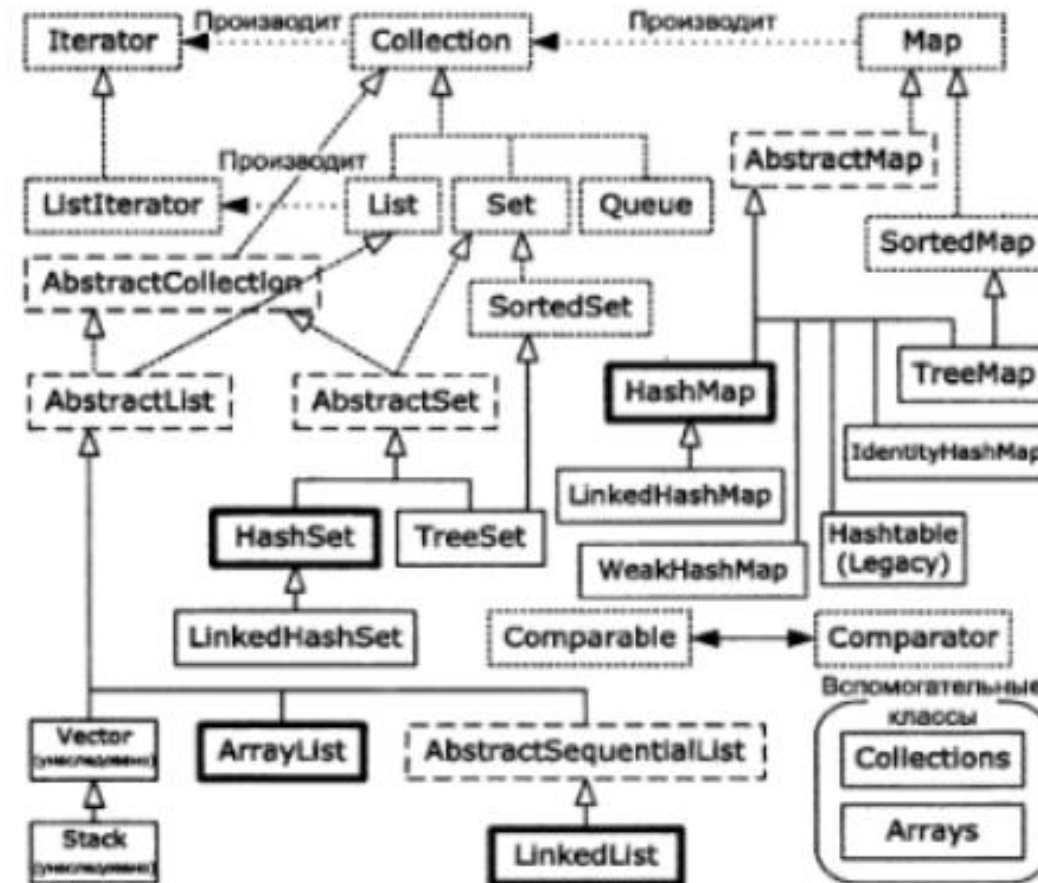


Контейнеры

- ▶ Queue - PriorityQueue - BlockingQueue
- ▶ ConcurrentMap - ConcurrentHashMap
- ▶ CopyOnWriteArrayList CopyOnWriteArraySet
- ▶ EnumSet EnumMap - в них идеальные хэш функции потому что ограниченное количество экземпляров



- ▶ Заполнение контейнеров - объекты через fill(всем местам ссылка на один объект)
- ▶ Списки передать с помощью параметра addAll
- ▶ Почти все объекты Collection имеют конструктор, получающий другой объект Collection, для нового
- ▶ Для создание функции заполнения Mapy с помощью генератора необходимо перегрузить метод создания для разных вариантов пары-ключа + новый класс с объектами пары-ключа
- ▶ Использование классов Abstract
- ▶ Каждый контейнер имеет свой класс Abstract, который представляет собой частичную реализацию контейнера - остается только реализовать необходимые методы для получения нужного контейнера. (можно реализовать контейнеры только ля чтения)
- ▶ В случае, если реализации - слишком много объектов или памяти используется Легковес. В каждом объекте Map.Entry
- ▶ Функциональность Collection(Map не производная от коллекции)

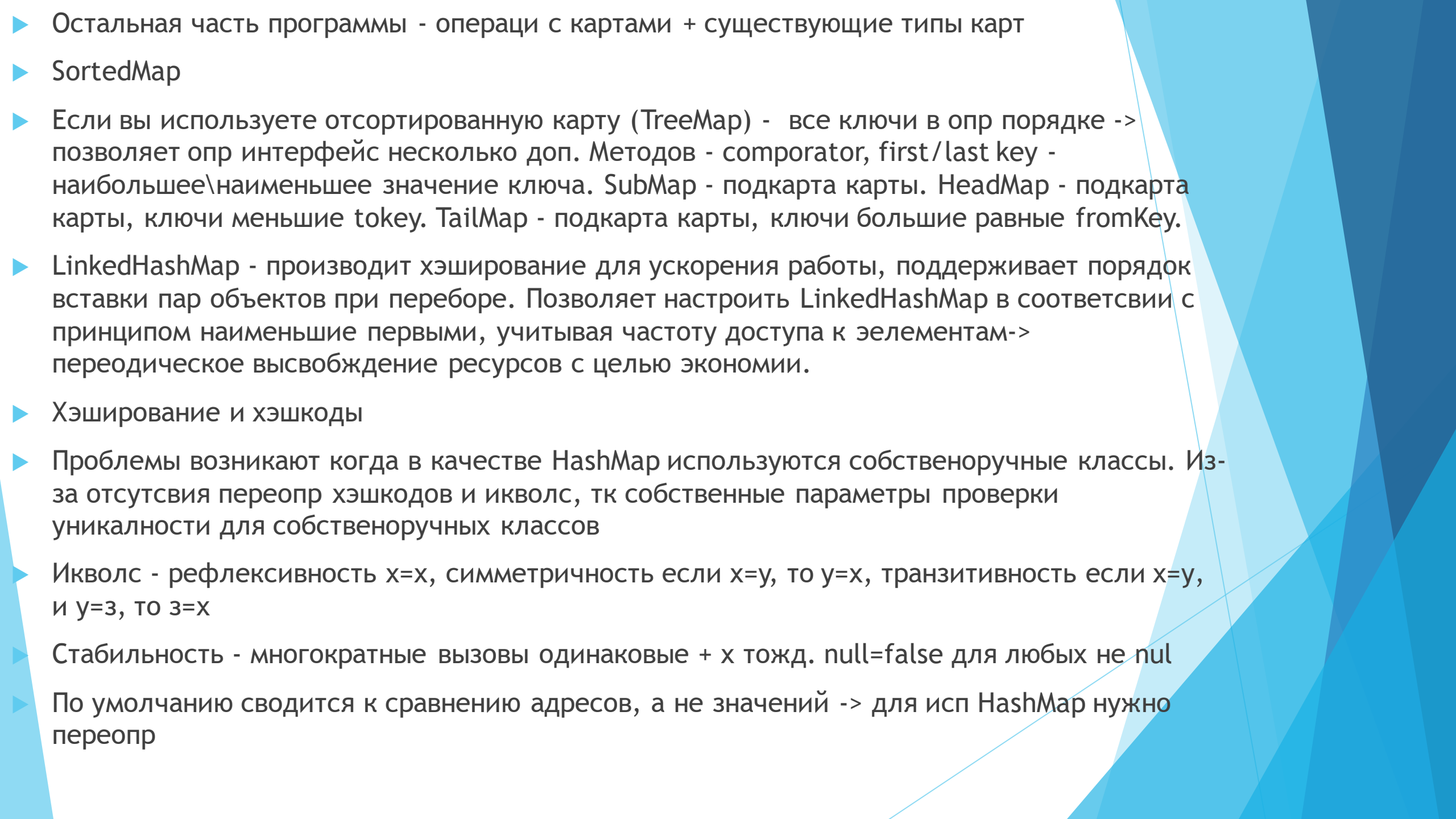
Метод	Предназначение
boolean add(T)	Проверяет, хранится ли в контейнере аргумент обобщенного типа T. Возвращает false, если аргумент не был добавлен. (Это «необязательный» метод, о нем мы узнаем чуть позже.)
boolean addAll(Collection<T> extends T>)	Добавляет все элементы, содержащиеся в аргументе. Возвращает true, если был добавлен хотя бы один элемент (необязательный метод)
void clear()	Удаляет все элементы из контейнера (необязательный метод)
boolean contains(T)	Возвращает true, если в контейнере хранится аргумент обобщенного типа T
boolean containsAll(Collection<T>)	Возвращает true, если в контейнере содержатся все элементы, имеющиеся в аргументе
boolean isEmpty()	Возвращает true, если в контейнере нет элементов

Метод	Предназначение
Iterator<T> iterator()	Возвращает объект Iterator<T>, который можно использовать для перемещения по элементам контейнера
boolean remove(Object)	Если аргумент содержится в контейнере, то один его экземпляр из него удаляется. Возвращает true, если удаление прошло успешно (необязательный метод)
boolean removeAll(Collection<T>)	Удаляет все элементы, содержащиеся в аргументе. Возвращает true, если было проведено хотя бы одно удаление (необязательный метод)
boolean retainAll(Collection<T>)	Оставляет в контейнере только те элементы, которые присутствуют в аргументе («пересечение» на языке теории множеств). Возвращает true, если произошли какие-либо изменения (необязательный метод)
int size()	Возвращает количество элементов в контейнере
Object[] toArray()	Возвращает массив, содержащий все элементы контейнера
<T> T[] toArray(T[] a)	Возвращает массив, содержащий все элементы контейнера, тип которых совпадает с типом массива-аргумента a (вместо Object)

- ▶ Необязательные операции
 - ▶ Методы, выполняющие операции добавление и удаления
 - ▶ Компилятор ограничивает вызов метода -> ничего не ломается в процессе
 - ▶ Все методы чтения - обязательны
 - ▶ Необязательные методы для предотвращения интерфейса в архитектуре. Если их не переопр - искл(не поддержив операция, но должен быть редким), нужно, если хотите наследоваться от интерфейса, но не хотите, чтобы можно было добавлять и удалять, но чтоб интерфейсов было не много
 - ▶ Происходит динамическая проверка можно или нельзя
 - ▶ Неподдерживаемые операции(когда необязательные не переопр)- используется для представления контейнеров как структуры данных
 - ▶ Немодифицируемые методы - упаковывают контейнер в объект-заместитель
 - ▶ Set и порядок хранения - Set должен определять порядок хранения. Set - идентичен Collection, не гарантирует опр порядок хранения, гарантирует, что хранимые элементы будут располагаться в опр. Порядке. HashSet - реализация Set, первоочередное значение - быстрый поиск, объекты должны опр метод hashCode. TreeSet - множество, реализованное на основе дерева. Можно извлекать упорядоченно посл. Объекты должны реализовывать интерфейс Comparable(). LinkedHashSet - скорость поиска + используя связный список -> запоминает порядок вставки
- Лучше всего HashSet + лучше всего переопр hashCode, если переопр equals().

- ▶ Без хэшкода в хэшсет -> дубликаты + ошибка времени выполнения. Трисет без компарабле - исключение
- ▶ SortedSet - хранятся в порядке сортировки, позволяет задействовать доп функц, обеспеч интерфейсом SortedSet
- ▶ Доп функции - first - наименьший элемент, last - наибольший элемент, subSet - подмножество множества, включающее элементы из аргументов (от до). HeadSet - подмножество множества, содержащее элементы меньше to Elements, tailSet - подмножество, большие\равные fromElement
- ▶ Очереди
- ▶ PriorityQueue+LinkedHashSet - порядок следования реализацией Comparable + переопределение hashCode
- ▶ Двусторонняя очередь - возможность добавление и удаления с двух сторон
- ▶ Нет явно определенного интерфейса -> LinkedList не может реализовать интерфейс, но можно создать с помощью композиции. Можно создать посредством композиции и предоставить доступ к соотв методам из LinkedList + добавить методы addFirst и addLast
- ▶ Карты - особенность карты - хранение связи ключ-значение -> обращение к значению по ключу

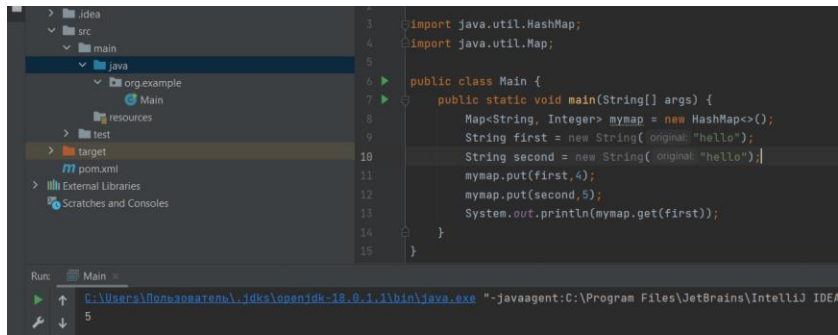
- ▶ HashMap, TreeMap, LinkedHashMap, WeakHashMap, ConcurrentHashMap, IdentityHashMap.
- ▶ Производительность карт
- ▶ Использование линейного поиска в get - плохая производительность., HashMap - увеличивает скорость поиска. Работает со спец значением - хэшкод.
- ▶ ХЭШКОД - способ преобр инф об объекте в относ-уникальное целое число, связанное с этим объектом. Метод hashCode() встроен в корневой класс Object. Карта задействует hashCode() для быстрого поиска ключа. Приводит к всплеску производительности
- ▶ HashMap - реализация карты на основе хэш-таблицы (вместо устаревшего Hashtable). Поиск и вставка пар - небольшое const время
- ▶ LinkedHashMap - как hashMap, но при переборе выдает пары в порядке добавления/согласно принципу LRU - ("наименее используемые первые")
- ▶ TreeMap - сортированные вид (красно-черное дерево). Есть метод subMap(), позволяет выделять из карты некоторую часть
- ▶ WeakHashMap - состоит из "слабых" ключей - не препятствуют освобождению объектов, на которые ссылается карта. Если ссылок на ключ нет за пределами значений карты - удаляется уборщиком мусора
- ▶ ConcurrentHashMap - потоково-безопасная версия Map, не исп синхрониз. Блокировку
- ▶ IdentityHashMap - хэш-таблица, использующая для сравн == вместо ikволс
- ▶ Самая распростран - хэширование.



- ▶ Остальная часть программы - операции с картами + существующие типы карт
- ▶ SortedMap
- ▶ Если вы используете отсортированную карту (TreeMap) - все ключи в опр порядке -> позволяет опр интерфейс несколько доп. Методов - comparator, first/last key - наибольшее\наименьшее значение ключа. SubMap - подкарта карты. HeadMap - подкарта карты, ключи меньше tokey. TailMap - подкарта карты, ключи больше равные fromKey.
- ▶ LinkedHashMap - производит хэширование для ускорения работы, поддерживает порядок вставки пар объектов при переборе. Позволяет настроить LinkedHashMap в соответствии с принципом наименьшие первыми, учитывая частоту доступа к элементам-> периодическое высвобождение ресурсов с целью экономии.
- ▶ Хэширование и хэшкоды
- ▶ Проблемы возникают когда в качестве HashMap используются собственоручные классы. Из-за отсутствия переопр хэшкодов и икволс, тк собственные параметры проверки уникальности для собственоручных классов
- ▶ Икволс - рефлексивность $x=x$, симметричность если $x=y$, то $y=x$, транзитивность если $x=y$, и $y=z$, то $z=x$
- ▶ Стабильность - многократные вызовы одинаковые + x тожд. `null=false` для любых не `null`
- ▶ По умолчанию сводится к сравнению адресов, а не значений -> для исп HashMap нужно переопр

- ▶ ХЭШКОД
- ▶ Хэширование нужно для поиска объекта по другому объекту в мапе
- ▶ Методы карты
- ▶ Put - помещает ключи, get - null если нет ключа, значение если есть, entrySet - объекты Map.Entry (хранит и получает ключи - значение)
- ▶ Хэширование ради скорости
- ▶ Медленная карта - не быстра, тк ключи в неупоряд сост, используется линейный поиск
- ▶ Нет хеширования для скорости
- ▶ ХЭШИРОВАНИЕ позволяет провести поиск оперативно. При хэшировании необходимо хранить ключи где-то - самая быстрая структура - массив -> используется для хранения информации о ключах(Но он же ограниченный???) -> один индекс массива - несколько ключей (по ключу строится объект для индексирования массива - число - хэшкод)
- ▶ Процесс поиска - вычисление хэш-кода, поиск его в массиве. Для избежание коллизий - внешнее связывание - элемент содержит не конкретное значение, а указывает на цепочку значений. Оюнаружение производится с помощью линейного алгоритма и икволс.
- ▶ Ячейки хэш-табл - узловые группы (buckets). Чтобы распр было более равномерным - кол-во узлов из простых чисел, но не очень, поэтому размер равный степени числа два.

- ▶ Переопр hashCode()
- ▶ Создание значение для индекс массива узл групп - зависит от заполнения контейнера и коэф. Загрузки таблицы. Значение, возвращаемое hashCode() будет обработано для получение индекс узла. >
- ▶ Если hashcode зависит от изменяемых данных, то изменение данных -> обновление ключа
- ▶ Для получения хэшкада не стоит брать ссылку this



The screenshot shows an IDE with a project structure on the left and Java code in the center. The project structure includes 'src', 'main', 'resources', 'test', 'target', 'pom.xml', 'External Libraries', and 'Scratches and Consoles'. The code in the center is as follows:

```
1 import java.util.HashMap;
2 import java.util.Map;
3
4 public class Main {
5     public static void main(String[] args) {
6         Map<String, Integer> mymap = new HashMap<>();
7         String first = new String( original "hello");
8         String second = new String( original "hello");
9         mymap.put(first, 4);
10        mymap.put(second, 5);
11        System.out.println(mymap.get(first));
12    }
13 }
```

The Run console at the bottom shows the command: `C:\Users\Nonna\AppData\Local\Temp\jdk-openjdk-18.0.1\bin\java.exe -javaagent:C:\Program Files\JetBrains\IntelliJ IDEA` and the output: `5`.

- ▶ Вот что будет, если сделать через нью стринг, хотя через просто стринг лучше всего, но у нью стринг одинаковые хэш коды, но можно положить с разными названиями, но все равно будет плохо

▶ Рецепт создание метода хэшкод

Тип поля	Вычисления
Boolean	<code>c = (f ? 0 : 1)</code>
byte, char, short или int	<code>c = (int)f</code>
long	<code>c = (int)(f ^ (f >>> 32))</code>
float	<code>c = Float.floatToIntBits(f);</code>
double	<code>long l = Double.doubleToLongBits(f);</code> <code>c = (int)(l ^ (l >>> 32))</code>
Object, где метод equals() вызывает equals() для каждого поля	<code>c = f.hashCode()</code>
Массив	К каждому элементу применяются описанные выше правила

- ▶ Существует только четыре разновидности контейнеров - карта (Map), список (List), множество (Set) и очередь (Queue)
- ▶ Разные типы queue - различия по тому, как они получают и передают значения
- ▶ LinkedList - двунаправленный список ArrayList - основан на массиве
- ▶ Выбор List
- ▶ ArrayList - все примерно равное
- ▶ LinkedList - быстрая вставка, больше на гет/сет
- ▶ Vector - устаревший
- ▶ Queue - быстрое удаление вставка в начало конец
- ▶ Опасности микротестов - необходимо изм только интересующие параметры
- ▶ Разные типы Setов
- ▶ TreeSet - добавление резко увеличивается
- ▶ HashSet - примерно равно, iterable резко увеличивает (не резко, чуть)
- ▶ LinkedHashSet - добавление резко увеличивается, содержит резко увеличивается
- ▶ Mapы
- ▶ TreeMap
- ▶ Положить резко увеличивается, HashMap - добавление и получить резко увеличивается, LinkedHashMap - положить резко увеличивается. Оно везде резко увеличивается, немного меняется итеративность

- ▶ Факторы, влияющие на производительность HashMap
- ▶ Емкость - количество узлов хэш-таблицы
- ▶ Начальная емкость - количество узлов при первичном создании хэш-таблицы, размер - количество заполненных узлов, находящихся в таблице, коэф загр - $\frac{\text{размер}}{\text{емкость}}$
- ▶ Хэширование происходит при $k \text{ загр} = 0.75$
- ▶ Получение неизменяемых коллекций и карт
- ▶ В необязательных функциях можно пробросить ошибку при насл интерфейса
- ▶ Срочный отказ - Механизм, не позволяющий изменить содержимое нескольких контейнеров сразу (например добавление элементов)
- ▶ Удержание ссылок - для уборки мусора
- ▶ Объект достижим, если в вашей программе его можно обнаружить -> есть ссылка на объект
- ▶ WeakHashMap - для хранения слабых ссылок.