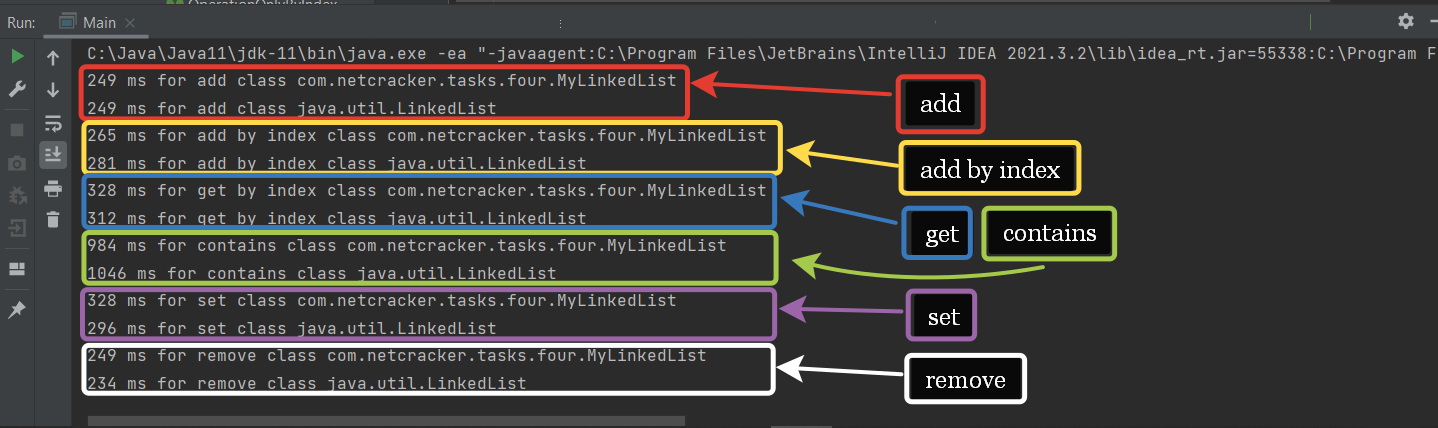
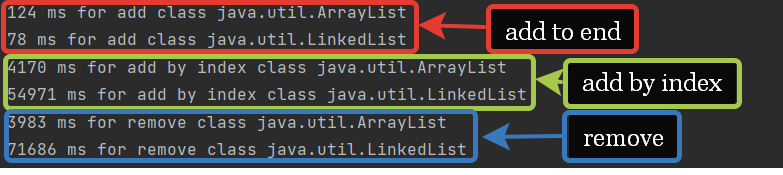
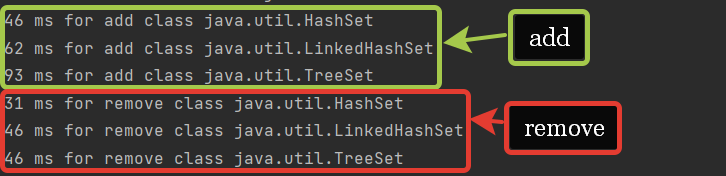
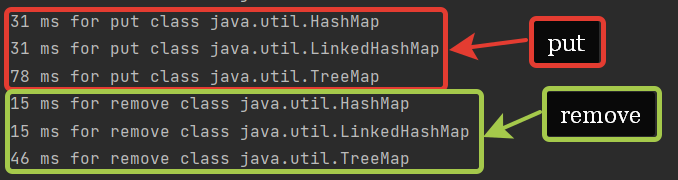
1. Итог: Односвязный созданный MyLinkedList немного проигрывает двусвязному LinkedList из стандартной библиотеки, все операции, где необходимо найти Node в двусвязном списке будут работать быстрее, тестовые данные от 10000 - 100000 элементов для разных операций. В качестве добавляемого типа был взят класс Person  
   
2. Сравнение коллекций из стандартной библиотеки
3. ArrayList vs LinkedList: добавление элементов в начало или конец коллекции выполняется быстрее у LinkedList, но операции добавления по индексу и удаление производятся быстрее. LinkedList рекомендуется использовать при создании коллекции для хранения и добавления/удаления элементов в начале или конце, для более быстрого поиска элементов, добавления, удаления рекомендуется использовать ArrayList. Выборка 100000-200000 элементов.  
   
4. HashSet vs LinkedHashSet vs TreeSet: HashSet показал лучшие результаты для добавления и удаления элементов, рекомендуется его использовать для работы с неупорядоченными и уникальными элементами. LinkedHashSet элементы упорядочены в порядке добавления в небольшой ущерб быстродействию. TreeSet упорядоченная коллекция, в ущерб производительности из-за сортировки и перестройки дерева. Какую из коллекций использовать зависит от необходимости упорядочивания элементов. Выборка 100000-200000 элементов.  
   
5. HashMap vs LinkedHashMap vs TreeMap: HashMap показал лучшие результаты для добавления и удаления элементов, рекомендуется его использовать для работы с неупорядоченными элементами. LinkedHashMap элементы упорядочены в порядке добавления с возможным небольшим ущербом быстродействию. LinkedHashMap существует только для итерации в порядке вставки (или доступа). TreeMap упорядоченная коллекция (красно-черное дерево), в ущерб производительности из-за сортировки и перестройки дерева. Какую из коллекций использовать зависит от необходимости упорядочивания элементов. Выборка 100000-200000 элементов.   
     
   

**Вывод**: **ArrayList** следует использовать, когда в приоритете доступ по индексу, так как эти операции выполняются за константное время. Добавление в конец списка в среднем тоже выполняется за константное время. Кроме того в ArrayList нет дополнительных расходов на хранение связки между элементами. Минусы в скорости вставки/удаления элементов находящихся не в конце списка, так как при этой операции все элементы правее добавляемого/удаляемого сдвигаются. **LinkedList** удобен когда важнее быстродействие операций вставки/удаления, которые в LinkedList выполняются за константное время. Операции доступа по индексу производятся перебором с начала или конца до нужного элемента. Дополнительные затраты на хранение связки между элементами.

**HashSet:** если нужно поддерживать порядок вставки и хранить уникальные объекты, то следует использовать HashSet. **LinkedHashSet:** если нужно сохранить порядок вставки элементов, то следует использовать LinkedHashSet. **TreeSet:** если нужно отсортировать элементы в соответствии с некоторым компаратором(или натуральной сортировкой), необходимо использовать TreeSet .

**HashMap** не дает абсолютно никаких гарантий относительно порядка элементов. Использовать, когда порядок элементов не имеет значение.

**TreeMap** элементы будут расположены в соответствии с натуральной сортировкой(compareTo() метод) или в соответствии с заданным компаратором. Следует использовать для уникальных пар ключ-значение, где необходима сортировка элементов.

**LinkedHashMap** тот же самый HashMap, но элементы сверху еще объединены в список в порядке вставки элементов в коллекцию.