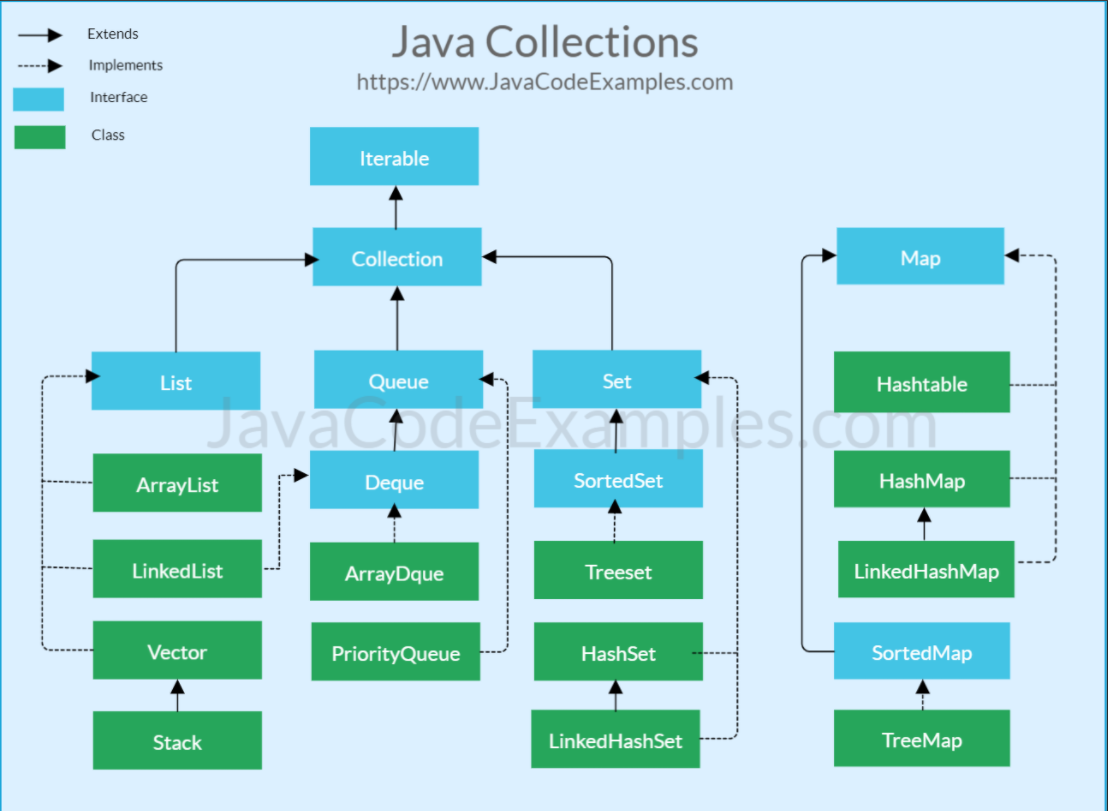
Java Collections Framework



Выше представлен общий вид Java Collections.

1. Все классы коллекции хранятся в пакете (java.util.ArrayList)
2. Все коллекции параметризированы
3. Параметром можно передать только ссылочный тип (примитивы нельзя)
4. При удалении элемента удаление происходит за O(n), так как ArrayList реализован как массив, но удаляя элемент мы сдвигаем все остальные элементы влево, коме случая конца, так как там происходить за O(1)
5. При нехватки ArrayList места, он увеличивается в два раза (плюс 1 к конечному результату), остальные элементы переносятся в новый массив, что делает добавление в массив не за O(1), а примерно за O(1.5)
6. По конвенции при создании коллекции мы ссылаем объекты на интерфейсы:

List<Integer> list1 = new ArrayList<Integer>();

6.1 Ссылаться на класс тоже можно, но тогда в ходе выполнения программы, нельзя будет сменить реализацию интерфейса:

List<Integer> list1 = new ArrayList<Integer>();  
list1 = new LinkedList<Integer>();

Код методов ниже

import java.util.ArrayList;  
import java.util.LinkedList;  
import java.util.List;  
  
public class Start {  
 public static void main(String[] args) {  
  
 List<Integer> list1 = new ArrayList<Integer>();  
 list1 = new LinkedList<Integer>();  
  
 for (int i = 0; i < 100; i++) {  
 list1.add(i);  
 }

// Так можно вывести все элементы ArrayList без цикла

System.*out*.println(list1);  
  
 // Получение элемента по индексу  
 System.*out*.println(list1.get(0));  
 System.*out*.println(list1.get(99));  
  
 // Пройтись по всем элементам ArrayList(не рассматриваем lambda function)  
 for (int i = 0; i < list1.size(); i++) {  
 System.*out*.print(list1.get(i) + " ");  
 }  
  
 System.*out*.println();  
  
 for(Integer elem : list1){  
 System.*out*.print(elem + " ");  
 }  
  
 // Удаление элемента  
 list1.remove(6);  
 }  
}

Как устроен ArrayList внутри

1. ArrayList реализован внутри, как массив, кога место заканчивается создаётся массив примерно в два раза большего размера. Старые элементы из переносятся в новый массив.
2. У класса есть несколько конструкторов, один из которых принимает начальные параметры инициализации массива (так же есть пустой конструктор)
3. При каждом добавлении эл-ма проверяется размер, если места недостаточно, то вызывается метод grow() и возвращается увеличенный массив.
4. При взятии элемента проверяется входит ли переданный индекс рамки размера ArrayList.
5. При удалении remove() Проверям, что удаляемый индекс существует, после удаления сдвигаем все элементы на позицию влево от удалённого.
6. Детально можно почитать официальную документацию или посмотреть реализацию класса в коде.

LinkedList<?>

1. LinkedList или связный список, это коллекция ,которая представлена в виде Node, которые указывают друг на друга, то есть LinkedList не хранится в памяти последовательно, а разбросан по всей памяти.
2. Поэтому при взятии элемента или добавлении элемента в LinkdeList, мы тратим намного больше времени, чем при использовании ArrayList, но если мы удаляем элемент, то удаление произойдёт очень быстро по сравнению с ArrayList, так как мы не двигаем оставшиеся элементы влево(если только это не конец массива), аналогично по скорости выигрывает LinkedList и по добавлению на первую позицию, так как при таком подходе мы не сдвигаем все элементы влево, а просто перебрасываем ссылку(head) на новый элемент.
3. Пример кода представлен ниже

import java.util.ArrayList;  
import java.util.LinkedList;  
import java.util.List;  
  
public class Start {  
  
 public static void measureTime(List<Integer> list) {  
  
 /// Блок сравнения добавлений элементов в list  
 long start1 = System.*currentTimeMillis*();  
  
 for (int i = 0; i < 100\_000; i++) {  
 list.add(i);  
 }  
  
 long end1 = System.*currentTimeMillis*();  
 System.*out*.println("Add elem: " + list.getClass() + " " + (end1 - start1));  
  
 /// Блок сравнения взятия элемента из list  
 long start2 = System.*currentTimeMillis*();  
  
 for (int i = 0; i < 100\_000; i++) {  
 list.get(i);  
 }  
  
 long end2 = System.*currentTimeMillis*();  
 System.*out*.println("Get elem: " + list.getClass() + " " + (end2 - start2));  
  
 ///Блок добавления элемента в начало list  
 long start3 = System.*currentTimeMillis*();  
  
 for (int i = 0; i < 100\_000; i++) {  
 list.add(0, i);  
 }  
  
 long end3 = System.*currentTimeMillis*();  
 System.*out*.println("Get pos(0, i) elem: " + list.getClass() + " " + (end3 - start3));  
 }  
  
 public static void main(String[] args) {  
  
 List<Integer> list1 = new ArrayList<>();  
 List<Integer> list2 = new LinkedList<>();  
  
 *measureTime*(list1);  
 *measureTime*(list2);  
  
 }  
}

1. Пример вывода консоли представлен ниже

C:\Users\Duldi\.jdks\openjdk-15\bin\java.exe "-javaagent:A:\IntellijIdeaPro\IntelliJ IDEA 2020.2.3\lib\idea\_rt.jar=65306:A:\IntellijIdeaPro\IntelliJ IDEA 2020.2.3\bin" -Dfile.encoding=UTF-8 -classpath E:\JavaProfessional\Practise\LinkedList\out\production\LinkedList Start

Add elem: class java.util.ArrayList 8

Get elem: class java.util.ArrayList 0

Get pos(0, i) elem: class java.util.ArrayList 2829

Add elem: class java.util.LinkedList 16

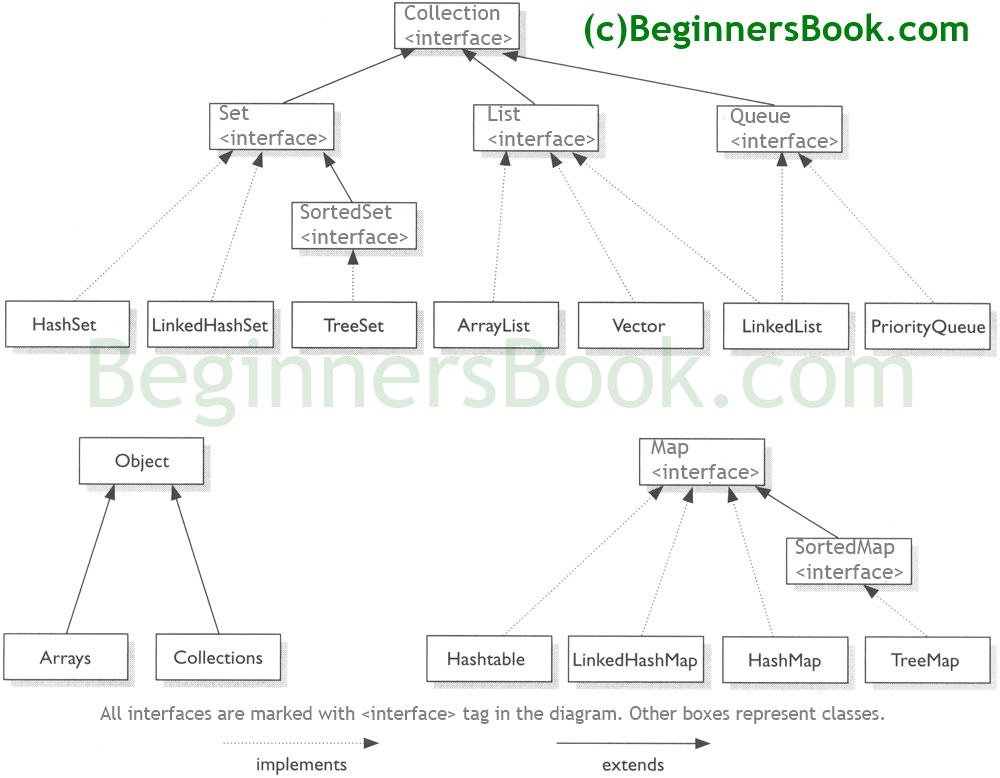
Get elem: class java.util.LinkedList 3748

Get pos(0, i) elem: class java.util.LinkedList 41

Process finished with exit code 0

Собственную реализацию LinkedList см в файле лекций от Yandex

HashMap



Java Collections Framework (схема на стр1 более детальная)

1. При реализации Map<Integer, String> не забываем указывать значения слева, а справа просто можно поставить пустые скобки, если это не сделать, то в Map будут передаваться значения класса Object.

import java.util.HashMap;  
import java.util.Map;  
  
public class Start {  
 public static void main(String[] args) {  
  
 Map<Integer, String> map = new HashMap<>();  
 map.put(1, "один");  
 map.put(2, "два");  
 map.put(3, "три");  
  
 // У map перегружен toString  
 System.*out*.println(map);  
 }  
}

1. Так же стоит отметить, что у Map перегружен метод toString, и выведется не просто имя класса, а набор ключ-значение в Map.
2. Если в Map вставить ключ, который уже существует, то в Map просто поменяется значения при этом ключе.
3. В Map реализован метод get(key), мы получаем значение по ключу, если ключа нет, то вернётся null.

// Метод get(когда значение существует и когда не существует)  
 System.*out*.println(map.get(1));  
 System.*out*.println(map.get(44));  
 }  
}

1. При прохождении по всем элементам Map, используем цикл, в который передаём entryset().

5.1 Так же стоит отметить, что в цикле мы можем получить значение по ключу и сам ключ.

5.2 Так же можно изменить значения при работе с ним в цикле

5.3 Дополнительные методы (узнать есть ли ключ в map или значение) представлены ниже.

import java.util.HashMap;  
import java.util.Map;  
  
public class Start {  
 public static void main(String[] args) {  
  
 Map<Integer, String> map = new HashMap<>();  
 map.put(1, "один");  
 map.put(2, "два");  
 map.put(3, "три");  
 map.put(3, "Новое значние");  
  
  
 // Пройтись по всем элементам Map<>  
 for(Map.Entry<Integer, String> entry : map.entrySet()){  
 System.*out*.println(entry.getKey() + " : " + entry.getValue());  
 entry.setValue("six");  
 }  
  
 System.*out*.println(map.containsKey(1));  
 System.*out*.println(map.containsValue("два"));  
 System.*out*.println(map.hashCode());  
   
 }  
}

HashMap не сортирует элементы, они могут располагаться и выводиться в случайном порядке. HashMap надо использовать, когда неважен порядок.

Если порядок важен, то надо использовать другие классы, реализующие интерфейс Map