Lecture 5

Wednesday September 17

Tue Oct 1 Mini Test # 1 (written) (10%)

Tue Oct 8 Lab Test # 1 (programming) (20%)

static int i] global to all instances

int j] local to each instance.

Managing Account ID: Manually

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

non-static

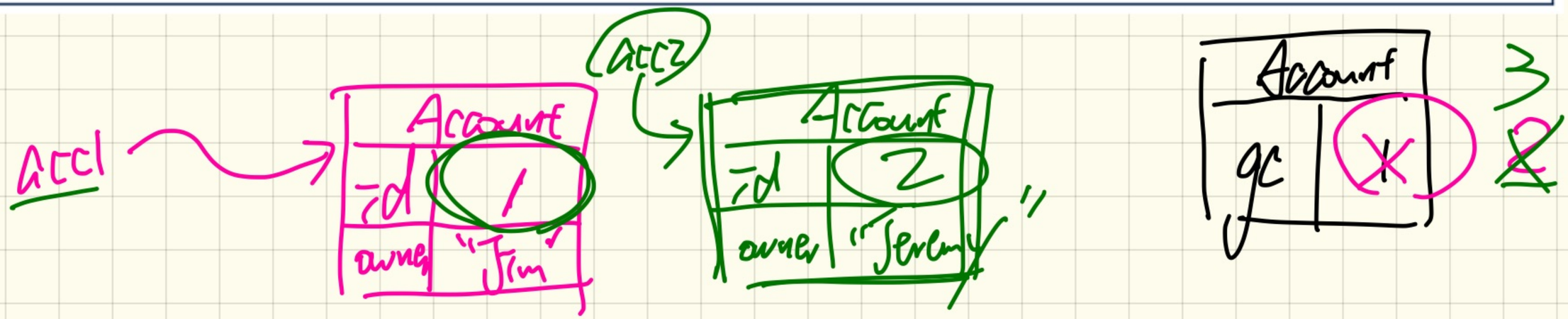
anyone wanting to create an instance of Account must supply an id themselves.

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

Managing Account ID: Automatically

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

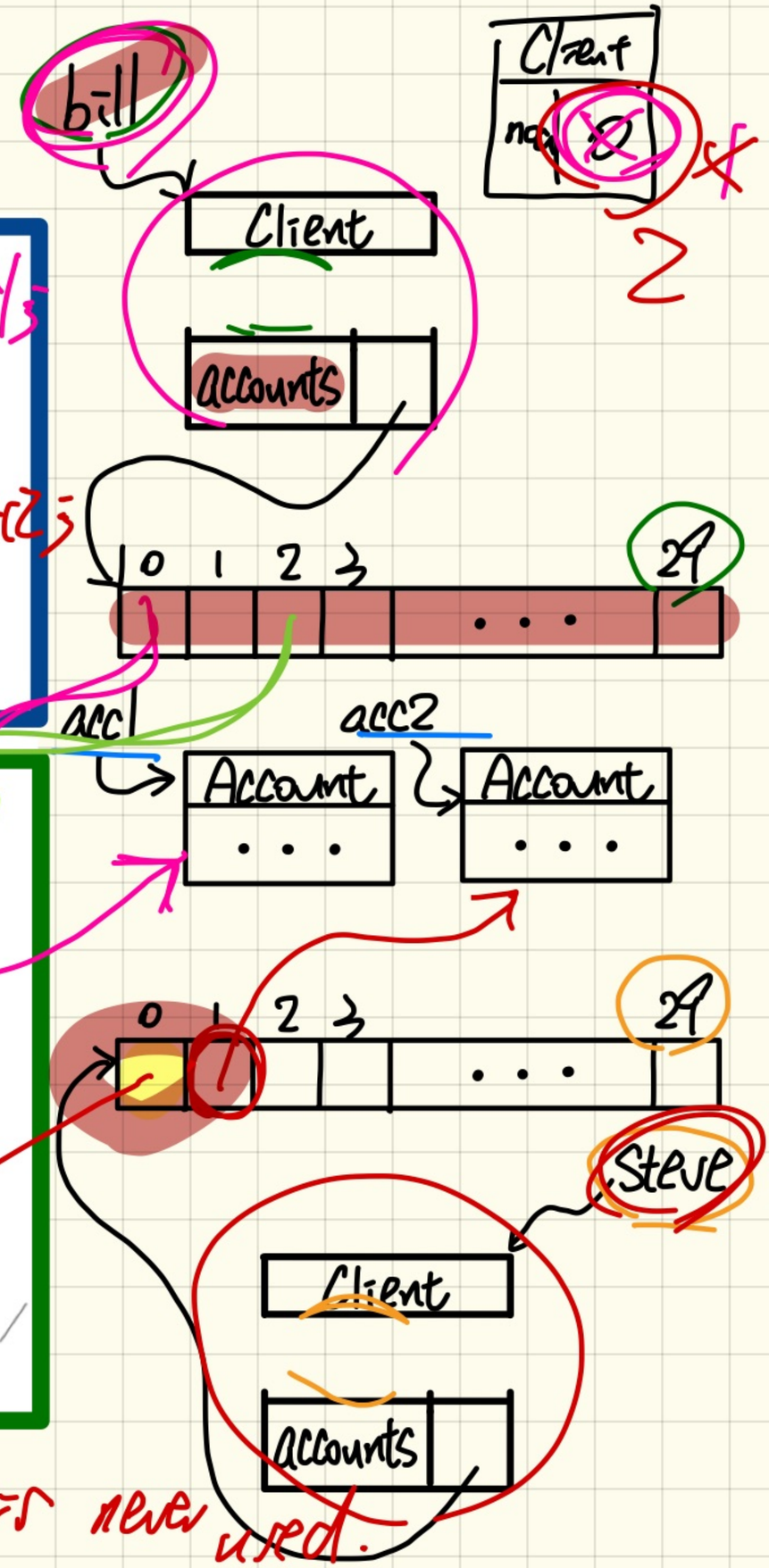
```
class AccountTester {
  Account acc1 = new Account("Jim");
  Account acc2 = new Account("Jeremy");
  System.out.println(acc1.id != acc2.id); }
```



Misuse of Static Variables

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

```
class ClientTester {
Client bill = new Client("Bill");
Client steve = new Client("Steve");
Account acc1 = new Account();
Account acc2 = new Account();
bill.addAccount(acc1);
   /* correctly added to bill.accounts[0] */
steve.addAccount(acc2);
   /* mistakenly added to steve.accounts[1]! */
}
```



Use of Static Variables: Common Errors

```
1  public class Bank {
2      public string branchName;
3      public static int nextAccountNumber = 1;
4      public static void useAccountNumber() {
5          System.out.println ("branchName" + ...);
6          nextAccountNumber ++;
7      }
8  }
```

static method

↳ accessed:

Bank.useAccount()

↳ not context object

non-static

↳ a context object is needed

branchName

→ Illegal use of non-static variable

in a static context.

useAccountNumber.

```
1  public class Bank {
2      public string branchName;
3      public static int nextAccountNumber = 1;
4      public static void useAccountNumber() {
5          System.out.println (branchName + ...);
6          nextAccountNumber ++;
7      }
8  }
```

static variables only

```
1  public class Bank {
2      public string branchName;
3      public static int nextAccountNumber = 1;
4      public static void useAccountNumber() {
5          System.out.println (branchName + ...);
6          nextAccountNumber ++;
7      }
8  }
```

```
1  public class Bank {
2      public string branchName;
3      public static int nextAccountNumber = 1;
4      public static void useAccountNumber() {
5          System.out.println (branchName + ...);
6          nextAccountNumber ++;
7      }
8  }
```

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 */
  }
}
```

caller : C1 m1

callee : C2 m2

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

C2 m2 to be caller and callee at the same time?

```
class C2 {
  m2() {
  }
}
```

→

```
C3 o2 = new C3();
o2.m3();
```

Error Handling via Console Messages: Circles

```
1  class Circle {
2    double radius;
3    Circle() { /* radius defaults to 0 */ }
4    void setRadius(double r) {
5      if ( r < 0 ) { System.out.println( "Invalid radius." ); }
6      else { radius = r; }
7    }
8    double getArea() { return radius * radius * 3.14; }
9  }
```

Caller?

Callee?

```
1  class CircleCalculator {
2    public static void main(String[] args) {
3      Circle c = new Circle();
4      c.setRadius( -10 );
5      double area = c.getArea();
6      System.out.println("Area: " + area);
7    }
8  }
```

Error Handling via Console Messages: Bank

```
class Account {
  int id; double balance;
  Account(int id) { this.id = id; /* balance defaults to 0 */ }
  void deposit(double a) {
    if ( a < 0 ) { System.out.println( "Invalid deposit." ); }
    else { balance += a; }
  }
  void withdraw(double a) {
    if ( a < 0 || balance - a < 0 ) {
      System.out.println( "Invalid withdraw." ); }
    else { balance -= a; }
  }
}
```

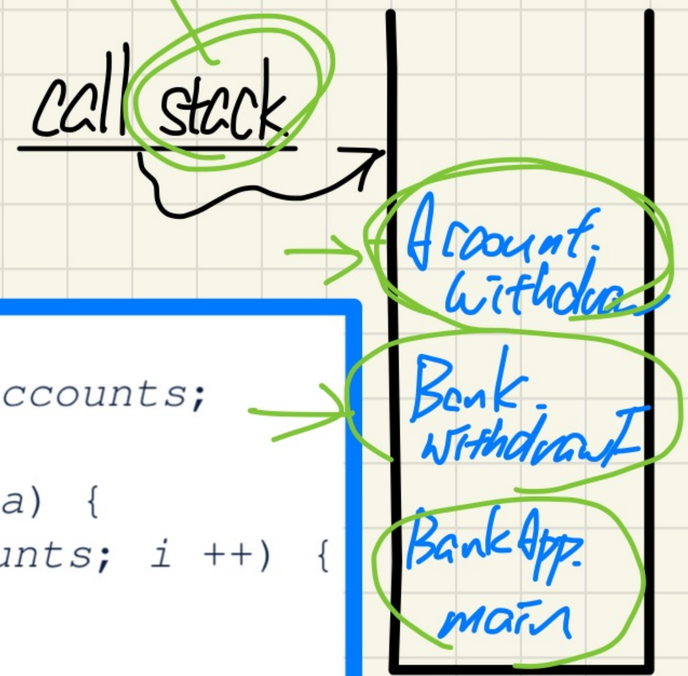
errors to console not appropriate

```
class Bank {
  Account[] accounts; int numberOfAccounts;
  Account(int id) { ... }
  void withdrawFrom(int id, double a) {
    for(int i = 0; i < numberOfAccounts; i ++) {
      if(accounts[i].id == id) {
        accounts[i].withdraw( a );
      }
    }
  } /*
}
```

```
class BankApplication {
  pubic static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    Bank b = new Bank(); Account acc1 = new Account(23);
    b.addAccount(acc1);
    double a = input.nextDouble();
    b.withdrawFrom(23, a );
  }
```

Caller?
Callee?

LIFO
Last In First Out



context	caller	callee
Account		
Bank	Bank. withdrawFrom	Account. withdraw
BankApp	BankApp. main	Bank. withdrawFrom

Run as Java Application

catch-or-specify

When an exception is thrown:

1. Specify:
   - include "throws ——" as part of the method
   - throws exception

2. Catch:
   - catch the exception that may be thrown and do something about.

```
try {
    ...
} catch {
    ...
}
```