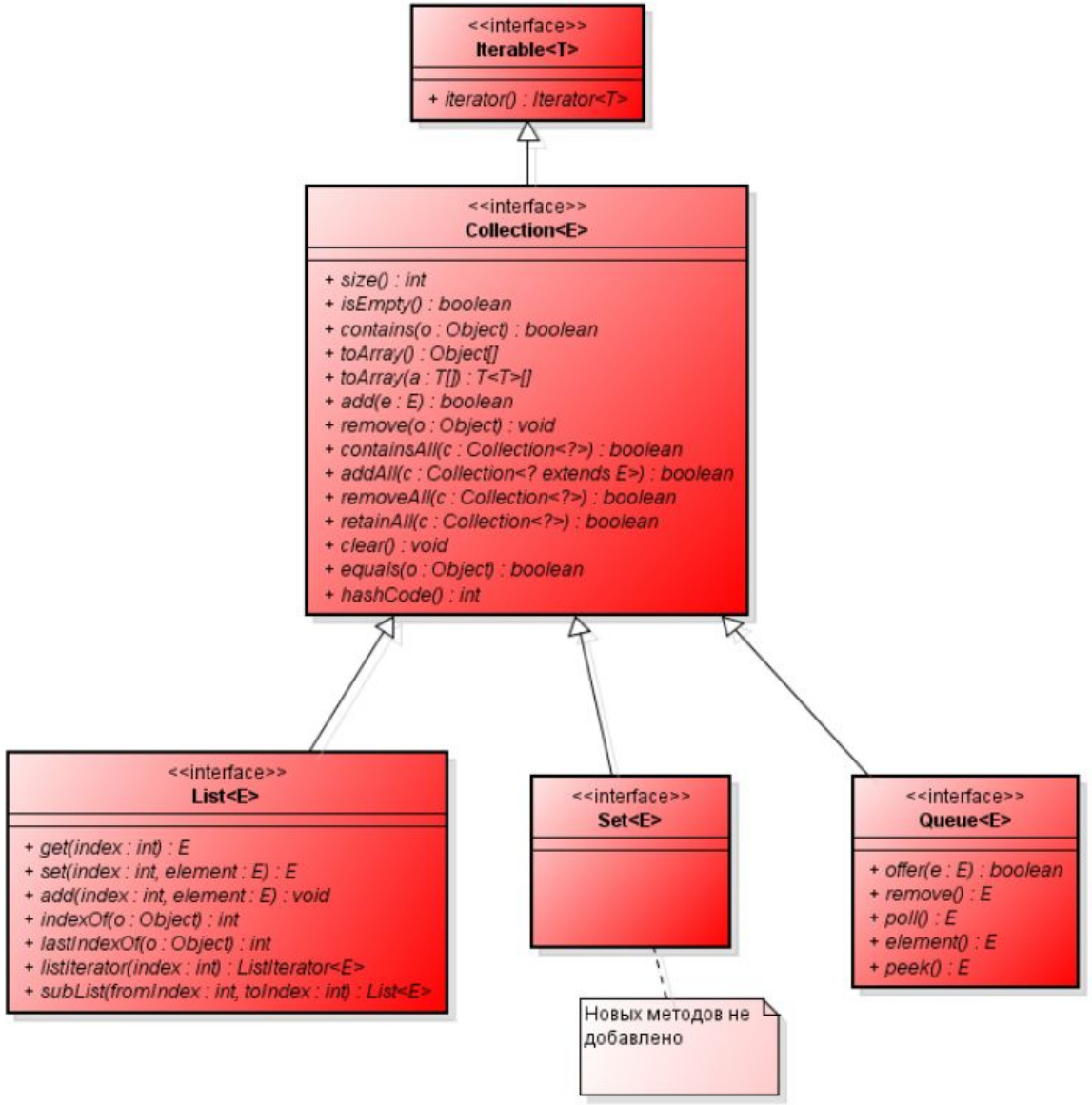
# Занятие 12
# Коллекции: Queue, Map
# Static modifier

# Структура коллекций




- **Collection<E>** - вершина иерархии остальных коллекций
- **List<E>** – специализирует коллекции для обработки списков;
- **Set<E>** – специализирует коллекции для обработки множеств, содержащих уникальные элементы.
- **Queue<E>** - коллекция, предназначенная для хранения элементов в порядке, нужном для их обработки.
- **Map<K,V>** – карта отображения вида “ключ-значение”;

# Queue

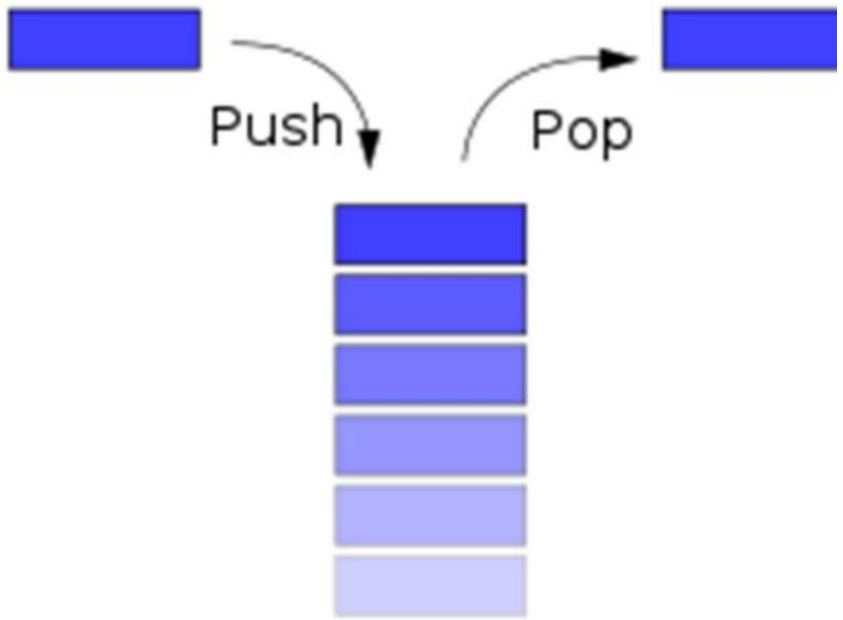
Очереди представляют структуру данных, работающую по принципу FIFO (first in - first out). То есть чем раньше был добавлен элемент в коллекцию, тем раньше он из нее удаляется. Это стандартная модель однонаправленной очереди. По сути это аналог любой очереди в реальном мире. Первым стал в очереди, первым что-то купил и ушел.

# Deque

Однако бывают и двунаправленные очереди - то есть такие, в которых мы можем добавить элемент не только в начала, но и конец. И соответственно удалить элемент не только из конца но и из начала. Интерфейс Deque расширяет вышеописанный интерфейс Queue и определяет поведение двунаправленной очереди.

# Deque

Deque может действовать как stack (стек), поскольку он предоставляет методы для работы в рамках механизма LIFO (Last In First Out) (последний добавленный элемент будет извлечен первым).



# ArrayDeque

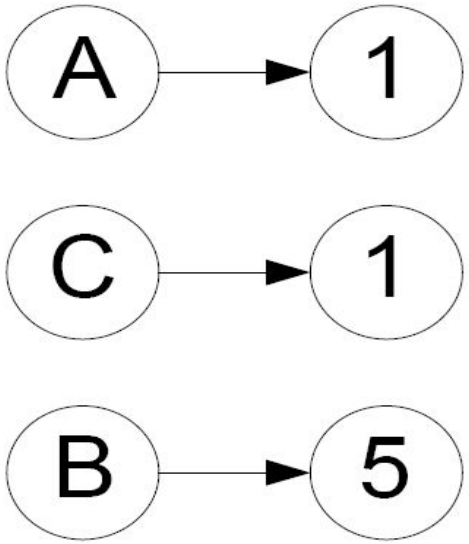
ArrayDeque - это класс в Java, который реализует Deque интерфейс. Это специальный класс, который реализует двустороннюю структуру данных очереди, где он может вставлять и удалять элементы с обоих концов.

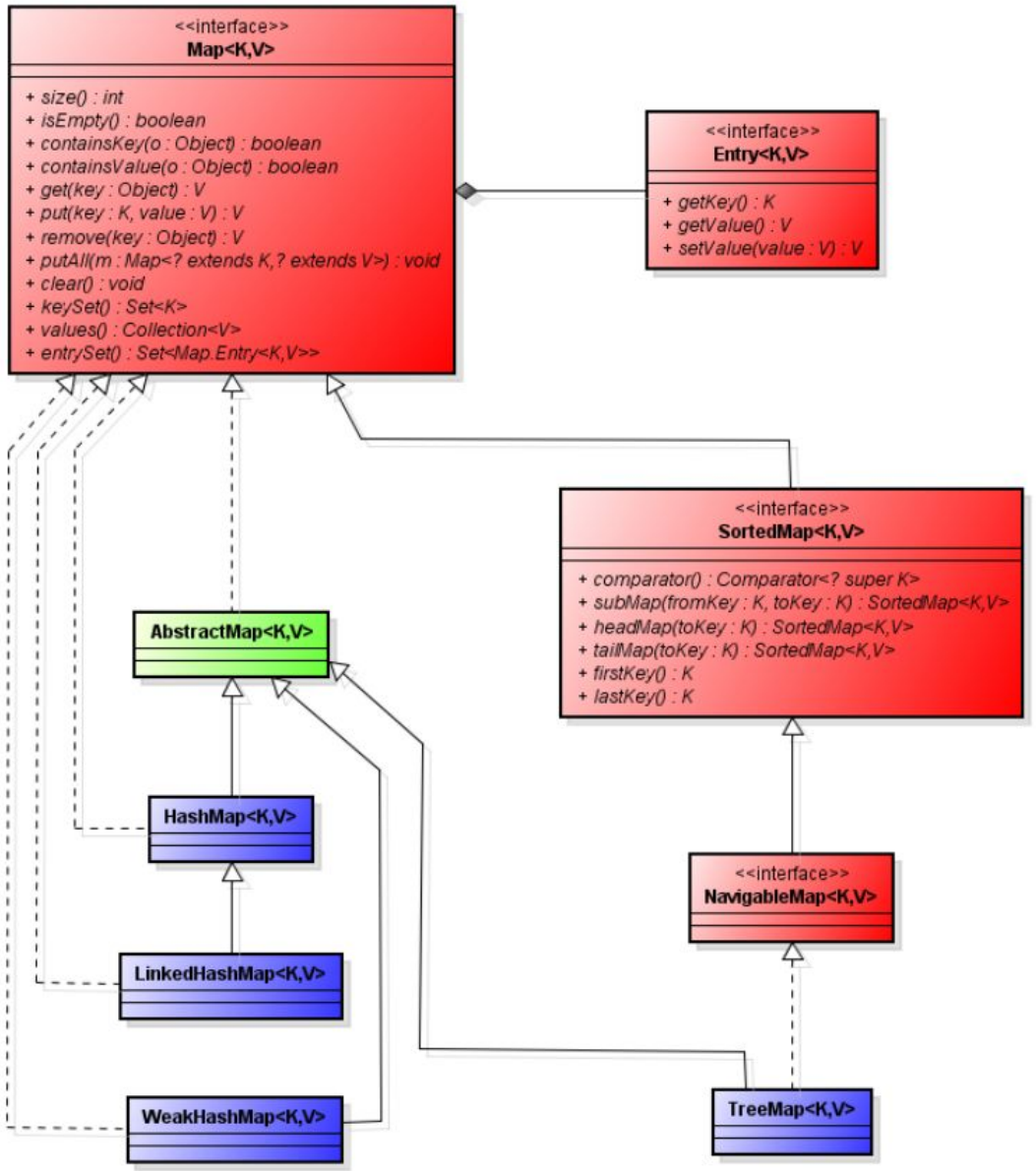
Мы можем вставлять элементы в ArrayDeque в Java, используя методы add() или offer(). Для вставки коллекции элементов мы можем использовать метод addAll(). Чтобы вставить значение в начало, используйте метод addFirst(), offerFirst() или push(), тогда как для вставки значений в конце мы можем использовать метод addLast() или offerLast().

Мы можем удалять элементы из ArrayDeque с помощью различных методов. Методы remove(), removeFirst(), poll(), pollFirst() и pop() удаляют

# Map

Интерфейс **Map** предоставляет базовые методы для работы с данными вида «ключ — значение». Map не может содержать повторяющихся ключей, каждому из которых соответствует не более одного значения.



# Методы Map

| | |
|---|---|
| void clear() | Очистка хеш-таблицы |
| boolean containsKey(Object key) | Функция проверки присутствия объекта по ключу |
| boolean containsValue(Object value) | Функция проверки присутствия объекта по значению |
| Set<Map.Entry<K,V>> entrySet() | Функция получения объекта в виде коллекции Set |
| boolean equals(Object object) | Функция сравнения с объектом object |
| Object get(Object key) | Функция получения записи по ключу |
| boolean isEmpty() | Функция проверки наличия записей |
| Set<K> keySet() | Функция получения записей в виде коллекции Set |
| void put(K key, V value) | Функция добавления записи |
| void putAll(Map<? extends K,? extends V> t) | Функция добавления записей |
| void remove(Object key) | Метод удаления объекта по ключу key |
| boolean remove(Object key, Object value) | Функция удаления записи по соответствию значений ключа и значения |
| void replace(K key, V value) | Замена значения value для записи с ключом key |
| boolean replace(K key, V oldValue, V newValue) | Замена значения oldValue на newValue для записи с ключом key |
| int size() | Функция определения количества записей в хеш-таблице |
| Collection<V> values() | Получение значений записей в виде коллекции |

# HashMap

HashMap – коллекция, хранящие значения в виде пары ключ-значение, при этом и то и то другое может принимать null-значение

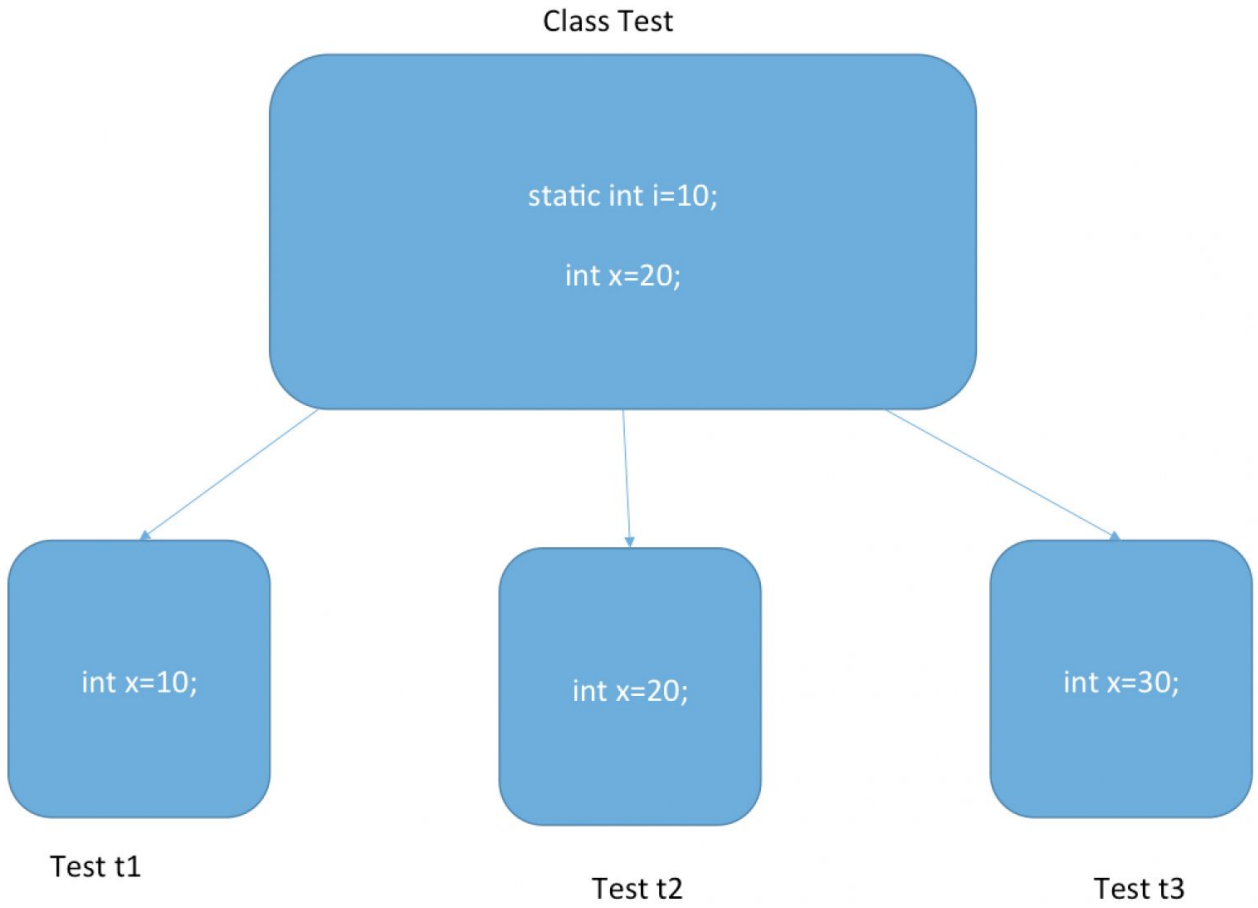
```
Map<String, String> map = new HashMap<>();
        map.put("key1", "value1");
        map.put("key2", "value2");
        map.put("key3", "value3");

        System.out.println(map);

        Map<String, String> map2 = new HashMap<>();
        map2.put("key4", "value4");
        map2.put("key5", "value5");
        map2.put("key6", "value6");

        map.putAll(map2);
        map.remove("key5");
        System.out.println("Size : " + map.size());
        System.out.println(map.containsKey("key2"));
        System.out.println(map.containsValue("value2"));

        Set<Map.Entry<String, String>> set = map.entrySet();
        for (Map.Entry<String, String> me : set) {
            System.out.print("Key : " + me.getKey() + ", Value = " + me.getValue());
        }
        map.clear();
```

# Задание:

1. Создать список номеров и стран мира, используя HashMap
2. Добавить к этому списку 3 страны
3. Удалить 2 страны
4. Вывести страну по ключу
5. Вывести список всех ключей
6. Вывести список всех стран
7. Проверить, содержится ли любые ключ/значение в коллекции

# Static

Модификатор **static** напрямую связан с классом. Если поле статично, значит оно принадлежит классу, если метод статичный — аналогично: он принадлежит классу. Исходя из этого, можно обращаться к статическому методу или полю, используя имя класса. Для работы со статическими полями используются статические методы. Статический метод не привязан к объекту, соответственно не может содержать указатель **this**.



```java
public class Program{

    public static void main(String[] args) {

        Person.displayCounter();    // Counter: 1

        Person tom = new Person();
        Person bob = new Person();

        Person.displayCounter();    // Counter: 3
    }
}
class Person{

    private int id;
    private static int counter = 1;

    Person(){
        id = counter++;
    }
    // статический метод
    public static void displayCounter(){

        System.out.printf("Counter: %d \n", counter);
    }
    public void displayId(){

        System.out.printf("Id: %d \n", id);
    }
}
```

# Задание:

Создать класс AmericanBus. Написать **статическое финальное** поле color и статический метод который выводит его цвет.

Реализовать в методе main другого класса вызов поля и метода не создавая объекта класса AmericanBus.