## Коллекции. Их интерфейсы и классы. Алгоритмы коллекций

Коллекции – это современная технология Java, предназначенная для работы с наборами объектов. Коллекции хранят ссылки на объекты. Если же возникает потребность в использовании простых типов в коллекциях, то необходимо воспользоваться классами-оболочками. Эта технология помогает использовать традиционные структуры данных такие, как списки, деревья, очереди и т.д. для создания надежных и эффективных программ.

Понятие коллекция включает две части: структуру и алгоритмы. Структура стандартизирует способы обработки групп объектов. Существует несколько различных типов классов-коллекций. Все они разработаны, по возможности следуя единой логике и определенным интерфейсам, и там где это возможно, манипулирование ими унифицировано. Однако все коллекции отличаются внутренними механизмами хранения, скоростью доступа к элементам, потребляемой памятью и другими деталями. Например, в некоторых коллекциях объекты или их еще называют элементы, могут быть упорядочены, в некоторых - нет. В некоторых типах коллекций допускается дублирование ссылок на объект, в других - нет. Способ организации данных внутри класса имеет очень большое значение для реализации методов, а также для достижения максимальной производительности.

Классы и интерфейсы обеспечивающие манипулирование коллекциями объектов, сведены в пакете java.util. Общие методы поведения коллекций определены в интерфейсах, представляющих следующую иерархию

Интерфейс *Collection* является корнем всей иерархии интерфейсов коллекций. Он определяет базовую функциональность любой коллекции - набор методов, которые позволяют добавлять элементы *add()*, удалять *remove()*, определять, существует ли некоторый элемент в коллекции *contains()*, размер коллекции *size()* и другие. Классы, которые имплементируют интерфейс *Collection*, могут содержать дубликаты и пустые (*null*) значения.

Интерфейс *List* определяет поведение коллекций, которые содержат упорядоченную последовательность объектов. Они хранятся в том порядке, в котором были добавлены. В этом интерфейсе определены методы, позволяющие вставлять *add()* или извлекать *get()* элементы с помощью их позиций в списке через отсчитываемый от нуля индекс.

Интерфейс *Set* определяет поведение коллекций, которые не допускают дублирования элементов. В коллекции этого типа допускается наличие только одной ссылки типа *null*. Интерфейс *Set* расширяет интерфейс *Collection* т.о. любой класс, имплементирующий *Set*, реализует все методы определенные в *Collection*. Любой объект, добавляемый в *Set*, должен реализовать метод *equals()* для того, что бы его можно было сравнить с другими.

Интерфейс *SortedSet* расширяет *Set* и объявляет поведение набора, отсортированного в возрастающем порядке. Он определяет методы, которые позволяют получить первый *first()* и последний *last()* элементы коллекций, подмножества наборов данного типа *headSet()*, *subSet()*, *tailSet()*.

Классы коллекций представляют иерархию следующего вида

*AbstractCollection*

*AbstractList*

*AbstractSequentialList*

*ArrayList*

*AbstractSet*

*HashSet*

*TreeSet*

*LinkedList*

Концевые классы этого дерева вполне готовы к использованию.

Класс *ArrayList* – список массива - поддерживает динамические массивы, которые могут расти по мере необходимости. Список массива создается с некоторым начальным размером. Когда этот размер превышается, коллекция автоматически расширяется. Когда объекты удаляются, массив может быть сокращен.

*ArrayList(Collection c)*

*ArrayList(int capacity)*

Класс *LinkedList* обеспечивает структуру данных связного списка и имеет два конструктора:

*LinkedList()*

*LinkedList(Collection c)*

Класс *HashSet* создает коллекцию, которая использует хэш-таблицу для хранения коллекций. Преимущество хэширования состоит в том, что оно сохраняет постоянным время выполнения основных операций: добавление, удаление, поиск элементов и т.д. Этот класс имеет несколько конструкторов.

*HashSet(int capacity, float ratio)*

где capacity – емкость хэш-таблицы, а ratio - ее коэффициент заполнения, представляет наибольший интерес.

Класс *TreeSet* использует древовидную структуру для хранения данных. Объекты хранятся в сортированном по возрастанию порядке. Один из конструкторов этого класса

*TreeSet(Comparator comp)*

позволяет создавать набор, элементы которого будут сортироваться в соответствии с компаратором comp.

Для последовательного прохождения элементов коллекций используется специальный объект, который реализует один из интерфейсов: либо Iterator, либо *ListIterator* для коллекций расширяющих абстрактный класс *AbstractList*. Итератор *Iterator* позволяет проходить коллекцию в прямом направлении, списковый же итератор *ListIterator* дает возможность обращаться с коллекцией в прямом и обратном направлении.

Порядок расположения элементов в отсортированных коллекциях определяет объект типа Comparator. Это интерфейс, в котором объявлен метод

*int compare(Object ob1, Object ob2).*

Предполагается, что этот метод возвращает 1, если *ob1>ob2*, если же *ob1<ob2*, то он возвращает -1 и, в случае если объекты равны, возвращаемое значение метода будет нулевым. Элементы коллекции располагаются по возрастанию согласно определенному компаратору.

Алгоритмы – другая важная часть механизма коллекций. Алгоритмы определены как статические методы в классе *Collections* и обеспечивают стандартные средства манипулирования коллекциями: нахождения максимального *max()* и минимального *min()* элементов, изменение порядка расположения элементов *reverse()*, *shuffle()*, упорядочение элементов согласно компаратору *sort()*.

**Вариант 1.**

Результаты сдачи сессии хранятся в связном списке (*LinkedList*). Элементами связного списка являются объекты класса *Student*. Информация о каждом студенте содержит фамилию и массив отметок. Сформировать другой связный список, каждый объект которого содержит фамилию студента и средний балл. Распечатать его на экран.

**Вариант 2.**

В файле “*a.txt*” записан список студентов, сдававших экзамен, с указанием фамилии, группы и отметки. Информация о каждом студенте хранится в отдельной строке. Прочитать информацию в связный список (*LinkedList*), отсортировать его по убыванию отметки, распечатать на экран сначала список студентов, получивших отлично, затем – хорошо и т.д.

**Вариант 3.**

В файле “*a.txt*” записано несколько строк. Каждая строка содержит название государства с указанием через пробел его столицы. Сформировать коллекцию типа *TreeSet*, отсортированную по названию столиц. Распечатать коллекцию на экран.

**Вариант 4.**

В файле “*a.txt*” записан список студентов, сдававших экзамен, с указанием фамилии, группы и отметки. Информация о каждом студенте хранится в отдельной строке. Прочитать информацию в связный список (*LinkedList*), отсортировать его, расположив фамилии по алфавиту и распечатать список на экран.

**Вариант 5.**

В файле “*a.txt*” записан список студентов, сдававших экзамен, с указанием фамилии, группы и отметки. Информация о каждом студенте хранится в отдельной строке. Прочитать информацию в связный список *LinkedList*, разбить его на четыре списка типа *LinkedList*, в каждом из которых находятся студенты, получившие одинаковые отметки. Списки распечатать на экран.

**Вариант 6.**

В файле “*a.txt*” записано несколько строк. Каждая строка содержит название государства с указанием через пробел его столицы. Сформировать коллекцию типа *LinkedList*, отсортированную по названию государства. Распечатать ее на экран в виде таблицы с заголовками «государство», «столица».

**Вариант 7.**

В файле “*a.txt*” записано несколько строк. Каждая строка содержит название государства с указанием через символ ”–“ его столицы. Сформировать коллекцию типа *LinkedList*, отсортировать ее по количеству букв в названии государств и распечатать ее на экран.

**Вариант 8.**

В файле “*a.txt*” хранится информация о рейсах, выполняемых в аэропорту.

Информация о каждом рейсе записана в отдельной строке в виде:

номер рейса, пункт назначения, время вылета, номер выхода.

В файл “*b*.*txt*” записать табло вылетов на 5 рейсов, отсортировав информацию по времени вылета, начиная с данного момента.

**Вариант 9.**

В файле “*a.txt*” хранится информация о рейсах, выполняемых в аэропорту.

Информация о каждом рейсе записана в отдельной строке в виде:

номер рейса, пункт назначения, время вылета, номер выхода.

Прочитать информацию в связный список *LinkedList*, разбить его на списки типа *LinkedList*, в каждом из которых находятся рейсы с одинаковыми пунктами назначения. Списки распечатать на экран.

**Вариант 10.**

В файле “*a.txt*” записан список студентов, сдававших экзамены, с указанием фамилии, группы и отметок. Информация о каждом студенте хранится в отдельной строке. Прочитать информацию в связный список (*LinkedList*). Создать две коллекции типа *TreeSet*. Элементами *TreeSet* являются объекты, содержащие фамилию и средний балл студента. В первой коллекции списки отсортировать по возрастанию среднего балла во второй – по убыванию. Новые коллекции распечатать на экран.