Lecture 6 - Sep 22

OOP Review

Use and Misuse of Static Variables
Use of this: Marriage Example
Reference-Typed Return Values
Anonymous Objects

Announcements/Reminders

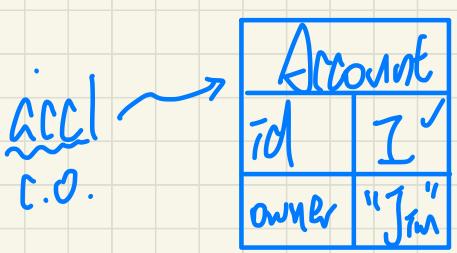
- Today's class: notes template posted
- Priorities:
  - + Review Lab0
  - + Complete Lab1; Due: Next Tuesday (Sep 30)
- ProgTest0 (this Wednesday, Sep 24):
  - + Guide & Mockup Test released
  - + **Required**: Go to your **enrolled** session.
  - + A physical **photo ID** is required.
- This Wednesday's class:
  - + We'll finish at about 12:30.
  - + 20 minutes to be covered in Section G's recording

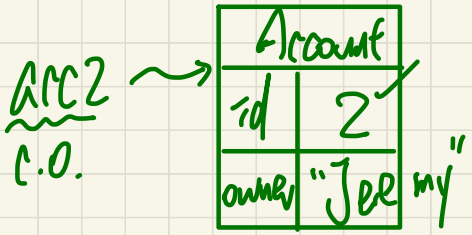
Alt. String id_str = "Id" + id;

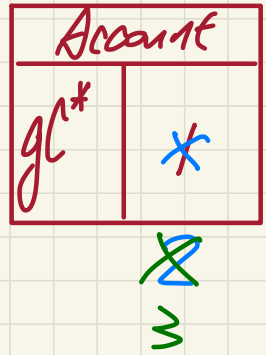
Managing Account IDs: Automatic


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

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



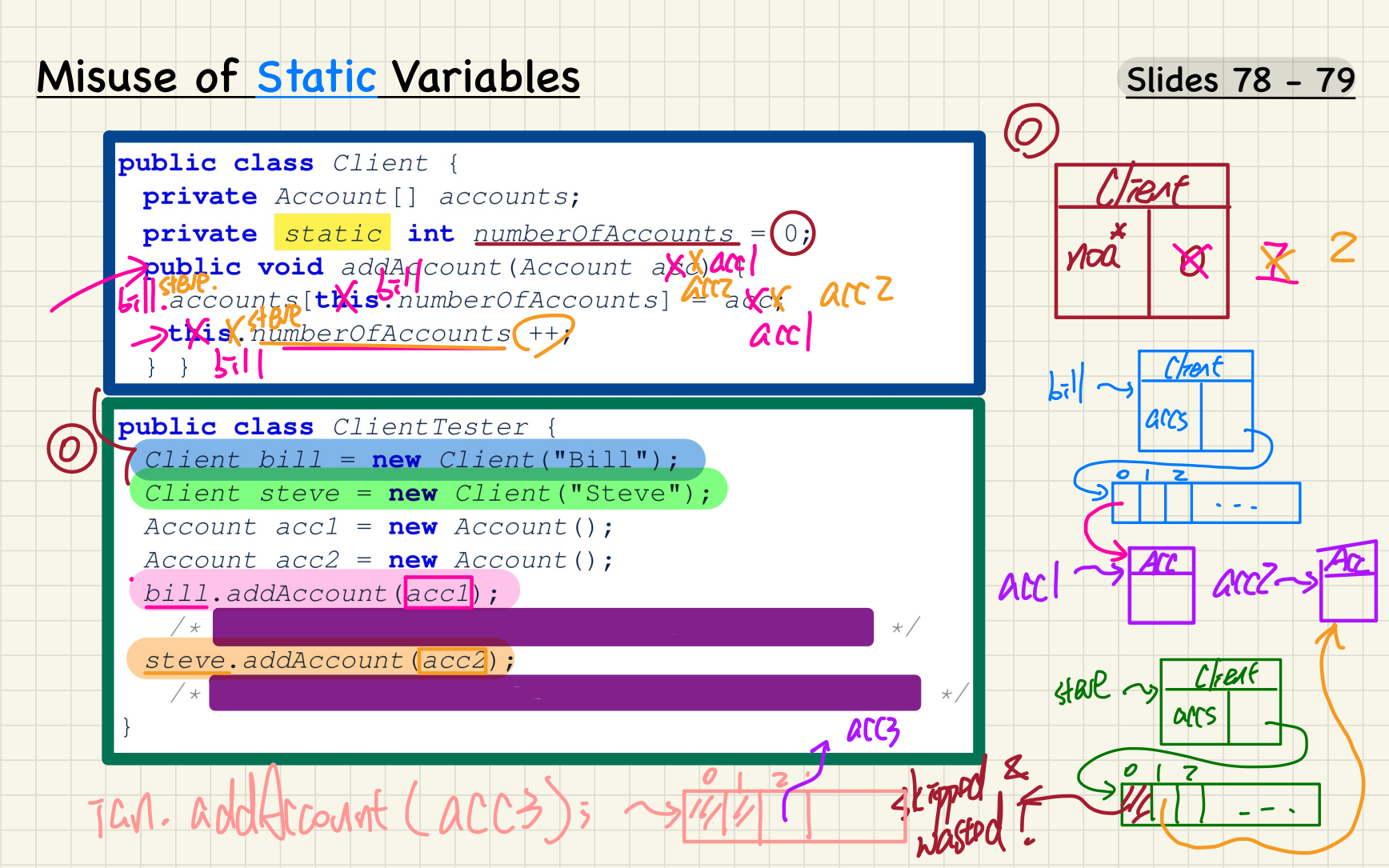




Misuse of Static Variables


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

```
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);
    /*                                          */
}
```



Use of Static Variables: Common Error


```
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  }
```

Bank

Compilation Error : branchName non-static var. used in getInfo static context.

Expected Usage : Bank.getInfo.

Bank — not a context object

Bank.branchName — not an object; non-static ✗

Fix1

```
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  }
```

remove the usage of non-static var

return # branches only.

Fix2

```
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  }
```

static

ok to use a static var here.

not good design : branch name should be instance-specific.

Example: Reference to this

recursive type

```
public class Person {
  private String name;
  private Person spouse;
  public Person(String name) {
    this.name = name;
  }
  public void marry(Person other) {
    if ( * marriage is legal. ) {
      ① this.spouse = other;
      ② other.spouse = this;
    } else { /* error */ }
}
```

when a person is created, spouse == null as default.

*. this.spouse == null

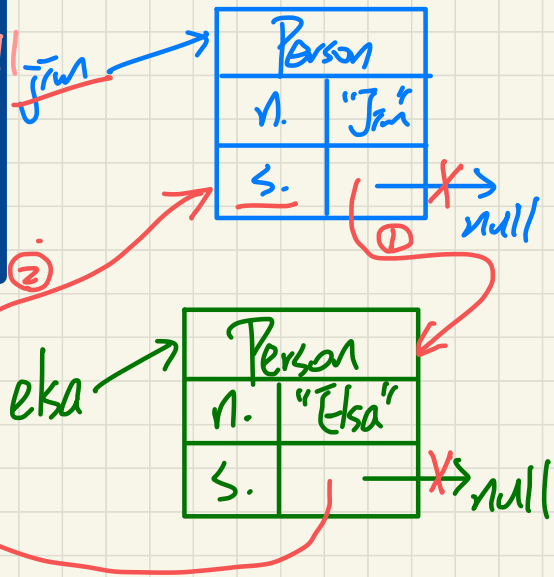
** other.spouse == null

C.O. (this) other

① jim.spouse = elsa;

② elsa.spouse = jim;

other. C.O. (this)




```
Person jim = new Person("Jim");
Person elsa = new Person("Elsa");
jim.marry(elsa);
```

C.O. Arg.

Math

(conjunction; and) $\wedge$

(disjunction; or) $\vee$

(negation; not) $\neg$

Programming

`&&`

`||`

short-circuit evaluation

`!`

Exercise

illegal ≡ !(legal)

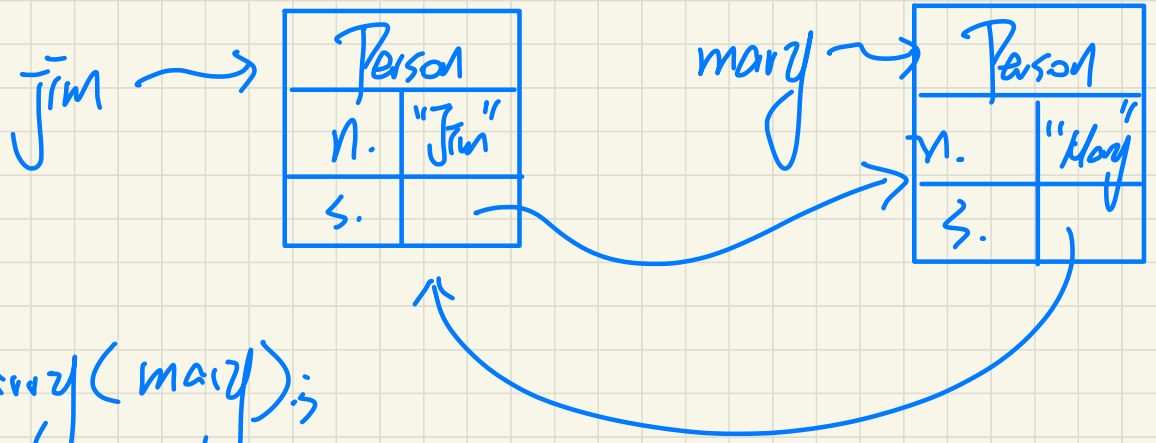
De Morgan

this == null
other == null

¬(P ∧ Q) ≡ ¬P ∨ ¬Q

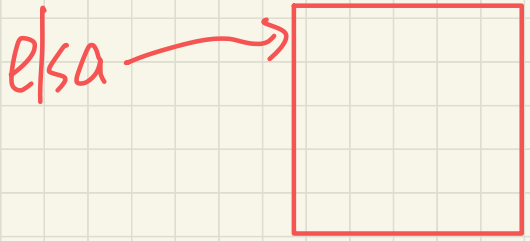
¬(P ∨ Q) ≡ ¬P ∧ ¬Q

```
public void marry(Person other) {
    if ( * proposed marriage is illegal ) { /* error */ }
    else {
        ①
        ②
    }
}
```




```
Jim.marry(mary);
```

```
Jim.marry(elsa);
```



Reference-Typed Return Values


```
public class Point {
   /* A mutator modifying the context Point object */
   public void moveUp (double i) {
      this.y = this.y + i;
   }

   /* An accessor returning a new Point object */
   public Point movedUpBy(double i) {
      Point np = new Point(this.x, this.y);
      np.moveUp(i);
      return np;
   }
}
```


```
public class PointTester {
   public static void main(String[] args) {
      Point p1 = new Point(2.5, -3.6);
      p1.moveUp(7.8);
      Point p2 = p1.movedUpBy(6.4);
      System.out.println(p1 == p2);
   }
}
```

Anonymous Objects


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

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

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

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

Exercise

```
class Member {
  private Order[] orders;
  private int noo;
  /* constructor ommitted */
  public void addOrder(Order o) {
    this.orders[this.noo] = o;
    this.noo++;
  }
  public void addOrder(String n, double p, double q) {



  }
}
```