Дәріс №10

Коллекциялар

Массивтер Java-да деректер жинағын сақтауға арналған. Дегенмен, оларды пайдалану әрдайым ыңғайлы емес, ең алдымен олардың нақты ұзындығы бар. Бұл мәселені Java-да коллекциялар шешеді. Дегенмен, мәселе тек көлемдегі объектілердің икемді жиынтығында ғана емес, сонымен қатар жинау кластары әртүрлі алгоритмдер мен мәліметтер құрылымын, мысалы, стек, кезек, ағаш және басқалары сияқты орындайтындығында.

Коллекциялар класы java.util папкасында орналасқан, сондықтан коллекцияларды қолданар алдында осы пакетті қосу керек.

Java-да көптеген коллекциялар бар болғанымен, олардың барлығы үйлесімді және логикалық жүйені құрайды. Біріншіден, барлық коллекциялар негізгі функционалдылықты анықтайтын бір немесе басқа интерфейсті қолдануға негізделген. Осы интерфейстердің арасында мыналарды ерекшелеуге болады:

- Collection: барлық коллекцияларға және басқа коллекциялық интерфейстерге арналған негізгі интерфейс;
- Queue: Collection интерфейсті мұра етеді және деректер құрылымына кезек ретінде қызмет етеді;
- Deque: Queue интерфейсін мұра етеді және екі бағытты кезектерге арналған функционалдылықты ұсынады;
- List: Collection интерфейсті мұра етеді және қарапайым тізімдердің функционалдығын ұсынады;
- Set: сонымен қатар Collection интерфейсін кеңейтеді және бірегей (уникалды) объектілер жиынтығын сақтау үшін қолданылады;
- SortedSet: Сұрыпталған коллекцияларды құруға арналған Set интерфейсін кеңейтеді.
- NavigableSet: Сәйкестік бойынша іздеуге болатын коллекциялар жасау үшін SortedSet интерфейсін кеңейтеді.
- Map: әр элемент нақты кілт пен мәнге ие болатын сөздік түрінде мәліметтер құрылымын құруға арналған. Басқа коллекциялық интерфейстерден айырмашылығы, ол Collection интерфейстен мұраланбаған.

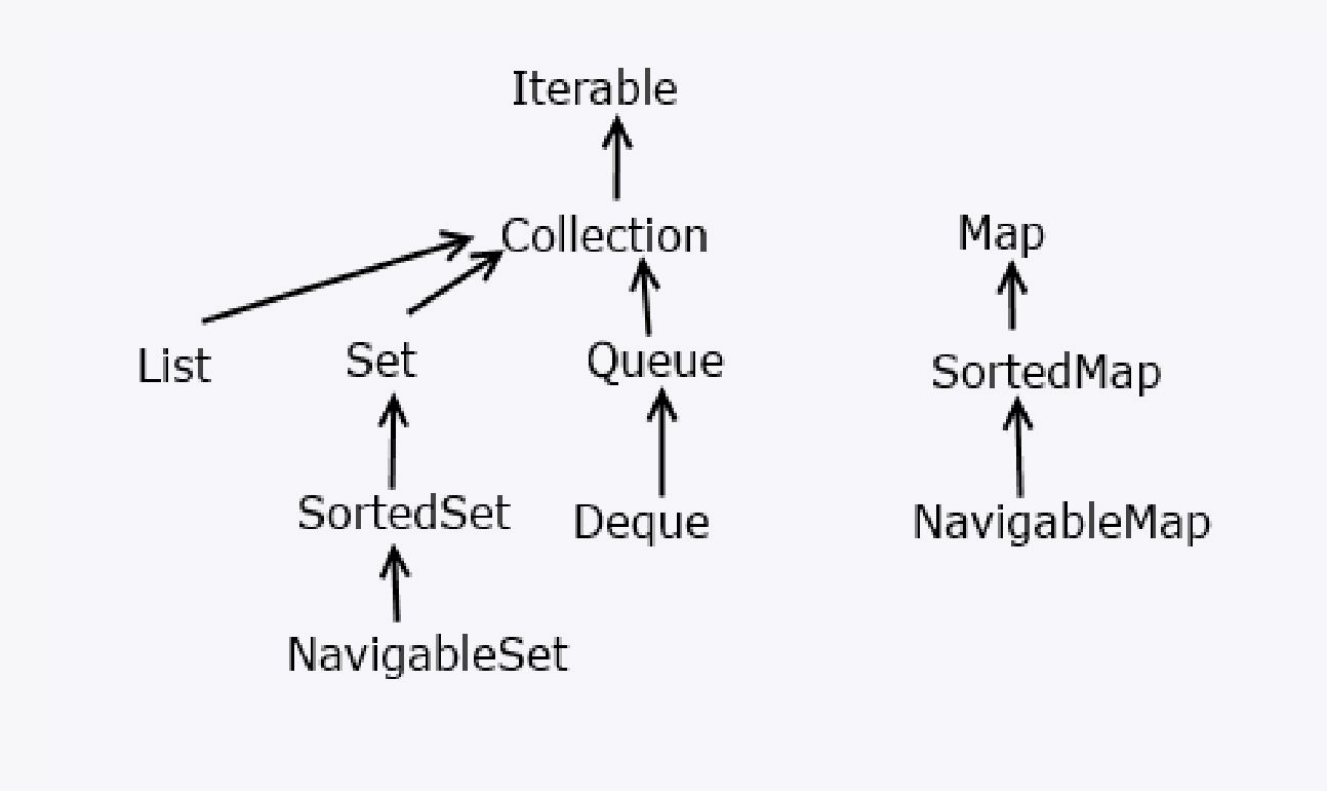
Бұл интерфейстер ішінара абстрактылы кластар арқылы жүзеге асырылады:

- AbstractCollection: Collection интерфейсті қолданатын басқа коллекциялар үшін негізгі базалық класс
- AbstractList: AbstractCollection класын кеңейтеді және тізімдер түрінде коллекцияларды құруға арналған List интерфейсін қолданады.
- AbstractSet: AbstractCollection класын кеңейтеді және жиындар түрінде коллекциялар жасау үшін Set интерфейсін қолданады
- AbstractQueue: AbstractCollection класын кеңейтеді және кезек пен стек түрінде коллекциялар құруға арналған Queue интерфейсін қолданады.
- AbstractSequentialList: сонымен қатар AbstractList класын кеңейтеді және List интерфейсін жүзеге асырады. Байланыстырылған тізімдерді жасау үшін қолданылады.
- AbstractMap: «кілт-мәні» жұбы түрінде объектілері бар сөздік типінің жиынтығын жасауға арналған Map интерфейсін қолданады.

Жоғарыда аталған интерфейстер мен абстрактылы кластарды қолдана отырып, Java коллекциялық кластарының кең тізімін - тізімдер, жиындар, кезектер, салыстырулар және басқаларын жүзеге асырады, олардың ішінде:

- ArrayList: объектілердің қарапайым тізімі
- LinkedList: байланыстырылған тізімді ұсынады
- ArrayDeque: коллекцияның басында да, соңында да қосуға және жоюға болатын екі бағытты кезек класы.
- HashSet: объектілер жиынтығы немесе хэш жиынтығы, онда әр элементтің кілті болады – бірегей (уникальный) хэш-код
- TreeSet: ағаш көрінісінде сұрыпталған объектілер жиынтығы
- LinkedHashSet: байланысты хэш жиынтығы
- PriorityQueue: Басымдық кезегі
- HashMap: әр объектінің өзіндік кілті және кейбір мәні бар сөздік түріндегі деректер құрылымы
- TreeMap: ағаш түріндегі деректер құрылымы, онда әр элементтің өзіндік кілті және кейбір мәні болады.

Сызбалық түрде бүкіл коллекция жүйесін келесі түрде қысқаша сипаттауға болады:



Collection интерфейсі. Коллекцияның интерфейсі барлық коллекциялар үшін базалық болып табылады, негізгі функцияны анықтайды:

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


    // әдістерді анықтау
}
```

Collection интерфейсі жалпыланған және Iterable интерфейсін кеңейтеді, сондықтан барлық коллекциялық объектілерді for-each типіндегі массив бойынша қарастырады.

Коллекция интерфейсі әдістерінің ішінде төмендегілерді ерекшелеуге болады:

- boolean add (E item): коллекцияға элемент қосады. Егер сәтті болса true мәнін, сәтсіз болса false мәнін қайтарады.
- boolean addAll (Collection<? extends E> col): col коллекцияға барлық элементтерді қосады. Егер сәтті болса true мәнін, сәтсіз болса false мәнін қайтарады.
- void clear (): коллекциядан барлық элементтерді алып тастайды.
- boolean contains (Object item): егер item объектісі коллекцияда болса true мәнін, әйтпесе false мәнін қайтарады.
- boolean isEmpty (): егер коллекция бос болса true мәнін, әйтпесе false мәнін қайтарады.
- Iterator<E> iterator (): коллекция элементтерін қарастыру үшін Iterator объектісін қайтарады.
- boolean remove (Object item): егер item объектісі коллекциядан сәтті алынып тасталса true мәнін, әйтпесе false мәнін қайтарады.
- boolean removeAll (Collection<?> col): ағымдағы коллекциядан col коллекциясындағы барлық объектілерін алып тастайды. Егер ағымдық коллекция өзгерсе true мәнін, әйтпесе false мәнін қайтарады.

- boolean retainAll (Collection<?> col): col коллекциясынан басқа барлық объектілерді ағымдағы коллекциядан шығарады. Егер ағымдағы коллекция жойылғаннан кейін өзгерсе true мәнін, әйтпесе false мәнін қайтарады.
- int size (): коллекциядағы элементтер санын қайтарады
- Object[] toArray (): коллекциядағы барлық элементтері бар массивті қайтарады.

Collection интерфейсінде бар осы және басқа әдістерді барлық коллекциялар орындайды, сондықтан тұтастай алғанда, коллекциялармен жұмыс жасаудың жалпы принциптері бірдей болады. Біртекті интерфейс түсінуді жеңілдетеді және әртүрлі коллекциялармен жұмыс жасайды. Сонымен, элементті add әдісін қолдану арқылы жүзеге асырылады және қосылған элементті параметр ретінде қабылдайды. Жою үшін remove() әдісі шақырылады. clear әдісі коллекцияны жояды, ал size әдісі коллекциядағы элементтер санын қайтарады.

ArrayList класы және List интерфейсі.

Қарапайым тізімдерді жасау үшін, коллекция интерфейсінің функционалдығын кеңейтетін List интерфейсін пайдаланамыз. Тізім интерфейсінің жиі қолданылатын әдістері:

- void add(int index, E obj): тізімге obj объектісін index бойынша қосады.
- boolean addAll(int index, Collection<? extends E> col): тізімге col коллекциясының барлық элементтерін index бойынша қосады. Егер тізім қосу нәтижесінде өзгертілген болса true мәнін, әйтпесе false мәнін қайтарады.
- E get(int index): тізімнен объектіні index бойынша қайтарады.
- int indexOf(Object obj): тізімде obj алғашқы пайда болу индексін қайтарады. Егер объект табылмаса, онда -1 қайтарылады.
- int lastIndexOf(Object obj): тізімдегі объект obj-ның соңғы пайда болу индексін қайтарады. Егер объект табылмаса, онда -1 қайтарылады.

- ListIterator<E> listIterator (): тізім элементтерін қарастыру үшін ListIterator объектісін қайтарады.
- static <E> List<E> of(элементтер) : элементтер жиынынан List объектісін құрады.
- E remove(int index): жойылған объектіні қайтару кезінде тізімнен объектіні index бойынша алып тастайды.
- E set(int index, E obj): obj объектісінің мәнін индексте орналасқан элементке тағайындайды.
- void sort(Comparator<? super E> comp): comp компаратордың көмегімен тізімді сұрыптайды.
- List<E> subList(int start, int end): тізімде тұрған start және end индекстері арасында орналасқан элементтердің жиынтығын алады.

Әдетте, Java-да осы интерфейстің ендірілген - ArrayList класы бар. ArrayList класы оның функционалдығын AbstractList класынан алатын және List интерфейсін қолданатын жалпы коллекцияны білдіреді. Қарапайым сөзбен айтқанда, ArrayList массивке ұқсас қарапайым тізімді ұсынады, тек ондағы элементтер саны бекітілмеген.

ArrayList-де келесі конструкторлар бар:

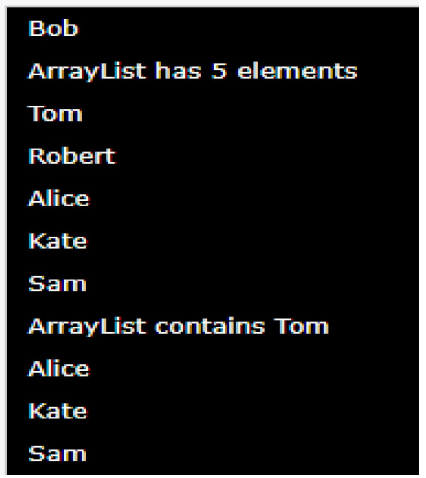
- ArrayList (): бос тізімді жасайды
- ArrayList(Collection <? extends E> col): col коллекцияның барлық элементтері қосылатын тізімді жасайды.
- ArrayList (int capacity): бастапқы capacity сыйымдылығы бар тізімді құрады.

ArrayList-дегі сыйымдылық объектілерді сақтау үшін қолданылатын массивтің көлемін білдіреді. Элементтер қосылған кезде, жады іс жүзінде қайта бөлінеді - жаңа массив құру және оған ескі массивтен элементтерді көшіру. Бастапқы ArrayList сыйымдылығы мұндай жадыны қайта бөлуді азайтады, осылайша өнімділікті жақсартады.

Программада ArrayList класын және оның кейбір әдістерін қолданамыз:

```java
import java.util.ArrayList;

public class Program{

    public static void main(String[] args) {

        ArrayList<String> people = new ArrayList<String>();
        // добавим в список ряд элементов
        people.add("Tom");
        people.add("Alice");
        people.add("Kate");
        people.add("Sam");
        people.add(1, "Bob"); // добавляем элемент по индексу 1

        System.out.println(people.get(1));// получаем 2-й объект
        people.set(1, "Robert"); // установка нового значения для 2-го объекта

        System.out.printf("ArrayList has %d elements \n", people.size());
        for(String person : people){

            System.out.println(person);
        }
        // проверяем наличие элемента
        if(people.contains("Tom")){

            System.out.println("ArrayList contains Tom");
        }

        // удалим несколько объектов
        // удаление конкретного элемента
        people.remove("Robert");
        // удаление по индексу
        people.remove(0);

        Object[] peopleArray = people.toArray();
        for(Object person : peopleArray){

            System.out.println(person);
        }
    }
}
```

Программаның консольдік нәтижесі:



Мұнда ArrayList объектісі String класымен анықталған, сондықтан тізім тек жолдарды сақтайды. ArrayList класы Collection<E> интерфейсін қолданатындықтан, біз тізімдегі объектілерді басқару үшін осы интерфейс әдістерін қолдана аламыз.

Қосу үшін add әдісі шақырылады. Оның көмегімен тізімнің соңына объектіні қоса аламыз: people.add("Tom"). Объектіні тізімдегі белгілі бір орынға қоса аламыз, мысалы, екінші орынға объектіні қосуға болады (яғни 1 индексінде, өйткені нөмірлеу нөлден басталады): people.add (1, «Боб»).

Size () әдісі коллекциядағы объектілердің санын білуге мүмкіндік береді.

Құрамында әдісі бар коллекцияда элементтің бар-жоғын тексеру remove әдісін қолдану арқылы іске асырылады. Белгілі бір элементті алып тастай аламыз: people.remove("Tom") немесе индекс арқылы people.remove (0).

Белгілі бір элементті get() әдісі арқылы индекстеу арқылы алуға болады: String person = people.get(1); және берілген әдісті қолдана отырып индекстеу арқылы элементті орнатуға болады: people.set (1, «Роберт»);

ToArray() әдісін қолдана отырып, тізімді объектілер массивіне айналдыра аламыз.

ArrayList класы Iterable интерфейсін қолданатын болғандықтан, біз тізімге for-each: циклі көмегімен өтуге болады for (String person : people).

ArrayList объектісіне қосымша объектілерді еркін қоса аламыз, бірақ массивтен айырмашылығы, ArrayList объектілерді қайтадан сақтау үшін массив қолданады. Әдепкі бойынша, бұл массив 10 объектіге арналған. Егер бағдарлама барысында көп объект қосылса, онда барлық мөлшерді жинай алатын жаңа массив құрылады. Мұндай жадыны қайта бөлу өнімділікті төмендетеді. Сондықтан, егер тізімде элементтердің белгілі бір санынан, мысалы, 25-тен аспайтынына сенімді болсақ, онда бұл санды конструкторда да бірден : ArrayList <String> people = new ArrayList <String> (25)), немесе ensureCapacity әдісін қолдана отырып: people.ensureCapacity (25); деп орната аламыз.

Кезектер және ArrayDeque класы. Кезектер FIFO (first in - first out) мәліметтер құрылымын білдіреді. Яғни коллекцияға неғұрлым ертерек қосылса, одан ертерек алынып тасталады. Бұл стандартты бір бағытты кезек моделі. Алайда, екі бағытқа ие, яғни элементті тек басына ғана емес, соңына да қосуға болатындар бар. Сонымен, элементті тек аяғынан ғана емес, басынан алып тастауға болады.

Кезек класының ерекшелігі - олар арнайы Queue немесе Deque интерфейстерін орындайды.

Queue интерфейсі. Жалпы Queue<E> интерфейсі базалық Collection интерфейсті кеңейтеді және класты бір бағытты кезек ретінде анықтайды. Ол өзінің функционалдығын келесі әдістер арқылы ашады:

- E element(): элементті кезектің алдыңғы жағынан қайтарады, бірақ жоймайды. Егер кезек бос болса, NoSuchElementException ерекшелігіне жіберіледі.
- boolean offer(E obj): obj элементін кезектің соңына қосады. Егер элемент сәтті қосылса, true мәнін әйтпесе false мәнін қайтарады.
- E peek(): элементті кезектің алдыңғы жағынан жоймай қайтарады. Егер кезек бос болса, null мәнін қайтарады.
- E poll(): элементті кезектің басынан алып тастағанда қайтарады. Егер кезек бос болса, null мәнін қайтарады.
- E remove(): элементті кезектің басынан алып тастағанда қайтарады. Егер кезек бос болса, NoSuchElementException ерекшелігіне жіберіледі.

Осылайша, осы интерфейсті қолданатын барлық кластарда кезекке қосудың offer әдісі, кезек басынан элементті алу әдісі poll, элементті кезек басынан алуға мүмкіндік беретін peek және element әдістері бар.

Deque интерфейсі. Deque интерфейсі жоғарыда сипатталған кезек интерфейсін кеңейтеді және тұрақты бір бағытты кезек ретінде жұмыс істейтін немесе LIFO принципіне сәйкес жұмыс жасайтын екі бағытты кезектің тәртібін анықтайды (соңғы енгізілген - бірінші шығарылған).

Deque интерфейсі келесі әдістерді анықтайды:

- void addFirst (E obj): элементті кезектің алдыңғы жағына қосады;
- void addLast (E obj): obj элементін кезектің соңына қосады;
- E getFirst(): элементті кезектің басынан шығармай қайтарады. Егер кезек бос болса, NoSuchElementException ерекшелігіне жібереді;
- E getLast(): кезектің соңғы элементін жоймай қайтарады. Егер кезек бос болса, NoSuchElementException ерекшелігіне жібереді;
- boolean offerFirst (E obj): obj элементін кезектің басына қосады. Егер элемент сәтті қосылса, true мәнін әйтпесе false мәнін қайтарады.
- boolean offerLast (E obj): obj элементін кезектің соңына қосады. Егер элемент сәтті қосылса, true мәнін әйтпесе false мәнін қайтарады.
- E peekFirst(): элементті кезектің алдыңғы жағынан жоймай қайтарады. Егер кезек бос болса, null мәнін қайтарады.

- E peekLast (): кезектің соңғы элементін жоймай қайтарады. Егер кезек бос болса, null мәнін қайтарады.
- E pollFirst (): элементті кезектің басынан алып тастағанда қайтарады. Егер кезек бос болса, null мәнін қайтарады.
- E pollLast (): кезектің соңғы элементін жойып қайтарады. Егер кезек бос болса, null мәнін қайтарады.
- E pop (): элементті кезектің басынан алып тастағанда қайтарады. Егер кезек бос болса, NoSuchElementException ерекшелігіне жібереді.
- void push (E элементі): элементті кезектің басына қосады
- E removeFirst (): элементті кезектің басынан алып тастағанда қайтарады. Егер кезек бос болса, NoSuchElementException ерекшелігіне жібереді.
- E removeLast (): элементті кезектің соңынан шығарумен қайтарады. Егер кезек бос болса, NoSuchElementException ерекшелігіне жібереді.
- boolean removeFirstOccurrence (Object obj): кезектен шыққан бірінші obj элементін жояды. Егер жою орын алса, true мәнін әйтпесе false мәнін қайтарады.
- boolean removeLastOccurrence (Object obj): кезектен шыққан соңғы obj элементті жояды. Егер жою орын алса, true мәнін әйтпесе false мәнін қайтарады.

Осылайша, pop және push әдістерінің болуы осы элементті іске асыратын кластарға стек ретінде әрекет етуге мүмкіндік береді. Сонымен қатар, қолданыстағы функционалдылық екі бағытты кезек құруға мүмкіндік береді, бұл интерфейсті қолдана отырып класты көп мүмкіндікті етеді.

ArrayDeque класы. Java-да кезектерді бірнеше кластар ұсынады. Олардың бірі - **ArrayDeque<E>** класы. Бұл класс AbstractCollection класынан функционалдылықты иеленетін және Deque интерфейсін қолдана отырып, жалпыланған екі бағытты кезекті білдіреді.

ArrayDeque класында келесі конструкторлар анықталған:

- ArrayDeque (): бос кезек жасайды
- ArrayDeque(Collection<? extends E> col): col коллекция элементтерінен толтырылған кезекті құрады;
- ArrayDeque(int capacity): бастапқы capacity сыйымдылығы бар кезек жасайды. Егер біз бастапқы сыйымдылықты нақты көрсетпесек, онда әдепкі сыйымдылық 16 тең.

Кластық қолдану мысалы:

```java
import java.util.ArrayDeque;

public class Program{

    public static void main(String[] args) {

        ArrayDeque<String> states = new ArrayDeque<String>();
        // стандартное добавление элементов
        states.add("Germany");
        states.addFirst("France"); // добавляем элемент в самое начало
        states.push("Great Britain"); // добавляем элемент в самое начало
        states.addLast("Spain"); // добавляем элемент в конец коллекции
        states.add("Italy");

        // получаем первый элемент без удаления
        String sFirst = states.getFirst();
        System.out.println(sFirst);     // Great Britain
        // получаем последний элемент без удаления
        String sLast = states.getLast();
        System.out.println(sLast);      // Italy

        System.out.printf("Queue size: %d \n", states.size());  // 5

        // перебор коллекции
        while(states.peek()!=null){
            // извлечение с начала
            System.out.println(states.pop());
        }

         // очередь из объектов Person
        ArrayDeque<Person> people = new ArrayDeque<Person>();
        people.addFirst(new Person("Tom"));
        people.addLast(new Person("Nick"));
        // перебор без извлечения
        for(Person p : people){

            System.out.println(p.getName());
        }
    }
}
class Person{

    private String name;
    public Person(String value){

        name=value;
    }
    String getName(){return name;}
}
```

Рахмет!