Cтруктуры данных Map в Java

**Контейнеры** - это хранилища, поддерживающие различные способы хранения и упорядочения объектов с целью обеспечения возможностей эффективного доступа к ним. Они представляют собой реализацию абстрактных структур данных, поддерживающих три основные операции [1, c. 253]:

* Добавление нового элемента в коллекцию.
* Удаление элемента из коллекции.
* Изменение элемента в коллекции.

Применение контейнеров обусловливается возросшими объемами обрабатываемой информации. В этих условиях массивы не могут удовлетворить всем требованиям.

Контейнеры в Java можно разделить на:

* Коллекции (Collections);
* Отображение вида «ключ-значение» (Map);

Такое разделение обусловлено тем, что все контейнеры кроме Map принадлежат к иерархии классов Collections. Контейнеры Map реализованы отдельно, но они используют коллекции для итерирования.

# Итераторы

**О**бъекты, абстрагирующие за единым интерфейсом доступ к элементам коллекции. Итератор позволяет получить доступ к элементам любой коллекции без вникания в суть ее реализации.

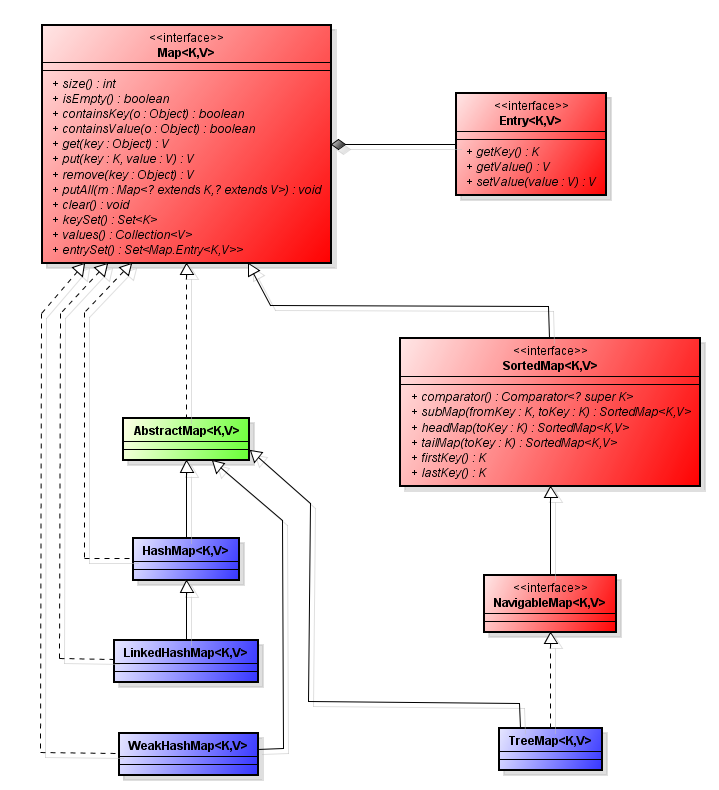
Иерархия итераторов представлена ниже  


Интерфейс **Iterator** позволяет организовать цикл для перебора коллекции получая либо удаляя элементы.

Интерфейс **ListIterator** добавляет к интерфейсу **Iterator** двунаправленный проход по списку и возможность модификации элементов.

# Реализации интерфейса Map

Интерфейс Map соотносит уникальные ключи со значениями. 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.



* **HashMap** - основан на хэш-таблицах, реализует интерфейс Map (что подразумевает хранение данных в виде пар ключ/значение). Ключи и значения могут быть любых типов, в том числе и null. Данная реализация не дает гарантий относительно порядка элементов с течением времени.
* **LinkedHashMap -** расширяет класс HashMap. Он создает связный список элементов в карте, расположенных в том порядке, в котором они вставлялись. Это позволяет организовать перебор карты в порядке вставки. Также можете создать объект класса LinkedHashMap, возвращающий свои элементы в том порядке, в котором к ним в последний раз осуществлялся доступ.
* **TreeMap** - расширяет класс AbstractMap и реализует интерфейс NavigatebleMap. Он создает коллекцию, которая для хранения элементов применяет дерево. Объекты сохраняются в отсортированном порядке по возрастанию. Время доступа и извлечения элементов достаточно мало, что делает класс TreeMap блестящим выбором для хранения больших объемов отсортированной информации, которая должна быть быстро найдена.
* **WeakHashMap** - коллекция, использующая слабые ссылки для ключей (а не значений). Слабая ссылка (англ. weak reference) - специфический вид ссылок на динамически создаваемые объекты в системах со сборкой мусора. Отличается от обычных ссылок тем, что не учитывается сборщиком мусора при выявлении объектов, подлежащих удалению.

Лямбда-выражения в Java

Java изначально полностью объектно-ориентированный язык. За исключением примитивных типов, все в Java – это объекты. Даже массивы являются объектами. Экземпляры каждого класса - объекты.

Поэтому до Java 8 не было возможности определить отдельно **(вне класса)** какую-нибудь функцию. И не было никакой возможности передать метод как аргумент или вернуть тело метода как результат другого метода, **не создавая класс**.

Лямбда-выражения привносят в Java элементы функционального программирования и тесно связаны с **функциональными интерфейсами**. Функциональные интерфейсы (Functional Interface) - это интерфейсы только с одним абстрактным методом, объявленным в нем. **java.lang.Runnable** - это пример функционального интерфейса.

Упрощенно лямбда-выражения можно рассматривать как анонимные методы, которые при этом представляют из себя объект, который можно присваивать переменной и передавать как аргумент в другие методы [2].

# Синтаксис и применение лямбда-выражений

Общая форма: ([parameter\_list])->{lambda\_body}.

* Выражение может не содержать параметров;
* Тело может состоять из одной инструкции, тогда скобки {} не нужны;
* Тело выражения может возвращать значение;
* Если типы могут быть определены компилятором из контекста (выводимые типы), то их можно не указывать явно;

Примеры лямбда-выражений [3, c. 560]:

|  |  |
| --- | --- |
| **Выражение** | **Коментарий** |
| ()->{} | Параметров нет, результат void |
| ()->42 | Параметров нет, тело из одного оператора, результат константа |
| ()->null | Параметров нет, тело из одного оператора, результат null |
| ()->{return 42;} | Параметров нет, тело – блок с возвратом |
| ()->{System.gc()} | Параметров нет, тело блока void |
| ()->{  int total = 0;  for (int number : numbers) {  if (number % 2 == 0) {  total += number;  }  }  return total;  } | Сложное тело блока с возвратом. |
| (int x)->x+1 | Один параметр объявленного типа |
| (x)->x+1 или x->x+1 | Один параметр, выводимый тип |
| (int x, int y)->x+y | Несколько параметров типы заданы явно |
| (x, y)->x+y | Несколько параметров, выводимые типы. |
| (x, int y)->x+y | Ошибка, смешивать выводимые и явные типы нельзя |
| (x, final y)->x+y | Ошибка, модификаторы в выводимыми типами недопустимы. |

1. <http://www.quizful.net/post/Java-Collections>
2. <http://info.javarush.ru/translation/2014/06/14/Lambda-%D0%B2%D1%8B%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F-%D0%BD%D0%B0-%D0%BF%D1%80%D0%B8%D0%BC%D0%B5%D1%80%D0%B0%D1%85.html>
3. Язык программирования Java SE 8. Подробное описание. Джеймс Гослинг, Билл Джой и др.