**ДОМАШНЕЕ ЗАДАНИЕ** по курсу "JAVA"

Выполнил: Губенко Антон

ЗАДАНИЕ:

Необходимо написать собственную реализацию HashMap. Обязательные методы: get, put, remove.

Реализация HashMap.

Листинг класс MyHashMapImp.

package org.example;  
  
import java.util.\*;  
  
public class MyHashMapImp<K,V> implements Map<K,V> {  
 private static final int *DEFAULT\_CAPACITY*=16;  
 private int capacity;  
 private double loadFactor=0.8;  
 private int size;  
 private Node [] table;  
  
 public MyHashMapImp(){  
 table=new Node[*DEFAULT\_CAPACITY*];  
 size=0;  
 capacity=*DEFAULT\_CAPACITY*;  
 }

private static class Node<K,V>{  
 int hash;  
 K key;  
 V value;  
 Node next;  
  
 Node(int hash,K key,V value, Node next){  
 this.hash=hash;  
 this.key=key;  
 this.value=value;  
 this.next=next;  
 }  
 }  
  
 @Override  
 public int size() {  
 return size;  
 }  
  
 @Override  
 public boolean isEmpty() {  
 return (size==0);  
 }

@Override  
 public boolean containsKey(Object key) {  
 if(size==0){  
 return false;  
 }  
 return checkKey(key);  
 }  
  
 @Override  
 public boolean containsValue(Object value) {  
 boolean result=false;  
 if(size==0){  
 return false;  
 }  
 for(int i=0;i<capacity;i++){  
 if(table[i]!=null){  
 result=checkValue(table[i],value);  
 }  
 if(result){break;}  
 }  
 return result;  
 }

@Override  
 public V get(Object key) {  
 if(size==0){  
 return null;  
 }  
 int hash=getHashCode(key);  
 int index=getIndex(hash);  
 V resVal=null;  
 if(table[index]!=null){  
 resVal=getNode(index,hash,key);  
 }  
 return resVal;  
 }

@Override  
 public V put(K key, V value) {  
 int hash=getHashCode(key);  
 int index=getIndex(hash);  
 V resVal=null;  
  
 if(table[index]==null){  
 Node <K,V>newNode=new Node(hash,key,value,null);  
 table[index]=newNode;  
 size++;  
 }  
 else {  
 resVal=addNode(index,hash,key,value);  
 }  
 resize();  
 return resVal;  
 }

@Override  
 public V remove(Object key) {  
 if(size==0){  
 return null;  
 }  
 int hash=getHashCode(key);  
 int index=getIndex(hash);  
 V resVal=null;  
 if(table[index]!=null){  
 resVal=removeNode(index,hash,key);  
 }  
 resize();  
 return resVal;  
 }  
  
 @Override  
 public void putAll(Map<? extends K, ? extends V> m) {  
 Set<? extends K>keys=m.keySet();  
 for(K key:keys){  
 V value=m.get(key);  
 this.put(key,value);  
 }  
 }

@Override  
 public void clear() {  
 for(int i=0;i<capacity;i++){  
 table[i]=null;  
 }  
 size=0;  
 }  
  
 @Override  
 public Set<K> keySet() {  
 if(size==0){  
 return Collections.*emptySet*();  
 }  
 Set<K>keys=new HashSet<>();  
 for(int i=0;i<capacity;i++){  
 if(table[i]!=null){  
 addKeys(table[i],keys);  
 }  
 }  
 return keys;  
 }

@Override  
 public Collection<V> values() {  
 if(size==0){  
 return Collections.*emptySet*();  
 }  
 Set<V>values=new HashSet<>();  
 Set<K>keys=keySet();  
 for(K key:keys){  
 values.add(get(key));  
 }  
 return values;  
 }  
  
 @Override  
 public Set<Entry<K, V>> entrySet() {  
 if(size==0){  
 return Collections.*emptySet*();  
 }  
 Set<Entry<K,V>>entries=new HashSet<>();  
 Set<K>keys=keySet();  
 for(K key:keys){  
 V value=get(key);  
 Entry<K,V>entry=new AbstractMap.SimpleEntry(key,value);  
 entries.add(entry);  
 }  
 return entries;  
 }

//дополнительные утильные методы для работы мапы  
  
 private <K> int getHashCode(K key){  
 return Objects.*hashCode*(key);  
 }  
  
 private int getIndex(int hash){  
 return hash & (capacity-1);  
 }  
  
 private V addNode(int index,int hash,K key,V value){  
 V resVal=null;  
 Node<K, V> currentNode = table[index];  
 Node<K, V> lastNode = null;  
  
 while (currentNode != null) {  
 if (currentNode.hash == hash &&  
 (currentNode.key == key || (key != null && key.equals(currentNode.key)))) {  
 resVal = currentNode.value;  
 currentNode.value = value;  
 break;  
 }  
 lastNode = currentNode;  
 currentNode = currentNode.next;  
 }  
 if(resVal == null && lastNode != null && lastNode.next == null) {  
 lastNode.next = new Node<>(hash, key, value, null);  
 size++;  
 }  
 return resVal;  
 }  
  
 private V getNode(int index,int hash,Object key) {  
 V resVal=null;  
 Node<K, V> currentNode = table[index];  
 while(currentNode!=null){  
 if (currentNode.hash == hash &&  
 (currentNode.key == key || (key != null && key.equals(currentNode.key)))){  
 resVal=currentNode.value;  
 break;  
 }  
 currentNode = currentNode.next;  
 }  
 return resVal;  
 }  
  
 private V removeNode(int index,int hash,Object key) {  
 V resVal = null;  
 Node<K, V> currentNode = table[index];  
 Node<K, V> lastNode = null;  
  
 while (currentNode != null) {  
 if (currentNode.hash == hash &&  
 (currentNode.key == key || (key != null && key.equals(currentNode.key)))) {  
 resVal = currentNode.value;  
 size--;  
 break;  
 }  
 lastNode = currentNode;  
 currentNode = currentNode.next;  
 }  
 if(resVal!=null && lastNode!=null && currentNode!=null){  
 lastNode.next=currentNode.next;  
 }  
 else if(resVal!=null && lastNode==null){  
 table[index]=currentNode.next;  
 }  
 return resVal;  
 }  
  
 private boolean checkKey(Object key) {  
 int hash=getHashCode(key);  
 int index=getIndex(hash);  
 boolean result=false;  
 if(table[index]!=null) {  
 Node<K, V> currentNode = table[index];  
 while (currentNode != null) {  
 if (currentNode.hash == hash &&  
 (currentNode.key == key || (key != null && key.equals(currentNode.key)))) {  
 result=true;  
 break;  
 }  
 currentNode = currentNode.next;  
 }  
 }  
 return result;  
 }

private boolean checkValue(Node<K,V>node,Object value){  
 boolean result=false;  
 while (node != null) {  
 if (value == null ? node.value == null : value.equals(node.value)) {  
 result=true;  
 break;  
 }  
 node = node.next;  
 }  
 return result;  
 }  
  
 void addKeys(Node<K,V>node,Set<K>set){  
 while(node!=null) {  
 set.add(node.key);  
 node = node.next;  
 }  
 }  
  
 private void resize(){  
 if(size>=capacity\*loadFactor){  
 int oldCapacity=capacity;  
 int newCapacity=(int)(capacity\*1.5);  
 Node[]oldTable=table;  
 table=new Node[newCapacity];  
 capacity=newCapacity;  
 addAllNodes(oldTable,oldCapacity);  
 }  
 else if(size<((capacity/1.5)\*loadFactor-1) && capacity>16){  
 int oldCapacity=capacity;  
 int newCapacity=(int)(capacity/1.5);  
 Node[]oldTable=table;  
 table=new Node[newCapacity];  
 capacity=newCapacity;  
 addAllNodes(oldTable,oldCapacity);  
 }  
 }  
  
 private void addAllNodes(Node[]oldTable,int oldCapacity){  
 for(int i=0;i<oldCapacity;i++){  
 if(oldTable[i]!=null){  
 Node<K,V>currentNode=oldTable[i];  
 while(currentNode!=null) {  
 this.putResize(currentNode.key, currentNode.value);  
 currentNode=currentNode.next;  
 }  
 }  
 }  
 }  
  
 public void putResize(K key, V value) {  
 int hash=getHashCode(key);  
 int index=getIndex(hash);  
  
 if(table[index]==null){  
 Node <K,V>newNode=new Node(hash,key,value,null);  
 table[index]=newNode;  
 }  
 else {  
 addNodeResize(index,hash,key,value);  
 }  
 }  
  
 private void addNodeResize(int index,int hash,K key,V value){  
 Node<K, V> currentNode = table[index];  
 Node<K, V> lastNode = null;  
 int flag=0;  
 while (currentNode != null) {  
 if (currentNode.hash == hash &&  
 (currentNode.key == key || (key != null && key.equals(currentNode.key)))) {  
 currentNode.value = value;  
 flag=1;  
 break;  
 }  
 lastNode = currentNode;  
 currentNode = currentNode.next;  
 }  
 if(flag==0 && lastNode != null && lastNode.next == null) {  
 lastNode.next = new Node<>(hash, key, value, null);  
 }  
 }  
  
 //утильный метод для проверки метода resize  
  
 public int getTableSize(){  
 return table.length;  
 }  
}

Реализованы методы:

public int size(),

public boolean isEmpty(),

public boolean containsKey(Object key),

public boolean containsValue(Object value),

public V get(Objet key),

public V put(K key,V value),

public V remove(Object key),

public void putAll(Map<? Extends K,? extends V>m),

public void clear(),

public Set<K>keySet(),

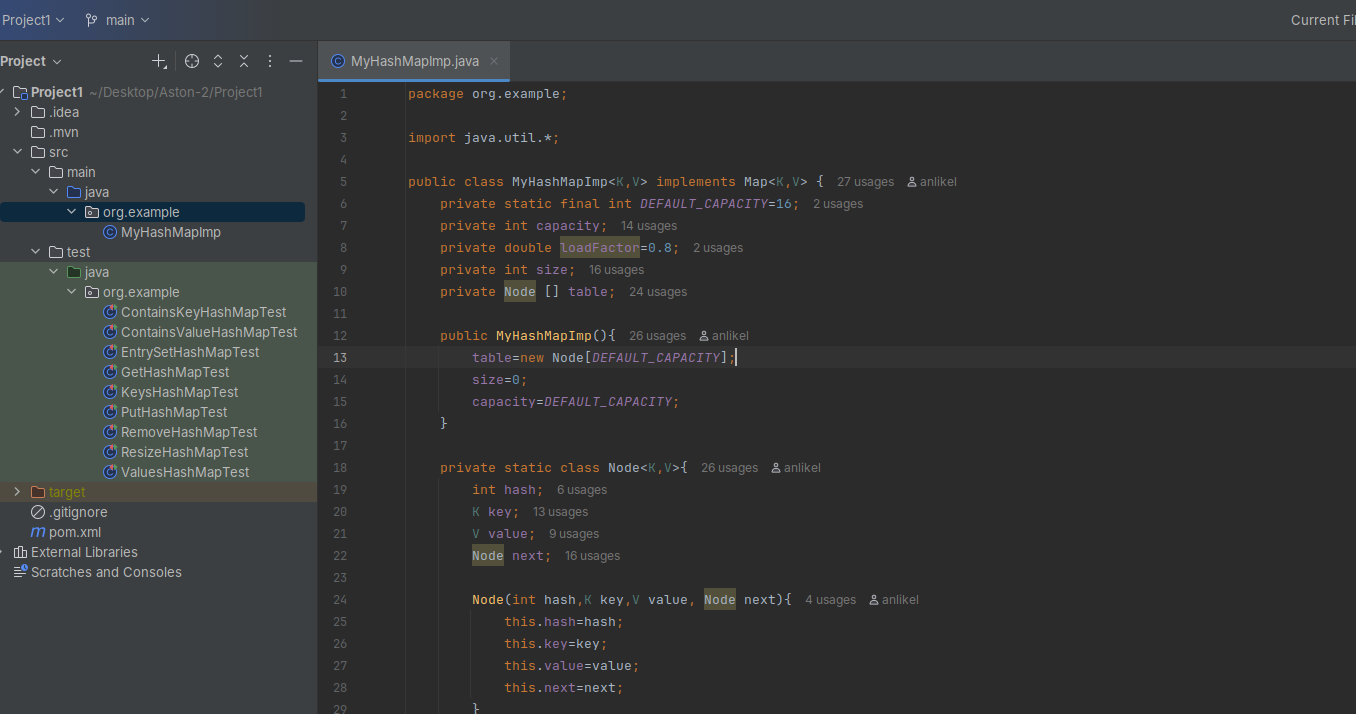
public Collection<V>values(),

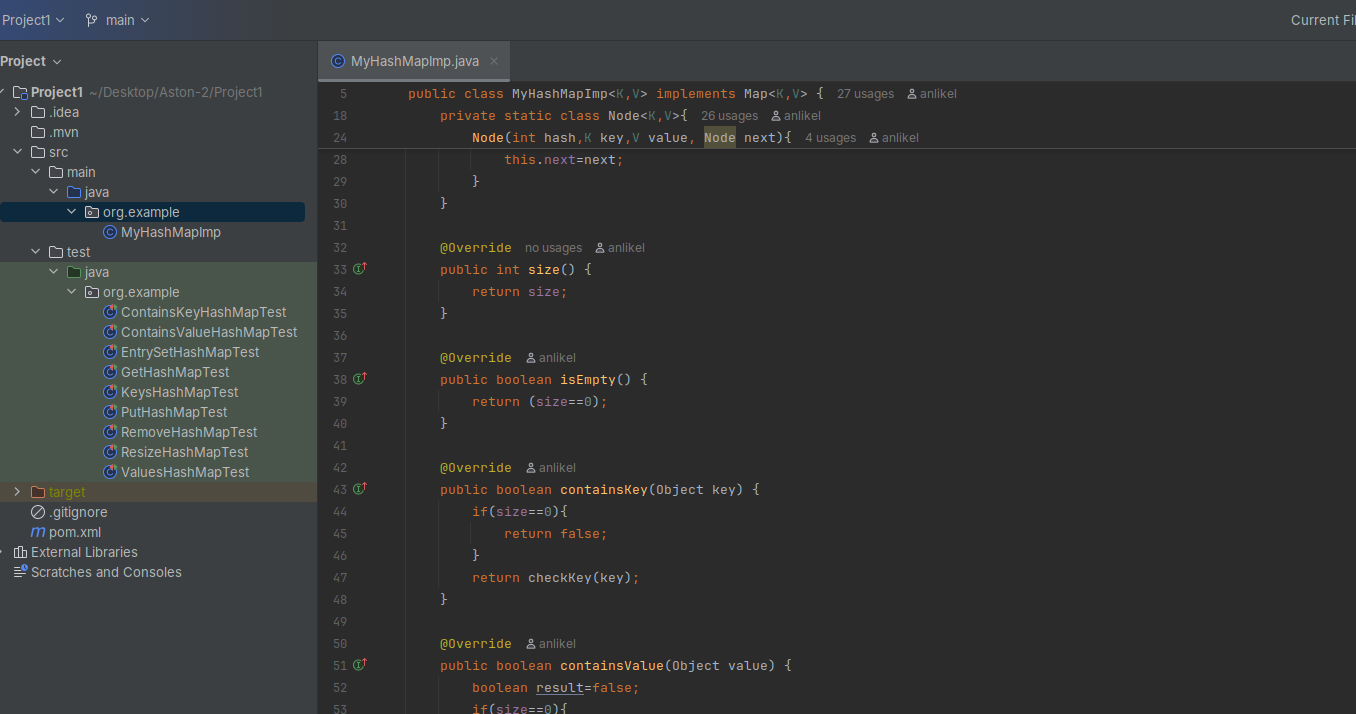
public Set<Entry<K,V>>entrySet().

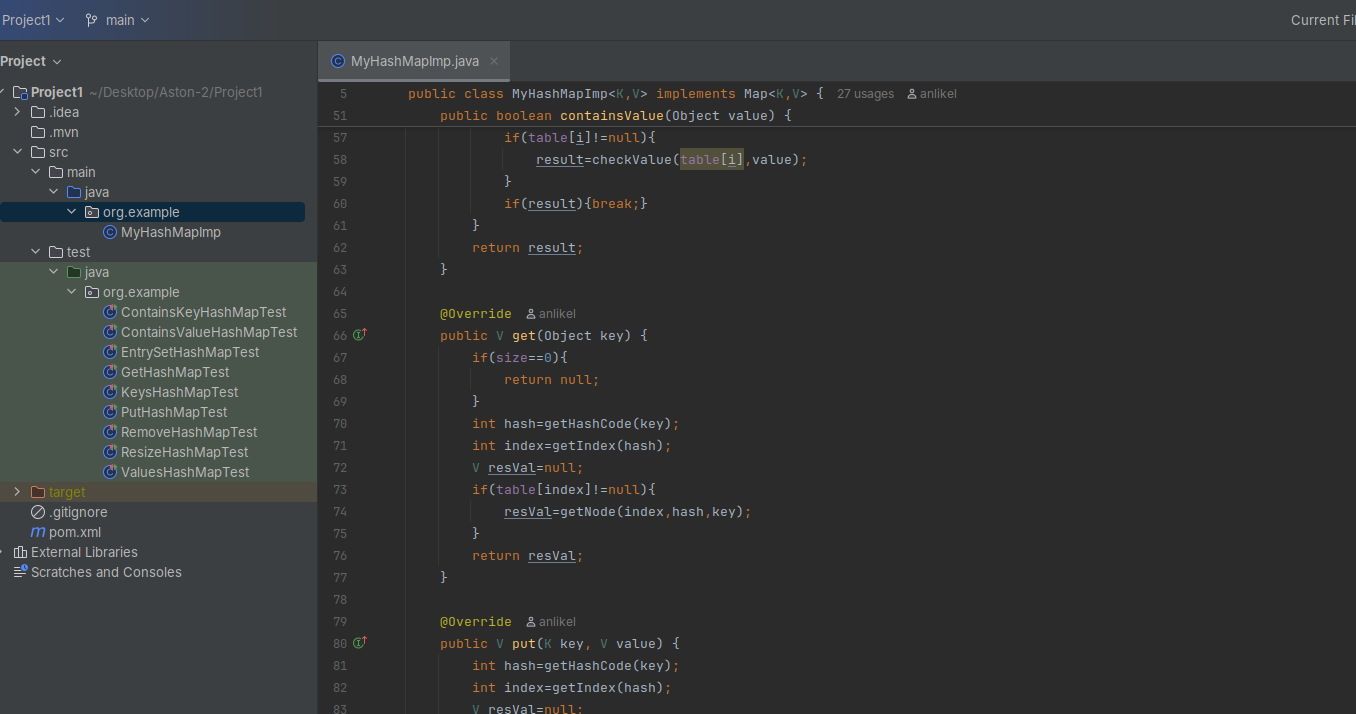
Так же был реализован внутренний статический класс

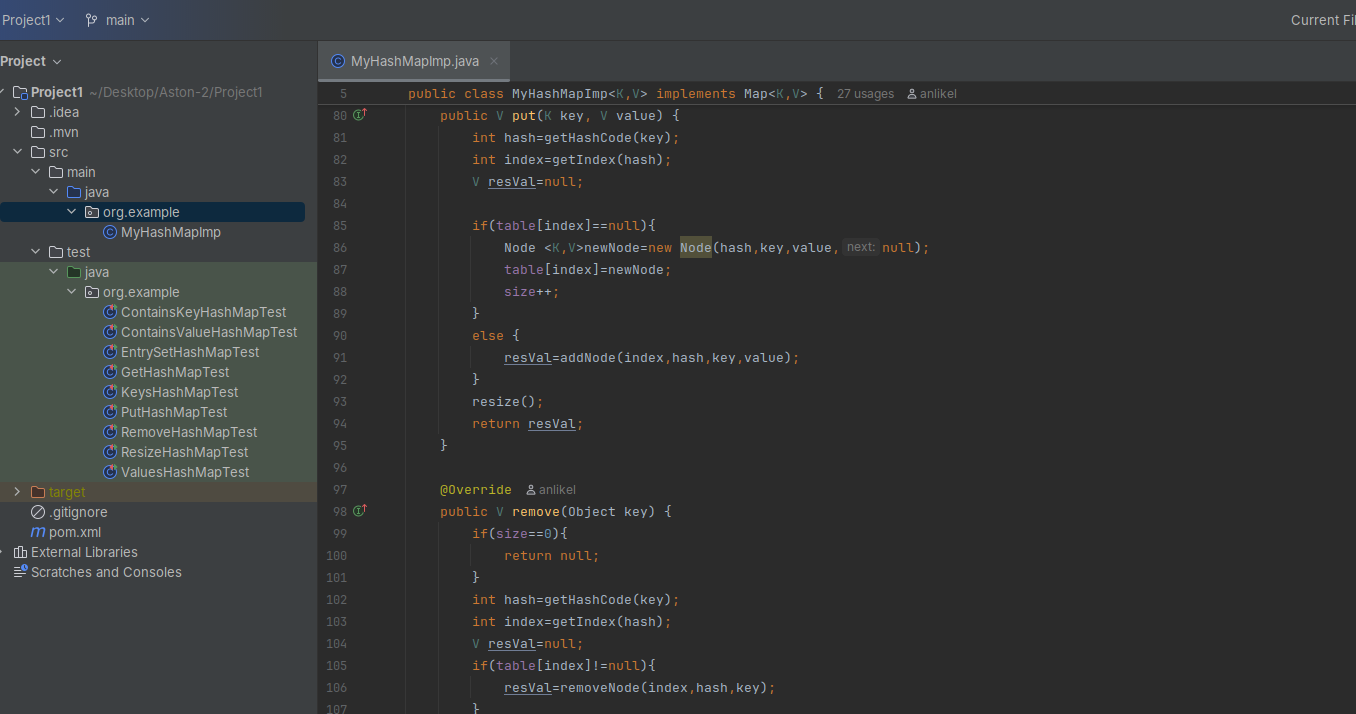
private static class Node<K,V> для хранения элементов коллекции.

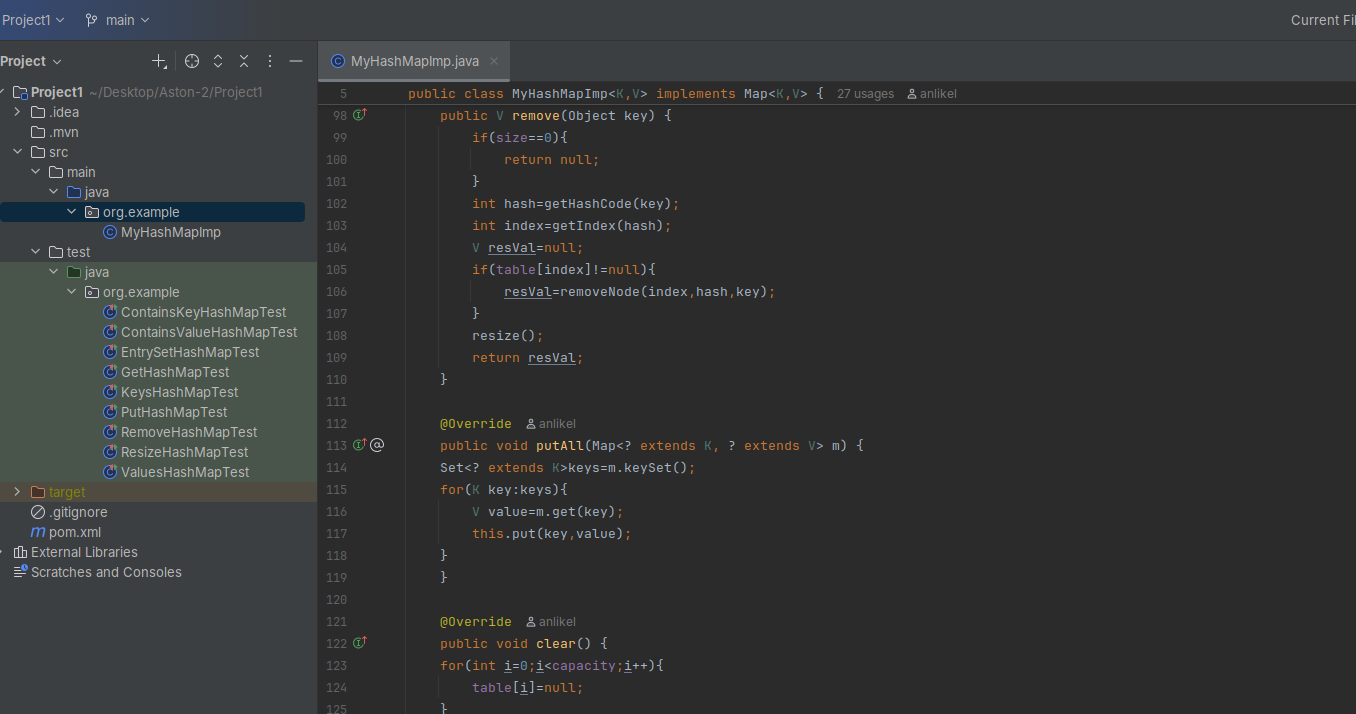
Скриншоты MyHashMapImp;

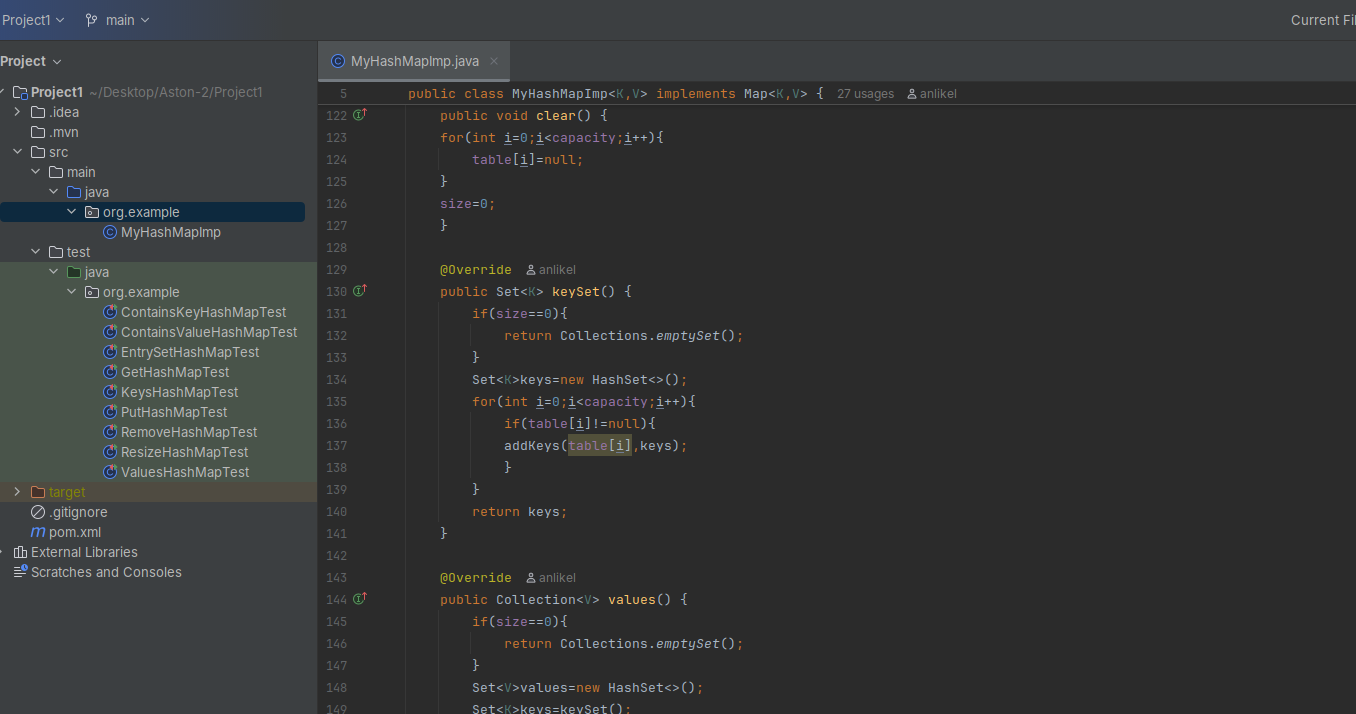


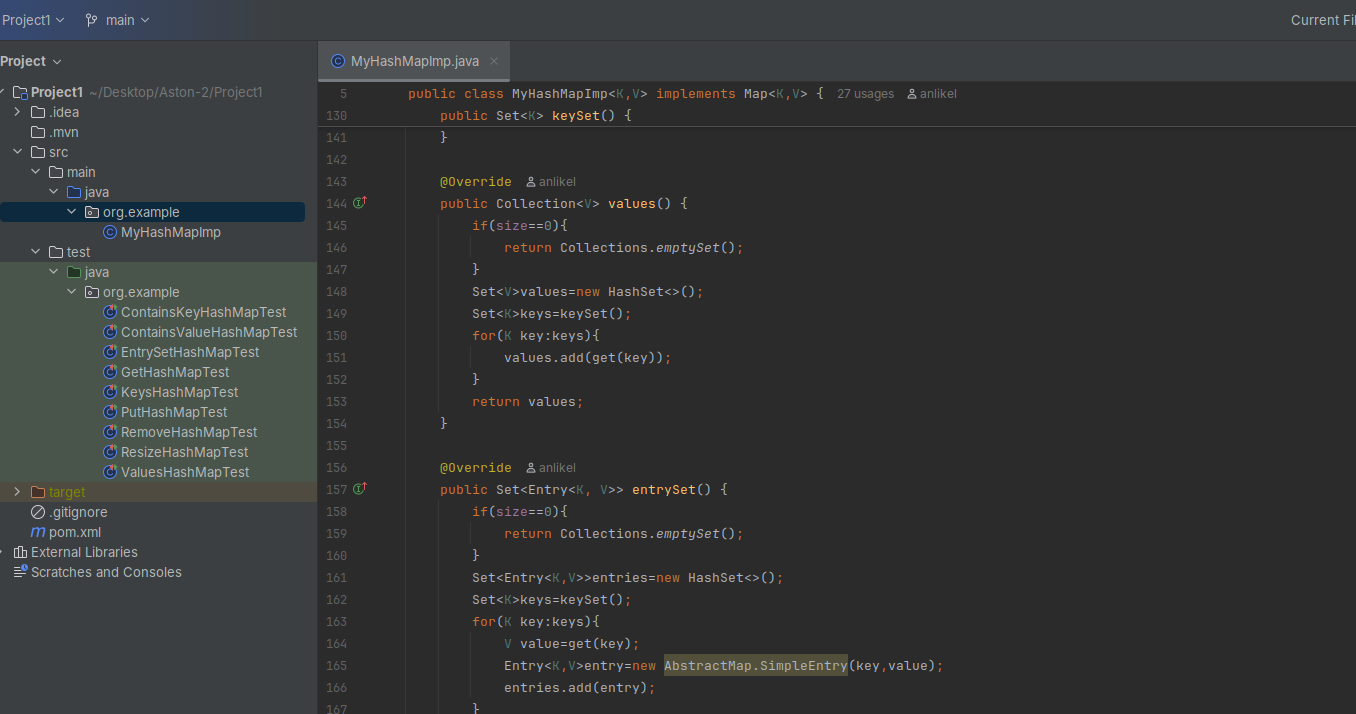


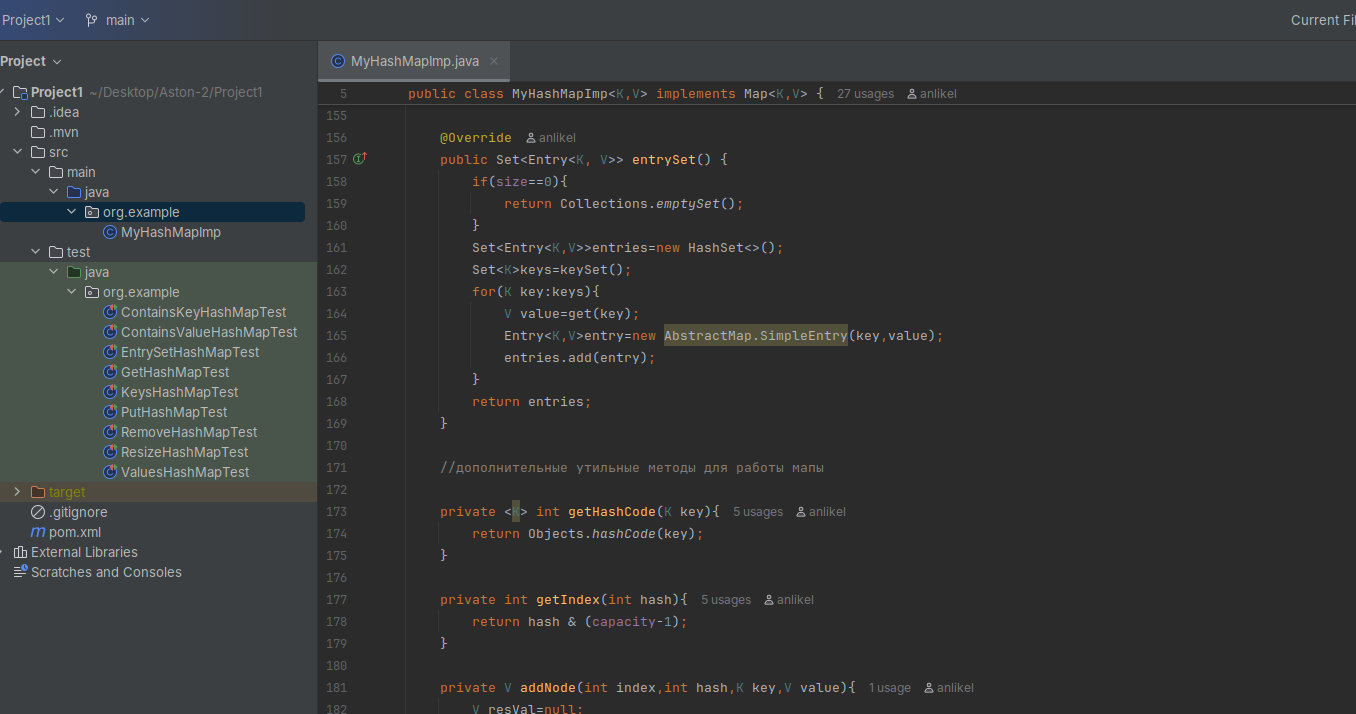


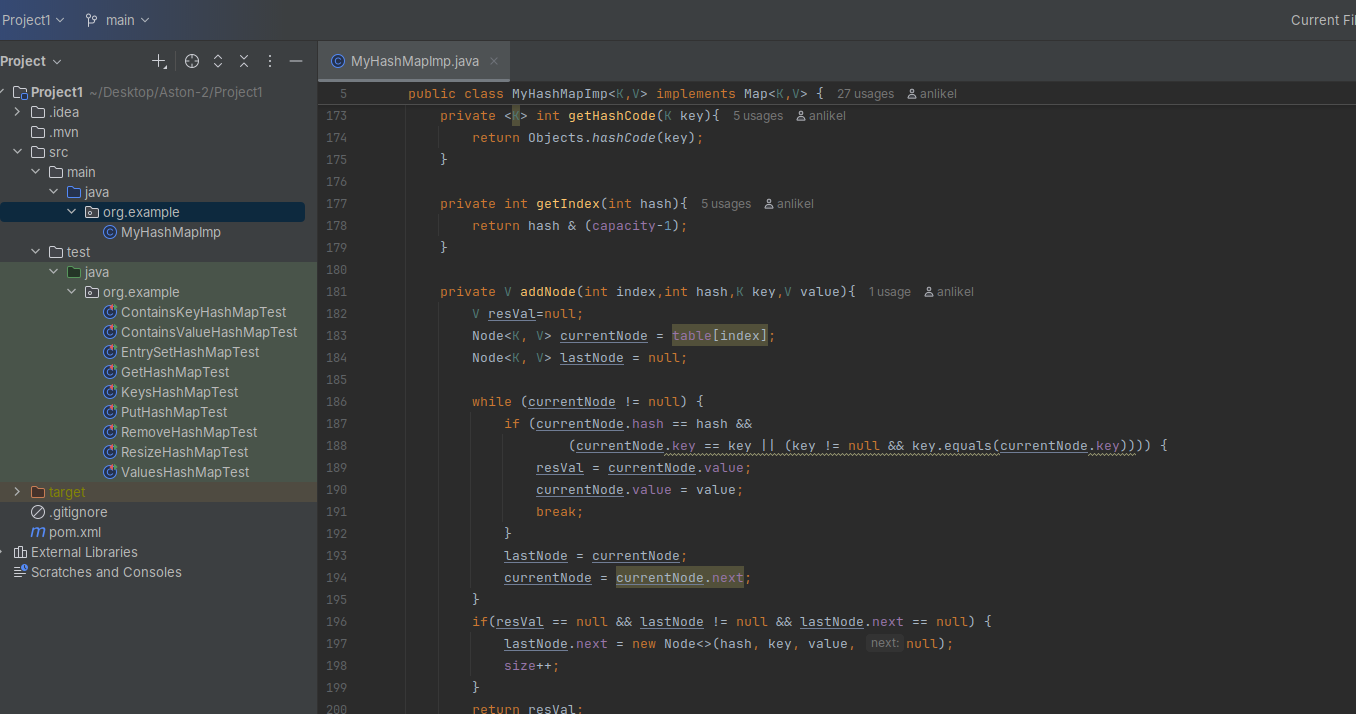


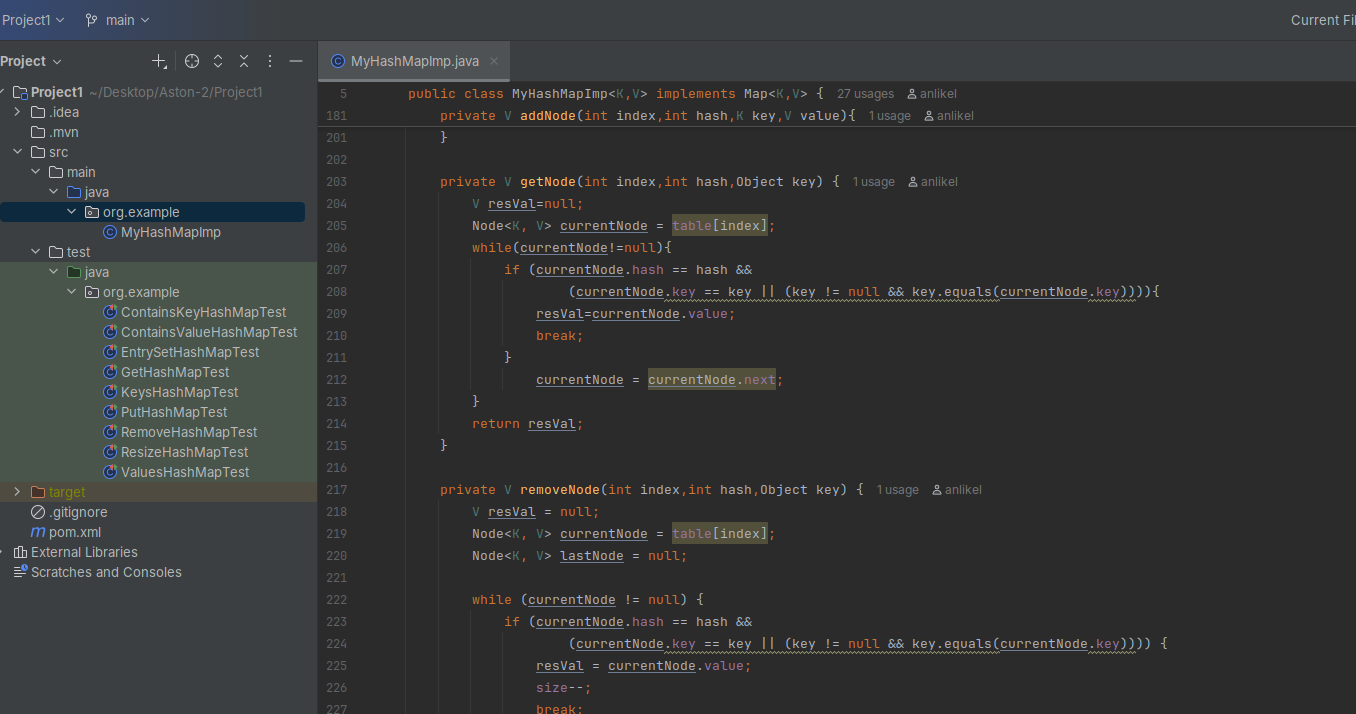


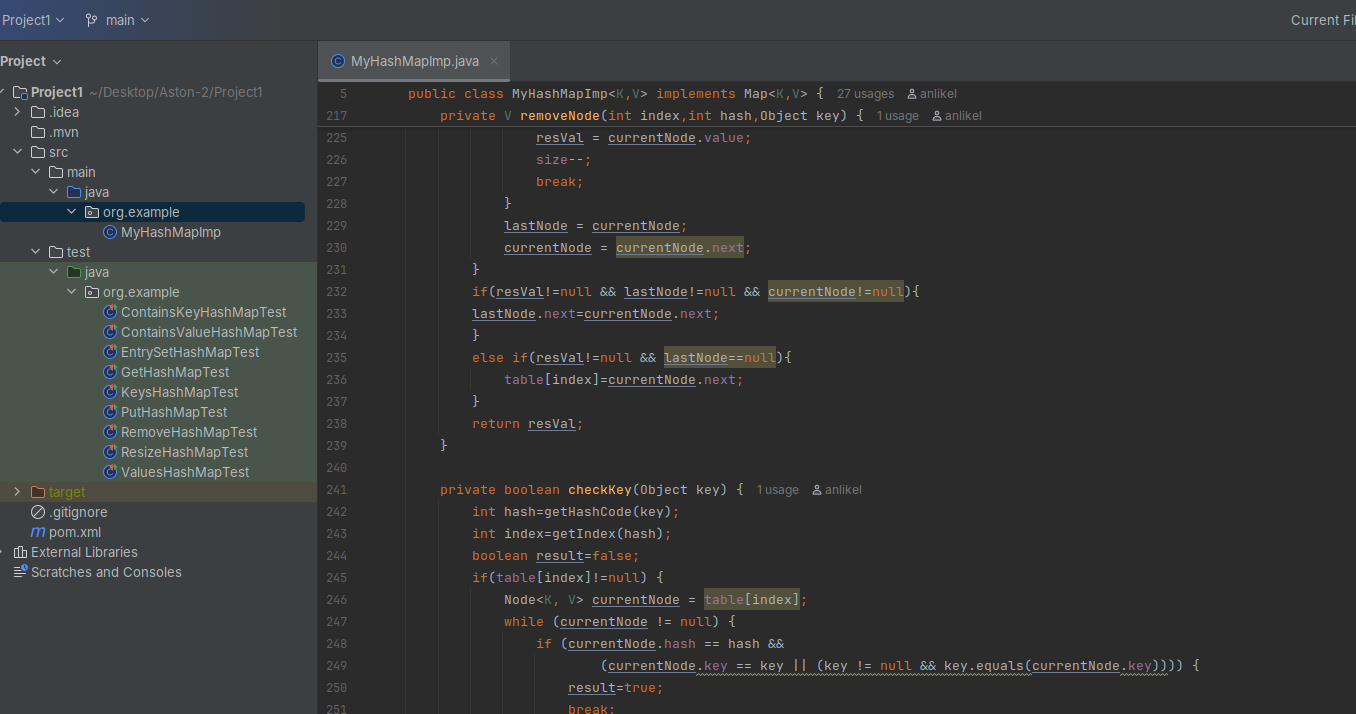


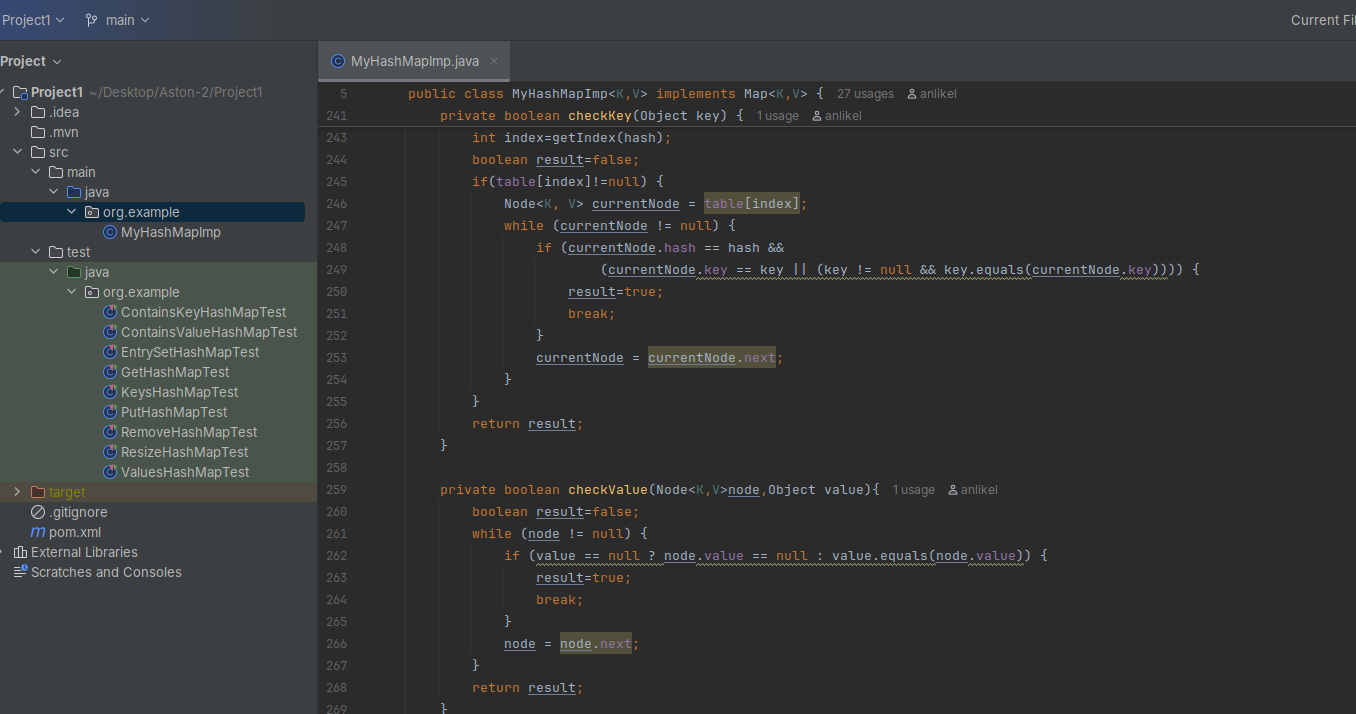


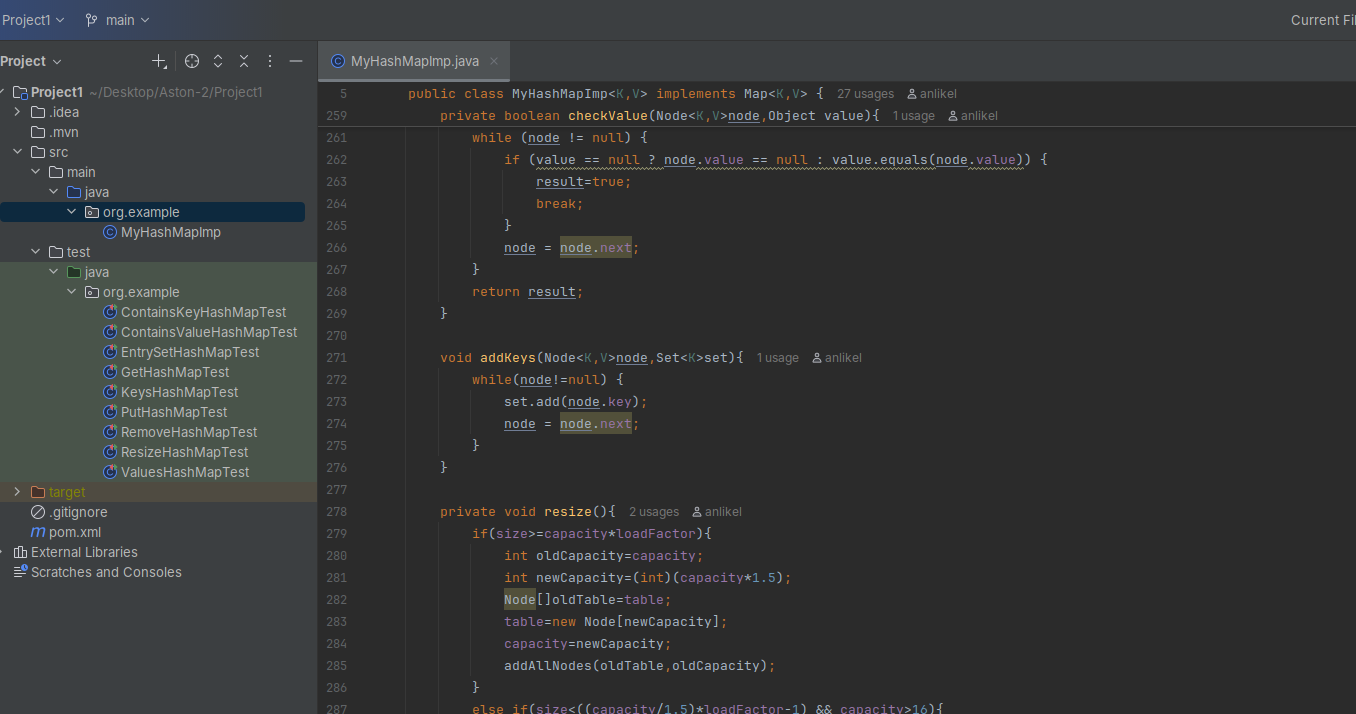


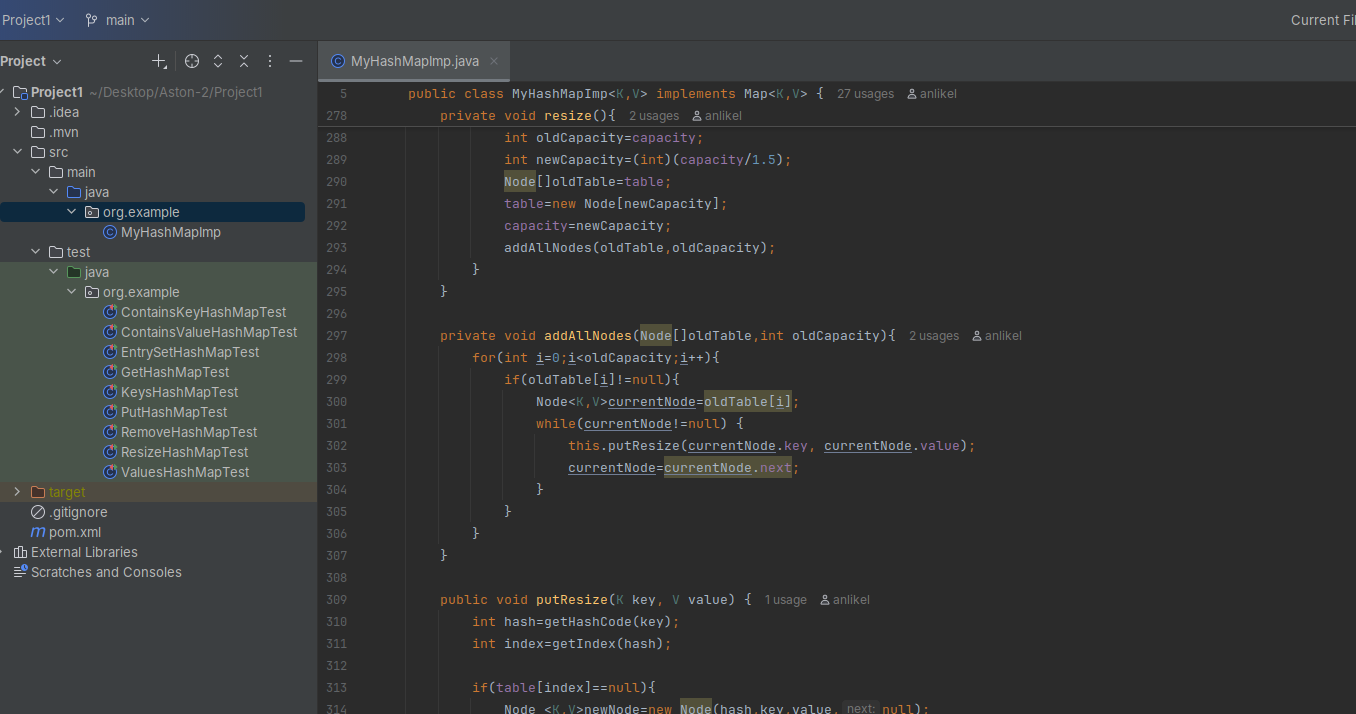


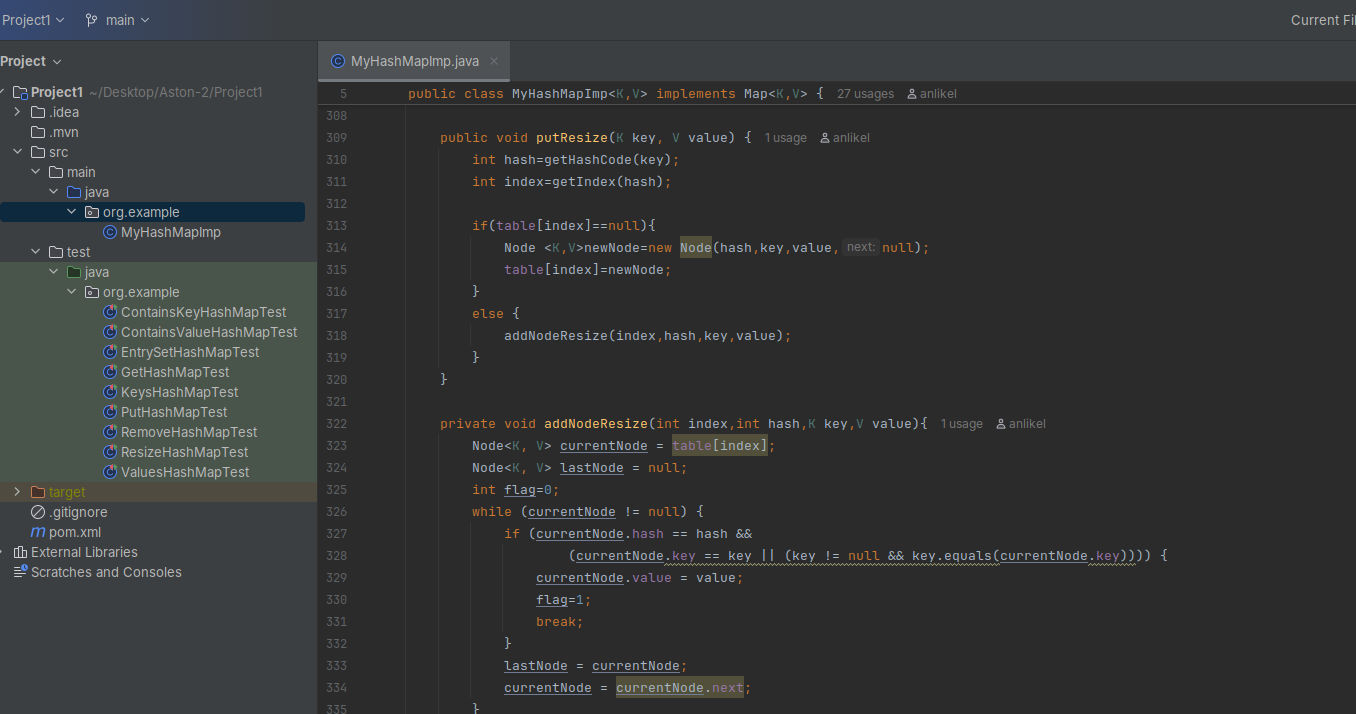


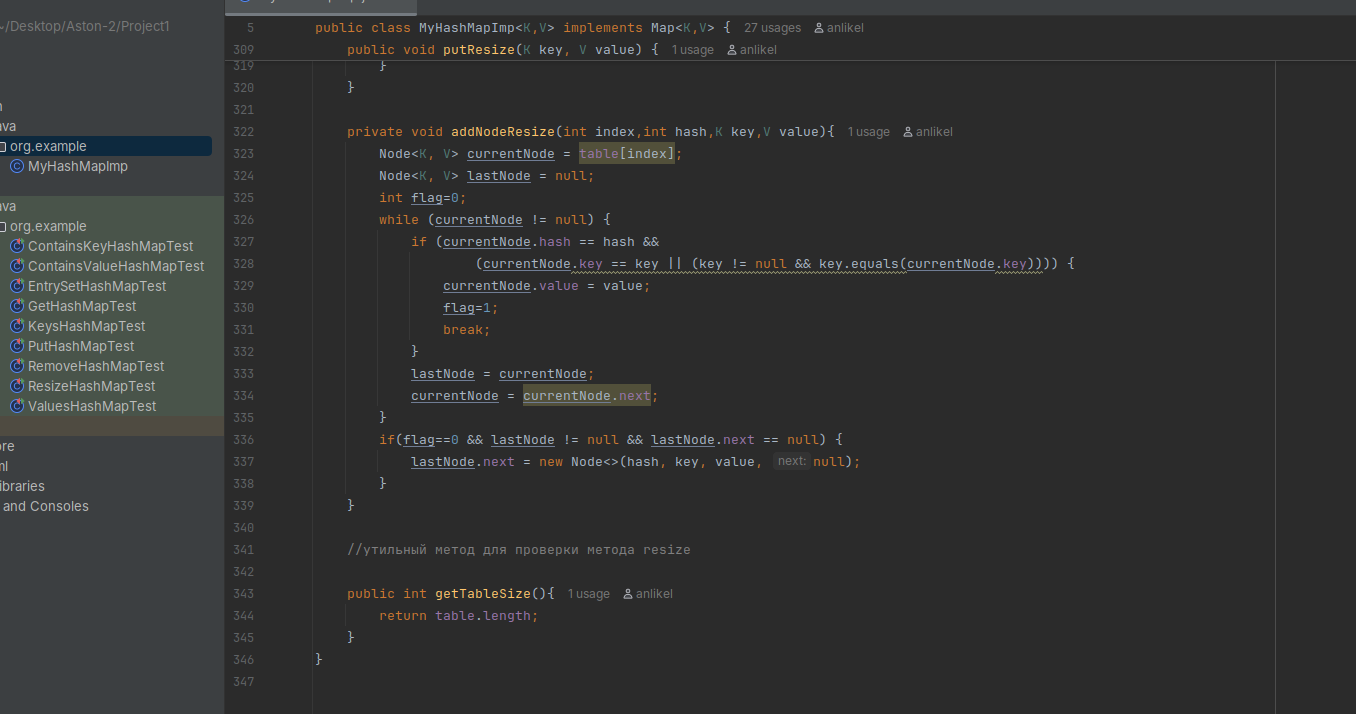




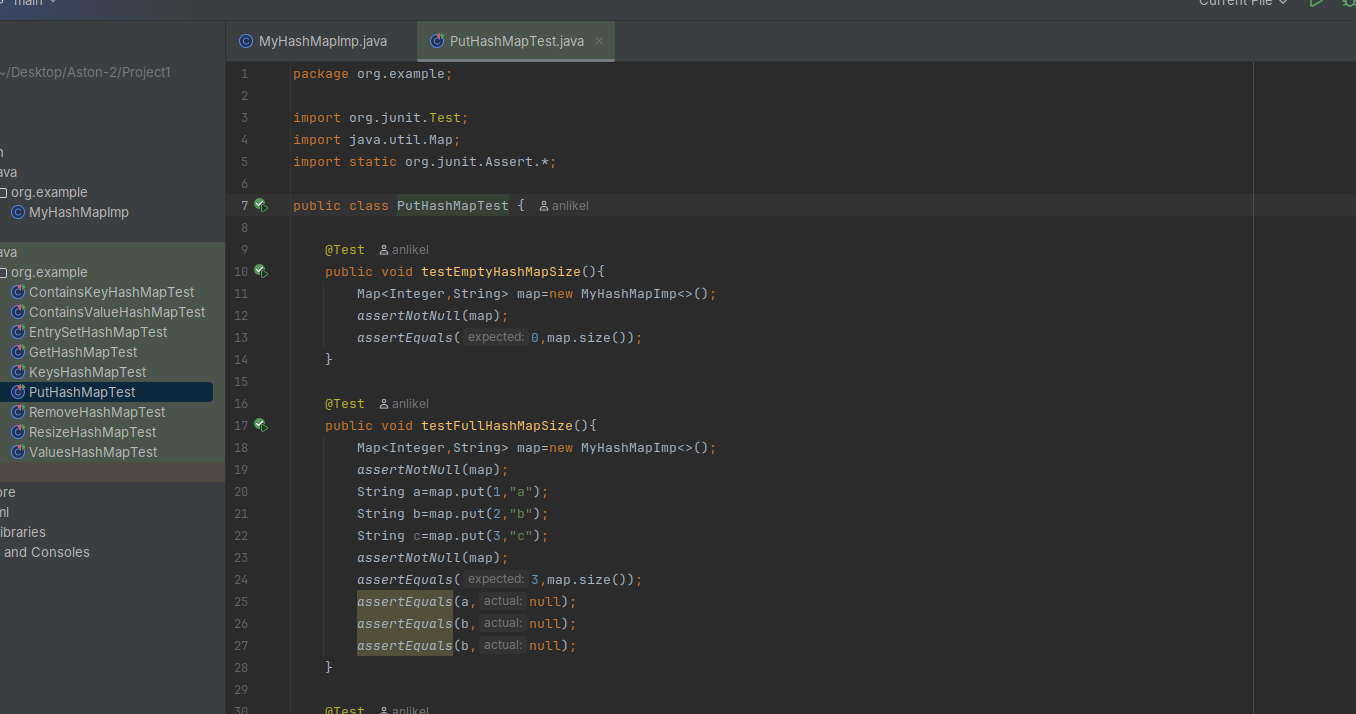


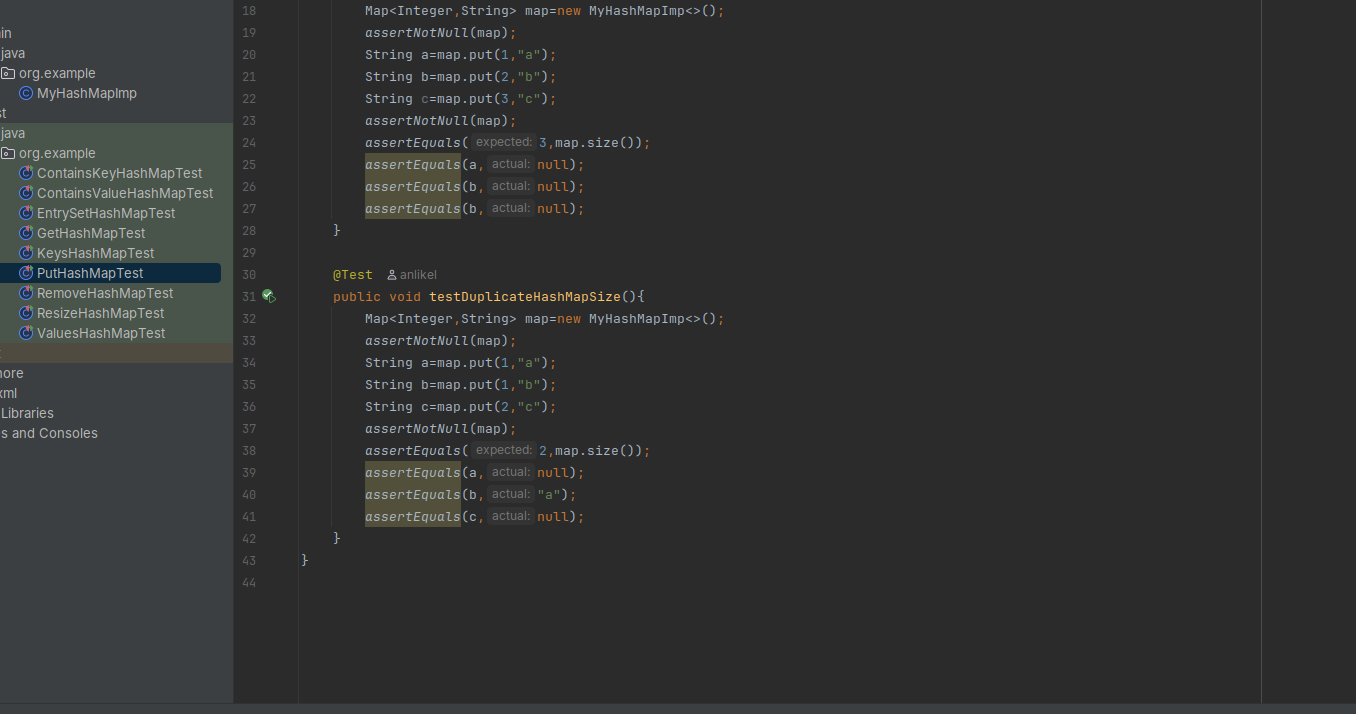


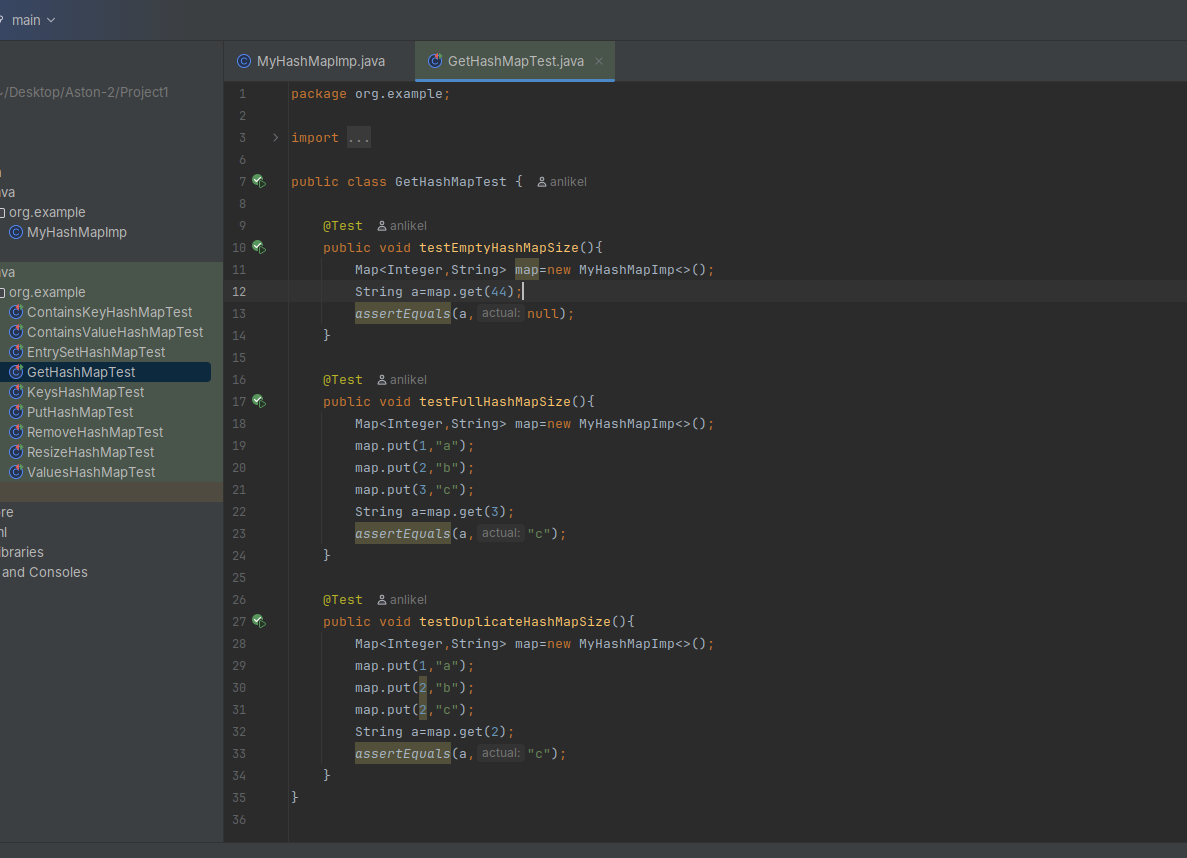


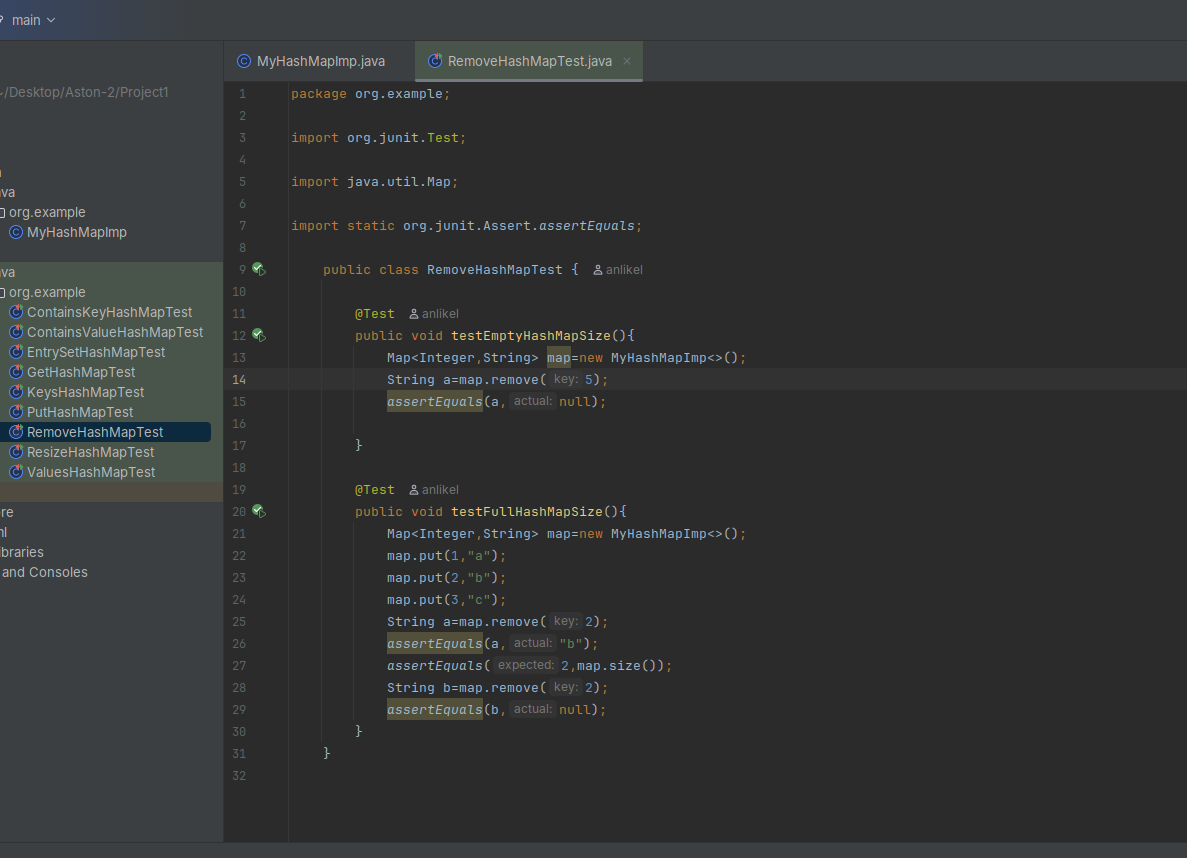


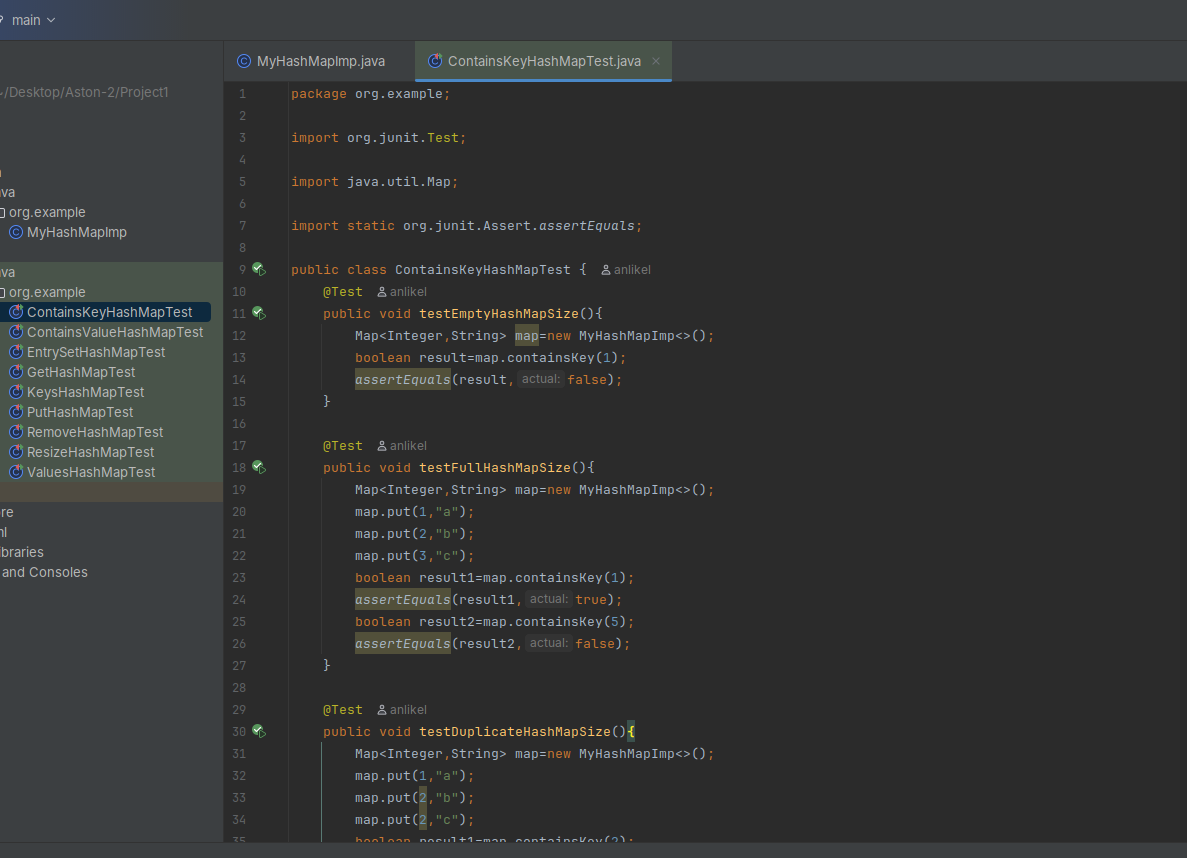
Скриншоты Тестов;

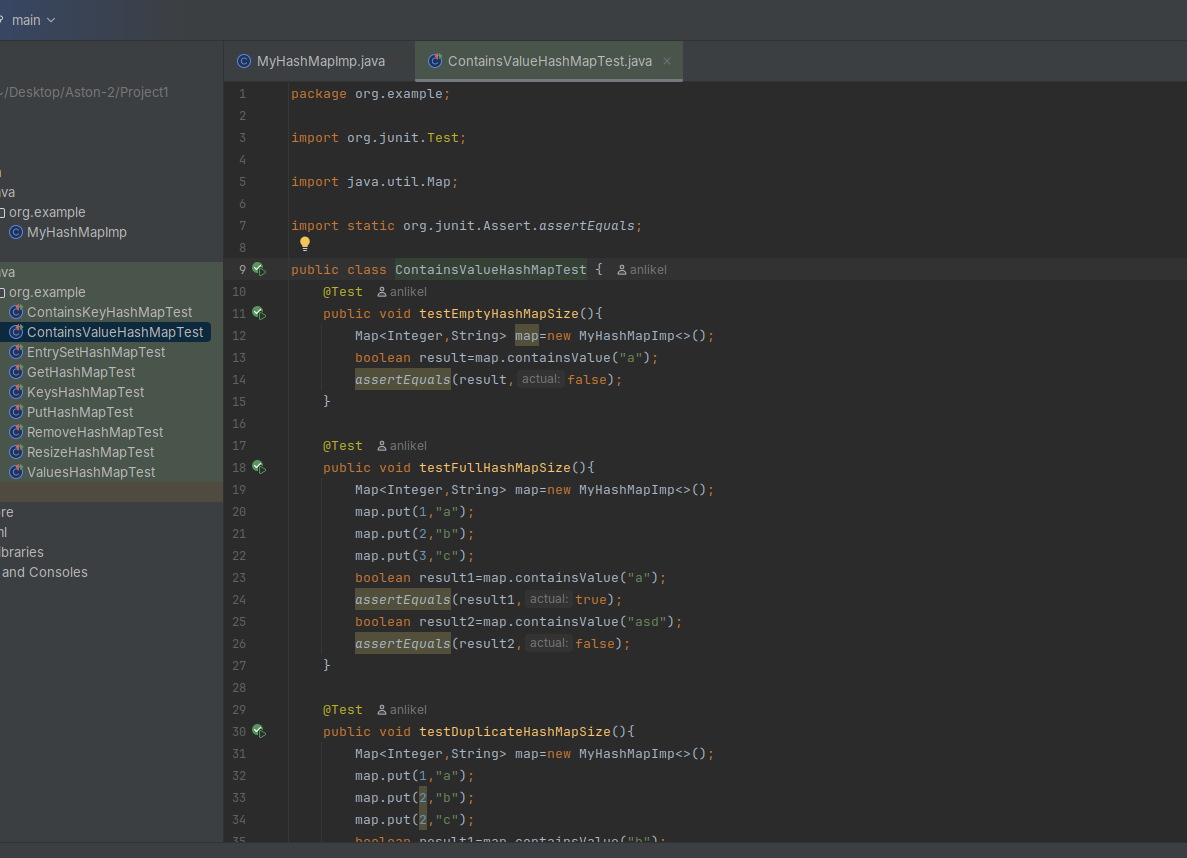


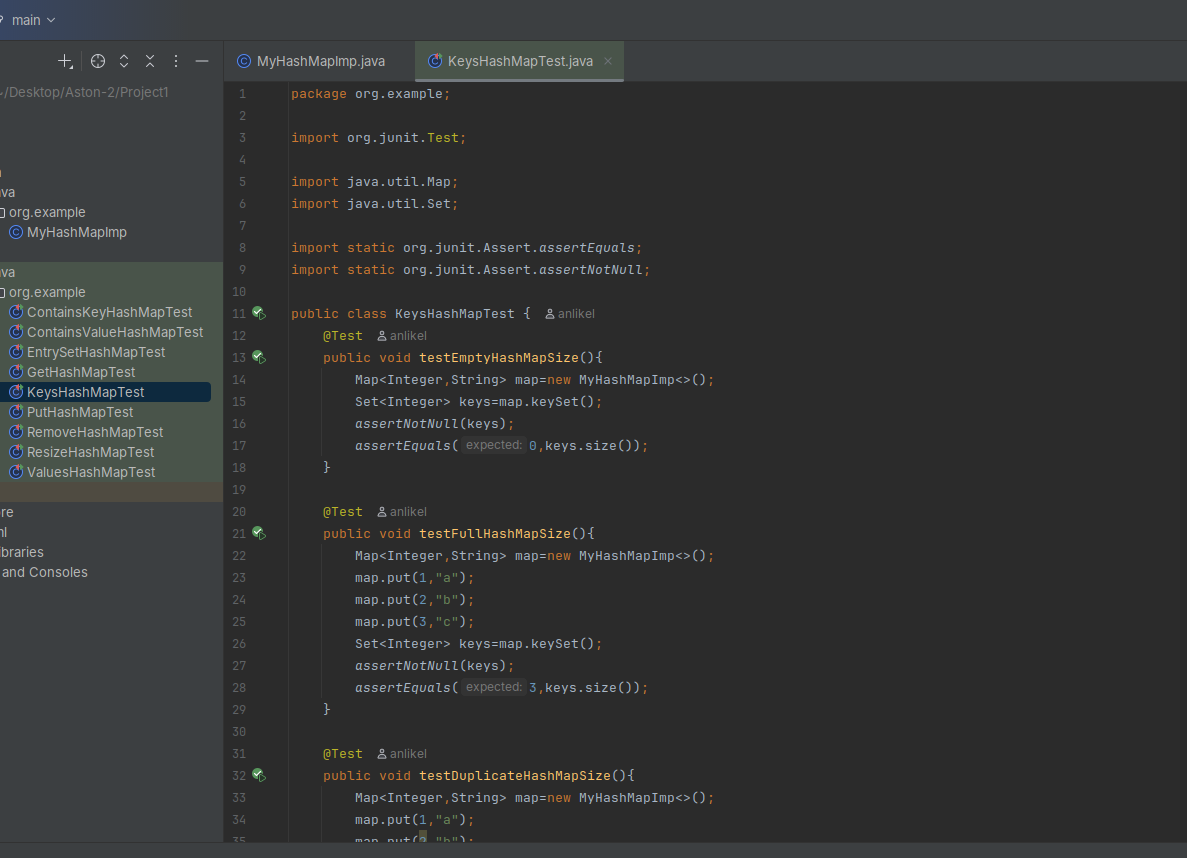
****

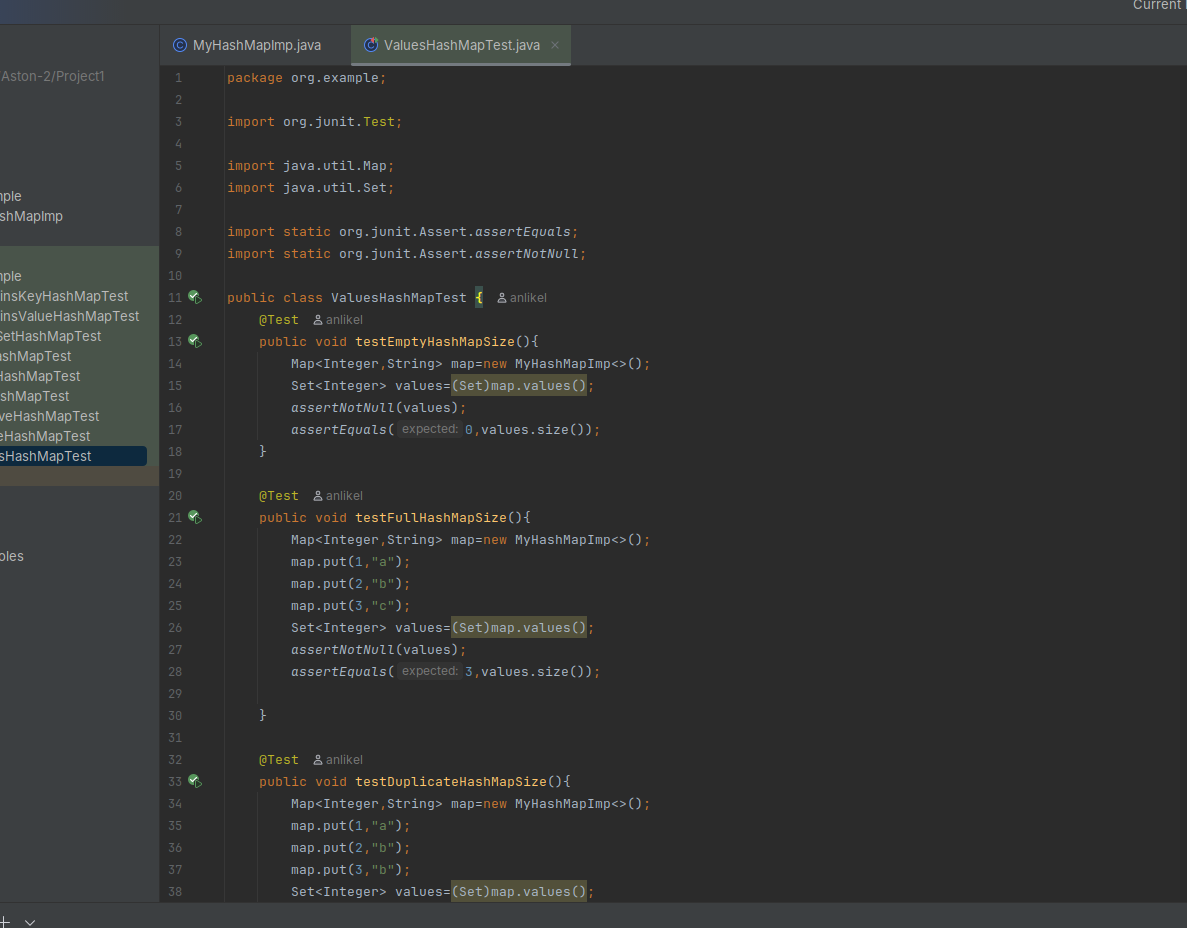


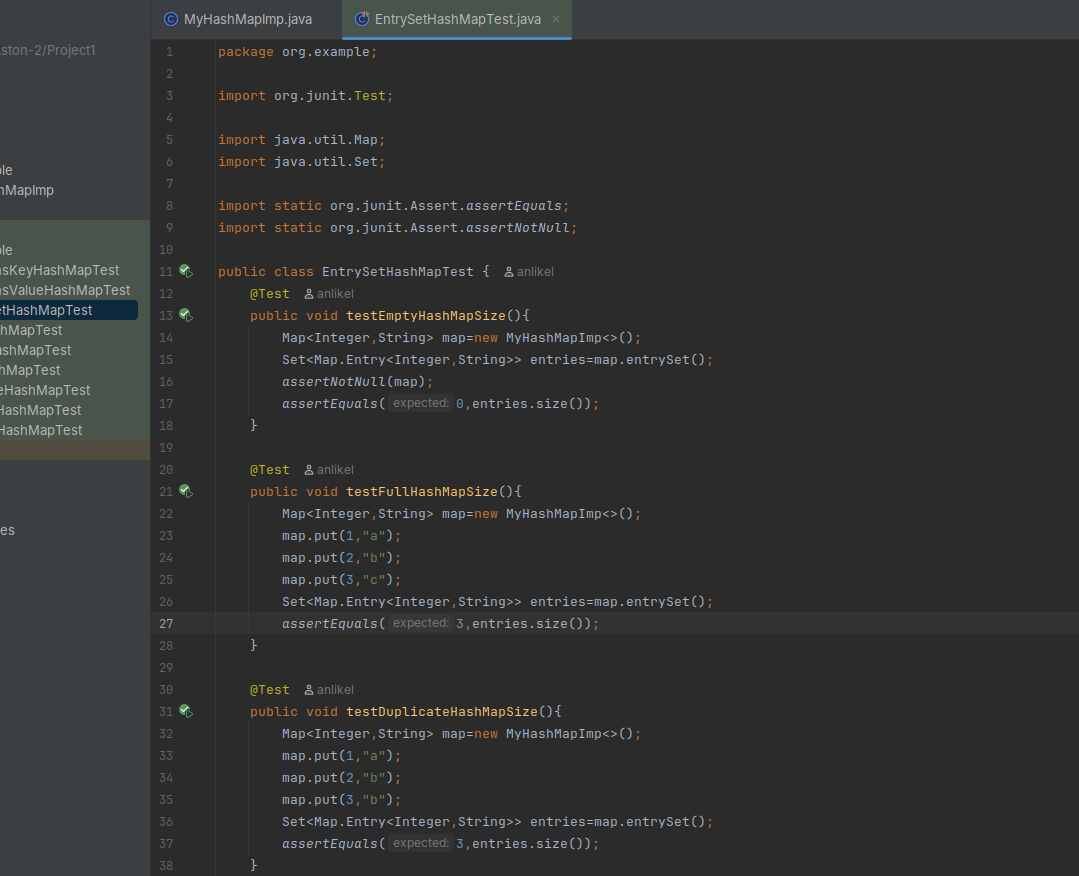


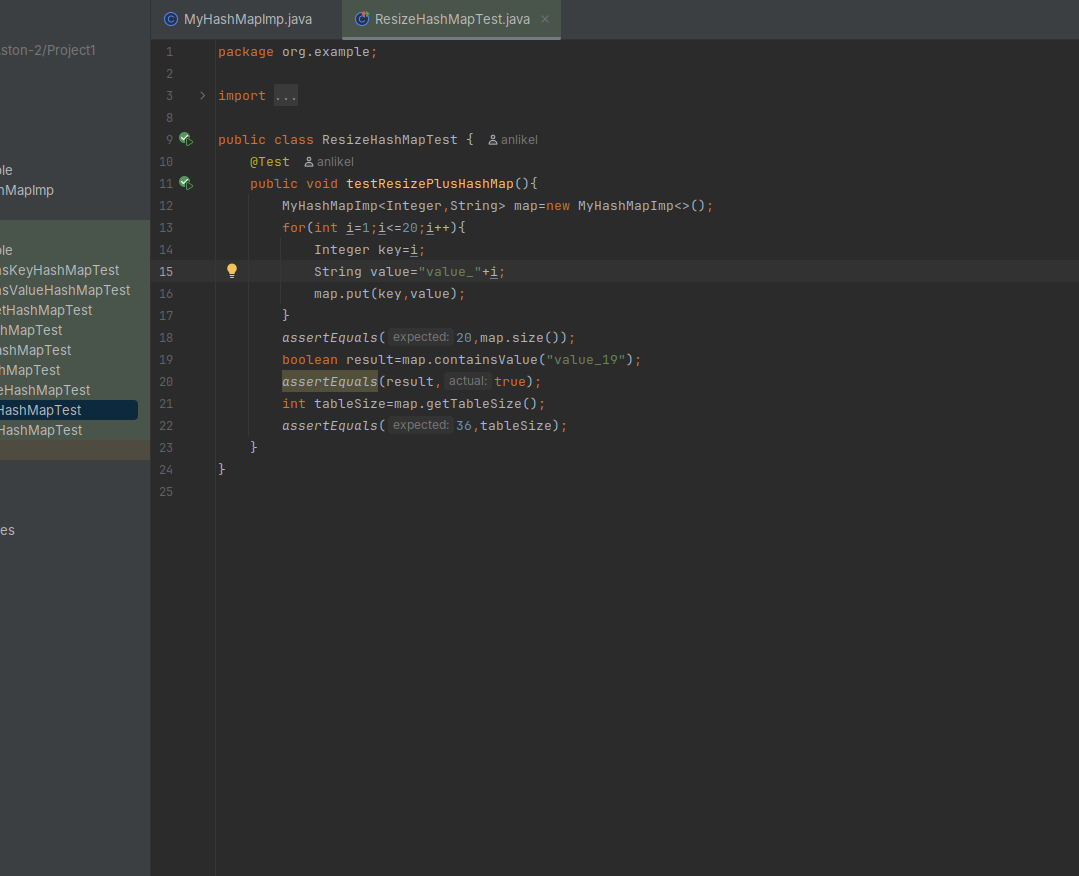












Результат выполнения тестов:

