Programozási nyelvek és paradigmák

Öröklődés

Kozsik Tamás (2020)

Öröklődés

- OOP paradigma fontos eszköze
- Egy típust egy másikból származtatunk
  - Csak a különbséget kell leprogramozni
  - Kibővítás további tagokkal, felüldefiniálás
- Altípusosság
  - v.ö. Ada
- Egyszeres vagy többszörös
  - Interfészek
- Open-closed principle (SOLID)

Szülőosztály

```
class ACCOUNT
create
    make
feature
    balance: INTEGER
    id: INTEGER

    deposit ( sum: INTEGER ) ...
    withdraw( sum: INTEGER ) ...

feature {NONE}
    make( id_: INTEGER ) ...

invariant
    balance >= 0;  ...
end --class ACCOUNT
```

Gyermekosztály

```
class SAVINGS_ACCOUNT

inherit
    ACCOUNT

create
    make

feature
    interest: INTEGER assign set_interest
    set_interest( interest_: INTEGER ) ...
    pay_interest ...

invariant
    interest >= 0
end --class SAVINGS_ACCOUNT
```

Gyermekosztály – műveletek implementációja

```
set_interest( interest_: INTEGER )
    require
        interest_ >= 0
    do
        interest := interest_
    ensure
        interest = interest_
        only interest
    end -- set_interest

pay_interest
    do
        deposit(balance*interest//100)
    ensure
        balance = old (balance + balance*interest//100)
        only balance
    end -- pay_interest
```

SAVINGS_ACCOUNT inicializálása

```
class ACCOUNT
create make
feature {NONE} make( id_: INTEGER ) ...
...

class SAVINGS_ACCOUNT
inherit ACCOUNT
create make
...

local
    sa: SAVINGS_ACCOUNT
do
    create sa.make(19878238)   -- öröklött
```

Altípusosság

```
local
    a: ACCOUNT
do
    create a.make(19878238)
```

Altípusosság

```
local
    a: ACCOUNT
do
    create a.make(19878238)

    create {SAVINGS_ACCOUNT} a.make(19878238)
```

Altípusosság

```
local
    a: ACCOUNT
do
    create a.make(19878238)

    create {SAVINGS_ACCOUNT} a.make(19878238)

    !SAVINGS_ACCOUNT! a.make(19878238) -- régi szintaxis
```

Altípusosság: LSP


Barbara Liskov

- Liskov's Substitution Princliple
- Liskov-féle helyettesítési elv

Egy A típus altípusa a B (bázis-)típusnak, ha az A egyedeit használhatjuk a B egyedei helyett, anélkül, hogy ebből baj lenne.

Statikus típusellenőrzés

```
local
    a: attached ACCOUNT
do
    create {SAVINGS_ACCOUNT} a.make(19878238)
    a.deposit(1000)
    a.set_interest(10)     -- fordítási hiba
```

Dinamikus típusellenőrzés (*Object Test*)

```
local
    a: attached ACCOUNT
do
    create {SAVINGS_ACCOUNT} a.make(19878238)
    a.deposit(1000)
    if attached {SAVINGS_ACCOUNT} a as sa then
        sa.set_interest(10)
    end
```

Dinamikus típusellenőrzés – régebbi szintaxis

```
local
    a: attached ACCOUNT
do
    create {SAVINGS_ACCOUNT} a.make(19878238)
    a.deposit(1000)
    if {sa: SAVINGS_ACCOUNT} a then
        sa.set_interest(10)
    end
```

Alapértelmezett szülőtípus: ANY

```
class ACCOUNT
create make
feature
...
```

Alapértelmezett szülőtípus: ANY

```
class ACCOUNT
create make
feature
...
```

```
class ACCOUNT
inherit ANY
create make
feature
...
```

Az ANY osztály

```
class ANY
feature
    out: STRING ...
    print( obj: detachable ANY ) ...
    -- egyenlőségvizsgálat
    -- másolás
    conforms_to (other: ANY): BOOLEAN
        -- Does type of current object conform to type
        -- of `other' (as per Eiffel: The Language)?
        require
            other_not_void: other /= Void
        external
            "built_in"
        end
...
```

Láthatóság

```
feature              -- mindenki számára

feature {ANY}        -- az ANY leszármazottainak

feature {NONE}       -- a NONE leszármazottainak (senkinek)

feature {}           -- régi szintaxis (senkinek)
```

Öröklött creation procedure

```
class ACCOUNT
create make
feature {NONE} make( id_: INTEGER ) ...
...

class SAVINGS_ACCOUNT
inherit ACCOUNT
create make
feature interest: INTEGER  -- 0-ra inicializálódik
...
```

Saját creation procedure

```
class SAVINGS_ACCOUNT
inherit ACCOUNT
create make_with_interest
feature {NONE}
    make_with_interest( id_, interest_: INTEGER )
        require
            interest_ >= 0
        do
            make(id_)
            set_interest(interest_)
        ensure
            id = id_
            interest = interest_
        end
...
```

Lehet több creation procedure

```
class SAVINGS_ACCOUNT

inherit
    ACCOUNT

create
    make_with_interest,     -- itt definiált
    make                    -- öröklött

feature {NONE}

    make_with_interest( id_, interest_: INTEGER )
        ...
...
```

Hívhatjuk make-nek a leszármazottban?

```
class SAVINGS_ACCOUNT

inherit
    ACCOUNT

create
    make

feature {NONE}

    make( id_, interest_: INTEGER )
        ...
...
```

Ez így nem jó, névütközés: két make is van!

Átnevezés

```
class SAVINGS_ACCOUNT
inherit
    ACCOUNT
        rename make as make_account
    end
create
    make
feature {NONE}
    make( id_, interest_: INTEGER )
        require interest_ >= 0
        do       make_account(id_)
                 set_interest(interest_)
        ensure   id = id_
                 interest = interest_
        end
...
```

Használat átnevezés után

```
local
    a: ACCOUNT
    sa: SAVINGS_ACCOUNT
do
    create a.make(193836)
    create sa.make(19383,5)
```

Átnevezés után is maradhat creation procedure

```
class SAVINGS_ACCOUNT
inherit
    ACCOUNT
        rename make as make_account
    end
create
    make, make_account
feature {NONE}
    make( id_, interest_: INTEGER )
        require interest_ >= 0
        do       make_account(id_)
                 set_interest(interest_)
        ensure   id = id_
                 interest = interest_
        end
...
```

Ha nincs megadva creation procedure: `default_create`

```
class ANY
create
    default_create
feature  -- Initialization
    default_create
        do
        end -- default_create
...
end
```

Ha nincs megadva creation procedure: default_create

```
class ANY
create
    default_create
feature  -- Initialization
    default_create
        do
        end -- default_create
...
end

class POINT
-- inherit ANY
-- create default_create
feature
...
end
```

Ha nincs megadva creation procedure

Ha egy osztály nem ad meg *creation procedure*-t, akkor benne az `ANY` osztályból megörökölt azon művelet lesz az egyetlen *creation procedure*, amelyet az `ANY` osztályban `default_create`-nek neveznek.

Rövid creation utasítás vagy kifejezés

Ha egy osztályban az `ANY`-ből megörökölt `default_create` az (egyik) creation procedure, akkor annak meghívása elhagyható.

```
class POINT
-- inherit ANY
-- create default_create
feature
...
end
```

```
create {POINT}
create {POINT}.default_create
```

Akár átnevezés után is

```
class POINT
inherit
    ANY
        rename default_create as make
    end
-- create make
feature
...
end

create {POINT}
create {POINT}.make
```

Felüldefiniálás

```
class CIRCLE

inherit ANY
        redefine default_create
    end

-- create default_create

feature
    x, y, r: REAL

    default_create
        do
            r := 1.0     -- x := 0.0 ; y := 0.0
        end
end
```

Felüldefiniálás és átnevezés

```
class CIRCLE

inherit ANY
        rename default_create as make
        redefine make
    end

-- create make

feature
    x, y, r: REAL

    make
        do
            r := 1.0     -- x := 0.0 ; y := 0.0
        end
end
```

Megörökölt implementáció meghívása

```
class IDOPONT
inherit DATUM
        redefine make_masnap
    end
...
feature
    ora, perc: INTEGER

    make_masnap( d: attached DATUM )
        do
            Precursor(d)
            if attached {IDOPONT} d as ip then
                ora := ip.ora; perc := ip.perc
            end
        end
...
end --class IDOPONT
```

Öröklődés és osztályinvariáns?

```
class DATUM
...
feature
    ev, honap, nap: INTEGER
    ...
invariant
    honap_nem_tul_kicsi: honap >= Januar
    honap_nem_tul_nagy:  honap <= December
    nap_nem_tul_kicsi:   nap >= 1
    nap_nem_tul_nagy:    nap <= napok_szama_a_honapban
end --class DATUM
```

Öröklött és deklarált osztályinvariáns

```
class IDOPONT
inherit DATUM ...
...
feature
    ora, perc: INTEGER
invariant
    ora_nem_tul_kicsi:  ora >= 0
    ora_nem_tul_nagy:   ora < 24
    perc_nem_tul_kicsi: perc >= 0
    perc_nem_tul_nagy:  perc < 60
end --class IDOPONT
```

$$\mathrm{INV}(\texttt{IDOPONT}) \;=\; \mathrm{INV}(\texttt{DATUM}) \wedge \texttt{invariant}_{\texttt{IDOPONT}}$$

Felüldefiniálás hatása az elő- és utófeltételekre

- Öröklés – altípusposság – specializáció
- Bázistípus műveleteire szerződés
- Szerződésmódosítás: a klienst nem zavarhatja!
  - Az előfeltétel gyengíthető
  - Az utófeltétel szigorítható

Bázisosztály műveletének szerződése

```
class DATUM
...
   from_array( arr: attached ARRAY[INTEGER] )
      require
         arr.count = 3
      do ...
      ensure
         ev = arr[arr.lower]
         honap = 1 + (arr[arr.lower+1].abs - 1) \\ 12
         nap = 1 +
               (arr[arr.lower+2].abs - 1) \\
                     napok_szama_a_honapban
      end
```

Alosztály műveletének szerződése

```
class IDOPONT
inherit
   DATUM
      redefine from_array, make_masnap
   end
create
   make, make_masnap, from_array
feature
   from_array( arr: attached ARRAY[INTEGER] )
      require else
         arr.count = 5
      do ...
      ensure then
         ora = 0 or else ora = arr[arr.lower+3] \\ 24
         perc = 0 or else perc = arr[arr.lower+4] \\ 60
      end
...
```

Felüldefiniált szerződés

- `require else`
- `ensure then`

Definiálja felül r' a megörökölt r rutint. Ekkor:

$$\mathrm{PRE}(r') \;=\; \mathrm{PRE}(r) \vee \texttt{require\_else}_{r'}$$

$$\mathrm{POST}(r') \;=\; \mathrm{POST}(r) \wedge \texttt{ensure\_then}_{r'}$$

Felüldefiniált szerződés

- `require else`
- `ensure then`

Definiálja felül r' a megörökölt r rutint. Ekkor:

$$\mathrm{PRE}(r') \;=\; \mathrm{PRE}(r) \vee \texttt{require\_else}_{r'}$$

$$\mathrm{POST}(r') \;=\; \mathrm{POST}(r) \wedge \texttt{ensure\_then}_{r'}$$

Így a szerződés:

$$\{\mathrm{PRE}(r') \wedge \mathrm{INV}(C)\}\ C.r'\ \{\mathrm{POST}(r') \wedge \mathrm{INV}(C)\}$$

illetve inicializációhoz:

$$\{\mathrm{PRE}(r')\}\ C.r'\ \{\mathrm{POST}(r') \wedge \mathrm{INV}(C)\}$$

Átírva a gondolatot az *lf*-jelölésre

Legyen az eredeti, r rutin szerződése: $Q \Rightarrow \mathrm{lf}(r, R)$.

Legyen a felüldefiniált, r' rutin szerződése: $Q' \Rightarrow \mathrm{lf}(r', R')$.

Ha $Q' = Q \vee ...$, akkor $Q \Rightarrow Q'$, és ha $R' = R \wedge ...$, akkor $R' \Rightarrow R$.

Az lf monotonitása (ismert tétel) szerint, ha $R' \Rightarrow R$, akkor $\mathrm{lf}(r', R') \Rightarrow \mathrm{lf}(r', R)$.

Mindezekből:

$$Q \Rightarrow Q' \Rightarrow \mathrm{lf}(r', R') \Rightarrow \mathrm{lf}(r', R).$$

Azaz az eredeti szerződést nem rúgja fel az r', hiszen $Q \Rightarrow \mathrm{lf}(r', R)$, így r' helyettesítheti r-et.

Hiányzó szerződés

- Ha nincs `require` → `require True`
- Ha nincs `require else` → `require else False`
- Ha nincs `ensure` → `ensure True`
- Ha nincs `ensure then` → `ensure then True`

Hiányzó szerződés

- Ha nincs `require` → `require True`
- Ha nincs `require else` → `require else False`
- Ha nincs `ensure` → `ensure True`
- Ha nincs `ensure then` → `ensure then True`

Pl.

- Nincs (további) megkötés a hívásra
- Nincs (további) információ a hatásról
- Nem változik a szerződés