Java Collections Framework

A *Java Collections Framework (JCF) Gyűjtemény-Keretrendszer* kifejezetten gyűjteményekhez készített interfészeket és azok implementációit, például gyűjteményeken végezhető rendezési, keresési műveletek megvalósítását tartalmazza.

Collection

A *gyűjtemény -collection-* azonos típusú objektumpéldányok egysége. A gyűjtemény egyes objektumpéldányai nem elszórtan helyezkednek el a memóriában, hanem egy szabályos konstrukcióban, ezáltal elérésük, szerkesztésük a gyűjtemény keretein belül, a gyűjtemény szabványos metódusaival történhet.

Collection ajánlott metódusai:

- *int size()*: a gyűjtemény elemeinek számát adja meg
- *boolean isEmpty()*: true, ha üres a gyűjtemény
- *boolean contains(Object element)* : true, ha az *Object* típusú *element* (általánosított) nevű elem a gyűjteményben van
- *boolean add(E element)*: elem hozzáadása (opcionális), true, ha a gyűjteményben történt változás (ha a gyűjtemény halmaz, és már tartalmazza az *element* elemet, akkor nem történik változás)
- *boolean remove(Object element)*: elem törlése (opcionális), true, ha az adott elem a gyűjtemény része volt
- *Iterator<E> iterator()*: bejárás *-iterátor-* objektum a gyűjtemény bejárásához

Elemcsoportokkal végezhető ajánlott metódusok:

- *boolean containsAll(Collection<?> c)*: true, ha c minden elemét tartalmazza a gyűjtemény – részhalmaz
- *boolean addAll(Collection<? extends E> c)*: c minden elemét hozzáadja a gyűjteményhez – unió (opcionális)
- *boolean removeAll(Collection<?> c)*: c elemeit eltávolítja a gyűjteményből – különbség (opcionális)
- *boolean retainAll(Collection<?> c)*: a c-vel közös elemeket hagyja a gyűjteményben – metszet (opcionális)
- *void clear()*: minden elemet töröl a gyűjteményből
- *Object[] toArray()*: a gyűjtemény elemeit tömbként adja vissza

Interface

Egy *kapcsolófelület -interface-* abban különbözik egy osztálytól, hogy az általa felkínált metódusoknak csak a fejlécét adja meg, azaz csak úgynevezett *metódus szignatúrákat* deklarál, azonban a metódusok törzsei üresek. Emiatt, ha egy osztály használni akar egy interfészt, ezt csak akkor teheti meg, ha ténylegesen kódolja, érvényesen megvalósítja az interfész összes metódusát, azaz implementálja azokat.

A *Collection* interfészként jelenik meg a JCF-ben. A Collection interfész deklarációja:

```
public interface Collection<E> extends Iterable<E>
```

A deklaráció elemeinek magyarázata:

A Collection interfész nyilvános -*public*-, azaz bármelyik osztály használhatja szolgáltatásait; A Collection interfész az Iterable interfész ősosztályból származik, annak kiterjesztése -*extends*-, azaz örökli az ősosztály metódusait illetve osztályváltozóit; A Collection interfész különböző elemtípusokkal paraméterezhető, ennek jelzésére az <*E*> jelet használjuk, ahol *E* jelentése: az alkotóelem –*ElementType*- típusa, <> a paraméter formális helye; Az <*E*> jelzés tehát az interfész megvalósításában szereplő objektumelemek egységes típusára utal.

A Collection interfész általános interfész, amelynek nincs konkrét megvalósítása a Javában. A Collection interfészből további interfészek származtathatók a Java öröklődési mechanizmusa segítségével. Ezek a származtatott interfészek értelemszerűen rendelkeznek a Collection összes metódus szignatúrájával. Ilyen interfész például a *List*<*E*>. A List<E> származtatott interfésznek már létezik standard fejlesztői megvalósítása az *ArrayList* osztályban: ***ArrayList<E> implements List<E> extends Collection<E> extends Iterable<E>.***

Az *Iterable*<*E*> interfész legfontosabb metódusa egy *Iterator*<*E*> értékkel visszatérő *iterator()* metódus, ami egy bejárási lehetőség (bejárás objektum) az *E* elemtípusú elemekkel szervezett gyűjteményen. Ezt a metódust kell megvalósítania az interfészt implementáló osztályoknak.

Az Iterable<E> interfész *iterator()* metódusának *Iterator*<*E*> visszatérési értéke szintén interfész! Ennek az Iterator<E> interfésznek a következő megvalósítandó metódusai vannak:

- *boolean hasNext()*: true, ha van következő eleme a gyűjteménynek
- *E next()*: a következő elemet adja vissza az iterációban. A következő elem típusa is *E* (az Iterable interfész szintén generikus -általános- típusparamétert használ; *E* az *element* elem *ElementType* típus paramétere)
- *void remove()*: törli az utoljára, next() -tel hívott elemet

A gyűjtemények bejárására 2 lehetőség is van, az egyik a *for-each* ciklus, a másik az *iterátor* -bejáró- alkalmazás. Ez utóbbi lehetővé teszi a gyűjtemény elemeinek eltávolítását is.

A *List* interfésznek két implementációja van, az egyik a már említett *ArrayList*, a másik pedig a *LinkedList*. ArrayList alkalmazása esetén az elemek elérése gyors, konstans idejű. Ha azonban gyakran adunk elemeket egy listához, vagy gyakran kell bejárni a listát egy elem törléséhez, a LinkedList gyorsabb, mint az ArrayList.

A JCF-nek vannak szabványos osztálykomponensei is. Ilyen a *Collection**s*** osztály. *A* Collections osztály a JCF-ben deklarált interfészek számos hasznos megvalósítását kínálja fel, melyeket gyűjteményekre alkalmazhatunk:

- *sort*: rendezi a gyűjtemény elemeit,
- *shuffle*: összekeveri az elemeket,
- *reverse*: megfordítja az elemek sorrendjét,
- *rotate*: az elemeket megadott távolsággal elmozdítja,
- *swap*: felcserél 2 elemet a gyűjteményben,
- *replaceAll*: az összes elemet kicseréli egy másik elemre,
- *fill*: felülírja az elemeket,
- *copy*: egy célgyűjteménybe másolja a forrásgyűjteményt,
- *binarySearch*: binárisan keres egy adott elemet,
- *indexOfSubList*: egy részgyűjtemény (részlista) előfordulásának kezdőindexével tér vissza,
- *lastIndexOfSubList*: egy részgyűjtemény (részlista) utolsó előfordulásának kezdő indexével tér vissza.

Rendezés: Comparable és Comparator interfészek

Az objektumok összehasonlíthatósága, illetve ezen keresztül rendezhetőségük a matematikai absztrakció fontos részét képezik. Ezért a *Comparable* interfészt a Java nyelv több, mint 150 osztályában implementálták a nyelv fejlesztői. Ezen osztályokban tehát létezik (implementálva van) egy, az osztály típusának megfelelő, alapértelmezett rendezési elv. Ilyen osztályok például: Byte, Character, Long, Integer, Short, Double, Float, Boolean, File, String, Date, ... stb.

A felsorolt osztályok objektumainak rendezésére a Collections osztály *Collections.sort(T)* metódusát használhatjuk, ahol *T* az objektumok típusát jelöli. Ha a gyűjtemény elemei nem valósítják meg a Comparable interfészt, akkor ClassCastException kivételt kapunk, vagyis nekünk kell az elemet definiáló osztályban megvalósítani a Comparable interfészt. Ha többféle értelmű rendezést is szeretnénk, akkor a Comparátor interfészt kell megvalósítanunk. A Comparable interfésznek ugyanis csak egyetlen metódusa van, a *compareTo* metódus, amelynek egy int értéket kell visszaadnia.

```
public interface Comparable<T> {
  public int compareTo(T o);
}
```

A visszaadott érték:

- negatív: ha az aktuális objektum kisebb, mint a paraméter,
- nulla: ha az aktuális objektum és a paraméter egyenlő,
- pozitív: ha az aktuális objektum nagyobb, mint a paraméter.