Bài 7.2: Bảng băm người dùng tự định nghĩa

- ✓ Sơ lược về các thao tác trên bit
- ✓ Tạo lớp Entry
- ✓ Tạo lớp HashTable
- ✓ Ví dụ minh họa
- ✓ Bài tập thực hành

Thao tác &|^~

- Các thao tác trên bit ta tìm hiểu gồm: &, |, ^, ~, <<, >>.
- Phép AND: thực hiện nhân 2 bit, cho kết quả là 1 khi cả hai bit tham gia cùng bằng 1.
- Phép OR: cho kết quả là 0 khi và chỉ khi cả hai bit tham gia đều bằng 0.
- Phép XOR: cho kết quả là 1 nếu hai bit tham gia khác nhau. Kết quả là 0 nếu hai bit giống nhau.

Operator	Bit 1	Bit 2	Bit Value (Result)
AND &	1	1	1
	1	0	0
	0	1	0
	0	0	0
OR |	1	1	1
	1	0	1
	0	1	1
	0	0	0
XOR ^	1	1	0
	1	0	1
	0	1	1
	0	0	0

Thao tác &|^~

- ~: NOT-lấy phần bù, bù của 0 là 1 và của 1 là 0. Đây là phép bù 2. NOT x = -x – 1.
- Khi ta thực hiện các phép toán trên với hai biến nguyên, chương trình tự chuyển đổi giá trị hệ 10 sang hệ 2 sau đó tiến hành thao tác được yêu cầu trên dãy nhị phân sau chuyển đổi.
- Ví dụ a & b với a = 5, b = 6 thì kết quả là 4.
  - 5 = 0101
  - 6 = 0110
  - a&b = 0100 => 4 ở hệ 10

Thao tác dịch bit

- << dịch trái dãy bit sang trái n bit, điền 0 vào các bit phải cùng. Dịch trái n bit tương đương nhân với 2^n.
- Ví dụ: xét số a 1 byte có biểu diễn 00010000 = 16. a << 2 cho biểu diễn là 01000000 = 64.
- >> dịch phải dãy bit sang phải n bit, điền bit dấu vào các bit trái cùng. Dịch phải n bit tương đương chia cho 2^n.
- Trong Java, các kiểu số đều có dấu. Số dương +a sẽ có bit dấu là 0, số âm có bit dấu là 1. Bit dấu luôn là bít trái cùng.
- Ví dụ: a = +16, a >> 2 cho kết quả là +4: 00000100.
- Ví dụ: a = 10010111 = -105. khi a >> 1 cho kết quả 11001011 = -53.

Tạo lớp Entry

- Tạo lớp inner static Entry<K,V> triển khai interface Entry của Map.
- Chứa các thuộc tính:
  - Mã băm: hash.
  - Key: key.
  - Value: value.
  - Con trỏ Entry<K, V>: next

```
private static class Entry<K, V> implements Map.Entry<K, V> {
    final int hash;     // mã băm
    final K key;        // khóa
    V value;            // giá trị liên kết với khóa
    Entry<K, V> next;   // con trỏ next

    protected Entry(int hash, K key, V value, Entry<K, V> next) {
        this.hash = hash;
        this.key = key;
        this.value = value;
        this.next = next;
    }
```

Tạo lớp Entry

- Ghi đè các phương thức yêu cầu, bổ sung equals, hashCode

```
@Override
public V setValue(V value) {
    if (value == null) {
        throw new NullPointerException();
    }
    V oldValue = this.value;
    this.value = value;
    return oldValue;
}

@Override
public boolean equals(Object o) {
    if (!(o instanceof Map.Entry)) {
        return false;
    }
    Map.Entry<?, ?> e = (Map.Entry<?, ?>) o;
    return (key == null ? e.getKey() == null : key.equals(e.getKey())) &&
           (value == null ? e.getValue() == null : value.equals(e.getValue()));
}

@Override
public int hashCode() { return hash ^ Objects.hashCode(value); }
```

Tạo lớp HashTable

```java
public class HashTable<K, V> {
    private static final int MAX_ARRAY_SIZE = Integer.MAX_VALUE - 8;
    private int count; // tổng số phần tử trong bảng băm
    private Entry<?, ?>[] table; // dữ liệu của bảng băm
    private int threshold; // ngưỡng
    private float loadFactor; // load factor
    private int modCount = 0; // số lần bảng băm sửa đổi cấu trúc, ví dụ băm lại

    public HashTable(int capacity, float loadFactor) {
        if (capacity < 0) {
            throw new IllegalArgumentException("Illegal Capacity: " + capacity);
        }
        if (loadFactor <= 0) {
            throw new IllegalArgumentException("Illegal Load: " + loadFactor);
        }
        if (capacity == 0) {
            capacity = 1;
        }
        this.loadFactor = loadFactor;
        table = new Entry<?, ?>[capacity];
        threshold = (int) Math.min(capacity * loadFactor, MAX_ARRAY_SIZE + 1);
    }
```

Phương thức get(key)

```java
/**
 * Phương thức lấy value tương ứng với key
 * @param key khóa cần lấy giá trị tương ứng trong bảng băm
 * @return giá trị value liên kết với key
 */
public V get(Object key) {
    Entry<?, ?>[] tab = table; // lấy bảng băm
    int hash = key.hashCode(); // lấy mã băm của key
    int index = (hash & 0x7fffffff) % tab.length; // lấy chỉ số trong bảng băm
    for (var e : HashTable<...>.Entry<...> = tab[index]; e != null; e = e.next) {
        if ((e.hash == hash) && e.key.equals(key)) { // tìm thấy
            return (V) e.value; // trả về giá trị value
        }
    }
    return null; // không tìm thấy, trả về null
}
```

Phương thức put(key, value)

```java
/**
 * Phương thức thêm mới một cặp key-value vào bảng băm
 *
 * @param key   // khóa
 * @param value // giá trị liên kết với khóa
 * @return giá trị value cũ tại vị trí index
 */
public V put(K key, V value) {
    if (value == null) {
        throw new NullPointerException();
    }
    Entry<?, ?>[] tab = table;
    int hash = key.hashCode();
    int index = (hash & 0x7fffffff) % tab.length;
    var entry = (Entry<K, V>) tab[index];
    for (; entry != null; entry = entry.next) {
        if (entry.hash == hash && entry.key.equals(key)) {
            V old = entry.value;
            entry.value = value;
            return old;
        }
    }
    addEntry(hash, key, value, index);
    return null;
}
```

Phương thức thêm mới

```java
/**
 * Phương thức thêm mới entry vào bảng băm
 * @param hash // mã băm của key
 * @param key // khóa
 * @param value // giá trị liên kết với khóa
 * @param index // vị trí
 */
private void addEntry(int hash, K key, V value, int index) {
    Entry<?, ?>[] tab = table;
    if (count >= threshold) { // nếu đạt ngưỡng băm lại bảng
        rehash(); // băm lại
        tab = table;
        hash = key.hashCode(); // lấy mã băm
        index = (hash & 0x7fffffff) % tab.length; // lấy chỉ số
    }
    var e = (Entry<K, V>) tab[index]; // phần tử next của entry mới
    tab[index] = new Entry<>(hash, key, value, e);
    count++; // tăng số phần tử trong bảng băm
    modCount++;
}
```

Nội dung tiếp theo

Xóa cặp key-value