# Lecture 5 - Sep. 19

## Review of OOP

***Anonymous Objects***
***Reference to this***
***Static Variables***

# Announcements/Reminders

- LabOP2 due tomorrow (Friday) at 12 noon!
- Lab1 to be released after LabOP2 is due.
- In-Lab demo on the Programming Pattern
- Today's office hour to be re-scheduled
- Mockup Programming Test next Fri (5pm or 6pm)

27th

# Anonymous Objects



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

anonymous

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

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

anonymous

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

object expression (without ref. a variable).

## 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) {
    [redacted]
  }
}
```

overloading

1. Order o = new Order(n, p, q);
   this.orders[noo] = o;
   noo ++;
   this.addOrder(o);
2. this.addOrder( );

(1) No anonymous objects.

↳ many local, intermediate variables

in-between is better

(2) No intermediate variable

↳ long expressions

# Example: Reference to this

Slide 59 - 61

Q. this.spouse = other.name

Person String

$\neg(p \vee q) \equiv \neg p \wedge \neg q$

$\neg(p \wedge q) \equiv \neg p \vee \neg q$

```
public class Person {
  private String name;
  private Person spouse;
  public Person(String name) {
    this.name = name;
  }
  public void marry(Person other) {
    if (*                    ) {
    else {
      this.spouse = other;
      other.spouse = this;
    }
  }
}
```

robust.

error.

this.spouse != null || other.spouse != null

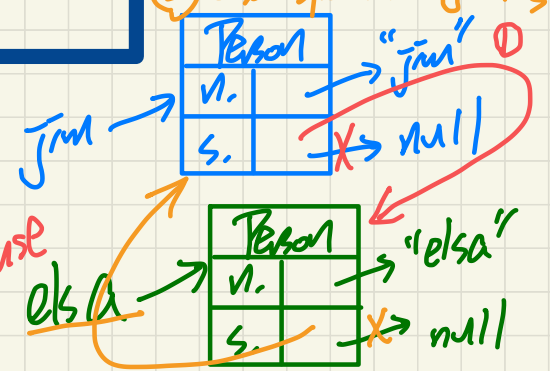
married

married

!(this.spouse == null && other.spouse == null)

jim.spouse = elsa;

elsa.spouse = jim;




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

this. other

jim.spouse == jim.spouse.spouse

# Exercise

```
void marry (Person other) {
    if (                                    ) {
        this.spouse = other;
        other.spouse = this;
    }
    else { error }
}
```

# Managing Account IDs: Manual



```java
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;
  }
}
```

Principles

different Account objects should have diff. ids.

```java
class AccountTester {
  Account acc1 = new Account(1, "Jim");
  Account acc2 = new Account(2, "Jeremy");
  System.out.println(acc1.getID() != acc2.getID());
}
```

too much burden on users of Account class

# Declaring Global Variables among Objects

Ref. to static var. : ClassName.varId

```
public class Counter {
    private int l;
    static int g = 0;

    public Counter() {
        this.l = 0;
    }

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

    public void incrementLocal() {
        this.l ++;
    }

    public void incrementGlobal() {
        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);

        c1.incrementLocal();

        c2.incrementLocal();

        c1.incrementGlobal();

        c2.incrementGlobal();

        Counter.g = Counter.g + 1; // Counter.global ++;
    }
}
```