Programozási nyelvek és paradigmák

Generikus programozás – típusparaméterek

Kozsik Tamás (2020)

Típussal való paraméterezés

- parametrikus polimorfizmus
- generic konstrukció
- Eiffel: csak osztály lehet generikus
  (rutin nem, v.ö. Java, C#, C++, Ada)

Univerzális kvantálás

```
class STACK[T]
 ...
```

Jelentése: minden `T` típusra definiáljuk a `STACK` osztályt úgy, hogy...

Egzisztenciális kvantálás

Javában, Haskellben stb. lehet egzisztenciális kvantálást írni

```
List<?> someList = new ArrayList<Integer>();
...
someList = new LinkedList<String>();
```

Létezik egy típus, amivel a `List` paraméterezve van, de nem tudjuk, melyik ez a típus.

Korlátozott parametrikus polimorfizmus

- Bounded quantification / constrained genericity
- A típusparaméterekre megkötések adhatók
- OOP-ben: bázisosztály (upper bound)

```
class HASHTABLE [ K -> HASHABLE, V ]
...
```

- Haskell: type class (overloading, ad-hoc polimorfizmus)
- Ada: típusosztályba tartozás, (típus)művelet típusa

Korlátozott parametrikus polimorfizmus Adában

```
generic
    type Item is private;
    type Index is (<>);
    type Vector is array (Index range <>) of Item;
    with function Op( A, B: Item ) return Item;
    Start: in Item;
function Fold ( V: Vector ) return Item;
```

Korlátozott parametrikus polimorfizmus Haskellben

```
class Eq a => Ord a where
  compare :: a -> a -> Ordering
  (<) :: a -> a -> Bool
  (<=) :: a -> a -> Bool
  (>) :: a -> a -> Bool
  (>=) :: a -> a -> Bool
  max :: a -> a -> a
  min :: a -> a -> a

instance Ord Integer
instance Ord a => Ord (Maybe a)

sort :: Ord a => [a] -> [a]
```

Korlátozott parametrikus polimorfizmus Javában

Lehet adni lower boundot is

```
public static <T extends Comparable<? super T>>
void sort(List<T> list)
```

Generic paraméterek varianciája

- Javában invariáns
  - `List<A>` $<:$ `List<B>` csak ha `A`$=$`B` vagy `B`$=?$
    - `List<Object> lst = new LinkedList<String>();` hibás
    - `List<Object> lst = new LinkedList<Object>();` ok
    - `List<?> lst = new LinkedList<Integer>();` ok
  - biztonságos statikus típushelyesség szempontjából
  - tömb típusoknál kovariáns: `String[]` $<:$ `Object[]`
  - tömböknél dinamikus típusellenőrzés kell a helyességhez
    - `Object[] os = new String[10];` ok
    - `os[0] = "hi";` ok
    - `os[1] = new Object();` fordítás ok, futás kivétel
- Scalában?
- Eiffelben?

Generic paraméterek varianciája

- Javában invariáns
- Scalában szabályozható (ko-, kontra- és invariáns)
  - biztonságos statikus típushelyesség szempontjából
  - kovariáns típusparaméter csak pozitív,
  - kontravariáns típusparaméter csak negatív,
  - invariáns pedig akármilyen pozíción állhat

```
trait C[+P,-M] {
     def f( m: M ) : P
}

val c: C[AnyRef,String] =
   new C[String,AnyRef]{ def f(m: AnyRef) = "hi " + m }

c.f("Martin")
```

- Eiffelben?

Generic paraméterek varianciája

- Javában invariáns
- Scalában szabályozható (ko-, kontra- és invariáns)
- Eiffelben (általában) a típusparaméter is kovariáns
  - ha `class STACK[T] ...`, akkor
  - `STACK[INTEGER]` $<:$ `STACK[ANY]`
  - statikusan nem biztonságos típusrendszer!
- kivéve, ha `frozen` a típusparaméter, mert akkor invariáns
  - ha `class STACK[frozen T] ...`, akkor
  - `STACK[INTEGER]` $<:$ `STACK[ANY]` nem áll fenn

Megkötések típusparaméterekre az Eiffelben

```
class VECTOR [ ... ] ...
```

esetén

```
VECTOR[G -> ADDABLE]          -- ADDABLE features usable on G
HASHTABLE[K -> HASHABLE, V]          -- more type parameters
VECTOR[G -> {ADDABLE, HASHABLE}]        -- more constraints
VECTOR[G -> ADDABLE create make end]   -- instantiation ok
VECTOR[?G -> ADDABLE]   -- self-initializing type required
VECTOR[G -> ADDABLE rename add as plus end]
        -- provide a new feature name to be used in VECTOR
VECTOR[G -> {A rename v as w, B}]    -- avoid name-clashes
VECTOR[frozen G -> ADDABLE]    -- invariant generic param.

VECTOR[frozen ?G -> {A rename v as w, B} create make end]
```

Tuple típusok Eiffelben

- Beépített típus
- Olyan, mint a generic, de akárhány típusparamétere lehet

```
t2: TUPLE[INTEGER,INTEGER]
t2 := [1,3]
t3: TUPLE[i,j: INTEGER; r:REAL]
t3 := [1, 3, 0.0]
t2.item(0)   -- ANY típusú
t3.i         -- INTEGER típusú
```

Altípusosság Tuple típusokra

- Kovariancia
- Olyan, mint funkcionális nyelvekben lehetne

`TUPLE[INTEGER,STRING]` $<:$ `TUPLE[INTEGER,ANY]` $<:$
`TUPLE[INTEGER]` $<:$ `TUPLE[ANY]` $<:$ `TUPLE`

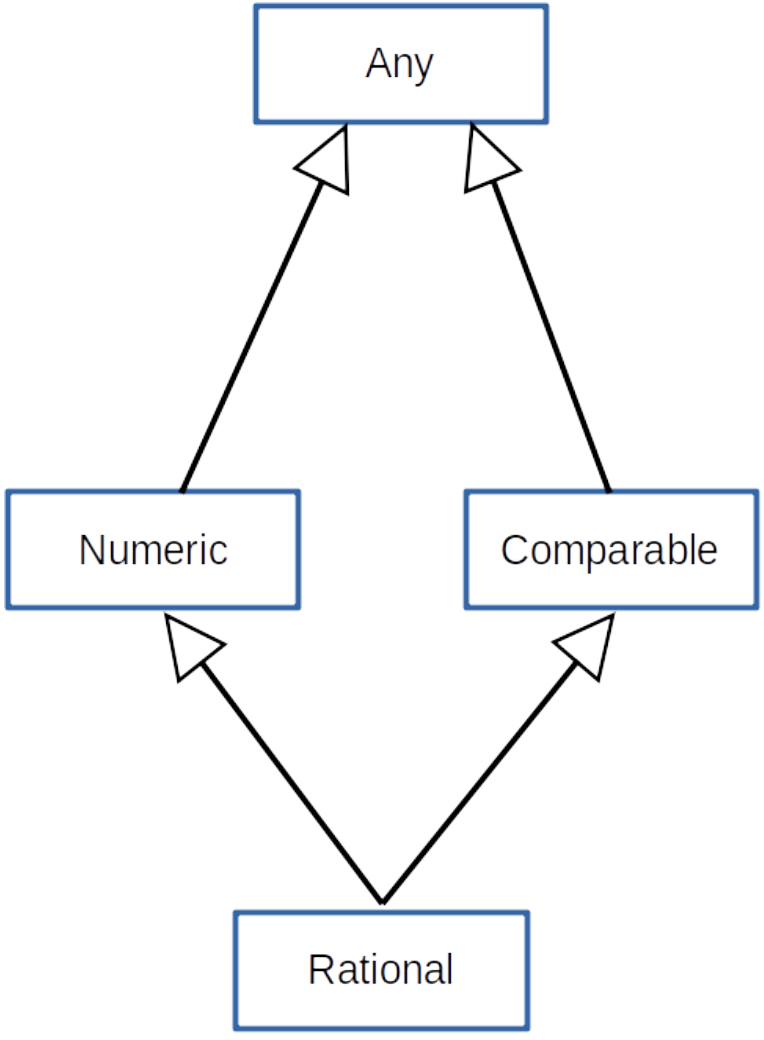
Mixin: többszörös öröklődés helyett

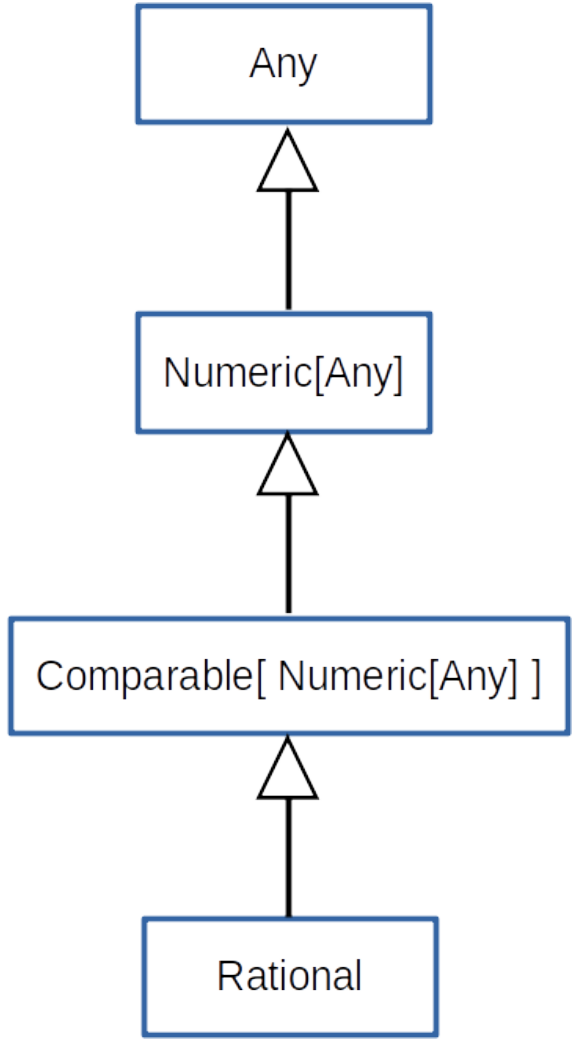
- típusparaméterből való öröklődés lehetősége
- Ada, C++ megengedi, az Eiffel (és pl. a Java) nem
- linearizálja az öröklést, így oldja fel a problémákat

```
class Numeric[T] inherit T ... end
class Comparable[T] inherit T ... end
class Rational inherit Comparable[Numeric[Any]] ... end
```

(nincs ilyen Eiffelben!)

Mixin versus többszörös öröklődés





Mixin C++-ban

```
template <typename T> class    Numeric : public T { ... };
template <typename T> class Comparable : public T { ... };
class Base {};
class Rational : Comparable< Numeric<Base> > { ... };
```

Korlátozás típusparaméterre a mixinben

```
class Rectangle ... end
class Bordered[ T <- Rectangle ] inherit T ... end
class   Filled[ T <- Rectangle ] inherit T ... end
class  Rounded[ T <- Bordered  ] inherit T ... end

class MyRect
inherit Rounded[ Filled[ Bordered[Rectangle] ] ]
...
end
```

(nincs ilyen Eiffelben!)

Mixin az Adában

```
package Rectangles is
    type Rectangle is tagged record
        X, Y, W, H: Integer;
    end record;
end Rectangles;
```

Mixin az Adában

```
package Rectangles is
    type Rectangle is tagged record
        X, Y, W, H: Integer;
    end record;
end Rectangles;


with Rectangles, Colors;
use Rectangles, Colors;
generic
        type R is new Rectangle with private;
package Bordering is
        type Bordered is new R with record
                Border_Width: Integer;
        end record;
end;
```

Scala trait: hasonló linearizálás

```
trait Numeric { ... }
trait Comparable { ... }
class Rational extends Numeric with Comparable { ... }
```

- ismételt öröklésnél csak egyszer (a “legjobboldalibbat”)
- a `super` dinamikusan köt

F-bounded polymorphism

- A típusparaméterre adott korlát függ saját magától

```
public static <T extends Comparable<T>>
T max( List<T> list );
```

F-bounded polymorphism

- A típusparaméterre adott korlát függ saját magától

```
public static <T extends Comparable<T>>
T max( List<T> list );

public static <T extends Object & Comparable<? super T>>
T max(Collection<? extends T> coll)
```