Lesson 11
Hash Tables:

Pure Consciousness is the Home of All Knowledge

Wholeness Statement

A hash table provides constant-cost find, insert, and delete operations on values by key; it can quickly decide where the data value should be located for fast insertion and retrieval.

The home of all the laws of nature is available and accessible in the field of pure consciousness.

Hash Table vs. Array

- Definition : An array into which data is inserted using a hash function is called a hash table.
- Array
  - An integer index is the key
- A Hash table is a generalization of an array
  - *keys* are used to look up corresponding *values*
  - Any object can be used as a key
- Example
  - A key field from a database can be used as a key in the hash table
  - the corresponding DB record could be the value in the hash table

- Two basic operations: (usually also have a remove(Object key) operation)
  - void put(Object key, Object value);
  - Object get(Object key)
- Simple Example (this is an elementary implementation of Hashtable ADT)

User's view

Char key	String value
'a'	"Adam"
'b'	"Bob"
'c'	"Charlie"
'w'	"William"

```
//insert into table
table.put('c', "Charlie");
//retrieve from table
table.get('c');   //returns "Charlie"
```

Implementation:

put(c, s):
obtain array index: c → (int)c → i = (int)c – 'a'
insert new Entry(c, s) into table[i]

get(c):
obtain array index: c → (int)c → i = (int)c – 'a'
Entry e = table[i]; return e.value

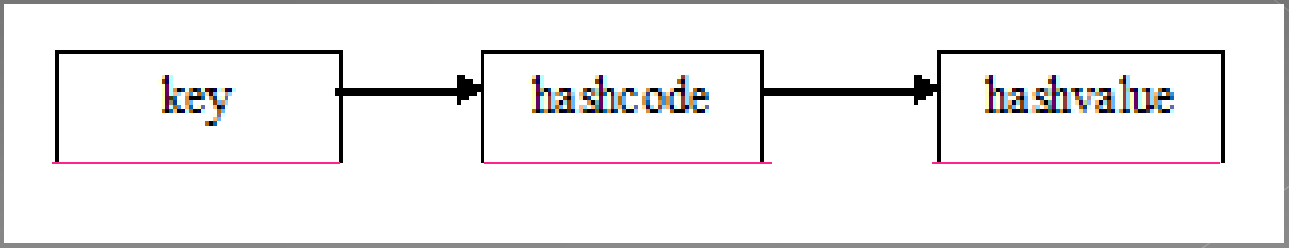
Pattern: KEY (non-number) → HASHCODE (number) → HASH VALUE (array index)

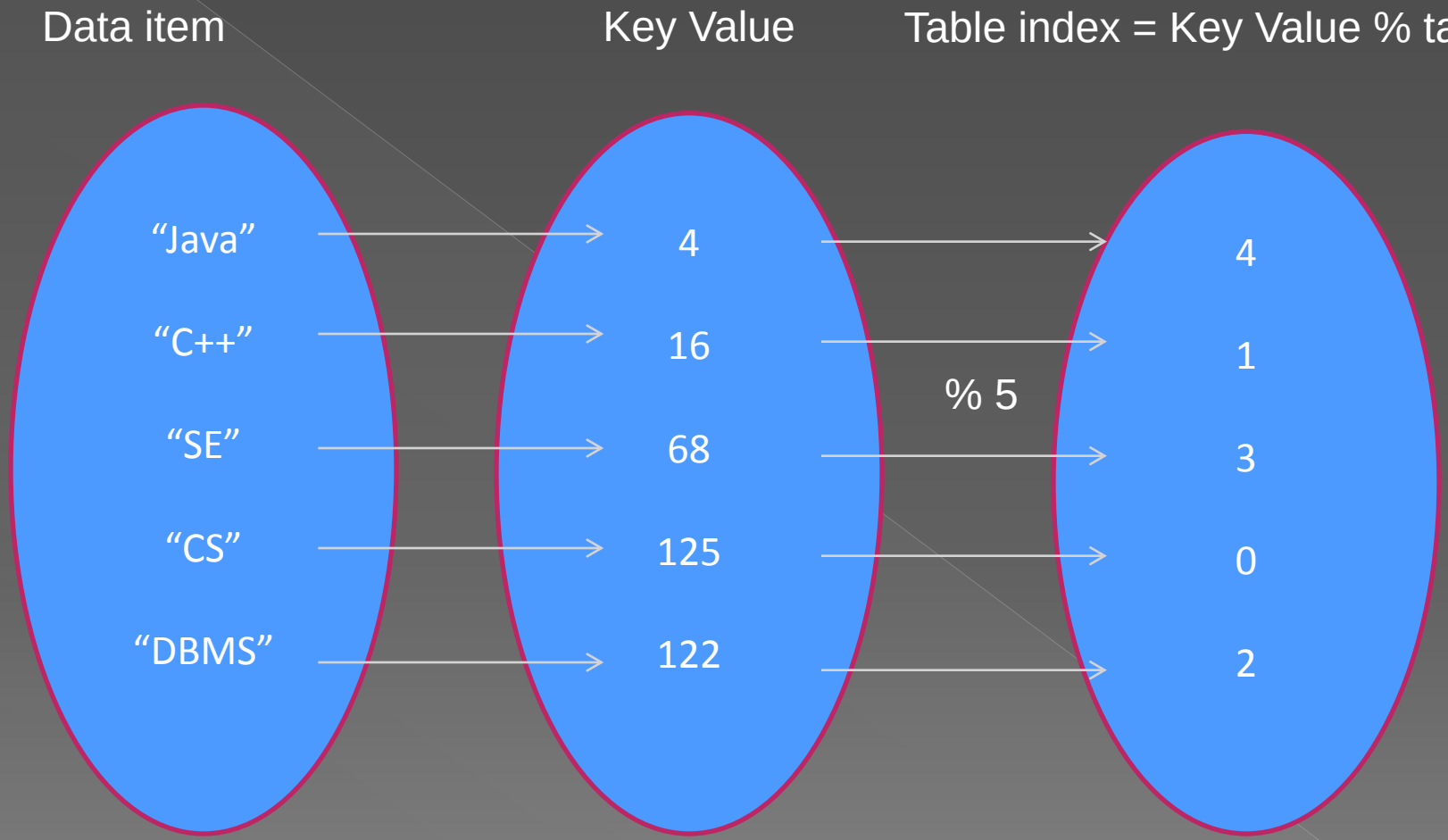
Basic steps in creating a hashtable data structure:

a. Devise a way of converting keys to integers so that different keys are mapped to different integers. This is what Java's hashCode() function is for – must be overridden by every class (since hashCode() is a method of the Object class).

b. Devise a way of converting hashcodes to smaller integers (called *hash values*) that will be the indices of a smaller array, called the *table.* To do this, you usually need to decide on the tableSize, which is the length of the array. A typical way (note: Java does it differently – see below) of making hashvalue from hashcode is by the formula

hashvalue = hashcode % tableSize





CS	C++	DBMS	SE	JAVA
0	1	2	3	4

Computing a hashcode for a string in Java

The method hashCode has different implementation in different classes.

Devise a way of converting keys to integers so that different keys are mapped to different integers. This is what Java's hashCode() function is designed to do.

Example: Any Java String is converted to an integer via hashCode() by this formula: (Note that hashCode() is overridden in the String class.

In the String class, hashCode is computed by the following formula

$s.charAt(0) * 31^{n-1} + s.charAt(1) * 31^{n-2} + ... + s.charAt(n-1)$

where *s* is a string and *n* is its length. An example

String s = “ABC”; n = 3 (Size of the string)

"ABC" = 'A' * 31^2 + 'B' * 31 + 'C' = 65 * 31^2 + 66 * 31 + 67 = 64578

Collisions

- Two values can hash to the same array index, resulting in collision. It can be resolved using :
- **Open Addressing**: Search the array in some systematic way for an empty cell and insert the new item there if collision occurs.
  - Linear Probing (Search sequentially for vacant cells until find an empty)
  - Quadratic Probing(In quadratic probing, probes go to x+1, x+4, x+9, and so on)
  - Double Hashing(Double the sequence with constant factor)
- **Separate chaining**: Create an array of linked list, so that the item can be inserted into the linked list if collision occurs.

Creating Good Hash Codes

- User implementation you must specify the size of the hash table.
- In order to reduce the chance for collisions, you should make a hash table somewhat larger than the number of elements that you expect to insert.
- The table size should be a prime number, larger than the expected number of elements.
- In the standard library, you don't need to supply a table size. If the hash table gets too full, a new table of twice the size is created, and all elements are inserted into the new table.

The Hash Table ADT

- Create an empty hash table

  new object constructor call

- Add a key-value pair to hash table

  void put(Object key, Object value)

- Retrieve the value associated with a key

  Object get(Object key)

- Remove a key-value pair from hash table

  Object remove(Object key)

- Remove all key-value pairs from hash table

  void removeAll( )

Hash Tables

- Advantages
  - Simpler and faster than binary search tree implementations
- Disadvantages
  - Does not efficiently support traversing the table in sorted order
  - Requires estimating the maximum number of table items

Demo : HashTableAPP.java (Linear Probing)

Main Point 1

A hash table has three components: a hash function to convert keys to slots in a table, an array containing key-value pairs, and a collision resolution strategy. The field of Pure Creative Intelligence contains all knowledge, in seed form, that is necessary for maintaining order in the Universe.

Hash Tables in j2se5.0

- java.util.HashMap
  - Allows null keys
  - Allows null values
  - Not synchronized for safe multithreading
- java.util.Hashtable
  - Does not allow null keys
  - Does not allow null values
  - Synchronized for safe multithreading

The above two classes are roughly equivalent except for the differences noted above.

Hash Table Applications

1. In-memory look-up tables
   - Employee records from a database could be stored by using Emloyee ID as key and the entire Employee record as value.

2. Avoid Duplicates(HashSet)

Predefined Library for Hash Concepts

HashSet : Hash table implementation of the Set interface.

- HashSet does not allow duplicate values but allow null value.
- It provides add method rather put method.
- HashSet can be used where you want to maintain a unique list.

HashMap : Hash table implementation of the Map interface.

- Like Hashtable it also accepts key value pair.
- It allows null for both key and value.
- Does not allow duplicate keys.
- It is unsynchronized. So come up with better performance
- Iterator is used to Iterate.

Hashtable : Hash table implementation of the Map interface.

- Hashtable is basically a datastructure to retain values of key-value pair.
- Does not allow duplicate keys.
- It didn’t allow null for both key and value. You will get NullPointerException if you add null value.
- It is synchronized.
- Enumeration is used to Iterate.

Java Implementation Of HashMap

```
HashMap<String, Employee> map =
                new HashMap<String,Employee>();

map.put("Bob", new Employee("Bob", 40000, 1996, 10, 2));
Employee emp = map.get("Bob");
```

Overriding the hashCode() Method

1. Any implementation of the Hashtable ADT in Java will make use of the hashCode() function as the first step in producing a hash value (or table index) for an object that is being used as a key.

2. Default implementation of hashCode() provided in the Object class is not generally useful.

Example: We wish to use pairs (firstName, lastName) as keys for Person objects in a hashtable. (See Demo)

Demo illustrates that default hashCode method is not useful. By default, it simply gives a numeric representation of the memory location of an object. If two Pair objects, created at different times, are equal (using the equals method), we would expect them to have the same hashCodes, so that, after hashing, they are sent to the same table slot. But default hashCode method does not take into account the fields used by equals method, so equal Pair objects may be assigned different slots in the table.

3. **Conclusion:** *Whenever* equals *is overridden in a class,* hashCode *must also be overridden.*

- **Example.** *Overriding hashCode in the Person-Pair example.* We must take in account the same fields in computing hashCode as those used in overriding equals. The fields in Pair are Strings, and Java already provides hashCodes for Strings. So we make use of these and combine them to produce a complex hashCode for Pair.

```
public int hashCode() {
        int result = 17; //seed
        int hashFirst = first.hashCode();
        int hashSecond = second.hashCode();
        result += 31 * result + hashFirst;
        result += 31 * result + hashSecond;
        return result;
}
```

Creating a Hash Value from Object Data (From Effective Java, 2nd Ed.)

- You are trying to define a hash value for each instance variable of a class. Suppose f is such an instance variable.
  - If f is boolean, compute (f ? 1 : 0)
  - If f is a byte, char, short, or int, compute (int) f.
  - If f is a long, compute (int) (f ^ (f >>> 32))
  - If f is a float, compute Float.floatToIntBits(f)
  - If f is a double, compute Double.doubleToLongBits(f) which produces a long f1, then return (int) (f1 ^ (f1 >>> 32))
  - If f is an object, compute f.hashCode()

Formula for creating your hashCode function

Step 1. Use the table above to produce a temporary hash of each variable in your class.

Example: You have variables u, v, w. Produce (using the chart above) temporary hash vals hash_u, hash_v, hash_w.

Step 2. Combine these temporary hashes into a final hashCode that is to be returned

Example:

```
int result = 17;
result += 31 * result + hash_u;
result += 31 * result + hash_v;
result += 31 * result + hash_w;
return result;
```

Demo Code

- HashSetDemo.java
- HashtableDemo.java
- HashMapDemo.java
- HashtableDemo3.java
- needoverridehashcode package

Use Of Data Structures

- Array List
  - Use: when main need is for a list with random access reads, infrequent adds beyond initial capacity (or maximum number of list elements is known in advance), or the list only needs to occasionally be sorted
  - Avoid: when many adds expected, but number of elements unpredictable, or inserts need to maintain some ordering by key.
- Linked List
  - Use: when insertions and deletions are frequent, and/or many elements need to be added, but total number is unknown in advance
  - Avoid: when there is a need for repeated access to the nth element (random access) (as in sorting)

Use Of Data Structures (cont.)

- Binary Search Tree
  - Use: when data needs to be maintained in sorted order by key for frequent searches
  - Avoid: when the extra benefit of keeping data in sorted order is not needed and rapid read access (e.g., iteration or random access) is needed
- Hash Table
  - Use: when random access to objects is needed but array indexing is not practical (possible index range is too large)
  - Avoid: when an ordering (possibly unrelated to keys(maybe by data input)) of data must be preserved, or iterating through values in the table is frequent

Use Of Data Structures (cont.)

- Set
  - Use: when elements do not need to be kept in a special order and searches for elements in the set are infrequent (do not need to be rapid)
  - Avoid: when an ordering of data must be preserved or searches for elements in the set are frequent (instead use TreeSet which uses a binary search tree, and is ordered)

Main Point 2

With the knowledge of data structures such as Lists, Stacks, Queues, Trees, and Hash Tables one can design programs that run most efficiently and simply. Knowledge and proper use of data structures illustrates the principle of “Do less and accomplish more”.

Unity Chart

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

Random access expanded from integer index to arbitrary index.

1. Arrays and ArrayLists provide highly efficient index-based access to a collection of elements.
2. The Hashtable ADT generalizes the behavior of an array by allowing non-integer keys (in fact, any object type can be used for a key), while retaining essentially random access efficiency for insertions, deletions, and lookups.

Transcendental Consciousness: TC is the home of all knowledge. The Upanishads declare "Know that by which all else is known" – this is the field of pure consciousness.

Impulses within the Transcendental field: *The infinite diversity of these impulses within the transcendental field, is possible due to the infinite creativity and intelligence.*

Wholeness moving within Itself*: In Unity Consciousness, one sees that the "key" to accessing complete knowledge of any object is the infinite value of that object, pure consciousness, which is known in this state to be one's own Self.*