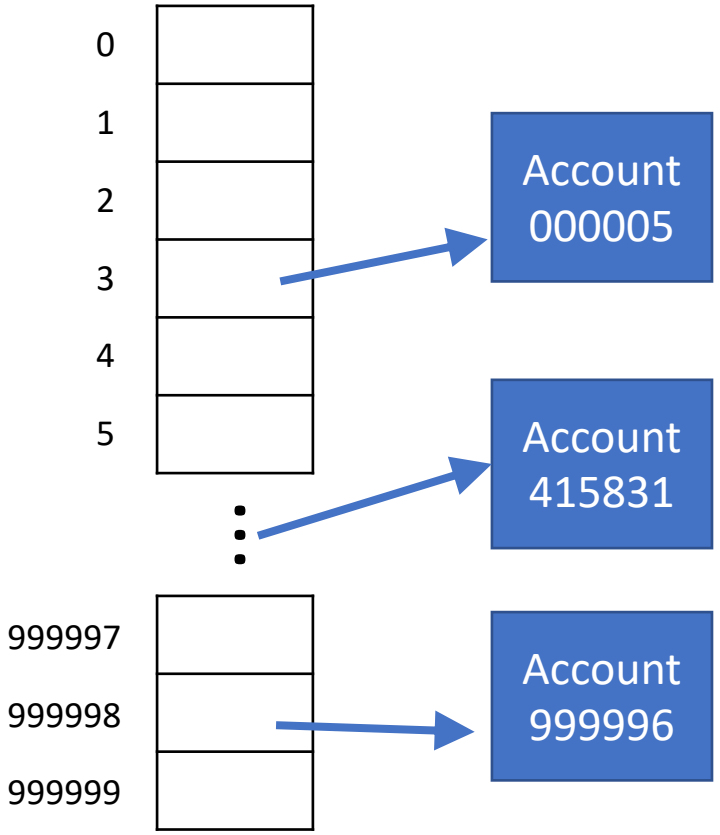
Class Notes: Introduction to Hashmaps (Programmer and Conceptual Views)

note: the top right of each slide will indicate whether the discussion is mostly for programmers or for conceptual understanding

Goal: improve the (currently linear) running-time of looking up accounts by their number

Let's assume that **each account has a 6-digit ID number**

Proposal: *store the accounts in an array*



Does this seem like a good data structure?

What if account numbers aren't consecutive?

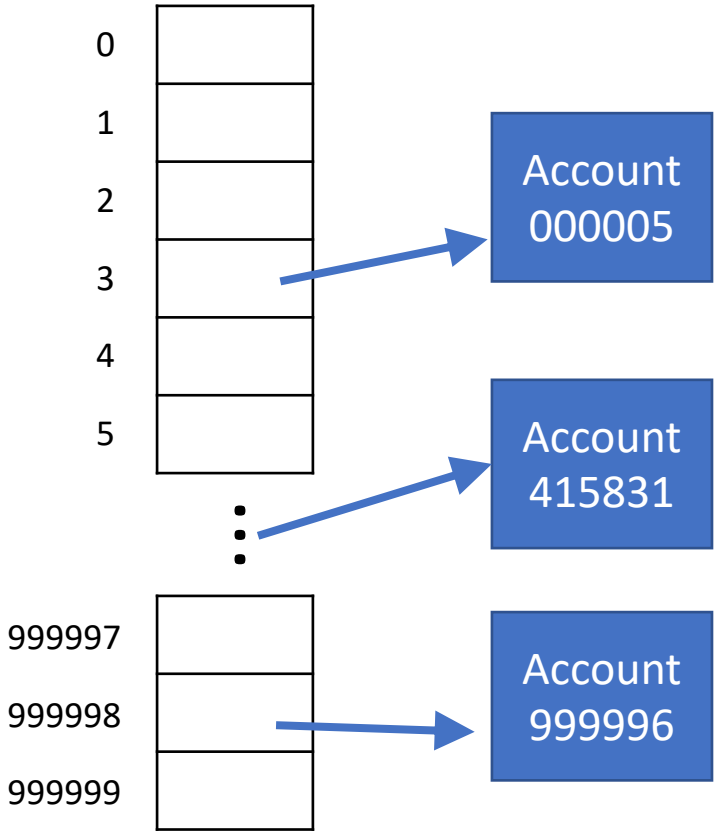
What if account numbers must start with non-0 digit?

If someone closes/deletes account, do we have to copy the remaining accounts into consecutive array slots?

Goal: improve the (currently linear) running-time of looking up accounts by their number

Let's assume that **each account has a 6-digit ID number**

Proposal: *store the accounts in an array*



Does this seem like a good data structure?

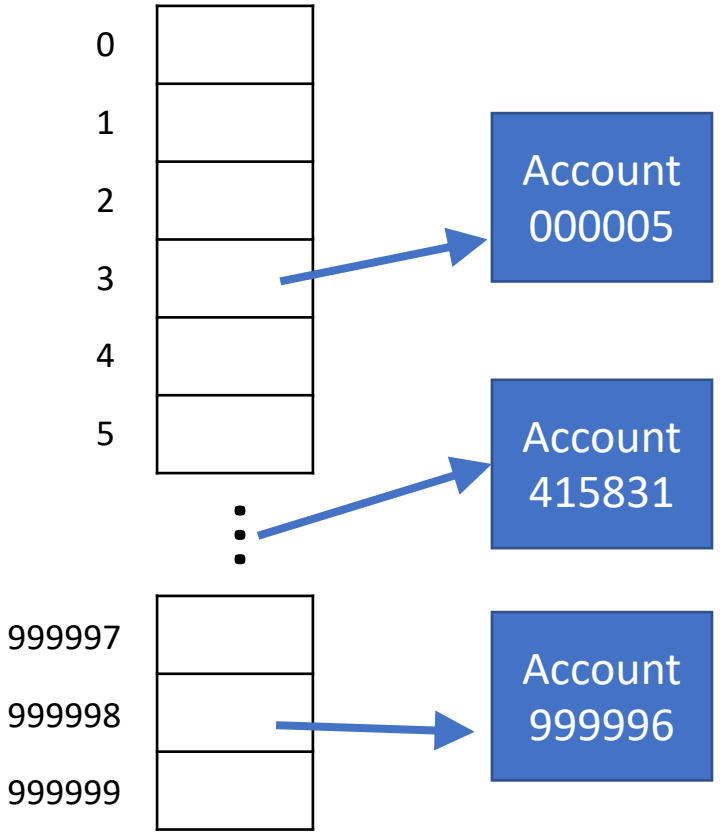
This offers constant-time lookup, but could waste space if we don't use all the account numbers. This is an example of trading off space for improved run-time (a common tradeoff in computing)

```
Account[] accounts = new Account[1000000];

Account findAccount(int withID) {
   return accounts[withID];
}
```

Goal: improve the (currently linear) running-time of looking up accounts by their number

Let's assume that **each account has a 6-digit ID number**

Proposal: *store the accounts in an array*



What if account numbers must start with non-0 digit?

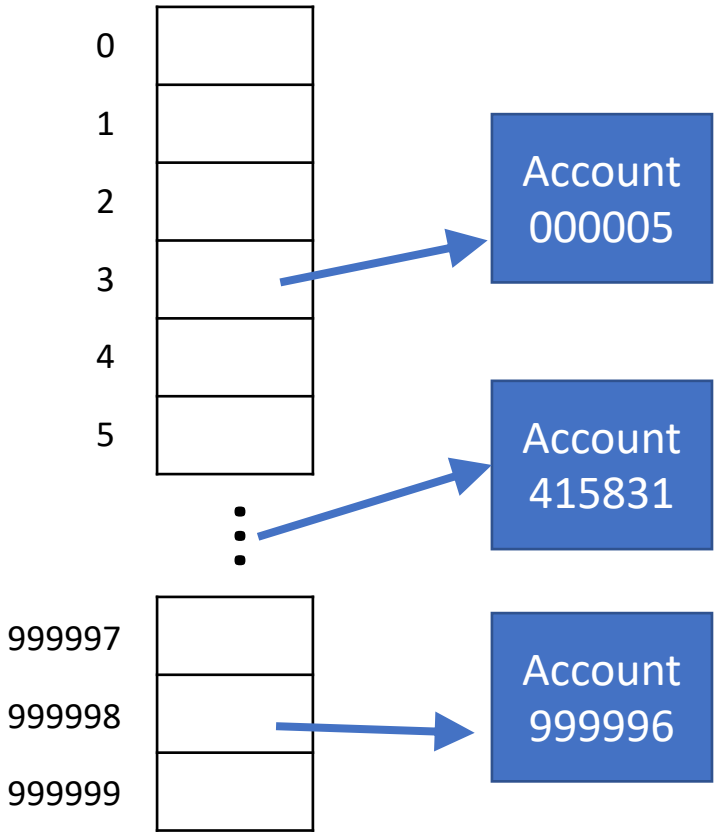
That's okay – we could allocate an array with 900,000 spaces and adjust the index which we look up in the array:

```
Account[] accounts = new Account[900000];

Account findAccount(int withID) {
   return accounts[withID - 100000];
}
```

Goal: improve the (currently linear) running-time of looking up accounts by their number

Let's assume that **each account has a 6-digit ID number**

Proposal: *store the accounts in an array*



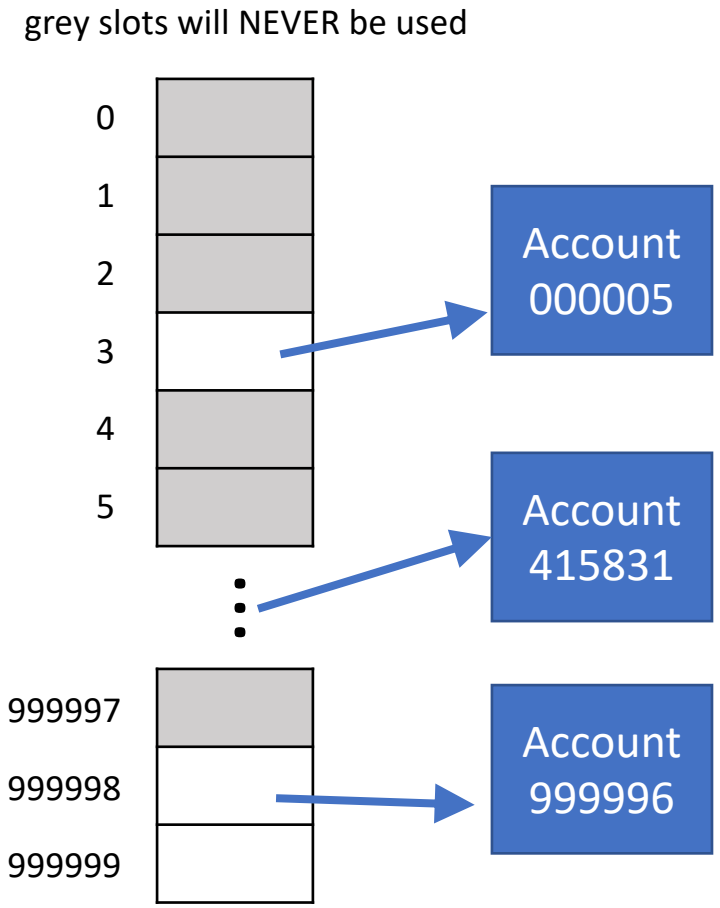
If someone closes/deletes account, do we have to copy the remaining accounts into consecutive array slots?

No! We only had to do copy-on-delete when we used arrays to implement lists (because we wanted to access list elements quickly by their position).

There is no need to use consecutive array slots when using arrays in general.

Goal: improve the (currently linear) running-time of looking up accounts by their number

Let's assume that each account has a 6-digit ID number, **but they aren't consecutive**



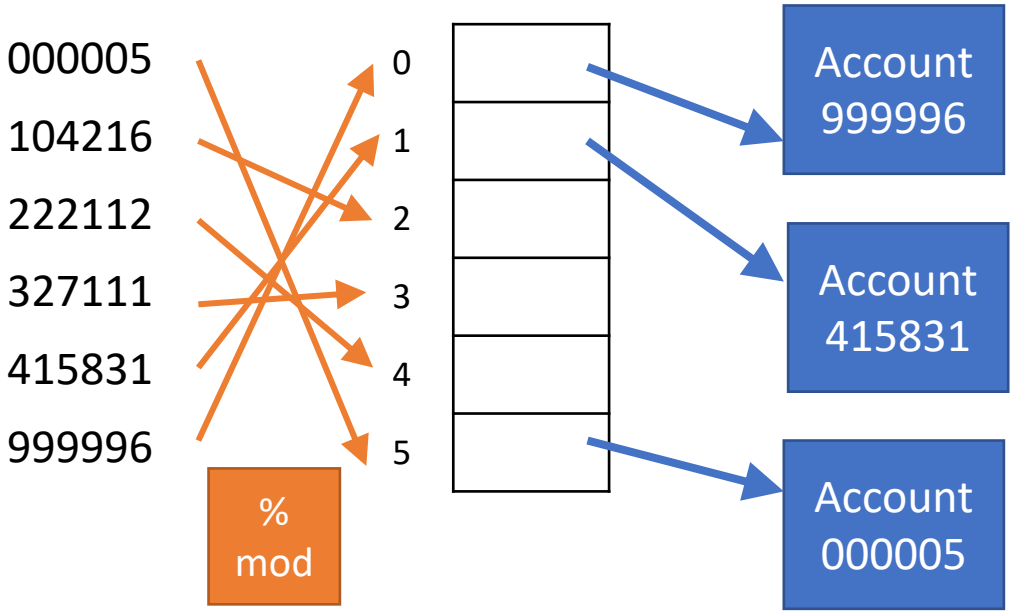
Does this still seem like a good data structure?

It would be nice to only have as many array slots as we actually need

Goal: how might we get constant-time lookup with fewer array slots than valid IDs?

Let's assume that each account has a 6-digit ID number, **but they aren't consecutive**

Let's use only an array as large as we need (for now, assume we need 6 slots)



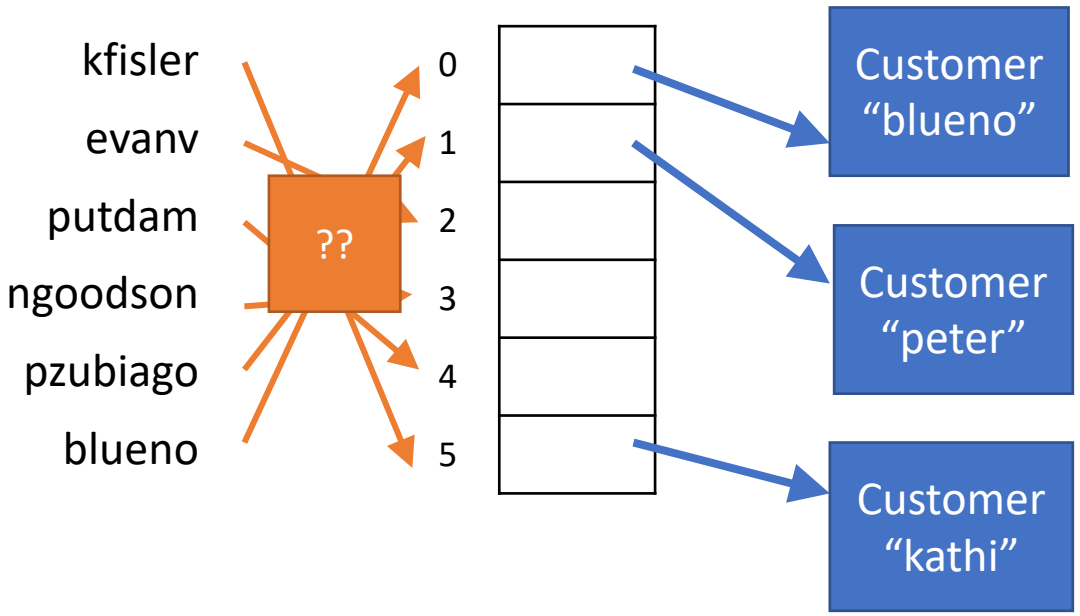
How might one map from the account IDs to the array indices?

Remember modulo (remainder under division)? That would take the account ID to an index in the range from 0 ... 5

```
Account[] accounts = new Account[6];

Account findAccount(int withID) {
   return accounts[withID % 6];
}
```

Question: But what if we wanted to map from usernames to Customers instead?

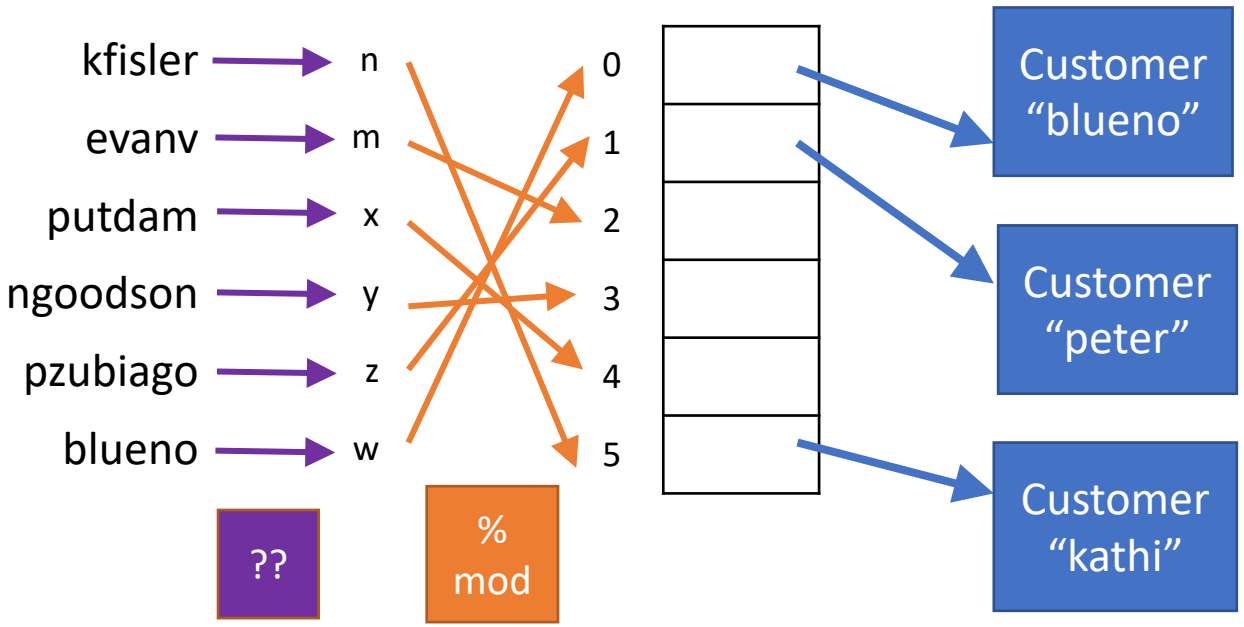
Our current mod-based strategy only works for numbers!



How might one map from the *usernames* to the array indices?

Question: But what if we wanted to map from usernames to Customers instead?

Our current mod-based strategy only works for numbers!



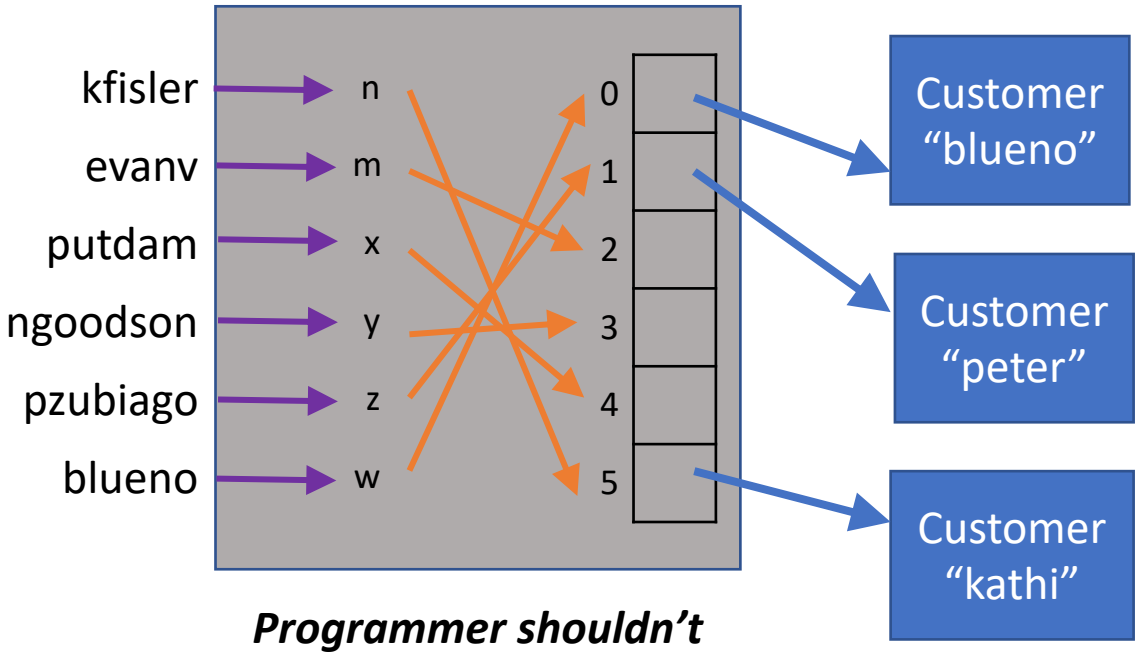
How might one map from the *usernames* to the array indices?

We would need some way to turn the username strings into numbers (the mystery purple function)

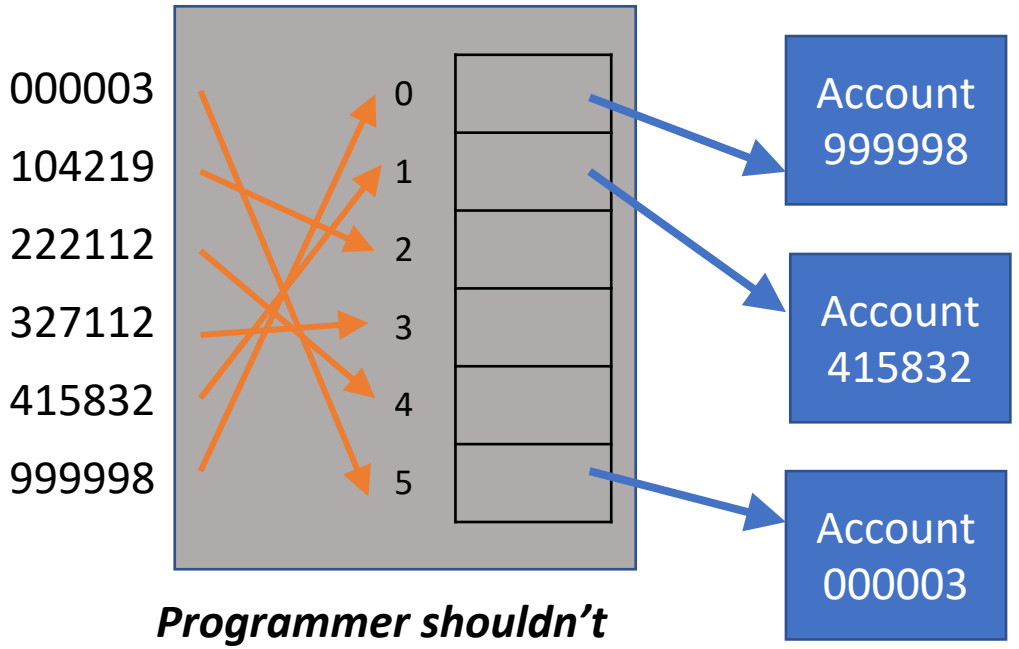
This is getting harder ...

Goal: Quickly map from any type to another (with arrays underneath)

This is sufficiently common that it is a built-in data structure



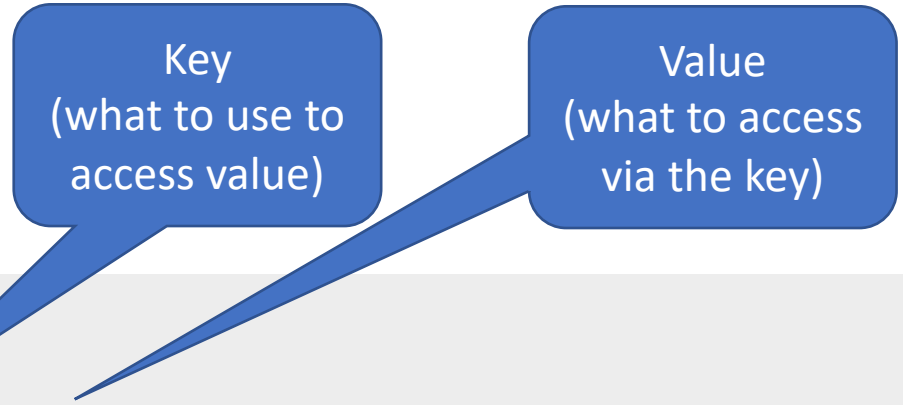
Programmer shouldn't see the gray part



Programmer shouldn't see the gray part

HashMap (Hashtable, Dictionary)

Hashmaps in Java




```
import java.util.HashMap;

private HashMap<Integer,Account> accounts = new HashMap<Integer, Account>();
private HashMap<String,Customer> customers = new HashMap<String, Customer>();

public void addCustomer(String username, String name, String pwd) {
    Customer c = new Customer(name, pwd);
    customers.put(username, c); // stores c under username in hashmap
}

Customer findCustomer(String username) {
   return customers.get(username); // retrieves value associated with username
}
```

Hashmaps in Java

What if we add a second value for the same key?

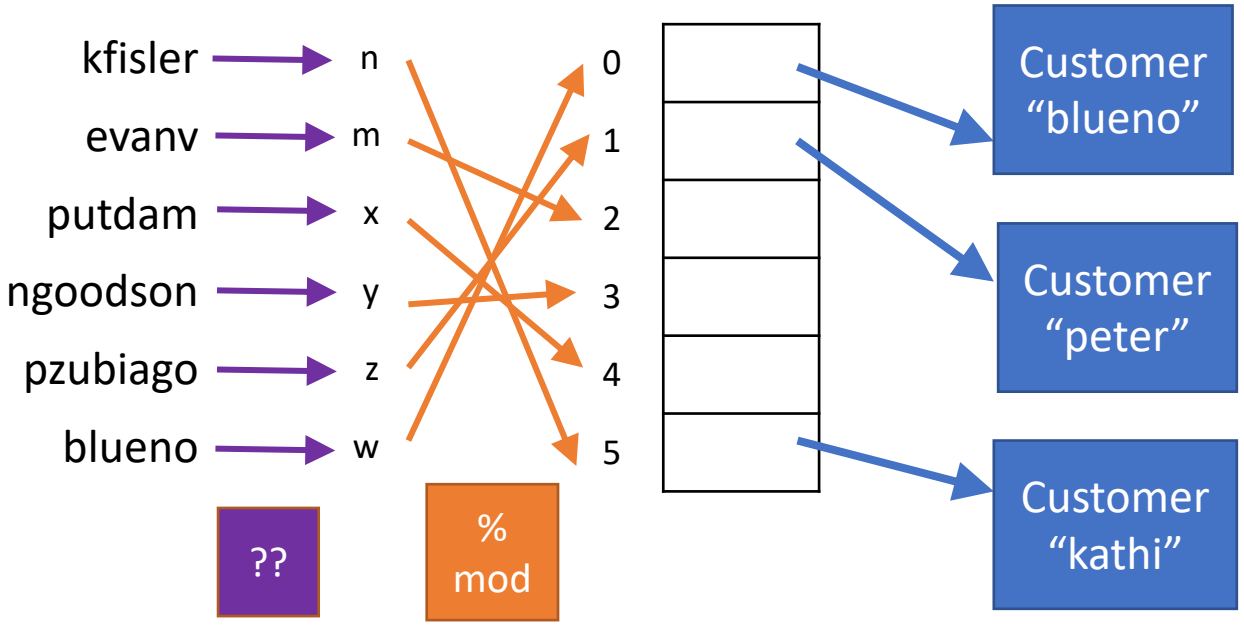
A hashmap allows only one value for each unique key.
If your application needs multiple, make your value type a list

```
import java.util.HashMap;

private HashMap<Integer,Account> accounts = new HashMap<Integer, Account>();
private HashMap<String,Customer> customers = new HashMap<String, Customer>();
private HashMap<String,LinkedList<Customer>) = ...
// map (non-unique) names to all customers with that name

public void addCustomer(String username, String name, String pwd) {
    Customer c = new Customer(name, pwd);
    if (customers.get(username) == null) // no value with this key
        customers.put(username, c); // stores a value under a key in hashmap
    else
        throw RuntimeException("duplicated username");
}
```

Question: Could we map each Customer to a list of their Accounts?

```
HashMap<Customer,LinkedList<Account>> custAccts = new HashMap<Customer,...>();
```

Let's use this to explore those purple arrows



The purple arrows represent a function from the KEY type to int

Java built-in classes have a method called *hashCode* for this

To use a class you defined as a key, you must provide this method as well

```
class Customer {
  public int hashCode() {
      return this.name.hashCode();
  }
} // we'll say more about this Friday
```

Key Takeaways

Programmer hat

- Hashmaps (a.k.a. dictionaries in Python) are a built-in data structure that provide constant-time access of values from keys
- There can be at most one value per unique key, but the values can be arbitrarily complex
- If you use your own class as a key, write a hashCode method for the class

Conceptual hat

- Under the hood, hashmaps are built on arrays
- Under the hood, some function maps keys (perhaps non-numeric) to integers, which are then mapped to array indices via modulo