Programozási nyelvek és paradigmák

Objektumorientált programozás Eiffelben

Kozsik Tamás (2020)

OOP Eiffelben

- Osztályalapú
- Statikusan típusellenőrzött
- Referencia- és kifejtett típusok
- Többszörös öröklődés
- Szerződésalapú objektumelvű

OOP Eiffelben

- Osztályalapú
- Statikusan típusellenőrzött
- Referencia- és kifejtett típusok
- Többszörös öröklődés
- Szerződésalapú objektumelvű

- Feature-ök
  - Attribútumok
  - Konstansok
  - Rutinok
    - Függvények
    - Eljárások

Osztályra példa

```
class DATUM
create
    make

feature

    ev, honap, nap: INTEGER                    -- attribútum

    Januar: INTEGER = 1                        -- konstans

    szokoev: BOOLEAN                           -- függvény
        do ...  end

    make( ev_, honap_, nap_: INTEGER )         -- eljárás
        do ...  end

end --class DATUM
```

Paraméterlista

- Üres formális vagy aktuális paraméterlista: nincs ()

```
szokoev: BOOLEAN
    do
    if ev \\ 400 = 0 then Result := True
    elseif ev \\ 100 = 0 then Result := False
    elseif ev \\ 4 = 0 then Result := True
    else Result := False
    end
end -- szokoev
```

Paraméterlista

- Üres formális vagy aktuális paraméterlista: nincs ()

```
szokoev: BOOLEAN
    do
    if ev \\ 400 = 0 then Result := True
    elseif ev \\ 100 = 0 then Result := False
    elseif ev \\ 4 = 0 then Result := True
    else Result := False
    end
end -- szokoev
```

- Attribútum és paraméter nélküli függvény hasonló
- Lekérdezés, query: attribútum, konstans, függvény

```
    d: DATUM
    ...
    if d.honap = d.Februar and then d.szokoev then
    ...                              -- lusta logikai operátor
```

Attribútum inicializálása

```
class DATUM

feature      -- konstansok

    Januar:   INTEGER = 1
    Februar:  INTEGER = 2

feature      -- attribútumok

    ev:     INTEGER attribute Result := 1970 end
    honap:  INTEGER attribute Result := Januar end
    nap:    INTEGER attribute Result := 1 end

...
```

Konstruktor – creation procedure

```
class DATUM

create
    make, make_masnap

feature

    ...

    make( ev_, honap_, nap_: INTEGER )
        do ...  end

    make_masnap( d: DATUM )  -- váljunk d másnapjává
        do ...  end

end --class DATUM
```

Aktuális objektum

```
class DATUM
    ...
feature
    ...
    make_masnap( d: DATUM )   -- váljunk d másnapjává
        do ...  end

    masnap: DATUM             -- az utánunk jövő nap
        do
            create Result.make_masnap(Current)
        end -- masnap

    masnapra_lep              -- lépjünk egy napot
        do
            make_masnap(Current)
        end -- masnapra_lep

end --class DATUM
```

Dokumentációs megjegyzések

- Az számít, hova írjuk

```
make_masnap( d: DATUM )
        -- Váljunk `d` másnapjává.
    do
        ...
    end
```

Dokumentációs megjegyzések

- Az számít, hova írjuk

```
make_masnap( d: DATUM )
        -- Váljunk `d` másnapjává.
    do
        ...
    end
```

- Hivatkozások, pl. `{DATUM}`

Dokumentációs megjegyzések

- Az számít, hova írjuk

```
make_masnap( d: DATUM )
        -- Váljunk `d` másnapjává.
        do
            ...
        end
```

- Hivatkozások, pl. `{DATUM}`
- Eiffel Studio

```
note
    description: "Időpont év, hónap és nap formában."
    date: "$Date 2020-09-19 22:47:11 +0000 $"
    revision: "$Revision: 42 $"
```

Osztályinvariáns

```
class DATUM
create
    make

feature

    ev, honap, nap: INTEGER

    ...

invariant
    honap >= Januar
    honap <= December
    nap >= 1
    nap <= napok_szama_a_honapban
end --class DATUM
```

Osztályinvariáns: nevesített állítások

```
class DATUM
create
    make

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
```

Mikor felel meg egy osztály a szerződésének?

- Ha a létrehozó műveletek ($C.cp$) beállítják az invariánst
- Ha a műveletek ($C.r$) megőrzik az invariánst

Mikor felel meg egy osztály a szerződésének?

- Ha a létrehozó műveletek ($C.cp$) beállítják az invariánst
- Ha a műveletek ($C.r$) megőrzik az invariánst
- Ezeket később még finomítani fogjuk

$$\{\texttt{REQ}_{cp}\}\ cp\ \{\texttt{ENS}_{cp} \wedge \texttt{INV}_C\}$$

$$\{\texttt{REQ}_r \wedge \texttt{INV}_C\}\ r\ \{\texttt{ENS}_r \wedge \texttt{INV}_C\}$$

Példa

```
class DATUM
create
    make
feature
    ...
    make( ev_, honap_, nap_: INTEGER )
        require
            honap_ >= Januar; honap_ <= December;
            nap_ >= 1; nap_ <= 31
        do ...
        ensure
            ev = ev_ ; honap = honap_ ; nap = nap_
        end
    ...
invariant
    honap >= Januar; honap <= December;
    nap >= 1; nap <= napok_szama_a_honapban
end --class DATUM
```

Példa

```
class DATUM
create
    make
feature
    ...
    make( ev_, honap_, nap_: INTEGER )
        require
            honap_ >= Januar and honap_ <= December and
            nap_ >= 1 and nap_ <= 31
        do ...
        ensure
            ev = ev_  and honap = honap_ and nap = nap_
        end
    ...
invariant
    honap >= Januar and honap <= December and
    nap >= 1 and nap <= napok_szama_a_honapban
end --class DATUM
```

Példa

```
...
    make( ev_, honap_, nap_: INTEGER )
        require
            honap_nem_tul_kicsi: honap_ >= Januar
            honap_nem_tul_nagy:  honap_ <= December
            nap_nem_tul_kicsi:   nap_ >= 1
            nap_nem_tul_nagy:    nap_ <= 31
        do ...
        ensure
            ev = ev_ and honap = honap_ and nap = nap_
        end

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
```

Attribútum inicializálása: *estudio* invariant violation

```
class DATUM

feature  -- attribútumok

    ev:    INTEGER attribute Result := 1970 end
    honap: INTEGER attribute Result := Januar end
    nap:   INTEGER attribute Result := 1 end

...

invariant
    0 < honap ...
end
```

- Az `attribute` inicializációja nem fut le az invariáns kiértékelése előtt (*estudio* bug?)
- Creation procedure szükséges

Invariáns

Az egyik legfontosabb fogalom.

Kifejtett (expanded) osztály

```
expanded class PONT
feature

    x, y: REAL
    ...

end --class PONT
```

Kifejtett (expanded) osztály

```
expanded class PONT
feature

    x, y: REAL
    ...

end --class PONT

class SZAKASZ
feature

    a, b: PONT
    ...

end --class SZAKASZ
```

Referenciák

- Indirekt hozzáférés
- Dinamikus memória
- Az OOP egyik alapja
  - Polimorf változók
  - Dinamikus kötés
- Aliasing
- Szemétgyűjtés

Referenciák

- Indirekt hozzáférés
- Dinamikus memória
- Az OOP egyik alapja
  - Polimorf változók
  - Dinamikus kötés
- Aliasing
- Szemétgyűjtés
- Üres referencia (Eiffel: `Void`)

Tony Hoare (2009)

"I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.''

Üres referencia

- `Void` (ill. `null`, `NULL`, `nil`) dereferálása hiba
  - *feature*-re való minősített hivatkozás
- Legtöbb nyelvben: dinamikus szemantikai hiba
  - Segmentation fault
  - `NullPointerException`
- Egyes nyelvekben fordítási hiba!

Segfault versus kivétel

```
String s = null;
System.out.println( s.length() );
```

```
$ java MyClass
Exception in thread "main" java.lang.NullPointerException
    at MyClass.main(MyClass.java:4)
```

Informatívabb hibaüzenet (JDK 14)

```
String s = null;
System.out.println( s.length() );
```

```
$ java -XX:+ShowCodeDetailsInExceptionMessages MyClass
Exception in thread "main" java.lang.NullPointerException:
Cannot invoke "String.length()" because "<local1>" is null
    at MyClass.main(MyClass.java:4)
```

Statikus elemzés

Fordítóprogram mellett használható

Annotáció

`javax.annotation.NonNull` és hasonlók...

Ada

```
type String_Access is access String;
subtype Safe_String_Access is not null String_Access;
S: Safe_String_Access;    -- automatikus inicializáció
```

Ada

```
type String_Access is access String;
subtype Safe_String_Access is not null String_Access;
S: Safe_String_Access;    -- automatikus inicializáció
```

- Altípusinvariáns megsértése: dinamikus típusellenőrzési hiba
- Fordítási idejű figyelmeztetés
  (`Constraint_Error` will be raised at run time)

Ada

```
type String_Access is access String;
subtype Safe_String_Access is not null String_Access;
S: Safe_String_Access := new String'("Lovelace");
```

Ada

```
type String_Access is access String;
subtype Safe_String_Access is not null String_Access;
S: Safe_String_Access := new String'("Lovelace");

...
S := null;     -- futási hiba (warning)
```

Ada

```
type String_Access is access String;
subtype Safe_String_Access is not null String_Access;
S: Safe_String_Access := new String'("Lovelace");
Z: String_Access := null;
```

Ada

```
type String_Access is access String;
subtype Safe_String_Access is not null String_Access;
S: Safe_String_Access := new String'("Lovelace");
Z: String_Access := null;

...
S := Z;     -- futási hiba (warning)
```

Ada

```
type String_Access is access String;
subtype Safe_String_Access is not null String_Access;
S: Safe_String_Access := new String'("Lovelace");
Z: String_Access := S;     -- altípusosság
```

Ada

```
type String_Access is access String;
subtype Safe_String_Access is not null String_Access;
S: Safe_String_Access := new String'("Lovelace");
Z: String_Access := S;     -- altípusosság

...
S := Z;     -- OK
```

Eiffelben régen: szerződésekkel

```
class PERSON
create
    set_name
feature
    name: STRING

    set_name( str: STRING )
        require
            str /= Void
        do
            name := str.twin
        end
    ...
invariant
    name /= Void
end
```

Fordítási időben garantált Void-biztonság

- Cecil
- Eiffel
- Kotlin
- F#
- C# nullable value types, (non-)nullable reference types

Cecil

"The void object is used to represent a lack of a value. It is used as the result of methods or expressions that have no useful result. The system will guarantee (statically in the presence of type checking) that void is never passed as an argument to a method."

(Cecil-spec 2.1.4)

Eiffel

```
acc: !ACCOUNT
maybe_acc: ?ACCOUNT
```

Eiffel

```
acc: !ACCOUNT
maybe_acc: ?ACCOUNT

acc: attached ACCOUNT
maybe_acc: detachable ACCOUNT
```

Átmeneti és jelenlegi szintaxis

- Átmeneti
  - `ACCOUNT` $\equiv$ `detachable ACCOUNT`
  - `detachable` elhagyható
  - `attached` kiírandó

Átmeneti és jelenlegi szintaxis

- Átmeneti
  - `ACCOUNT` ≡ `detachable ACCOUNT`
  - `detachable` elhagyható
  - `attached` kiírandó
- Jelenlegi
  - `ACCOUNT` ≡ `attached ACCOUNT`
  - `attached` elhagyható
  - `detachable` kiírandó

Átmeneti és jelenlegi szintaxis

- Átmeneti
  - `ACCOUNT` ≡ `detachable ACCOUNT`
  - `detachable` elhagyható
  - `attached` kiírandó
- Jelenlegi
  - `ACCOUNT` ≡ `attached ACCOUNT`
  - `attached` elhagyható
  - `detachable` kiírandó
- Vizsga-szintaxis
  - `attached` kiírandó
  - `detachable` kiírandó

Eiffelben statikus típusellenőrzéssel

```
class PERSON

create
    set_name

feature
    name: attached STRING

    set_name( str: attached STRING )
        do
            name := str.twin
        end

    ...

end
```

Altípusosság

```
attached ACCOUNT <: detachable ACCOUNT
```

A kétfajta típus együttélése

```
if d /= Void then                    -- elvileg működni kellene
        a := d                       -- de nem...
        d.deposit(100)
end

if attached d as ad then
        a := ad
        ad.deposit(100)
end
```

Kotlin (és hasonlók)

```
var a: String = "abc"
a = null // compilation error

var b: String? = "abc" // can be set null
b = null // ok
```

Kotlin (és hasonlók)

```
var a: String = "abc"
a = null // compilation error

var b: String? = "abc" // can be set null
b = null // ok

val len1 = if (b != null) b.length else -1
```

Kotlin (és hasonlók)

```kotlin
var a: String = "abc"
a = null // compilation error

var b: String? = "abc" // can be set null
b = null // ok

val len1 = if (b != null) b.length else -1

val maybeLen: Int? = b?.length  // safe navigation op.
```

Kotlin (és hasonlók)

```kotlin
var a: String = "abc"
a = null // compilation error

var b: String? = "abc" // can be set null
b = null // ok

val len1 = if (b != null) b.length else -1

val maybeLen: Int? = b?.length  // safe navigation op.

val len2 = b?.length ?: -1      // Elvis-operator
```

Kotlin (és hasonlók)

```kotlin
var a: String = "abc"
a = null // compilation error

var b: String? = "abc" // can be set null
b = null // ok

val len1 = if (b != null) b.length else -1

val maybeLen: Int? = b?.length  // safe navigation op.

val len2 = b?.length ?: -1      // Elvis-operator

val mayFail: Int = b!!.length
```

Ajánlott gyakorlat a `null` elkerülése

- Kerüljük az üres referencia használatát extremális értékként
- `Maybe`, `Optional`/`optional` vagy `Option`/`option`

```haskell
data Maybe a = Nothing | Just a
```

Ajánlott gyakorlat a `null` elkerülése

- Kerüljük az üres referencia használatát extremális értékként
- `Maybe`, `Optional`/`optional` vagy `Option`/`option`

```haskell
data Maybe a = Nothing | Just a
```

```java
Optional<String> getParameter( String name )
```

A probléma részben megmarad

... bár jobban láthatóvá válik

- Futási hibát kiváltó művelet (`fromJust`, `get`)

```
Optional<String> getParameter( String name ) ...
void log( String param ) ...
...
log( getParameter("version").get() );
```

A probléma részben megmarad

... bár jobban láthatóvá válik

- Futási hibát kiváltó művelet (`fromJust`, `get`)

```
Optional<String> getParameter( String name ) ...
void log( String param ) ...
...
log( getParameter("version").get() );
```

- Sikertelen mintaillesztés, parciális függvény

```
double (Just str) = str ++ str
```

Monadikus stílus

```
def getParameter( name: String ) : Option[String]
...
val upper = for {
                    name <- getParameter("author")
                    trimmed <- Some(name.trim)
                    upper <- Some(trimmed.toUpperCase)
                          if trimmed.length != 0
              } yield upper
println(upper.getOrElse(""))
```

(Forrás: Scala API doc)

Funkcionális stílus

```
val name:Option[String] = getParameter("author")
val upper = name    map { _.trim }
                  filter { _.length != 0 }
                     map { _.toUpperCase }
println(upper.getOrElse(""))
```

(Forrás: Scala API doc)

Swift

- Intenzíven használt `Option` típus
- Szintaktikus támogatás

```
var optionalInteger: Int?
var optionalInteger: Optional<Int>
```

Swift

- Intenzíven használt `Option` típus
- Szintaktikus támogatás

```
var optionalInteger: Int?
var optionalInteger: Optional<Int>

optionalInteger = 42
optionalInteger = nil
```

Swift

- Intenzíven használt `Option` típus
- Szintaktikus támogatás

```
var optionalInteger: Int?
var optionalInteger: Optional<Int>

optionalInteger = 42
optionalInteger = nil

print( optionalInteger! )
```

Swift

- Intenzíven használt `Option` típus
- Szintaktikus támogatás

```
var optionalInteger: Int?
var optionalInteger: Optional<Int>

optionalInteger = 42
optionalInteger = nil

print( optionalInteger! )

print( optionalInteger ?? "" )
```

Swift

- Intenzíven használt `Option` típus
- Szintaktikus támogatás

```
var optionalInteger: Int?
var optionalInteger: Optional<Int>

optionalInteger = 42
optionalInteger = nil

print( optionalInteger! )

print( optionalInteger ?? "" )

var dangerous: Int! = 42
var x: Int = 2*dangerous          // nem kell !
```