Lecture 6 - May 22

Review of OOP, Exceptions

Misuse of Static Variables
Marriage Example: Advanced Use of this
Reference-Typed Return Values
Caller vs. Callee

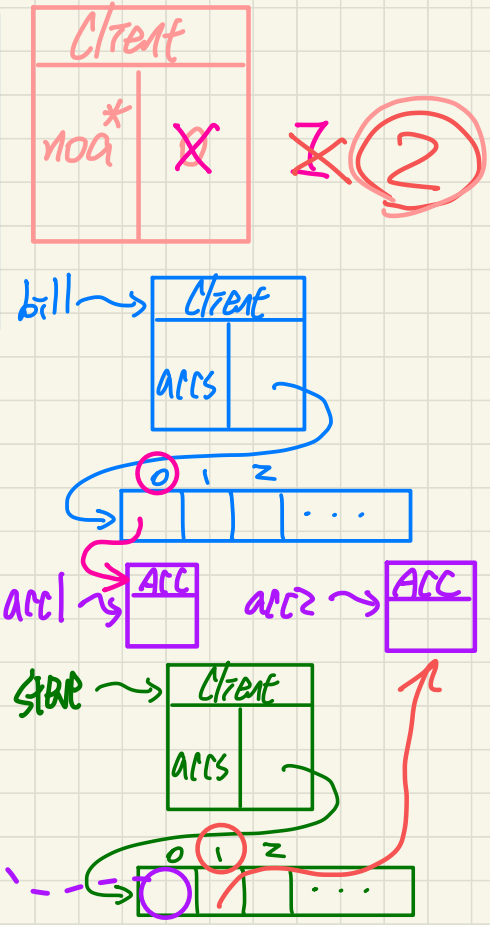
Announcements/Reminders

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

Misuse of Static Variables


```
public class Client {
  private Account[] accounts;
  private static int numberOfAccounts = 0;
  public void addAccount(Account acc) {
    this.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        X return this.branchName + nextAccountNumber;
8     }
9  }
```

Bank

branchName

getInfo

Compilation Error: non-static variable used in static context

Expected Usage: Bank.getInfo()

not a context object

not a legal C.O.

↳ Bank.branchName()

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 ref. to non-static variable

Fix2

static

works programmatically but inappropriate design

Bank.getInfo() → # of branches

```
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  }
```

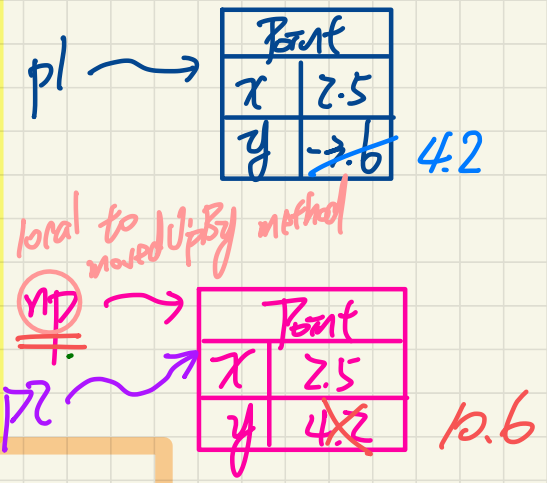
1. Bo/Grace
2. Miss.
3. ↳ supposed to be instance-specific

Reference-Typed Return Values


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

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



return np's address

np's address

false

Example: Reference to this

jim.spouse
jim.spouse.spouse → jim

recursive type

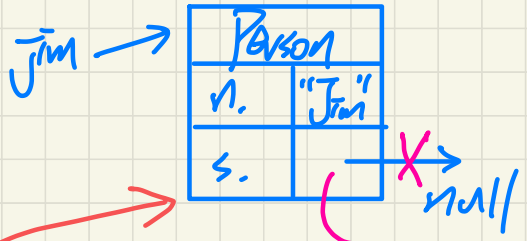
```
public class Person {
  private String name;
  private Person spouse;
  public Person(String name) {
    this.name = name;
  }
  public void marry(Person other) {
    if( /* marriage is legal */ ) {
      this.spouse = other;
      other.spouse = this;
    } else { /* error */ }
  }
```

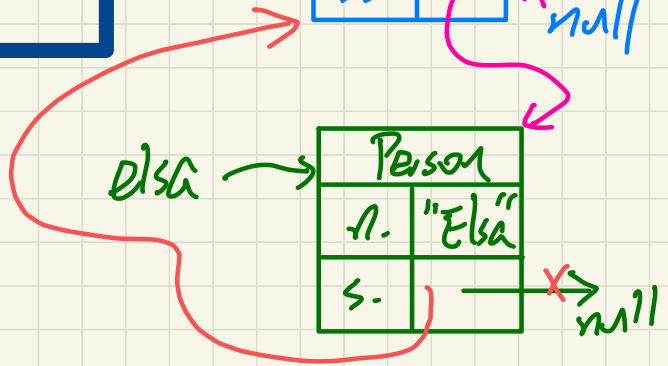
this.spouse == null && other.spouse == null

what if it is used?
exercise? test case?

C.O. this other

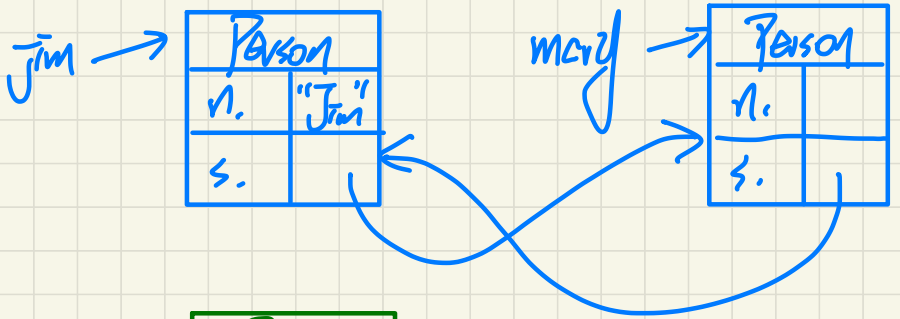
① jim.spouse = elsa;
② elsa.spouse = jim;
elsa: other
jim: this.





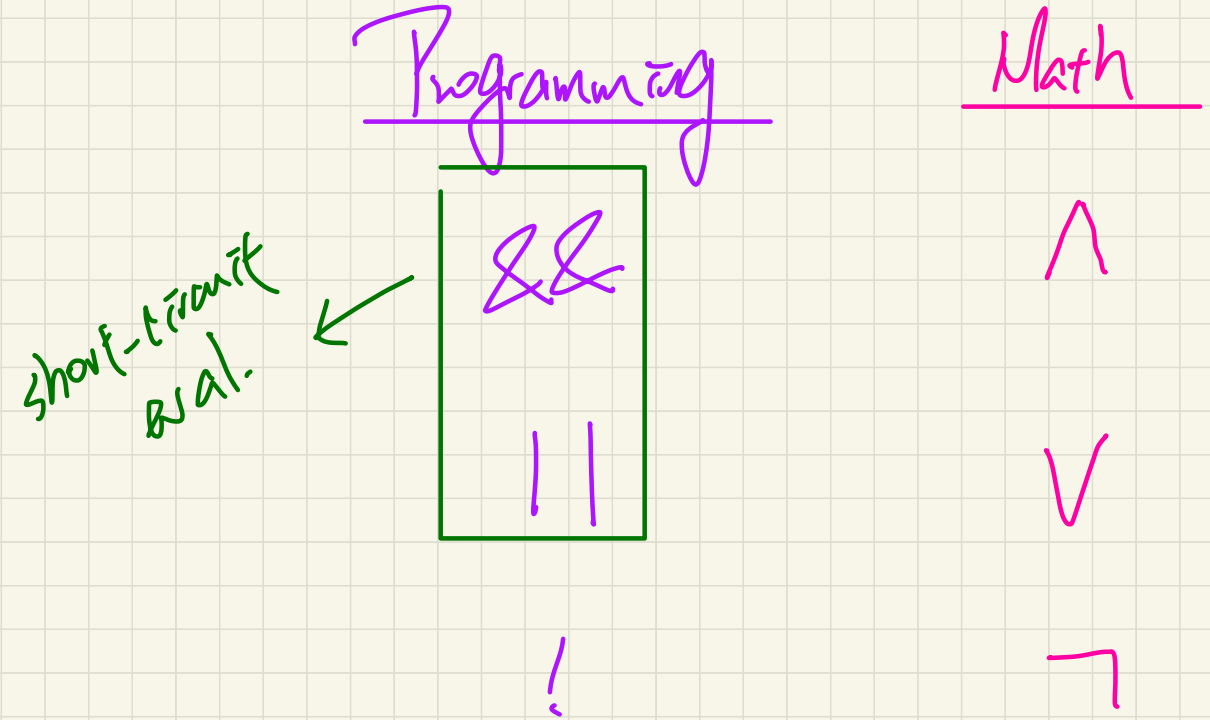

```
Person jim = new Person("Jim");
Person elsa = new Person("Elsa");
jim.marry(elsa);
```

C.O. arg.



jim.marry(elsa);

↳ error


Programming
Math
&&
||
!
∧
∨
¬
short-circuit eval.

Exercise

illegal

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

legal

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

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

Caller vs. Callee

method that calls another method callee

method that's called by another method caller

- 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 */
  }
}
```

caller

callee

caller: C1.m1 does not imply that m1 is static!

callee: C2.m2

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