Metódusok

Metódusok

- minden alprogram metódus
  - nincsenek függvények/eljárások
- nem kötelező osztályba ágyaznunk a főprogramot
  - a Ruby automatikusan megteszi ezt
  - a főprogram is objektum (Object típusú)
- a főprogramban definiált metódusok erre az objektumra hajtódnak végre
- transzparens
  - teljesen objektum-orientált
  - de továbbra is természetesen használható

Metódusok definiálása

- def method_name(arg1, arg2, arg3)
  ...
  end
  - metódusok nevét kisbetűvel kezdjük
    - konvenció
    - Ruby engedné a nagybetűt is, de problémákat okozhat
- nevesített argumentumok
  - paraméterátadás nem egy tömbön keresztül történik
- def method_name
  ...
  end
- nem kötelező kitenni a zárójelet, ha nincs paraméter

methods.rb

Metódusok definiálása

- metódus neve
  - végződhet ? karakterrel
  - végződhet ! karakterrel
- visszatérési érték az utolsó kifejezés értéke
  - return kulcsszóval explicit jelezhető
- a formális paraméterekhez a metódus törzsére lokális változó tartozik
  - nem érhető el kívűlről
  - élettartama kicsit hosszabb is lehet
    - mark-and-sweep gc
- a Ruby futási időben ellenőrzi az aktuális és a formális paraméterek számát
  - hiba, ha ezek nem egyenlőek

default-args.rb
default-args-log.rb

Metódusok definiálása

- def method_name(arg1 = “def”, arg2 = “def2”)
  ...
  end
- default értékek megadhatók minden paraméter számára
- ha egy paraméternek default értéket adunk, akkor az összes utána következőnek is kötelező
  - ellenkező esetben nem lehetne az elhagyott paramétereket kezelni
- default argumentum értéke nem csak konstans lehet
  - lehet kifejezés
  - lehet egy másik argumentum értéke

modulo.rb

Változó argumentumszám

- ha nem fix a paraméterek száma
- def method_name(arg1, arg2, *rest)
  ...
  end
- a * jelöli, hogy a maradék argumentumok kerüljenek összegyűjtésre
  - rest egy tömb lesz (Array)
- csak egy lehet
  - az első mindenképpen benyelné az összes elemet
  - szintaktikai hiba is

Iterátorok implementálása

- iterátor
  - olyan Ruby metódus, amely képes blokkot fogadni
  - valamilyen (absztrakt) adatszerkezet elemeinek
    - ... felsorolására
    - ... feldolgozására
  - egyéb esetek is elképzelhetők
    - pl. File.open
- egy metódus legfeljebb egy blokkot kaphat paraméterként
  - alternatív lehetőségekkel ez megkerülhető, de blokk csak egy lehet

Iterátorok implementálása

- az iterátor eldöntheti, hogy kapott-e blokkot
  - block_given?
- meghívhatja a blokkot
  - yield
  - paraméterül csak a blokknak szánt paramétereket kell adni
  - argumentumszám-ellenőrzés futási időben
    - nem egyezés nem vezet hibához
    - ha kevesebb, a maradék nil lesz
    - ha több
      - 1 argumentuma van a blokknak, akkor a blokk egy Array objektumot kap + egy warning
      - 0 vagy több, mint egy argumentuma van a blokknak, akkor a többi argumentum figyelmen kívül marad

clock.rb

Iterátorok implementálása

- def my_iterator(arg1, arg2)
  yield(arg1, 2)
  ...
  end
- az argumentumlistában nem jelenik meg a blokk
  - block_given?
- a yield paramétereit kapja meg a blokk paraméterként

proc.rb

Proc objektumok

- kód blokkokat kezelhetünk objektumként
  - closure
- Proc.new
  - új Proc objektum létrehozása
  - a paraméterül adott blokkot alakítja Proc objektummá
  - ha nincs paraméterül adott blokk, akkor a hívás helyén az aktuális blokk
- később a Proc objektum meghívható
  - prc.call(...)
  - prc[...]
- megtudható az aritása
  - prc.arity
- Kernel#proc

block-param.rb

Blokk paraméterek

- blokk paraméterek kezelhetőek
  - Proc objektummá konvertálhatóak
- ennek két módja van
  1. Proc.new hívás anélkül, hogy paraméterül adnánk blokkot
  2. utolsó paraméter elé & jelet kell tenni a formális paraméterek listájában
     - def method_name(arg1, &block_arg)
       ...
       end
     - ha nincs blokk, akkor nil

Metódushívások

- receiver.method_name(arg1, arg2) { ... }
  - receiver: az üzenet fogadója
  - a metódus neve
  - argumentumok
  - opcionális blokk paraméter
- zárójel az argumentumok körül elhagyható
  - legtöbb esetben célszerű kitenni
- receiver
  - lehet osztály, vagy modul is
    - File.size("filename")
    - Process.fork
- receiver
  - ha elmarad, akkor a self lesz

Osztályok

Osztályok

- Ruby-ban minden objektum, mindennek van típusa
- az osztály is objektum
  - típusa: Class
  - MetaClass nincs
- a new az osztály, mint objektum metódusa
  - nem kulcsszó
  - MyClass.new(args)
- nincs többszörös öröklődés
  - helyette: modulok
    - jobb, mint a Java interface-k
- egy osztály definíciója mindig nyitott marad

Osztályok definíciója

- osztályok neve konstans
  - nagybetűvel kell kezdődnie
  - ha nem, akkor a fordító hibát jelez
- class MyClass
  ...
  end
- osztály törzsében
  - metódus definíciók
  - attribútum definíciók
  - alias definíciók

Konstruktorok

- Ruby a new metódusban helyet foglal, inicializál
  - ezt nem célszerű felüldefiniálni
    - Class#new
  - meghívja az osztály initialize metódusát, ha definiált
    - ezt kell definiálni
    - példánymetódus
- class MyClass
  def initialize(arg1, arg2)
  ...
  end
  end
- nem szükséges visszatérési értéket megadni

Destruktorok

- Ruby nem tartalmazza ezt a feature-t
  - bonyolítaná és lassítaná a gc működését
- ObjectSpace.define_finalizer
  - egy objektumhoz definiálható egy finalizer
  - a finalizerben az objektum már nem elérhető, csak egy object ID-t kap
  - szándékos a definíció furcsa helye
- ObjectSpace.define_finalizer(object) { |id|
  ...
  }

Destruktorok

- helyette az iterátorok nyújtotta előnyök használhatóak ki
- például a File.open
- osztálymetódus
- a megnyitott filet reprezentáló objektum csak a paraméterül adott blokkban létezik
- File.open { |file|
  ...
  }
  - megnyitja a file-t
  - a file objektumot átadja a blokknak
  - kezeli, ha fellép valamilyen kivétel
  - végül mindenképpen lezárja a file-t

Objektum változók

- az objektumhoz tartozó változók neve @ jellel kezdődik
  - @attribute
- minden metódusban használt egyéb változó lokális
- def my_method
  @attr = 2
  end
- nem inicializált értékű objektum változóra való hivatkozás nem fatális hiba
  - értéke nil
  - warning!
- az osztályon kívűlről nem érhető el sehogy

Osztály változók

- osztály változók @@ jellel kezdődnek
  - @@my_class_variable
- inicializálatlan osztály változóra való hivatkozás hiba
  - futási idejű hiba
- az osztályon kívűlről nem elérhető

Osztály konstansok

- láthatóak az osztályon kívűlről is
- :: minősítővel érhetők el
- class MyClass
  MyConstant = 1
  end
  - MyClass::MyConstant

Osztály metódusok

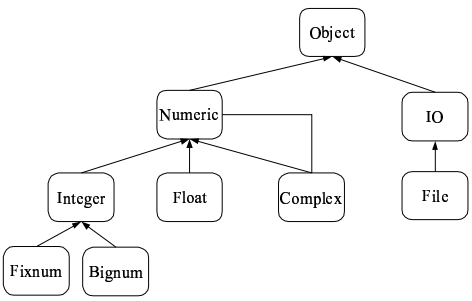
- egy osztály definícióján belül osztály metódusok is megadhatók
- class MyClass
  def my_instance_method
  ...
  end

  def MyClass.my_class_method
  ...
  end
  end
- Class.new
- File.delete(filename)

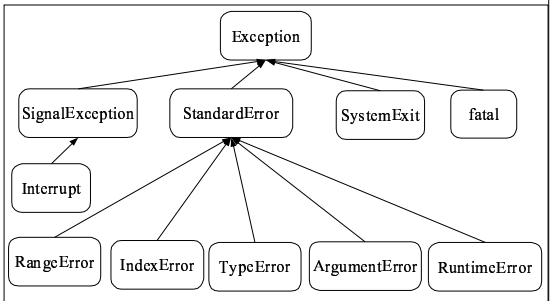
Öröklődés

- class MyChildClass < MyParentClass
  ...
  end
- nincs többszörös öröklődés
  - helyette modulok
- attribútumok is öröklődnek
  - elérhetők a leszármazott osztályokban
- nem szabályozható az öröklődés
  - nincs public, private, stb. öröklődés
- MyChildClass.superclass
  - megadja a szülő osztályt
    - Class
  - csak osztályra működik, a Class osztály metódusa

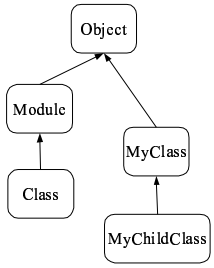
Ruby osztályhierarchia



Ruby osztályhierarchia



Ruby osztályhierarchia



- minden osztály objektum
  - minden osztály a Class egy példánya
- minden osztálynak őse az Object
- nem jelzett Ruby osztályok szülője az Object
  - Array
  - Hash
  - Proc
  - Range
  - String
  - stb.

super

- super(...)
  - a szülőben lévő metódus meghívása
  - nem csak konstruktorban
  - a hívás helyének megfelelő metódus meghívása
- super
  - ha nem adunk meg paramétert, akkor a megkapott paramétereket adja tovább

Attribútumok

- az objektum belső állapotát attribútumok írják le
- Ruby attribútumok neve @ jellel kezdődik
  - az osztályon kívülről nem érhetőek el
- getter metódus kell, ha kívűlről lekérdezhetővé akarjuk tenni
  - minden attribútumra külön
- attr_reader :attribute
  - getter metódus automatikus generálása
  - kompakt, olvasható
- attr_reader :attr1, :attr2
  - több is felsorolható

Attribútumok

- attribútumok kívűlről történő módosításához setter metódust kell definiálni
- def setAttr(newValue)
  ...
  end
  - kevéssé konzisztens
- def attr=(newValue)
  ...
  end
  - ezután használható:
    - object.attr = newValue
- attr_writer :attribute
  - a setter metódus automatikus generálása
  - nem definiálja a gettert automatikusan

Attribútumok

- attr :attribute, writeable
  - writeable logikai
  - ha hamis, akkor csak getter definiálódik
  - ha igaz, akkor getter és setter is
- attr_accessor :attribute
  - attr :attribute, true
  - definiál getter és setter metódust is
- Module osztály privát példánymetódusai!
  - attr_reader
  - attr_writer
  - attr
  - attr_accessor

Hozzáférés szabályozás

- metódusokhoz való hozzáférés szabályozható
  - attribútumok soha nem elérhetőek
  - public, protected, private
    - eltér a szokásostól
- public
  - mindenki által hozzáférhető kívűlről is
- protected
  - csak a definiáló osztály illetve a leszármazottak *objektumai* hívhatják
- private
  - csak a definiáló osztály illetve a leszármazottak hívhatják

Hozzáférés szabályozás

- protected
  - csak az osztály metódusain belül hívható
  - lehet receiver-e
- private
  - nem adható meg receiver
  - emiatt csak az adott objektumon belül hívható

Hozzáférés szabályozás

- alapértelmezés szerint public
- class MyClass

```
    def method_this
    end

    def method_that
    end
    protected :method_this, :method_that
end
```

  - fel kell sorolni a metódusokat szimbólumokkal

access-control.rb

Hozzáférés szabályozás

- class MyClass

```
protected
    def method_this
    end

    def method_that
    end
end
```

- mindkét metódus protected lesz
  - minden következő metódus definíció hozzáférhetőségének alapértelmezett értékét módosítja
- Module osztály privát példánymetódusai

Nyílt osztálydefiníciók

- Ruby nyelven minden osztály definíciója nyílt
- bármikor hozzáaadhatok új szolgáltatásokat
- több definíció tartalma merge-elődik
- class MyClass

```
    def method_1
    end
end
...
class MyClass
    def method_2
    end
end
```


extend-array.rb
nil.rb

Nyílt osztálydefiníciók

- öröklődéssel ez nem lenne teljesen megvalósítható
- már meglevő osztályok módosítása nagy lehetőség
  - példa a Set osztály
  - ha használjuk, akkor módosítja például az Array osztályt
    - hozzáaad egy to_set metódust

Fagyasztás

- freeze metódus
  - Object#freeze
- egy objektum állapota befagyasztható
  - visszavonhatatlan
  - az objektum attribútumai nem módosíthatók
  - ha osztály, akkor a metódusai nem módosíthatók, nem definiálható új, és nem törölhető meglévő
- s = “dog”
  s.freeze
  s[0] = “f”
  - in `[]=': can't modify frozen string (TypeError)
- vizsgálható egy objektum fagyasztott állapota
  - s.frozen?

Alias

- alias newName oldName
  - alias hozható létre
  - paraméterek lehetnek nevek, vagy szimbúlumok
    - alias new_method old_method
    - alias :new_method :old_method
- alias készíthető
  - metódusokra
    - operátorokra
  - globális változókra
  - reguláris kifejezések változóira
    - $&, $`, $', $+

Alias

- gyakran csak egy funkció több néven nevezéséhez használt
  - Enumerable#map, Enumerable#collect
  - Hash
    - each_pair visszaad minden elemet (kulcs, érték) párok formájában
    - Enumerable miatt szükséges egy each metódus
    - alias each each_pair
- Kombinálva azzal, hogy minden osztály definíciója nyílt, nagy lehetőségeket rejt
  - felüldefiniálható a Class.new, alias-val megőrizve az eredeti new-t
  - stb.

Modulok

Modulok

- mit nyújtanak?
  - névtér elkülönítés
  - többszörös öröklődés hiányának enyhítése
- nem osztály
  - nem példányosítható
  - funkcionalitást hordozhat
- metódus definíciókat tartalmaz
  - tartalmazhat hivatkozásokat más metódusokra
  - fordítási időben nem ellenőrzött
    - megj: Ruby-ban máshol sem
  - tartalmazhat osztálydefiníciókat
    - ilyenkor a névtér funkciója érdekes

Modulok

- egy osztályba keverhető a modul tartalma
  - module mixin
- a modul hívhat olyan metódusokat, amiket a modul nem definiál
  - ezek azok a metódusok, amiket a modul vár a kliens osztálytól
  - futási idejű hibát okoz ezek hiánya
- module kulcsszóval definiálható egy modul
- include kulcsszóval lehet egy osztályt “mixelni”
  - ezt egy modul is megteheti!

dumper.rb

Modulok

```
module MyModule
    def module_method
        method_in_the_class
    end
end
...
class MyClass
    include MyModule

    def method_in_the_class
    end
end
```


extend-enumerable.rb

Nyílt moduldefiníciók

- modulok definíciója is nyílt
- hozzáadhatok új metódusokat
  - jelentkezni fog azokban az osztályokban, amelyek használják a modult
  - teljesen dinamikusan

Modul attribútumok

- egy modul kezelhet attribútumokat
  - ezeknek csak egy osztályon belül van értelme!

```
module MyModule
    attr_reader :attr
    def method
        @attr = 1
    end
end
```

- ekkor az osztálynak is lesz attr attribútuma, és hozzá egy getter metódus
- nem szerencsés
  - a modul alapvetően funkcionalitást tartalmazzon, adattagokat ne

Ruby standard modulok

- Ruby számos standard modult tartalmaz
- mixin-re szánt modulok
  - Comparable
  - Enumerable
- funkcionalitást csoportosító modulok
  - névtérként funkcionál
  - FileTest
    - File osztály include-olja
  - GC
  - Kernel
  - Marshal
  - Math
  - ObjectSpace
  - Process

comparable-arrays.rb

Comparable

- függ a <=> metódustól
  - az osztály, ami include-olja, definiál egy ilyen metódust
- a <=> operátor segítségével definiál
  - <, >, <=, >=, ==
  - between?
- emlékeztető:
  - Ruby nem definiál != metódust
    - a != b mindig !(a==b) formára alakul
  - == metódus definiálásával a fenti is használható lesz
- használja például:
  - String
  - Numeric

Enumerable

- szükséges egy each metódus
  - iterátor, amely felsorolja az elemeket
- szolgáltatásai
  - collect, map
  - detect, find
  - each_with_index
  - select, find_all
  - grep
  - include?, member?
  - reject
  - to_a, entries
- használja például:
  - Array
  - Hash

Enumerable

- collect, map
  - transzformáció elvégzése a bemeneti adatokon
  - blokkot vár paraméterként
- grep
  - reguláris kifejezés alapján szűr
  - ha van blokk paraméter, akkor minden, a regexpre illeszkedő elemre meghívja azt
- select, find_all
  - blokkot vár paraméterként
  - ha a blokk igazat ad vissza, akkor az elem megfelel a szűrésnek
- reject
  - ! select
- stb.

Enumerable

- ha a kliens osztály definiál <=> operátort, akkor használható továbbá
  - min
  - max
  - sort

Kivételek

Kivételek

- Ruby-ban a kivételek kezelése is OO
- kivételkezelés során objektumokat dobunk
  - mi mást is dobhatnánk?
- begin...end blokk közötti kód által dobott kivételeket figyeljük
- rescue
  - kivételkezelő ágak
- else
  - ha nem történt kivétel
- ensure
  - mindenképp lefut
  - más nyelvekben általában: finally
- nincs kivételkezelési kényszer, mint pl. Java-ban
  - nem specifikálható, hogy mit dobhat

Kivételek

- az elkapott kivétel a $! változóban található
- begin
  ...
  rescue
  puts “ERROR CAUGHT: #$!”
  end
  - ha nincs megadva semmi, akkor a rescue elkap minden kivételt
- begin
  ...
  rescue => e
  puts “ERROR CAUGHT: #{e}”
  end
  - elkap mindent, és a kivétel az 'e' változóba kerül

Kivételek

- begin
  ...
  rescue RuntimeError
  puts “ERROR CAUGHT: #$!”
  end
  - RuntimeError és leszármazottait kezeli
- begin
  ...
  rescue RuntimeError => e
  puts “ERROR CAUGHT: #{e}”
  end
  - RuntimeError és leszármazottai, hiba az e változóban

Kivételek

- begin
  ...
  rescue RuntimeError, NameError => e
  ...
  end
  - RuntimeError, illetve NameError leszármazottai
- begin
  ...
  rescue
  puts “ERROR”
  else
  puts “All OK”
  end

Kivételek

- begin
  file = File.open(filename)
  ...
  rescue
  ...
  ensure
  file.close() unless file.closed?
  end
  - mindenképp végrehajtódik

Kivételek

- retry utasítás
  - Eiffel-ből ismerős
  - a kivételvédett blokk újrakezdése
  - csak rescue ágban hívható
  - ensure ág csak egyszer hajtódik végre
- begin
  ...
  rescue
  ...
  retry
  end

Kivételek

- raise
  - kivételt dobhatunk mi is
  - nem dobható minden
  - Exception-t kell dobni
- raise “Something's wrong”
  - RuntimeError típusú kivétel lesz ebből
- raise ArgumentError, “Parameter must be String”
  - ArgumentError típusú kivétel keletkezik, beállítható hozzá egy üzenet
- raise ArgumentError, “No param”, caller
  - stack backtrace is megadható

Reguláris kifejezések

Reguláris kifejezések

- természetesen ez is teljesen OO
- minden reguláris kifejezés a Regexp osztály egy példánya
- minden mintaillesztés eredménye egy MatchData objektum
- re = /\d{2}/
  - Regexp típusú objektum
- re.match(“hello”)
  - ha nem illeszkedik, akkor a match visszatérési értéke nil
  - ha illeszkedik, akkor egy MatchData objektum

Reguláris kifejezések

- /\d+/
  - az elhatároló módosítható
- %r{/usr/bin/(.*)}
  - %r{}
  - elhatároló sokféle lehet
- Regexp.compile
  - Regexp.new("/usr/bin/(.*)")
    - /\/usr\/bin\/(.*)/
    - Regexp.compile egy szinoníma
- módosítók
  - i: case insensitive
    - /Hello/i
    - /Hello/i.match("hello")
      - #<MatchData:0x403108c8>

Reguláris kifejezések

- /alma$/.match("alma\nfa")
  - $ mindenképp illeszkedik a sor végére
  - ^ mindenképp illeszkedik a sor végére
  - Perl-ben ez a szemantika csak az 'm' módosítóval él
- m: multiline
  - a . illeszkedik az újsor karakterre is
  - Perl-ben ezt a funkciót az 's' módosító adja!
  - /alma.narancs/.match("alma\nnarancs")
    - nil
  - /alma.narancs/m.match("alma\nnarancs")
    - #<MatchData:0x402d12f0>

Reguláris kifejezések

- x: extended legibility
- re = /
  (\w+) # hónap
  \
  (\d{2}) # hónap napja
  \
  (\d{2}) # óra
  :
  (\d{2}) # perc
  /x
- re.match("Sat Apr 10 11:18:06 CEST 2004")
  - #<MatchData:0x402e10b0>

Reguláris kifejezések

- o: substitute once
  - a helyettesítéseket csak egyszer végzi el a regexpben
- /hello #{name}/
  - minden illesztés alkalmával elvégzi a helyettesítést
  - name változó aktuális értékével
- /hello #{name}/o
  - csak az első illesztés alkalmával cserél
- olyan alkalom számít, ahol a regexp, mint literál szerepel, és új Regexp objektumot generál

MatchData

- s = "Sat Apr 10 11:18:06 CEST 2004"
- md = /(\w+) (\d{2}) (\d{2}):(\d{2})/.match(s)
- md[0]
  - a teljes illesztett sztring
  - "Apr 10 11:18"
- md[i], ha i > 0
  - az i. illesztett csoport
  - md[1] == "Apr"
  - md[2] == "10"
  - md[3] == "11"
  - md[4] == "18"

MatchData

- md.begin(1)
  - a *sztring* hányadik karakterén kezdődik az 1. számú csoport?
  - 4
- md.end(1)
  - a *sztring* hányadik karaktere az, amely az első csoportot követi?
  - 7
- md.to_a
  - md[i] elemek egy tömbben
  - ["Apr 10 11:18", "Apr", "10", "11", "18"]

Perl-szerű szintaxis

- a Perl inspirálta szintaxis használható Ruby-ban
  - a regexpek kezelése továbbra is OO marad
  - wrapper réteg az OO réteg fölé
- =~
  - Regexp#=~
  - String#=~
- “Hello world!” =~ /hello/i
  - String#=~
  - visszatérési érték az illeszkedés pozíciója
    - 0
- /hello/i =~ “Hello world!”
  - Regexp#=~
  - visszatérési érték az illeszkedés pozíciója
    - 0

Perl-szerű szintaxis

- mintaillesztés eredménye
  - $~
    - MatchData
    - utolsó illesztés eredménye
  - $1, $2, ..., $9
    - ezek a $~ változótól függenek
  - $&
    - match: az illeszkedő sztringrészlet
  - $`, $'
    - pre-, postmatch
  - $+
    - az utolsó illesztett csoport

Cserék

- String#sub
  - egyszeri cserék
- String#sub(pattern, replacement)
  - pattern isa Regexp
  - replacement
    - egy sztring, amire cserélni kell
    - a mintában lévő csoportok a \1, \2, ..., \9 hivatkozásokkal érthetők el
    -
  - “hello 21”.sub(/(\d+)/, ':\1:')
    - “hello :21:”
    - “:\1:” mást jelent!
- String#sub! a fogadót (receiver) módosítja

Cserék

- String#sub(pattern) { |s| ... }
  - a blokk paraméterként kapja az illeszkedő részsztringet
  - a blokk visszatérési értékével helyettesíti az illesztett részsztringet
  - $1, $2, ..., $9 megfelelően be van állítva
- “hello#21”.sub(/#(\d+)/) { |s|

```
  a = " "
  a[0] = $1.hex
  a
}
```

  - “hello!”

Cserék

- String#gsub(pattern, replacement)
  - minden előfordulást helyettesít
- String#gsub!(pattern, replacement)
  - minden előfordulást helyettesít és módosítja a fogadó (receiver) objektumot
- String#sub(pattern) { |o| ... }
  - a blokk visszatérési értéke lesz a helyettesített sztring
- String#sub!(pattern) { |o| ... }
  - a blokk visszatérési értéke lesz a helyettesített sztring, és a fogadó (receiver) módosul

Ami kimaradt

- I/O
- catch and throw
- szálak
- RD
- Kernel modul metódusai
- osztályokról információ lekérdezése
  - példánymetódusok listája
  - osztálymetódusok listája
  - konstansok listája
- példányhoz tartozó metódusok (singleton methods)

Itt a vége fuss el véle...