# Коллекции

## Понятие коллекции

Для хранения большого количества однотипных данных могут использоваться массивы, но они не всегда являются идеальным решением. Во-первых, длина массива задается заранее и в случае, если количество элементов заранее неизвестно, придется либо выделять память «с запасом», либо предпринимать сложные действия по переопределению массива. Во-вторых, элементы массива имеют жестко заданное размещение в его ячейках, поэтому, например, удаление элемента из массива не является простой операцией.

В программировании давно и эффективно используются такие структуры данных как очередь, список, множество, map и т.д., объединенные общим названием коллекция. Коллекция — это группа элементов с операциями добавления, извлечения и поиска элемента. Механизм работы операций существенно различается в зависимости от типа коллекции. Например, очередь позволяет получить лишь первый элемент, т.к. элементы добавляются в один конец последовательности, а «забираются» с другого. Список, же наоборот, позволяет получить элемент из любого места последовательности, а множество вообще не упорядочивает элементы и позволяет (помимо добавления и удаления) только узнать, содержится ли в нем данный элемент.

Чтобы выбрать коллекцию, которая лучше всего подходит условию задачи, необходимо знать особенности каждой из них.

## Классы-коллекции

Язык Java предоставляет библиотеку стандартных коллекций, которые собраны в пакете java.util

Коллекции в данной библиотеке представлены набором интерфейсов и реализующих их классов.

Каждый класс реализует некоторую коллекцию со специфичным для нее набором операций доступа к элементам. Чтобы использовать коллекцию в своей программе, нужно создать объект соответствующего класса.

Элементы большинства коллекций имеют тип Object. Это значит, что (в отличие от обычного массива) вы не должны заранее указывать тип элементов, которые будете помещать в коллекцию. Вы можете добавлять в нее объекты любого класса (за исключение примитивов), поскольку все классы являются наследниками класса Object, более того — в одной коллекции могут храниться объекты совершенно разных классов.

Конечно, это может привести и к трудностям, если вы захотите совершать какие-то операции над элементом коллекции, вы не сможете воспользоваться его методами, не приведя объект к его «настоящему» классу посредством явного приведения типов (а вы помещаете в коллекцию объекты именно для того, чтобы потом их извлекать и обрабатывать).

Это не является проблемой, просто нужно помнить, объекты какого класса вы помещаете в коллекцию и выполнять соответствующее преобразование типа над каждым побывавшим в коллекции элементом.

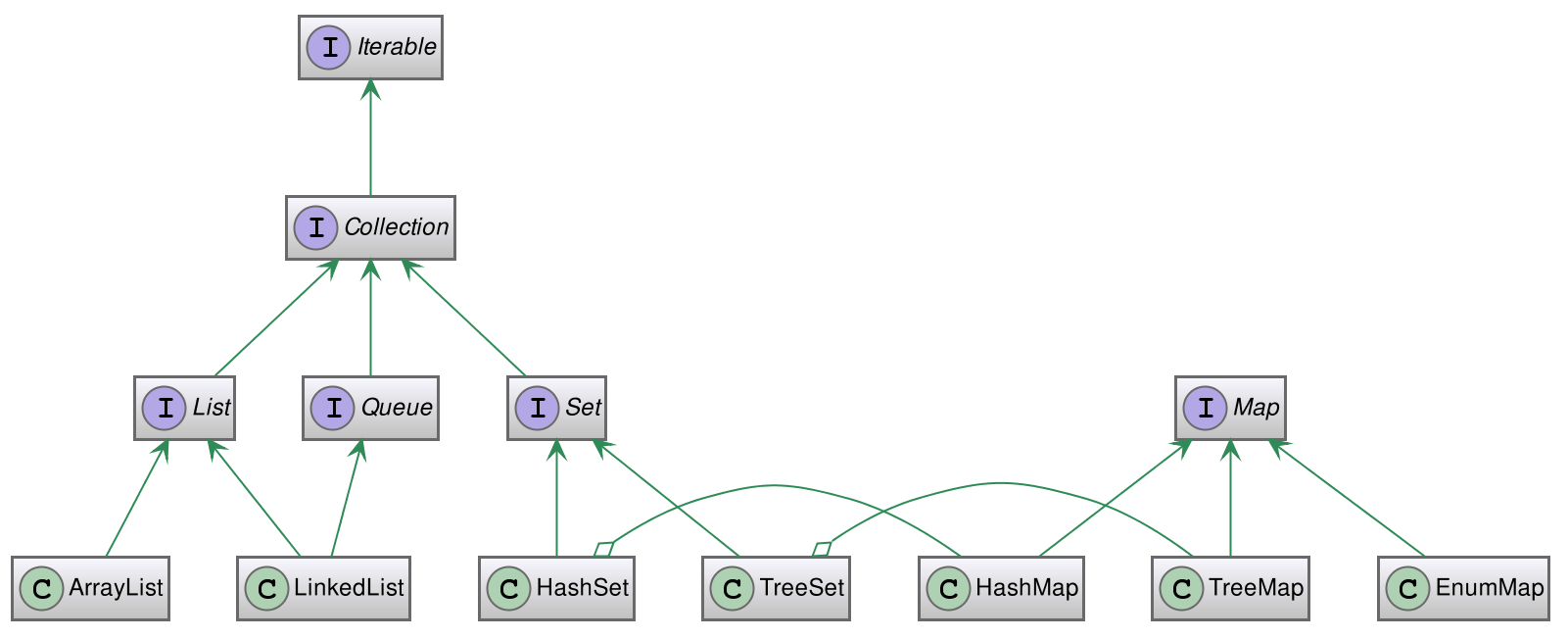
Но если попытаться привести объект к неправильному типу, возникнет ошибка в программе.

Так что, воспользоваться возможностью помещать в коллекцию любые объекты, скорее всего, не получится. Представьте себе, что ваша программа должна «помнить», что первый элемент списка — строка, второй — число, третий — собака, четвертый — опять число. И обработать их все в цикле она, наверное, не сможет, поскольку у этих объектов нет общих методов. В результате теряются все преимущества использования коллекции, поэтому в каждую коллекцию следует помещать объекты только одного класса (и производных от него).

## Интерфейсы коллекции

Ключевое слово interface используется для создания полностью абстрактных классов. Создатель интерфейса определяет имена методов, списки аргументов и типы возвращаемых значений, но не тела методов. Интерфейс может использоваться в качестве типа данных так же, как и класс. Разница лишь в том, что объекты интерфейсного типа нельзя создать напрямую — необходимо выбрать класс, поддерживающий этот интерфейс и вызвать его конструктор.

## Иерархия наследования



## Паттерн проектирования Iterator

В программировании существует несколько испытанных временем и детально проработанных приемов структурной организации программы, называемых паттернами (шаблонами) проектирования.

Один из таких паттернов называется Iterator. Идея заключается в том, что к коллекции «привязывается» объект, единственное назначение которого — выдать все элементы этой коллекции в некотором порядке, не раскрывая ее внутреннюю структуру.

В пакете java.util описан интерфейс Iterator, воплощающий этот паттерн проектирования. Он имеет всего три метода:

* next() возвращает очередной элемент коллекции, к которой «привязан» итератор (и делает его текущим). Порядок перебора определяет сам итератор.
* hasNext() возвращает true, если перебор элементов еще не закончен
* remove() удаляет текущий элемент

## Класс ArrayList

Класс ArrayList - реализация интерфейса List, которая инкапсулирует в себе обычный массив, длина которого автоматически увеличивается при добавлении новых элементов. Так как ArrayList использует массив, то время доступа к элементу по индексу минимально (в отличие от LinkedList).

При удалении произвольного элемента из списка, все элементы находящиеся «правее» смещаются на одну ячейку влево, при этом реальный размер массива (его емкость, *capacity*) не изменяется. Если при добавлении элемента, оказывается, что массив полностью заполнен, будет создан новый массив размером (n \* 3) / 2 + 1, в него будут помещены все элементы из старого массива + новый, добавляемый элемент.

## Класс LinkedList

LinkedList - реализация интерфейсов List и Deque, которая инкапсулирует в себе двусвязный список. Это структура данных, состоящая из узлов, каждый из которых содержит как собственно данные, так и  две ссылки («связки») на следующий и предыдущий узел списка.

Доступ к произвольному элементу осуществляется за линейное время.

В целом же, LinkedList в абсолютных величинах проигрывает ArrayList и по потребляемой памяти и по скорости выполнения операций.

## Класс HashSet

HashSet - реализация интерфейса Set, инкапсулирующая в себе объект HashMap, то есть использует для хранения хэш-таблицу.

Хеш-таблица хранит информацию, используя так называемый механизм хеширования, в котором содержимое ключа используется для определения уникального значения, называемого хеш-кодом. Этот хеш-код затем применяется в качестве индекса, с которым ассоциируются данные, доступные по этому ключу. Преобразование ключа в хеш-код выполняется автоматически — вы никогда не увидите самого хеш-кода. Также ваш код не может напрямую индексировать хеш-таблицу. Выгода от хеширования состоит в том, что оно обеспечивает константное время выполнения методов *add*, *contains*, *remove* и *size*, даже для больших наборов.

Реализует интерфейс Set.

## Класс TreeSet

TreeSet - реализация интерфейса Set, которая инкапсулирует в себе TreeMap, который в свою очередь использует красно-черное дерево для хранения элементов.

TreeSet хорош тем, что для операций add, remove и contains потребуется гарантированное время log(n).

## Интерфейс Map

Интерфейс Map представляет собой коллекцию пар уникальный ключ-значение. Ключ — это объект, который вы используете для последующего извлечения данных.

Задавая ключ и значение, вы можете помещать значения в объект карты. После того как это значение сохранено, вы можете получить его по ключу.

## Класс HashMap

HashMap - реализация интерфейса Map, основанная на хэш-таблицах со списком переполнения.

Ключи и значения в данной коллекции могут быть любых типов, в том числе и null. Данная реализация не дает гарантий относительно порядка элементов с течением времени.

Если Вы хотите использовать HashMap для хранения объектов СВОИХ классов, то вы ДОЛЖНЫ переопределить методы hashCode() и equals(), иначе два логически-одинаковых объекта будут считаться разными, так как при добавлении элемента в коллекцию будет вызываться метод hashCode() класса Object (который скорее-всего вернет разный хэш-код для ваших объектов).  
Важно отметить, что класс HashSet не гарантирует упорядоченности элементов, поскольку процесс хеширования сам по себе обычно не порождает сортированных наборов. Если вам нужны сортированные наборы, то лучшим выбором может быть другой тип коллекций, такой как класс TreeSet.

## Класс TreeMap

TreeMap - реализация интерфейса Map, использующая красно-черное дерево для хранения элементов, за счет такой структуры объекты сохраняются в отсортированном порядке.

Время доступа и извлечения элементов достаточно мало, что делает класс TreeMap блестящим выбором для хранения больших объемов отсортированной информации, которая должна быть быстро найдена.

В случае, когда к коллекции Map не выставляется требование отсортированности элементов HashMap выигрывает TreeMap в абсолютных величинах.

## Устаревшие коллекции

Следующие коллекции являются устаревшими, и их использование не рекомендуется, но не запрещается:

1. Enumeration — аналог интерфейса Iterator.

2. Vector — аналог класса ArrayList; поддерживает упорядоченный список элементов, хранимых во "внутреннем" массиве.

3. Stack — класс,  производный от Vector,  в который добавлены методы вталкивания (push) и выталкивания (pop) элементов,  так что список может трактоваться в терминах, принятых для описания структуры данных стека (stack).

4. Dictionary — аналог интерфейса Map, хотя представляет собой абстрактный класс, а не интерфейс.

5. Hashtable — аналог HashMap.

Все методы Hashtable, Stack, Vector являются синхронизированными, что делает их менее эффективными в одно поточных приложениях.

## Синхронизированные коллекции

Получить синхронизированные объекты коллекций можно с помощью соответствующих методов утилитного класса Collections:

Map map = Collections.synchronizedMap(new HashMap());

List list = Collections.synchronizedList(new ArrayList());

Начиная с java 5 появились более производительные потокобезопасные реализации коллекций – они находятся в пакете java.util.concurrent, например:

* ConcurrentHashMap
* CopyOnWriteArrayList
* ArrayBlockingQueue
* и другие