

```
1 import java.util.Objects;
2
3 public class Book{
4     private String title;
5     private int year;
6     private int pages;
7
8     public Book(String title, int year, int pages){
9         this.title = title;
10        this.year = year;
11        this.pages = pages;
12
13    }
14    public int getPages() {
15
16        return pages;
17    }
18    public int getYear() {
19        return year;
20    }
21    public String getTitle() {
22        return title + "(" + year + ")";
23    }
24    public String toString(){
25        return "Название: " + title + ", Год выпуска
: " + year + ", Страницы: " + pages;
26    }
27
28    @Override
29    public boolean equals(Object o){
30        if(this == o) return true;
31        if(o == null || getClass() != o.getClass())
return false;
32        Book book = (Book) o;
33        return year == book.year && pages == book.
pages && title.equals(book.title);
34    }
35
36    @Override
37    public int hashCode(){
38        return Objects.hash(title, year, pages);
```

```
39     }  
40  
41 }  
42
```

```
1 import java.util.ArrayList;
2 import java.util.Comparator;
3 import java.util.stream.Stream;
4
5 public class main {
6     public static void main(String[] args) {
7         // Создаем переменную студента
8         ArrayList<Student> students = new ArrayList
9         <>();
10
11
12         Student student1 = new Student("MalcolmX");
13         //Создаем книгу
14         Book book1 = new Book("Harry Potter and the
15 Philisipher's ston", 1997, 150);
16         Book book2 = new Book("Harry Potter and the
17 Chambers of Secrets", 1998, 200);
18         Book book3 = new Book("Harry Potter and the
19 Prisoner of Azkaban", 1999, 215);
20         Book book4 = new Book("Harry Potter and the
21 Goblet of Fire", 2000, 300);
22         Book book5 = new Book("Harry Potter and the
23 Orden of the Phoenix", 2003, 600);
24
25         Student student2 = new Student("John");
26         Book book6 = new Book("Harry Potter and the
27 Half-Blood Prince", 2005, 500);
28         Book book7 = new Book("Harry Potter and the
29 Deathly Hallows", 2007, 550);
30         Book book8 = new Book("A Daughter of the
31 Snows", 1902, 250);
32         Book book9 = new Book("The Kempton-Wace
33 Letters", 1903, 400);
34         Book book10 = new Book("Martin Eden", 1909,
35 600);
36         Book book11 = new Book("War and Peace", 2007
37 , 550);
38
39         students.add(student1);
```

```

30         students.add(student2);
31
32         student1.addBook(book1);
33         student1.addBook(book2);
34         student1.addBook(book3);
35         student1.addBook(book4);
36         student1.addBook(book5);
37         student2.addBook(book6);
38         student2.addBook(book7);
39         student2.addBook(book8);
40         student2.addBook(book9);
41         student2.addBook(book10);
42         student2.addBook(book11);
43
44         students.stream()
45             // Вывести каждого студента
46             .peek(System.out::println)
47             // Получить для каждого студента
48             список книг
49             .flatMap(s -> s.getBooks().stream())
50             // Отсортировать книги по количеству
51             страниц
52             .sorted(Comparator.comparingInt(Book
53                 ::getPages))
54             // Оставить только уникальные книги
55             .distinct()
56             // Отфильтровать книги, выпущенные
57             после 2000 года
58             .filter(book -> book.getYear() > 2000
59             )
60             // Ограничить на 3 элементах
61             .limit(3)
62             // Получить годы выпуска
63             .map(Book::getYear)
64             // Найти первый подходящий элемент
65             .findFirst()
66             // Вывести результат
67             .ifPresentOrElse(
68                 year -> System.out.println("
69                 Год выпуска найденной книги: " + year),
70                 () -> System.out.println("

```

```
64 Книга отсутствует")
65             );
66     }
67 }
```

```

1 import java.util.*;
2
3 public class HashSet {
4     private static final int SIZE = 16;
5     private static final float LOAD_FACTOR = 0.75f;
6
7     private Node[] table; //Массив корзин
8     private int size; //количество элементов
9     private int capacity; //текущая вместимость
10    private float loadFactor; // коэффициент
    загрузки
11    private int threshold; //порог для resize
12
13    //Внутренний класс для элементов
14    private static class Node {
15        Object key; // хранитель элементов
16        Node next; // ссылка на следующий узел
17        int hash; // кэширование хэш ключа
18
19        Node(Object key, Node next, int hash) {
20            this.key = key;
21            this.next = next;
22            this.hash = hash;
23        }
24    }
25
26    //Конструктор по умолчанию
27    public HashSet(){
28        this(SIZE, LOAD_FACTOR);
29    }
30
31    //Основной конструктор
32    public HashSet(int initialCapacity, float
    loadFactor){
33        this.capacity = initialCapacity; //
    вместимость склада
34        this.loadFactor = loadFactor; //коэффициент
    загрузки
35        this.threshold = (int)(capacity *
    loadFactor); //порог срабатывания расширений
36        this.table = new Node[capacity]; //массив

```

```

36 корзин
37         this.size = 0; //счетчик элементов
38     }
39
40
41         //Вспомогательный метод для вычисления хэша
42     private int hash(Object key){
43         if(key == null) return 0; // null всегда в
корзине 0
44         int h = key.hashCode(); //получаем
стандартный хэш
45         return (h ^ (h >>> 16)); //улучшаем
распределение
46     }
47
48     //метод add
49     public boolean add(Object key){
50         // 1. Вычисляем хэш и индекс полки
51         int hash = hash(key); //вычисляем улучшенный
кэш
52         int index = (capacity-1) & hash; //находим
номер полки
53
54         // 2. Проверяем дубликаты
55         Node current = table[index];
56         while (current != null){
57             if(current.hash == hash && (current.key
== key || (key != null && key.equals(current.key))))
58                 {
59                     //нашли дубликату - не добавляем
60                     return false;
61                 }
62             current = current.next;
63         }
64
65         // 3. Добавляем новый элемент
66         table[index] = new Node(key, current, hash);
67         size++;
68
69         // 4. Проверяем расширение
70         if (size > threshold){

```

```

71         resize();
72     }
73     return true; // Добавлено
74 }
75
76 public boolean contains (Object key){
77     int hash = hash(key); //вычисляем улучшаем
78     кэш int index = (capacity-1) & hash; //находим
79     номер полки
80     //идем по цепочке
81     Node current = table[index];
82     while (current != null){
83         //та же проверка
84         if(current.hash == hash && (current.key
85         == key || (key != null && key.equals(current.key
86         )))){
87             return true; //нашли элемент
88         }
89         current = current.next;
90     }
91     return false; //прошли до конца и не нашли
92 }
93 //метод remove
94 public boolean remove(Object key){
95     int hash = hash(key);
96     int index = (capacity-1) & hash;
97
98     Node current = table[index];
99     Node prev = null;
100
101     while (current != null){
102         if (current.hash == hash && (current.key
103         == key || (key != null && key.equals(current.key
104         )))){
105             //Нашли элемент для удаления
106             //удаляем

```



```

106         prev = current; // запоминаем текущую
           как предыдущую
107         current = current.next; //переходим к
           следующей
108         if (prev == null){
109             //удаляем коробку на полке
110             table[index] = table[index].next;
111         }else{
112             //удаляем с середины или с конца
113             prev.next = current.next;
114         }
115         size--; //уменьшаем счетчик
116         return true; //удалили
117     }
118     return false; //не нашли
119 }
120
121 public void resize(){
122     int newCapacity = capacity * 2; //увеличиваем
       вместимость в 2 раза
123     int newThreshold = (int)(newCapacity *
loadFactor); //пересчитываем порог для нового размера
124
125     Node[] newTable = new Node[newCapacity];
126
127     //проходим по полкам
128     for(int i = 0; i < capacity; i++){
129         Node current = table[i];
130
131         //проходим по каждой коробке на полке
132         while (current != null){
133             Node next = current.next; //
       запоминаем следующую
134
135             //Вычисляем НОВЫЙ индекс для коробки
136             int nextIndex = (newCapacity-1) &
current.hash;
137
138             //Вставляем коробку в новую таблицу
139             current.next = newTable[nextIndex];
140             newTable[nextIndex] = current;

```

```
141
142             current = next; // переходим к
           следующей коробке
143         }
144     }
145     //обновляем все переменные
146     table = newTable; // исп новый массив
147     capacity = newCapacity; //обн вместимость
148     threshold = newThreshold; //обн порог
149 }
150
151 }
```

```
1 import java.util.Collection;
2 import java.util.List;
3 import java.util.ArrayList;
4
5 public class Student {
6     public String name;
7     List<Book> books;
8
9     public Student (String name){
10         this.name = name;
11         this.books = new ArrayList<>();
12     }
13
14
15     public void addBook(Book book){
16         books.add(book);
17     }
18
19     public String getName() {
20         return name;
21     }
22     public List<Book> getBooks() {
23         return books;
24     }
25
26     @Override
27     public String toString() {
28         return "Имя студента: "+name+"Список книг"+
29         books;
30     }
31 }
32
33
34
```

```
1 public class ArrayList {
2     private static final int DEFAULT_CAPACITY = 100;
3     private static final float GROW_FACTOR = 1.5f;
4
5
6     private Object[] elements; // внутренний массив
       для хранения элементов
7     private int size; // текущее кол-во элементов в
       списке
8     private int capacity; // текущая вместимость
       массива
9
10    public ArrayList() {
11        this.capacity = DEFAULT_CAPACITY; //
       устанавливаем вместимость = 10
12        this.elements = new Object[capacity]; //
       создаем массив на 10 элементов
13        this.size = 0; // пустой список
14    }
15
16    public ArrayList(int initialCapacity) {
17        this.capacity = initialCapacity; //
       используем переданную вместимость
18        this.elements = new Object[capacity]; //
       создаем массив нужного размера
19        this.size = 0; // пустой список
20    }
21
22    public void add(Object element) {
23        // проверяем, не заполнен ли массив полностью
24        if (size >= capacity) {
25            resize();
26        }
27        // добавляем элемент в свободную ячейку
28        elements[size] = element;
29
30        // >счетчик элементов
31        size++;
32    }
33
34    private void add(int index, Object element) {
```

```
35         //проверяем индекс
36         if (index > size || index < 0) {
37             throw new ArrayIndexOutOfBoundsException
38         ();
39         }
40         //Расширяем массив?
41         if (size >= capacity) {
42             resize();
43         }
44         //сдвигаем элементы справа от index
45         for (int i = size; i > index; i--) {
46             elements[i] = elements[i - 1];
47         }
48         //всатвляем новый элемент
49         elements[index] = element;
50
51         //увеличиваем счетчик
52         size++;
53     }
54
55     private void resize() {
56         //вычисляем новый размер
57         int newCapacity = (int)(capacity * GROW_FACTOR
58     );
59
60         //новый массив
61         Object[] newElements = new Object[newCapacity
62     ];
63
64         for(int i = 0; i < size; i++) {
65             newElements[i] = elements[i];
66         }
67
68         //обновление ссылки на новый массив и новую
69         вместимость
70         elements = newElements;
71         capacity = newCapacity;
72     }
73
74     public Object get(int index) {
```

```
72         //проверка наличие элемента
73         if (index >= size || index < 0) {
74             throw new ArrayIndexOutOfBoundsException
75         };
76         //возвращаем элемент
77         return elements[index];
78     }
79
80     public Object remove(int index) {
81         //Проверка валидности
82         if (index >= size || index < 0) {
83             throw new ArrayIndexOutOfBoundsException
84         };
85         //сохраняем и удаляем элементы для возврата
86         Object removedElement = elements[index];
87
88         //сдвиг всех элементов вправо от удаляемого
89         на 1 позицию влево
90         for (int i = index; i < size - 1; i++) {
91             elements[i] = elements[i + 1];
92         }
93
94         //очищаем последнюю ячейку
95         elements[size - 1] = null;
96         //уменьшаем счетчик элементов
97         size--;
98         //возвращаем удаленный элемент
99         return removedElement;
100     }
101 }
```