

Ragnar Geir Brynjólfsson

Forritun í Java



2. hluti

Útgáfa 2.1
Heimili og menntir ehf
04. 07. 2005

Forritun í Java

Útgáfa 1, Heimili og menntir ehf 2003.

Óprentuð, gefin út á netinu á pdf formi.

Útgáfa 2. Heimili og menntir ehf 2004.

1. prentun heildarútgáfa ág. og sept. 2004.

2. prentun, 1. hluti og 2. hluti, okt. 2004.

Útgáfa 2.1. Forritun í Java, II. hluti. júlí 2005.

© Ragnar Geir Brynjólfsson.

Öll réttindi áskilin. Bók þessa má ekki afrita með neinum hætti, að hluta eða í heild án skriflegs leyfis höfundar og útgefanda.

Skopmyndir eftir Randy Glasbergen eru birtar með leyfi hans.

Umbrot, frágangur og prentun:

Heimili og menntir ehf. Ragnar Geir Brynjólfsson.



Ragnar Geir Brynjólfsson

Um höfundinn:

Ragnar Geir Brynjólfsson er kerfisstjóri og forritunarkennari við Fjölbrautaskóla Suðurlands á Selfossi þar sem hann hefur starfað síðan 1989. Hann útskrifaðist með BS próf í tölvunarfræði og próf í kennsluréttindum frá Háskóla Íslands árið 1989. Hann hefur gefið út eftirtaldar kennslubækur og fjölrít:

- Windows 3.1 : Word 2.0 ritvinnsla, Access gagnagrunnur : Æfingar fyrir byrjendur. Höfundur gaf út 1994. - gormabók, 104 s. : teikningar, skjámyndir.
- Windows 3.11 : Tölvugrunnur : Æfingar fyrir byrjendur. Höfundur gaf út 1995, 1996 og 1997. - gormabindi 46 s. : skjámyndir.

Efnisyfirlit

Inngangur	6
Val forritunarmáls	6
Um lestur bókarinnar	7
Um íslenskun	8
T203-1. Grunnhugtök hlutbundinnar forritunar	
Hlutbundna hugmyndafræðin	10
Aðferðir—hegðun	19
Hjúpun	21
Stöðupróf T203SP1	27
Forritunarverkefni	29-35
T203-2. Fleiri hugtök hlutbundinnar forritunar	
Fjölbinding aðferða	36
Meira um stika	39
Staðvær gögn og gildissvið	40
Frátekna orðið this	43
Javaforrit	43
Erfðir	44
Yfirtaka aðferða	47
Klasahönnun með UML	47
Hugrænir klasar	48
Einfaldar erfðir	51
Viðmót	51
Fjölbreytni	54
Stöðupróf T203SP2	56
Forritunarverkefni	59
T203-3. Frábrigði, villur straumar og skrár	
Frábrigði	64
Throwable klasinn	69
Frávikum kastað	70
Straumar og skrár	71
Lestur frá lykllaborðinu	74
Skrár	76
Skrár með beinum aðgangi	78
Stöðupróf T203SP3	82
Forritunarverkefni	84-90
T203-4. Gagnaskipan, safnklasur og pakkur	
Stafli	92
Biðröð	93
Tengdur listi	94
Tré	95
Mengi	96
Java safnklasur	98
Útfærsla stafla og biðraðar	100
Útfærsla mengja	104
Dálitill fróðleikur um pakka	106

Stöðupróf T203SP4	109
Forritunarverkefni	111-115
T203-5. Myndræn notendaskil í Java	116
Tölvugrafík	116
Algeng skráasnið	118
Inngangur að litafræði	120
Grafísk notendaskil Java	121
Kynning á nýtlum (appletum)	123
Einfaldir nýtlar	127
Teikning í nýtlum	130
Sjálfstæð forrit	132
Takkar og atburðir	136
Valpunktar	141
Stöðupróf T203SP5	146
Forritunarverkefni	149-153
T203-6. Röðun—netsamskipti	154
Fleiri röðunaralgrím	158
Netsamskipti	160
TCP/IP og UDP/IP	160
Vefsíða lesin af neti	161
Stöðupróf T203SP6	166
Forritunarverkefni	168-173
Hugtakasafn	174-183
Heimildaskrá	184-185

Í 1. hluta bókarinnar er að finna eftirtalda kafla:

T103-1. Tölvukerfi og forritun

Hugbúnaður og vélbúnaður
Um vinnu- og þróunarumhverfi
Kynning á Linux
Textavinnsla í Linux
Textavinnsluskipanir í Joe
Stafræna tæknin
Aðferð við lausn verkefna
Java forritunarmálið
Meira um athugasemdir
Kennimerki og frátekin orð
Bil og inndráttarvenjur
Kynslóðir forritunarmála
Þýðendur og túlkar
Málskipan og merkingarfræði
Aðferðafræði við „aflúsun“ forrita
Samantekt
Stöðupróf T103SP1
Forritunarverkefni

T103-2. Frumgögn

Frumgögn og hlutir

Strengir
Lausnarrunur
ASCII og Unicode
Frumgögn—frumbreytur
Reiknivirkjar, segðir og reiknisetningar
Forgangsröð virkja
Umskráning gagna
Stöðupróf T103SP2
Forritunarverkefni

T103-3. Hlutir og klasar

Hlutbundin og óhlutbundin mál
String klasinn—hlutir búnir til
Klasasöfn og pakkar
Import skilgreiningin
Aðferðir vaktar
Kyrrelegar aðferðir
Innlestur frá lykllaborði
Stöðupróf T103SP3
Forritunarverkefni

T103-4. Búlskar segðir og skilyrðissetningar

Inningarröð
Skilyrðissetningar
Hreiðraðar if-setn. og kommutölunálganir
Samanburður kommutalna
Switch setningin
Stöðupróf T103SP4
Forritunarverkefni

T103-5. Lykkjur

Hækkunar og lækkunarvirkjar
Gildisveitingarvirkjar
Skilyrðisvirkinn
Forgangur ýmissa virkja
Lykkjur
While lykkjan
Do lykkjan
For lykkjan
Fleiri dæmi um notkun á lykkjum
Stöðupróf T103SP5
Forritunarverkefni

T103-6. Fylki og röðun

Skilgreining fylkis
Annar ritháttur fylkjaskilgreininga
Núllstillingar fylkja
Fylki send til aðferða
Fylki hluta
Margvíð fylki
Röðun
Stöðupróf T103SP6
Forritunarverkefni.....

Inngangur

Bók þessi er fyrst og fremst skrifuð sem námsefni fyrir framhaldsskólanema sem leggja stund á nám í forritun en hún getur vonandi nýst öllum sem áhuga hafa á forritun og vilja tileinka sér hana. Við gerð bókarinnar var því tekið mið af áfangalýsingum TÖL áfanga framhaldsskólans eins og þær eru settar fram í Aðalnámsskrá framhaldsskóla sem gefin var út 1999. Einnig var tekið tillit til þess að þegar er komið út efni sem að hluta til kemur til móts við þessar lýsingar og reynt að láta skörunina verða sem minnsta. Hér er átt við bókina „*Kennslubók í tölvufræði fyrir framhaldsskóla*“ eftir Atla Harðarson. Sú bók reifar sögu tölvufræðinnar og helstu grundvallaratriði hennar og er gert ráð fyrir að hún sé lesin meðfram þessu efni sem fyrst og fremst er ætlað að koma til móts við forritunarþátt áfangamarkmiðanna.

Val forritunarmáls

Forritunarmálið Java var valið af ýmsum ástæðum. Í fyrsta lagi er mikilvægt að málið geti nýst til innleiðslu á grunnhugtökum svo sem breytum, tögum og flæði aðgerða. Í öðru lagi æskilegt að sem best samfella verði milli TÖL áfanganna 103, 203 og 303 svo ekki tapist tími vegna innleiðingu nýs forritunarmáls. Notkun forritunarmáls í þrem áföngum gerir kröfu um fjölhæfni, að það nýtist í flóknu umhverfi og að það sé hlutbundið. Ennfremur er æskilegt að sem mest af forrituninni geti farið fram í textaritþór en krefjist ekki þróunarumhverfis. Bóklegt forritunarnám í framhaldsskóla snýst um að byggja upp hugtaka- og færnigrunn sem reiknað er með að byggt sé ofan á í framhaldsnámi. Valið stóð því fyrst og fremst milli þriggja forritunarmála sem öll komu til greina. C++, Java og C# (lesist sís eða sí-sharp). Þó C++ sé líklega ásamt forvera sínum C mest notaða forritunarmálið þá sker það sig úr því það skortir fjölhæfnina sem Java býður upp á. Java og C# eru hlutbundin frá grunni og málskipan þeirra og

grunnforsendur eru líkar. Þekking á öðru málinu mun því í mörgum tilfellum nýtast í hinu, sérstaklega hvað varðar grunnatriði. Java er samt þegar þetta er skrifað mun fjölhæfara en C#, það keyrist á öllum helstu notendastýrikerfunum (Macintosh, Linux, Windows og fleiri kerfum). C# er orðið til vegna þarfar á sérhæfingu fyrir Microsoft kerfi. Þessar forsendur hafa orðið til þess að Java málið varð fyrir valinu. Þau ár sem ég hef kennt forritun hef ég oft heyrt í gegnum nemendur að forritunarmálið sem kennt er sé úrelt, hvergi notað, eða þá að það sé óþarflega flókið eða of torskilið fyrir byrjendakennslu “. Ég ráðlegg nemendum að taka þess konar „ráðgjöf“ með fyrirvara. Að fullyrða slíkt við nemendur sem oft á tíðum eiga í baráttu við sjálfa sig við að halda sér að námi er mikill ábyrgðarhluti. Hafa ber í huga að kunnátta og færni yfirfærast oftast vel milli forritunarmála. Það er oftast nærtækara að líkja mun á forritunarmálum við muninn á náskyldum tungumálum frekar en t.d. mun á hljóðfærum. Það er líklega auðveldara fyrir nemanda í tungumálum að skipta úr dönslu yfir í norsku en fyrir tónlistarnemanda að skipta af kontrabassa yfir á fiðlu svo tekið sé dæmi.

Um lestur

Lesið 10 -15 mínútur í einu, takið þá 5 mínútna hlé, gerið eitthvað skemmtilegt og byrjið svo aftur. Hæfileiki heilans til einbeitingar nær ekki yfir lengri tíma, en endurnýjast eftir hvíldina. Þegar þetta vinnulag er viðhaft verður lesturinn skemmtilegri. Lesið fyrst lauslega **og notið áherslupenna** til að merkja við mikilvæg atriði. Lesið svo aftur, ítarlega og vel og reynið að skilja öll atriði. Lesið svo í þriðja skiptið og reynið að ná aðalatriðunum og rifja upp hvað þið lærðuð. Notið spássíurnar til að skrifa strax niður athugasemdir og spurningar sem



Áherslupennar— munið eftir þeim.

vakna við lesturinn. Komið svo spurningunum á framfæri ef þið finnið svörin ekki í bókinni. Nám í forritun á ýmsar hliðstæður við tungumálanám. Verkefnið snýst fyrst og fremst um að læra orð og hugtök og hvernig þau tengjast saman. Áður en hægt er að forrita er nauðsynlegt að hafa numið mörg orð og hugtök. Verið því þolinmóð og gefið ykkur tíma til að læra grunnatriðin vel.

Sýnidæmi bókarinnar, leyst forritunarverkefni og annað sem að gagni má koma við forritunarnámið má nálgast á vefnum á vefsvæðinu <http://www.home.is/ragnar/forritunijava/>. Ég hvet ykkur til að þýða og keyra sýnidæmin og leystu forritunarverkefnið til að ná enn betri skilningi á þeim.

Nýjasta Java þróunar- og keyrsluumhverfið (Java Software Development Kit eða Java SDK) má á hverjum tíma nálgast á vefslóðinni <http://java.sun.com>. Ef einungis er ætlunin að hlaða inn keyrsluumhverfinu (Java Runtime Environment eða JRE) á sem einfaldastan hátt dugar að fara á <http://www.java.com>.

Um mun á útgáfum

Fyrstu útgáfu bókarinnar sem kom út í ágúst 2003 var dreift á netinu á pdf formi. Sú leið var ekki farin núna. Bókin er prentuð og pdf skráin hefur verið tekin af netinu. Auk þess hefur umbrot verið fært til betri vegar, letur stækkað, hugtakasafni og myndum bætt við auk þess sem leiðréttingar hafa verið gerðar. Vonast er til að þessar breytingar geri bókina auðveldari í notkun.

Um íslenskun

Ég hef farið þá leiðina við gerð þessa námsefnis að nota fyrst og fremst íslenskuð orð og hugtök enda hægt um vik að sækja þau í orðaforða Tölvuorðasafnsins, en það byggir á starfi orðanefndar Skýrslutæknifélags Íslands. Flest öll íslensku heitin eru sótt þangað. Meirihluti nemenda er ánægður með þetta fyrirkomulag. Ég reyni að létta fólki lesturinn og hugtakanámið með

því að birta enskt heiti orðsins eða hugtaksins innan sviga. Að lokum vil ég þakka menntamálaráðuneytinu sem hefur styrkt gerð þessa námsefnis árin 2002, 2003 og 2004 sem og Fjölbrautaskóla Suðurlands fyrir afnot af prentara við útprentun prófarkar.

Framhaldið

Þótt staðar sé numið við lok TÖL203 áfangans í þessari útgáfu er þó ljóst að enn er hægt að auka við. Bæði er það að bæta má við efni fyrir TÖL303 sem og hitt að Java málið býður upp á ýmis atriði sem vel eiga heima í viðbót við þessa bók. Hvað framtíðin ber í skauti sér er þó alveg undir styrkveitingum sem og viðbrögðum lesenda við þessari útgáfu komið.

Selfossi í júlí 2004.

Ragnar Geir Brynjólfsson.

Viðbót við útgáfu 2.1

Stuðningur við útgáfuna—reikningsnúmer

Gerð kennslubóka á íslensku er ekki ábatasöm atvinnugrein, sér í lagi ef markaður bókanna eru fámennir áfangar framhaldsskólans, áfangar á borð við forritunaráfanga. Forritunaráfangar eru einungis í boði sem kjörsviðsáfangar fyrir fjölmennar brautir á borð við náttúrufræðibraut, viðskiptabraut, málabraut eða félagsfræðibraut. Án dyggilegs stuðnings Menntamálaráðuneytisins væri vart mögulegt að gefa út kennslubækur fyrir þessa og aðra fámenna áfanga. Ráðuneytinu eru færðar þakkir fyrir stuðning sinn við þetta verkefni en jafnframt er leitað eftir frekari stuðningi allra sem nota bókina. Greiða má inn á tékkareikning (höfuðbók 26), reikningsnúmer 38816 í KBbanka — aðalbanka, bankanúmer 301. Kennitala: 310361-4539. Væntanlegum styrktaraðilum eru færðar þakkir fyrir framlag sitt.

Ragnar Geir Brynjólfsson. Selfossi í júlí 2005.

T203-1. Grunnhugtök hlutbundinnar forritunar

Hlutbundna hugmyndafræðin

Af hverju þurfum við hlutbundna forritun? Hvað hefur hlutbundið forritunarmál fram yfir venjulegt forritunarmál þar sem breytur og aðferðir þurfa ekki að tilheyra tilteknum klasa? Þetta skýrist ef við veltum fyrir okkur vanda óhlutbundinnar forritunar. Þegar forrit í óhlutbundnu forritunarmáli verður stórt verður það flókið og erfitt í viðhaldi. Áður en hlutbundin mál festu sig í sessi var leitað lausna með *föllum* (functions) og *stefjum* (procedures) sem líkja má við föll í stærðfræði og *einingum* (modules), sem þjappa á sama stað endurteknum skipunum í forriti. Þetta var kallað mótuð forritun (structured programming). En þegar forritin stækkuðu ennþá dugðu þessi ráð ekki lengur til að halda utan um flókin forrit. Oft var erfitt að hanna og þróa óhlutbundin forrit því aðal byggingareiningar þess, föllin og *gagnagrindurnar* (data structures) voru ekki heppileg til líkanagerðar. Það eru hlutir og klasar í hlutbundnu forritunarmáli aftur á móti.

Í TÖL103 lærðum við að Java vinnur með upplýsingar á tvennan hátt. Sem *frumgögn* (primitive data) eða sem *hluti* (objects). Hlutir eru notaðir til að lýsa samsettum gögnum sem mynda heild ásamt aðferðum sem hægt er að beita á hlutina. T.d. eins og bíl í bílaforriti og því sem hann á að geta gert, t.d. tekið af stað eða stöðvað.

Hlutur tilheyrir alltaf tilteknum *klasa* (class) og segja má að klasinn sé gagnatag hlutarins. Í klasanum eru enn fremur skilgreindar breytur eða aðferðir sem hægt er að kalla á. Java er hlutbundið frá grunni sem þýðir að allar aðferðir í málinu tilheyra einhverjum klasa. Klasanum má því líkja við uppdrátt, fyrirmæli eða skapalón af því hvernig hluturinn eigi að vera. Hægt er að hugsa sér að ef klasi sé smákökuuppskrift þá séu hlutirnir

Hugtök

Fall:

Safn skipana í óhlutbundnu forritunarmáli sem leysa afmarkað verkefni og skila frá sér einni breytu eða vistfangi.

Steffa:

Safn skipana í óhlutbundnu forritunarmáli sem leysa afmarkað verkefni og geta sent frá sér mörg skilagildi gegnum stika.

Eining:

Notað til að lýsa samantekt falla og steffa sem leysa tiltekið verkefni.

Gagnagrind/

Gagnaskipan:

Byggingareiningar forrits sem notaðar eru til að auðvelda úrvinnslu gagna. Dæmi: frumgögn, strengir, fylki.

Frumgögn:

Gagnagerð sem er innbyggð í Java málið, t.d. int, float, char, boolean.

Hlutur:

Sjálfstæð eining í forriti sem hægt er að vinna með og breyta. Eitt eintak af tilteknum klasa, inniheldur eigin gögn og samsvarandi aðferðir.

Klasi:

Safn skilgreininga sem lýsir því hvernig hlutir eigi að vera.

Hugtök

Tilvik:

Íslenskun enska orðsins 'instance.' Annað orð yfir hlut í hlutbundnu forritunarmáli.

kökurnar sem eru bakaðar samkvæmt uppskriftinni. Við borðum ekki uppskriftina, né leggjum hana á borð, en í uppskriftinni felast samt nákvæmar leiðbeiningar um hvernig útlit kakanna eigi að vera sem og um ýmsa eiginleika þeirra svo sem bragð. Annað orð er notað um hlutina en það er orðið *tilvik* (instance) sem er rökrétt því þá er hægt að tala um ákveðin tilvik klasans. Skoðum fyrst sýnidæmi T203Sd1_1 til að glöggva okkur á því hvernig klasaskilgreining er sett upp.

```
/**
*****
Skrá: T203Sd1_1.java
Höfundur: Ragnar Geir Brynjólfsson.
Dagsetning: 2.07.2002.
Skilgreining og eintökun klasa sýnd
*****
*/
//Klasaskilgreining
class A {}
class T203Sd1_1 {
    public static void main(String[] s) {
        A tilvik1 = new A();
    }
}

Úttak þessa forrits er ekkert:
[ragnar@forritun tol203]$ javac T203Sd1_1.java
[ragnar@forritun tol203]$ java T203Sd1_1
[ragnar@forritun tol203]$
```

Í T203Sd1_1 er sýnd málfræði klasaskilgreiningar. Klasinn A, sem gerir ekkert og á ekkert er skilgreindur með orðunum `class A { }`. Ekkert er innan bálksins (slaufusvigaparsins) á eftir A og því á klasinn hvorki gögn né aðferðir. Þetta er samt málfræðilega rétt klasaskilgreining og því er hægt að eintaka (búa til) hlut af A með línunni:

```
A tilvik1 = new A();
```

Reyndar eru tvær klasaskilgreiningar í þessari skrá. Fyrst er klasinn A skilgreindur og síðan er klasinn T203Sd1_1 skilgreindur. T203Sd1_1 er sérstakur því hann á `main()` aðferð, en ekki er nauðsynlegt að allir klasar eigi slíka aðferð, heldur

aðeins þeir klasar sem ætlast er til að Java túlkurinn keyri. Þetta forrit er því keyrt með skipuninni:

```
java T203Sd1_1
```

Ekkert úttak birtist því engar úttaksskipanir eru gefnar í T203Sd1_1. Forritið allt þarf svo að vera í skrá sem heitir T203Sd1_1.java því að það er sá klasi sem á public aðferðina main().

Ekki er skilyrði að klasar séu í sömu skrá og klasarnir sem eintaka þá eins og í T203Sd1_1. Hægt hefði verið að búa til sérstaka skrá með klasanum A án þess að hafa klasann T203Sd1_1 með í sömu skrá. Í T203Sd1_2 er gengið skrefi lengra í því að gera ekkert en í T203Sd1_2. Hérna er aðeins klasaskilgreining en engin public aðferð og engin main aðferð. Það skiptir því ekki lengur máli hvað skráin heitir sem klasinn er í, en það er ekki hægt að keyra skrána með Java túlkinum eins og sést á úttakinu frá honum:

```
Exception in thread "main"
java.lang.NoClassDefFoundError: T203Sd1_2
```

Hérna er Java túlkurinn að segja að engin klasaskilgreining hafi fundist í skránni fyrir klasann

```
/**
*****
Skrá: T203Sd1_2.java
Höfundur: Ragnar Geir Brynjólfsson.
Dagsetning: 14.07.2002.
Skilgreining klasa sýnd
í sinni einföldustu mynd
*****
*/
//Klasaskilgreining
class B {}

Úttak þessa forrits er ekkert en hér er dæmi um nokkrar skipanir:
[ragnar@forritun tol203]$ javac T203Sd1_2.java
[ragnar@forritun tol203]$ java T203Sd1_2
Exception in thread "main"
java.lang.NoClassDefFoundError: T203Sd1_2
[ragnar@forritun tol203]$ ls B.class
B.class
[ragnar@forritun tol203]$
```

Hugtök

Bytecode:

Hið þýdda mál Java gengur undir þessu nafni. Þetta mál líkist vélamáli en það þarfnast javatúlks til að keyrast, ekki örgjörva eins og er með flest vélamál.

Eiginleikar:

Annað orð yfir breytur tiltekins klasa.

Hegðun:

Annað orð yfir aðferðir tiltekins klasa.

Aðferð:

Safn skipana sem ætlað er að leysa afmarkað verkefni.

Kyrrleg breyta/aðferð:

Breyta/Aðferð sem þarfnast ekki eintekningar, þ.e. að hlutir séu búnir til að þær séu vaktar.

Tilviksbreyta/aðferð:

Breyta/aðferð sem þarfnast eintekningar, þ.e. að hlutur sé búinn til, til að hægt sé að nota þær. Þær tilheyra þá þessum tiltekna hlut.

Klasabreyta/aðferð:

Sama og Kyrrleg aðferð/breyta.

Smiður:

Aðferð sem heitir sama og klasinn sem hún tilheyrir. Hefur ekkert skilagildi. Ætluð til að eintaka (búa til) hluti.

T203Sd1_2, en túlkurinn var að leita að klasa með því nafni til að keyra main aðferð í honum. Það sem gerist í bæði Sd1_1 og Sd1_2 er að til verða sérstakar skrár með nafnviðaukanum „class“, A.class í Sd1_1 og B.class í Sd1_2. Þessar skrár myndast í sama skráasafni og textaskrárnar T203Sd1_1.java og T203Sd1_2.java eru þýddar í og þær sjást ef `ls *.class` skipun er gefin í linux. Skipunin `ls B.class` sýnir líka að skráin er til staðar eins og sést á úttakinu hér að ofan. Í class skrám eru klasar geymdir á bytecode formi. Þegar aðrir klasar eru þýddir verða klasar og gögn í class skrám í sama skráasafni aðgengilegir. Klasi B getur staðið einn og sér og heiti skrárinnar sem hann er hluti af skiptir ekki máli því ekki er ætlunin að keyra B heldur verður hann eintakaður eða keyrður í öðrum klösum. Það sem skiptir máli er bytecode skráin B.class sem myndast við þýðinguna, þ.e. þegar skipunin `javac T203Sd1_2.java` er gefin.

Höldum nú áfram með klasagerðina og fikrum okkur frá því að gera ekkert til þess að búa til klasa sem lýsir bíl. Klasinn `Bill` samanstendur af tveim megingerðum upplýsinga: *Eiginleikum* (properties) sem breytur sýna og *hegðun* (behaviour) sem aðferðirnar (methods) sýna. Aðferðir má þekkja á því að þær heita eitthvað og á eftir þeim kemur slaufusvigabálkur (`{ }`) sem oftast inniheldur skipanir.

Klasinn `Bill` í T203Sd3_1 á kyrrlegu (static) breytuna `fjoldi_bila`. Hann á einnig tilviksbreyturnar `Gerd`, `Numer`, `Stada`, `Argerd`, `Hradi` og `Ekinn`. Munurinn á kyrrlegum breytum og tilviksbreytum er sá að tilviksbreytur eru bara sýnilegar og nothæfar í hverju einstöku tilviki (hlut) fyrir sig en kyrrlegar breytur eru sýnilegar í öllum tilvikum (hlutum) tiltekins klasa. Því er einnig algengt að kalla þær *klasabreytur* (class variables). Á eftir upptalningu breytanna kemur svokallaður *smiður* (constructor). Smiðir þekkjast á því að þeir eru aðferðir með engu skilagildi sem heita það sama og klasinn. Í þessu tilfelli eru tveir smiðir

```

/**
*****
Skrá: T203Sdl_3.java
Höfundur: Ragnar Geir Brynjólfsson.
Dagsetning: 14.07.2003.
Klasinn Bill skilgreindur
*****
*/
class Bill {
    static int fjoldi_bila;
    String Gerd;
    String Numer,Stada;
    int Argerd,Hradi,Ekinn;
    Bill() {
        fjoldi_bila++;
    }
    Bill(String g,String n,int a,int e) {
        fjoldi_bila++;
        Gerd=g;Numer=n;Argerd=a;Ekinn=e;
        Stada = "Billinn er kyrrstæður";
    }
    public void keyra(int h) {
        Hradi = h;
        Stada = "Billinn er á keyrslu" + " á hraðanum " + Hradi;
    }
    public void stansa(int km) {
        Ekinn+=km;
        Stada = " Billinn er kyrrstæður. ";
    }
    void skrifa_upplys () {
        System.out.println("\nGerð: " + Gerd
            + "\nNúmer " + Numer
            + "\nEr ekinn: " + Ekinn + " kílómetra."
            + Stada);
    }
}

```

Klasinn er notaður í öðrum forritum.

Bill í klasanum. Sá fyrri gerir ekkert annað en skila breytunum núllstilltum og hækka kyrrlegu breytuna fjoldi_bila um einn. Hinn síðari gefur tilviksbreytunum gildi auk þess að hækka fjoldi_bila um einn. Lítum aðeins nánar á hvað gerist þegar kallað er á new:

Kallað er á smið fyrir klasann sem tekur frá minnissvæði og eintakar hlutinn.

Tilviksbreytur hlutarins sem ekki fá sérstök gildi eru núllstilltar, þ.e. tölubreytur fá 0, hlutir fá null (þ.e. vísa ekki á minnissvæði), boolean breytur fá false og char breytur fá '\0'

Tvö atriði greina smiði frá venjulegum aðferðum:

Þeir heita sama og klasinn en það geta aðferðir aldrei. Þeir eru ekki af neinu tilteknu tagi, ekki einu sinni void og því geta þeir ekki skilað gildi með return setningu.

Á eftir smiðunum koma void aðferðirnar `keyra()` og `skrifa_upplys()`. Það að aðferðirnar eru void þýðir að þær skila ekki neinu gildi til annarra aðferða heldur gera eitthvað sjálfar, svo sem gefa breytum gildi eða skrifa eitthvað út. Lítum nú á `T203Sd1_4` sem nýtir bílaklasann, býr til eintök af honum og notar aðferðirnar til að lýsa hegðun eintakanna.

Í `T203Sd1_4` er ekki nauðsynlegt að sækja bíla-

```
/**
*****
Skrá: T203Sd1_4.java
Höfundur: Ragnar Geir Brynjólfsson.
Dagsetning: 14.07.2002.
Klasi sem eintakar Bill
*****
*/
class T203Sd1_4 {
    public static void main(String[] s) {
        Bill a = new Bill();
        Bill b = new Bill("Bjalla", "AD 123", 2003, 0);
        a.keyra(50);
        b.keyra(90);
        a.skrifa_upplys();
        b.skrifa_upplys();
        a.stansa(14);
        b.stansa(55);
        a.skrifa_upplys();
        b.skrifa_upplys();
        System.out.println("Bílar alls: " + Bill.fjoldi_bila);
    }
}
```

Úttak þessa forrits er:

```
[ragnar@forritun tol203]$ javac T203Sd1_4.java
[ragnar@forritun tol203]$ java T203Sd1_4
```

```
Gerð: null
Númer null
Er ekinn: 0 kílómetra. Bíllinn er á keyrslu á hraðanum 50
```

```
Gerð: Bjalla
Númer AD 123
Er ekinn: 0 kílómetra. Bíllinn er á keyrslu á hraðanum 90
```

```
Gerð: null
Númer null
Er ekinn: 14 kílómetra. Bíllinn er kyrrstæður.
```

Frh. Úttaks úr T203Sd1_4:

```
Gerð: Bjalla  
Númer AD 123  
Er ekinn: 55 kílómetra. Bíllinn er kyrrstæður.  
Bílar alls: 2  
[ragnar@forritun tol203]$
```

klasann né hafa hann í sömu skrá því búið er að þýða hann með `javac` skipun og skráin `Bill.class` er til staðar í sama skráasafni. Byrjað er á því að búa til tvo bíla, fyrst **a** með stikalaus smiðnum og svo **b** með smiðnum með stikunum sem gefa tilviksbreytunum gildi. Stiki er breyta sem send er til aðferðar í gegnum haus hennar. Báðir smiðirnir telja kyrrlegu breytuna `fjoldi_bila` upp um einn. Bílarnir eru látnir taka af stað með `keyra()` aðferðinni og tilviksbreytunni `Hradi` gefið gildi í báðum tilfellum. Þá eru upplýsingar skrifaðar út með `skrifa_upplys()` aðferðinni. Loks eru bílarnir látnir stansa eftir ákveðinn ekinn kílómetrafjölda og tilviksbreytu beggja, `Ekinn` gefið gildi um leið. Því næst eru upplýsingar skrifaðar út og að lokum er gildi kyrrlegu breytunnar `fjoldi_bila` skrifað. Takið eftir hvernig tilviksgögnin og aðferðirnar eru háð því að kallað sé á þær með nafni hlutarins sem þær tilheyra og punktvirkja og hversu ólíkt það er og þegar gildi kyrrlegu breytunnar `fjoldi_bila` er skrifað út. Í tilfelli kyrrlegra gagna nægir að skrifa heiti klasans ásamt punktvirkja fyrir framan. Auðvelt er að þróa þetta forrit, t.d. þannig að hægt sé að reikna hversu langt bíllinn hefur ekið þegar hann er á ferð í stað þess að skrá fjölda ekinna kílómetra þegar hann hefur stansað, en við það verður dæmið flóknara.

Hægt er að hugsa sér forritagerð á fleiri en einn hátt. Ein aðferð er að líta svo á að forrit sér runa skipana sem framkvæmdar eru hver á eftir annarri. Þessi aðferð er venjulega kölluð á ensku „procedural programming;“. Margir stíga sín fyrstu forritunarskref í óhlutbundnum málum þar sem þetta verklag var viðhaft. Eitt af því fyrsta sem forritari lærir í þessum málum er hvernig brjóta megi

vandamál niður í afmarkaða hluta sem hver um sig inniheldur fáar skipanir.

Hlutbundin forritun horfir á málið frá öðrum sjónarhóli. Lausnin er hönnuð með tilliti til verkefnisins fremur en hvernig best sé að leysa það í tölvu. Hlutbundið forrit er mengi hluta sem leysa tiltekið verkefni. Hver hlutur er sjálfstæð eining sem hefur samskipti við aðra hluti á sérstakan hátt. Þetta gerir að verkum að í tiltölulega stuttum forritum er hægt að setja upp líkön, að vísu frumstæð eins og í tilfelli bílaklasans en líkön samt. Skoðum nú hvernig klasar og hlutir eru notaðir í flóknara forriti. Í forriti T203Sd1_5 ímyndum við okkur að til séu örsmá vélmenni, nanóvélar sem hægt er að senda inn í blóðráðs og vinna tiltekin verkefni, svo sem að eyða veirum. Skoðum fyrst klasann `Nanovel` sem getur verið í hvaða skrá sem er, t.d. `T203Sd1_5`.

Tilviksbreytur klasans `Nanovel` eru:

```
String stada;
int numer;
boolean fundin_veira = false;
boolean hlut_eytt = false;
```

Tilviksbreyta (instance variable) geymir eiginleika ákveðins tiltekins hlutar. Þessir eiginleikar geta verið og eru oftast mismunandi milli einstakra tilvika/hluta, þ.e. einstakra nanóvéla í þessu tilfelli. Hægt er að gefa tilviksbreytum gildi þegar hlutur er búinn til eða *eintakaður* (instantiated) og þessi gildi geta verið þau sömu á meðan hluturinn er til. Það er einnig hægt að breyta þeim meðan á keyrslu forritsins stendur.

Kyrrleg breyta eða *klasabreyta* (static variable - class variable) geymir eiginleika alls klasans og allra tilvika hans (hluta). Aðeins eitt gildi er geymt fyrir hverja klasabreytu án tillits til hversu mörg tilvik (hlutir) eru til af klasanum. Dæmi um klasabreytur í `Nanovel` klasanum eru breyturnar sem innihalda tímann og heildarfjölda eyddra veira.

Hugtök

Aðferð:

Safn skipana sem ætlað er að leysa afmarkað verkefni.

Eintökun:

Þýðing á 'instantiation'. Notað yfir það þegar hlutur er búinn til.

Kyrrleg breyta/aðferð:

Breyta/Aðferð sem þarfnast ekki eintekningar, þ.e. að hlutir séu búnir til að þær séu vaktar.

Tilviksbreyta/aðferð:

Breyta/aðferð sem þarfnast eintekningar, þ.e. að hlutur sé búinn til, til að hægt sé að nota þær. Þær tilheyra þá þessum tiltekna hlut.

Klasabreyta/aðferð:

Sama og Kyrrleg aðferð/breyta.

```

/**
*****
    Skrá: T203Sd1_5.java
    Höfundur: Ragnar Geir Brynjólfsson.
    Dagsetning: 30.06.2002.
    Forrit sem býr til klasa ímyndaðra
    nanóvélmenna sem eyða veirum.
*****
*/
import java.util.Random;
class Nanovel {
    //Klasabreyturnar skilgreindar.
    static int eyddar_veirur;
    static int timi;
    //Tilviksbreyturnar skilgreindar.
    String stada;
    int numer;
    boolean fundin_veira = false;
    boolean hlut_eytt = false;
    //Aðferð sem prentar út upplýsingar.
    void athugaStodu() {
        if (fundin_veira) {
            System.out.println("Nanóvél " + numer
                + " eyddi veiru...");
            Nanovel.eyddar_veirur++;
            System.out.println("Eyddar veirur alls: "
                + Nanovel.eyddar_veirur);
            hlut_eytt=true;
        }
    }
}

```

Úttak þessa forrits er ekkert en skráin `Nanovel.class` verður til á skrá-asafninu við þýðingu:

```

[ragnar@forritun tol203]$ javac T203Sd1_5.java
[ragnar@forritun tol203]$ ls Nanovel.class
Nanovel.class
[ragnar@forritun tol203]$

```

```

static int eyddar_veirur;
static int timi;

```

Ef tilviksbreyta geymdi tímann þá myndi hver hlutur hafa sinn sérstaka tíma. Það gæti haft í för með sér vandamál ef róbótarnir eiga að framkvæma eitthvað verk á tilteknum tíma. Klasabreytan `timi` gerir að verkum að þetta er ekki vandamál því hún er sýnileg úr öllum hlutunum og allir hlutirnir geta breytt henni. Klasabreytur eru þannig notaðar til samskipta milli hluta í sama klasa eða til að hlutir geti fylgst með upplýsingum um ástand mála. Sýnidæmi `T203Sd1_6` inniheldur klasa sem kallar á `Nanovel` og nýtir aðferðir hans í ímyndaðri veiruleit. Úttak þess er birt hér á eftir og

Úttak T203Sd1_6 er t.d.:

```
[ragnar@forritun tol203]$ java T203Sd1_6
...
Nanóvél 79303 eyddi veiru...
Eyddar veirur alls: 1519
Liðnar mínútur: 0
Nanóvél 79318 eyddi veiru...
Eyddar veirur alls: 1520
Liðnar mínútur: 0
Nanóvél 79404 eyddi veiru...
Eyddar veirur alls: 1521
Liðnar mínútur: 0
Nanóvél 79491 eyddi veiru...
Eyddar veirur alls: 1522
Liðnar mínútur: 0
...
```

sýnidæmið má nálgast í fullri lengt á vefslóðinni sem gefin er upp í inngangi bókarinnar.

Aðferðir - hegðun**Hugtök****Aðferð:**

Safn skipana sem ætlað er að leysa afmarkað verkefni.

Hegðun vísar til þess sem klasi eða hlutir geta gert við sig sjálfa eða aðra hluti. Hegðun getur verið að breyta eiginleikum hlutar, tilviksbreytunum, taka á móti upplýsingum frá öðrum hlutum og senda tilkynningar til hluta og biðja þá að gera eitthvað. Nanóvélaróbótarnir hafa t.d. þá hegðun að tilkynna ef þeir finna veiru, telja upp heildarfjölda eyddra veira og eyða sjálfum sér úr módelinu með því að stilla búlska breytu. Hegðun klasa er útfærð með *aðferðum* (methods) og þessi fyrrnefnda hegðun nanóvélaaklasans er útfærð í aðferðinni:

```
void athugaStodu()
```

Aðferðir eru mengi skyldra setninga í klasa sem leysa afmarkað verkefni. Aðferðirnar eru notaðar á svipaðan hátt og föll og undirforrit í aðgerðamálunum. Hlutir hafa samskipti sín á milli með því að nota aðferðir. Klasi eða hlutur getur kallað á aðferðir í öðrum klasa eða hlut af mörgum ástæðum t.d. til að :

Tilkynna breytingu á öðrum hlut.

Til að segja hlutum að breyta einhverju hjá sér.

Til að biðja annan hlut að framkvæma eitthvað.

Til dæmis væri hægt að útfæra Nanóvélarnar frekar þannig að þær nýttu aðferðir til að skiptast á staðsetningum sínum og forðast árekstra.

Alveg á sama hátt og það eru tilviksbreytur og klasabreytur þá eru til tilviksaðferðir og klasaaðferðir. *Tilviksaðferðir* (Instance methods) eru svo algengar að þær eru bara kallaðar aðferðir (methods). Aðferðin `athugaStodu()` í klasa `Nanóvélanna` er t.d. tilviksaðferð því orðið `static` stendur ekki fyrir framan hana. Tilviksaðferðir eru notaðar þegar verið er að vinna með tilviksbreytur hlutanna. Klasaaðferðir vinna með klasann í heild sinni. Orðið `static` fyrir framan aðferðina gerir hana að klasaaðferð. `main()` aðferðin er því klasaaðferð. Ólíkt tilviksaðferðum þá þarf ekki að búa til hlut til að hægt sé að kalla á klasaaðferð. Ef frátekna orðið `static` er fyrir framan heiti aðferðarinnar þá er hún klasaaðferð. Klasaaðferðir geta verið aðgengilegar öðrum klösnum ef þær eru með `public` aðgangsstýringar en eru það ekki ef þær eru með `private` aðgangsstýringu. Tökum sem dæmi gamlan kunningja úr TÖL103. Við getum sagt:

```
double root = Math.sqrt(453.0);
```

Hérna er `sqrt` klasafall í `Math` sem kallað er á með því að skrifa heiti klasans sem það tilheyrir fyrir framan og setja punkt á milli. Aðrir klasar sem bjóða upp á klasaaðferðir eru t.d. klasarnir sem eru samnefndir grunnklösunum, þ.e. `Integer`, `Float` og `Boolean` klasarnir sem oft eru notaðir í bland með grunnklösunum `int`, `float` og `boolean`. Ástæðan fyrir vinsældum þessara klasa er sú að hægt er að gera margt með þeim sem ekki er hægt með frumbreytunum, t.d. eins og að lesa gildi úr streng. Í setningunni:

```
int count = Integer.parseInt("42");
```

Hugtök

Tilviksaðferð:

Aðferð sem er ekki kyrrleg. Þ.e. hver hlutur fær sitt eigið eintak af aðferðinni. Til að kalla á aðferðina verður að ná í hlutinn sem hún tilheyrir.

er strengurinn "42" lesinn inn og `parseInt` aðferðin sem er klasaaðferð í `Integer` breytir innihaldi strengsins í `int` og skilar frá sér.

Hjúpun

Hægt er að nálgast klasa og hluti á tvennan hátt. Annars vegar þarf að úthugsa þá í smáatriðum á meðan verið er að hanna þá og forrita en hins vegar þarf að vera hægt að hugsa um þá eins og einingar sem okkur koma lítið við að öðru leyti en því að þær geri það sem þær eiga að gera. Við viljum geta notað tilvik/hluti án þess að þurfa að skilja til hlítar innviði þeirra. Við viljum geta kallað á aðferðir sem vinna sitt verk án þess að við þurfum að eyða tíma í að brjóta þær til mergjar. Sá eiginleiki að þurfa ekki að huga að neinu öðru en inntaki, úttaki og skilagildi er einn af ávinningum hlutbundinnar forritunar því þar með opnast sá möguleiki að endurnýta klasa. Þessir endurnýttu klasar geta verið klasar sem forritarinn sjálfur eða aðrir hafa gert. Við þurfum t.d. og sem betur fer ekki að skilja til hlítar hvernig gírkassi í bíl virkar til þess að geta ekið bíl. Það myndi auka fyrirhöfn okkar verulega ef við þyrftum að rifja það upp í hvert skipti sem við ætluðum að nota bílinn. Þannig má segja að gírstöng á bíl sé hjúpun á gírkassanum. Af þessum orsökum er talið æskilegt að tilviksbreytum eða klasabreytum sé ekki breytt nema með sérstökum aðferðum sem skilgreindar eru í klasanum það má segja að þær séu *hjúpaðar* (encapsulated). Með sérstökum *aðgangsstýringum* (visibility modifiers) er síðan hægt að loka algerlega á aðgang frá utanaðkomandi klösum eða takmarka hann verulega. Þeir klasar sem hingað til hafa verið sýndir hafa ekki haft neinar aðgangsstýringar sem hefur gefið öðrum klösum leyfi til að breyta innihaldi þeirra. Aðgangsstýringar skiptast í *private*, *protected* og *public*.

Hugtök

Hjúpun:

Notað yfir það þegar aðgangur að breytum eða aðferðum er takmarkaður.

Aðgangsstýring:

Hugtak sem lýsir því hversu langt á að ganga í hjúpun (takmörkun aðgangs að breytum og föllum). Aðgangsstýringar geta verið *private*, *protected* eða *public*. Sjá töflu T203_1_1.

Tafla T203_1_1. Aðgangsstýringar

<code>private</code>	Enginn aðgangur leyfður frá öðrum klösum
<code>protected</code>	Aðgangur leyfður frá klösum í sama pakka sem og erfingjum klasans (sjá erfðahugtakið í næsta kafla).
<code>public</code>	Aðgangur heimill frá öllum klösum
Ekkert tekið fram	Sama og <code>public</code>

Ef breyta, aðferð eða klasi er **private** þá er enginn aðgangur leyfður frá köllum sem koma frá aðferðum utan klasans. Ef aðgangsstýring er **protected** þá er aðgangur leyfður frá öllum klösum í sama pakka og öllum undirklösum allsstaðar en ef aðgangsstýring er **public** þá er aðgangur heimill frá öllum klösum hvar sem er. Ef ekkert er tekið fram eins og hingað til hefur verið þá er aðgangur heimill frá öllum klösum í sama pakka. Það er því ljóst að hægt er að laga verulega klasa nanóvél-mennanna hérna að framan. Í stað þess að kalla beint á tilviksbreytur klasans `Nanovel` úr öðrum klasa með setningu á borð við:

```
n[i].fundin_veira=true;
```

væri betra að láta `fundin_veira` og hinar tilviksbreyturnar fá `private` aðgangsstýringu inni í `Nanovel` og skrifa sérstakar aðferðir `eyddur()` og `veira_fannst()` til að breyta gildi þeirra. Þannig breyttur liti klasinn út eins og í T203S-d1_7.

Takið eftir að þetta er sami klasi og í T203Sd1_5 en núna breyttur þannig að tilviksbreyturnar `hlut_eytt` og `fundin_veira` eru hjúpaðar með aðferðunum `eyddur()` og `veira_fannst`

```

/**
*****
Skrá: T203Sd1_7.java
Höfundur: Ragnar Geir Brynjólfsson.
Dagsetning: 15.07.2003.
Forrit sem býr til klasa
nanóvélmenna sem eyða veirum.
*****
*/
class Nanovel {
    //Klasabreyturnar skilgreindar.
    static int eyddar_veirur;
    static int timi;
    //Tilviksbreyturnar skilgreindar.
    private String stada;
    private int numer;
    private boolean fundin_veira = false;
    private boolean hlut_eytt = false;
    //aðferð sem prentar út upplýsingar.
    public void athugaStodu() {
        if (fundin_veira) {
            System.out.println("Nanóvél " + numer
                + " eyddi veiru...");
            Nanovel.eyddar_veirur++;
            System.out.println("Eyddar veirur alls: "
                + Nanovel.eyddar_veirur);
            hlut_eytt=true;
        }
    }
    public boolean eyddur() {
        return hlut_eytt;
    }
    public void veira_fannst() {
        fundin_veira = true;
    }
}

```

(). Einnig er búið að setja private aðgangsstýringu á tilviksbreyturnar sem tryggir að aðrir klasar hafa ekki aðgang að þeim. Það er því eingöngu hægt að breyta breytunni fundin_veira með því að kalla á aðferðina veira_fannst() og það er eingöngu hægt að sækja gildið á hlut_eytt með því að kalla á aðferðina eyddur (). Þessar breytingar kunna að virðast léttvægar í þessu dæmi en breytingar af þessu tagi eru samt almennt til bóta þegar hlutirnir eru settir í víðara samhengi samanber rökstuðninginn að framan.

Meira um smíði

Í T203Sd1_6 þurfti sérstaka setningu í forritinu til að úthluta hverju tilviki raðnúmeri. Það var gert

með for-lykkjunni:

```
for(int i=0;i<n.length;i++) {
    //Nánóvélar eintakaðar og númeraðar.
    n[i] = new Nanovel();
    n[i].numer = i+1;
}
```

Þ.e. það er línan:

```
n[i].numer = i+1;
```

sem sér um það. Þetta verklag brýtur gegn grunnhugmyndinni um hjúpun sem kveður á um að aðeins aðferðir klasans skuli vinna með tilviksbreytur hans. Til að ráða bót á þessu eru *smiðirnir* (constructors) sem keyrast þegar hlutir eru búnir til — eintakaðir. Smiðirnir heita svo því þeir smíða hlutina ef svo má segja. Sjálfgefni smiðir eru reyndar innbyggðir í málið svo ekki er þörf á að forrita smiði sé einungis ætlunin að eintaka hluti sem koma með allar breytur með núllgildi. Kallað er á sjálfgefni smið fyrir hvern klasa þegar kallað er á new ef ekki er búið að forrita smið. Hægt er að skilgreina smiði ef æskilegt er að einhver atriði séu framkvæmd um leið og hluturinn er búinn til, t.d. ef gefa þarf tilviksbreytum hans ákveðin gildi sem eru þá önnur en þau sjálfgefni. Í dæminu um nanóvélarar hentar ekki að gefa öllum hlutum raðnúmerið 0 eins og sjálfgefni smiðurinn gerir því þeir eiga allir að fá sitt sérstaka raðnúmer.

Að svo mæltu er rétt að birta nanóvélaforritið aftur í heild sinni en núna nokkuð breytt í T203S-d1_8. Það hefur fengið smið sem sér um að úthluta hverju eintaki raðnúmeri og aðferðir sem vinna með tilviksbreyturnar sem fengið hafa private aðgangsstýringar.

Takið eftir því hvernig þetta forrit er frábrugðið hinu fyrra, þ.e. Kominn er smiður sem gefur raðnúmeri hlutarins gildi og hægt væri að kalla á með einni línu:

```
n[i] = new Nanovel(i+1);
```

Hugtök

Smiður: Aðferð sem keyrist þegar hlutur er eintakaður. Smiður hefur ekkert skilagildi og heitir alltaf sama og klasinn sem hann tilheyrir. Smiðir eru oft **fjölbundnir** (overloaded), þ.e. í hverjum klasa geta verið margir smiðir, og ræður þá stikaföldi- og gerð kallsins því hvaða smiður er vakinn.

```

/**
*****
Skrá: T203Sd1_8.java
Höfundur: Ragnar Geir Brynjólfsson.
Dagsetning: 2.7.2002.
Endurbættur klasi nanóvélmenna sem eyða veirum.
*****
*/
import java.util.Random;
class Nanovel {
    //Smiður skilgreindur
    Nanovel (int radnumber) {numer=radnumber;}
    //Klasabreytur skilgreindar.
    static int eyddar_veirur,timi;
    //private tilviksbreytur skilgreindar
    private String stada;
    private int numer;
    private boolean fundin_veira,hlut_eytt;
    //aðferð sem prentar út upplýsingar.
    public void athugaStodu() {
        if (fundin_veira) {
            System.out.println("Nanóvél " + numer
                               + " eyddi veiru...");
            Nanovel.eyddar_veirur++;
            System.out.println("Eyddar veirur alls: "
                               + Nanovel.eyddar_veirur);
            hlut_eytt=true;
        }
    }
    public boolean eyddur() {return hlut_eytt;}
    public void veira_fannst() {fundin_veira = true;}
}

```

og einnig væri hægt að kalla á aðferðirnar sem gefa tilviksbreytunum gildi með kalli á borð við:

```
if(!n[i].eyddur())
```

```
    og
    n[i].veira_fannst();
```

Með þessum breytingum má segja að nanóvélaforritið komi til móts við kröfur um hjúpun því í stað beinna kalla á tilviksbreytur eru komin köll á aðferðir. Einnig er rétt að vekja athygli á orðinu return sem kemur fyrir í aðferðinni:

```
public boolean eyddur() {
    return hlut_eytt;
}
```

Return verður að koma fyrir í aðferðinni því til-

greint er í haus hennar að hún skili frá sér boolean gildi, (þ.e. er ekki void). Ef skilagildi aðferðar er ekki void þá skilar hún hlut af tiltekinni tegund þegar hún hættir vinnslu. Taka verður sérstaklega fram inni í aðferðinni hvenær á að skila gildinu með return setningu. Gildið sem kemur fyrir aftan return er það sem aðferðin skilar frá sér, í þessu



Forritunarnemendur í Fjölbrautaskóla Norðurlands Vestra á Sauðárkróki á fjarfundi vorið 2003. Ljós. RGB.

tilfelli gildinu í tilviksbreytunni `hlut_eytt`. Þessi sýnidæmi ásamt forritum sem keyra þau má nálgast í heild sinni á vefslóðinni sem gefinn er upp í inngangi bókarinnar. Nemendur eru hvattir til að sækja þau, þýða og keyra.

Stöðupróf T203Sp1

1.
Tilviksbreytur (instance variables)
 - A. eru bara sýnilegar í einni aðferð.
 - B. eru skilgreindar utan aðferða.
 - C. eru auðþekktar á orðinu `static`
 - D. koma fyrir utan klasabálksins.
 - E. eru alltaf inni í `main()`.

2.
Klasabreytur (class variables):
 - A. Má þekkja á því að orðið „`classic`“ kemur fyrir í skilgreiningu þeirra.
 - B. Má þekkja á því að orðið „`static`“ kemur fyrir í skilgreiningu þeirra.
 - C. eru sýnilegar í einstökum aðferðum.
 - D. Hafa einstök gildi í hverju tilviki.
 - E. eru alltaf skilgreindar inni í `main()`.

3.
Klasaaðferð (class method) þekkist á því að:
 - A. Skilagildi hennar er ávallt klasi.
 - B. Til að hægt sé að nota þær þarf að eintaka klasann.
 - C. Hún fær senda raunstika.
 - D. Haus hennar inniheldur formstika.
 - E. Orðið „`static`“ kemur fyrir í skilgreiningu hennar.

4.
Í skilgreiningunni á `main`: `public static void main(String[] arguments){}`
 - A. Þá þýðir `static` að `main()` geti komið fyrir utan klasa.
 - B. Þá þýðir `public` að aðrir klasar hafi ekki aðgang að aðferðinni.
 - C. Þá þýðir `static` að `main()` sé klasafall.
 - D. Þá þýðir `void` að aðferðin skilar frá sér gildi.
 - E. Kemur fram að aðferðin sé fjölbundin og skili gildi með `return` setningu.

5.
Smiðir (Constructor methods):
 - A. eru sérstakir því ekki er hægt að fjölbinda þá.
 - B. Skila frá sér gildum með `return` setningum.
 - C. eru klasaaðferðir.
 - D. Heita alltaf sama nafni og klasinn sem þeir tilheyra.
 - E. Hafa ekkert gildissvið.

6.
Hjúpun (encapsulation):
 - A. Gerir að verkum að forritarar þurfa að þekkja innviði allra klasa sem þeir nota.
 - B. Verndar gögn klasa fyrir öllum óviðkomandi.

- C. Auðveldar notkun gildissviðsins.
- D. Krefst þess að return setning sé notuð.
- E. Kemur fyrir utan klasa í Java.

7.

```
class abc {}
```

- A. Er málfræðilega rétt skilgreining á klasanum abc.
- B. Myndi framkalla villu í þýðingu.
- C. Lýsir hlutnum abc.
- D. Lýsir klasanum abc sem á eina aðferð.
- E. Lýsir klasanum abc sem á eina tilviksbreytu.

8.

```
Klasaskilgreiningin: class C { static int a; int b; } Lýsir klasanum C sem á
```

- A. eina aðferð a og eina tilviksbreytu b.
- B. tvær aðferðir a og b.
- C. tvær tilviksbreytur a og b.
- D. eina kyrrbreytu (klasabreytu) a og eina tilviksbreytu b.
- E. eina klasaaðferð a.

9.

```
Klasaskilgreiningin class D { D () {} public void skrifa() {} } Lýsir klasa sem á
```

- A. Tvo smíði D () og skrifa().
- B. Tvær tilviksbreytur D () og skrifa().
- C. Einn smíð D () og eina aðferð skrifa().
- D. Klasabreytuna D() og tilviksbreytuna skrifa().
- E. Eina tilviksbreytu D().

10.

Smiðir

- A. eru tilviksbreytur sem heita sama og klasinn.
- B. eru klasabreytur sem heita sama og klasinn.
- C. eru aðferðir með void skilagildi sem heita sama og klasinn.
- D. eru aðferðir með engu skilagildi og heita sama og klasinn.
- E. eru aldrei fleiri en einn í hverjum klasa.

11.

Þegar klasaaðferð er notuð í öðrum klösum

- A. þarf alltaf fyrst að vera búíð að eintaka hlut.
- B. þarf fyrst að búa til hlut með new virkja.
- C. þá verður hún að vera með private aðgangsstýringu.
- D. þá þarf ekki að eintaka hlut.
- E. þá má hún ekki vera með public aðgangsstýringu.

12.

Ef textaskráin t.java inniheldur klasaskilgreininguna class E {} og er þýdd með skipuninni javac t.java

- A. Þá er í lagi að gefa skipunina java E.
- B. Þá myndast bytecode skráin E.class í sama skráasafni.
- C. Kemur villa því skráin verður að heita E.java.
- D. Þá kemur villa því main aðferð er ekki til staðar.
- E. Þá kemur villa því engar tilviksbreytur eru til staðar.

Forritunarverkefni T203Fv1_1

Skrifið klasann Braud sem er klasi sem lýsir brauði. Klasinn á int klasabreytuna fjoldi_brauda sem gefur til kynna hversu mörg brauð er búið að baka og String tilviksbreyturnar tegund sem getur t.d. tekið gildin fint, gróft, rúgbrauð o.s.frv og , litur sem lýsir lit brauðsins. Hann á einnig double tilviksbreyturnar thyngd sem segir hversu þungt brauðið er og verd. Klasinn á að hafa tvo smíði, einn sem gerir ekkert annað en telja upp fjolda_brauda og annan sem auk þess að telja upp fjolda_brauda gefur tilviksbreytunum gildi

```
/**
*****
Skrá: T203Fv1_1.java
Höfundur: ...
Dagsetning: ...
Klasi sem lýsir brauði
*****
*/
class Braud {

... Forritið klasann ...

}

public class T203Fv1_1 {
    public static void main(String[] inn) {
        Braud b1 = new Braud("Gróft", "Brúnt", 120.0, 900.0);
        Braud b2 = new Braud("Fint", "Hvitt", 250.0, 1200.0);
        b1.skrifa_uppl();
        b2.skrifa_uppl();
        ... gerið það sem beðið er um ...
    }
}
```

sem koma úr stikum. Búið til eina aðferð `skrifa_uppl()` sem skrifar út eiginleika hvers brauðs auk þess að segja hve mörg brauð er búið að baka. Skrifið síðan prófunarklasa `T203Fv1_1` sem kallar á smiðina tvo, býr til 3 brauð ásamt upplýsingum og skrifar þær svo út. Skrifið út upplýsingar um þrjú brauð, eitt sem búið er til með einfalda smiðnum og tvö sem búin eru til með hinum smiðnum.

Forritunarverkefni T203Fv1_2

Vinnið áfram með `T203Fv1_1` og búið nú til fylki með brauði. Lengd fylkisins á að vera 2 svo það

Úttak þessa forrits er:

```
[ragnar@forritun tol203]$ javac T203Fv1_2.java
[ragnar@forritun tol203]$ java T203Fv1_2
Les upplýsingar um brauð númer 1
Sláðu inn tegund brauðsins: Hrökkbrauð
Sláðu inn lit brauðsins: Ljósbrúnt
Sláðu inn verð brauðsins: 199.0
Sláðu inn þyngd brauðsins: 50
Les upplýsingar um brauð númer 2
Sláðu inn tegund brauðsins: Samlokubrauð
Sláðu inn lit brauðsins: Brúnt
Sláðu inn verð brauðsins: 120.0
Sláðu inn þyngd brauðsins: 850.0

Brauð:
Tegund: Hrökkbrauð
Litur: Ljósbrúnt
Verð: 199.0
Þyngd: 50.0

Brauð:
Tegund: Samlokubrauð
Litur: Brúnt
Verð: 120.0
Þyngd: 850.0

[ragnar@forritun tol203]$
```

getur bara innihaldið 2 brauð. Bætið við aðferðinni `lesa_uppl()` við klasann sem les inn upplýsingar frá lyklaborðinu og setur þær í tilviksbreyturnar. Notið tvær for lykkjur, eina til að lesa upplýsingarnar um brauðið inn með kalli á `lesa_uppl()` og aðra til að skrifa þær út með `skrifa_uppl()`. Ábendingar: `Lyklabord.java` verður að vera þýddur og til staðar á skráasafninu til að hægt sé að nota aðferðir hans. Til að sjá hvernig þið gerið fylki hluta er hægt að skoða nanóvélaforritið í

```

/**
*****
Skrá: T203Fv1_3.java
Höfundur: Ragnar Geir Brynjólfsson.
Dagsetning: 03.07.2002.
Flytur út klasa sem gefur heiltöluslembitölur.
*****
*/
import java.util.Random;
class IntSlemba {
    Random r = new Random();
    public int slembiInt() {
        return this.slembiInt(0,99);
    }
    public int slembiInt(int minni,int staerri) {
        int sponnin = staerri-minni+1;
        return Math.abs(r.nextInt(sponnin))+minni;
    }
}
public class T203Fv1_3 {
    public static void main(String[] innstr) {
        final int PRUFUR = 100;
        IntSlemba i = new IntSlemba();
        System.out.println("Fyrri prufa, sjálfgefin gildi 0-99:");
        for(int j=0;j<PRUFUR;j++)
            System.out.print(i.slembiInt() + " ");
        System.out.println();
        System.out.println("Síðari prufa, bilið -4 til 5: ");
        for(int j=0;j<PRUFUR;j++)
            System.out.print(i.slembiInt(-4,5) + " ");
        System.out.println();
    }
}

```

Úttak þessa forrits er:

```

[ragnar@forritun tol203]$ javac T203Fv1_3.java
[ragnar@forritun tol203]$ java T203Fv1_3
Fyrri prufa, sjálfgefin gildi 0-99:
7 42 8 20 88 35 99 0 63 7 55 81 53 79 70 4 65 64 51 67 76 63 94 57
47 77 32 12 56 21 14 44 66 83 7 37 25 22 89 68 63 90 81 20 6 42 76
48 18 16 20 96 3 11 47 33 43 32 87 35 10 75 49 49 20 52 31 28 62
80 62 36 98 61 41 73 85 15 14 87 92 38 47 41 79 85 33 32 49 9 72
58 26 35 46 91 55 88 91 20
Síðari prufa, bilið -4 til 5:
-4 5 1 -1 -2 3 4 -1 5 2 3 1 -3 2 5 -3 1 4 3 -2 0 3 -4 -3 -4 5 1 3
-1 3 3 3 -4 4 0 5 4 4 -1 -4 2 1 -4 1 3 0 4 0 -1 -3 -4 5 -1 4 2 0 4
0 3 4 -2 -3 -1 5 1 -2 0 0 3 4 0 1 5 5 1 -3 0 -2 -2 -4 3 -4 4 2 -1
0 5 -3 5 2 1 5 1 3 3 -1 1 -3 0 1
[ragnar@forritun tol203]$

```

kafnanum.

Forritunarverkefni T203Fv1_3 (Leyst)

Skrifið klasann `IntSlemba` sem flytur út aðferð `slembiInt()` sem skilar heiltöluslembitölum á ákveðnu bili sem sent er til hennar. Ef ekkert bil er sent til aðferðarinnar þá skilar hún póstífum heiltöluslembitölum frá 0-99 að báðum meðtöldum. Skrifðu klasa sem prófar báðar aðferðirnar. Áb. Heimilt er að nota `nextInt(n)` aðferðina úr `Random` klasanum sem gefur heil-töluslembitölur frá 0-n.

Forritunarverkefni T203Fv1_4

Skrifið forrit sem nýtir klasana `Lyklabord` og

Úttak þessa forrits er t.d.:

```
[ragnar@forritun tol203]$ javac T203Fv1_4.java
[ragnar@forritun tol203]$ java T203Fv1_4
Sláðu inn hæsta gildi fyrri tölu: 8
Sláðu inn hæsta gildi síðari tölu: 9
Veldu aðgerð: (+/-): -
3 - 2 = 1
:-)
6 - 5 = 1
:-)
7 - 7 = 0
:-)
8 - 0 = 1
:-(
4 - 2 = 2
:-)
5 - 3 = 1
:-(
3 - 9 = -6
:-)
3 - 8 = -4
:-(
1 - 6 = -5
:-)
5 - 2 = 3
:-)
Einkunn: 7
Viltu keyra aftur (j/n)? n
[ragnar@forritun tol203]$
```

SlembiInt í byggingu reiknikennsluforritys fyrir börn. Forritið á að bjóða upp á tvo möguleika á útreikningum: Samlagningu og frádrátt. Notandinn slær inn á hvaða bilum tölurnar tvær mega liggja og velur svo frádrátt eða samlagningu. Búið til aðferð reikna() sem fær send tvö gildi auk aðferðar og skilar útkomunni. Forritið birtir tíu þrautir og svarar með broskalli eða fýlukalli við hvert rétt eða rangt skipti. Að lokinni keyrslu er einkunn nemandans birt. Síðan er boðið upp á nýja keyrslu. Klasinn Lyklabord er gefinn.

Úttak þessa forritys er t.d:

```
[ragnar@forritun tol203]$ javac T203Fv1_5.java
[ragnar@forritun tol203]$ java T203Fv1_5
Sláðu inn klukkustundir: 23
Sláðu inn mínútur: 65
Sláðu inn sekúndur: 120
Réttur tími: 0:7:0
[ragnar@forritun tol203]$
```

Hann þarf að þýða og láta liggja í sama skráasafni og forritið.

Forritunarverkefni T203Fv1_5

Forritið sjálfstæða klasann Timi sem les inn klukkustundir, mínútur og sekúndur og skrifar síðan út tímanna réttan m.t.t. sólarhrings, mínútna og sekúndna. Klasinn inniheldur þrjár tilviksbreytur, h, m og s og tvær aðferðir. lesaTima() og skrifaTima(). Hægt er að rétta tímanna í annarri hvorri aðferðinni en einnig er leyfilegt að skrifa sérstaka aðferð rjettaTima() sem leiðréttir tímanna.

Úttak þessa forritys er t.d:

```
[ragnar@forritun tol203]$ javac T203Fv1_6.java
[ragnar@forritun tol203]$ java T203Fv1_6
Sláðu inn klukkustundir: 23
Sláðu inn mínútur: 59
Sláðu inn sekúndur: 50
Klst: 23 Mín: 59 Sek: 50 Tugklukkan er: 23.9972222222222224
Klst: 23 Mín: 59 Sek: 51 Tugklukkan er: 23.9975
Klst: 23 Mín: 59 Sek: 52 Tugklukkan er: 23.9977777777777777
Klst: 23 Mín: 59 Sek: 53 Tugklukkan er: 23.9980555555555556
Klst: 23 Mín: 59 Sek: 54 Tugklukkan er: 23.9983333333333335
Klst: 23 Mín: 59 Sek: 55 Tugklukkan er: 23.9986111111111111
Klst: 23 Mín: 59 Sek: 56 Tugklukkan er: 23.9988888888888889
Klst: 23 Mín: 59 Sek: 57 Tugklukkan er: 23.9991666666666667
Klst: 23 Mín: 59 Sek: 58 Tugklukkan er: 23.9994444444444446
Klst: 23 Mín: 59 Sek: 59 Tugklukkan er: 23.9997222222222222
[ragnar@forritun tol203]$
```

Forritunarverkefni T203Fv1_6

Aukið við klasann í forritunarverkefni T203Fv1_5. Bætið við hann aðferðunum `min()` og `sek()` sem skila mínútum og sekúndum. Bætið við aðferðinni `tuga_klst()` sem umbreytir tímanum yfir á tugakerfisform: hh.xxxxx. Aukið við prufuklasann svo tilkoma þessara aðferða verði sýnileg í úttakinu. Ábending: Til að umbreyta sekúndum í tugamínútur er deilt í þær með 60. Útkoman (kvótinn) er lögð við venjulegu mínúturnar og deilt aftur með 60 í summuna. Kvóti þeirrar deilingar er svo lagður við klukkustundina svo úr verður kommutala. Til að fínþússa úttakið má svo setja `DecimalFormat` á það.

Forritunarverkefni T203Fv1_7

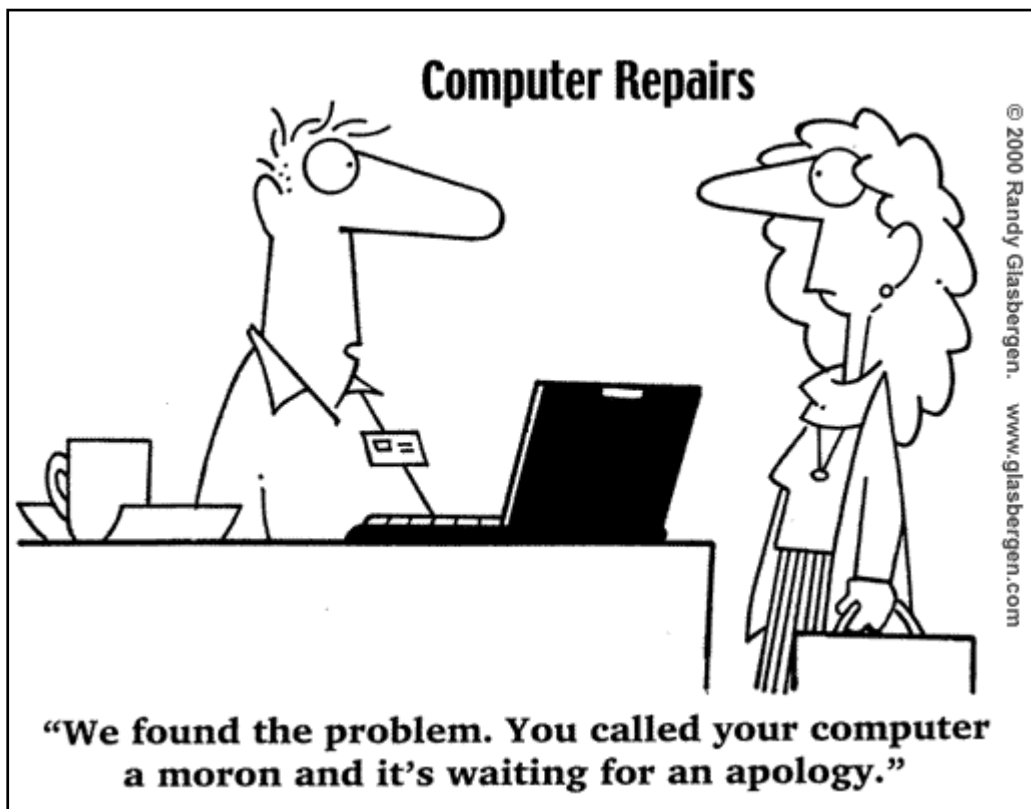
Staðsetning á jörðinni er mæld í lengd og breidd (longitude and latitude). Lengd er talin í austur eða vestur 180 gráður frá núllbaug sem liggur milli norðurskaups og suðurskaups í gegnum Greenwich á Bretlandseyjum. Breidd er talin í norður eða suður 90 gráður frá miðbaug. Hefðbundin túlkun þessara stærða er sú að í hverri lengdargráðu eru 60 mínútur. Í hverri mínútu eru 60 sekúndur. Sama gildir fyrir breiddargráðurnar. Bókstafir W eða E fyrir lengd og N eða S fyrir breidd ákveða hvort staðurinn er á vestur, austur, norður- eða suðurhveli. Hægt er að umreikna þessar stærðir í tugakerfisstærðir, þ.e. túlka breidd eða lengd með kommutölu í stað mínútna og sekúndna. Kommutölurnar eru ívið einfaldari í útreikningum og því útbreiddari hin seinni ár sérstaklega eftir tilkomu GPS tækjanna. Breidd á norðurhveli er pósítíf en negatíf á suðurhveli. Lengd á vesturhveli er negatíf en pósítíf á austurhveli. Dæmigerð staðsetning á Íslandi gæti því verið breidd: 64.006 og lengd: -20.125.

Búið til klasann `Stadur` sem finnur fjarlægð milli tveggja punkta á yfirborði jarðar. Klasinn á tvær tilviksbreytur, `lengd` og `breidd`, Forrita þarf að-

```
class Stadur { ...
  public double fjarlaegd(Stadur s2) {
    final double RADIUS = 6378.137;
    final double RADIAN = 57.295779;
    double sb1=Math.sin(breidd()/RADIAN);
    double sb2=Math.sin(s2.breidd()/RADIAN);
    double cb1=Math.cos(breidd()/RADIAN);
    double cb2=Math.cos(s2.breidd()/RADIAN);
    double c12l1=Math.cos(s2.lengd()/RADIAN -lengd()/RADIAN);
    double s = (sb1*sb2)+(cb1*cb2*c12l1);
    return RADIUS * Math.acos(s);
  }
} ...
```

Úttak er t.d:

```
[ragnar@forritun tol203]$ java T203Fv1_7
Sláðu inn breidd: 66.490
Sláðu inn lengd: -17.495
Sláðu inn breidd: 66.478
Sláðu inn lengd: -17.524
Fjarlægðin milli þeirra er: 1,856 kílómetrar
```



„Við fundum út hvað var að. Þú kallaðir tölvuna þína fábjárna og hún er að bíða eftir afsökunarbeiðni.“