Eiffel – http://www.eiffel.com

Marek Zäuram

Ülevaade

- Loodud 1986.a Bertrand Meyer
- Objektorienteeritud
- Hetkel arendab edasi Eiffel Software
- Levinumad realisatsioonid:
  - EiffelStudio, SmartEiffel, Visual Eiffel
- Mõjutatud:
  - Ada, Simula, Z
- Ise mõjutanud:
  - Sather, Ruby, Java, C#, D
- Tugev tüübikontroll

Üldisi omadusi

- Objektorienteeritud
- Integreeritud *Design by Contract* ideoloogia
- Automaatne mäluhaldus
- Pärimine
- Üldistav programmeerimine (*Generic Programing)*
- Ühtne tüübisüsteem (klassi põhine)
- Tüübikindlus (staatilised tüübid)
- Tühjuse kindlus (*void safety*)
- Agendid
- Ühekordsed rutiinid (*Once routines*)
- Märksõnade põhine süntaks
- Tõstutundetus

Eiffeli eesmärgid

- Deklaratiivsed laused mitte protseduuriline kood
- Vältida “kodeerimistrikke”
- Teha kood loetavaks

Eiffeli taust

- Rakendused ja arenduskeskkonnad
  - EiffelStudio IDE (litsentsid kas vaba tarkvara või komertsrakendus)
  - EiffelEnvision
  - Gobo Eiffel, SmartEiffel, Visual Eiffel
- Spetsifikatsioonid ja standardid
  - Keel on alates 2005.a ISO standardiseeritud
  - Gobo / SmartEiffel

Süntaks ja semantika

- Üleüldine struktuur
  - Klassid
  - Klastrid (alamklastrid)
  - Featuurid
  - Andmetüübid
  - *root* klass
  - Eiffeli on 6 käivitatavat põhi instruktsiooni

Skoobid

- Eiffeli muutujad on privaatsed ja nende muutmine on lubatud “*setter*” featuuride läbi/abil
- Klassi muutjad on kättesaadavad kõigile klassi featuuridele, featuuri muutuja ainult featuuri siseselt

Näide

```
class
    HELLO_WORLD
create
    make
feature
    make
        do
            print ("Hello, world!%N")
        end
end
```

Design by Contract

- DbC konseptsioon on tihedalt keelde integreeritud
- Eelkontrollid
- Järelkontrollid
- Klassi invariandid
- Kontrolli instruktsioonid (“*check instructions*”)

Design by Contract

```
class interface
    ACCOUNT
...
feature
  deposit (sum: INTEGER)
      require
            non_negative: sum >= 0
      ensure
            one_more_deposit: deposit_count = old
    deposit_count + 1
            updated: balance = old balance + sum
invariant
    consistent_balance: balance = all_deposits.total
end
```

Featuurid

- Kahte tüüpi featuure
  - Päringud (*Queries*)
  - Käsud (*Commands*)

Eristatakse neid

- ühtse ligipääsu ja
- päringu-käsu eraldamise

printsiipide pärast

Ülelaadimine

- Klass ühendab featuure nende nimedega
- Featuuride nimesid saab uuesti kasutada erinevates klassides

Üldistamine (Genericity)

- Tugi parametiseeritud klassidele
- Parameetrid saavad tüübi alles käivitamisel
- Võimaldab kasutada konteiner objekte
- class C [G] – G näitab, et klass on üldine
- Piiratud üldistus
  - Näiteks: **class** C [G -> COMPARABLE]

Pärimine

Klass võib:

- pärineda ühest või mitmest klassist
- ülekirjutada mõned või kõik featuurid

```
class C inherit
    A
        redefine f, g, h end
    B
        redefine u, v end
```

Ümbernimetamine

- Pärides vanemklassidest featuure, siis on võimalik neid ümber nimetada.

```
class
    C
inherit
    A
        rename
            count as capacity, item as array_item
        end
    B
    ...
end
```

Deferred classes and features

- Seisikunud klass (*deferred*)
- Seisikunud featuur
- Alamklassis ei *deferred* featuuri üledefineerimisklauslis kasutama
  - Jääb ära *redefine*
- *Deferred* klasse kasutatakse:
  - Ühiste omadustega kõrgetaseme klassides
  - Defineerimaks üldisi kõrgtasemeid (näiteks *library*'te struktuure)
  - Defineerimaks süsteemi disaini komponentide arhidektuuri (ilma kohustuseta neid realiseerida)

Korteezid (*tuples*)

- Korteezi tüüpi võib vaadata kui lihtsat klassi, mis pakub ainult *setter* protseduuri.

Näide:

```
TUPLE [name: STRING; weight: REAL;
    date: DATE]
t := ["Brigitte", 3.5, Last_night]
t.weight := t.weight + 0.5
```

Agendid

- “tegevuste mähkijaid”
- Kasutatakse
  - Graafilistes kasutajaliidestes
  - Andmestruktuurde iteratsioonides
  - Numbrilistes arvutustes

Näited:

- your_icon.click_actions.extend (agent your_routine)
- your_list.do_all (agent your_proc)

Ühekordse käivitused

- once routine
  - asendada **do** klausel **once** märksõnaga

```
shared_object: SOME_TYPE
    once
        create Result.make (args)
    end
```

- On sarnase effekti ja eesmärgiga kui teistes keeltes kasutatava ***singelton pattern***'iga

Tüübiteisendused

- Eiffelis on toetatud erinevad tüübiteisendused
- On kindel reegel:

  Tüüpe ei saa konverteerida tüüpideks millest nad otseselt või kaudselt pärinevad

- **my_string := my_date**

  ||

  create my_string.make_from_date (my_date)

Veahaldus

- Veahaldus Eiffelis on lahendatud rutiini (*routine*) tasemel.
- ***rescue*** – vea korral käivitatav kood
- ***retry*** – saab käivitada ***rescue*** põjustanud koodi uuesti

```
do
   server.connect
rescue
   if attempts < 10 then
         attempts := attempts + 1
         retry
   end
end
```

Concurrency

- Keeles ei ole sisse ehitatud veel võrgu ja lõimede tuge.
- On olemas välised *library*'id – EiffelNet ja EiffelThreads
- Samaaegsuse mudel Eiffeli jaoks on SCOOP ehk *Simple Concurrent Object-Oriented Programming* (*Design by Contract* järgi) ja ei ole hetkel keele ametlikus versioonis

Lekslilised ja süntaktilised omadused

- Eiffel ei ole tõstutundilik
  - make == Make == MAKE == maKe
- Kommentaar algab “- -” (kahe miinus märgiga) ning lõppeb reavahetusega
- Semikoolon on instruktsionide eraldaja, kuid ta on mittekohustuslik.
- Semikoolon on vajalik, kui instruktsioonid on kõik ühel real
- Tavaliselt eraldatakse arusaadavuse saavutamiseks featuurid omavahel *feature* märksõnaga.
- Eiffel teeb selget vahet avaldisel ja instruktsioonil (mitte nagu enamus loogiliste sulgudega keeli)

Stiili tava/konvensioon (*convention*)

- Klassi nimed trükitähtedega
- Featuuride nimed väikeste tähtadega
- Konstandid esimese suurtähega
- Mitmesõnalist nime soovitatakse jätkata alakriipsuga

Põimimine teiste keeltega

- Eiffel toetab “*inline*” C st Eiffeli programmi sisse on võimalik kirjutada lühikesi protseduure C keeles
- Otsest suhtlust Eiffelil ja C ei ole, vaid Eiffeli kompilaator väljastab C koodi kui vahekeelt, mille edastab C kompilaatorile optimiseerimiseks ja kompileerimiseks.
- .NET’i puhul väljastatakse CIL (Common Intermediate Language) .NET’i virtuaal masina jaoks
- SmartEiffel kompilaator väljastab Java baitkoodi

Kokkuvõtvalt erinevusi teiste programmeerimiskeeltega

- Klassisisest ülelaadmist ei ole
- Ei ole viitasid nagu C-s
- Ei ole GOTO
- Realõpp tähendab avaldise lõppu, mitte semikoolon (võib ka panna)
- “:=“ on omistamine, “=“ tähendab võrdlust