing major changes to the rest of the system.

Í forritun í stórum stíl (*programming in the large*) er skynsamlegt að líta á einingu frá þremur sjónarhornum, frá sjónarhorni *notenda*, *smiða* og *hönnuðar*.

When programming in the large it is advantageous to view a module from three viewpoints. From the viewpoints of *users*, *implementors* and *the designers*.

Eðlilegt er að gera ráð fyrir að fleiri en einn aðili sé notandi sömu einingar, að einingin sé notuð í fleiri en einu kerfi.

It is natural to assume that more than one party is a user of the same module, that the module is used in more than one system.

Einnig er eðlilegt að gera ráð fyrir að fleiri en einn aðili sé smiður sams konar einingar, að eining sé smíðuð oftar en einu sinni án þess að munur sé á notkun einingarinnar.

It is also natural to assume that more than one party implements the same kind of module, that the module is implemented more than once without there being a difference in how the modules are used.

Hins vegar er eðlilegt að gera ráð fyrir að aðeins einn aðili sé hönnuður einingar. Hönnun einingar felst þá í því að skrifa lýsingu einingarinnar, sem innihaldi allar þær sameiginlegu upplýsingar fyrir notendur og smiði sem nauðsynlegar eru til að nota eða smíða eininguna, en ekki meiri upplýsingar.

On the other hand it is natural to assume that only one party is the designer of a module. Desigining a module consists of writing a description of the module, which contains all the common information for users and implementors which are necessary to use or implement the module, but not more information.

Hönnunarskjal skal uppfylla eftirfarandi skilyrði — A design document should fulfil the following conditions:

- Gefa skal notanda einingar nægilega miklar upplýsingar um smíð einingarinnar til að hann geti notað hana, *en ekki meiri upplýsingar*.

  The user of a module should be given enough information about the imp-

lementation of the module that he can use it, *but not more information*.

- Gefa skal smið einingar nægilega miklar upplýsingar um notkun einingarinnar til að hann geti smíðað hana, *en ekki meiri upplýsingar*.

  The implementor of a module should be given enough information about the use of the module so that he can implement it, *en ekki meiri upplýsingar*.

2 Fastayrðingar gagna — Data Invariants

Þegar unnið er í einingaforritun er oft um einingar að ræða sem vinna með einhverja nýja gerð gagna. Þessi nýja gerð gagna byggir þá á öðrum gerðum gagna sem fyrir eru. Til dæmis getum við ímyndað okkur einingu sem nota skal til að vinna með runur heiltalna.

When we do modular programming we often have modules that work with some new sort of data. This new sort of data is then based on other pre-existing sorts of data. For example we can imagine a module that is intended to be used to work with sequences of integers.

Hugmyndin um runur heiltalna er hugmynd sem er sértekin (*abstract*) og hefur oftast ekki beina hliðstæðu í þeim gerðum gagna sem eru innbyggð í viðkomandi forritunarmál.

The idea of sequences of integers is an abstract ides and usually has no direct correspondence in the data types that are built into the programming language in question.

Við þurfum því að finna leið til að tákna runur heiltalna með þeim gerum gagna sem tiltækar eru í forritunarmálinu. integers using the types of data available in the programming language. Í Morpho, til dæmis, gætum við táknað rununa $x_1, x_2, \ldots, x_n$ af heiltölunum $x_1, x_2, \ldots, x_n$ með listanum $[x_1, x_2, \ldots, x_n]$.

We therefore need to find a way to represent sequences of integers using the types of data available in the programming language. In Morpho, for example, we might represent the sequence $x_1, x_2, \ldots, x_n$ of the integers $x_1, x_2, \ldots, x_n$ with the list $[x_1, x_2, \ldots, x_n]$.

Ef við forritum einingu til að vinna með runur heiltalna þá gætum við ákveðið að innan þeirrar einingar séu runurnar táknaðar á ofangreimdan hátt. Þær upplýsingar ættu hins vegar helst ekki að vera eðgengilegar notendum einingarinnar samkvæmt meginreglunni um upplýsingahuld, því vera má að seinna viljum við breyta því hvernig slíkar runur eru táknaðar.

If we program a module to work with sequences of integers we might decide that within this module the sequences are represented as above. That information should, however, preferably not be available to the users of the module, according to the principle of information hiding, because, perhaps, later we might want to change how such sequences are represented.

Við viljum hins vegar skjalfesta þessar upplýsingar innan einingarinnar. Við köllum þá skjölun *fastayrðingu gagna*.

We do however want to document this information within the module. We call that documentation a *data invariant* (sometimes *representation invariant*).

Dæmi um slíka fastayrðingu gagna er því:

> Runa $x_1, x_2, \ldots, x_n$ af heiltölum $x_1, x_2, \ldots, x_n$ er táknuð með listanum $[x_1, x_2, \ldots, x_n]$

An example of such a data invariant is therefore:

> A sequence $x_1, x_2, \ldots, x_n$ of integers $x_1, x_2, \ldots, x_n$ is represented with the list $[x_1, x_2, \ldots, x_n]$

2.1 Fastayrðing gagna er stöðulýsing — A Data Invariant Is A State Description

Fastayrðing gagna fyrir eitthvað tiltekið gagnamót (*data structure*) er stöðulýsing og skal vera sönn eftir smíð á hverju eintaki af viðkomandi gagnamóti og skal vera sönn fyrir og eftir sérhverja opinbera aðgerð sem beitt er á eintak af gagnamótinu.

A data invariant for a given data structure is a state description and should be true after the construction of each instance of the data structure and should be true before and after each public operation performed on an instance of the data structure.

2.2 Fastayrðingar gagna og hönnunarskjöl — Data Invariants And Design Documents

Ef forritari hefur í höndunum hönnunarskjal fyrir gagnamót sem skrifað er í samræmi við upplýsingahuld og hefur einnig í höndunum fastayrðingu gagna þá á forritarinn að hafa nægilega miklar upplýsingar til að forrita sérhverja opinbera aðgerð einingarinnar.

If a programmer possesses a design document for a data structure written in accordance with information hiding and also possesses a data invariant then the programmer should have enough information to program each public operation in the module.

3 Hlutbundin forritun í Morpho — Object Oriented Programming in Morpho

3.1 Einfaldur hlutbundinn hlaði — A Simple Object Oriented Stack

Eftirfarandi eining er dæmi um klasaskilgreiningu í Morpho.

The following module is an example of a class definition in Morpho.

```
;;; Hönnun/Design
;;;
;;;   Útflutt/Exported
;;;
;;;     Notkun: s = stack();
;;;     Fyrir:  Ekkert.
;;;     Eftir:  s er nýr tómur hlaði með pláss
;;;             fyrir ótakmarkaðan fjölda gilda
;;;             meðan minnisrými tölvunnar leyfir.
;;;
;;;     Usage:  s = stack();
;;;     Pre:    Nothing.
;;;     Post:   s is a new empty stack with space
;;;             for an unlimited number of values
;;;             while the memory capacity of the
;;;             computer allows.
;;;
;;;   Innflutt/Imported
;;;
;;;     Notkun: s.push(x);
;;;     Fyrir:  s er hlaði.
;;;     Eftir:  Búið er að setja x ofan á s.
;;;
;;;     Notkun: x = s.pop();
;;;     Fyrir:  s er hlaði, ekki tómur.
;;;     Eftir:  x er gildið sem var efst á s,
;;;             það hefur verið fjarlægt af s.
;;;
;;;     Notkun: b = s.isEmpty();
;;;     Fyrir:  s er hlaði.
;;;     Eftir:  b er true ef s er tómur, annars
;;;             false.
;;;
;;;     Usage:  s.push(x);
;;;     Pre:    s is a stack.
;;;     Post:   x has been pushed on s.
;;;
;;;     Usage:  x = s.pop();
;;;     Pre:    s is a stack, not empty.
;;;     Post:   x is the value that was at the
```

```
41 ;;;                top of s, it has now been
42 ;;;                removed from s.
43 ;;;
44 ;;;        Usage:  b = s.isEmpty();
45 ;;;        Pre:    s is a stack.
46 ;;;        Post:   b is true if s is empty,
47 ;;;                otherwise false.
48
49 "stack.mmod" =
50 {{
51 stack =
52   obj()
53   {
54     var list;
55
56     ;;; Fastayrðing gagna: Hlaði sem inniheldur gildi
57     ;;; x1,...,xN, frá toppi til botns er táknaður
58     ;;; með list = [x1,...,xN].
59
60     ;;; Data invariant: A stack that contains values
61     ;;; x1,...,xN, from top to bottom, is represented
62     ;;; with list = [x1,...,xN].
63
64     msg push(x)
65       {
66         list = x:list;
67       };
68
69     msg pop()
70       {
71         val res = head(list);
72         list = tail(list);
73         res;
74       };
75
76     msg isEmpty()
77       {
78         list==[];
79       };
80     };
81 }}
82 ;
```

Þessi forritstexti varpar ljósi á nokkur grundvallaratriði — This program text illustrates a few fundamental points:

- Einingin sem skilgreind er hér hefur eitt útflutt stef, stack. Það stef er smiður (*constructor*) fyrir hluti sem eru hlaðar, þ.e. þeir hlutir bregðast rétt við boðunum push, pop og isEmpty.

  The module defined has one exported function, stack. This function is a constructor of objects that are stacks, i.e. objects that respond appropriately to the messages push, pop og isEmpty.

- Takið eftir að breytan list er tilviksbreyta.

  Note that the variable list is an instance variable.

- Takið eftir að í aðferðum boðanna er bæði unnt að nota tilviksbreytuna og viðföngin sem send eru í boðin.

  Note that in the methods for the messages we can use both the instance variable and the parameters of the message.

- Vert er að taka fram að tilviksbreytu er aðeins unnt að nota í aðferðum sem fylgja smið þess klasa sem inniheldur tilviksbreytuna. Ekki er hægt að nota tilviksbreytuna utan frá né í hlutum sem erfa frá viðkomandi hlut.

  It is noteworthy that the instance variable can only be used in methods that are associated with the constructor of the class that contains the instance variable. It is not possible to access the instance variable from the outside nor in objects that inherit from the object in question.

3.2 Hlaði með víxlunarboði — A Stack With A Swap Message

Síðan getum við haldið áfram og búið til annan klasa sem erfir frá þessum — We can then proceed and create another class that inherits from this one:

```
1  ;;; Hönnun/Design
2  ;;;
3  ;;;   Útflutt/Exported
4  ;;;
5  ;;;      Notkun: h := stack()
6  ;;;      Eftir:  h er nýr tómur aukinn hlaði
7  ;;;
8  ;;;      Usage:  h := stack()
9  ;;;      Post:   h is a new empty augmented stack
10 ;;;
```

```
11 ;;;    Innflutt/Imported
12 ;;;
13 ;;;      Notkun: h.swap()
14 ;;;      Fyrir:  h er aukinn hlaði með a.m.k. tvö gildi
15 ;;;      Eftir:  búið er að víxla efstu tveimur gildunum
16 ;;;              á h
17 ;;;
18 ;;;      Einnig eru boðin push, pop og isEmpty
19 ;;;      innflutt, og hafa sömu lýsingar og fyrir
20 ;;;      stack.mmod.
21 ;;;
22 ;;;      Usage:  h.swap()
23 ;;;      Pre:    h is an augmented stack with at least two
24 ;;;              values
25 ;;;      Post:   The top two values on the stack have been
26 ;;;              swapped
27 ;;;
28 ;;;      Also, the messages push, pop and isEmpty
29 ;;;      are imported, and have the same descriptions as
30 ;;;      for stack.mmod.
31
32 "stack2.mmod" =
33 {{
34 stack =
35   obj() super(stack())
36   {
37     msg swap()
38     {
39       var x,y;
40       x = this.pop();
41       y = this.pop();
42       this.push(x);
43       this.push(y);
44     };
45   };
46 }}
47 *
48 "stack.mmod"
49 ;
```

Í þessu dæmi kemur fram að lykilorðið this stendur fyrir hlut þann sem viðkomandi aðferð er verið að framkvæma í. Lykilorð this má aðeins nota inni í aðferð fyrir boð.

Þá er lykilorðið super, þegar það kemur fyrir í haus klasaskilgreiningar (skilgreining smiðs), notað til að tilgreina klasa sem erft er frá. Stefkallið á eftir lykilorðini (í þessu dæmi kallið stack()) verður að skila Morpho hlut.

In this example we see that the keyword this stands for the object that the current method is executed in. Furthermore, the keyword super, when used in the header of the class definition (constructor definition) s used to denote a class that is inherited from. The function call following the keyword (in this case the call stack()) must return a Morpho object.

3.3 Hlaðaeining með innfluttum arfi — A Stack Module With Imported Inheritance

Einnig er hægt að búa til almennari fjölnota klasa sem erfir frá innfluttum klasa (innfluttum smið) — We can also make a more general generic class that inherits from an imported class (imported constructor):

```
1  ;;; Hönnun/Design
2  ;;;
3  ;;;   Útflutt/Exported
4  ;;;
5  ;;;     Notkun: s = stack()
6  ;;;     Eftir:  s er nýr tómur aukinn erfður hlaði,
7  ;;;             þar sem erfði hlaðinn kemur úr
8  ;;;             innflutta stefinu stack, sem lýst
9  ;;;             er að neðan, og aukningin felst
10 ;;;             í því að s hefur aðferðir fyrir
11 ;;;             boðin height og maxHeight, sem lýst
12 ;;;             er að neðan
13 ;;;
14 ;;;     Usage:  s = stack()
15 ;;;     Post:   s is a new empty augmented inherited
16 ;;;             stack, where the inherited stack
17 ;;;             comes from the imported function
18 ;;;             stack, described below, and the
19 ;;;             augmentation consists of s having
20 ;;;             methods for the messages height
21 ;;;             and maxHeight, described below.
22 ;;;
23 ;;;   Innflutt/Imported
24 ;;;
25 ;;;     Notkun: s = stack()
26 ;;;     Eftir:  s er erfður hlaði.  Erfði hlaðinn má
```

```
27 ;;;                ekki hafa önnur boð en push og pop
28 ;;;                sem breyta fjölda gilda á hlaðanum
29 ;;;
30 ;;;        Notkun: n = s.height()
31 ;;;        Fyrir:  s er aukinn erfður hlaði
32 ;;;        Eftir:  n er fjöldi gilda á s
33 ;;;
34 ;;;        Notkun: n = s.maxHeight()
35 ;;;        Fyrir:  s er aukinn erfður hlaði
36 ;;;        Eftir:  n er mesti fjöldi gilda sem verið hafa
37 ;;;                samtímis á s
38 ;;;
39 ;;;        Auk þess eru boðin push og pop innflutt og
40 ;;;        hafa sömu lýsingu og í erfða hlaðanum.
41 ;;;
42 ;;;        Notkun: s = stack()
43 ;;;        Eftir:  s er erfður hlaði.  Erfði hlaðinn má
44 ;;;                ekki hafa önnur boð en push og pop
45 ;;;                sem breyta fjölda gilda á hlaðanum
46 ;;;
47 ;;;        Usage:  n = s.height()
48 ;;;        Pre:    s is an augmented inherited stack
49 ;;;        Post:   n is the number of values on s
50 ;;;
51 ;;;        Usage:  n = s.maxHeight()
52 ;;;        Pre:    s is an augmented inherited stack
53 ;;;        Post:   n is the most number of values that
54 ;;;                s has contained simultaneously.
55 ;;;
56 ;;;        In addition the messages push and pop are
57 ;;;        imported and have the same descriptions as
58 ;;;        in the inherited stack.
59
60 "stackstat.mmod" =
61 {{
62 stack =
63   obj() super(stack())
64   {
65     var count=0, max=0;
66     ;;; Fastayrðing gagna:
67     ;;;   count er fjöldi gilda á hlaðanum,
68     ;;;   max er hámarksfjöldi gilda sem verið hafa
```

```
69      ;;;    samtímis á hlaðanum.
70
71      ;;; Data invariant:
72      ;;;    count is the number of values on the stack,
73      ;;;    max is the maximum number of values that s
74      ;;;    has contained simultaneously.
75
76      msg height()
77        {
78          count;
79        };
80      msg maxHeight()
81        {
82          max;
83        };
84      msg push(x)
85        {
86          count = inc(count);
87          count > max && (max = count);
88          super.push(x);
89        };
90      msg pop()
91        {
92          count = dec(count);
93          super.pop();
94        };
95    };
96  }}
97  ;
```

Hér sést að lykilorðið super er einnig notað inni í aðferðum til að tiltaka að kalla skuli á erfða aðferð fyrir tiltekið boð. Takið eftir að það er merkingarmunur á segðunum this.pop() og super.pop().

Here we see that the keyword super is also used inside methods to specify that the method used should be an inherited one for the message in question. Note that there is a semantic difference between the expressions this.pop() and super.pop().

Fyrri segðin kallar á „venjulegu“ pop aðferðina fyrir hlutinn, þ.e. þá aðferð sem er fremst í arfgengiskeðjunni. Seinni segðin kallar á þá pop aðferð sem er fyrir aftan núverandi aðferð í arfgengiskeðjunni. Þetta er alvanalegt í hlutbundinni forritun, svipað eins og í C++, Java og flestöllum öðrum hlutbundnum forritunarmálum.

The former expression calls the "usual"pop method for the object, i.e. the method that is foremost in the chain of inheritance. The second expression calls the pop met-

hod that is after the current method in the chain of inheritance. This is a common thing in object oriented programming, similar to C++, Java and most object oriented programming languages.

Vert er að benda sérstaklega á að einninguna " stackstat .mmod" má nota með hvaða hlaðaklasa sem er, sem hagar sér eins og klasarnir í "stack .mmod" og "stack2 .mmod". Til dæmis má skrifa:

It is worth pointing out that the module " stackstat .mmod" can be used wit any stack class, provided it behaves like the classes in "stack .mmod" and "stack2 .mmod". For example we can write:

"stack3 .mmod" = "stackstat .mmod" ∗ "stack.mmod" ;

og/and

"stack4 .mmod" = "stackstat .mmod" ∗ "stack2.mmod" ;

3.4 Samband boða og aðferða — The Relationship Between Messages And Methods

Í öllum hlutbundnum forritunarmálum fylgir hverjum hlut vörpun sem varpar boði í aðferð. Oftast er þetta gert þannig að hverjum klasa fylgir slík vörpun, og oft er vörpunin útfærð sem einfalt fylki þar sem vísirinn er heiltala sem stendur fyrir boðið og gildið er fallsbendir eða lokun sem stendur fyrir aðferðina. Þetta fylki er kallað á ensku *Virtual Method Table*, skammstafað VMT. Við getum kallað þetta fylki *aðferðatöflu* á íslensku.

In all object oriented programming languages each object has an associated mapping that maps messages to methods. Usually this is done by each class having such a mapping, and often the mapping is implemented as a simple array where the index is an integer that represents the message and each value in the array is a pointer to a function or closure, which stands for the method. This array is called the *Virtual Method Table*, acronym VMT.

Í Morpho er aðferðatafla fyrir hvern klasa sem búinn er til með hlutstefi, þ.e. stefi á eftirfarandi sniði — In Morpho there is a VMT for each class that is constructed by a constructor, i.e. a function definition of the following form

```
f =
  obj(...)  ...
  {
    ...
  }
```

Þegar boð er sent til hlutar er leitað að boðinu í aðferðatöflunni og ef aðferð finnst fyrir boðið er hún framkvæmd. Ef ekki þá er athugað hvort hluturinn erfir frá einhverjum öðrum hlut. Ef svo er þá er leitað að aðferðinni í erfða hlutnum, og svo koll af kolli. Ef engin aðferð finnst fyrir boðið þá er það villa. Slík villa getur að sjálfsögðu

ekki gerst í rammtöguðum forritunarmálum, en í Morpho, Fjölni, SmallTalk og öðrum keyrslutöguðum hlutbundnum forritunarmálum getur það gerst.

When a message is sent to an object the VMT is searched and if an associated method is found for the message, the method is called. If not, a check is made whether the object inherits from some other object. If so, a search is made in the VMT of the inherited object, and so on, along the chain of inheritance. If no method is found for the message an error is generated. Such an error can, of course, not happen in strongly typed programming languages, but in Morpho, SmallTalk and other object oriented programming languages with runtime typing this can happen.

Í töguðum hlutbundnum forritunarmálum svo sem Java, C++ og Object Pascal er, eins og rætt hefur verið, yfirleitt farið aðeins öðruvísi að. Þá fylgir hverjum klasa aðeins ein aðferðatafla, jafnvel þótt um arfgengi sé að ræða. Sú aðferðatafla varpar boði beint í aðferð, hvort sem aðferðin er útfærð í viðkomandi klasa eða í yfirklasa. Þetta er ekki hægt í Morpho því yfirklasi er ekki þekktur á þýðingartíma, og er reyndar ekki þekktur fyrr en hluturinn verður til.

In typed object oriented programming languages such as Java, C++ and Object Pascal the implementation is usually different. In that case each class has one and only one VMT, even in the case of inheritance. That VMT maps each message to a method, regardless of whether the method is implemented in the current class or in a superclass. This is not possible in Morpho because the superclass is not known at compile time, and is in fact not known until the object is costructed.

Í öllum tilfellum er boð því tilvísun á aðferð, þegar gefinn er hlutur eða klasi. Boð *er ekki* aðferð því mismunandi hlutir hafa mismunandi aðferðir fyrir sama boð. Það er einmitt stóri kosturinn við hlutbundna forritun að sama boð getur haft mismunandi aðferðir í mismunandi hlutum.

In all cases, therefore, a message is a key to a method, when a n object or a class is given. A message *is not* a method because different objects have different methods for the same message. It is in fact the big advantage of object oriented programming that the same message can have different methods on different objects.

3.5 Hlaði með uppnefndum boðnöfnum — A Stack With Renamed Message Names

Ef nú svo vill til að nauðsynlegt reynist að nota hlaða þar sem nöfn stefja og boða eru á íslensku þá má nota eftirfarandi:

Ef it turns out to be necessary to use a stack where the names of functions and messages are in Icelandic, we may use the following:

```
1 ;;; Hönnun/Design
2 ;;;
3 ;;;    Útflutt/Exported
4 ;;;
```

```
 5 ;;;       Notkun: h = hlaði();
 6 ;;;       Eftir:  h er nýr tómur hlaði
 7 ;;;
 8 ;;;       Usage:  h = hlaði();
 9 ;;;       Post:   h s a new empty stack
10 ;;;
11 ;;;    Innflutt/Imported
12 ;;;      Boðin setja, sækja og erTómur eru innflutt og
13 ;;;      hafa sömu virkni og samsvarandi boð push, pop
14 ;;;      og isEmpty í einingunni "stack.mmod".
15 ;;;
16 ;;;      The messages setja, sækja and erTómur are
17 ;;;      imported and have the same functionality
18 ;;;      as corresponding messages push, pop and
19 ;;;      isEmpty in the module "stack.mmod".
20
21 "hladi.mmod" =
22 {{
23 hlaði = fun stack();
24 }}
25 *
26 "stack.mmod"
27 *
28 {{
29 push = msg setja(x);
30 pop = msg sækja();
31 isEmpty = msg erTómur();
32 }}
33 ;
34
35 ;;; Prófunarforrit fyrir hlaða
36 ;;; Test program for stacks
37 "hladaprof.mexe" = aðal in
38 {{
39 aðal =
40         fun()
41         {
42                 var x = [1,2,3,4];
43                 var h = hlaði();
44                 while( x )
45                 {
46                         h.setja(head(x));
```

```
47                              x = tail(x);
48                      };
49                      while( !h.erTómur() )
50                      {
51                              writeln(h.sækja());
52                      };
53              };
54  }}
55  *
56  "hladi.mmod"
57  *
58  BASIS
59  ;
```