ЗАНЯТИЕ 20

ЕСТЬ ПРОБЛЕМКА...

- Student[] s = new Student[5];
- Массивы эффективны в некоторых случаях, но есть некоторые ограничения:
- 1) Массивы имеют фиксированный размер, т.е. после создания массива с определенным размером размер массива не увеличивается или уменьшается
- 2) Массив хранит только «однородные» данные:

```
Object[] array = new String[3];
array[0] = "a";
array[1] = 1;   // throws java.lang.ArrayStoreException
```

- 3) Массив не создан «по стандарту» каких-то структур данных, у него нет методов

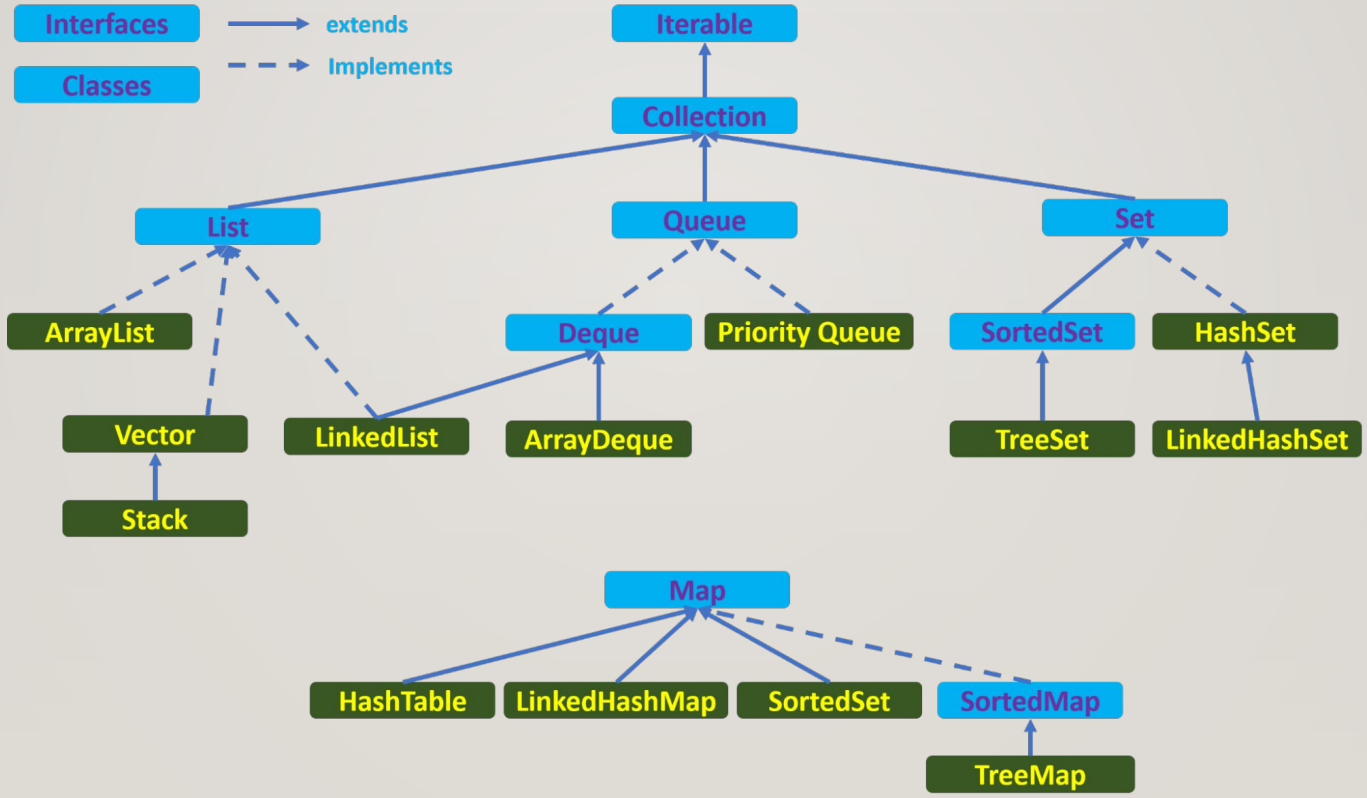
НАДО ЧТО-ТО С ЭТИМ ДЕЛАТЬ...

- Для преодоления этих недостатков или ограничений массива нам нужен Collection Framework (фреймворк состоит из интерфейсов, их реализаций и утилитарных классов для работы):
- 1) Не думаем о размерах
- 2) Можем хранить «неоднородные» данные
- 3) Существует огромное количество методов

ЧТО ТАКОЕ КОЛЛЕКЦИЯ?

- Коллекция – хранилище, контейнер объектов с различными способами накопления и упорядочивания. Это абстрактная структура данных, которая поддерживает три основные операции: 1)добавление, 2)удаление, 3)изменения элемента в (из) коллекции.
- Коллекции располагаются в пакете java.util

ИЕРАРХИЯ


Interfaces
Classes
extends
Implements
Iterable
Collection
List
Queue
Set
ArrayList
Deque
Priority Queue
SortedSet
HashSet
Vector
LinkedList
ArrayDeque
TreeSet
LinkedHashSet
Stack
Map
HashTable
LinkedHashMap
SortedSet
SortedMap
TreeMap

ITERABLE

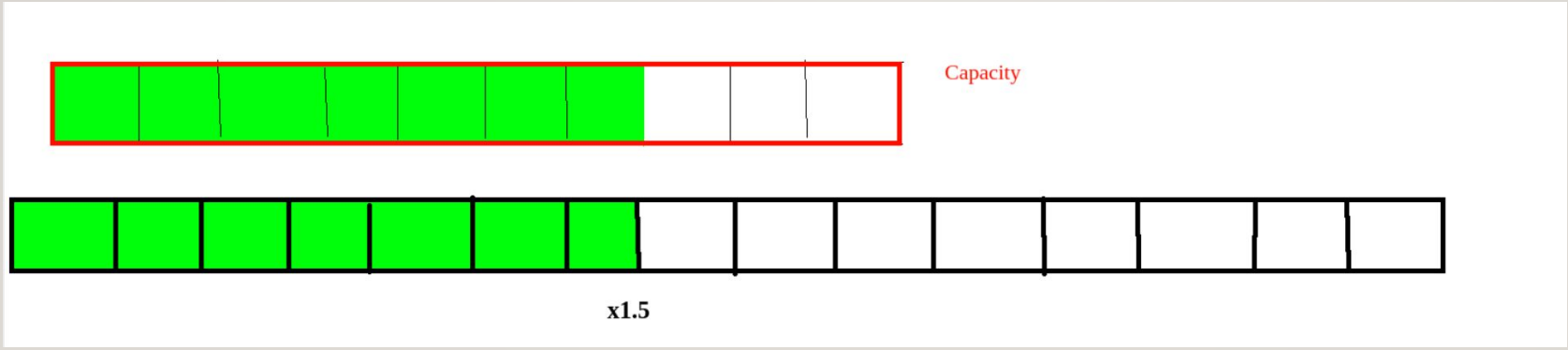
- Iterable – самый главный интерфейс. Задает возможность итерироваться (перебирать) элементы ("for-each loop")
- Интерфейс Iterable содержит один метод iterator() (с Java 8 добавились еще 2).

LIST

- List используется для хранения упорядоченных элементов (могут быть одинаковые).
- Например, последовательность букв в слове: буквы могут повторяться, при этом их порядок важен.
- Основные реализации List: ArrayList, LinkedList

ARRAYLIST

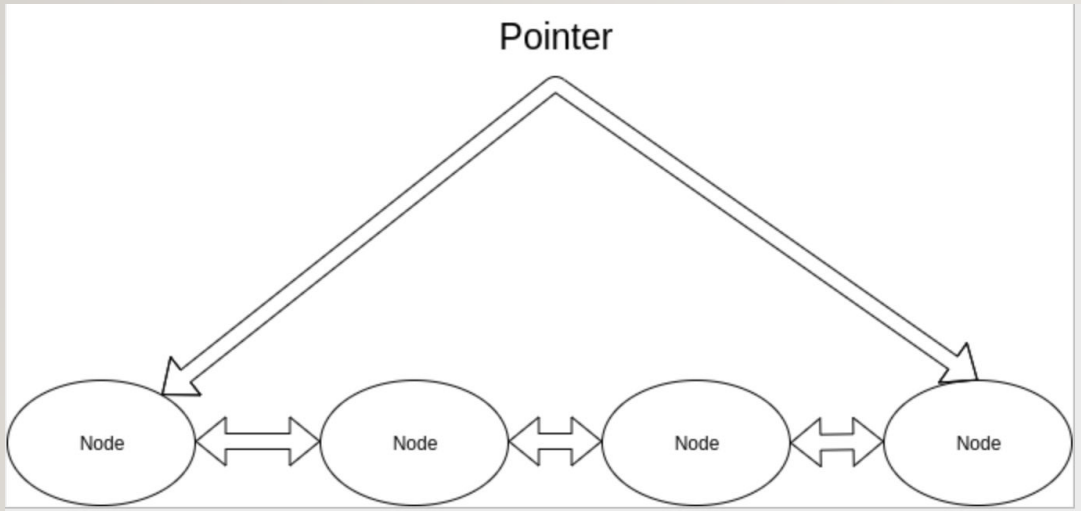
- Строится на базе обычного массива. Если при создании не указать размерность, то под значения выделяется 10 ячеек. При попытке добавить элемент, для которого места уже нет, массив автоматически расширяется — программисту об этом специально заботиться не нужно.



Point_1/ArrayListExample

LINKEDLIST

- Класс **LinkedList** реализует одновременно **List** и **Deque**. Это список, в котором у каждого элемента есть ссылка на предыдущий и следующий элементы:



Благодаря этому добавление и удаление элементов выполняется быстро — времязатраты не зависят от размера списка, так как элементы при этих операциях не сдвигаются: просто перестраиваются ссылки.

Point_1/LinkedListExample

AL VS LL

- если добавлять и удалять элементы с произвольными индексами в списке нужно чаще, чем итерироваться по нему, то лучше **LinkedList**. В остальных случаях — **ArrayList**.

QUEUE, DEQUE («ДЭК»)

- Очередь. В таком списке элементы можно добавлять только в хвост, а удалять — только из начала. Так реализуется концепция **FIFO** (**first in, first out)** — «первым пришёл — первым ушёл». (как в магазине ☺). А ещё есть **LIFO (last in, first out)**, то есть «последним пришёл — первым ушёл». Пример — стопка рекламных буклетов на ресепшене отеля: первыми забирают самые верхние (положенные последними). Структуру, которая реализует эту концепцию, называют стеком.
- Deque может выступать и как очередь, и как стек. Это значит, что элементы можно добавлять как в её начало, так и в конец. То же относится к удалению.

ОПЕРАЦИИ С QUEUE

- 1. **add()** - добавляет элемент в конец очереди.
- 2. **remove()** и **poll()** - удаляет верхний элемент из очереди.
- 3. **offer()** - пытается вставить элемент в конец очереди («предлагает", ибо в очередях с фиксированным размером не получится вставить).
- 4. **peek()** и **element()** - показывают верхний элемент очереди
- Будем рассматривать такую реализации Queue как **PriorityQueue** - элементы в ней располагаются не только по порядку, но и **по приоритету**. Этот приоритет можно задавать самостоятельно, но пока Вам стоит знать, что если создавать очередь из стандартных типов данных (**Integer**, **Float**, **String**), у нее будет стандартный приоритет:
- для чисел по возрастанию
- для строк по алфавиту

- Point_2: QueueExample

SET

- Коллекция, которая не содержит повторяющиеся элементы. Это неупорядоченное множество уникальных элементов.
- Например, мешочек с бочонками для игры в лото: каждый номер от 1 до 90 встречается в нём ровно один раз, и заранее неизвестно, в каком порядке бочонки вынут при игре.
- Основные реализации: TreeSet, HashSet

HASHSET

- Класс **HashSet** использует для хранения данных в хеш-таблице (структура данных, в которой все элементы помещаются в бакеты(buckets - корзины), соответствующие результату вычисления хеш-функции). Это значит, что при манипуляциях с элементами используется хеш-функция — **hashCode()** в Java.
- Добавление, поиск и удаление элементов при такой организации происходит за постоянное время, независимо от числа элементов в коллекции.
- Point_3/subpoint_1/ HashSetExample.java

TREESET

- О классе **TreeSet** вспоминают в тех случаях, когда множество должно быть упорядочено. Каким образом упорядочивать — определяет разработчик при создании нового **TreeSet**. По умолчанию элементы располагаются в естественном порядке.
- Работает по принципу красно-черного дерева. Если ну очень интересно, то https://ru.wikipedia.org/wiki/Красно-чёрное_дерево
- Point_3/subpoint_2/ TreeSetExample
- А как быть с объектами? https://metanit.com/java/tutorial/5.6.php
- Point_3/subpoint_2

ПРАКТИКА

- Task 1
  Вход на вечеринку только по списку.
  Есть список имен (коллекция). Пользователь вводит с консоли свое имя, а программа проверяет его наличие в списке, после чего говорит может он пройти или нет
- Task 2
  Есть TreeSet чисел, нужно отсортировать его в обратном порядке
- Task 3
- Есть HashSet магазинов. У магазина есть название и ID. Создать несколько магазинов (через оператор new Shop(..)) с одинаковым ID и названием и добавить их в ваш Set.
- Task 4
- Есть TreeSet имен, необходимо выбрать все от "H" до "W" (есть 2 способа: 1 – через использование 1 метода, а 2 – просто перебирая. Выбирайте по душе☺)