Funkcprog 2

Kollekciók Scalában

A parametrikus típussal ellátott osztályt generic osztálynak nevezzük, szintaxisa: [„típus”] nem lehet neki típust megadni, mert csak futásidőben dől el a típusa.

List

A típus neve List van belőle beépített adatszerkezete (scala.collection.immutable). Generic típus, a lista elemeinek típusát kapja paraméterül, ha a lista üres NIL objektum. Egy nem üres lista két részből áll fej és farok.

szintaxis:

-val list: List[Int] = ::(1, ::(4, ::(2. Nil)))

-val list = 1 :: 4 :. 2 :: Nil

-val list = List(1, 4, 2)

Műveletek:

-filter és map: minden List[t] osztályban szerepelnek, a filter kiválasztja azokat az elemeket amelyekre a feltétel igaz, míg a map alkalmaz egy függvényt minden elemre létrehozva egy új listát az elemekkel.

list.filter(x => x > 1)

list.map(x => x * 2)

-reduce: bináris műveletet alkalmaz a lista elemeire, redukálva azokat egyetlen értékké

-foldLeft: egy balról jobbra haladó művelet

-match: lehet mintaillleszteni nem üres listára különválasztva a fejet és a farkat.

-monád: monádok esetén definiálva van a flatmap, flatten és map

-flatMap: egy függvényt vár amelyet alkalmazza a monádra

-flatten: kicsomagolja a monádot és egy monádon belüli monádot lapít össze.

-map: egy függvényt alkalmaz a megadott elemre, létrehoz egy új listát a tranaszformált értékekkel.

Monád:

List Vector és Set egy monád hiszen generikus, mert van neki flatMapje, van megfelelő konstruktor és teljesülnek az alábbi összefüggések:

-egy monád egy generikus osztály (a generikusság lehetővé teszi, hogy különböző típusú értékekkel dolgozzunk, de a monád struktúrája megmaradjon)

-unit: rendelkezik egy függvénnyel, amely létrehoz egy monádot egy adott értékből

-kell legyen egy flatMap metódusa, a függvény amit a flatMap kap argomentumként, az általánosan egy függvény, amely egy értéket transzformál egy másik monáddá. Ezután a flatMap összefűzi ezeket a monádokat és egy új monádot eredményez.

Monád axiómák:

A monád axiómák azok az alapvető tulajdonságok amelyeknek teljesülniük kell minden monádban:

1. Az egységelem (unit element) balról semleges elem a monádban:

   -A unit(x) függvény alkalmazása a flatMap műveletre nem változtatja meg az eredményt

   -pl.: unit(x).flatMap(f) = f(x)

2. A jobbegység egységelemet tartalamzó transzformáció azonos legyen az eredeti monáddal:

   -ha egy értéket transzformálunk, majd hozzáadjuk az egységelemet akkor az eredmény ugyanaz legyen, mintha az eredeti értéket használtuk volna

   -pl.: m.flatMap(unit) = m

3. Monádikus kompozíció asszociatív:
   -a műveletek sorrendje nem számít
   -egymás után alkalmazva egy műveletet három monádon, az eredmény ugyanaz lesz, függetlenül a műveletek sorrendjétől
   -pl.: m.flatMap(f).flatMap(g) = m.flatMap(x => f(x).flatMap(g))

Tehát a List generikus tipus és monádként viselkedik mivel rendelkezik flatMap, flatten és map metódusokkal és megfelelő konstruktorral.

(pl.: List.apply)

-flatMap: lehetővé teszi az értékeke transzformációját és a monádok egymás utáni végrehajtását

-flatten: egy speciális flatMap, amely a monádokat egyszerűsíti egy sik listává alakítja

-map: egy unáris tagfüggvény, amely lehetővé teszi a lista elemeinek transzformációját egyenként

Filter:

Egy listaművelet pl.: legyűjteni pozitív értékeket egy Int listában. A predikátumok olyan függvények, amelyek az objektumból pl.: int Booleanba képződnek. Ezeket a predikátumokat lehet átadni a filter vagy más hasonló műveletknek, ezzel egységesíti a szűrési feltételeket.

Fold/reduce:

A fold és reduce olyan műveletek, amelyek a lista elemein hajtanak végre egy kétváltozós műveletet és az eredményt adják vissza.

fold:

-két változata van foldLeft és foldRight

-foldLeft: az aggregálást a lista elejéről kezdi és az aktuális értéket mindig az operáció második operandusaként használja, A lista elemeit jobbról adja hozzá az aktuális értékhez.

-foldRight: az aggregálás a lista végéről indul és mindig a lista aktuális elemét használja a művelet második operandusaként, majd az eredményt hozzáadja az aktuális értékhez.

reduce:

-egy speciális esete a foldnak, ahol a lista elemeinek típusa azonos a kiszámított eredmény típusával

-reduceLeft: az aggregálás mindig a lista elejéről indul és az értéket mindig az operáció második operandusaként használja

-reduceRight: az aggregálás a lista végéről indul és mindig a lista aktuális elemét használja az operáció második operandusaként

kezdőérték és asszociativitás:

-ha a lista üres a fold és reduce műveletek problémát okozhatnak, kivéve, ha megadjuk a kezdőértéket

-az asszociativitás azt jelenti hogy a kétváltozós műveletek sorrendje nem számít

-a fold és reduce műveletek az asszociatív művelettel végeznek aggregálást és ezáltal biztosítja a helyes működést

kezdőérték nélküli kezelés:

-ha a lista üres és nem adunk kezdőértéket a fold-nál hibát kapunk, míg a reduce-nál egy üres lista esetén az eredmény típusának meg kell egyeznie az üres lista elemeinek típusával, azaz használható kezdőérték nélkül is

foreach:

A List[T] osztály rendelkezik egy szintén generikus foreach metódussal, ami kap egy függvényt, ami inputként a lista elemeinek típusát várja, outputként pedig tetszőleges típust tud visszaadni ezért generikus. Kiértékeli a függvényt a lista összes elemén az eredményt pedig eldobja.

Lehet Intet és Stringet összeadni, majd kiiratni mégpedi a string interpolációval

string interpoláció:

Lehetőségünk van a string-ekbe változókat, kifejezéseket beilleszteni. Ha egy macskakörmök közti string elé az ’s’ karaktert írjuk, akkor a benne lévő $ jelekkel változókat vagy tetszőleges kifejezéseket adhatunk meg.

-pl.: ’$x’ behelyettesítődik az x értékével a string-be

-ha ’$’ karaktert szeretnénk kiírni akkor két dollárjelet írunk

-$ jel után kapcsoszárójelben megadhatunk tetszőleges kifejezést, amit kiértékel a scala runtime és az eredmény kerül behelyettesítésre

-a print(x) mindig lefut mert minden objektumnak van toString metódusa, amelyet a print automatikusan meghív

-két változata van print(s String) és print(a: Any)

-ha printet egy tetszőleges típussal hívjuk meg (Any) akkor ez a metódus meghívja a toString metódust az objektumon

-ha felülírjuk a toStringet akkor a fordító nem generál automatikusan csak ha nem tesszük

for comprehension:

Be tudjuk vele járni a kollekciókat elemenként, viszont ha a Scala fordító egy for alakú kifejezést talál akkor automatikusan átalakítja foreach alakúra és ezt fordítja le.

map vs foreach:

-mindekettő kiértékeli a változót a lista összes elemén

-a foreach eldobja az eredményeket és visszakapjuk értékként a Unitot

-a map az eredményekből készít egy új listát és azt adja vissza

for yield:

Lehetőségünk van a monádokkal való szekvenciális műveletek végrehajtására és az eredmények transzformálására. A for kifejezés egy ciklust hajt végre vagy iterál addig a for yield kifejezés egy monáddá csomagolja be, a yield kulcsszóval határozzuk meg a végén az eredményt.

Vector:

A Vector egy szekvenciális kollekció, ami hatékonyan támogatja az elemek hozzáadását, eltávolítását és hozzáférését.

-apply metódus: lehetővé teszi, hogy az adott indexen lévő elemhez hozzáférjünk. apply(index: Int)

-updated metódus: kicserélhető egy adott elem az adott indexen lévő elemmel, az eredmény egy új Vector lesz amely tartalmazza a módosított elemet updated(index: Int, value: Int)

-append: elem hozzá adása a Vector végéhez

-prepend: elem hozzáadása a Vector elejéhez

-konverziós metódusok: a Vector tartalmaz konveriós metódusokat toVector és toList, amelyek lehetővé teszik az átalakítást más konténer típusok között

A Vector is egy monád, tehát van neki flatMap, flatten és map metódusa. A Vector immutábilis tehát minden módosítás során új Vector-t eredményez.

Set

A Set egy olyan kollekció, amely csak egyedi elemeket tartalmazhat és nem tárolja az elemek sorrendjét.

-immutable: nem lehet az elemeket közvetlenül módosítani, új Set-et kell létrehozni hozzá

-flatMap, map, filter, foreach

-hashSet hashCode alapján tárolja az elemeket

hashCode:

Úgyanúgy mint a toString metódus a hashCode(): Int metódus is létezik minden objektumon.

-egy int számot rendel az objektumhoz

-két egyenlő elem hash kódja is meg kell hogy egyezzen

-létrehoz egy Vector[List[t]]-t és hashCode alapján számítja ki, hogy a a vektor hanyadik elemébenlévő listába szúrja be a beszúrandó elemet.

-toList, toVector metódus

Kovariancia

-deklaráció: Option[+T]

- ha A egy altípusa B-nek (A <: B), akkor C[A] is altípusa C[B]-nek

Kontravariancia

-deklaráció: Option[-T]

- ha A egy altípusa B-nek (A <: B), akkor C[B] lesz altípusa C[A]-nak

Invariáns

-alapból minden típus invariáns (Option[T])

- Sem C[A] <: C[B], sem C[B] <: C[A] nem következik abból, hogy A <: B