

Коллекции

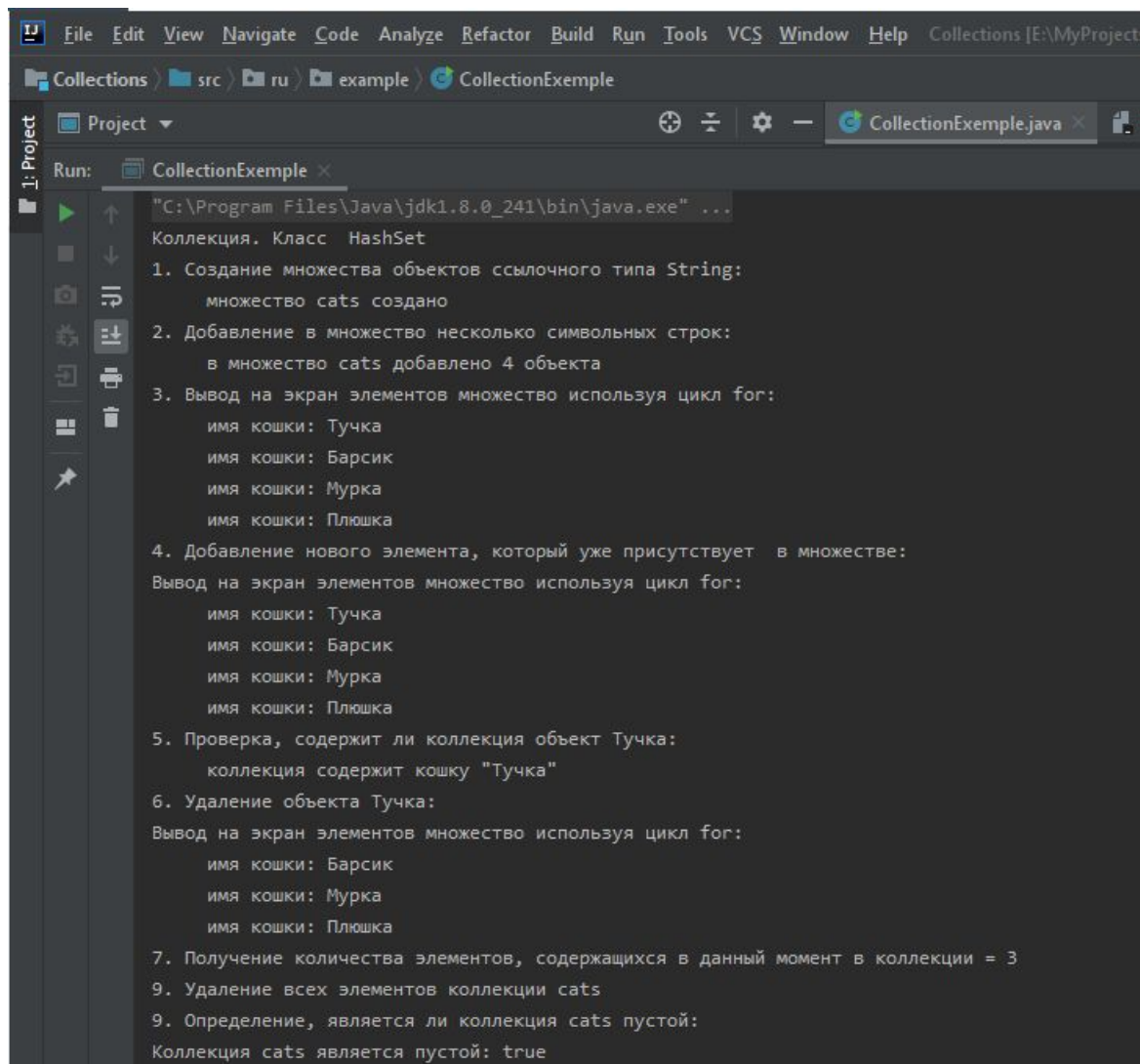
Интерфейсы Collection, Set, Map. Классы HashSet, HashMap

Файзрахманова Карина Эмильевна,
доцент кафедры АСУ, 6-315
malikovakarina@gmail.com

Задание на практическую работу 2

1. Изучить теоретический материал (слайды 5-17). Внимательно ознакомиться с ожидаемыми результатами выполнения практической работы (слайды 3-4).
2. Выполнить исследование предметной области, которая была выбрана на практической работе 1, на предмет создания коллекций HashSet и HashMap, а также формирования их структурных элементов. Например, для первого варианта задания можно создать коллекции списков учебных групп. Для второго варианта – списков сотрудников и т.п.
3. Коллекция HashSet. Создать коллекцию (согласно своему варианту) – множество объектов ссылочного типа, используя конструктор класса HashSet. Добавить в множество минимум пять объектов, которые являются совместимыми с типом данных коллекции. Вывести на экран элементы множества используя цикл for. Добавить новый элемент, который уже присутствует в множестве. Определить, содержит ли коллекция определенный объект. Удалить из коллекции любой объект. Получить количество элементов, содержащихся в коллекции на данный момент. Удалить все элементы множества. Проверить, является ли коллекция HashSet пустой (слайд-помощи – 18, слайды-подсказки – 19-20).
4. Коллекция HashMap. Создать коллекцию (согласно своему варианту) – отображение каждого элемента из одного множества объектов (ключей) на другое (значений). Добавить в коллекцию минимум пять пар «ключ-значение». Несколько ключей должны иметь одно и то же значение. Выполнить прямой перебор коллекции используя цикл for. Добавить новый элемент с уже имеющимся в отображении ключом. Вынести список всех ключей из HashMap в отдельную коллекцию. Вынести список всех значений из HashMap в коллекцию HashSet и получить количество уникальных значений. Определить, содержит ли коллекция HashMap определенный ключ. Определить, содержит ли коллекция HashMap определенное значение. Получить количество элементов, содержащихся в данный момент в коллекции HashMap. Удалить из коллекции выбранный объект по ключу и по значению. (слайд-помощи – 21, слайды-подсказки – 22-23).
5. Результат выполнения практической работы загрузить в СДО УГАТУ следующим образом. Сформировать архив, содержащий файлы программного кода, а также файл формата DOC, который содержит экранные формы результата выполнения.

Результат выполнения практической работы (задание 3)



```
"C:\Program Files\Java\jdk1.8.0_241\bin\java.exe" ...
Коллекция. Класс HashSet
1. Создание множества объектов ссылочного типа String:
   множество cats создано
2. Добавление в множество несколько символьных строк:
   в множество cats добавлено 4 объекта
3. Вывод на экран элементов множество используя цикл for:
   имя кошки: Тучка
   имя кошки: Барсик
   имя кошки: Мурка
   имя кошки: Плюшка
4. Добавление нового элемента, который уже присутствует в множестве:
   Вывод на экран элементов множество используя цикл for:
   имя кошки: Тучка
   имя кошки: Барсик
   имя кошки: Мурка
   имя кошки: Плюшка
5. Проверка, содержит ли коллекция объект Тучка:
   коллекция содержит кошку "Тучка"
6. Удаление объекта Тучка:
   Вывод на экран элементов множество используя цикл for:
   имя кошки: Барсик
   имя кошки: Мурка
   имя кошки: Плюшка
7. Получение количества элементов, содержащихся в данный момент в коллекции = 3
9. Удаление всех элементов коллекции cats
9. Определение, является ли коллекция cats пустой:
   Коллекция cats является пустой: true
```

Результат выполнения практической работы (задание 4)

```
Run: CollectionExample x
Коллекция. Класс HashMap
1. Создание множества объектов ссылочного типа String:
2. Добавление в HashMap несколько символьных строк:
   в HashMap cats добавлено 4 объекта
3. Вывод на экран элементов множество используя цикл for:
key: 4b20 value: Вонючка
key: 1m12 value: Мурка
key: 3b16 value: Барсик
key: 2t15 value: Тучка
4. Добавление элемента с ключом 3b16, который уже присутствует в HashMap:
key: 4b20 value: Вонючка
key: 1m12 value: Мурка
key: 3b16 value: Мурка
key: 2t15 value: Тучка
5. Вынесение списка всех ключей из HashMap в отдельную коллекцию:
key = 4b20
key = 1m12
key = 3b16
key = 2t15
6. Вынесение списка всех значений из HashMap в коллекцию HashSet, а также получение количества уникальных значений: 3
value = Тучка
value = Мурка
value = Вонючка
7. Проверка, содержит ли коллекция определенный ключ 3b16:
Содержит ключ 3b16
8. Проверка, содержит ли коллекция определенное значение:
Содержит значение Мурка
9. Получение количества элементов, содержащихся в данный момент в коллекции: 4
10. Удаление из коллекции выбранного объекта по ключу 3b16:
key: 4b20 value: Вонючка
key: 1m12 value: Мурка
key: 2t15 value: Тучка
11. Удаление из коллекции выбранного объекта по ключу 2t1 и значению Тучка:
key: 4b20 value: Вонючка
key: 1m12 value: Мурка
```

Хеш-таблицы (hash table)

Хеш-таблица (hash table) – это структура данных, обеспечивающая быструю вставку, поиск и удаление объектов. Хеш-таблицы реализуются на основе массивов. Элементы таблицы хранятся в виде пар ключ-значение. Для каждого объекта выполняется преобразование в уникальное число (адрес ячейки) массива. Генерация уникального числа получается в результате хеширования на основе его ключа. Ячейки массива называются корзинами (buckets), слотами (slots).

Хеширование (hash) – процесс преобразования объекта в целочисленную форму (хеш-код), которое выполняется с помощью метода хеш-функции.

Хеш-функция (hash function) – функция, осуществляющая преобразование входных данных произвольной длины в (выходную) битовую строку установленной длины, выполняемое определенным алгоритмом. Основной целью хеш-функции является преобразование диапазона ключей в диапазон индексов, обеспечивающее равномерное распределение ключей по индексам хеш-таблицы. Требования к хеш-функциям. 1. Быстрое вычисление хэш-кода по значению ключа – сложность вычисления хэш-кода не должна зависеть от n – количества элементов в хеш-таблице. 2. Детерминированность – для заданного значения ключа хеш-функция всегда должна возвращать одно и то же значение. 3. Равномерность – хеш-функция должна равномерно заполнять массив. Класс Object имеет метод hashCode(), который используется для эффективного размещения объектов, добавляемых в коллекцию. В классах объектов данный метод должен быть переопределен (override) для эффективной работы.

Хеш-код (hash code) – результат сужающего преобразования (хеш-функции) множества значений объекта до некоторого подмножества целых чисел. Каждый Java-объект имеет свой хеш-код.

Коэффициент заполнения хеш-таблицы (load factor, fill factor) – отношение числа хранимых объектов хеш-таблицы к размеру массива (среднее число элементов на одну ячейку). От этого коэффициента зависит среднее время выполнения операций добавления, поиска и удаления объектов. Информация о диапазоне значений ключей обычно известна. На основе этого выбираются размер хеш-таблицы и хеш-функция.

Длина пробирования – количество шагов, необходимых для обнаружения элемента.

Хеш-таблицы . Методы hashCode() и equals()

Вычисление хэш-кода объекта и точное сравнение объектов – способы, которые используются для сравнения объектов. Базовый класс Object реализует методы – hashCode(), equals(), getClass(), toString(), notify(), notifyAll(), finalize(), clone().

public native int hashCode() – возвращает хэш-код объекта, который передается в качестве параметра. Допускается определять собственную реализацию метода hashCode(). Для эффективной работы с коллекцией, хэш-код не должен повторяться для неэквивалентных объектов.

```
System.out.println("Мурка".hashCode()); //1005126255
```

public boolean equals(Object obj) – используется для проверки объектов на равенство. Метод equals() должен обеспечивать: симметричность, рефлексивность и транзитивность.

```
System.out.println("Мурка".equals("Тишка")); //false
```

Эквивалентность и хэш-код взаимосвязаны между собой – хэш-код вычисляется на основании содержимого объекта: – если у двух объектов разные хэш-коды, то содержимое объектов гарантированно должно быть различно; – если у двух объектов одинаковые хэш-коды, то объекты не обязательно эквивалентны или равны (коллизия); – для одного объекта, хэш-код всегда будет одинаковым; – если объекты эквиваленты, то их хэш-коды одинаковые.

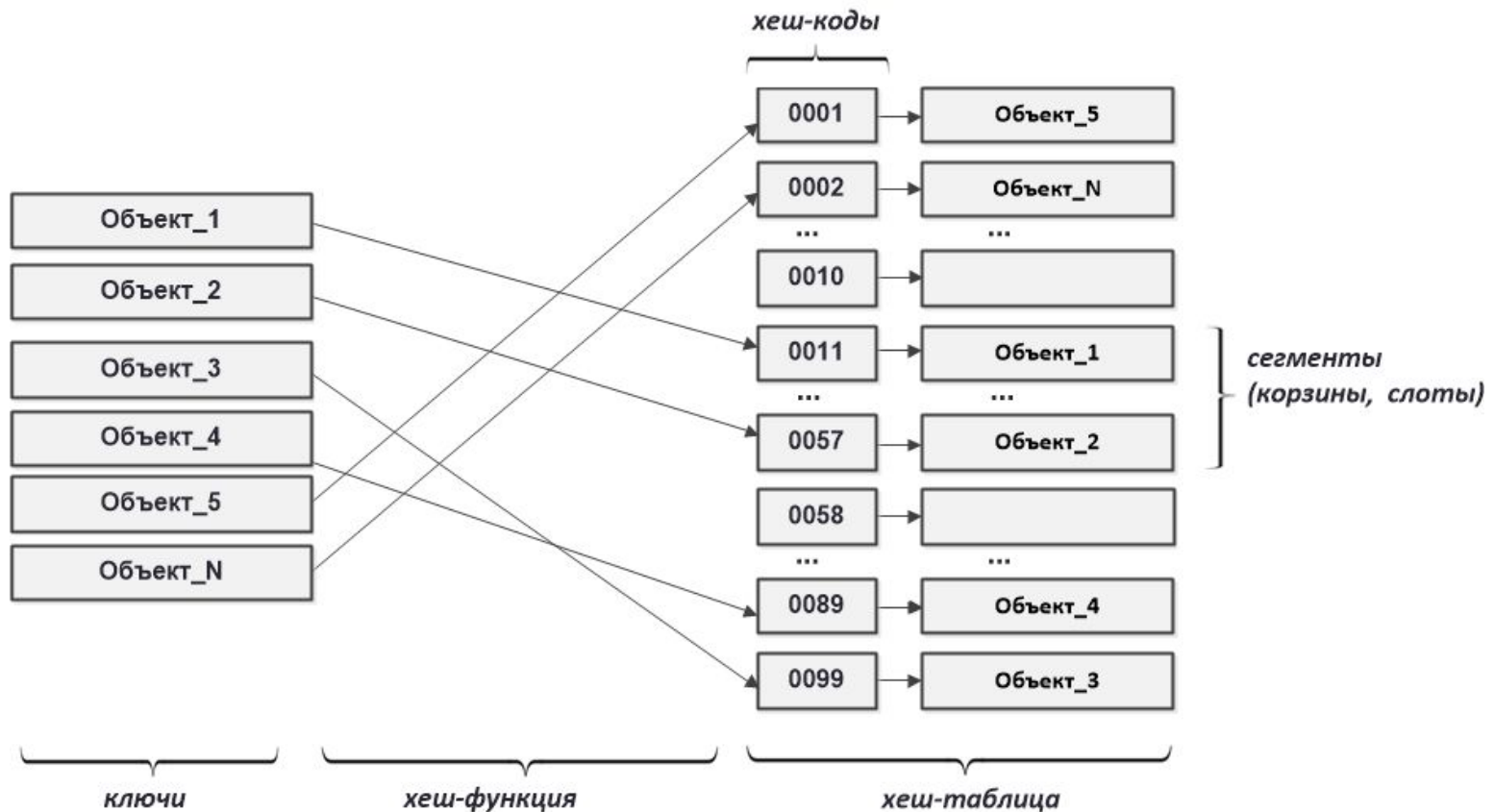
Например, есть некоторое множество объектов `Animals = {cat, lion, dog}`. Если `hashCode(cat) != hashCode(dog)`, то содержимое рассматриваемых объектов гарантированно разное. Обратное утверждение не всегда является верным – у двух разных объектов может быть различное содержимое, но одинаковые хэш-коды. Поэтому точное сравнение производится вызовом метода `equals()`, который возвращает `true`, если объекты действительно равны.

Хеш-таблицы (hash table)

Работа с элементами хеш-таблицы выполняется по следующему алгоритму.

1. **Вызов хеш-функции.** Добавление нового или поиск существующего объекта (ключа) требует вызова хеш-функции и определения хеш-кода, под которым хранится или должен быть сохранен объект.

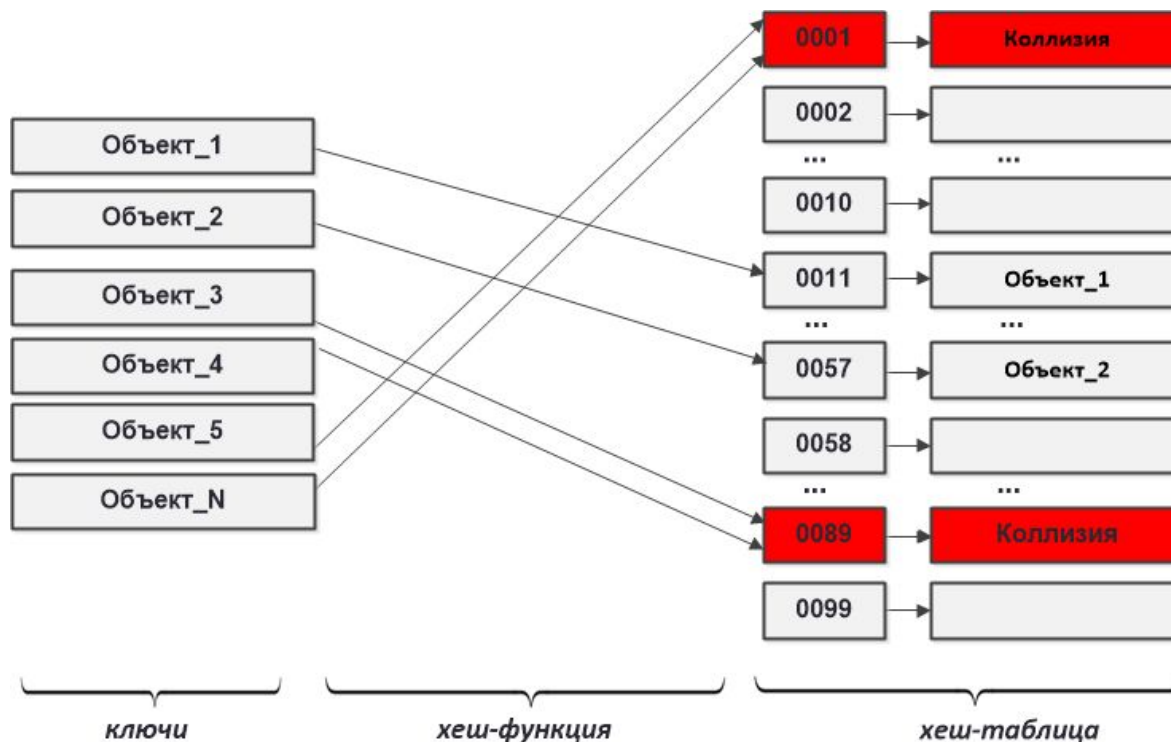
2. **Обращение к элементу массива.** Определение сегмента (корзины, слота), который соответствует полученному хеш-коду.



Хеш-таблицы. Понятие коллизии

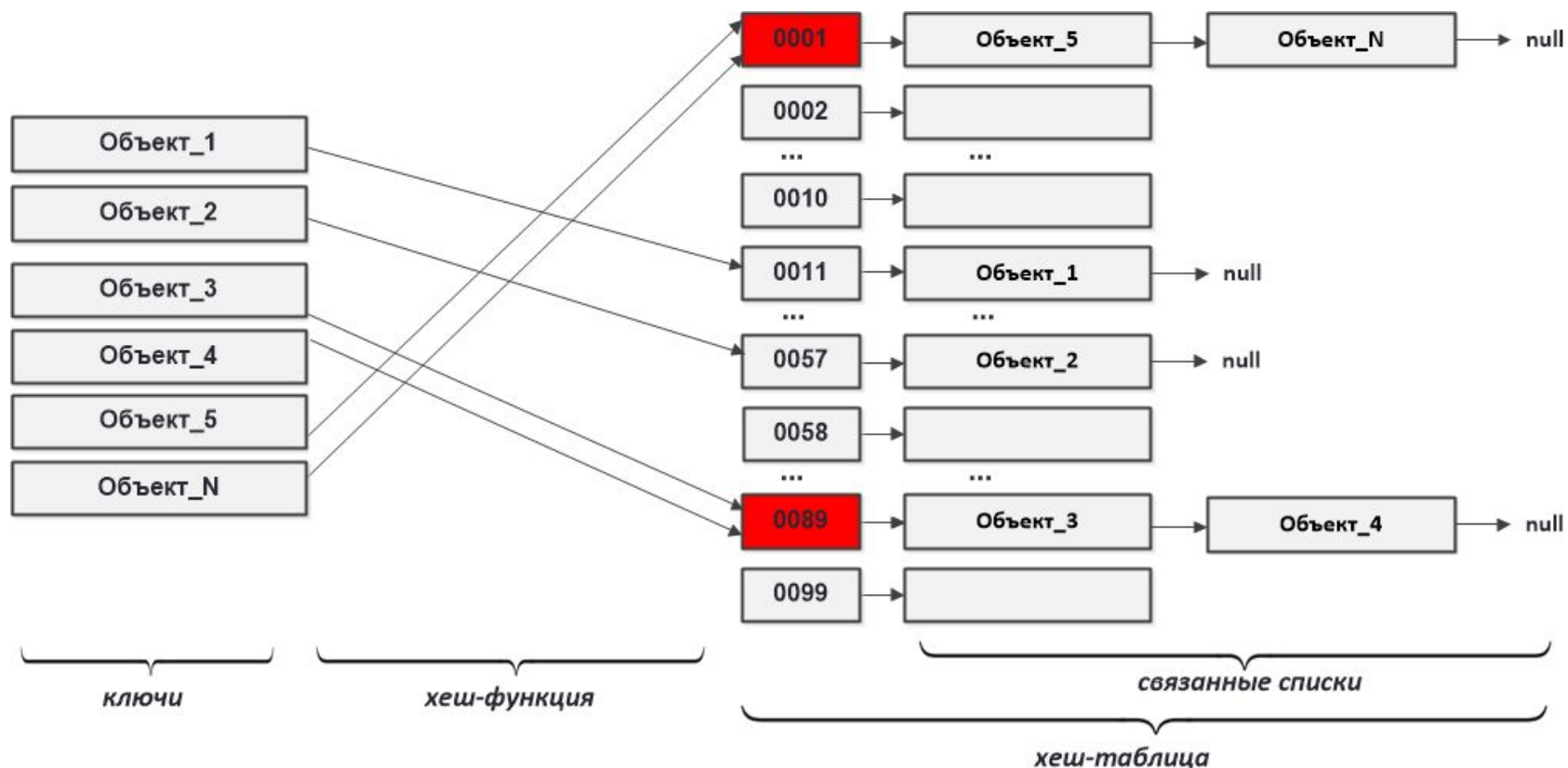
Коллизия – ситуация, при которой два разных объекта получают одинаковый хеш-код (см. рисунок). Причина коллизий возникает из особенности вычисления хеш-кода – выполняется отображение из большого пространства значений (объектов, ключей) в пространство с меньшим значением (хеш-кодов).

Существует две основных реализации хэш-таблицы: открытая адресация и метод цепочек. Например, если объект не удастся разместить в ячейке с индексом, вычисленным посредством хеш-функции, метод открытой адресации выполняет поиск другой ячейки. Другое возможное решение основано на методе цепочек – для каждого индекса в хеш-таблице вводится связанный список. Ключ элемента хешируется в индекс, а полученный элемент вставляется в связанный список по данному индексу. Другие элементы, хешируемые в тот же индекс, а затем добавляются в список.



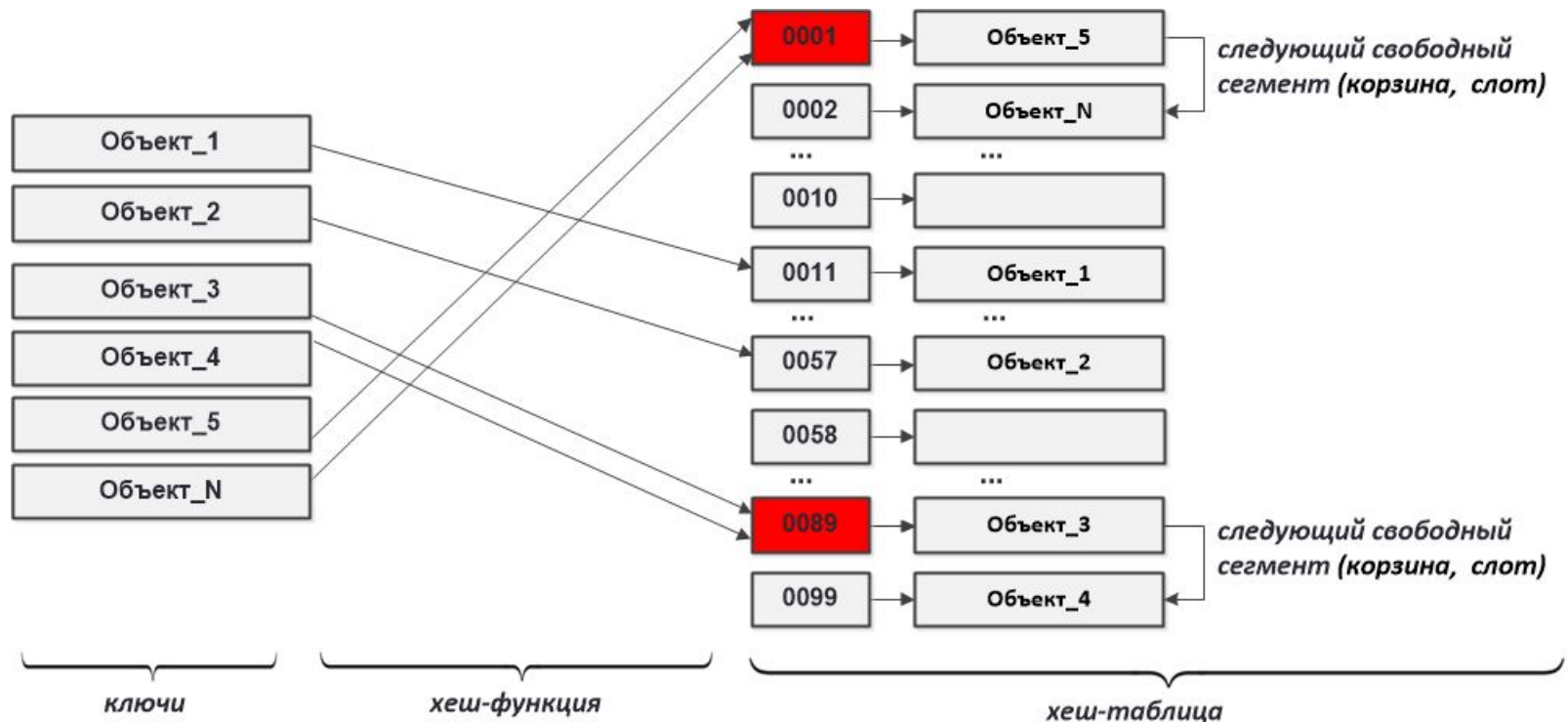
Хеш-таблицы. Разрешение коллизий. Метод цепочек

Метод цепочек (chaining) – каждой ячейке массива соответствует свой связный список. При возникновении коллизии осуществляется добавление нового элемента в данный список, т.е. элементы с одинаковым значением хеш-функции объединяются в связный список. Указатель на список хранится в соответствующей ячейке хеш-таблицы. Поиск и удаление элемента требуют просмотра всего списка. В Java для разрешения коллизий используется модифицированный метод цепочек.



Хеш-таблицы. Разрешение коллизий. Открытая адресация

Открытая адресация (open addressing) – в каждой ячейке хеш-таблицы хранится не указатель на связный список, а один элемент (ключ, значение). Если ячейка с индексом $\text{hash}(\text{key})$ занята, то осуществляется поиск следующих позиций таблицы. Основные недостатки структур с методом открытой адресации: – количество элементов в таблице не может превышать размера массива; – по мере увеличения числа элементов в таблице и повышения коэффициента заполнения (load factor) производительность структуры резко снижается, поэтому необходимо проводить повторное хеширование; – сложно организовать удаление элемента. Основное преимущество хэш-таблицы с открытой адресацией – отсутствие затрат на создание и хранение объектов списка.



Хеш-таблицы. Выводы

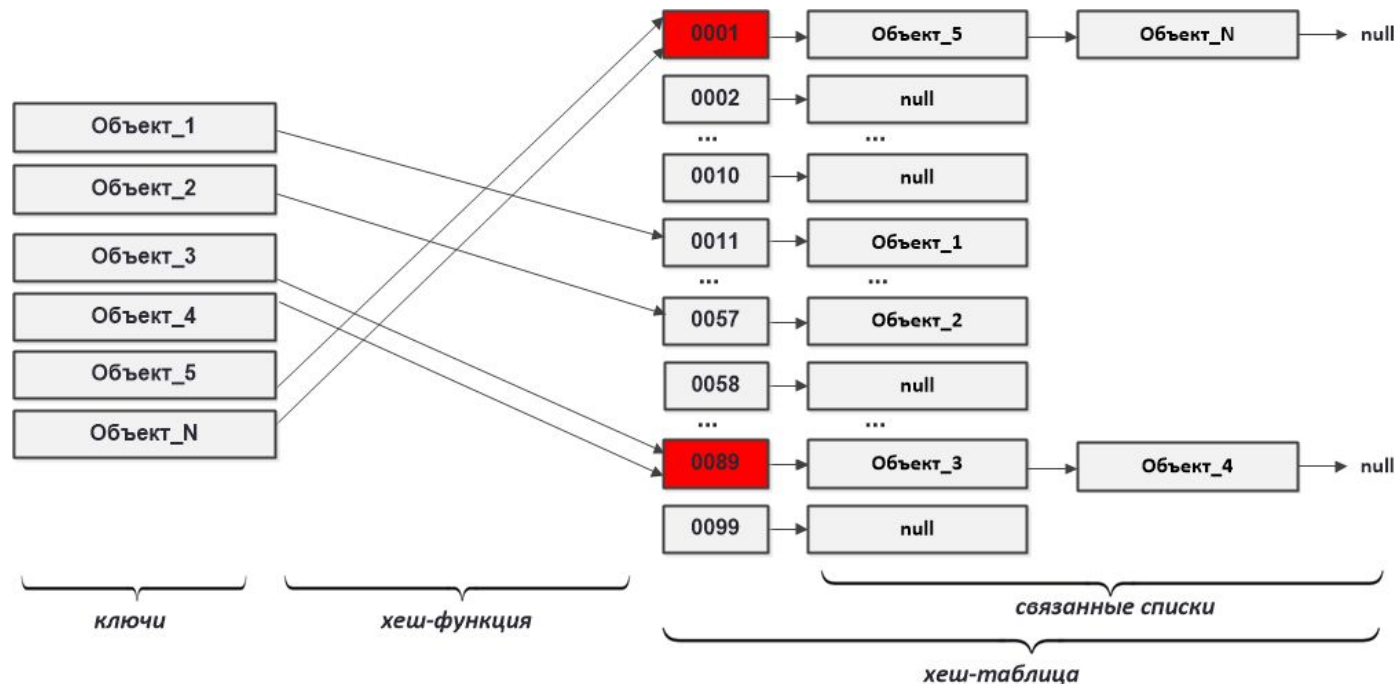
1. Хеш-таблица создается на основе массива. Главное преимущество хеш-таблиц – скорость добавления, поиска и удаления объектов.
2. Хеш-функция преобразует значение ключа в индекс массива. Основной целью хеш-функции является преобразование диапазона ключей (объектов) в диапазон индексов, обеспечивающее равномерное распределение ключей по индексам хеш-таблицы
3. Соответствие между хеш-кодом ключа (объекта) и хеш-кодом сегмента (корзины, слота) не хранится, а вычисляется. Конечно, появляется необходимость сравнивать объекты методом `equals()` но количество таких объектов ограничено.
4. Хеширование ключа в уже заполненную ячейку, корзину (buckets), слот (slots) называется коллизией.
5. Хеширование может осуществляться умножением кодов символов на разные степени констант, суммированием произведений и применением оператора вычисления остатка (%) для сокращения результата до размера хеш-таблицы.
6. Существует две основные схемы разрешения коллизий: открытая адресация и метод цепочек. При открытой адресации объекты, хешируемые в заполненную ячейку массива, размещаются в другой ячейке. В методе цепочек каждый элемент массива содержит связанный список, а все объекты, хешируемые в заданный индекс массива, вставляются в данный связанный список.
7. В Java для разрешения коллизий используется модифицированный метод цепочек.
8. Количество шагов, необходимых для обращения к элементу, называется длиной пробирирования.
9. Коэффициентом заполнения – отношение количества элементов данных в хеш-таблице к размеру массива. Диапазон значений ключей обычно превышает размер массива. Заполнение хеш-таблицы более чем наполовину (или на две трети), приведет к увеличению продолжительности пробирирования и, как следствие, к снижению быстродействия.

Интерфейсы Set, SortedSet, NavigableSet

Интерфейс Set расширяет интерфейс Collection (см. практическую работу 1, слайды 7-8) и определяет поведение коллекции – множество, не допускающее дублирования элементов. В Интерфейсе Set не определяются дополнительные методы. Для установления связей между элементами коллекции, используется хеш-код. Хеш-код зависит от содержимого объекта. Следовательно, взаимосвязь хеш-кода с объектом основывается на содержимом объекта.

Интерфейс SortedSet расширяет интерфейс Set и определяет поведение множеств, отсортированных в порядке возрастания. Помимо методов, предоставляемых интерфейсом Set, в интерфейсе SortedSet объявляются собственные методы.

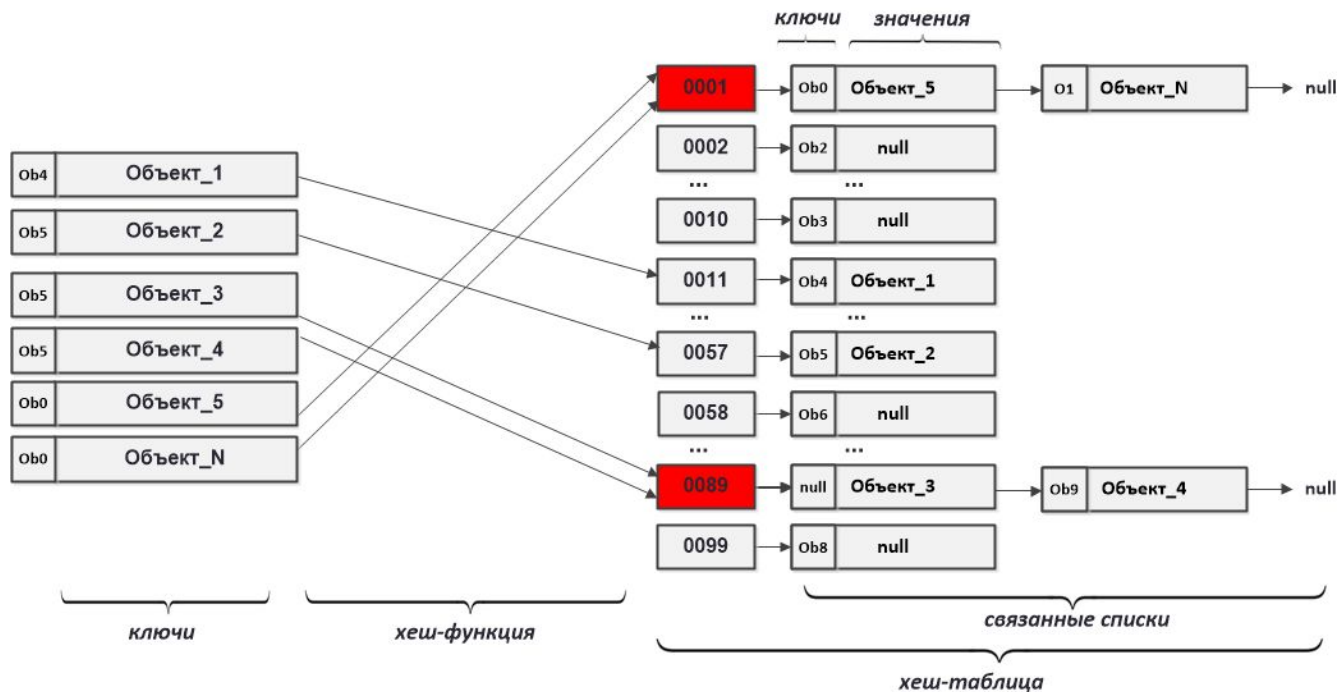
Интерфейс NavigableSet расширяет интерфейс SortedSet, добавляя методы для повышения скорости поиска упорядоченных элементов.



Интерфейс Map

Интерфейс Map – является самостоятельным интерфейсом (не расширяет интерфейс Collection, см. практическую работу 1, слайды 7-8), описывающим функциональность ассоциативных массивов. Интерфейсы Map и Collection не совместимы, т.к. созданы для различных структур данных. Map используется для отображения каждого элемента из одного множества объектов (ключей) на другое (значений): – каждому элементу из множества ключей ставится в соответствие множество значений (список); – одному элементу из множества значений может соответствовать один или более элементов из множества ключей; – параметризируется при помощи двух обобщенных типов: один для ключей, другой для значений.

Реализующие классы HashMap, LinkedHashMap, SortedMap, TreeMap хранят неупорядоченный набор объектов парами ключ-значение, в порядке возрастания ключей. Порядок следования пар ключ/значение не определен.



Основные методы интерфейса Map

Метод	Описание
void clear()	Удаляет все пары "ключ-значение" из вызывающего отображения
boolean containsKey (Object k)	Возвращает логическое значение true, если вызывающее отображение содержит указанный ключ k, иначе – false
boolean equals (Object объект)	Возвращает true, если заданный объект является отображением типа Map, содержащим одинаковые значения, иначе – false
V get (Object k)	Возвращает значение, связанное с указанным ключом k. Если же ключ не найден, то возвращается пустое значение null
boolean isEmpty()	Возвращает true, если вызывающее отображение пусто, иначе – false
Set<K> keySet()	Возвращает множество, содержащее ключи из вызывающего отображения. Следовательно, этот метод возвращает представление ключей в вызывающем отображении в виде множества
V put (K k, V v)	Вводит новое значение v в вызывающее отображение, перезаписывая любое предшествующее значение, связанное с заданным ключом k. Возвращает пустое значение null, если ключ ранее не существовал. В противном случае возвращается предыдущее значение связанное с ключом
V remove (Object k)	Удаляет запись по заданному ключу k
default V replace (K k ,V v)	Если в вызывающем отображении присутствует заданный ключ k, то значение по этому ключу заменяется новым значением v и возвращается прежнее значение, а иначе - пустое значение null
int size ()	Возвращает количество пар "ключ-значение" в вызывающем отображении
Collection<V> values()	Возвращает коллекцию, содержащую значения из вызывающего отображения. Данный метод возвращает представление значений в вызывающем отображении в виде коллекции

Классы коллекций

Класс	Описание
AbstractCollection	Реализует большую часть интерфейса Collection
AbstractList	Расширяет класс AbstractCollection и реализует интерфейс List
AbstractQueue	Расширяет класс AbstractCollection и реализует интерфейс Queue
AbstractSequentialList	Расширяет класс AbstractList для применения в коллекциях, использующих последовательности вместо случайного доступа к элементам
ArrayList	Реализует динамический массив, расширяя класс AbstractList
LinkedList	Реализует связный список, расширяя класс AbstractSequentialList
HashSet	Расширяет класс AbstractSet для применения вместе с хеш-таблицами
HashMap	Реализует интерфейс Map
ArrayDeque	Реализует динамическую двухстороннюю очередь, расширяя класс AbstractCollection и реализует интерфейс Deque
AbstractSet	Расширяет класс AbstractCollection и реализует большую часть и интерфейса Set
EnumSet	Расширяет класс AbstractSet для применения вместе элементами типа enum
LinkedHashSet	Расширяет класс HashSet, разрешая итерацию с вводом элементов в определенном порядке
PriorityQueue	Расширяет класс AbstractQueue для поддержки очередей по приоритетам
TreeSet	Реализует множество, хранимое в древовидной структуре. Расширяет класс AbstractSet

Реализация интерфейса Set. Класс HashSet

Класс HashSet – реализует интерфейс Set, необходим для создания неупорядоченного множества уникальных элементов.

1. HashSet реализован на основе хеш-таблицы:

– порядок объектов коллекции определяется по хеш-коду; – множество HashSet может содержать только одно null значение.

3. Класс HashSet реализован с использованием метода цепочек.

2. Класс HashSet реализует интерфейс Set и объявляется следующим образом:

```
class HashSet <E>
    HashSet<String> myHashSet = new HashSet<>();
```

3. В классе HashSet определены следующие конструкторы. HashSet() – создает хеш-множество по умолчанию. HashSet(Collection <? extends E > c) – создает хеш-множество, инициализируемое элементами из коллекции c. HashSet(int емкость) – создает хеш-множество, имеющее начальную емкость. Емкость – размер базового массива, используемого для хранения объектов коллекции. По мере ввода элементов в хеш-множество емкость динамически увеличивается. HashSet(int емкость, float коэффициент_заполнения) – создает хеш-множество, имеющее начальную емкость и коэффициент заполнения (емкость загрузки). Коэффициент заполнения определяет насколько заполненным должно быть хеш-множество, прежде чем будет изменен его размер. Диапазон принимаемых значений – от 0,0 до 1,0. Таким образом, когда количество объектов превысит емкость множества, умноженной на коэффициент заполнения, то множество расширяется. В конструкторах, которые не принимают коэффициент заполнения в качестве параметра, выбирается значение – 0,75.

4. **Преимущества.** Высокая скорость добавления, поиска и удаления объектов: – не требуется перебирать элементы в определенном порядке; – сравнение хэш-кода происходит многократно быстрее; – соответствие между хеш-кодом ключа и хеш-кодом сегмента (корзины, слота) не хранится, а вычисляется.

5. **Недостатки.** Отсутствие удобного способа перебора элементов в определенном порядке (например, от меньших к большему). Не все сегменты (корзины, слота) заполнены списками, на нулевые сегмента (корзины, слота) также затрачивается память.

Реализация интерфейса Map. Класс HashMap

Класс HashMap – реализует интерфейс Map, необходим для создания множества (отображения, карты) с сохранением связи между ключами и значениями в виде пар "ключ-значение".

1. **HashMap реализован на основе хеш-таблицы:** – порядок объектов коллекции определяется по хеш-коду; – ключи должны быть уникальными, а значения могут быть дублированными; – множество HashMap может содержать один null ключ и множество null значений; – по заданному ключу можно найти значение объекта; – в качестве ключей и значений могут выступать любые объекты (числа, строки или объекты других классов).

2. **Класс HashMap реализован с использованием метода цепочек.**

3. **Класс HashMap реализует интерфейс Map** и объявляется следующим образом:

```
class HashMap <E>
HashMap<String> myHashMap = new HashMap<>();
```

4. **В классе HashMap определены следующие конструкторы.** HashMap() – создает хеш-отображение по умолчанию. HashMap (Map<? extends K, ? extends V> m) – хеш-отображение иницируется элементами отображения m. HashMap (int емкость) – задается емкость хеш-отображения HashMap (int емкость, float коэффициент_заполнения) – емкость и коэффициент заполнения хеш-отображения задаются в качестве аргументов конструктора.

5. **Преимущества.** Высокая скорость добавления, поиска и удаления объектов: – не требуется перебирать элементы в определенном порядке; – сравнение хэш-кода происходит многократно быстрее; – организован доступ к объекту по ключу; – соответствие между хеш-кодом ключа и хеш-кодом сегмента (корзины, слота) не хранится, а вычисляется; – ключом Map может быть любое значение (объект, строка, символ, примитивы); – ключи в Map упорядочены.

6. **Недостатки.** Отсутствие удобного способа перебора элементов в определенном порядке (например, от меньших к большим). Не все сегменты (корзины, слота) заполнены списками, на нулевые сегмента (корзины, слота) также затрачивается память.

Применение класса HashSet

1. Создадим множество HashSet, например, cats, содержащее объекты ссылочного типа String, используя конструктор, определенный в классе HashMap: `Set<String> cats = new HashSet<>();` или `Set<String> cats = new HashSet<>(5);`.

2. Добавим в множество несколько символьных строк. Символьные строки, заключенные в кавычки, преобразуются в объекты ссылочного типа String. Добавляемые в коллекцию объекты должны быть совместимы с типом данных коллекции:

```
cats.add("Барсик");  
cats.add("Мурка");  
cats.add("Плюшка");  
cats.add("Тучка");
```

3. Выведем на экран элементы множества используя цикл for:

```
for(String cat: cats) {  
    System.out.println("\t имя кошки: " + cat);  
}
```

4. Добавим новый элемент, который уже присутствует в множестве: `cats.add("Мурка");`

5. Определим, содержит ли коллекция определенный объект: `if cats.contains("Тучка") {}`

6. Удалим из множества выбранный объект: `cats.remove("Тучка");`

7. Получим количество элементов, содержащихся в данный момент в коллекции cats: `cats.size();`

8. Удалим все элементы из множества: `cats.clear();`

9. Проверим, является ли коллекция HashSet пустой: `cats.isEmpty();`

Применение класса HashSet (начало)

```
CollectionExample.java x Collections.iml x Bred.java x HashMap.java x ListIterator.java x
1 package ru.example;
2
3 import java.util.*;
4
5 public class CollectionExample {
6
7     public static void main(String[] args) {
8
9         System.out.println("Коллекция. Класс HashSet");
10        System.out.println("1. Создание множества объектов ссылочного типа String:");
11        //Set<String> cats = new HashSet<>();
12        Set<String> cats = new HashSet<>(initialCapacity: 5);
13        System.out.println("\t множество cats создано");
14
15        System.out.println("2. Добавление в множество несколько символьных строк:");
16        cats.add("Барсик");
17        cats.add("Мурка");
18        cats.add("Плюшка");
19        cats.add("Тучка");
20        System.out.println("\t в множество cats добавлено 4 объекта");
21
22        System.out.println("3. Вывод на экран элементов отображения используя цикл for:");
23        for(String cat: cats) {
24            System.out.println("\t имя кошки: " + cat);
25        }
26
27        System.out.println("4. Добавление нового элемента, который уже присутствует в множестве:");
28        cats.add("Мурка");
29        System.out.println("Вывод на экран элементов множество используя цикл for:");
30        for(String cat: cats) {
31            System.out.println("\t имя кошки: " + cat);
32        }
33    }
34 }
```

Применение класса HashSet (окончание)

```
33
34 System.out.println("5. Проверка, содержит ли коллекция объект Тучка:");
35 if (cats.contains("Тучка")) {
36     System.out.println("\t коллекция содержит кошку \"Тучка\"");
37 }
38 System.out.println("6. Удаление объекта Тучка:");
39 cats.remove( o: "Тучка");
40 System.out.println("Вывод на экран элементов множество используя цикл for:");
41 for(String cat: cats) {
42     System.out.println("\t имя кошки: " + cat);
43 }
44
45 System.out.println("7. Получение количества элементов, содержащихся в данный момент в коллекции = " + cats.size());
46
47 System.out.println("9. Удаление всех элементов коллекции cats");
48 cats.clear();
49
50 System.out.println("9. Определение, является ли коллекция cats пустой:");
51 System.out.println("Коллекция cats является пустой: " + cats.isEmpty());
52
```

Применение класса HashMap

1. Создадим отображение HashMap, например, catMap, содержащее объекты ссылочного типа String, используя конструктор, определенный в классе HashMap. Укажем тип для ключей (String), а также тип для значений (String): `Map<String, String> catMap = new HashMap<>()` или `Map<String, String> catMap = new HashMap<>(5)`.

2. Добавим в множество несколько пар «ключ-значение». Несколько ключей должны иметь одно и тоже значение. Чтобы избежать дублирования в качестве ключа зададим регистрационный номер cats, а в качестве значения – имя:

```
catMap.put("1m12", "Мурка");
catMap.put("2t15", "Тучка");
catMap.put("3b16", "Барсик");
catMap.put("4b20", "Барсик");
```

3. Выведем на экран элементы множества используя цикл for:

```
for (Map.Entry<String, String> cat : catMap.entrySet()){
    System.out.println("key: " + cat.getKey() + " value: " + cat.getValue());
}
```

4. Добавим новый элемент с уже имеющимся в отображении ключом: `catMap.put("3", "Мурка");`

5. Вынесем список всех ключей из HashMap в отдельную коллекцию:

```
List<String> keys = new LinkedList<>(catMap.keySet());
```

6. Вынесем список всех значений из HashMap в коллекцию HashSet и получить количество уникальных значений:

```
Set<String> values = new HashSet<>(catMap.values());
System.out.println(values.size());
```

7. Определим, содержит ли коллекция определенный ключ: `if (catMap.containsKey("3")){}`

8. Определим, содержит ли коллекция определенное значение: `if (catMap.containsValue("Мурка")){}`

9. Получим количество элементов, содержащихся в данный момент в коллекции: `catMap.size();`

10. Удалим из коллекции выбранный объект по ключу и значению:

```
catMap.remove("3b16");
catMap.remove("2", "Тучка");
```

Применение класса HashMap (начало)

```
53
54 System.out.println("Коллекция. Класс HashMap");
55 System.out.println("1. Создание множества объектов ссылочного типа String:");
56 Map<String, String> catMap = new HashMap<>();
57
58 System.out.println("2. Добавление в HashMap несколько символьных строк:");
59 catMap.put("1m12", "Мурка");
60 catMap.put("2t15", "Тучка");
61 catMap.put("3b16", "Барсик");
62 catMap.put("4b20", "Вонючка");
63 System.out.println("\t в HashMap cats добавлено 4 объекта");
64
65 System.out.println("3. Вывод на экран элементов множество используя цикл for:");
66 for (Map.Entry<String, String> cat : catMap.entrySet()) {
67     System.out.println("key: " + cat.getKey() + " value: " + cat.getValue());
68 }
69
70 System.out.println("4. Добавление элемента с ключом 3b16, который уже присутствует в HashMap:");
71 catMap.put("3b16", "Мурка");
72 for (Map.Entry<String, String> cat : catMap.entrySet()) {
73     System.out.println("key: " + cat.getKey() + " value: " + cat.getValue());
74 }
75
76 System.out.println("5. Вынесение списка всех ключей из HashMap в отдельную коллекцию:");
77 List<String> keys = new LinkedList<>(catMap.keySet());
78 for (String key : keys) {
79     System.out.println("key = " + key);
80 }
81
```

Применение класса HashMap (окончание)

```
81
82 System.out.print("6. Вынесение списка всех значений из HashMap в коллекцию HashSet, а также получение количества уникальных значений: ");
83 Set<String> values = new HashSet<>(catMap.values());
84 System.out.println(values.size());
85 for(String value : values) {
86     System.out.println("value = " + value);
87 }
88
89 System.out.println("7. Проверка, содержит ли коллекция определеннй ключ 3b16:");
90 if (catMap.containsKey("3b16")) {
91     System.out.println("Содержит ключ 3b16");
92 }
93
94 System.out.println("8. Проверка, содержит ли коллекция определенное значение:");
95 if (catMap.containsValue("Мурка")) {
96     System.out.println("Содержит значение Мурка");
97 }
98
99 System.out.println("9. Получение количества элементов, содержащихся в данный момент в коллекции: " + catMap.size());
100
101 System.out.println("10. Удаление из коллекции выбранного объекта по ключу 3b16:");
102 catMap.remove( key: "3b16");
103 for (Map.Entry<String, String> cat : catMap.entrySet()) {
104     System.out.println("key: " + cat.getKey() + " value: " + cat.getValue());
105 }
106 System.out.println("11. Удаление из коллекции выбранного объекта по ключу 2t1 и значению Тучка:");
107 catMap.remove("2t15", "Тучка");
108 for (Map.Entry<String, String> cat : catMap.entrySet()) {
109     System.out.println("key: " + cat.getKey() + " value: " + cat.getValue());
110 }
111
```