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

Recall our current findCustomer/findAccount methods

```
private Account findAccount(int forAcctNum) throws AcctNotFoundExn {
    for (Account acct:accounts) {
        if (acct.numMatches(forAcctNum))
            return acct;
    }
    throw new AcctNotFoundExn(forAcctNum);
}

private Customer findCustomer(String custname) throws CustNotFoundExn {
    for (Customer cust:customers) {
        if (cust.nameMatches(custname)) {
            return cust;
        }
    }
    throw new CustNotFoundExn(custname);
}
```

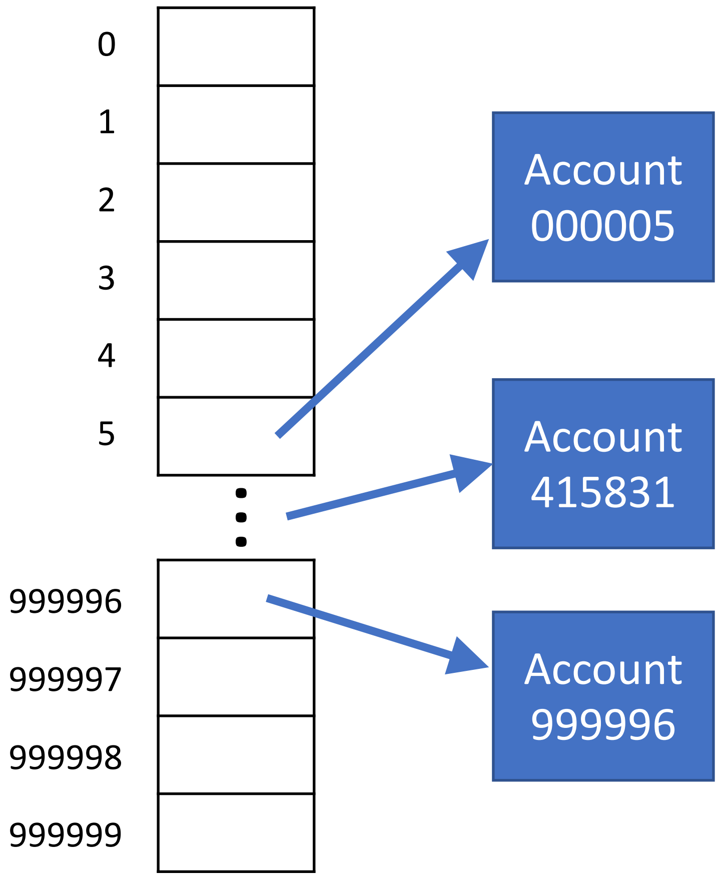
Can we do better than linear time?

Programmer Hat

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*



```
private Account findAccount(int forAcctNum)
  throws AcctNotFoundExn {
    for (Account acct:accounts) {
        if (acct.numMatches(forAcctNum))
            return acct;
    }
    throw new AcctNotFoundExn(forAcctNum);
}
```

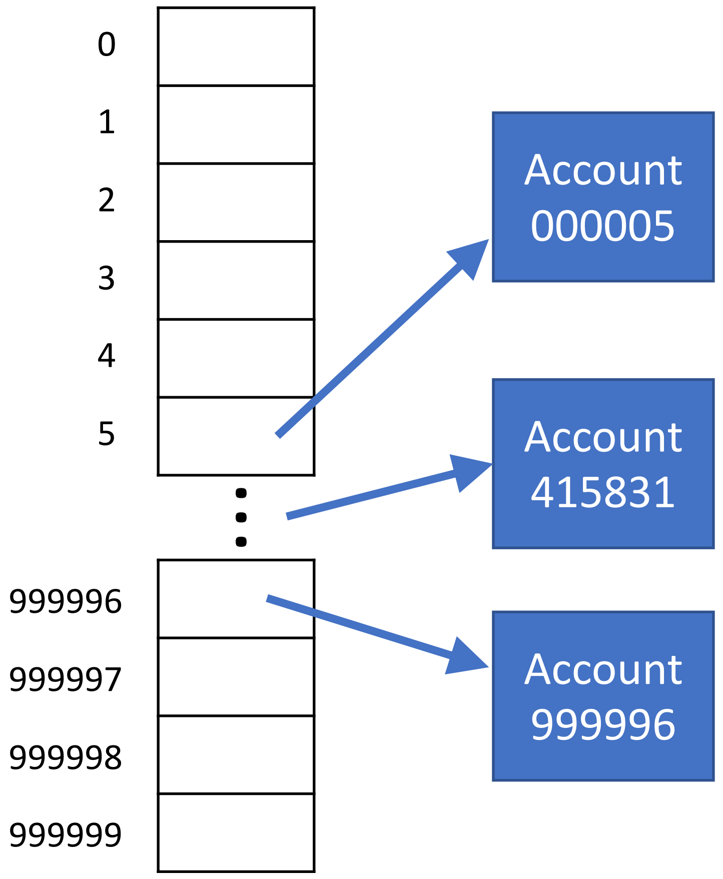
```
Account[] accounts = new Account[1000000];

private Account findAccount(int forAcctNum) {
   if (forAcctNum >= 0) && (forAcctNum < 1000000)
        && (accounts[forAcctNum] != null) {
      return accounts[forAcctNum];
   } else {
     throw new AcctNotFoundExn(forAcctNum);
   }
}
```

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?

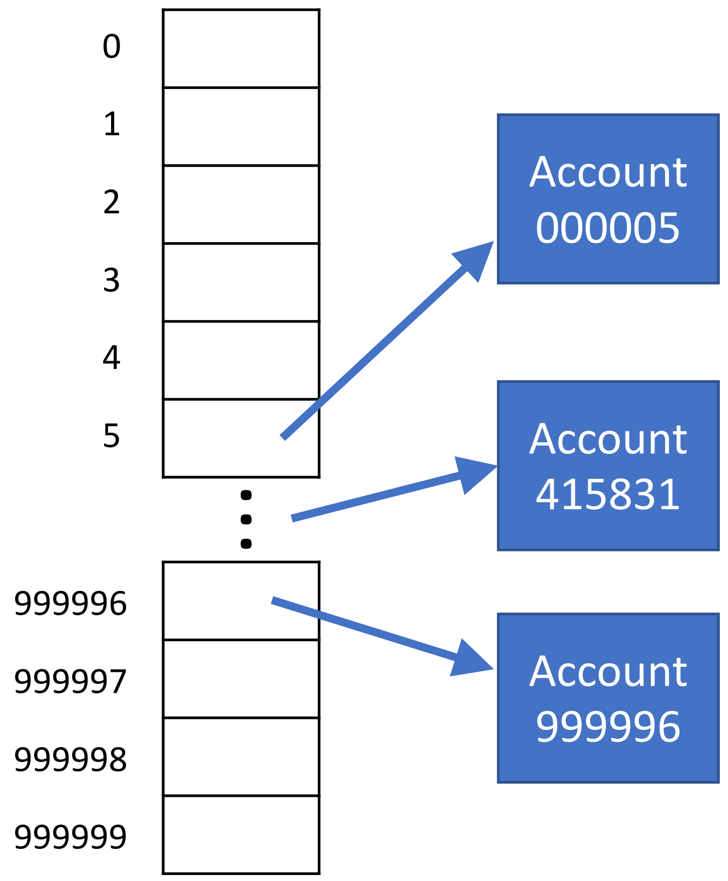
Yes, we get constant-time lookup, but could waste space if we don’t use all of the account numbers. This is an example of trading space for run-time (got fast by potentially having unused array slots).

Programmer Hat

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*



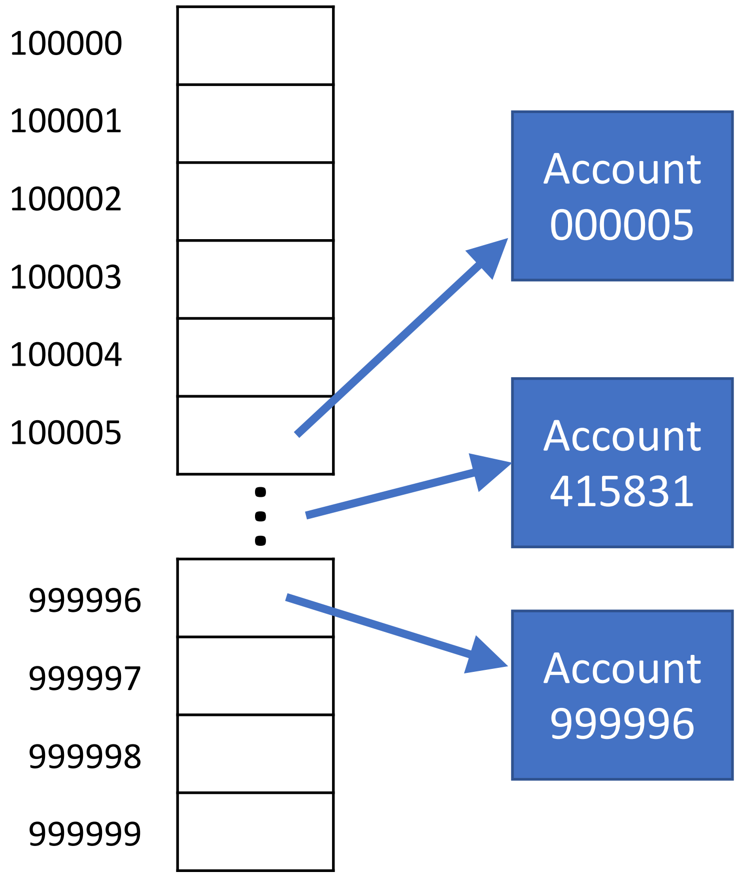
If someone closes/deletes account, do we have to shift any of the accounts around?

When you work with arrays, think about what the indices represent. If they represent positions (as in ArrayList), need to shift the elements into consecutive slots. If they represent data that we use to access items (as in Accounts), don’t want to shift.

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, but adjust the index on lookup. The index computation is still constant time.

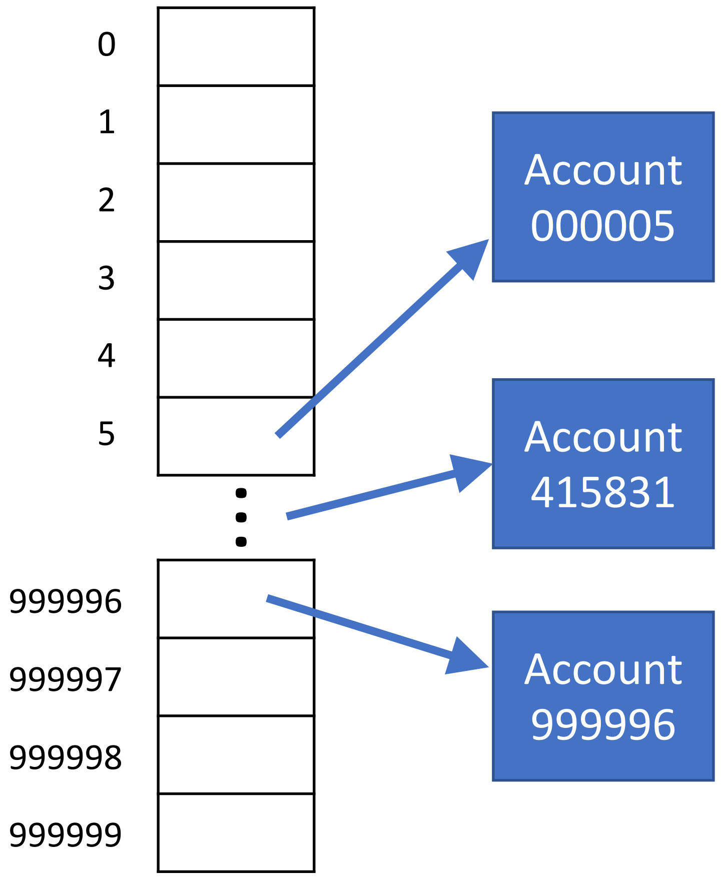
```
Account[] accounts = new Account[900000];

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

Programmer Hat

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**



Upshot: this proposal seems like a good idea.

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

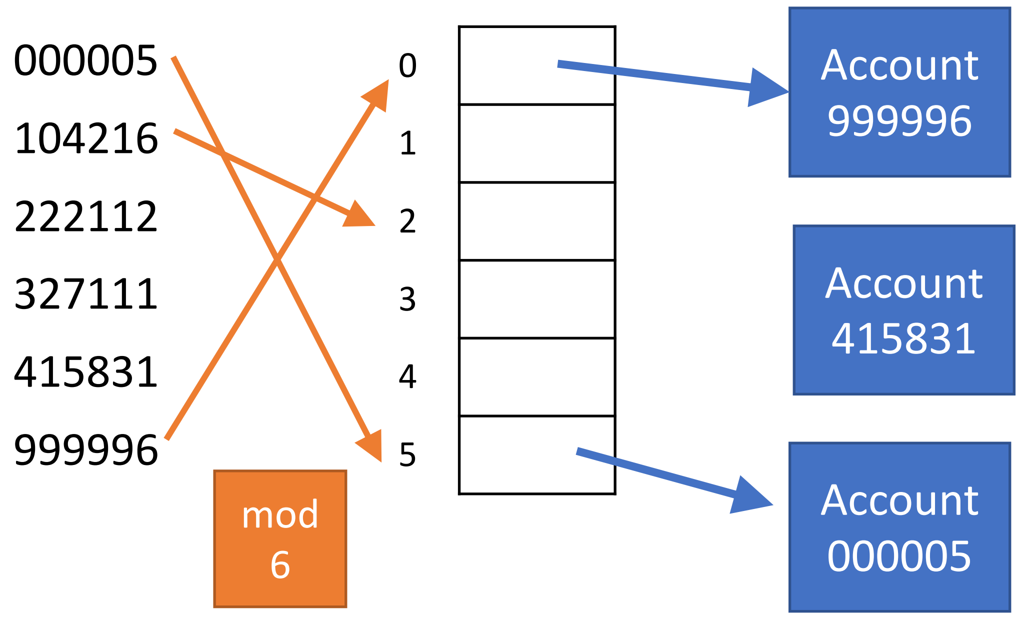
Could we reduce the number of array slots but still <u>predictably compute</u> where an object is located if we have fewer array slots?

Programmer Hat

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 (reminder under division)? That could map each account ID to an index in the range 0 .. 5

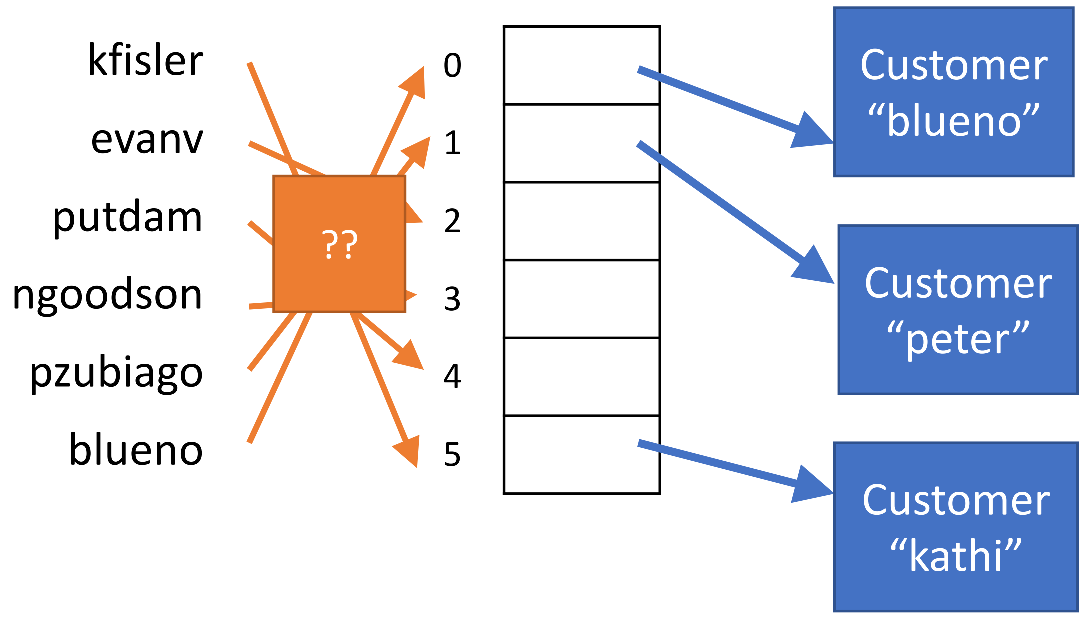
```
Account[] accounts = new Account[6];

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

(yes, multiple accounts could land in same slot. We'll return to that next time)

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

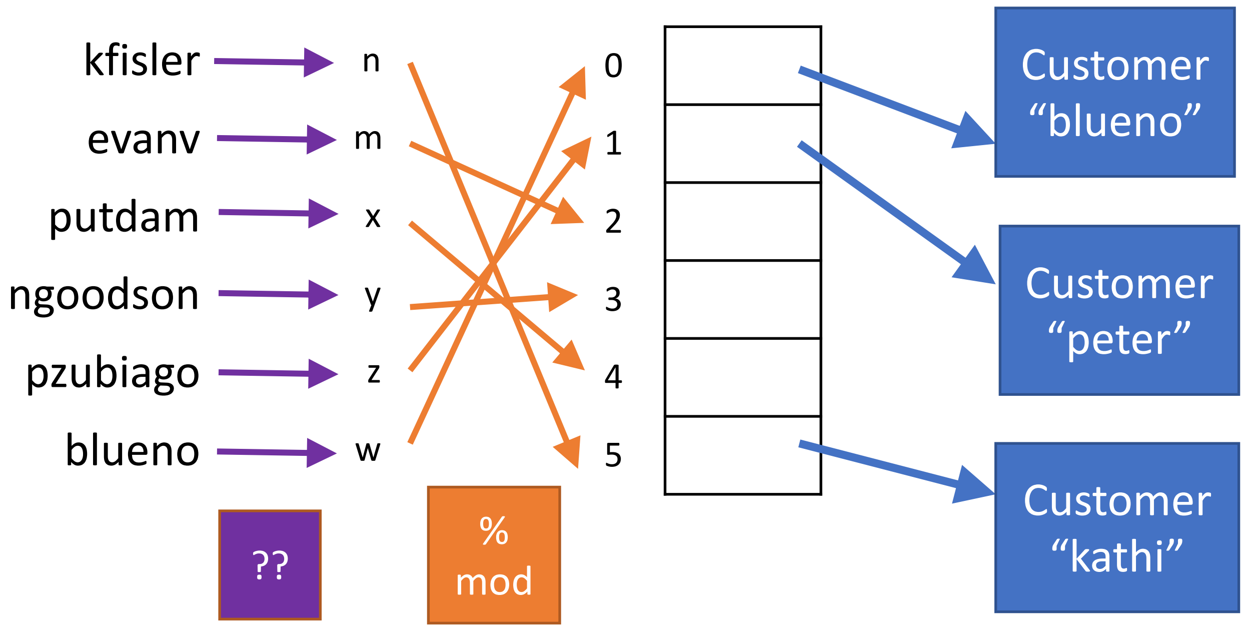
Our current mod-based strategy only works for numbers!



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!



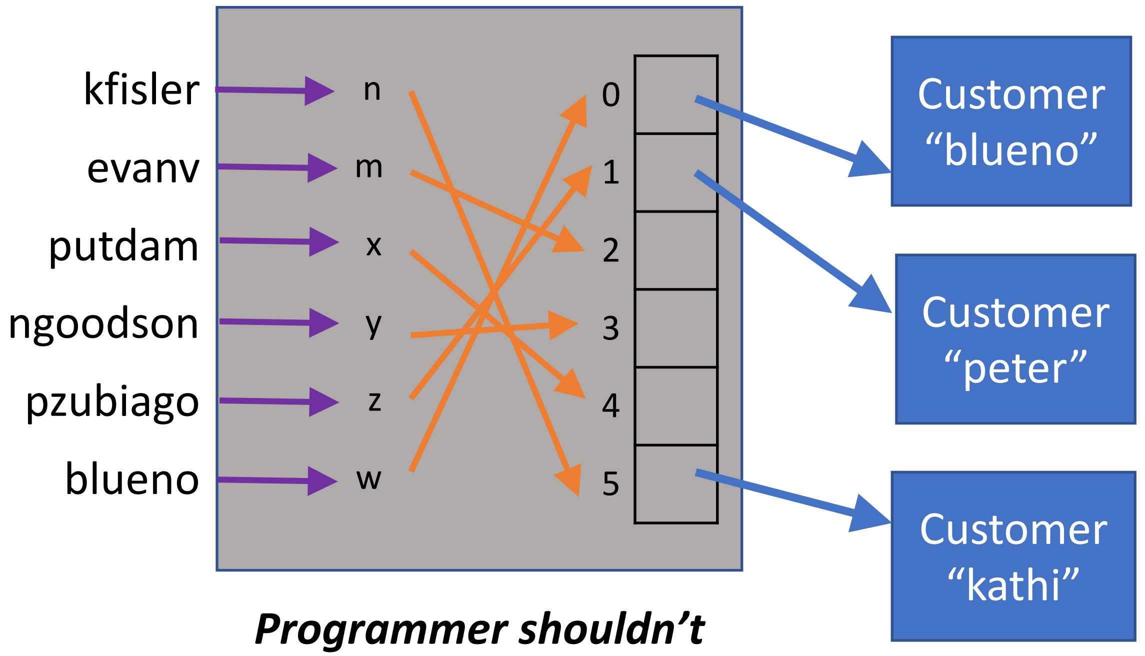
How might one map from the *usernames* to the array indices?

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

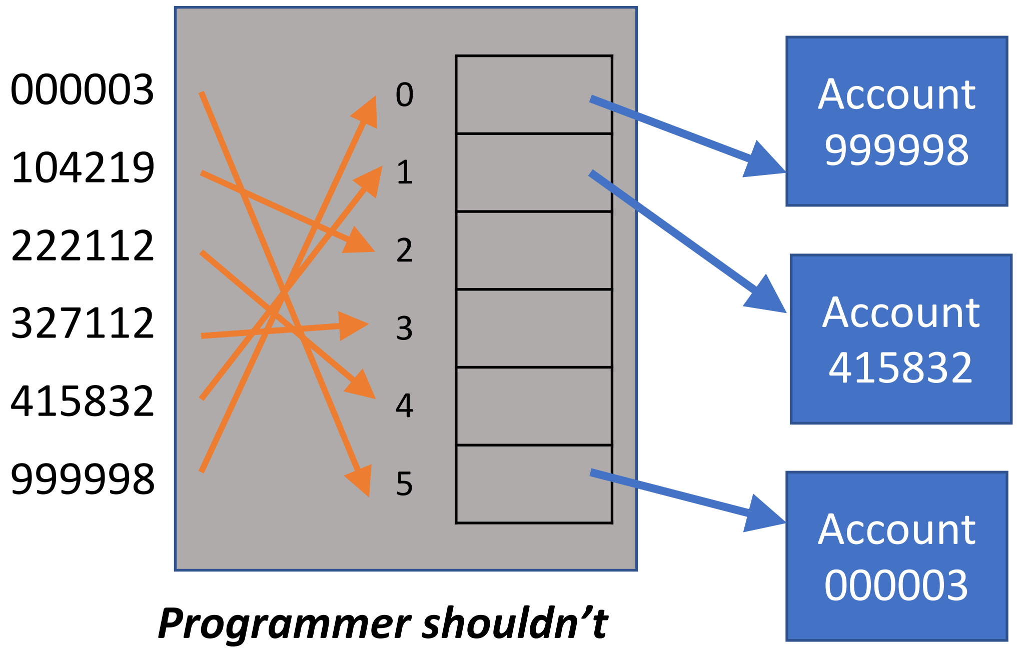
This is getting harder ...

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)

Programmer Hat

Hashmaps in Java

Key
(what to use to
access value)

Value
(what to access
via the key)

```
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");
}
```

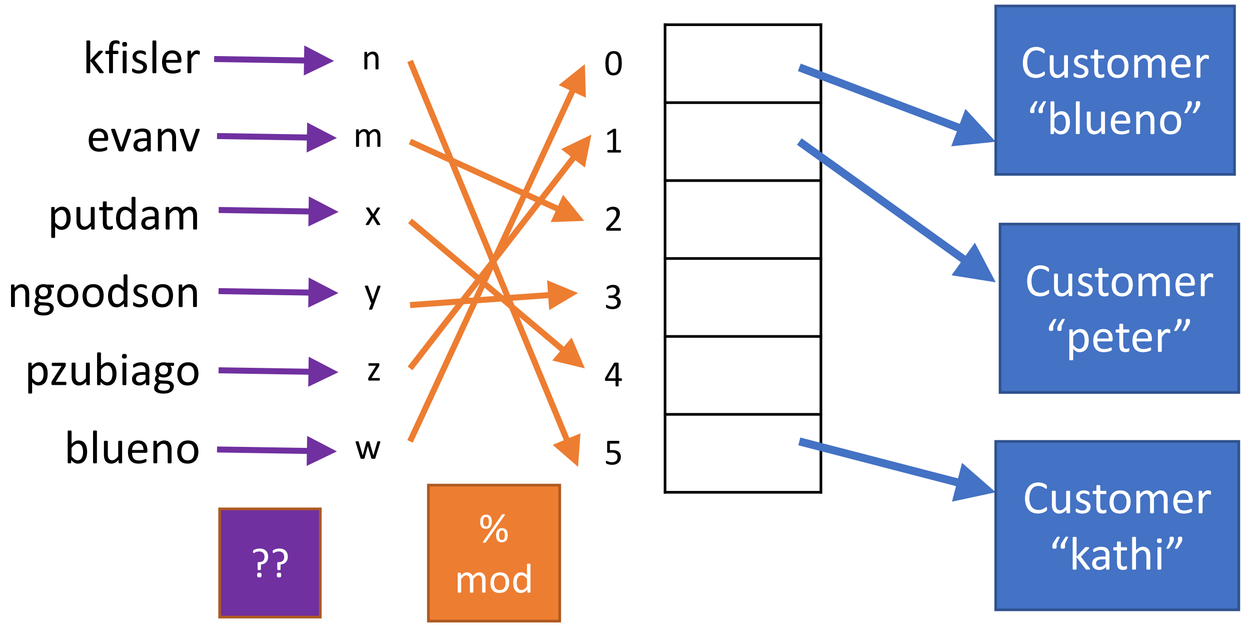
(we didn't cover this slide in class, kept here fyi)

Programmer Hat

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 next time
```

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 (so you can store a list of items under a key, for example)
- 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