© Kiskapu Kft. Minden jog fenntartva

A Mix-in osztálytípus

Bemutatjuk a Pythonban alkalmazható bekeveredő osztályokat.

A bekeverő programozás olyan programozási stílus, amellyel a szolgáltatásokat osztályokban készítik el, majd pedig más osztályokba belekeverik azokat. Első hallásra talán egyszerű öröklésnek hangzik, de a Mix-inek néhány dologban eltérnek a hagyományos osztályalapú szerkezettől. Gyakran megesik, hogy a Mix-in egyetlen adott osztálynak sem „elsődleges” őse, nem számít, milyen osztályokat használtak fel, ugyanis több, az osztályrangsorban elszórtan elhelyezkedő osztályból épül fel és dinamikusan, futásidőben lesz bevezetve.
Több érv is szól a Mix-inek használata mellett. A létező osztályokat új szolgáltatásokkal bővítik, anélkül, hogy szerkeszteni, karbantartani vagy egybeolvasztani kellene a forráskódjukat. Segítenek a projekt összetevőit (mint a tartomány-keretrendszer, a felület-keretrendszer) elkülönítetten tárolni. Megkönnyítik az új osztályok létrehozását, mert szolgáltatások tárházát nyújtják, melyeket aztán igény szerint lehet összeválogatni. Túllépnek az öröklődés egy korlátján, mivel egy osztály megváltoztatása után is nyugodtan lehet használni az eredeti osztályt a program más részeiben.
A Mix-inek használata nem különleges szakmai lehetősége a Pythonnak, de előnyei kedvéért megéri, hogy vessünk rá egy pillantást.
A Python eszményi nyelv a bekeverő programozáshoz, mivel támogatja a többszörös öröklést, a teljes körű dinamikus kötést (binding) és lehetővé teszi az osztályok dinamikus módosítását.
Mielőtt azonban fejest ugranánk a Pythonba, hadd említsem meg, hogy a Mix-inek nem éppen új keletű dolgok. Ezen a néven elsőként az egykori Talingent projekt megismerésekor láttam, mely a Pink operációs rendszerről és a CommonPoint alkalmazáskörnyezetről volt ismert.
Mivel a C++ nem támogatja a második nyelvi szolgáltatást (teljesen dinamikus kötés), valamint a harmadikat sem (futásidejű dinamikus változtatásokat), nem lennék meglepve, ha ez a megközelítés nem válna be annyira, mint ahogy azt megalkotói eredetileg remélték.

1. lista Mix-in dinamikus telepítése

```
[gt]python
Python 2.0 (#1, Oct 16 2000, 18:10:03)
[GCC 2.95.2 19991024 (release)] on linux2
>>>
>>> class Friendly:
...     def hello(self):
...             print ´Hello´
...
>>> class Person:
...     pass
...
>>> p = Person()
>>> p.hello()
Traceback (most recent call last):
  File "lt;stdin[gt]", line 1, in ?
AttributeError: 'Person' instance has no
➥attribute ´hello´
```

Ismerek egy másik példát is a Mix-in stílusú programozásra, igaz, eltérő név alatt. Az Objective-C rendelkezik egy „kategória” nevű, ötletes nyelvi elemmel, amely lehetővé teszi, hogy tagfüggvényeket adjunk hozzá vagy cseréljünk le meglévő osztályokban akár anélkül, hogy forráskódjukhoz hozzányúlnánk.
Ez nagyszerű lehetőség a létező rendszerosztályok javítására és képességeik bővítésére. Ráadásul a könyvtárak dinamikus betöltésének képességével együtt, a kategóriák igen hatásosak lehetnek az alkalmazások kódszerkezetének fejlesztésében és a kód méretének csökkentésében.
A grapevine arról tájékoztat, hogy a Symbolics’ objektumközpontú Flavors rendszere valószínűleg a Mix-inek legkorábbi komoly megjelenése. A tervezőket egy híres fagylaltozó (Steve’s Ice Cream Parlor) ihlette meg, ahol a vásárlók valamilyen alapjégkrém-változattal kezdtek (vanília, csokoládé stb.), majd ehhez rakhatták hozzá a kiegészítőket ízlés szerint (mogyorót, csokoládéreszeléket, tejszínhabot stb.). A Symbolic rendszerében a nagy önálló osztályok voltak az ízek, míg a kisebb, más osztályok kiegészítésére tervezett osztályok voltak a Mix-inek. További írások olvashatók hálózaton
➲ http://www.kirkrader.com/examples/cpp/mixin.htm oldalon.

A Python képességei

Miközben tiszteletünket fejezzük ki a néhai Taligent, a tetszhalott Objective-C és legendás Symbolics iránt, kezdjünk el kutatni azok közt a képességek közt, melyek a Mix-in programozás terén oly nagyszerű nyelvvé emelik a Pythont.
Először is, a Python támogatja a többszörös öröklést. Ez a Pythonban azt jelenti, hogy egy osztály akár több osztálytól is örökölhet:

```
class Server(Object, Configurable):
    pass
```

Ezenkívül a Python támogatja a teljesen dinamikus kötést. Amikor `obj.load(filename)` típusú üzenetet küldünk egy objektumnak, a Python a megadott üzenet neve és obj osztály öröklési szabályai alapján futásidőben határozza meg, milyen tagfüggvényt kell meghívni. Ez a tulajdonság pontosan úgy működik, ahogy elvárná az ember, és megjegyezni is könnyű. Akkor is működik, ha az osztály öröklési rendje vagy a tagfüggvény megváltozik a futás során.
A többszörös öröklés alkalmazása során a keresési sorrendet mindig szem előtt kell tartani. A keresési sorrend az alaposztályok közt balról jobbra értelmezett, egy adott alaposztályon belül pedig végigvezet a szülőosztályok során. Ha Mix-inekkel dolgozunk, mindig figyeljünk az esetleg előforduló névütközésekre. Ha jól elkülönített Mix-in osztályokat készítünk, és átgondoltan elnevezett tagfüggvényeket használunk, általában elkerülhetjük a meglepetéseket. Végül a Python támogatja az osztályrendszer dinamikus változtatását is.
A legtöbb Python-elem, legyen szó listáról, szótárról, osztályról vagy példányról (instance), rendelkezik elérhető tulajdonságkészlettel.
A Python osztályoknak van egy __bases__ nevű tulajdonsága, amely az adott osztály alaposztályának tárhelye (tuple). A Python tervezésével összhangban, ezt az értéket futásidőben megváltoztathatjuk.
Az első listában látható Python interaktív parancsértelmezőben készí-

© Kiskapu Kft. Minden jog fenntartva

2. lista Tetszőleges változó elérésére

```
class NamedValueError(KeyError):
   pass


class _NoDefault:
   pass


class NamedValueAccessible:

   def valueForKey(self, key,
default=_NoDefault):

      ''' A feltételezett kulcs a 'foo'.
          Ez az eljárás az alábbi értékekkel
          tér vissza a következő sorrendben:
             1. Eljárások a nem-eljárások
                előtt
             2. Nyilvános tulajdonságok,
                a nem nyilvánosak előtt

          Még pontosabban, ez az eljárás a
          következők egyikével tér vissza:
             * self.foo()
             * self._foo()
             * self.foo
             * self._foo

          ...vagy az alapértelmezéssel, ha van
          ilyen, egyébként kivétel.
      '''
      assert key
      klass    = self.__class__
      underKey = '_' + key
      attr     = None
      method   = getattr(klass, key, None)
      if not method:
         method = getattr(klass, underKey,
                  None)
         if not method:
            attr = getattr(self, key, None)
            if not attr:
                attr = getattr(self,
                       underKey, None)
                if not attr:
                    if default!=_NoDefault:
                       return default
                    else:
                       raise
                       NamedValueError, key
      if method:
         return method(self)
      if attr:
         return attr
```

tett példában két osztályt készítünk, majd megváltoztatjuk az öröklési viszonyokat. Az *1. listában* látható People (személy) nem túl barátságos, változtassuk hát meg. Sőt, változtassunk meg minden személyt, egyszer és mindenkorra megoldva a gondot:

```
>>> Person.__bases__ += (Friendly,)
>>> p.hello()
Hello
```

A fenti első utasítás a Person alaposztályát változtatja meg. A += használatával (az egyszerű „=” használata helyett) elkerülhetjük, hogy véletlenül töröljünk létező alaposztályokat, különösképpen, ha a kód későbbi változatában a Person más osztályoktól is örököl. A mókás kinézetű `(Friendly,)` kifejezés tuple-t határoz meg, amit általában zárójelek közé kell tenni. Igen ám, de míg a Python az (x,y) jelölést önműködően két elem tuple-jének veszi, az (x) jelölést zárójeles kifejezésként ismeri fel. A vessző hozzáadása a tuple-ként való felismerést kényszeríti ki.

MySQLdb sormutató Mix-in

A Mix-inek készítésének magától értetődő módja, ha a tervezés közben, modul készítésekor adjuk meg azokat. Az egyik legkedveltebb külső fejlesztésű Python modul, a MySQLdb, pontosan ezt teszi. A Python az adatbázis-elérésekhez egy DB API nevű szabványos csatolófelületet ➲ http://www.python.org/topics/database/) határoz meg. *Andy Dustman* MySQLdb modulja ezt a csatolófelületet használja, így ezen keresztül a Python-programozók kapcsolatokat hozhatnak létre, illetve lekérdezéseket küldhetnek a MySQL kiszolgálónak. Megtalálható a ➲ http://dustman.net/andy/python/MySQLdb/ címen. A MySQLdb három fontosabb szolgáltatást nyújt az általa készített sormutató objektumhoz. Ha szükséges, figyelmeztetéseket küld, igény szerint tárolja az eredménykészletet az ügyféloldalon, vagy használja őket a kiszolgálóoldalon, végül az eredményt tuple-ként (például rögzített listákként) vagy szótárként adja vissza. Ahelyett, hogy mindezt egyetlen hatalmas osztályba tömörítené, a MySQLdb mindegyikükhöz egy-egy Mix-in osztályt rendel:

```
class CursorWarningMixIn:
class CursorStoreResultMixIn:
class CursorUseResultMixIn
class CursorTupleRowsMixIn:
class CursorDictRowsMixIn(CursorTupleRowsMixIn):
```

Ne feledjük, a Mix-inek osztályok, és mint ilyenek, kihasználhatják az öröklődés minden előnyét, amint azt a CursorTupleRowsMixIn-től öröklő CursorDictRowsMixIn esetében meg is figyelhetjük. A fenti Mix-inek egyike sem állhat önmagában: a BaseCursor osztály bármely típusú sormutatóhoz szükséges alapszolgáltatásokat biztosítja. A fenti Mix-inek és a BaseCursor összekapcsolásával a figyelmeztetések (warnings), a tároló- és eredménytípusok (result types) minden elképzelhető változatát előállíthatjuk (összesen nyolcfélét). Amikor létrehozzuk az adatbázis-kapcsolatot, a kívánt sormutatótípust adhatjuk át:

```
conn = MySQLdb.connection
➥(cursorclass=MySQLdb.DictCursor)
```

A Mix-inek nemcsak a MySQLdb program megalkotásakor segítettek, hanem könnyebben bővíthetővé is teszik, mert a saját testreszabott sormutató osztályainkhoz egyszerűen kiválasztható képességeket fűzhetünk. Figyeljük meg, hogy az osztálynevek végéhez a Mix-in szócskát szoktuk illeszteni, ami az adott osztály természetéről árulkodik. Másik általános szokás, hogy az „-able” vagy „-ible” végződést fűzzük a névhez, amint az a Configurable vagy NamedValueAccessible esetén is megfigyelhető.

NamedValueAccessible

Használjuk fel példánkban ez utóbbit. A NamedValueAccessible Mix-In a valueForKey( ) tagfüggvény képességével ruház fel minden

4. lista Mix-in függvényünk végső változata

```
import types

def MixIn(pyClass, mixInClass, makeAncestor=0):
    if makeAncestor:
      if mixInClass not in pyClass.__bases__:
         pyClass.__bases__ = (mixInClass,)
➥+ pyClass.__bases__
    else:
      #Rekurzívan letiltja a mix-in elődosztályt,
      #hogy támogassa az öröklődést
      baseClasses = list(mixInClass.__bases__)
      baseClasses.reverse()
      for baseClass in baseClasses:
         MixIn(pyClass, baseClass)

      # beillesztjük a mix-in eljárást az
      # osztályba
      for name in dir(mixInClass):
            if not name.startswith('__'):
      #átugorjuk a titkos tagokat
            member = getattr(mixInClass, name)
            if type(member) is
types.MethodType:
                member = member.im_func
            setattr(pyClass, name, member)
```

osztályt, amelybe bekeverjük. Az obj.valueForKey(name) hívás esetén a tagfüggvény a következők egyikét fogja visszaadni:

- obj.name( )
- obj._name( )
- obj.name
- obj._name

Másként fogalmazva: a valueForKey( ) tagfüggvényt vagy tulajdonságot keres – legyen az belső vagy nyilvános –, hogy az adott kulcshoz tartozó értéket visszaadhassa. E tagfüggvény tervezése azt tükrözi, hogy a Python-objektumok esetében gyakran tulajdonságokon és tagfüggvényeken keresztül is elérhető ugyanaz az adat. A megvalósítást lásd a *2. listában*.
Egyik hasznos tulajdonsága e Mix-innek, hogy általános kódot tartalmaz a naplózáshoz (lásd a *3. listát*). Egyszerűen csak új kulcsokat kell a logColumns( ) tagfüggvényhez adni, és a napló, illetve az azt készítő kód (amely a logEntry() részben található) megváltoztatása nélkül kibővíthető. Talán még érdekesebb, hogy a logColumns( ) akár egy egyszerű beállításfájlból is kiolvashatja a lista mezőit.
A rugalmas valueForKey( ) tagfüggvény jóvoltából, a tranzakcióobjektum kötetlenül képes a kért értékeket tagfüggvényeken vagy tulajdonságokon keresztül szolgáltatni. A Mix-inek rugalmas készítése nagymértékben növeli használhatóságukat, és akár művészetnek is nevezhetjük megalkotásukat.

Utólagos keverés

Eddig példákat láthattunk arra, miképpen lehet a Mix-ineket az osztályok készítése közben használni. Csakhogy a Python azt is lehetővé teszi számunkra, hogy a szolgáltatásokat futásidőben keverjük be! Legegyszerűbb megoldás, ha megváltoztatjuk az adott osztály alaposztályait, mint azt korábban már leírtuk. Egy függvény felhasználásával lehetőség nyílik arra, hogy ezt a műveletet átlátszóan végezzük, és később – amennyiben szükséges – továbbfejlesszük:

```
def MixIn(pyClass, mixInClass):
    pyClass.__bases__ += mixInClass
```

Nézzünk egy olyan helyzetet, amely nyilvánvalóvá teszi a MixIn( ) módszer létjogosultságát. Egy internetes alkalmazás készítésekor, a tartományosztályok és a felületosztályok szétválasztása általában jó ötlet. A tartományosztályok képviselik az adott alkalmazás fogalmait, adatait és műveleteit, valamint függetlenek az operációs rendszertől, a felhasználói felülettől, az adatbázistól stb. Néhány szerző üzleti objektumként (business object), modellobjektumként vagy lényegi objektumként hivatkozik a tartományobjektumokra.
A tartomány és a felület elválasztása több okból is hasznos lehet. A célkitűzés általában két kulcsterületre bontható, amelyek nagyjából függetlenek egymástól. Mi a nehézség tárgya? Illetve, hogyan jelenítsük ezt meg? Új felületeket készíthetünk a tartományobjektumok megváltoztatása vagy újraírása nélkül is. Tulajdonképpen akár több felületet is készíthetünk.
Egy regénykiadó rendszer tartomány objektumai lehetnének például a Regény, a Szerző és a Székhely. Ezek az osztályok létfontosságú tulajdonságokat (például cím, szöveg, név, e-mail stb.) és sok műveletet hordoznak (elment, betölt, kiadás stb).
Egy ilyen rendszerhez az egyik felület lehet egy honlap, ami lehetővé teszi a felhasználók számára, hogy regényeket készítsenek, szerkesszenek, töröljenek és kiadjanak. Mikor ilyen honlapot fejlesztünk, hasznos lenne, ha a tartományosztályaink, mint például a Regény, rendelkeznének olyan tagfüggvényekkel, mint a renderView( ) vagy a renderForm( ), amelyek megjelenítik, vagy szerkesztésre felkínálják a regényt HTML formában.
Mix-inek használatával ezeket a szolgáltatásokat is a tartományosztályokon kívül fejleszthetjük.

```
StoryInterface osztály:
    def renderView(self):
        # A regény kiírása HTML formátumban
        pass
    def renderForm(self):
        # A regény megjelenítése szerkesztésre
        pass
```

A honlapot működtető kódba pedig keverjük be őket:

```
from MixIn import MixIn
from Domain.Story import Story
MixIn(Story, StoryInterface)
```

Ha úgy döntünk, hogy grafikus felületet készítünk a kiadórendszerhez, nem kell a teljes HTML szerkezetet magunkkal cipelnünk (vagy fordítva). A tartományosztályok a szükséges adatokra és műveletekre összpontosítanak, hogy a GUI fejlesztésekor pontosan azt kapjuk, amire szükségünk van.
Mondhatná valaki, hogy miért ne készíthetnénk egy új osztályt, hogy összehozzuk a kettőt:

```
class StoryInterface:
    ...
from Domain.Story import Story
class Story(Story, StoryInterface): pass
```

Vagy mondhatná azt, hogy a StoryInterface miért ne lehetne a Regény alosztálya, melynek megszerezi a képességeit. Vizsgáljuk

© Kiskapu Kft. Minden jog fenntartva

© Kiskapu Kft. Minden jog fenntartva

meg azt az esetet, ahol a Regény osztálynak már eleve voltak más tartományalosztályai:

```
class Story: ...
class Editorial(Story): ...
class Feature(Story): ...
class Column(Story): ...
```

A Regény létező alosztályaira semmilyen módon nincsen hatással az új Regény osztály vagy alosztály készítése. A dinamikus Mix-in készítése azonban egyaránt hat a további osztályokra is. Ez az, ami miatt az állandó megoldás sokszor nem működik a gyakorlatban, és ami a dinamikus megoldást nemcsak egyszerűen ügyes, hanem egyenesen szükséges megoldássá teszi.
Továbbá, vizsgáljuk meg azt az esetet, amikor a Regény objektumok a kód olyan részében készülnek, ahol a Regény közvetlenül kódolva van. Habár gyenge megoldás, mégis meglehetősen általános gyakorlat. Ebben az esetben a Regény alosztályok készítésének semmiféle hatása nem lenne azon a kódon, ami figyelmen kívül hagyja őket.
Egy figyelmeztetés a dinamikus Mix-inek használata kapcsán: megváltoztathatják a létező objektumok viselkedését (mivel megváltoztatják az objektumok osztályát is). Ez pedig váratlan eredményekre vezethet, mivel a legtöbb osztályt nem az ilyen típusú változások szem előtt tartásával tervezik meg. A dinamikus Mix-inek használatának biztonságos módja, hogy még az objektumok készítése előtt, az alkalmazás indításakor telepítjük őket.

A MixIn( ) továbbfejlesztett változatai

Az első fejlesztés, amit a MixIn( ) függvényhez adhatunk, hogy ellenőrizzük, ne keverjük be kétszer ugyanazt az osztályt:

```
def MixIn(pyClass, mixInClass):
    if mixInClass not in pyClass.__bases__
        pyClass.__bases__ += (mixInClass,)
```

A gyakorlatban többnyire arra van szükségünk, hogy a Mix-in tagfüggvények kapják az elsőbbséget, s akár helyettesítsék is az örökölt tagfüggvényeket amennyiben szükséges. A függvény következő változata a keverék osztályt az alaposztályok sorozatának elejére helyezi, de lehetőséget nyújt arra is, hogy egy tetszés szerint megadható változóval megváltoztassuk ezt a viselkedést:

```
def MixIn(pyClass, mixInClass, makeLast=0):
  if mixInClass not in pyClass.__bases__
    if makeLast:
      pyClass.__bases__ += (mixInClass,)
    else:
      pyClass.__bases__ = (mixInClass,) +
      ➥pyClass.__bases__
```

A Python hívások olvashatóságáért a jelzőkhöz ajánlott a nevük használata:

```
# nem túl olvasható:
MixIn(Story, StoryInterface, 1)
# Lényegesen jobb:
MixIn(Story, StoryInterface, makeLast=1)
```

Ez az új változat még mindig nem teszi lehetővé, hogy az éppen használt osztály tagfüggvényeit a Mix-in tagfüggvényei felülírják, ezért a Mix-in tagfüggvényeket tulajdonképpen az osztályba kell telepíteni. Szerencsére a Python elég dinamikus ahhoz, hogy ezt megtehessük. A *4. lista* bemutatja MixIn( ) függvényünk végső változatának forráskódját. Alapértelmezés szerint a Mix-in tagfüggvényeit közvetlenül a célosztályba telepíti, miközben figyel a Mix-in alaposztályainak átvitelére is. A hívás kinézete:

MixIn (Story, StoryInterface)

A kiegészítő `makeAncestor=1` érték segítségével olyan MixIn( ) függvényt kapunk, ami az eredeti elvet követve működik (azaz a Mix-int a célosztály szülőjévé teszi). Azt a képességet, ami a Mix-int az alaposztályok végére helyezi, eltávolítottuk, mivel a gyakorlatban szinte soha nem volt rá szükségünk.
A függvény még kifinomultabb változata visszaadhatná (esetleg választhatóan) a két osztály között ütköző tagfüggvények listáját, vagy ha átfedést talál, meghívhatna egy kivételt ezzel a listával.

Önműködő Mix-in telepítés

Az utólagos keverés komoly használata esetén a MixIn( ) függvény hívása többszörözötté válhat. Például egy GUI alkalmazásnak minden létező tartományosztályhoz lehet keveréke, így valami ilyesmi hívás szükséges mindegyikhez:

```
from Domain.User import User
MixIn(User, UserMixIn)
```

Az egyik megoldás a Mix-inek és a célosztály név alapján történő összekapcsolása és telepítése az alkalmazás indításakor. Például az összes Mix-int elnevezhetjük közvetlenül az után az osztály után, amit módosít és betehetjük a MixIns/ könyvtárba.

Egyéb lehetőségek

Bár mókás dolog egyre kifinomultabb MixIn( ) változatokat fejleszteni, a legfontosabb dolog mégis az, hogy a programfejlesztésben fel tudjuk őket használni. Ösztönzésképpen álljon itt néhány ötlet:

- Az osztály kibővítheti saját magát egy Mix-innel, miután olvas a beállításfájlból. Például egy webkiszolgáló osztály bekeverhet egy Threading vagy Forking osztályt, a beállítástól függően, miként lett beállítva.
- A program bővítményeket is használhat: olyan programcsomagokat, melyeket futásidőben keres meg és tölt be a program, hogy kiterjessze a képességeit. Azok, akik bővítményeket készítenek, kihasználhatják a MixIn( ) függvényt, ezáltal gyarapítva az eredeti programosztályokat.

Összefoglalás

A bekeveredők nagyszerű eszközök a részekre bontott fejlesztéskor és a már létező osztályok kiegészítésére anélkül, hogy azok forráskódjához hozzá kellene nyúlni. Ez elősegíti az olyan programozói elvek betartását is, mint a tartomány és a felület szétválasztása, dinamikus beállítás és a bővítmények használata. Ha Pythont használunk, mindig gondoljuk át, milyen Mix-inek tehetnék még hatékonyabbá a programunkat.

Chuck Esterbrook
(echuck@mindspring.com) tanácsadó, író és vállalkozó, lelkes Python- és Webwarefelhasználó.

Kapcsolódó címek

- ➲ http://www.python.org/
- ➲ http://webware.sourceforge.net/