|  |
| --- |
| **ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ** |
|  |
|  |
|  |
| **Практическое занятие №10.**  **Тема: Программирование и исследование**  **процесса формирования списков на базе коллекций**  **ArrayList и LinkedList** |
|  |
|  |
|  |

**Цель:** получение практических навыков использования коллекций ArrayList и LinkedList и исследование их характеристик на примере формирования списков.

**Методические рекомендации**

**Общие сведения о коллекциях**

В библиотеке коллекций **Java** существует два базовых интерфейса:

* **Collection** - определены основные методы для работы с данными;
* **Map** - описывает коллекцию, состоящую из пар (ключ-значение).

Интерфейс Collection реализуют интерфейсы **List, Set и Queue**

* **List** - пронумерованная, неупорядоченная, допускающая повторяющиеся элементы коллекция;
* **Set** - не упорядоченная, не допускающая повторяющихся элементов, коллекция;
* **Queue** – очередь по принципу FIFO  
  Реализации интерфейса List:  
     - **ArrayList** - инкапсуляция массива;  
     - **LinkedList** - двусвязный список.

Интерфейс **Collection** является базовым для всех коллекций, определяя основной функционал:

|  |  |
| --- | --- |
|  | **public interface Collection<E> extends Iterable<E>{**  **// определения методов**  **}** |

Интерфейс **Collection** является обобщенным и расширяет интерфейс **Iterable**, поэтому все объекты коллекций можно перебирать в цикле по типу **for-each**.

Некоторые из методов интерфейса **Collection**:

* **boolean add (E item)**: добавляет в коллекцию объект item. При удачном добавлении возвращает true, при неудачном – false;
* **voidbclear ():** удаляет все элементы из коллекции;
* **booleanbcontains (Object item):** возвращает true, если объект item содержится в коллекции, иначе возвращает false;
* **boolean isEmpty ():** возвращает true, если коллекция пуста, иначе возвращает false;
* **Iterator<E>iterator ():** возвращает объект Iterator для обхода элементов коллекции;
* **Boolean remove (Objectitem):** возвращает true, если объект item удачно удален из коллекции, иначе возвращается false;
* **int size ():** возвращает число элементов в коллекции;
* **object[] toArray ():** возвращает массив, содержащий все элементы коллекции.

**Класс ArrayList**

По умолчанию в **Java** есть встроенная реализация интерфейса **Collection -** класс **ArrayList**. Класс **ArrayList** представляет обобщенную коллекцию, которая наследует свою функциональность от класса **AbstractList** и применяет интерфейс **List**. Проще говоря, **ArrayList** представляет простой список, аналогичный массиву, за тем исключением, что количество элементов в нем не фиксировано.

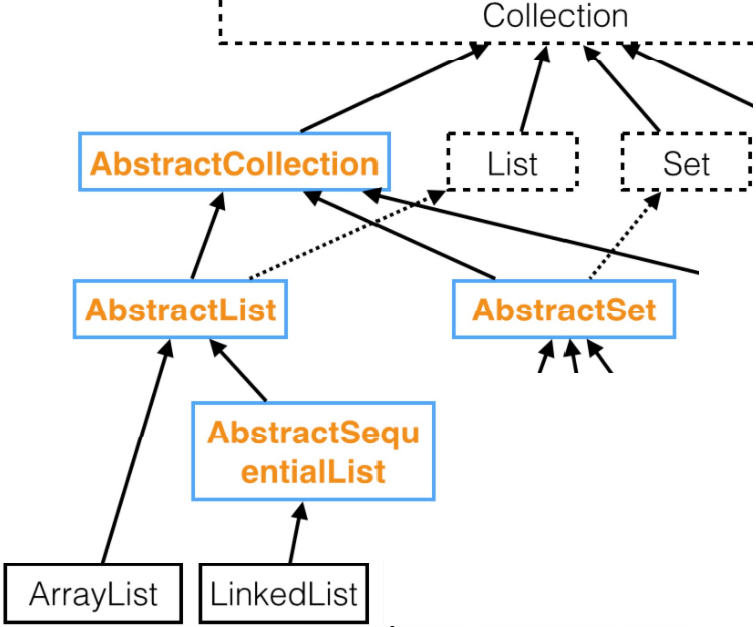


Рис.1. Класс ArrayList в иерархии коллекций (фрагмент)

Пример использования класса ArrayList.

**import java.util.ArrayList;**

**public class Program{**

**public static void main(String[] args) {**

**ArrayList<String>people = new ArrayList<String>();**

**// добавление в список ряда элементов**

**people.add("Tom");**

**people.add("Alice");**

**people.add("Kate");**

**people.add("Sam");**

**people.add(1,"Bob");//добавление элемента по индексу 1**

**// получение 2-го объекта**

**System.out.println(people.get(1));**

**// установка нового значения для 2-го объекта**

**people.set(1,"Robert");**

**System.out.printf("ArrayList has %d elements \n", people.size());**

**for(String person : people){**

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

**// проверка наличия элемента**

**if(people.contains("Tom")){**

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

**}**

**// удаление конкретного элемента**

**people.remove("Robert");**

**// удаление по индексу**

**people.remove(0);**

**Object[] peopleArray = people.toArray();**

**for(Object person : peopleArray){**

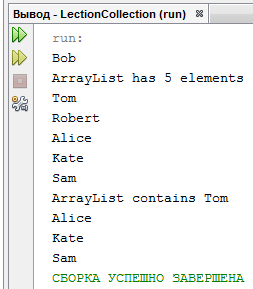
**System.out.println(person);**

**}**

**}**

**}**

Вид экрана:



Здесь объект **ArrayList** типизируется классом **String**, поэтому список будет хранить только строки. Поскольку класс **ArrayList** применяет интерфейс **Collection<E>**, то можно использовать методы данного интерфейса для управления объектами в списке.

Интерфейс Collection содержит объявление метода **iterator()** следующего вида:

**Iterator <E> iterator();**

Метод **iterator()** возвращает объект-итератор, реализующий интерфейс **Iterator**, который используется для последовательного обращения к элементам набора данных.

В интерфейсе Iterator определены следующие три основных метода:

**public interface Iterator<E>**

**{**

**Enext();**

**boolean hasNext();**

**void remove();**

**}**

Реализация интерфейса **Iterator** предполагает, что с помощью вызова метода **next()** можно получить следующий элемент. С помощью **метода  hasNext()** можно узнать, есть ли следующий элемент, и не достигнут ли конец коллекции. И если элементы еще имеются, то **hasNext()** вернет значение true. Метод **hasNext()** следует вызывать перед методом **next(),** так как при достижении конца коллекции метод **next()** генерирует исключение *NoSuchElementException*. И метод  **remove()** удаляет текущий элемент, который был получен последним вызовом **next().**

Пример перебора коллекции.

**import java.util.\*;**

**public class CollectionApp**

**{**

**public static void main(String[] args)**

**{**

**List<String>names = new ArrayList<String>();**

**names.add("Белоконь");**

**names.add("Гончарова");**

**names.add("Дмитриева");**

**names.add("Кашников");**

**Iterator<String> iter = names.iterator();**

**while(iter.hasNext()){**

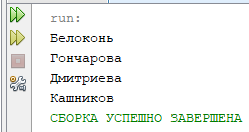
**System.out.println(iter.next());**

**}**

**}**

**}**

Вид экрана:

****

Порядок перебора элементов коллекции зависит от типа и набора элементов. Если используется объект **ArrayList**, то итератор начинает с индекса 0 и увеличивает индекс на 1 на каждом шаге. Если объект имеет тип **HashSet**, то порядок следования элементов коллекции может оказаться случайным.

Интерфейс **Collection** помимо рассмотренных ранее методов, имеет метод **iterator(),** который возвращает итератор для данной коллекции, готовый к ее обходу. С помощью такого итератора можно обработать все элементы любой коллекции следующим способом:

**Iterator iter = coll.iterator();** /\* coll — коллекция, элементы которой обрабатываются\*/

**while (iter.hasNext())** {/\* обработка объекта, возвращаемого

методом **iter.next()**}\*/

Для коллекций, элементы которых проиндексированы, определен более функциональный итератор, позволяющий двигаться как в прямом, так и в обратном направлении, а также добавлять в коллекцию элементы. Такой итератор имеет интрефейс **ListIterator**, унаследованный от интерфейса **Iterator** и дополняющий его следующими методами:

**previous()** — возвращает предыдущий элемент (и делает его текущим);

**hasPrevious()** — возвращает **true**, если предыдущий элемент существует (т.е. текущий элемент не является первым элементом для данного итератора);

**add(Object item)** — добавляет новый элемент перед текущим элементом;

**set(Object item)** — заменяет текущий элемент;

**nextIndex()** и **previousIndex()** — служат для получения индексов следующего и предыдущего элементов соответственно.

В интерфейсе **List** определен метод **listIterator(),** возвращающий итератор **ListIterator** для обхода данного списка.

**Класс LinkedList**

Обобщенный класс **LinkedList<E>** представляет структуру данных в виде связанного списка. Он наследуется от класса **AbstractSequentialList** и реализует интерфейсы **List, Dequeue** и **Queue**. То есть он соединяет функциональность работы со списком и функциональность очереди.

Класс **LinkedList** имеет следующие конструкторы:

**LinkedList():** создает пустой список

**LinkedList(Collection<? extends E> col):** создает список, в который добавляет все элементы коллекции col.

**LinkedList** содержит все те методы, которые определены в интерфейсах **List, Queue, Deque**. Некоторые из них:

**addFirst() / offerFirst():** добавляет элемент в начало списка

**addLast() / offerLast():** добавляет элемент в конец списка

**removeFirst() / pollFirst():** удаляет первый элемент из начала списка

**removeLast() / pollLast():** удаляет последний элемент из конца списка

**getFirst() / peekFirst():** получает первый элемент

**getLast() / peekLast():** получает последний элемент

Пример применения связанного списка:

**import java.util.LinkedList;**

**public class Program{**

**public static void main(String[] args) {**

**LinkedList<String> states = new LinkedList**

**<String>();**

// добавляем в список ряд элементов

**states.add("Germany");**

**states.add("France");**

**states.addLast("Great Britain");** /\* добавляем на

последнее место\*/

**states.addFirst("Spain");**//добавляем на 1-е место

**states.add(1, "Italy");** /\* добавляем элемент по

индексу 1 \*/

**System.out.printf("List has %d elements \n", states.size());**

**System.out.println(states.get(1));**

**states.set(1, "Portugal");**

**for(String state : states){**

**System.out.println(state);**

**}**

// проверка на наличие элемента в списке

**if(states.contains("Germany")){**

**System.out.println("List contains Germany");**

**}**

**states.remove("Germany");**

**states.removeFirst();** // удаление первого элемента

**states.removeLast();** // удаление последнего элемента

**LinkedList<Person> people =new LinkedList<Person>();**

**people.add(new Person("Mike"));**

**people.addFirst(new Person("Tom"));**

**people.addLast(new Person("Nick"));**

**people.remove(1);** // удаление второго элемента

**for(Person p : people){**

**System.out.println(p.getName());**

**}**

**Person first = people.getFirst();**

**System.out.println(first.getName());**/\*вывод первого элемента\*/

**}**

**}**

**class Person{**

**private String name;**

**public Person(String value)**

**name=value;**

**}**

**String getName(){return name;}**

**}**

Здесь создаются и используются два списка: для строк и для объектов класса **Person**. При этом в дополнение к методам **addFirst/removeLast** и т.д., также доступны стандартные методы, определенные в интерфейсе **Collection**: **add(), remove, contains, size** и другие. Поэтому мы можем использовать разные методы для одного и того же действия. Например, добавление в самое начало списка можно сделать так:

**states.addFirst("Spain");**,

а можно сделать так:

**states.add(0,"Spain");.**

**Практикум**

В соответствии с *индивидуальным заданием* (см. приложение) разработать базовый класс и классы-наследники. Сформировать линейный список из 100 000 объектов производных классов.

Разработать меню, обеспечивающее:

* создание списка;
* поиск элемента списка по значению параметра;
* удаление объекта из списка с заданным параметром;
* расчет *Вычисляемого показателя*;
* выход из программы.

Вычислить время выполнения каждой операции со списком при использовании коллекции **ArrayList** и коллекции **LinkedList**. Результаты свести в таблицу и сделать выводы.

|  |  |  |
| --- | --- | --- |
| **Операция** | **Время** | |
| **ArrayList** | **LinkedList** |
| Создание списка |  |  |
| **. . .** |  |  |

Отчет оформляется по общеустановленным правилам в *электронном виде* со следующим содержанием:

1. титульный лист,
2. тема и цель лабораторной работы,
3. задание на лабораторную работу,
4. листинг программы с комментариями,
5. результаты работы программы и
6. выводы с анализом применения коллекций ArrayList и LinkedList при работе со списком.

ПРИЛОЖЕНИЕ

**Варианты заданий**

|  |  |  |  |
| --- | --- | --- | --- |
| **№**  **п/п** | **Базовый класс** | **Производные классы** | **Вычисляемый показатель** |
|  | Компьютер | Настольный компьютер, ноутбук | Экземпляр с наибольшей оперативной памятью |
|  | Товар | Посуда, продукты питания | Сумма покупки |
|  | Предприятие малого бизнеса | Магазин, парикмахерская | Название предприятия с максимальным числом сотрудников |
|  | Представитель университета | Преподаватель, студент | Количество студентов, обучающихся у конкретного преподавателя |
|  | Транспортное средство | Легковой автомобиль, грузовой автомобиль | Максимальная емкость бензобака |
|  | Телефон | Смартфон, кнопочный | Самая новая модель |
|  | Программный продукт | Операционная система, текстовый редактор | Последняя версия |
|  | Документ | Паспорт, студенческий билет | Количество документов на заданную фамилию |
|  | Периферийное устройство компьютера | Принтер, монитор | Минимальная цена устройства |
|  | Строительный товар | Сыпучие материалы, инструмент | Сумма покупки |
|  | Страховой полис | Полис обязательного медицинского страхования, страхования жилища | Количество полисов на заданную фамилию |
|  | Насекомое | Бабочка, жук | Максимальный размер крыльев |
|  | Недвижимость | Таунхаус, квартира в многоквартирном доме | Максимальная жилая площадь |
|  | Часы | Электронные часы, механические часы | Самый дорогой экземпляр |
|  | Товар | Телевизор, холодильник | Количество товаров заданной фирмы |
|  | Магнитная карта для проезда на транспорте | Карта общего назначения для проезда в метро, льготная транспортная карта учащегося | Количество карт без поездок |
|  | Книга | Электронная, на бумаге | Количество книг одного автора |
|  | Канцелярские товары | Тетрадь, авторучка | Количество товаров заданной фирмы |
|  | Периферийное устройство компьютера | Клавиатура, сканер | Количество экземпляров определённой страны-производителя |
|  | Документ | Паспорт, студенческий билет | Документ, выданный раньше всех других |
|  | Путешествие (тур) | Морской круиз, отдых на море | Самый дешевый тур на 7 и более дней |
|  | Товар | Одежда, обувь | Самый дорогой товар |
|  | Страховой полис | Полис обязательного медицинского страхования, страхования жилища | Полис с максимальным сроком действия |
|  | Грузовое транспортное средство | Самосвал, трейлер | Максимальный пробег на полном бензобаке |
|  | Предприятие малого бизнеса | Магазин, парикмахерская | Название предприятия с максимальным числом сотрудников |
|  | Представление | Театр, кино | Минимальное число зрителей в зале |
|  | Запоминающее устройство | Флэш-карта, карта памяти | Экземпляры с максимальным и минимальным размером |
|  | Принтер | Струйный, лазерный | Представитель с наибольшей производительностью |
|  | Компьютер | Настольный компьютер, ноутбук | Самый дешевый компьютер |
|  | Грузовое транспортное средство | Самосвал, трейлер | Максимальный пробег на полном бензобаке |