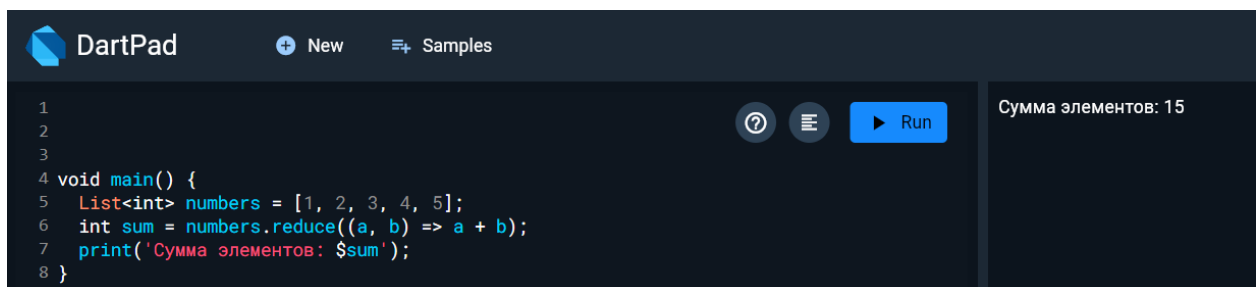


Задание № 1: Сумма элементов списка

Создайте список целых чисел и найдите сумму всех его элементов.

Описание алгоритма решения:

1. Создайте список целых чисел.
2. Используйте метод reduce или цикл for для суммирования элементов списка.
3. Выведите результат.



```
1
2
3
4 void main() {
5   List<int> numbers = [1, 2, 3, 4, 5];
6   int sum = numbers.reduce((a, b) => a + b);
7   print('Сумма элементов: $sum');
8 }
```

Сумма элементов: 15

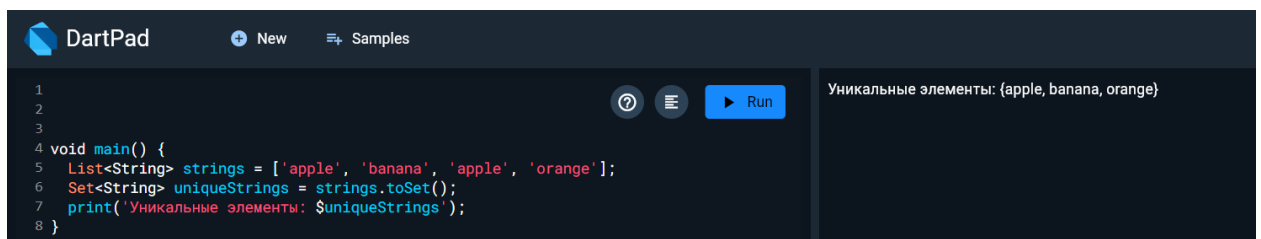
Рисунок №1 – Результат решения задания №1.

Задание № 2: Уникальные элементы

Создайте список строк и найдите все уникальные элементы в нем.

Описание алгоритма решения:

1. Создайте список строк.
2. Преобразуйте список в множество (Set), чтобы удалить дубликаты.
3. Выведите результат



```
1
2
3
4 void main() {
5   List<String> strings = ['apple', 'banana', 'apple', 'orange'];
6   Set<String> uniqueStrings = strings.toSet();
7   print('Уникальные элементы: $uniqueStrings');
8 }
```

Уникальные элементы: {apple, banana, orange}

Рисунок №2 – Результат решения задания №2.

Задание № 3: Фильтрация списка

Создайте список целых чисел и отфильтруйте его, оставив только четные числа.

Описание алгоритма решения:

1. Создайте список целых чисел.
2. Используйте метод where для фильтрации списка по условию ($\text{число} \% 2 == 0$).
3. Выведите результат



```
1
2
3 void main() {
4   List<int> numbers = [1, 2, 3, 4, 5, 6];
5   List<int> evenNumbers = numbers.where((number) => number % 2 ==
6   0).toList();
7   print('Четные числа: $evenNumbers');
8 }
```

Четные числа: [2, 4, 6]

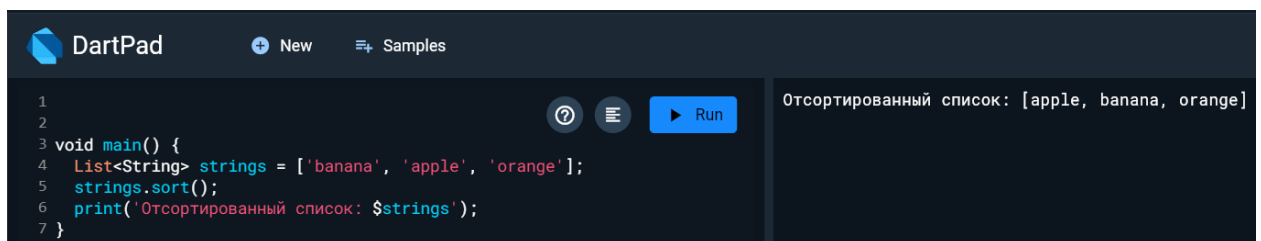
Рисунок №3 – Результат решения задания №3.

Задание № 4: Сортировка списка

Создайте список строк и отсортируйте его в алфавитном порядке.

Описание алгоритма решения:

1. Создайте список строк.
2. Используйте метод sort для сортировки списка.
3. Выведите результат



```
1
2
3 void main() {
4   List<String> strings = ['banana', 'apple', 'orange'];
5   strings.sort();
6   print('Отсортированный список: $strings');
7 }
```

Отсортированный список: [apple, banana, orange]

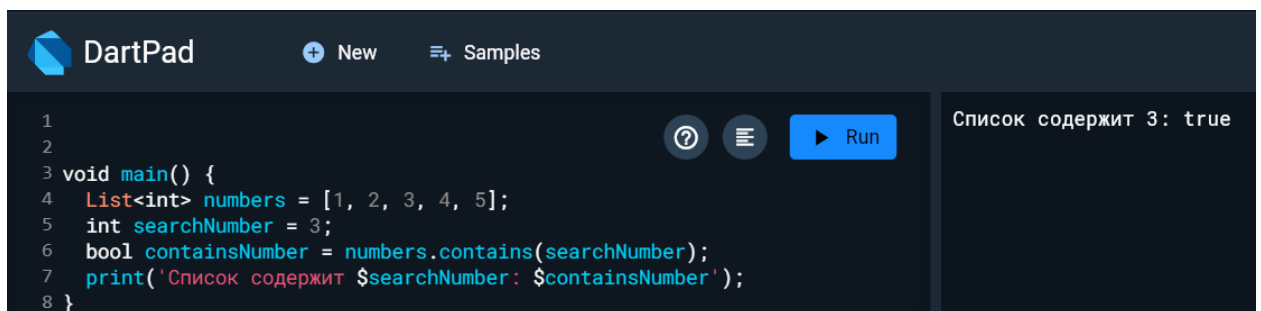
Рисунок №4 – Результат решения задания №4.

Задание № 5: Поиск элемента в списке

Создайте список целых чисел и проверьте, содержится ли в нем определенное число.

Описание алгоритма решения:

1. Создайте список целых чисел.
2. Используйте метод *contains* для проверки наличия элемента в списке.
3. Выведите результат



```
1
2
3 void main() {
4   List<int> numbers = [1, 2, 3, 4, 5];
5   int searchNumber = 3;
6   bool containsNumber = numbers.contains(searchNumber);
7   print('Список содержит $searchNumber: $containsNumber');
8 }
```

Список содержит 3: true

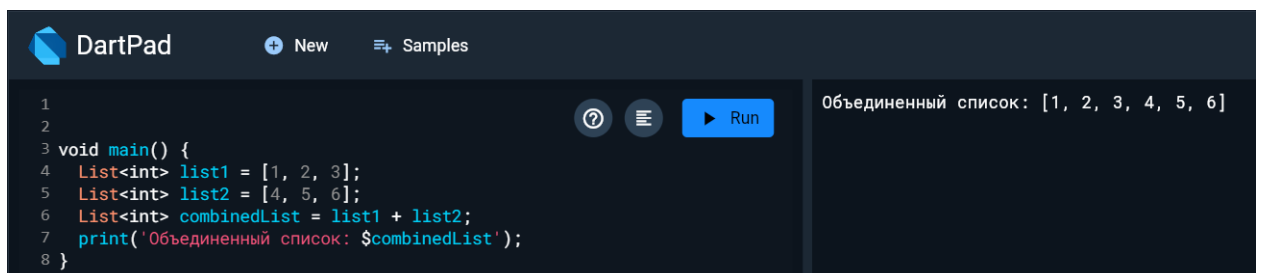
Рисунок №5 – Результат решения задания №5.

Задание № 6: Объединение списков

Создайте два списка целых чисел и объедините их в один.

Описание алгоритма решения:

1. Создайте два списка целых чисел.
2. Используйте оператор `+` или метод *addAll* для объединения списков.
3. Выведите результат



```
1
2
3 void main() {
4   List<int> list1 = [1, 2, 3];
5   List<int> list2 = [4, 5, 6];
6   List<int> combinedList = list1 + list2;
7   print('Объединенный список: $combinedList');
8 }
```

Объединенный список: [1, 2, 3, 4, 5, 6]

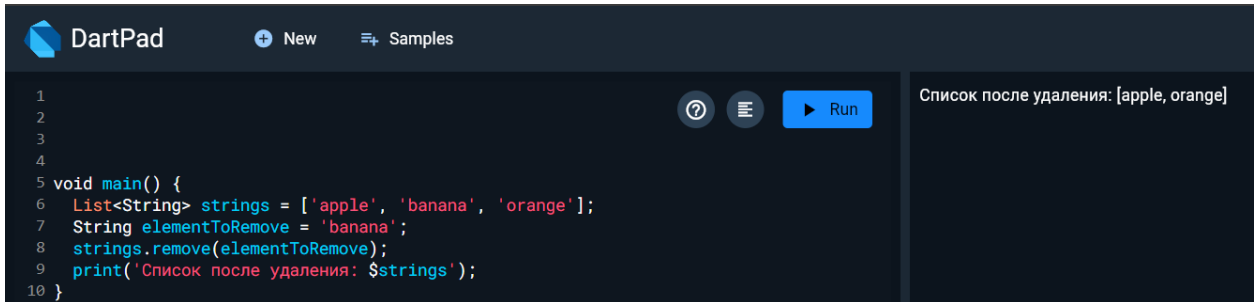
Рисунок №6 – Результат решения задания №6.

Задание № 7: Удаление элемента из списка

Создайте список строк и удалите из него определенный элемент.

Описание алгоритма решения:

1. Создайте список строк.
2. Используйте метод remove для удаления элемента из списка.
3. Выведите результат.



```
1
2
3
4
5 void main() {
6   List<String> strings = ['apple', 'banana', 'orange'];
7   String elementToRemove = 'banana';
8   strings.remove(elementToRemove);
9   print('Список после удаления: $strings');
10 }
```

Список после удаления: [apple, orange]

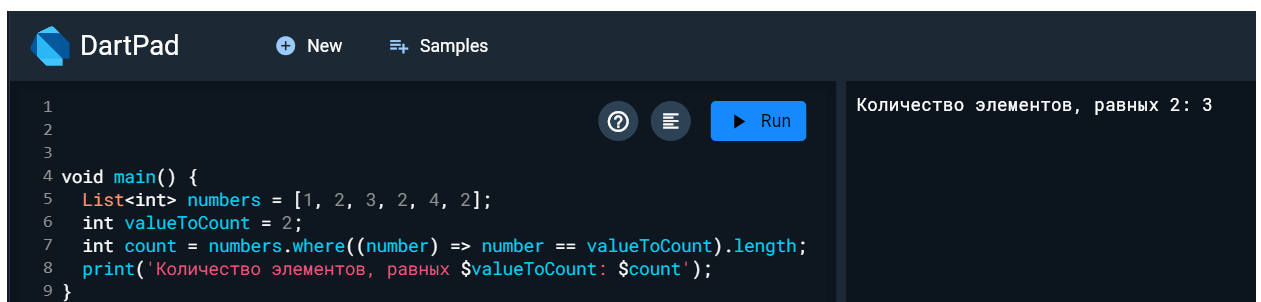
Рисунок №7 – Результат решения задания №7.

Задание № 8: Подсчет элементов в списке

Создайте список целых чисел и подсчитайте количество элементов, равных определенному значению.

Описание алгоритма решения:

1. Создайте список целых чисел.
2. Используйте метод where для фильтрации элементов, равных определенному значению, и метод length для подсчета их количества.
3. Выведите результат



```
1
2
3
4 void main() {
5   List<int> numbers = [1, 2, 3, 2, 4, 2];
6   int valueToCount = 2;
7   int count = numbers.where((number) => number == valueToCount).length;
8   print('Количество элементов, равных $valueToCount: $count');
9 }
```

Количество элементов, равных 2: 3

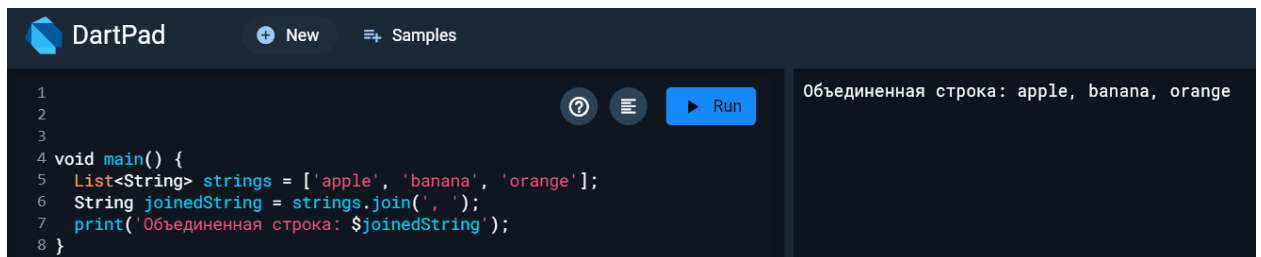
Рисунок №8 – Результат решения задания №8.

Задание № 9: Преобразование списка в строку

Создайте список строк и объедините его элементы в одну строку с разделителем.

Описание алгоритма решения:

1. Создайте список строк.
2. Используйте метод *join* для объединения элементов списка в одну строку с указанным разделителем.
3. Выведите результат



```
1
2
3
4 void main() {
5   List<String> strings = ['apple', 'banana', 'orange'];
6   String joinedString = strings.join(', ');
7   print('Объединенная строка: $joinedString');
8 }
```

Объединенная строка: apple, banana, orange

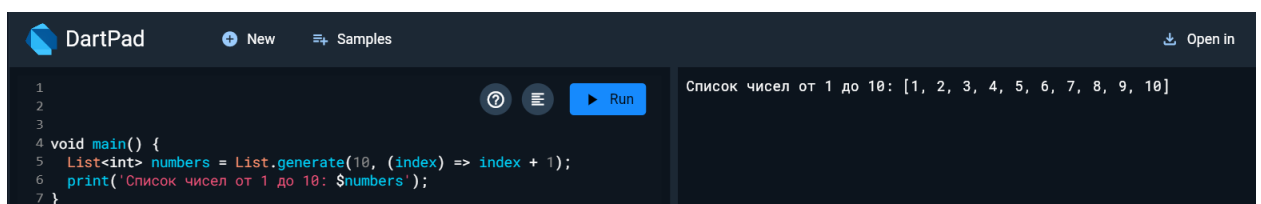
Рисунок №9 – Результат решения задания №9.

Задание № 10: Создание списка из диапазона

Создайте список целых чисел, содержащий числа от 1 до 10.

Описание алгоритма решения:

1. Используйте метод *List.generate* или *List.from* для создания списка из диапазона чисел.
2. Выведите результат



```
1
2
3
4 void main() {
5   List<int> numbers = List.generate(10, (index) => index + 1);
6   print('Список чисел от 1 до 10: $numbers');
7 }
```

Список чисел от 1 до 10: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

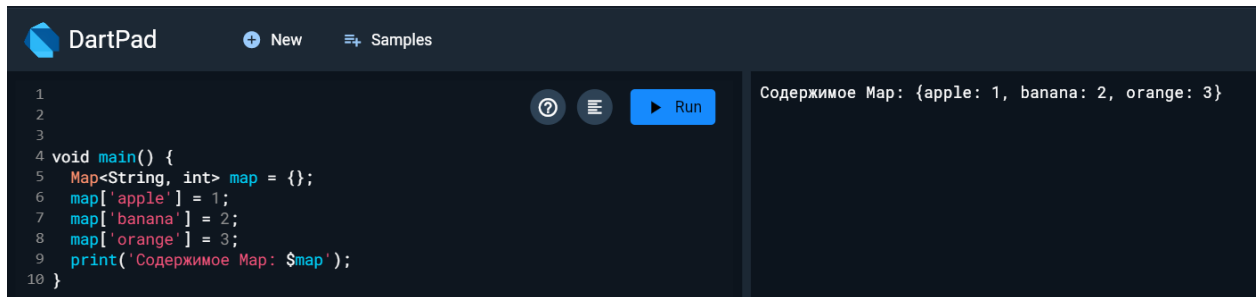
Рисунок №10 – Результат решения задания №10.

Задание № 11: Использование Map

Создайте Map, где ключами будут строки, а значениями — целые числа. Добавьте несколько пар ключ-значение и выведите их.

Описание алгоритма решения:

1. Создайте пустой Map.
2. Добавьте несколько пар ключ-значение в Map.
3. Выведите содержимое Map



The screenshot shows the DartPad interface. On the left, the code editor contains the following Dart code:

```
1
2
3
4 void main() {
5   Map<String, int> map = {};
6   map['apple'] = 1;
7   map['banana'] = 2;
8   map['orange'] = 3;
9   print('Содержимое Map: $map');
10 }
```

On the right, the output console displays the result of the code execution: "Содержимое Map: {apple: 1, banana: 2, orange: 3}".

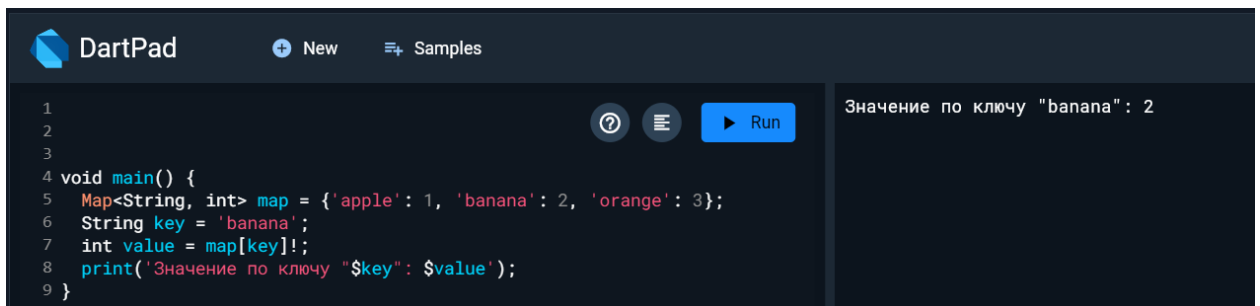
Рисунок №11 – Результат решения задания №11.

Задание № 12: Получение значения по ключу

Создайте Map и получите значение по определенному ключу.

Описание алгоритма решения:

1. Создайте Map.
2. Используйте оператор `[]` для получения значения по ключу.
3. Выведите результат.



The screenshot shows the DartPad interface. On the left, the code editor contains the following Dart code:

```
1
2
3
4 void main() {
5   Map<String, int> map = {'apple': 1, 'banana': 2, 'orange': 3};
6   String key = 'banana';
7   int value = map[key]!;
8   print('Значение по ключу "$key": $value');
9 }
```

On the right, the output console displays the result of the code execution: "Значение по ключу "banana": 2".

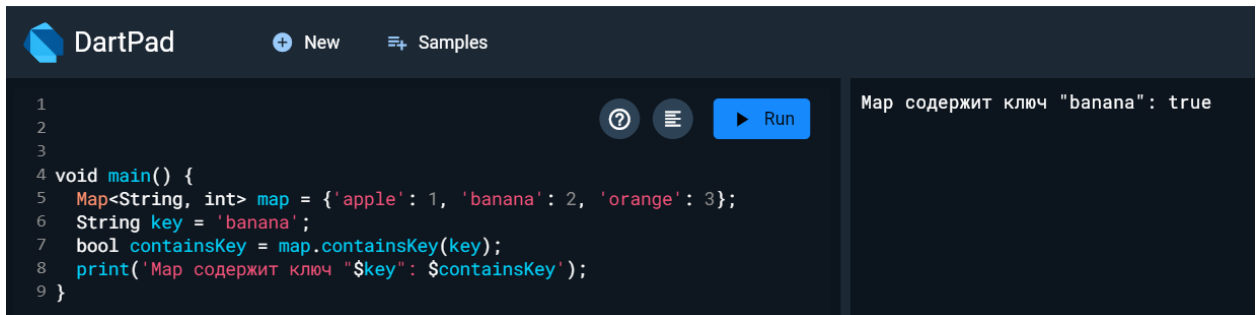
Рисунок №12 – Результат решения задания №12.

Задание № 13: Проверка наличия ключа в Map

Создайте Map и проверьте, содержится ли в нем определенный ключ.

Описание алгоритма решения:

1. Создайте Map.
2. Используйте метод containsKey для проверки наличия ключа в Map.
3. Выведите результат



```
1
2
3
4 void main() {
5   Map<String, int> map = {'apple': 1, 'banana': 2, 'orange': 3};
6   String key = 'banana';
7   bool containsKey = map.containsKey(key);
8   print('Map содержит ключ "$key": $containsKey');
9 }
```

Map содержит ключ "banana": true

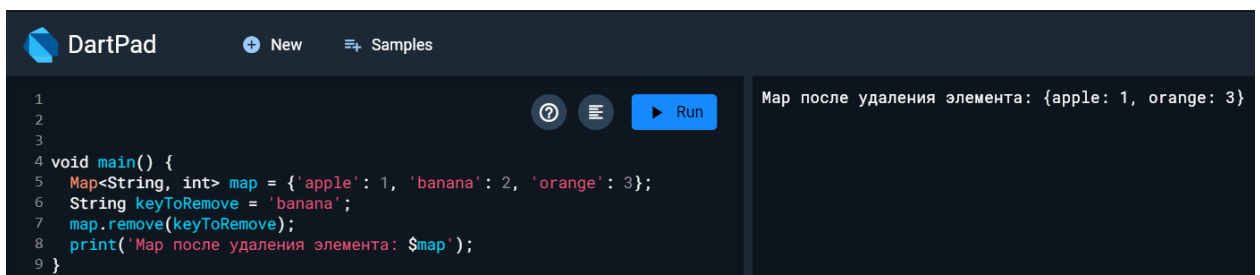
Рисунок №13 – Результат решения задания №13.

Задание № 14: Удаление элемента из Map

Создайте Map и удалите из него элемент по определенному ключу.

Описание алгоритма решения:

1. Создайте Map.
2. Используйте метод remove для удаления элемента по ключу.
3. Выведите результат



```
1
2
3
4 void main() {
5   Map<String, int> map = {'apple': 1, 'banana': 2, 'orange': 3};
6   String keyToRemove = 'banana';
7   map.remove(keyToRemove);
8   print('Map после удаления элемента: $map');
9 }
```

Map после удаления элемента: {apple: 1, orange: 3}

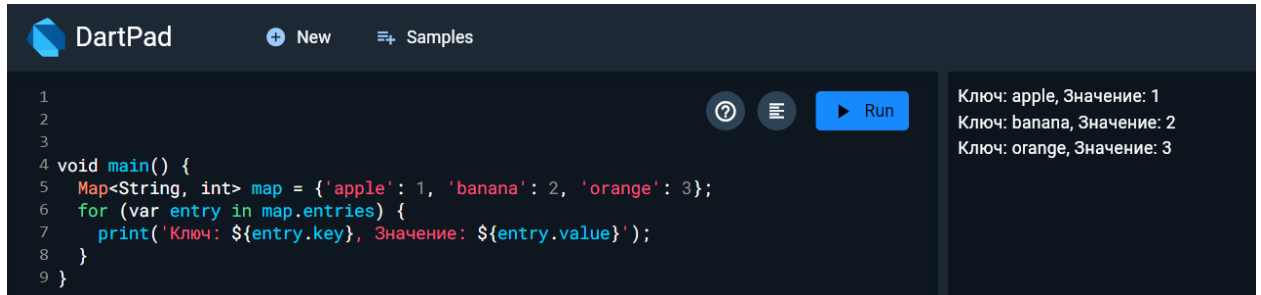
Рисунок №14 – Результат решения задания №14.

Задание № 15: Обход Map

Создайте Map и выведите все его ключи и значения.

Описание алгоритма решения:

1. Создайте Map.
2. Используйте цикл for-in для обхода Map и вывода ключей и значений.



The screenshot shows the DartPad interface. The code editor contains the following Dart code:

```
1
2
3
4 void main() {
5   Map<String, int> map = {'apple': 1, 'banana': 2, 'orange': 3};
6   for (var entry in map.entries) {
7     print('Ключ: ${entry.key}, Значение: ${entry.value}');
8   }
9 }
```

The output console on the right displays the results of the program execution:

```
Ключ: apple, Значение: 1
Ключ: banana, Значение: 2
Ключ: orange, Значение: 3
```

Рисунок №15 – Результат решения задания №15.

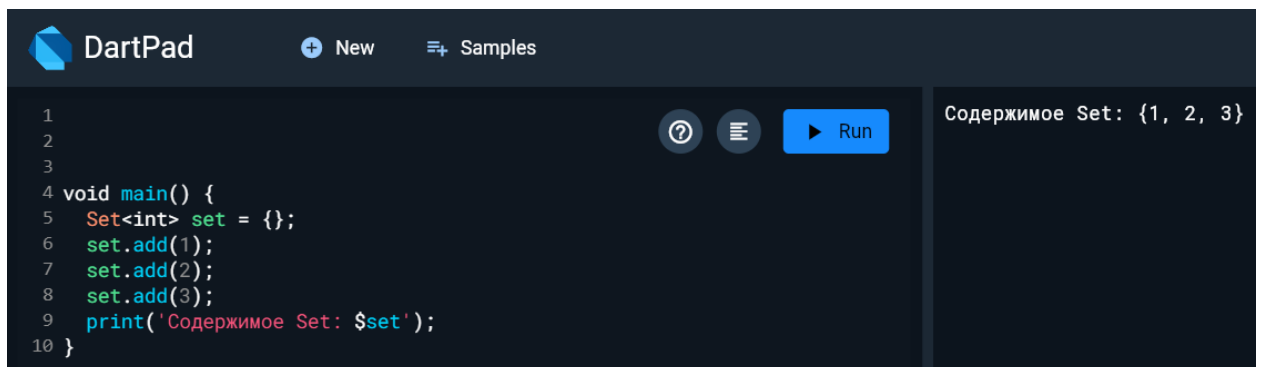
Задание № 16: Использование Set

Создайте Set целых чисел и добавьте в него несколько элементов.

Выведите содержимое Set.

Описание алгоритма решения:

1. Создайте пустой Set.
2. Добавьте несколько элементов в Set.
3. Выведите содержимое Set



The screenshot shows the DartPad interface. The code editor contains the following Dart code:

```
1
2
3
4 void main() {
5   Set<int> set = {};
6   set.add(1);
7   set.add(2);
8   set.add(3);
9   print('Содержимое Set: $set');
10 }
```

The output console on the right displays the result of the program execution:

```
Содержимое Set: {1, 2, 3}
```

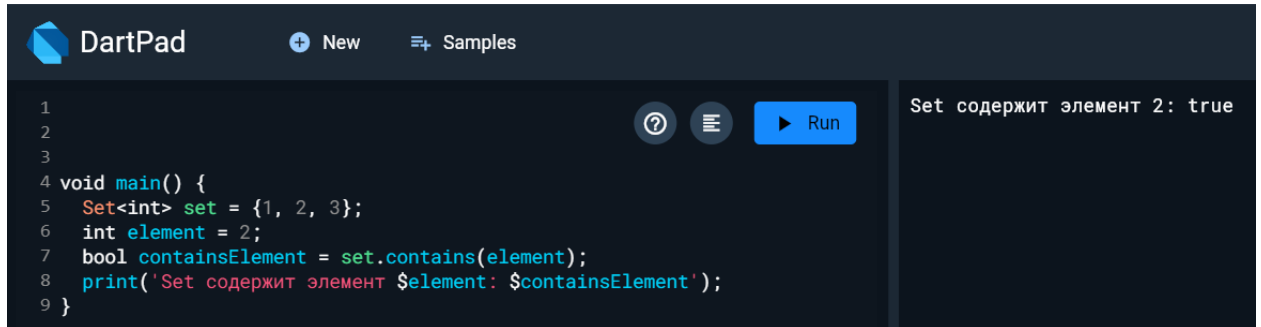
Рисунок №16 – Результат решения задания №16.

Задание № 17: Проверка наличия элемента в Set

Создайте Set и проверьте, содержится ли в нем определенный элемент.

Описание алгоритма решения:

1. Создайте Set.
2. Используйте метод contains для проверки наличия элемента в Set.
3. Выведите результат.



```
1
2
3
4 void main() {
5   Set<int> set = {1, 2, 3};
6   int element = 2;
7   bool containsElement = set.contains(element);
8   print('Set содержит элемент $element: $containsElement');
9 }
```

Set содержит элемент 2: true

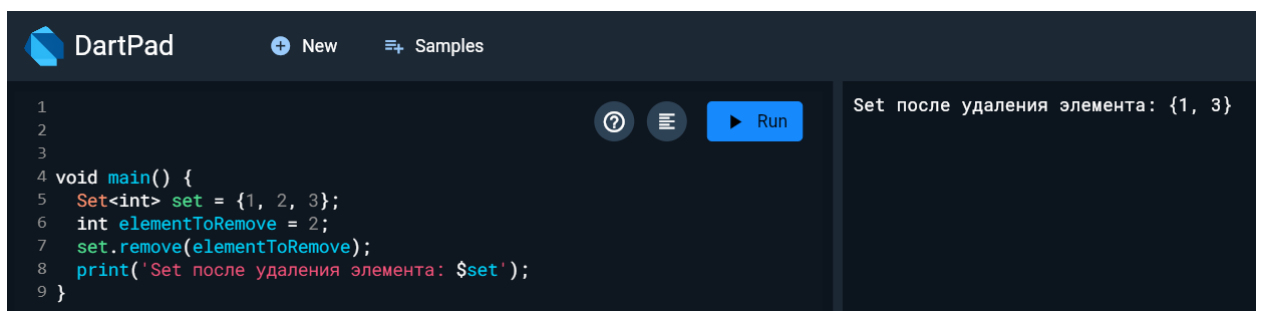
Рисунок №17 – Результат решения задания №17.

Задание № 18: Удаление элемента из Set

Создайте Set и удалите из него определенный элемент.

Описание алгоритма решения:

1. Создайте Set.
2. Используйте метод remove для удаления элемента из Set.
3. Выведите результат



```
1
2
3
4 void main() {
5   Set<int> set = {1, 2, 3};
6   int elementToRemove = 2;
7   set.remove(elementToRemove);
8   print('Set после удаления элемента: $set');
9 }
```

Set после удаления элемента: {1, 3}

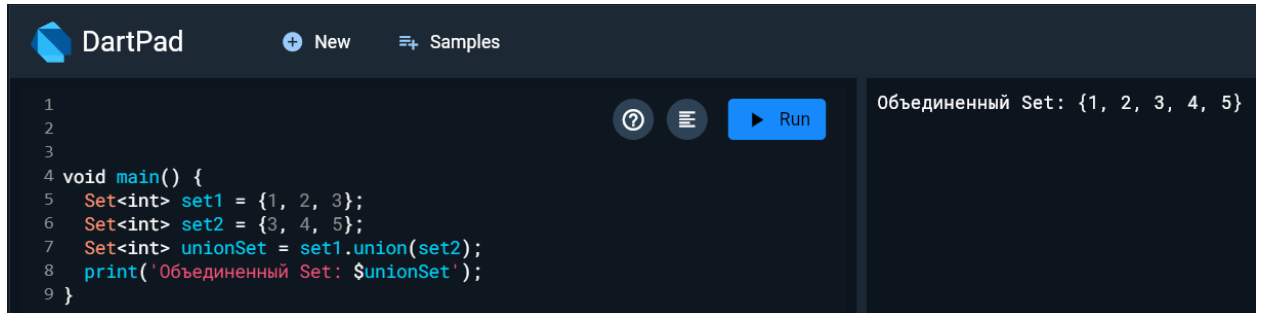
Рисунок №18 – Результат решения задания №18.

Задание № 19: Объединение Set

Создайте два Set и объедините их в один.

Описание алгоритма решения:

1. Создайте два Set.
2. Используйте метод union для объединения Set.
3. Выведите результат.



```
1
2
3
4 void main() {
5   Set<int> set1 = {1, 2, 3};
6   Set<int> set2 = {3, 4, 5};
7   Set<int> unionSet = set1.union(set2);
8   print('Объединенный Set: $unionSet');
9 }
```

Объединенный Set: {1, 2, 3, 4, 5}

Рисунок №19 – Результат решения задания №19.

Задание № 20: Пересечение Set

Создайте два Set и найдите их пересечение.

Описание алгоритма решения:

1. Создайте два Set.
2. Используйте метод intersection для нахождения пересечения Set.
3. Выведите результат



```
1
2
3
4 void main() {
5   Set<int> set1 = {1, 2, 3};
6   Set<int> set2 = {3, 4, 5};
7   Set<int> intersectionSet = set1.intersection(set2);
8   print('Пересечение Set: $intersectionSet');
9 }
```

Пересечение Set: {3}

Рисунок №20 – Результат решения задания №20.

Индивидуальные задания для закрепления материала

Базовые задания:

1. **Телефонная книга:** Создайте класс PhoneBook, который будет хранить записи о контактах в коллекции HashMap<String, String>. Реализуйте методы для добавления, удаления, поиска и вывода всех контактов.
2. **Список задач:** Создайте класс ToDoList, который будет хранить задачи в коллекции ArrayList<String>. Реализуйте методы для добавления, удаления, пометки задачи как выполненной и вывода всех задач.
3. **Корзина покупок:** Создайте класс ShoppingCart, который будет хранить товары в коллекции LinkedList<String>. Реализуйте методы для добавления, удаления, изменения количества товаров и вывода содержимого корзины.
4. **Стек и очередь:** Реализуйте классы Stack<T> и Queue<T> с использованием коллекций LinkedList<T>. Реализуйте методы push, pop, peek для стека и enqueue, dequeue, peek для очереди.
5. **Сортировка коллекций:** Создайте коллекцию ArrayList<Integer> и заполните ее случайными числами. Отсортируйте коллекцию с использованием метода Collections.sort() и выведите результат.

Средний уровень:

1. **Универсальный список:** Создайте класс GenericList<T>, который будет реализовывать интерфейс List<T> с использованием коллекции ArrayList<T>. Реализуйте методы add, remove, get, size.
2. **Универсальный стек:** Создайте класс GenericStack<T>, который будет реализовывать интерфейс Stack<T> с использованием коллекции LinkedList<T>. Реализуйте методы push, pop, peek.
3. **Универсальная очередь:** Создайте класс GenericQueue<T>, который будет реализовывать интерфейс Queue<T> с использованием коллекции LinkedList<T>. Реализуйте методы enqueue, dequeue, peek.
4. **Универсальный словарь:** Создайте класс GenericMap<K, V>, который будет реализовывать интерфейс Map<K, V> с использованием коллекции HashMap<K, V>. Реализуйте методы put, get, remove, containsKey, size.
5. **Универсальный набор:** Создайте класс GenericSet<T>, который будет реализовывать интерфейс Set<T> с использованием

коллекции `HashSet<T>`.

Реализуйте

методы `add`, `remove`, `contains`, `size`.

6. **Универсальный список с ограничениями:** Создайте класс `BoundedList<T extends Number>`, который будет реализовывать интерфейс `List<T>` с использованием коллекции `ArrayList<T>`. Реализуйте методы `add`, `remove`, `get`, `size`.
7. **Универсальный стек с ограничениями:** Создайте класс `BoundedStack<T extends Comparable<T>>`, который будет реализовывать интерфейс `Stack<T>` с использованием коллекции `LinkedList<T>`. Реализуйте методы `push`, `pop`, `peek`.
8. **Универсальная очередь с ограничениями:** Создайте класс `BoundedQueue<T extends Comparable<T>>`, который будет реализовывать интерфейс `Queue<T>` с использованием коллекции `LinkedList<T>`. Реализуйте методы `enqueue`, `dequeue`, `peek`.
9. **Универсальный словарь с ограничениями:** Создайте класс `BoundedMap<K extends Comparable<K>, V>`, который будет реализовывать интерфейс `Map<K, V>` с использованием коллекции `TreeMap<K, V>`. Реализуйте методы `put`, `get`, `remove`, `containsKey`, `size`.
10. **Универсальный набор с ограничениями:** Создайте класс `BoundedSet<T extends Comparable<T>>`, который будет реализовывать интерфейс `Set<T>` с использованием коллекции `TreeSet<T>`. Реализуйте методы `add`, `remove`, `contains`, `size`.

Продвинутый уровень:

1. **Универсальный список с итератором:** Создайте класс `GenericList<T>`, который будет реализовывать интерфейс `List<T>` с использованием коллекции `ArrayList<T>`. Реализуйте метод `iterator()`, который будет возвращать объект класса `GenericListIterator<T>`, реализующий интерфейс `Iterator<T>`.
2. **Универсальный стек с итератором:** Создайте класс `GenericStack<T>`, который будет реализовывать интерфейс `Stack<T>` с использованием коллекции `LinkedList<T>`. Реализуйте метод `iterator()`, который будет возвращать объект класса `GenericStackIterator<T>`, реализующий интерфейс `Iterator<T>`.
3. **Универсальная очередь с итератором:** Создайте класс `GenericQueue<T>`, который будет реализовывать интерфейс `Queue<T>` с использованием коллекции `LinkedList<T>`. Реализуйте

метод `iterator()`, который будет возвращать объект класса `GenericQueueIterator<T>`, реализующий интерфейс `Iterator<T>`.

4. **Универсальный словарь с итератором**: Создайте класс `GenericMap<K, V>`, который будет реализовывать интерфейс `Map<K, V>` с использованием коллекции `HashMap<K, V>`. Реализуйте метод `entrySet()`, который будет возвращать объект класса `GenericMapEntrySet<K, V>`, реализующий интерфейс `Set<Map.Entry<K, V>>`.
5. **Универсальный набор с итератором**: Создайте класс `GenericSet<T>`, который будет реализовывать интерфейс `Set<T>` с использованием коллекции `HashSet<T>`. Реализуйте метод `iterator()`, который будет возвращать объект класса `GenericSetIterator<T>`, реализующий интерфейс `Iterator<T>`.
6. **Универсальный список с компаратором**: Создайте класс `GenericList<T>`, который будет реализовывать интерфейс `List<T>` с использованием коллекции `ArrayList<T>`. Реализуйте метод `sort(Comparator<? super T> c)`, который будет сортировать элементы списка с использованием переданного компаратора.
7. **Универсальный стек с компаратором**: Создайте класс `GenericStack<T>`, который будет реализовывать интерфейс `Stack<T>` с использованием коллекции `LinkedList<T>`. Реализуйте метод `sort(Comparator<? super T> c)`, который будет сортировать элементы стека с использованием переданного компаратора.
8. **Универсальная очередь с компаратором**: Создайте класс `GenericQueue<T>`, который будет реализовывать интерфейс `Queue<T>` с использованием коллекции `LinkedList<T>`. Реализуйте метод `sort(Comparator<? super T> c)`, который будет сортировать элементы очереди с использованием переданного компаратора.
9. **Универсальный словарь с компаратором**: Создайте класс `GenericMap<K, V>`, который будет реализовывать интерфейс `Map<K, V>` с использованием коллекции `HashMap<K, V>`. Реализуйте метод `sort(Comparator<? super K> c)`, который будет сортировать элементы словаря по ключам с использованием переданного компаратора.
10. **Универсальный набор с компаратором**: Создайте класс `GenericSet<T>`, который будет реализовывать интерфейс `Set<T>` с использованием коллекции `HashSet<T>`. Реализуйте метод `sort(Comparator<? super T> c)`, который будет сортировать элементы набора с использованием переданного компаратора.