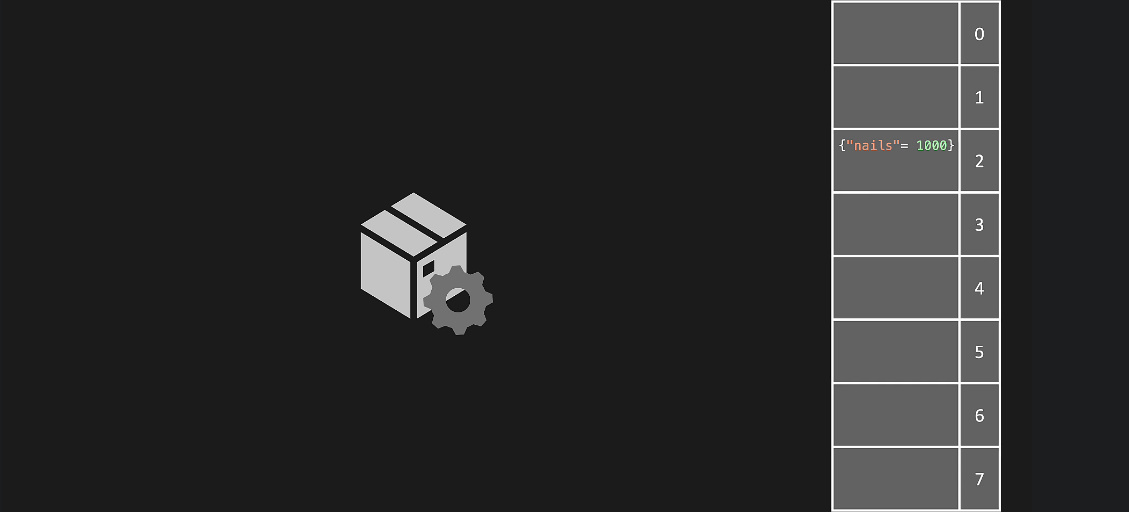
**HashTable**

* Fiecare element Este format din perechea **key-value**
* E thread safe
* Pentru a putea adauga elemente, obiectele folosita ce key trebuie sa suprascrie metode **hashCode()** si **equals()**
* Key care noi o specificam este hashed apoi si hash code rezultat este folosit ca index pentru a pastra perechea key value
* 





* hashMethod() este unidirectionala, adica poate transfora key in hash code(adresa), dar nu poate transforma hash code(adresa) in key inapoi
* Aceeasi key va genera mereu acelasu hash code
* Diferite keys se poate intampla sa genereze acelasi Hash Code, si aici se creaza o **Coliziune.** Oricum, perechea va fi pusa la acelasi index chiar de acolo deja e o pereche existenta si vor exista ambele

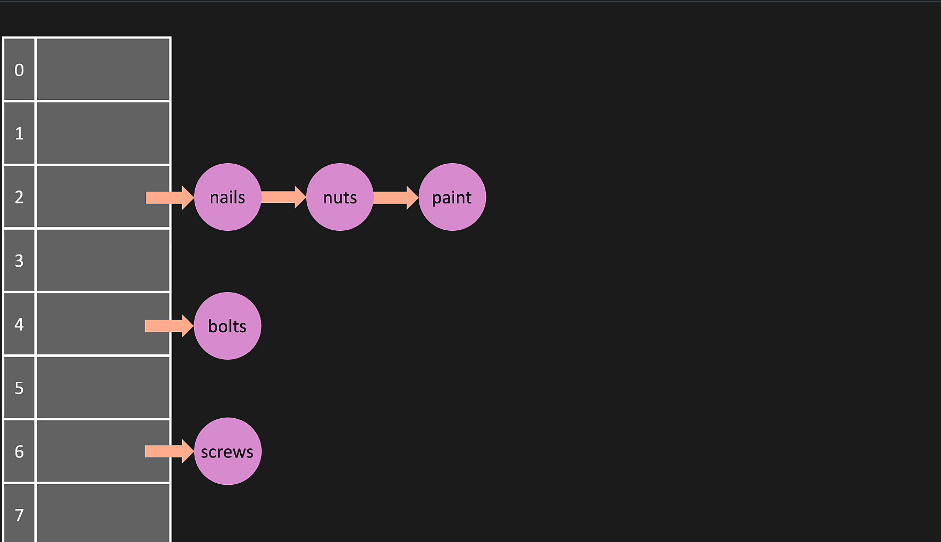
**Coliziunea**

* Diferite keys se poate intampla sa genereze acelasi Hash Code, si aici se creaza o **Coliziune**
* **Exista diferite solutii pentru a ne ocupa de asta:**

1. **Separate Chaining –** punem toate perechile cu acelasi Hash Code la acelasi index:

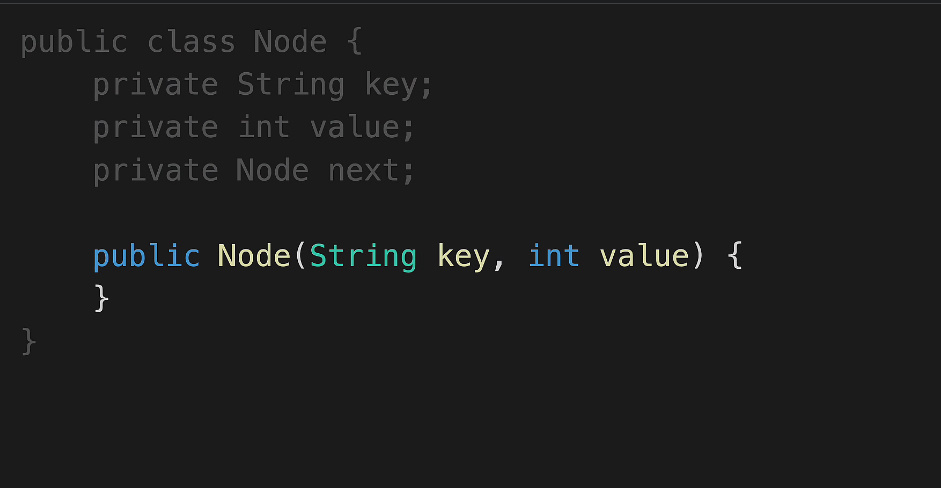


Daca un index are mai multe pairs, ele vor fi stocate intr-un linkedlist:



1. **Linear Probing** - Daca 2 sau mai multe key au acelasi HashCode, atunci nu inseram mai multe perechi la acelasi index, dar perechea care ar trebuie sa fie pusa va cauta urmatorul index disponibil, si acolo va fi pusa

**Methods**

* HashTable are un initial size mereu. Cel mai bine e ca size sa fie un numar prim, si asa sansele de coliziuni sunt mai mici
* 

public class HashTable {

private Node[] dataMap;

private int size = 7;

class Node{

Node next;

String key;

int value;

public Node(String key, int value){

this.key = key;

this.value = value;

}

}

public HashTable(){

dataMap = new Node[size];

}

public Node[] getDataMap() {

return dataMap;

}

public void printTable() {

for(int i = 0; i < dataMap.length; i++) {

System.out.println(i + ":");

if(dataMap[i] != null) {

Node temp = dataMap[i];

while (temp != null) {

System.out.println(" {" + temp.key + ", " + temp.value + "}");

temp = temp.next;

}

}

}

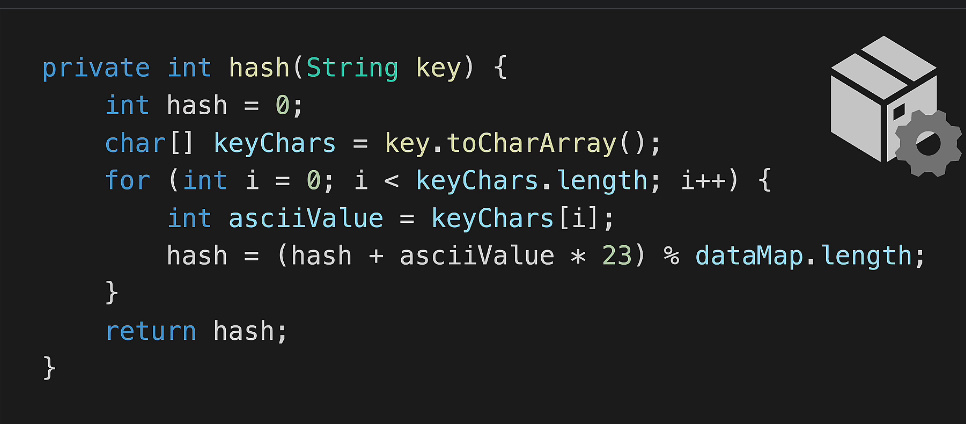
}

}

* private hash(key)

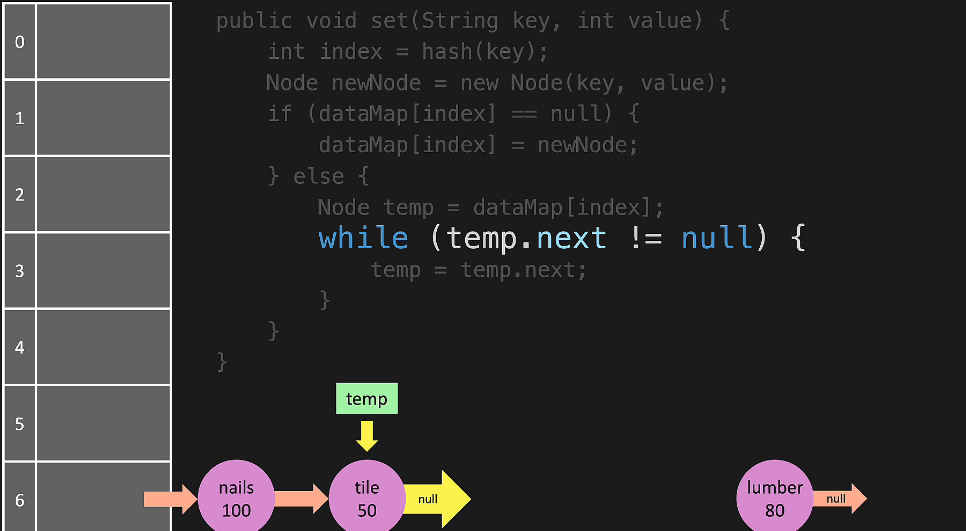
Atentie!Metoda e private. Ea va fi folosita doar de alte metode din clasa HashTable

Putem folosi orice logica pentru hash, dar mai jos e un exemplu:

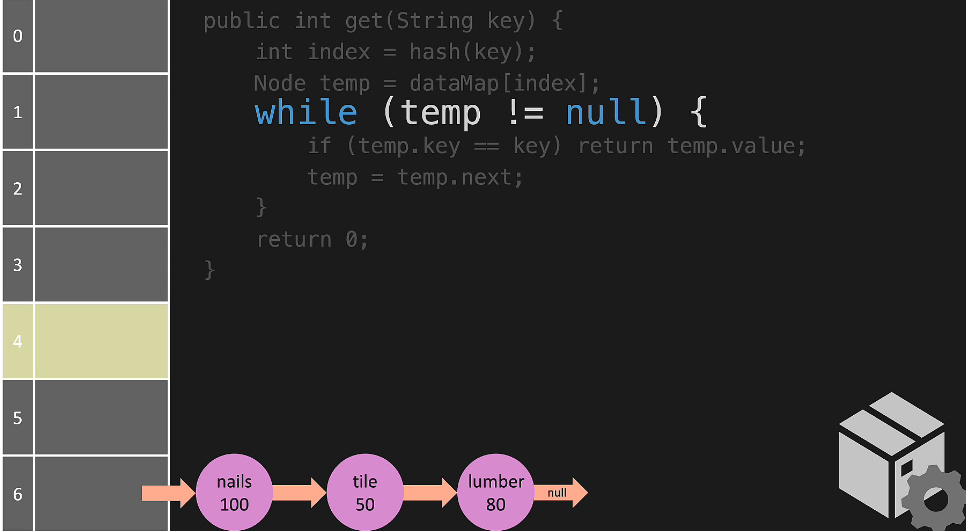


23 este un numar prim, nici prea mare, nici prea mic, dar poate fi folosit si altul

* set(key,value)



* get(key)



**Big O**

* **hash**() – O(1)
* **set**() – O(1)
* **get**(key) – O(1)

Elementele sunt puse mai mult random, de aia se socoate O(1)

* **get(value) –** O(n)

**Interview Questions**

1. Sa se scrie un program care afla daca 2 arrays contin un element comun.

* Putem face asta cu 2 for(), dar complexitatea e O(n^2), si nu e cam buna



Asa va fi O(n):

