CSCI 232:

Data Structures and Algorithms

Hashing (Part 2)

Reese Pearsall

Spring 2024

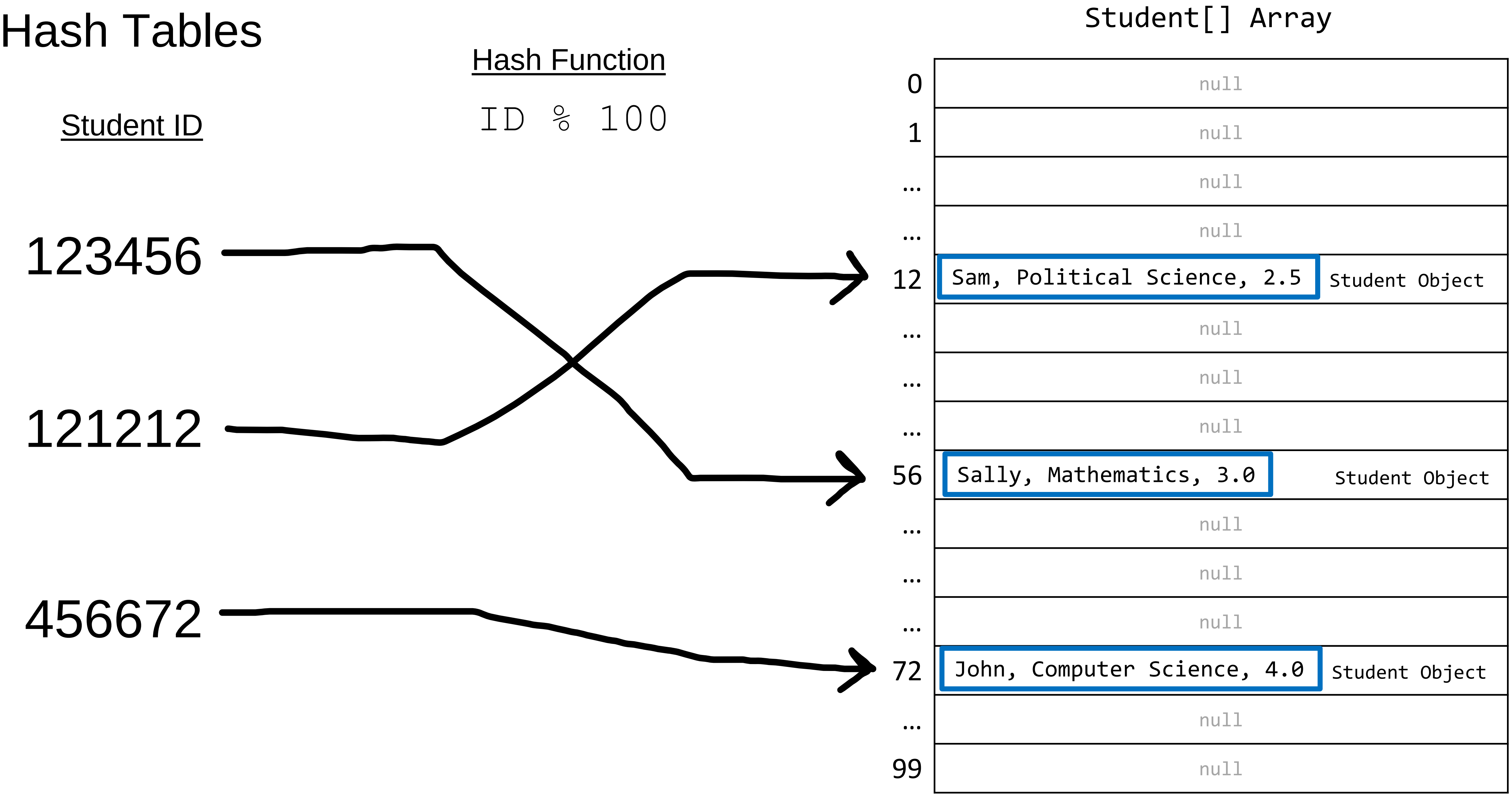
Announcements

Lab 5 due **tomorrow** at 11:59 PM

Program 1 due **Tuesday** at 11:59 PM

No class on Thursday 2/29

Hash Tables



Hash Function

`ID % 100`

Student ID

123456

121212

456672

Student[] Array

0	null	
1	null	
…	null	
…	null	
12	Sam, Political Science, 2.5	Student Object
…	null	
…	null	
…	null	
56	Sally, Mathematics, 3.0	Student Object
…	null	
…	null	
…	null	
72	John, Computer Science, 4.0	Student Object
…	null	
99	null	

Hash Tables

Hash Function

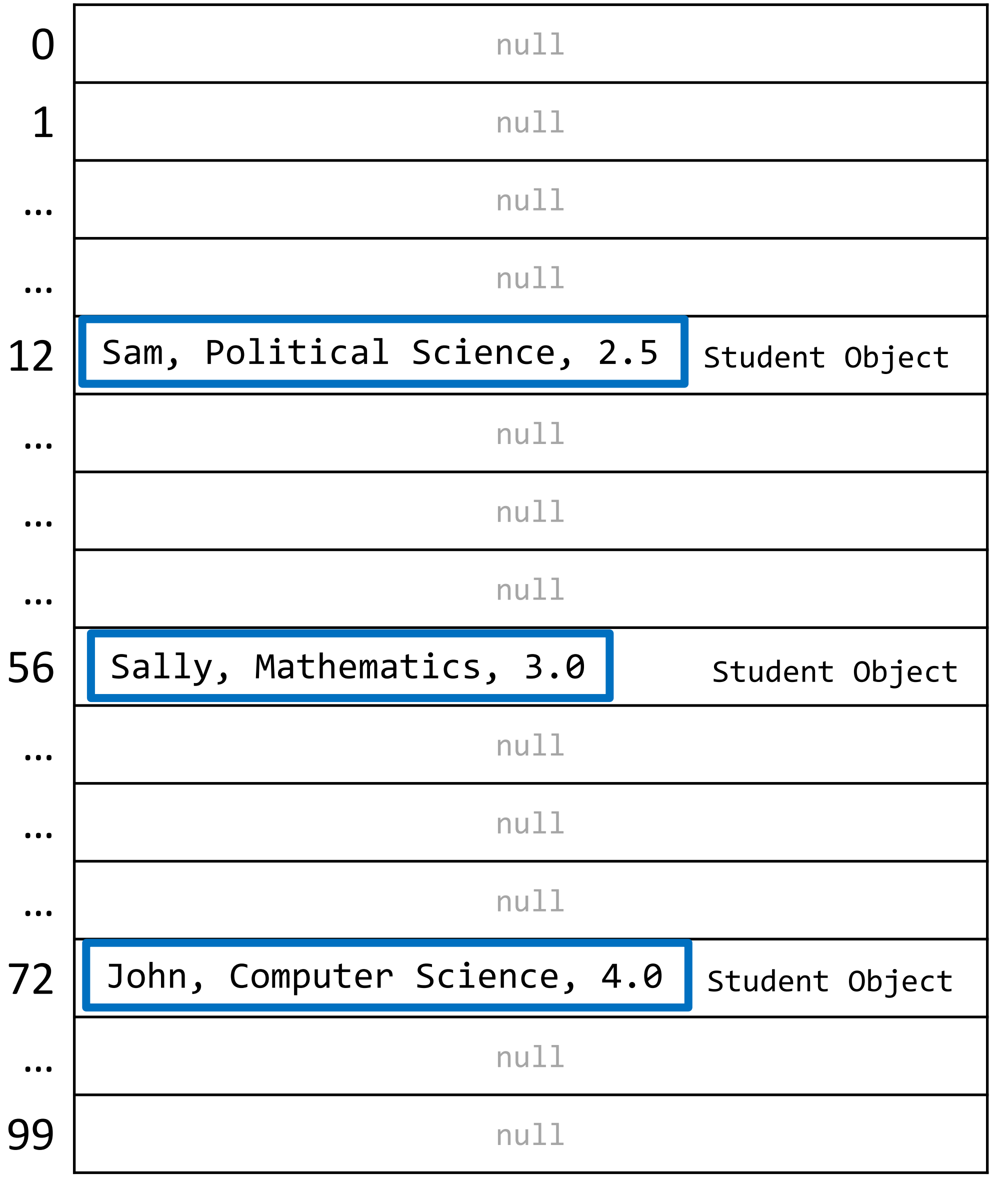
Student ID

ID % 100

Lookup time?

O(1) if you have the key

`Student[] Array`   n = 100



0 null

1 null

… null

… null

12 Sam, Political Science, 2.5 Student Object

… null

… null

… null

56 Sally, Mathematics, 3.0 Student Object

… null

… null

… null

72 John, Computer Science, 4.0 Student Object

… null

99 null

Hash Tables

Student ID

Hash Function

`ID % 100`

Lookup time?

O(1) if you have the key

O(n) if you don’t have the key

`Student[] Array` n = 100

0	null	
1	null	
…	null	
…	null	
12	Sam, Political Science, 2.5	Student Object
…	null	
…	null	
…	null	
56	Sally, Mathematics, 3.0	Student Object
…	null	
…	null	
…	null	
72	John, Computer Science, 4.0	Student Object
…	null	
99	null	

Hash Tables

Student ID

Hash Function

```
ID % 100
```

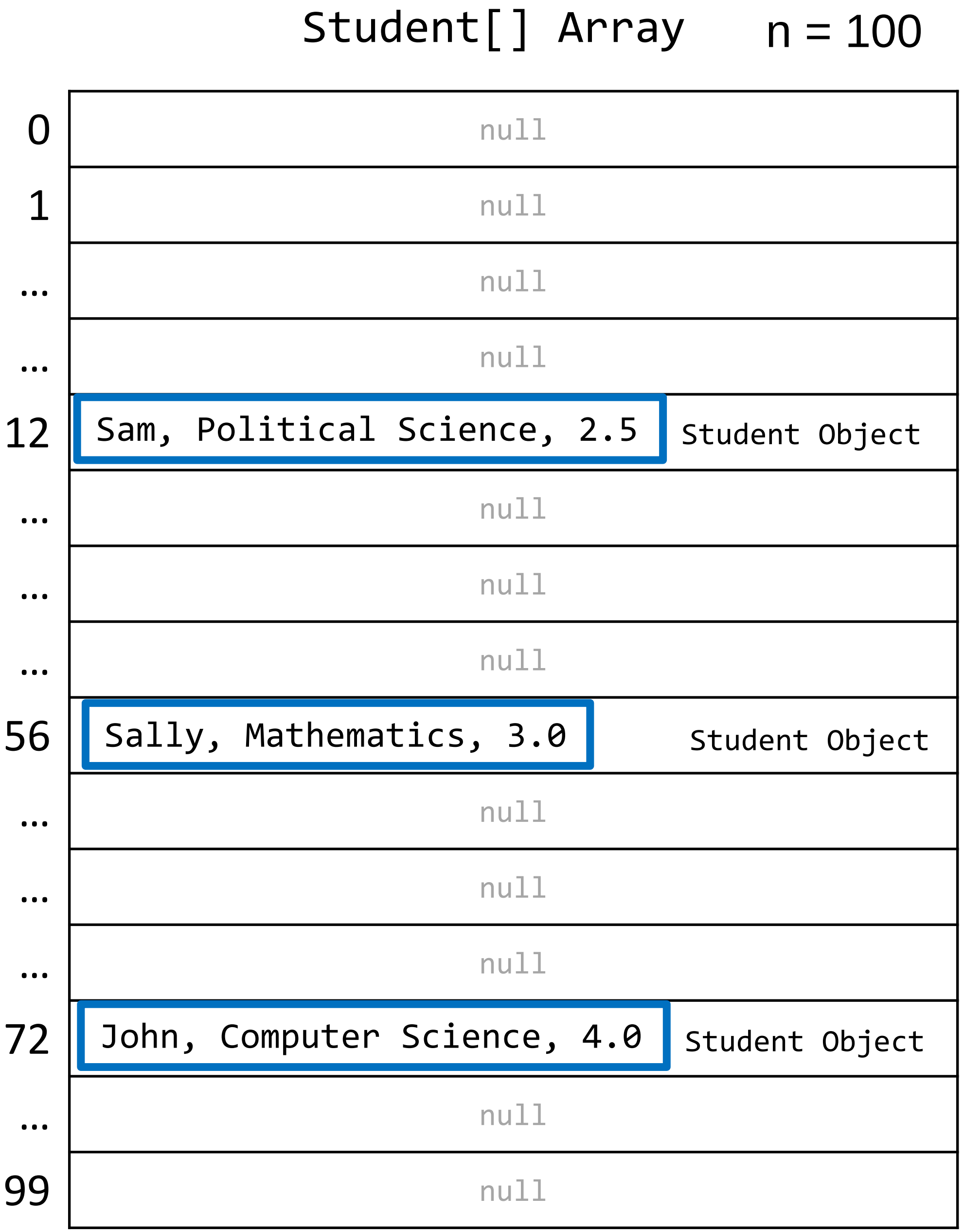
Lookup time?

O(1) if you have the key

~~**O(n)** if you don’t have the key~~

O(k) if you don’t have the key

k = | keyspace |



Student[] Array   n = 100

0	null	
1	null	
…	null	
…	null	
12	Sam, Political Science, 2.5	Student Object
…	null	
…	null	
…	null	
56	Sally, Mathematics, 3.0	Student Object
…	null	
…	null	
…	null	
72	John, Computer Science, 4.0	Student Object
…	null	
99	null	

Hash Tables

Student ID

Hash Function

`ID % 100`

Lookup time?

O(1) if you have the key*

n = # of elements in data structure

Array – **O(logn)****

BST – **O(logn)*****

Linked List – **O(n)**

* If we can avoid collisions

**if the array is sorted

***if the tree is balanced

`Student[] Array` n = 100

0	`null`	
1	`null`	
…	`null`	
…	`null`	
12	`Sam, Political Science, 2.5`	`Student Object`
…	`null`	
…	`null`	
…	`null`	
56	`Sally, Mathematics, 3.0`	`Student Object`
…	`null`	
…	`null`	
…	`null`	
72	`John, Computer Science, 4.0`	`Student Object`
…	`null`	
99	`null`	

Hash Tables

Student ID

Hash Function

`ID % 100`

Insertion time?

O(1) *

n = # of elements in data structure

Array – **O(n)**

BST – **O(logn)****

Linked List – **O(1)**

* If we can avoid collisions

**if the tree is balanced

`Student[] Array` n = 100

0	null	
1	null	
…	null	
…	null	
12	Sam, Political Science, 2.5	Student Object
…	null	
…	null	
…	null	
56	Sally, Mathematics, 3.0	Student Object
…	null	
…	null	
…	null	
72	John, Computer Science, 4.0	Student Object
…	null	
99	null	

Hash Tables

Student ID

Hash Function

```
ID % 100
```

Removal time?

O(1) *

n = # of elements in data structure

Array – **O(n)**

BST – **O(logn)****

Linked List – **O(1) / O(n)**

* If we can avoid collisions

**if the tree is balanced

`Student[] Array` n = 100

0	null	
1	null	
…	null	
…	null	
12	Sam, Political Science, 2.5	Student Object
…	null	
…	null	
…	null	
56	Sally, Mathematics, 3.0	Student Object
…	null	
…	null	
…	null	
72	John, Computer Science, 4.0	Student Object
…	null	
99	null	

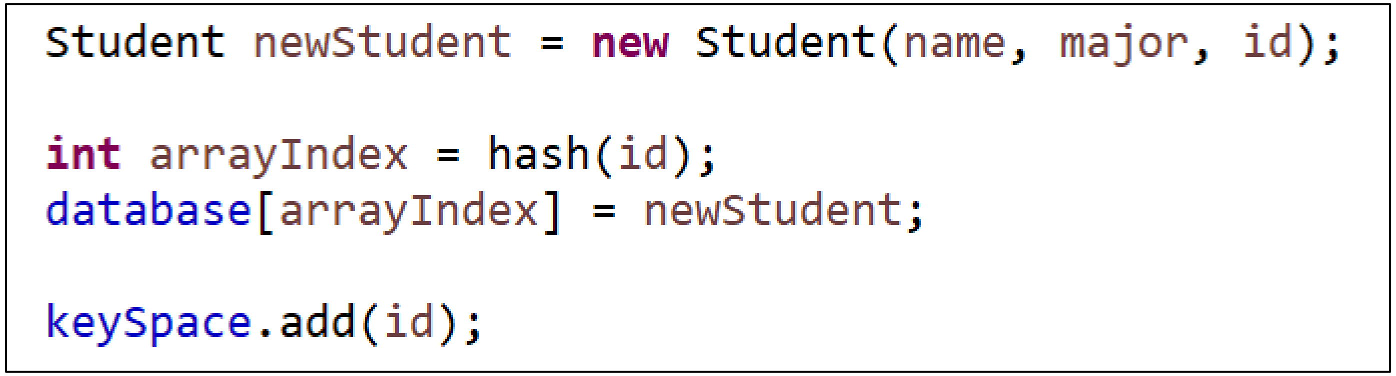
Hash Tables

Student ID

Hash Function

ID % 100

Insertion

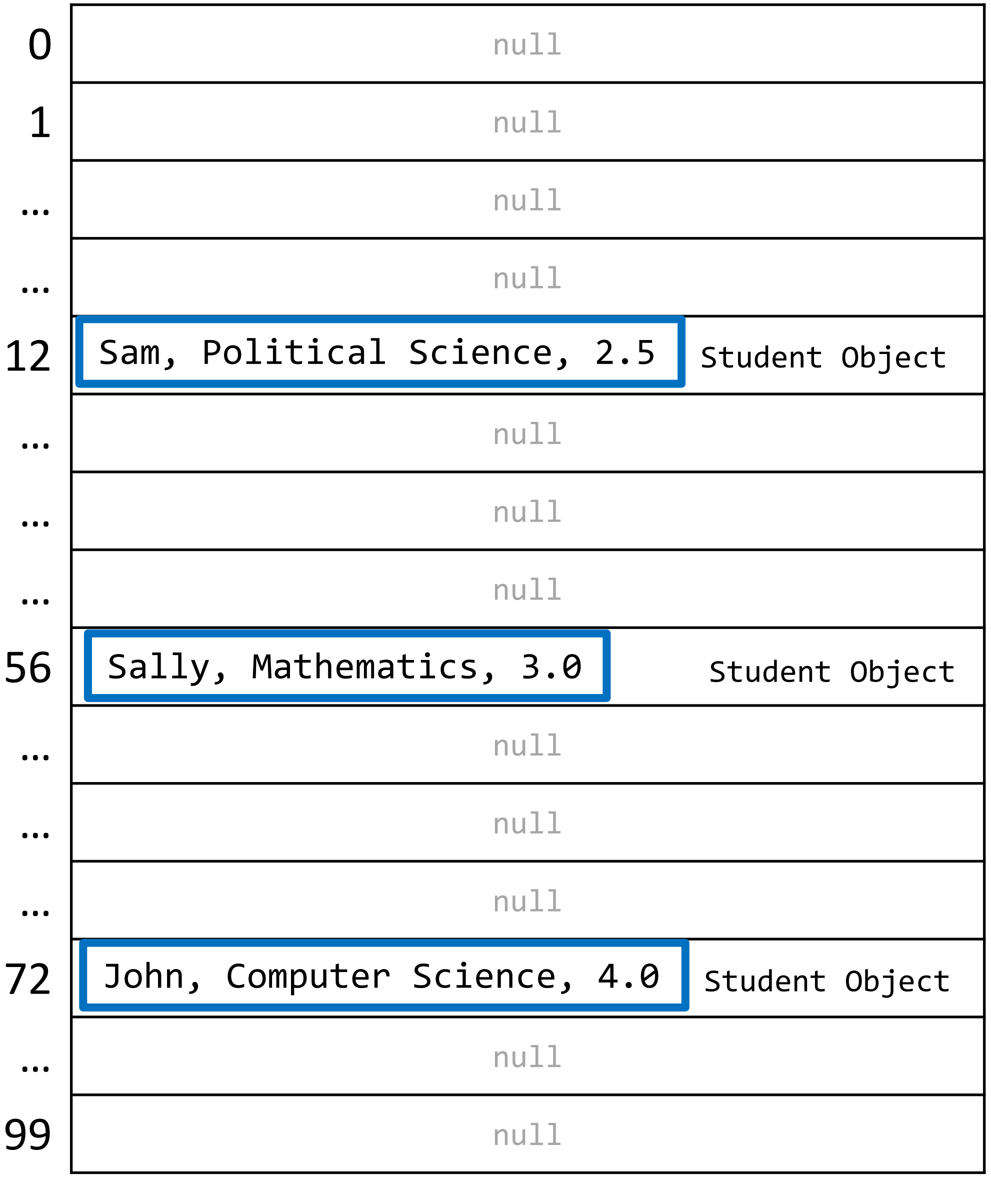


```
Student newStudent = new Student(name, major, id);

int arrayIndex = hash(id);
database[arrayIndex] = newStudent;

keySpace.add(id);
```

Student[] Array   n = 100



0	null	
1	null	
…	null	
…	null	
12	Sam, Political Science, 2.5	Student Object
…	null	
…	null	
…	null	
56	Sally, Mathematics, 3.0	Student Object
…	null	
…	null	
…	null	
72	John, Computer Science, 4.0	Student Object
…	null	
99	null	

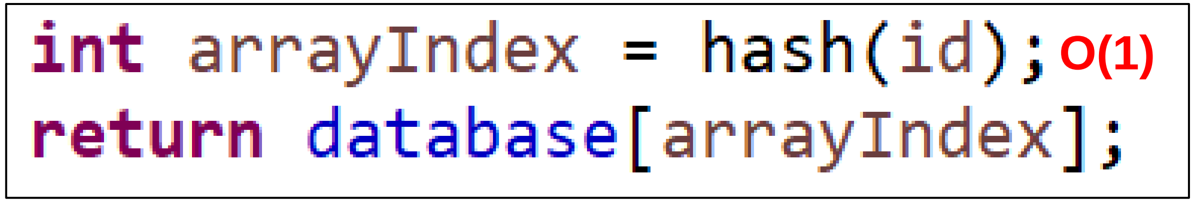
Hash Tables

Student ID

Hash Function

`ID % 100`

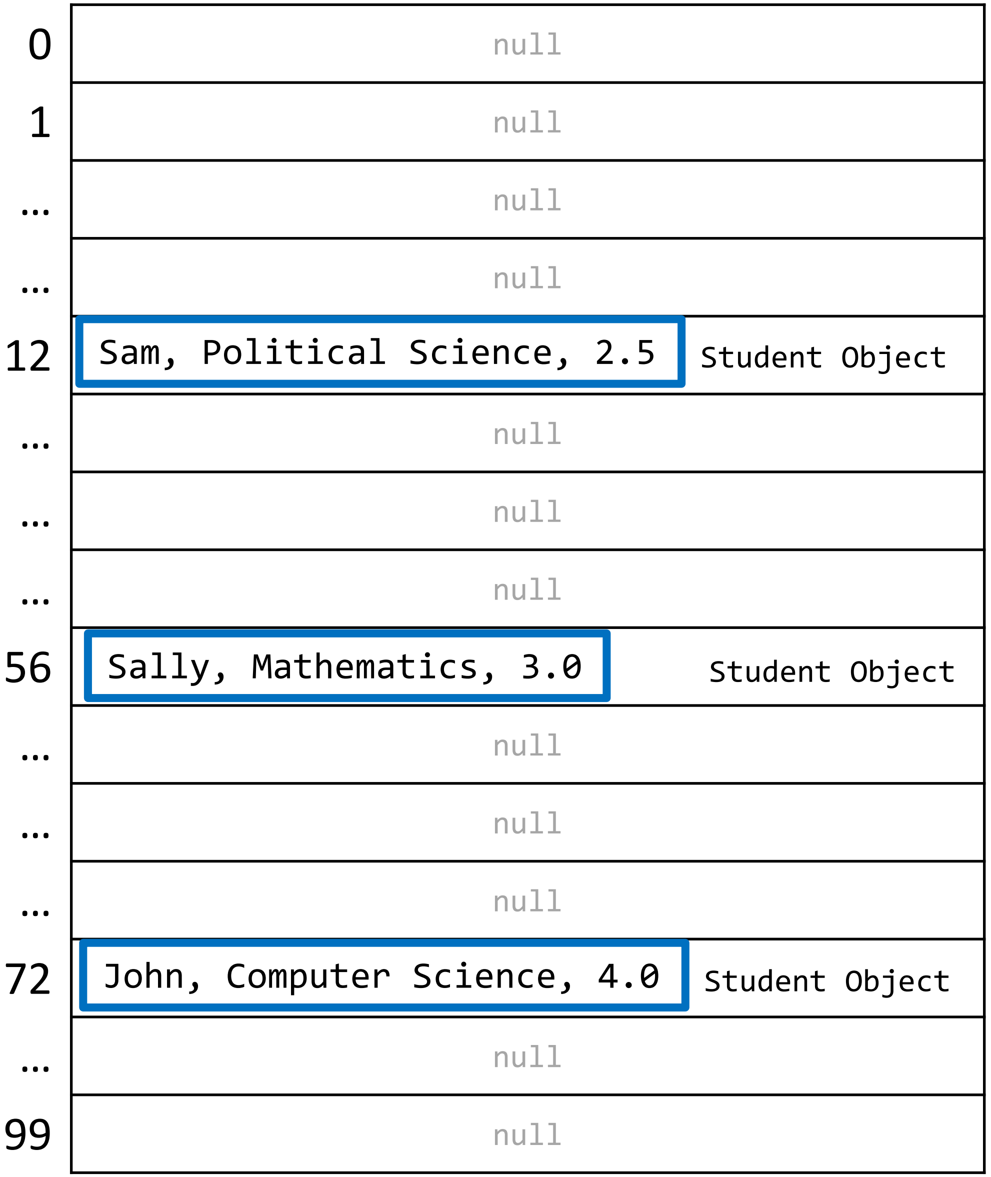
Lookup (**get**)



```
int arrayIndex = hash(id);
return database[arrayIndex];
```

O(1)

`Student[] Array`   n = 100



0	null	
1	null	
…	null	
…	null	
12	Sam, Political Science, 2.5	Student Object
…	null	
…	null	
…	null	
56	Sally, Mathematics, 3.0	Student Object
…	null	
…	null	
…	null	
72	John, Computer Science, 4.0	Student Object
…	null	
99	null	

Hash Tables

Hash Function

`ID % 100`

Student ID

Remove Method

`Student[] Array`   n = 100

0	null	
1	null	
…	null	
…	null	
12	Sam, Political Science, 2.5	Student Object
…	null	
…	null	
…	null	
56	Sally, Mathematics, 3.0	Student Object
…	null	
…	null	
…	null	
72	John, Computer Science, 4.0	Student Object
…	null	
99	null	

Hash Tables in Java

Typically, we will never have to create our own **`HashTable`** class, instead we will **import** the one that Java provides

```java
import java.util.HashMap;
import java.util.HashSet;
```

Hash Maps

Hash Maps are a collection of key-values pairs (**`Map`**) that uses hashing when inserting, removing, lookup, etc

```
HashMap<String, String> capitalCities = new HashMap<String, String>();
```

This is a HashMap that maps Strings (keys) to Strings (values)

Adding a new Key-Value pair

```
capitalCities.put("England", "London");
capitalCities.put("Germany", "Berlin");
capitalCities.put("Norway", "Oslo");
capitalCities.put("USA", "Washington DC");
```

Retrieving a Value

```
capitalCities.get("England");
```

Removing a Value

```
capitalCities.remove("England");
```

Other Helpful Methods

- `keySet()` → returns set of keys
- `values()` → returns set of values
- `containsKey()`
- `containsValue()`
- `replace()`
- `size()`

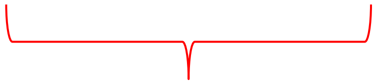
Hash Sets

Hash Sets is an implementation of the **`Set`** interface that uses a Hash Map under the hood

A **set** is a collection of elements with no duplicate elements

- You can think of this as a List, but without the ability to use indices

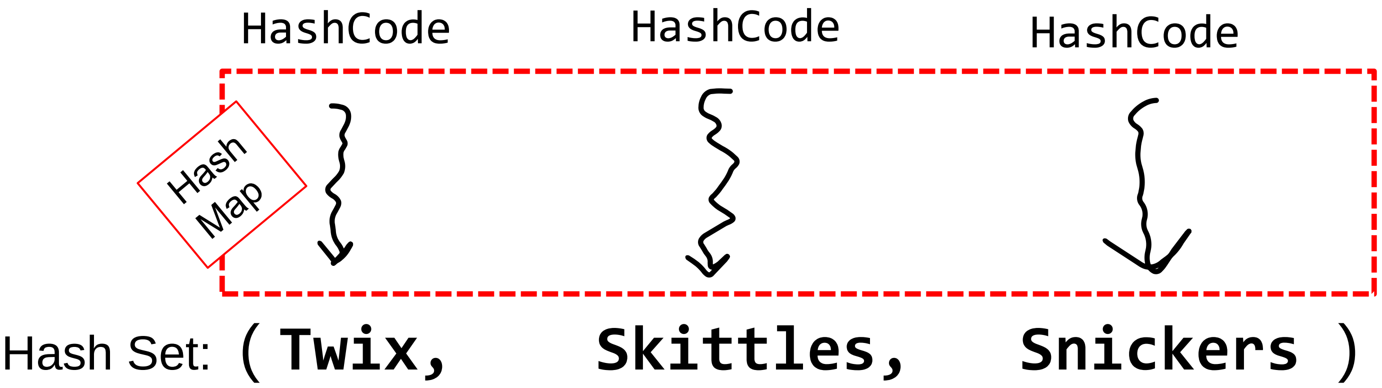
```
HashSet<String> candy = new HashSet<String>;
```



Hash Set that stores Strings

```
candy.add(“Twix”);
candy.add(“Skittles”);
candy.add(“Snickers”);

candy.contains(“Skittles”);
candy.remove(“Twix”);
```



Hash Set: (**`Twix,`** **`Skittles,`** **`Snickers`**)

Hash Sets

Hash Sets is an implementation of the **`Set`** interface that uses a Hash Map under the hood

A **set** is a collection of elements with no duplicate elements

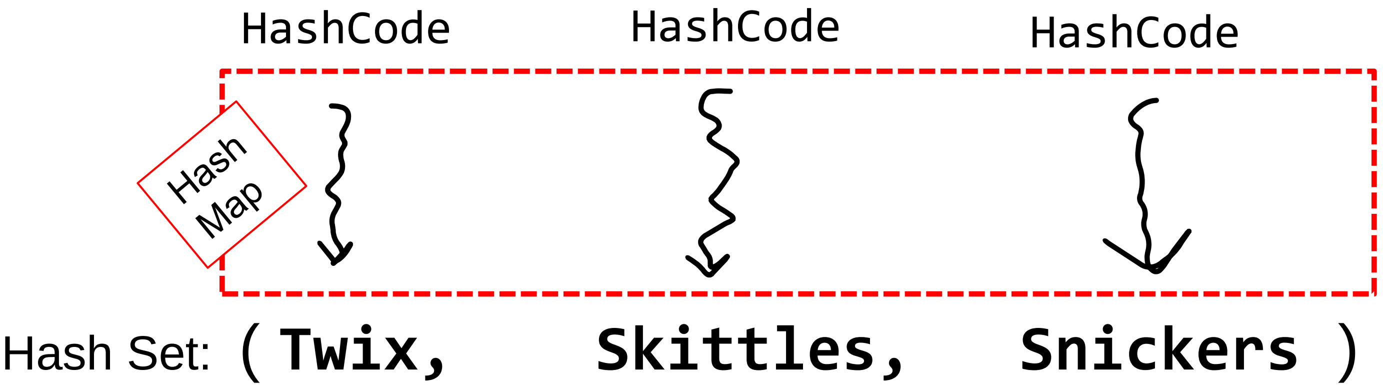
- You can think of this as a List, but without the ability to use indices

```
HashSet<String> candy = new HashSet<String>;
```

Hash Set that stores Strings

```
candy.add(“Twix”);
candy.add(“Skittles”);
candy.add(“Snickers”);

candy.contains(“Skittles”);
candy.remove(“Twix”);
```



Today’s Mandatory Fun

Updating our Student Database Class

- Replace Array with HashMap
- Replace ArrayList with HashSet

- Write a method that will compute the number of CS majors, Math Majors, History majors, etc
- Add method that will compute which student(s) have a 4.0, 3.0, 3.1, etc

Write a program that will convert an English sentence to sentence in Pirate

“Hello” → “Ahoy”
“Friends” → “Mateys”