```
import static java.lang.reflect.Array.newInstance;


@SuppressWarnings("unchecked")
public class insuranceComparator<K, V> {
    //Maximum inital map length
    private static final int MAX_MAP_LENGTH = 89999;
    //Intializes hashtable of key-value entries
    private Entry<K,V>[] hashTable;
    //current size of table
    private int size;
    //current capacity of table
    private int capacity;

    //base constructor with MAX_MAP_LENGTH
    public insuranceComparator() {
        capacity = MAX_MAP_LENGTH;
        //Insiantiates new array of Entry<K,V> objects with
MAX_MAP_LENGTH as capacity
        hashTable = (Entry<K,V>[]) newInstance(Entry.class,
MAX_MAP_LENGTH);
    }

    //Paramaterized constructor with capcity passed
  private insuranceComparator(int capacity){
        this.capacity = capacity;
        //Instantiates new array of Entry<K,V> objects with capacity
as capacity
        hashTable = (Entry<K,V>[]) newInstance(Entry.class, capacity);
    }

    //hash function for the prefacture number
    private int hash(K key){
        return Math.abs(key.hashCode()) % capacity;

    }

    //puts key-value pairs into the hashtable
    public void put(K key, V value){
        if(size >= capacity * 0.75){
            resize(2 * capacity);
        }
        //creates hashing index using hash funtion
        int index = hash(key);
        while(hashTable[index] != null){
            //If a value already exists with the key, replace the
value
            if(hashTable[index].getPreNum().equals(key)){
                hashTable[index].setFecha(value);
                return;
```

```
            }
            //increment index
            index = (index + 1) % capacity;
        }
        //assume no key was found, create new entry in table
        hashTable[index] = new Entry<>(key, value);
        size++;
    }
        //Resizes the map if the capacity is reached
    private void resize(int newCap){
        insuranceComparator<K,V> tempMap = new
insuranceComparator<>(newCap);
        for(int i = 0; i < capacity; i++){
            if(hashTable[i]!= null){
                //puts all elements from old hashtable into new
hashtable
                tempMap.put(hashTable[i].getPreNum(),
hashTable[i].getFecha());
            }
        }
        //creates new hashtable from the temp
        hashTable = tempMap.getHashTable();
        this.capacity = newCap;
    }

    //get method to fidn the value of specific keys
    public V get(K key){
        //calls hash function
        int index = hash(key);
        while(hashTable[index] != null){
            if(hashTable[index].getPreNum().equals(key)){
                return hashTable[index].getFecha();
            }
            index = (index + 1) % capacity;
        }
        //returns null if key not found in hashtable
        return null;
    }

    //generic remove method for hashtable
    public V remove(K key){
        int index = hash(key);

        while(!hashTable[index].getPreNum().equals(key)){
            index = (index + 1) % capacity;
        }
        V toReturn = hashTable[index].getFecha();
        hashTable[index] = null;
        size--;
```

```
        while(hashTable[index = (index + 1) % capacity] != null){
            Entry<K, V> nextItem = hashTable[index];
            hashTable[index] = null;
            size--;
            put(nextItem.getPreNum(), nextItem.getFecha());
        }
        return toReturn;

    }

    //returns the hashtable
    public Entry<K, V>[] getHashTable(){
        return hashTable;
    }

    //returns size
    public int getSize(){
        return size;
    }

    //Basic toString method for hashtables
    @Override
    public String toString(){
        if(size == 0){
            return "[]";
        }
        StringBuilder sb = new StringBuilder("[");
        for(Entry<K, V> entry : hashTable){
            if(entry != null){

sb.append(entry.getPreNum()).append("=").append(entry.getFecha()).appe
nd(", ");
            }
        }
        sb.setLength(sb.length() - 2);
        sb.append("]");
        return sb.toString();
    }

}
```