

Lecture 6 - Sep. 24

Review of OOP, Exceptions

***Static Variables, Common Errors
Caller vs. Callee***

Announcements/Reminders

- **Lab1** released
- **Mockup Programming Test** this Fri (5pm or 6pm)
- Guides for **WrittenTest1** and **ProgTest1** to be released
- Reminder of rules for class **attendance checks**

Managing Account IDs: Manual

Slide 75

```
public class Account {  
    private int id;  
    private String owner;  
    public int getID() { return this.id; }  
    public Account(int id, String owner) {  
        this.id = id;  
        this.owner = owner;  
    }  
}
```

explicit parameter, presumably not duplicated

```
class AccountTester {  
    Account acc1 = new Account(1, "Jim");  
    Account acc2 = new Account(1, "Jeremy");  
    System.out.println(accl.getID() != acc2.getID());  
}
```

1. burden on the user of Account to specify unique id.
2. not a compilation error if ids are duplicated.

Declaring Global Variables among Objects

if not int. in the same tmp, it will be int. to default value (not Const.).

* each instance/object of Counter has its own copy (instance-specific value)

** all instances share the same copy (global).

```
public class Counter {  
    private int l; // non-static var.  
    static int g = 0; // int. (attributes)  
    // static var (global).  
    public Counter() {  
        this.l = 0;  
    } // only int. non-static var. in Const.
```

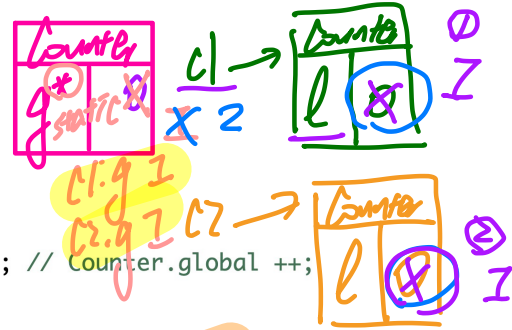
```
    public int getLocal() {  
        return this.l;  
    }
```

```
    public void incrementLocal() {  
        this.l ++;  
    } // c1 c2
```

```
    public void incrementGlobal() {  
        g ++; // warning (no need of t.o.)  
    } // this.g ++  
}
```

```
public class CounterTester {  
    public static void main(String[] args) {  
        Counter c1 = new Counter();  
        Counter c2 = new Counter();  
  
        System.out.println("c1's local: " + c1.getLocal());  
        System.out.println("c2's local: " + c2.getLocal());  
        System.out.println("Global accessed via c1: " + c1.g);  
        System.out.println("Global accessed via c2: " + c2.g);  
        System.out.println("Global accessed via Counter: " + Counter.g);  
    }
```

- ① c1.incrementLocal();
 - ② c2.incrementLocal();
 - ③ c1.incrementGlobal();
 - ④ c2.incrementGlobal();
- ```
Counter.g = Counter.g + 1; // Counter.global ++;
}
```



Counter.g ++;  
Counter.ClassName.varName g

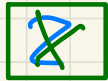
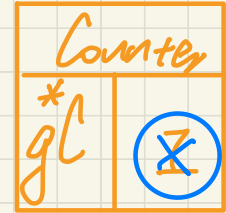
c1.g 2  
c2.g 2

# Managing Account IDs: Automatic

Slides 76 - 77

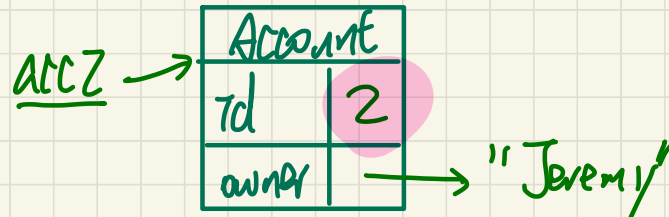
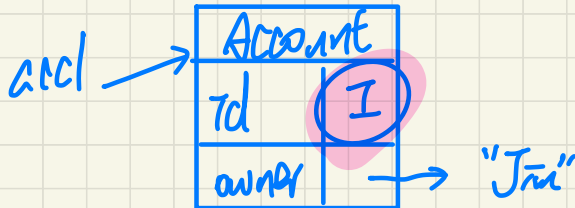
```
class Account {
 private static int globalCounter = 1;
 private int id; String owner;
 public Account(String owner) {
 *this.id = globalCounter;
 globalCounter ++;
 this.owner = owner; } }
```

parameter list  
not including  
id. burden of  
user of Account  
any more!



3

```
class AccountTester {
 Account acc1 = new Account("Jim");
 Account acc2 = new Account("Jeremy");
 System.out.println(acc1.getID() != acc2.getID()); }
```

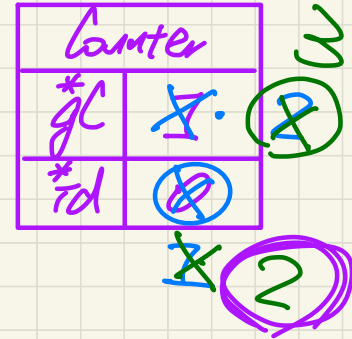


```

class Account {
 private static int globalCounter = 1;
 private int id; String owner;
 public Account(String owner) {
 this.id = globalCounter;
 globalCounter ++;
 this.owner = owner; } }

```

Handwritten annotations: A purple checkmark is above the `globalCounter` line. A purple arrow points to `static`. A blue arrow points to `id`. A green arrow points to `this.id`. A green arrow points to `globalCounter ++`. A green arrow points to `this.owner`.

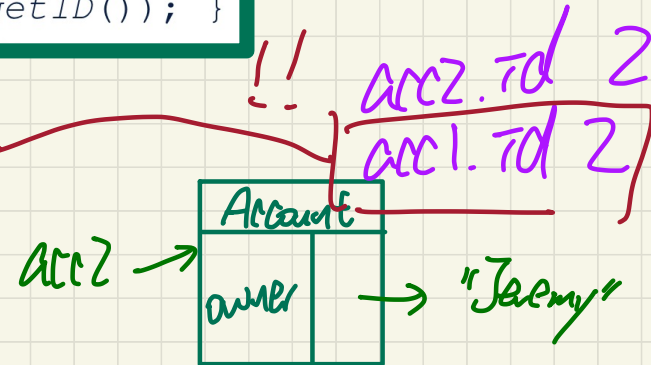
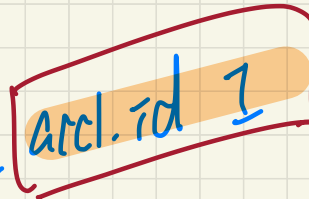


```

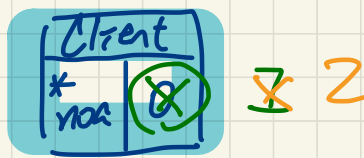
class AccountTester {
 Account acc1 = new Account("Jim");
 Account acc2 = new Account("Jeremy");
 System.out.println(acc1.getID() != acc2.getID()); }

```

Handwritten annotations: A blue arrow points to `new`. A green arrow points to `new`. A blue arrow points to `acc1`. A blue arrow points to `acc2`. A blue arrow points to `acc1.getID()`. A blue arrow points to `acc2.getID()`.



# Misuse of Static Variables



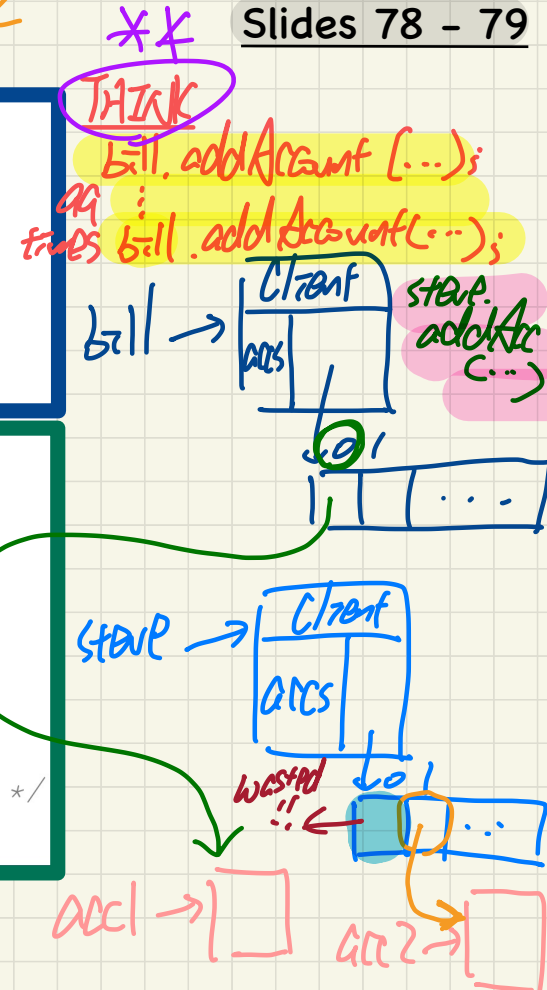
Slides 78 - 79

```
public class Client {
 private Account[] accounts;
 private static int numberOfAccounts = 0;
 public void addAccount(Account acc) {
 accounts[this.numberOfAccounts] = acc;
 this.numberOfAccounts++;
 }
}
```

Annotations:   
- `static` is circled in green.  
- `numberOfAccounts` is underlined in green.  
- `this.numberOfAccounts++` is underlined in green.  
- `addAccount` is circled in green.  
- `accounts[this.numberOfAccounts]` is underlined in green.  
- `acc` is underlined in green.  
- `bill` is written in green below the closing brace.  
- `stop` is written in green below the closing brace.

```
public class ClientTester {
 Client bill = new Client("Bill");
 Client steve = new Client("Steve");
 Account acc1 = new Account();
 Account acc2 = new Account();
 bill.addAccount(acc1);
 /* ... */
 steve.addAccount(acc2);
 /* ... */
}
```

Annotations:   
- `bill` is underlined in blue.  
- `steve` is underlined in blue.  
- `acc1` is underlined in blue.  
- `acc2` is underlined in blue.  
- `bill.addAccount(acc1)` is underlined in green.  
- `steve.addAccount(acc2)` is underlined in orange.



# Use of Static Variables: Common Error

Bank c-ibc = new Bank();  
c-ibc.branchName

Slides 80 - 82

```
1 public class Bank {
2 private string branchName;
3 public String getBrachName() { return this.branchName; }
4 private static int nextAccountNumber = 0;
5 public static String getInfo() {
6 nextAccountNumber++;
7 return this.branchName + nextAccountNumber;
8 }
9 }
```

should be replaced  
by some obj.

Expected Usage: Bank.getInfo() [no context obj.]

↳ Bank.branchName X ∵ branchName  
non-static

Fix 1

```
1 public class Bank {
2 private string branchName;
3 public String getBrachName() { return this.branchName; }
4 private static int nextAccountNumber = 0;
5 public static String getInfo() {
6 nextAccountNumber++;
7 return this.branchName + nextAccountNumber;
8 }
9 }
```

S =  
String "23" +  
"46"  
[S] "2346"

Fix 2

```
1 public class Bank {
2 private string branchName;
3 public String getBrachName() { return this.branchName; }
4 private static int nextAccountNumber = 0;
5 public static String getInfo() {
6 nextAccountNumber++;
7 return this.branchName + nextAccountNumber;
8 }
9 }
```

not ideal  
∴ each Bank obj. should have diff. branchName.

## Caller vs. Callee

- **caller** is the **client** using the service provided by another method.
- **callee** is the **supplier** providing the service to another method.

```
class C1 {
 void m1() {
 C2 o = new C2();
 o.m2(); /* static type of o is C2 */
 }
}
```

*calling context (caller)*

*C2.m2 being called (callee)*

Q: Can a method be a **caller** and a **callee** simultaneously?