

Lecture 5 - May 20

Review of OOP

*Tracing OOP: Arrays and Aliasing
Static vs. Non-Static Variables
Using Static Variables to Manage IDs*

Announcements/Reminders

- Today's class: notes template posted
- ProgTest0 this Friday (May 23)
 - + Guide (policies & requirements) posted
 - + PracticeTest0 posted
- Priorities:
 - + Lab 1
 - + Review slides on Classes and Objects

Anonymous Objects

Slide 56 - 58

```
1 double square(double x) {  
2     double sqr = x * x;  
3     return sqr; } → named exp
```

many methods
exp.

```
1 double square(double x) {  
2     return x * x; }
```

```
1 Person getP(String n) {  
2     Person p = new Person(n);  
3     return p; } → named obj.
```

many
obj.

```
1 Person getP(String n) {  
2     return new Person(n); }
```

Exercise

```
class Member {  
    private Order[] orders;  
    private int noo;  
    /* constructor omitted */  
    public void addOrder(Order o) {  
        this.orders[this.noo] = o;  
        this.noo++;  
    }  
    public void addOrder(String n, double p, double q) {  
        Order no = new Order(n, p, q);  
        this.orders[this.noo] = no;  
        this.noo++;  
    }  
}
```

N1: *Order no = new Order(n, p, q);*

N2: *this.orders[this.noo] = no;*

N3: *this.addOrder(no);*

Annotations:

- Red circle around "new Order(n, p, q);": *helper method call.*
- Red circle around "this.addOrder(no);": *anonymous object passed as arg.*
- Red circle around "this.orders[this.noo] = no;": *as a helper method.*
- Red circle around "Order no = new Order(n, p, q);": *duplicates code!*

N1 helper method call.
N2 anonymous object passed as arg.
N3 this.addOrder(new Order(n, p, q))

Copying Reference Values: Aliasing

Slide 52

```

Person alan = new Person("Alan");
Person mark = new Person("Mark");
Person tom = new Person("Tom");
Person jim = new Person("Jim");
Person[] persons1 = {alan, mark, tom}; →
Person[] persons2 = new Person[persons1.length];
for(int i = 0; i < persons1.length; i++) {
    persons2[i] = persons1[i]; } →
persons1[0].setAge(70);
System.out.println(jim.getAge()); 0
System.out.println(alan.getAge()); 70
System.out.println(persons2[0].getAge()); 70
persons1[0] = jim;
persons1[0].setAge(75);
System.out.println(jim.getAge()); 75
System.out.println(alan.getAge()); 70
System.out.println(persons2[0].getAge()); 70.

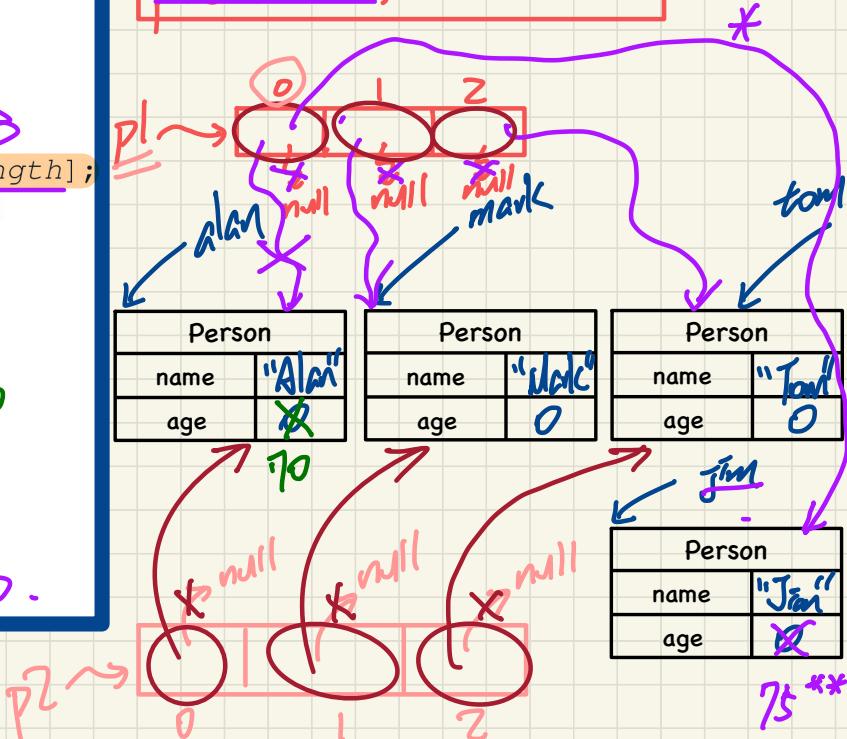
```

* Person[] pl = new Person[3];

pl[0] = alan;

pl[1] = mark;

pl[2] = tom;



* If #1: $i == 0 \Rightarrow p2[0] = pl[0]$

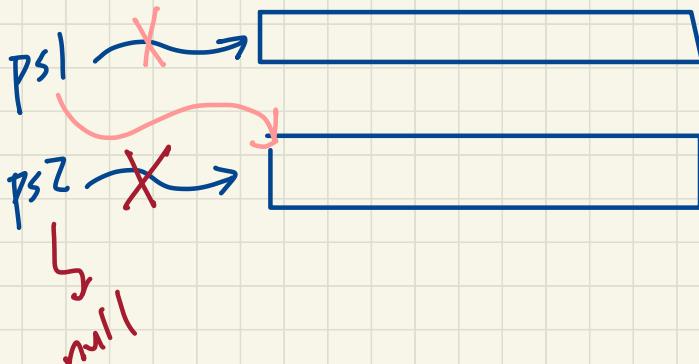
If #2: $i == 1 \Rightarrow p2[1] = pl[1]$

If #3: $i == 2 \Rightarrow p2[2] = pl[2]$

Re-Assigning Array References

`Person[] ps1 = new . . . ;`

`Person[] ps2 = new . . . ;`



	①	②	③
<code>ps1 == null</code>	F	F	F
<code>ps2 == null</code>	F	F	T
<code>ps1 == ps2</code>	F	T	F

① *Copy the address stored in ps1 into ps2*

`ps1 = ps2 ;`

②

`ps2 = null ;`

③

Arrays and Aliasing

multiple variables sharing the same address.

All alias paths
to "Mark"?

①

3 variables
sharing the
address of
"Mark" obj

alan

Person	
name	"Alan"
age	0

mark

Person	
name	"Mark"
age	0

tom

Person	
name	"Tom"
age	0

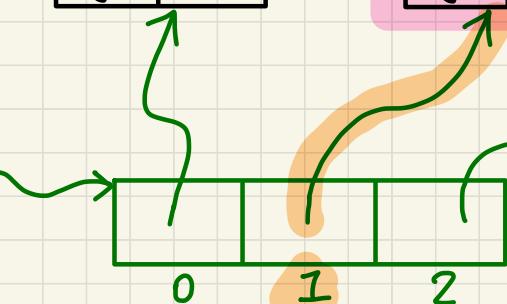
jim

Person	
name	"Jim"
age	0

②

mark
persons1[1]
persons2[1]

persons2



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.

Goal

Auto ID

generation &
management

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

2. when id's are duplicated,
not a compilation error.

1. burden on the Account user to specify
unique id.

→ Init. Attribute in Constructors

Declaring Global Variables among Objects

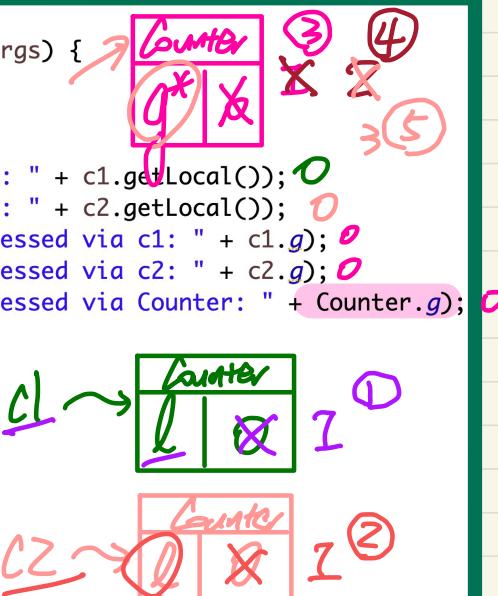
- * Each Counter/instance/obj has its own copy (Instance-specific value).
- ** All Counter/instances share the same copy (global)

```
public class Counter {  
    private int l; non-static variable  
    static int g = 0; static var. (attribute)  
    public Counter() { (global)  
        this.l = 0;  
    }  
  
    public int getLocal() {  
        return this.l;  
    }  
  
    public void incrementLocal() {  
        this.l++; X(s.l++)  
    } C1  
  
    public void incrementGlobal() {  
        this.g++; this.g++ is (warning)  
    }  
}
```

Counter.g += 1
recommended. ClassName.nameOfStaticVar.

```
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()); ①  
        System.out.println("c2's local: " + c2.getLocal()); ②  
        System.out.println("Global accessed via c1: " + c1.g); ③  
        System.out.println("Global accessed via c2: " + c2.g); ④  
        System.out.println("Global accessed via Counter: " + Counter.g); ⑤  
  
        ① c1.incrementLocal();  
        ② c2.incrementLocal();  
        ③ c1.incrementGlobal();  
        ④ c2.incrementGlobal();  
        ⑤ Counter.g = Counter.g + 1; // Counter.global ++;  
    }  
}
```

The diagram illustrates two instances of the Counter class, labeled C1 and C2. Each instance has a local variable 'l' (represented by a green box) and a shared global variable 'g' (represented by a red box). The global variable 'g' is initially 0. As the code executes, both C1 and C2 increment their local 'l' variables (labeled 1 and 2 respectively) and increment the shared 'g' variable (labeled 3 and 4 respectively). Finally, the global variable 'g' is updated to 1 (labeled 5), which is then printed by the main method.



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; } }
```

list of parameters
not including id
conforming to
(not burden of user of
Account const.
any more).

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

acc1 →

Account	
id	1
owner	"Jim"

acc2 →

Account	
id	2
owner	"Jeremy"

Counter

gc*	X
-----	---

