# 6. előadás

## Generikus osztályok

Az UML-beli paraméteres osztályok Java-ban generikus osztályok segítségével valósíthatóak meg

Az UML-ben jelölt paraméterek lesznek a generikus paraméterek

A generikus paraméterek Java-ban osztálynevek lehetnek, melyek segítségével a generikus osztály definíciójában paraméterezhető típusok adhatóak meg

A generikus osztályok használatakor meg kell adni a generikus paraméterek konkrét értékeit (tehát a konkrét osztályneveket), melynek hatására a kapott konkrét osztályban a generikus paraméterekkel jelölt típusok már a konkrét típusok lesznek

A generikus osztályok használata is egyfajta absztrakció, de az absztrakt osztályokkal ellentétben itt nem az elvégzendő műveletek ismeretlenek, hanem az adatok típusa ismeretlen (legalább részben), amelyeken a műveleteket végezzük

## Gyűjtemények

A gyűjtemény egy absztrakt adatszerkezet: változó számosságú adatok csoportosítását végzi, mely adatok az adott probléma megoldása szempontjából egyformán fontosak, és rajtuk szabályozott módon műveleteket kell végezni

Legtöbbször a tárolt adatok egyforma típusúak, vagy legalábbis ugyanabból a típusból származó típusúak

A tömböket nem tekintjük gyűjteményeknek, mert rögzített mérettel rendelkeznek. Igaz, a gyűjtemények megvalósításához gyakran használunk tömböket

### Kép

* Generikus osztályokkal valósíthatóak meg
* a tárolt elemek száma
* üres-e?
* tartalmazza-e a megadott objektumot?
* megad egy iterátort a bejáráshoz
* hozzáadja a megadott elemet
* eltávolítja a megadott elemet
* benne van-e a megadott gyűjtemény minden eleme
* hozzáadja a megadott gyűjtemény elemeit
* eltávolítja a megadott gyűjtemény elemeit
* meghagyja a megadott gyűjtemény elemeit
* eltávolítja az összes elemet
* tömbbé konvertálja

## Gyűjtemények bejárása

Egy gyűjtemény bejárásakor a gyűjtemény minden elemét sorra vesszük, és minden elemmel elvégezünk egy adott műveletet

Általában a gyűjtemények nem indexelhetőek, ezért egy úgynevezett iterátor segítségével járhatóak be



Egyszerűbben: $foreach$ ciklussal



## Gyűjtemények megvalósítása

A $Collection<E>$ interfészt, vagy akár valamelyik speciálisabb interfészét kell megvalósítani

Érdemes a megvalósított gyűjteménynek is generikus osztálynak lennie, hogy tetszőleges típusú adat tárolására alkalmas legyen

A gyűjtemény műveleteinek absztrakt formái a megvalósítandó interfészben már adottak, így csak a tárolt adatok reprezentációjával és a műveletek függvénytörzseinek meghatározásával kell törődnünk

Az $AbstractCollection<E>$ osztály már tartalmazza a szokásos gyűjteményi viselkedést, így érdemes abból származtatni a gyűjteményünket, és csak a lényegre koncentrálni

### Sor megvalósítása x segítségével

|  |  |
| --- | --- |
| Sor megvalósítása tömb segítségével | Sor megvalósítása láncolt listával |
| package collections**;****import** java**.**util**.**AbstractCollection**;****import** java**.**util**.**Iterator**;****import** java**.**util**.**NoSuchElementException**;****import** java**.**util**.**Queue**;**public class LinkedListQueue**<**E**>** **extends** AbstractCollection**<**E**>** **implements** Queue**<**E**>** **{** private static class Node**<**E**>** **{** E item**;** LinkedListQueue**.**Node**<**E**>** next**;** Node**(**E element**,** LinkedListQueue**.**Node**<**E**>** next**)** **{** **this.**item **=** element**;** **this.**next **=** next**;** **}** **}**  private Node**<**E**>** first**;** private Node**<**E**>** last**;** private int size**;**  public LinkedListQueue**(){** first **=** last **=** **null;** size **=** 0**;** **}** @Override public Iterator**<**E**>** iterator**()** **{** **return** **new** Iterator**<**E**>()** **{** private Node**<**E**>** actElement **=** first**;**  @Override public boolean hasNext**()** **{** **return** actElement **!=** **null;** **}** @Override public E next**()** **{** **if** **(**actElement **==** **null){** **throw** **new** NoSuchElementException**();** **}** E e **=** actElement**.**item**;** actElement **=** actElement**.**next**;** **return** e**;** **}** @Override public void remove**()** **{** **throw** **new** UnsupportedOperationException**();** **}** **};** **}** @Override public boolean add**(**E e**)** **{** Node**<**E**>** newNode **=** **new** Node**<>(**e**,** **null);** **if** **(**last **==** **null){** first **=** last **=** newNode**;** **}else{** last**.**next **=** newNode**;** last **=** newNode**;** **}** **++**size**;** **return** **true;** **}** @Override public int size**()** **{** **return** size**;** **}** @Override public boolean offer**(**E e**)** **{** **return** add**(**e**);** **}** @Override public E remove**()** **{** E e **=** poll**();** **if** **(**e **==** **null)** **{** **throw** **new** NoSuchElementException**();** **}** **else** **{** **return** e**;** **}** **}** @Override public E poll**()** **{** **if** **(**first **==** **null){** **return** **null;** **}** E e **=** first**.**item**;** first **=** first**.**next**;** **--**size**;** **return** e**;** **}** @Override public E element**()** **{** E e **=** peek**();** **if** **(**e **==** **null)** **{** **throw** **new** NoSuchElementException**();** **}** **else** **{** **return** e**;** **}** **}** @Override public E peek**()** **{** **if** **(**first **==** **null){** **return** **null;** **}else{** **return** first**.**item**;** **}** **}****}** | package collections**;** **import** java**.**util**.**AbstractCollection**;****import** java**.**util**.**Iterator**;****import** java**.**util**.**NoSuchElementException**;****import** java**.**util**.**Queue**;**public class ArrayQueue**<**E**>** **extends** AbstractCollection**<**E**>** **implements** Queue**<**E**>** **{** private E**[]** elements**;** private final int capacity**;** private int size**;** public ArrayQueue**()** **{** capacity **=** 10**;** elements **=** **(**E**[])** **new** Object**[**capacity**];** size **=** 0**;** **}** @Override public boolean add**(**E e**)** **{** **if** **(**size **==** capacity**)** **{** **throw** **new** IllegalStateException**();** **}** elements**[**size**]** **=** e**;** **++**size**;** **return** **true;** **}** @Override public boolean offer**(**E e**)** **{** **try** **{** **return** add**(**e**);** **}** **catch** **(**IllegalStateException ex**)** **{** **return** **false;** **}** **}** @Override public E remove**()** **{** E e **=** poll**();** **if** **(**e **==** **null)** **{** **throw** **new** NoSuchElementException**();** **}** **else** **{** **return** e**;** **}** **}** @Override public E poll**()** **{** **if** **(**isEmpty**())** **{** **return** **null;** **}** **else** **{** E e **=** elements**[**0**];** E**[]** newElements **=** **(**E**[])** **new** Object**[**capacity**];** System**.**arraycopy**(**elements**,** 1**,** newElements**,** 0**,** size**);** elements **=** newElements**;** size**--;** **return** e**;** **}** **}** @Override public E element**()** **{** E e **=** peek**();** **if** **(**e **==** **null)** **{** **throw** **new** NoSuchElementException**();** **}** **else** **{** **return** e**;** **}** **}** @Override public E peek**()** **{** **if** **(**isEmpty**())** **{** **return** **null;** **}** **else** **{** **return** elements**[**0**];** **}** **}** @Override public int size**()** **{** **return** size**;** **}** @Override public Iterator**<**E**>** iterator**()** **{** **return** **new** Iterator**<**E**>()** **{** private int actIndex **=** 0**;** @Override public boolean hasNext**()** **{** **return** actIndex **<** size**;** **}** @Override public E next**()** **{** **if** **(**actIndex **==** size**)** **{** **throw** **new** NoSuchElementException**();** **}** E e **=** elements**[**actIndex**];** **++**actIndex**;** **return** e**;** **}** @Override public void remove**()** **{** **throw** **new** UnsupportedOperationException**();** **}** **};** **}****}** |

## A Java gyűjteményei

### $$java.util.Collection$$



### $$java.util.Map$$



## Algoritmusok

Általában az adatszerkezetekhez tartozó műveletek

Gyűjtemények esetében az adott gyűjtemény elemein elvégzendő művelet

Meg tudunk határozni olyan algoritmusokat, amelyeket gyakran szoktunk gyűjtemények elemein elvégezni, például: rendezés, keverés, feltöltés, kiválasztás, stb.

Az algoritmusokat érdemes úgy megvalósítani, hogy tetszőleges gyűjtemény elemein használható legyen

Ebből következik, hogy az elemek bejárásához csupán a gyűjtemény iterátorát tudjuk használni

Ilyenek például a tanult programozási tételek is, például a maximumkiválasztás

### Algoritmusok gyűjteményeken ($java.util.Collections$) és tömbökön ($java.util.Arrays$)

→<http://docs.oracle.com/javase/7/docs/api/java/util/Collections.html>
→<http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html>

## ZH

Kurzusmail lesz még róla.

Előadás vége.