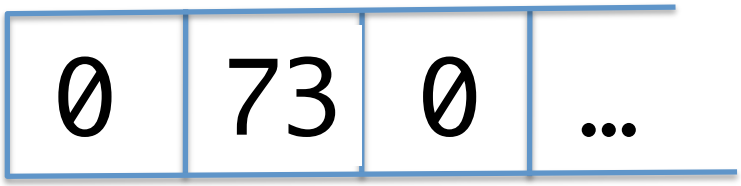
Declaring an array

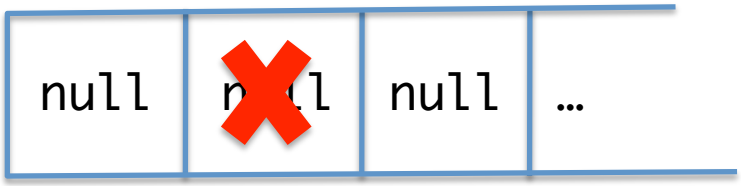
using int primitive data type

```
int [] score = new int[10];
score[1] = 73;
```



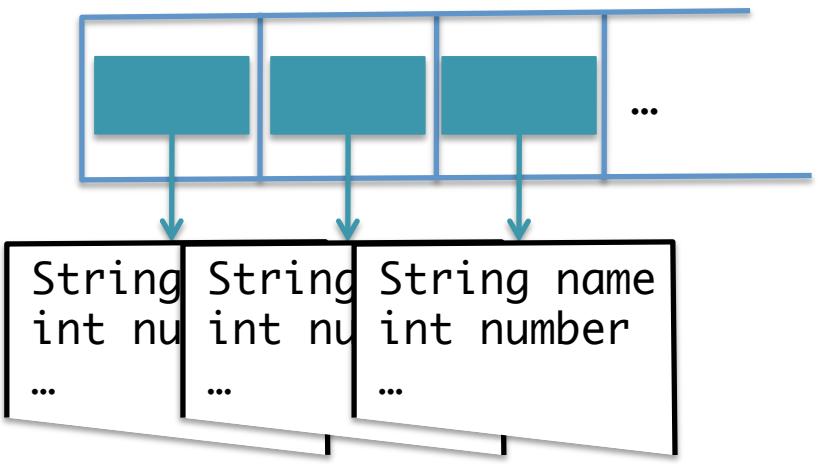
using Account objects

```
Account [] customer = new Account[10];
customer[1].setNumber(459302);
```



Declaring an array of objects

```
Account [] customer = new Account[10];

for (int i = 0; i<customer.length; i++){
   customer[i] = new Account();
}
```



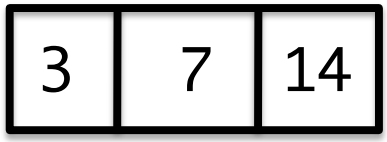
Declaring an array of objects

```
Account [] customer = new Account[10];

for (int i = 0; i<customer.length; i++){
   customer[i] = new Account();
}
```

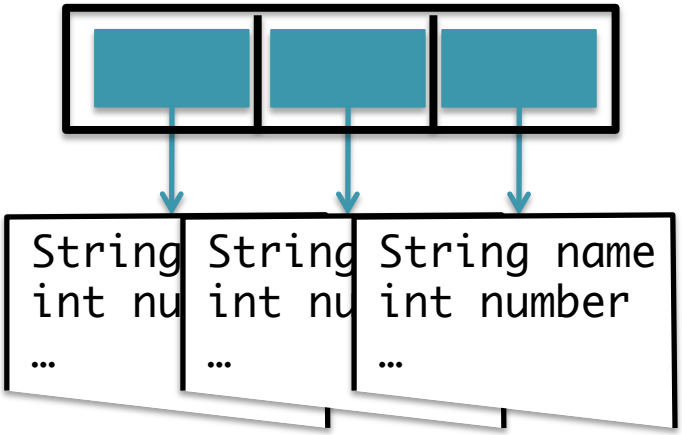

Use a constructor to create the new objects, one at a time

Declaring an array of objects

```
int [] score = {3, 7, 14};
```

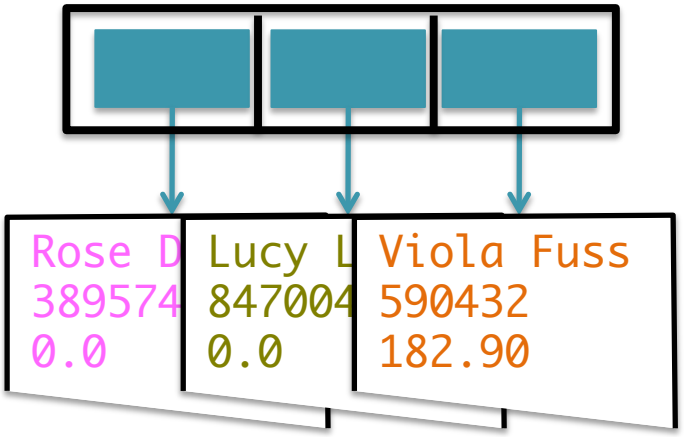



```
Account[] customer = {new Account(),
                      new Account(), new Account()};
```



Declaring an array of objects

```
Account[] customer = {new Account("Rose Dior", 389574),
                      new Account("Lucy Lastic", 847004),
                      new Account("Viola Fuss", 590432, 182.90)};
```



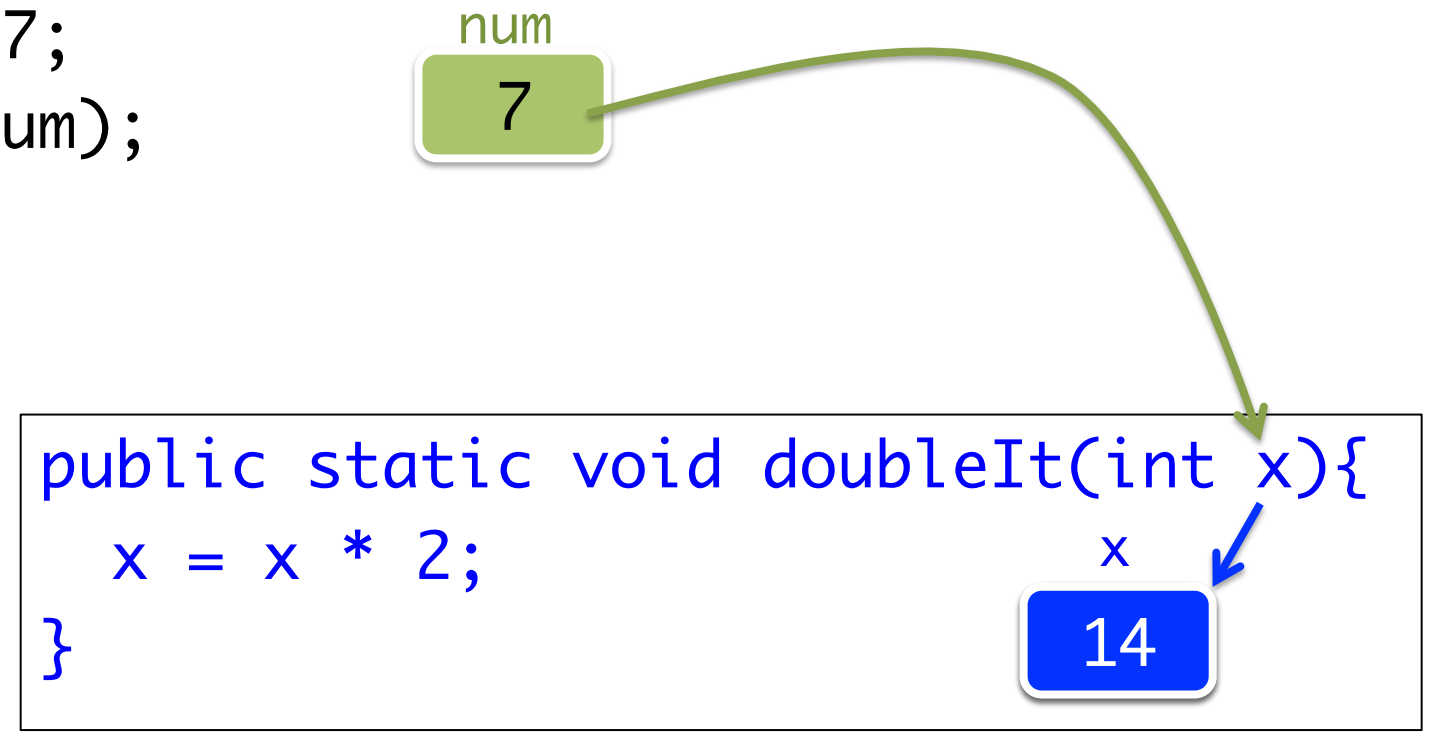
Calling an instance method

```
customer[1].setNumber(459302);

if(customer[i].equals(customer[i+1]))…

out.println(customer[found]);
```

Passing objects as parameters

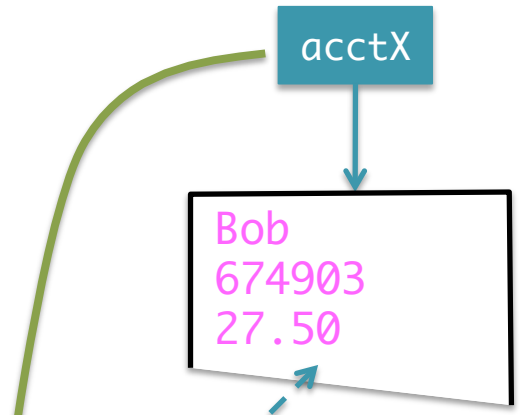
Passing primitive data types

```
int num = 7;
doubleIt(num);
…
```



Passing objects as parameters

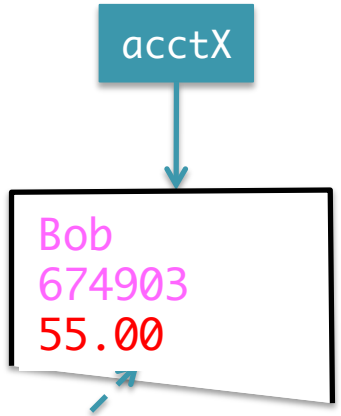
```
Account acctX = new Account("Bob", 674903, 27.50);
doubleIt(acctX);
…
```




```
public static void doubleIt(Account x){
   x.deposit(x.getBalance()*2);
}
```

Passing objects as parameters

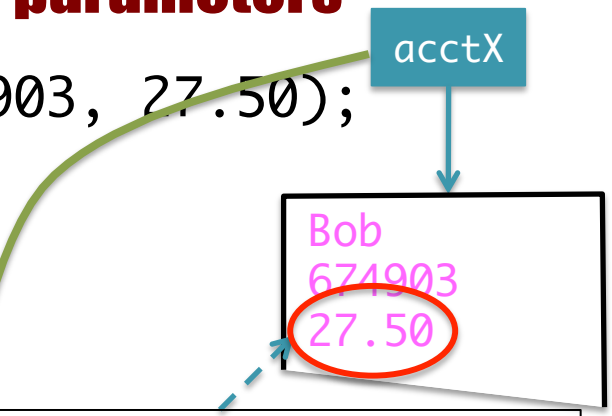
```
Account acctX = new Account("Bob", 674903, 27.50);
doubleIt(acctX);
…
```



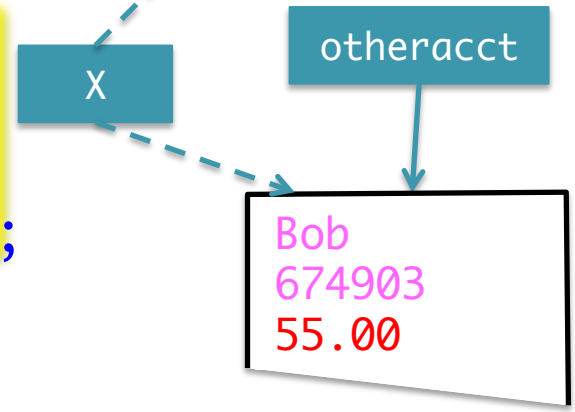

```
public static void doubleIt(Account x){
   x.deposit(x.getBalance()*2);
}
```

A common error when passing objects as parameters

```
Account acctX = new Account("Bob", 674903, 27.50);
doubleIt(acctX);
…
```

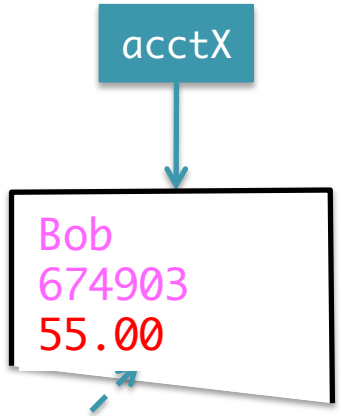



```
public static void doubleIt(Account x){
   Account otherAcct = new Account();
   otherAcct.setName(x.getName());
   otherAcct.setNumber(x.getNumber());
   otherAcct.deposit(x.getBalance()*2);
   x = otherAccount;
}
```



Passing objects as parameters

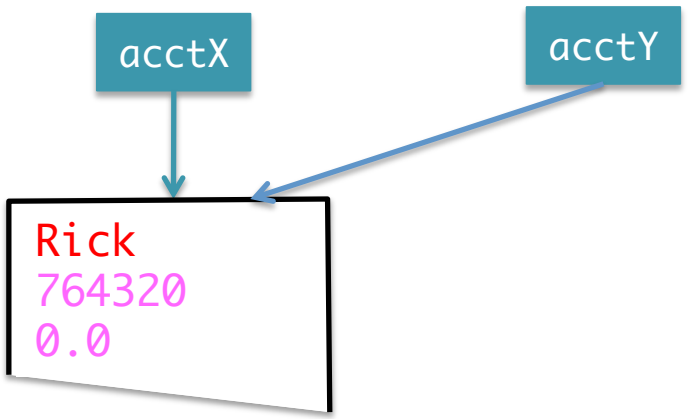
```
Account acctX = new Account("Bob", 674903, 27.50);
doubleIt(acctX);
…
```




```
public static void doubleIt(Account x){
   x.deposit(x.getBalance()*2);
}
```

Making one object equal to another

```
Account acctX = new Account("Mary", 764320);
Account acctY = acctX;
acctY.setName("Rick");
```

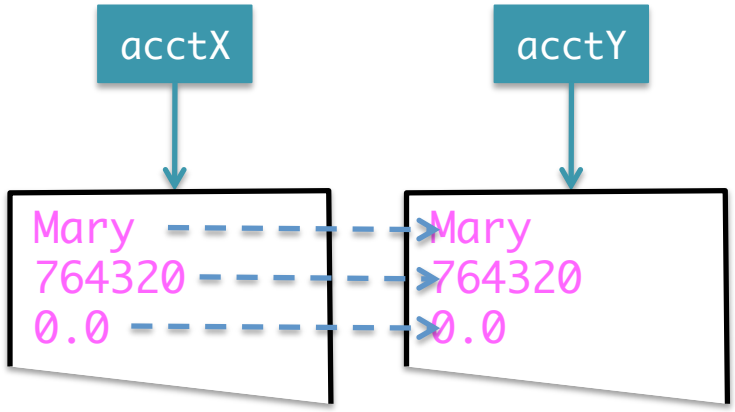


Making a copy of an object

```
public void makeCopyOf(Account original){
   this.setName = original.name;
   this.setNumber = original.number;
   this.deposit(original.getBalance());
}
```

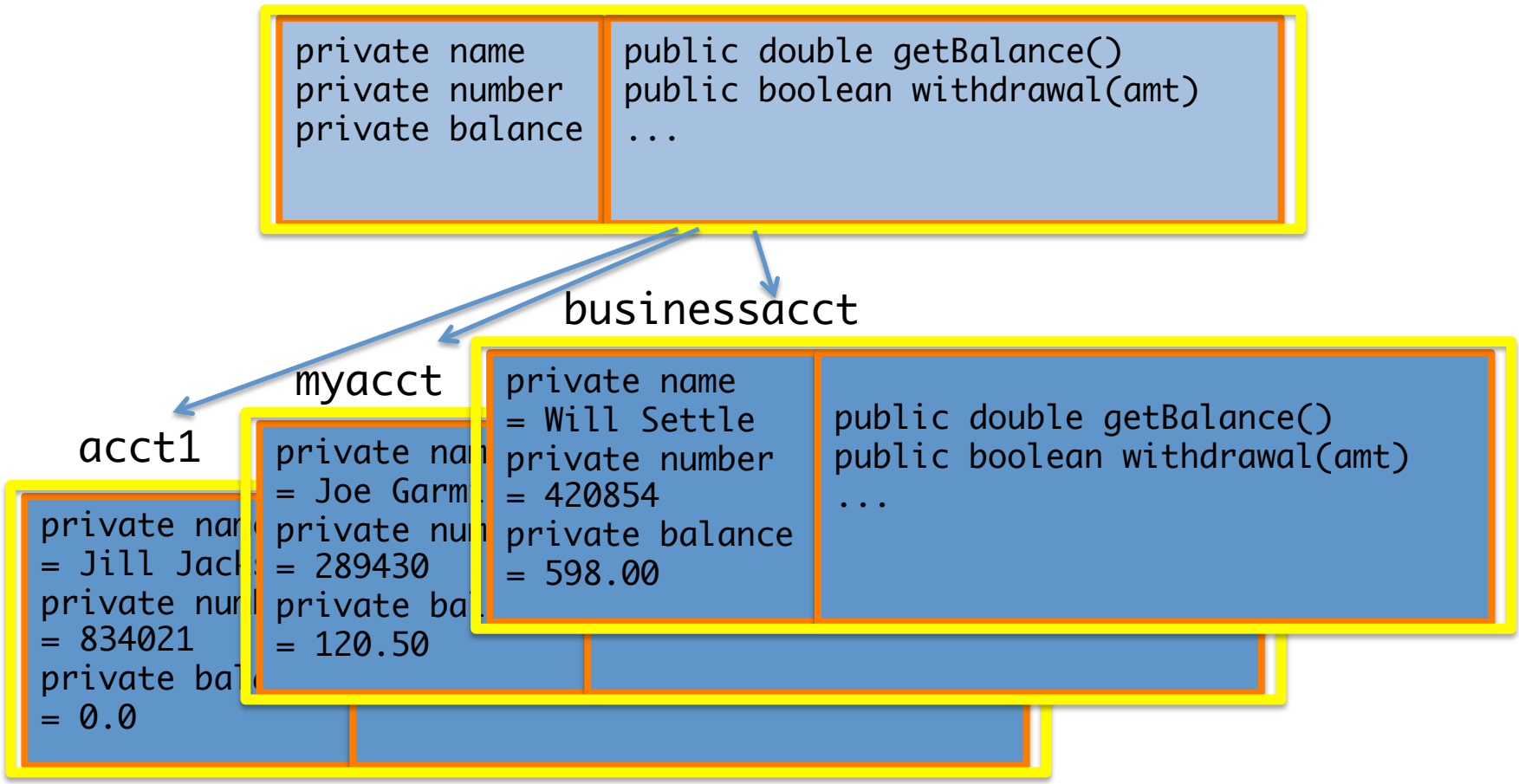
In a client program:

```
Account acctY = new Account();
acctY.makeCopyOf(acctX);
```

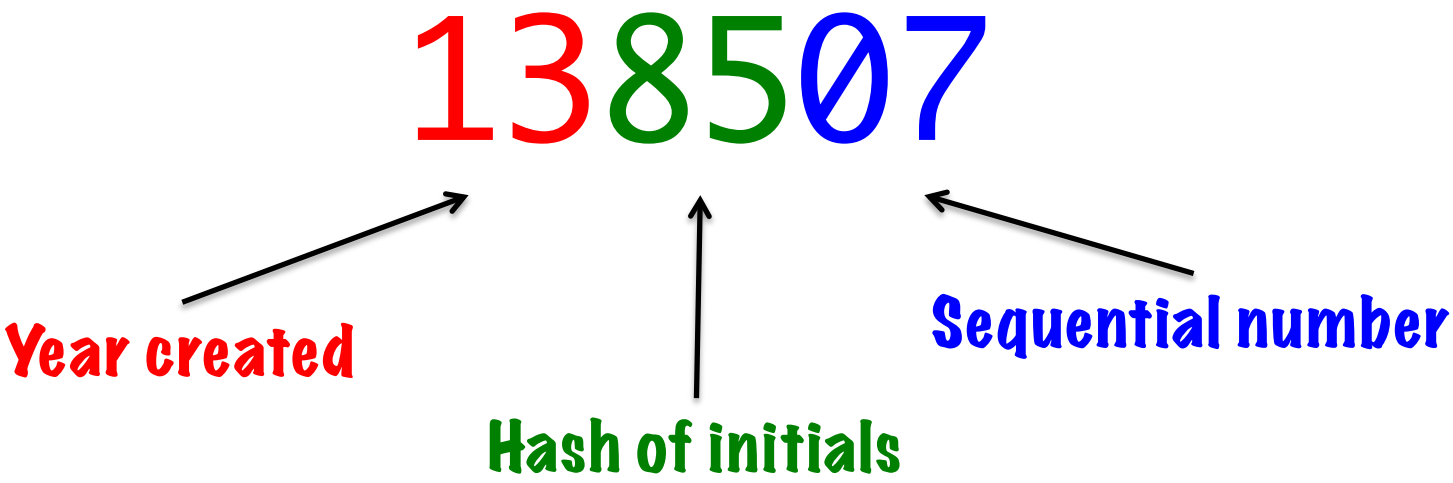


What happened to *static*?

```
public static void calcTotal(num1, num2){
...

public void getTotal(){
...
```


private name
private number
private balance
public double getBalance()
public boolean withdrawal(amt)
...
businessacct
myacct
acct1
private name
= Will Settle
private number
= 420854
private balance
= 598.00
public double getBalance()
public boolean withdrawal(amt)
...
= Joe Garm
= 289430
= 120.50
= Jill Jack
= 834021
= 0.0

Account Number

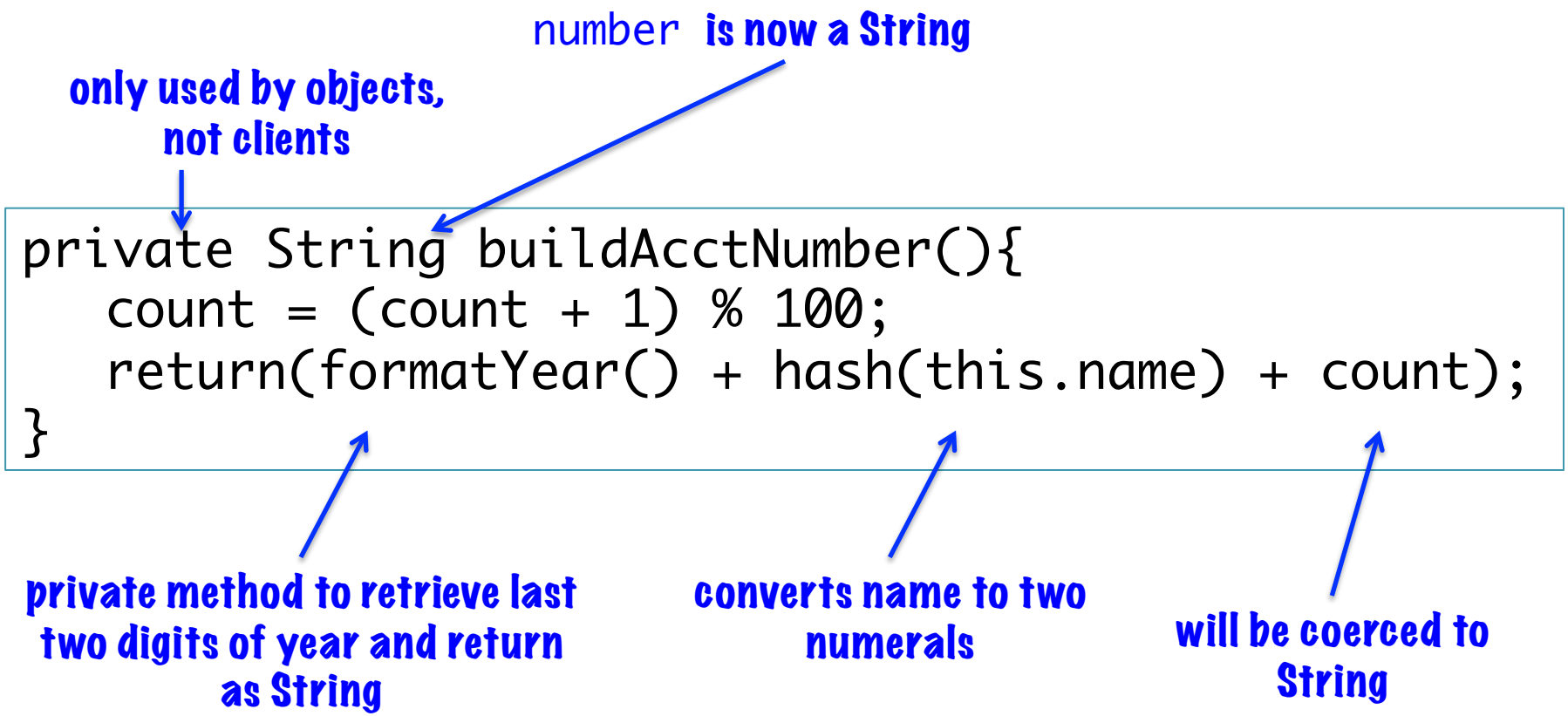


Building an account number

- Better if number is a String
- Thanks to abstraction, no problem to our client programs

```
private String buildAcctNumber(){
   count = (count + 1) % 100;
   return(formatYear() + hash(this.name) + count);
}
```

Building an account number



Constructors in the account class

```
public Account(){

}

public Account(String newName, int newNumber){
    name = newName;
    number = newNumber;
}

public Account(String newName, int newNumber, double initialBalance)
{
    this(newName, newNumber);
    balance = initialBalance;
}
```

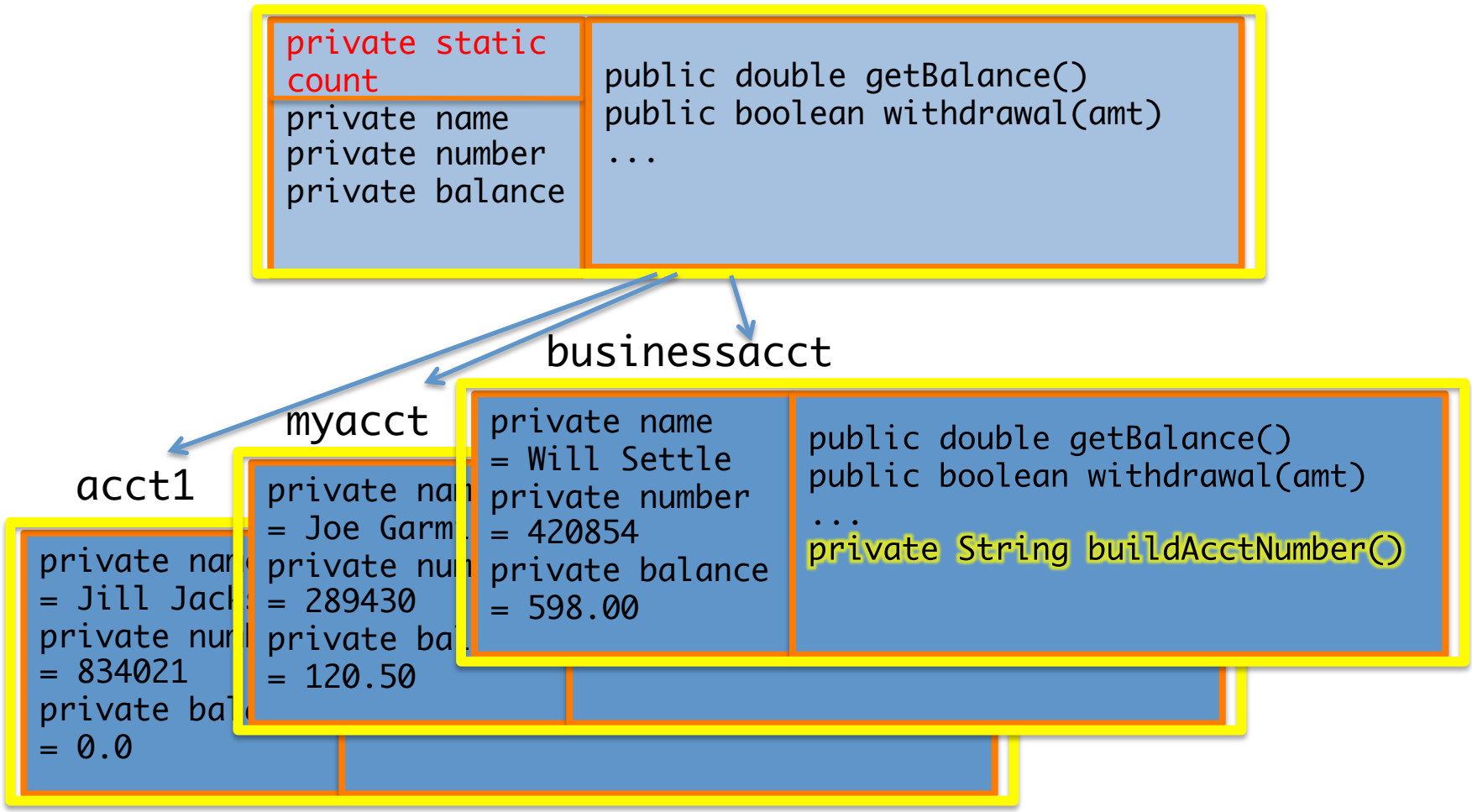
Constructors in the account class

```
public Account(String newName){
    name = newName;
    number = buildAcctNumber();
}

public Account(String newName, double initialBalance){
    this(newName);
    balance = initialBalance;
}
```

delete `setNumber()` **method**

count variable

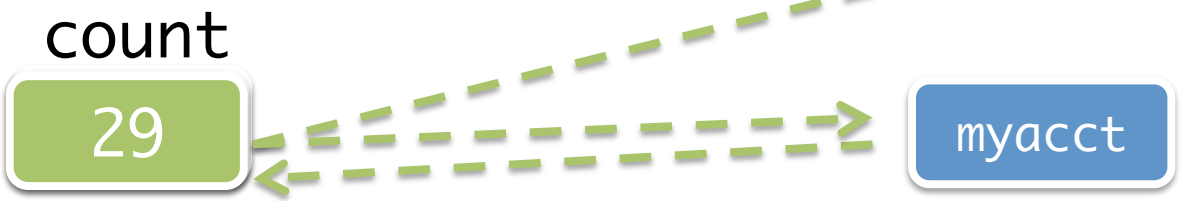
- How can we keep track of the next number?
- Every object should see the *same* count
- Access and increment with the construction of each object
- `private`

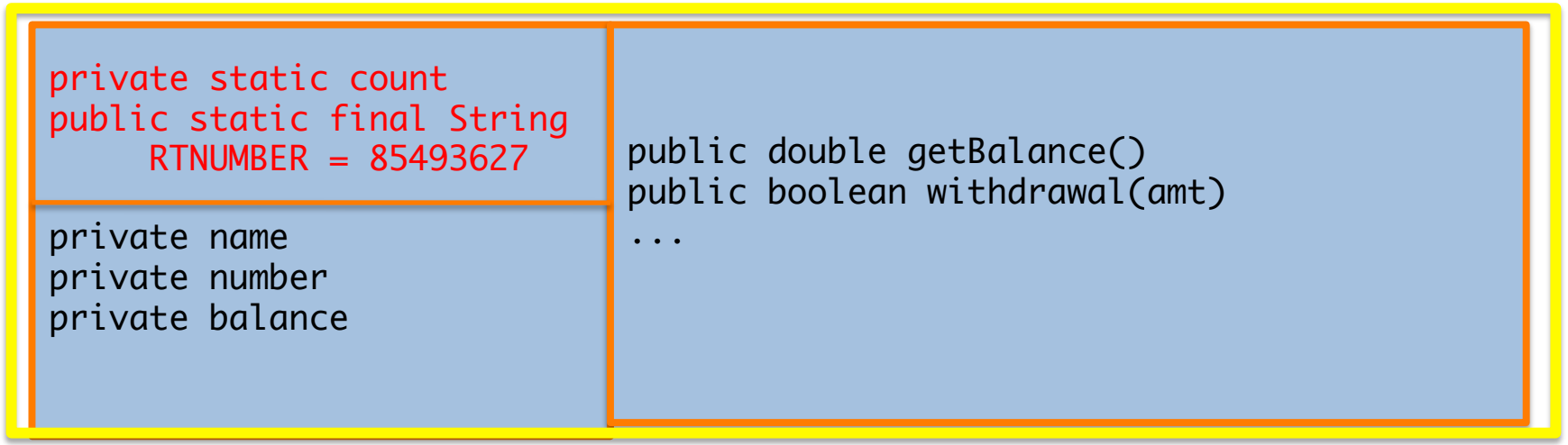

private static count
private name
private number
private balance
public double getBalance()
public boolean withdrawal(amt)
...
businessacct
myacct
acct1
private name
= Will Settle
private number
= 420854
private balance
= 598.00
public double getBalance()
public boolean withdrawal(amt)
...
private String buildAcctNumber()
= Joe Garm
= 289430
= 120.50
= Jill Jack
= 834021
= 0.0

Building an account number

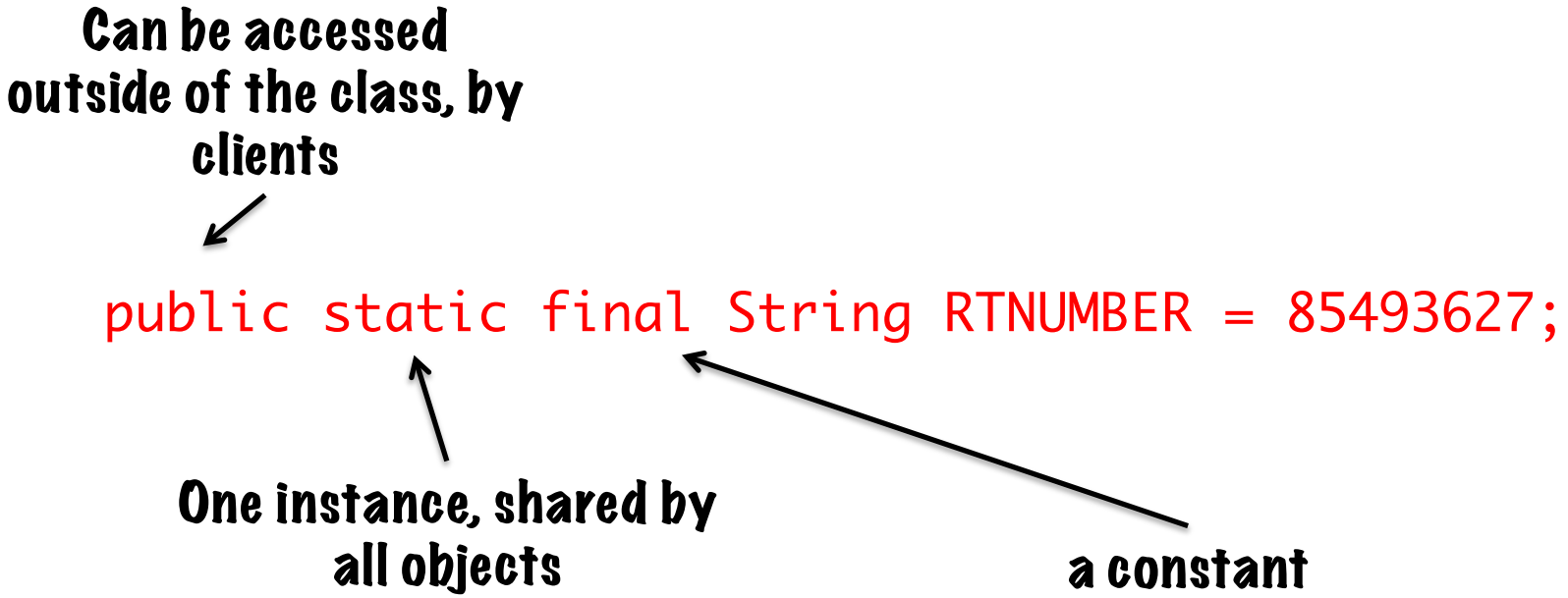
all objects access the same count variable, thus keeping a running count of the total number of objects created thus far

```
private String buildAcctNumber(){
    count = (count + 1) % 100;
    return(formatYear() + hash(this.name) + count);
}
```




private static count
public static final String
RTNUMBER = 85493627
private name
private number
private balance
public double getBalance()
public boolean withdrawal(amt)
...

Static class constant defined in the class file



Static class constant

Accessed by a client -

```
out.println(Account.RTNUMBER);
```

Accessed by an object in the class -

```
return(RTNUMBER + this.number);
```

Static methods in a class

- method definition resides in class, not copied to each object
- a method that may be called even if no objects have been created yet

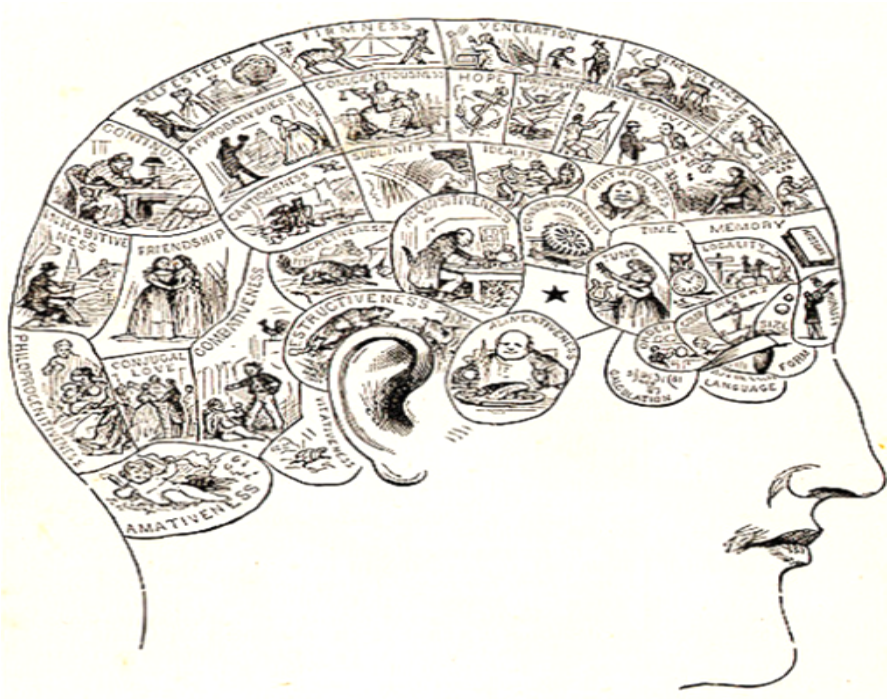
```
out.println(Account.getCurrentRate());
```

```
asset = Account.totalDeposit(savings, check1,
                             check2);
```

logical that one object might make the call

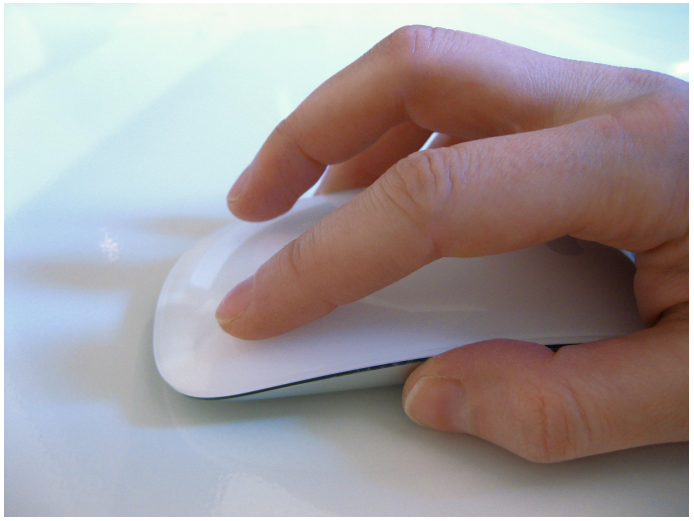
New Objects in Old Places

New Objects in Old Places



New Objects in Old Places

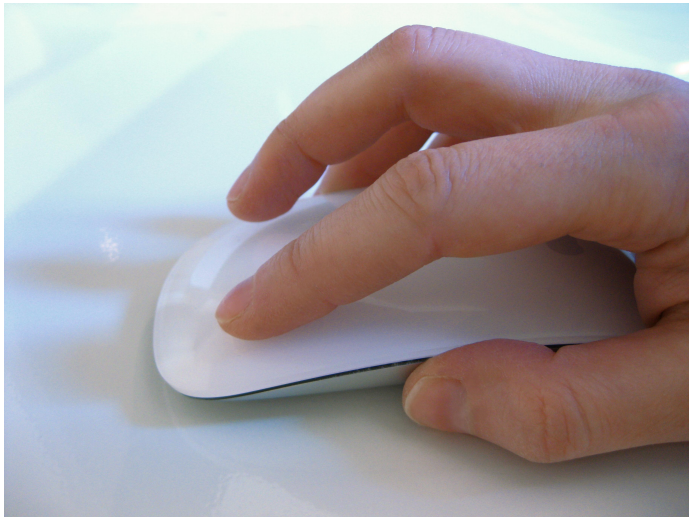
New Objects in Old Places



New Objects in Old Places

In the client file practice:

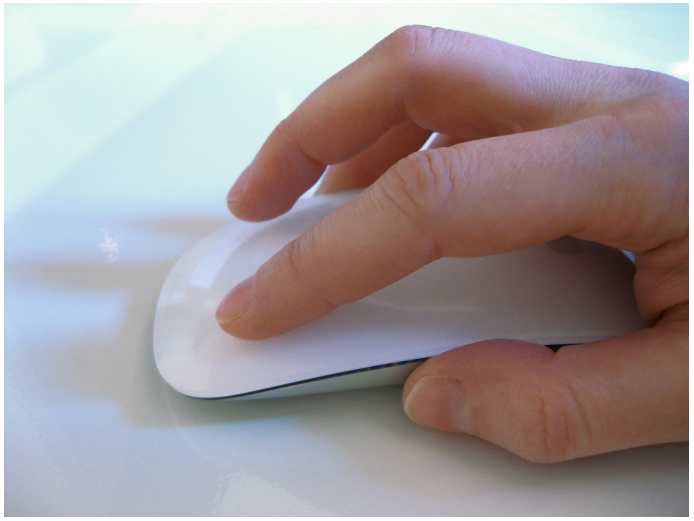
- ✓ Creating arrays of Account objects
- ✓ Writing & calling methods that have Account objects as parameters



New Objects in Old Places

In the client file practice:

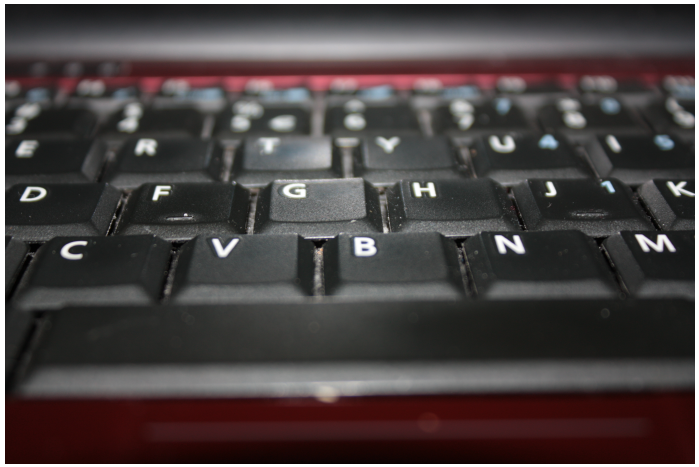
- ✓ Creating arrays of Account objects
- ✓ Writing & calling methods that have Account objects as parameters



New Objects in Old Places

In the client file practice:

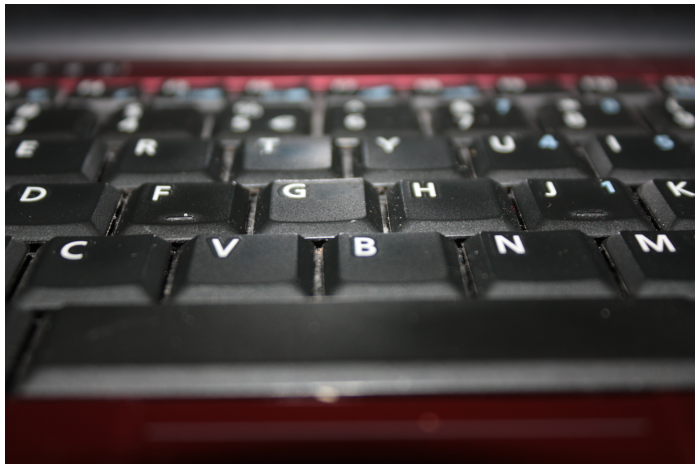
- ✓ Creating arrays of Account objects
- ✓ Writing & calling methods that have Account objects as parameters



New Objects in Old Places

In the class file practice:

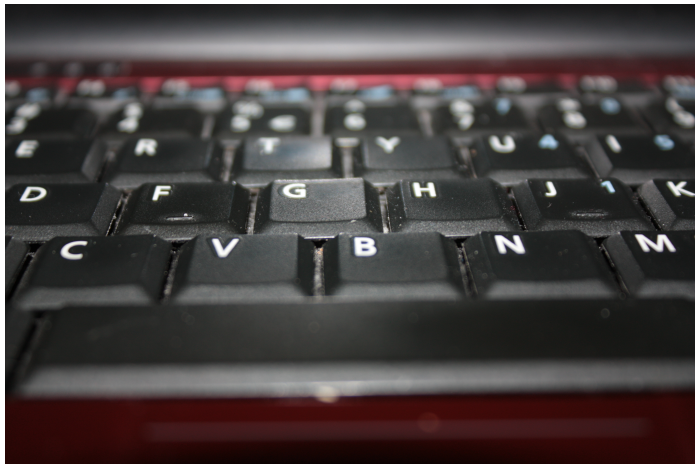
- ✓ Add static variable count
- ✓ Make changes to the constructors
- ✓ Test your constructors in the client file



New Objects in Old Places

In the class file practice:

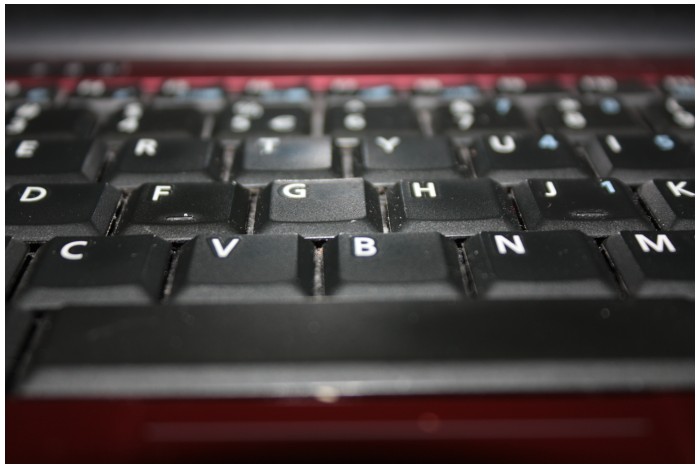
- ✓ Add static variable count
- ✓ Make changes to the constructors
- ✓ Test your constructors in the client file



New Objects in Old Places

In the class file practice:

- ✓ Add static variable count
- ✓ Make changes to the constructors
- ✓ Test your constructors in the client file



New Objects in Old Places

In the class file practice:

- ✓ Add static variable count
- ✓ Make changes to the constructors
- ✓ Test your constructors in the client file

New Objects in Old Places

Practice makes perfect!