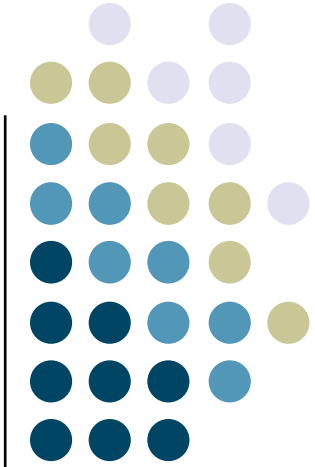
# Java Object-oriented Programming I




John Stasko

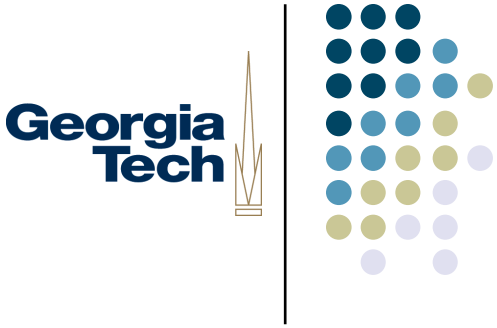
CS 6452

Prototyping Interactive Systems

# HWs Redux

- HW 3
- HW 4

# Learning Objectives



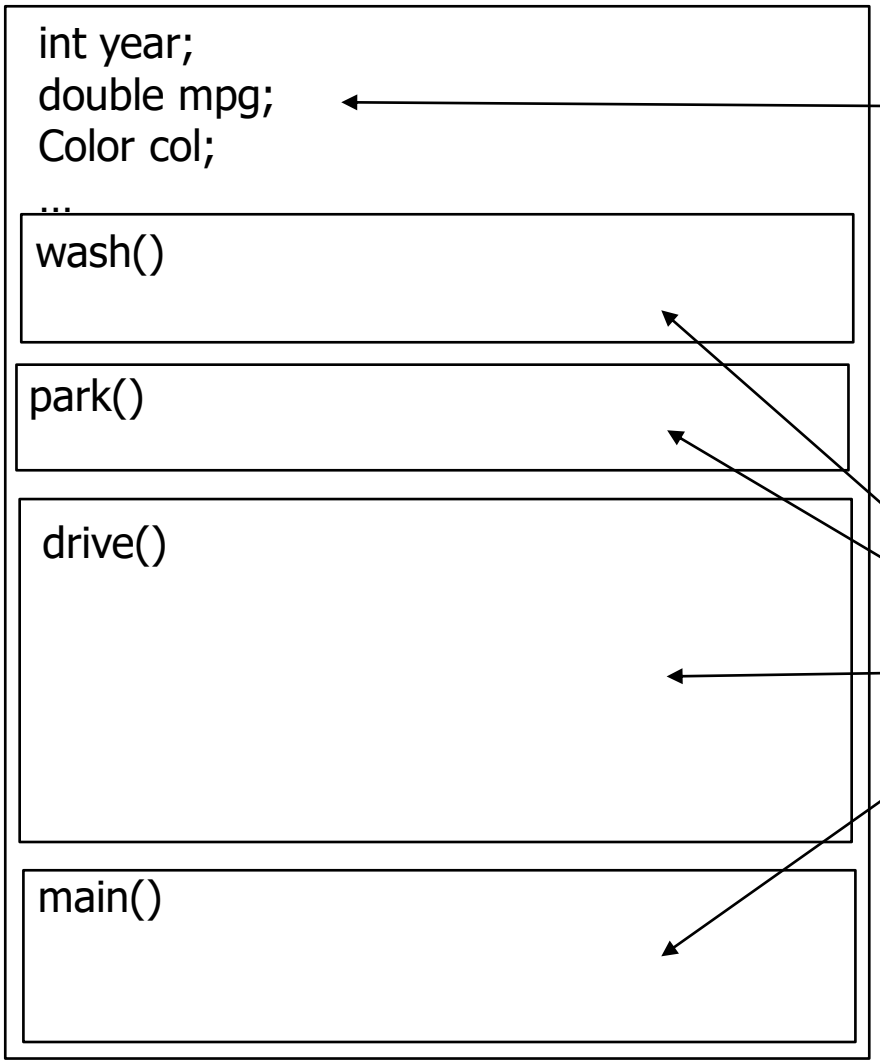

- Java classes and objects
  - Instance data
  - Methods
  - Constrcutors
  - Visibility
  - Scope
  - Static

# Modeling Objects

- Car
  - General attributes: year, color, VIN #, horsepower, speed, mpg, …
  - Behaviors: drive, brake, wash, park

- Individual instances of a car
  - Hayley's, Larry's, …

# class Car




Data declarations

Methods

A class is a "type"

All class members
- Data (instance vars)
- Methods

# Instance Data

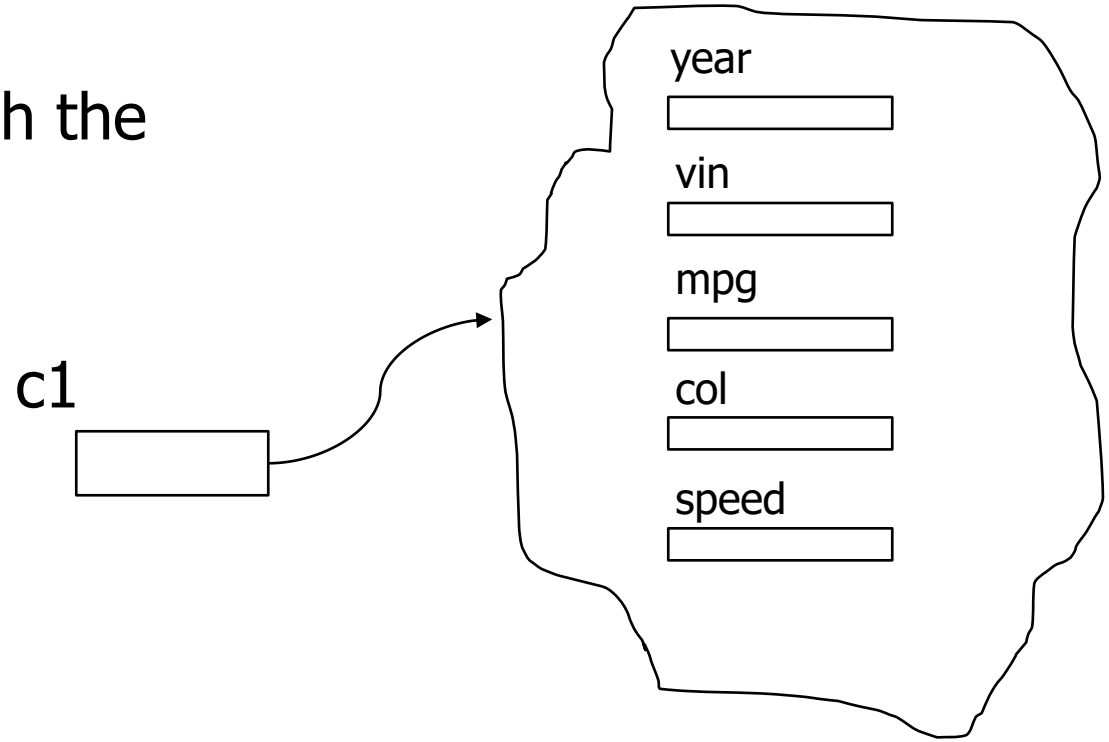

- Put values inside class but not in method
- Each object that gets instantiated for a class receives its own copy of them
- Variables are automatically initialized, but good practice to do it manually (in constructor)

```
Car c1 = new Car();
```

Use the `new` operator to create an instance

Access fields through the `.` operator

```
c1.year
c1.vin
c1.mpg
c1.col
c1.speed
```



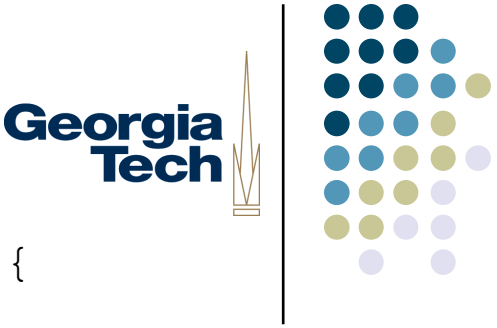
# Methods

- Functions/procedures (behaviors) within a class

```
public double drive(int time) {
    double distance;

    distance = time * speed;
    return distance;
}
```

- When we do `c1.drive(20);` control flows to method, through it, then returns

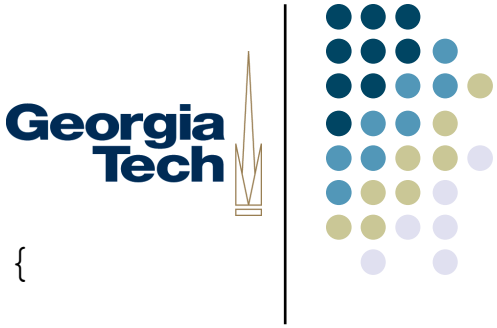
# Methods



```
public double drive(int time) {
    double distance;

    distance = time * speed;
    return distance;
}
```

- `return` statement – Control immediately goes back (need not be at end of method)
- Local variables – declared inside a method and only visible there (e.g., `distance`)

# Methods



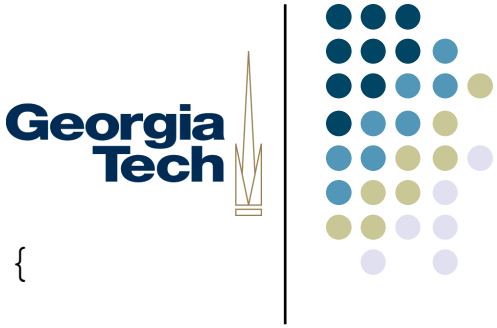
```
public double drive(int time) {
    double distance;

    distance = time * speed;
    return distance;
}
```

- Other code

```
double total;
total = 100.0 + c1.drive(24);
```

- The method `drive` returns a double that added to 100.0 and copied into `total`

# Methods



```
public double drive(int time) {
    double distance;

    distance = time * speed;
    return distance;
}
```

- Parameters – values passed in to method
  - Formal params – Names of params in header
  - Actual params – Values passed in when running
- Formal params are just local variables literally

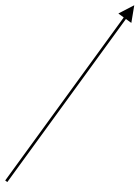
# Methods

```
int a;
a = 12;
total = 100.0 + c1.drive(a+3);

// elsewhere

      public double drive(int time) {
        time = 1;
        return time;
      }
```

- At execution time, values copied into formal parameters
- Parameters passed in call by value method

# Methods

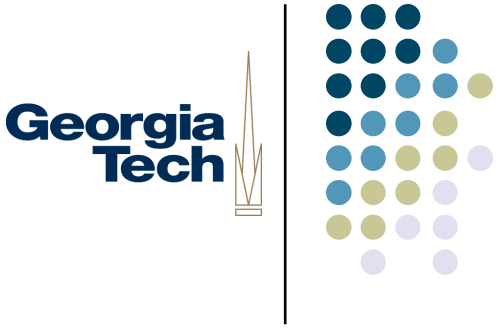
```
public double drive(int time) {
    double distance;

    distance = time * speed;
    return distance;
}
```



- Nothing in front of `speed`
- Which speed?

- The instance variable within the object upon which this method was called

# Methods



Other code

```
double d;
Car c3 = new Car();
d = c3.drive(50);
 // in this case, it uses c3's speed
```

It's like

```
distance = time * <thecallingobject>.speed;
```

or

```
distance = time * (c3).speed;
```

or

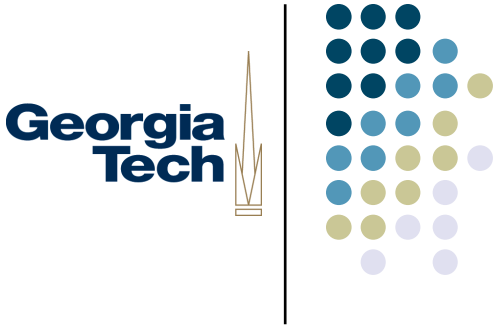
```
distance = time * this.speed;
```

# Methods

```
distance = time * this.speed;
```

- `this` – java reserved word used inside methods
  - It refers to object upon which method was invoked
- These type of method calls always performed in the context of an object

# Example Program

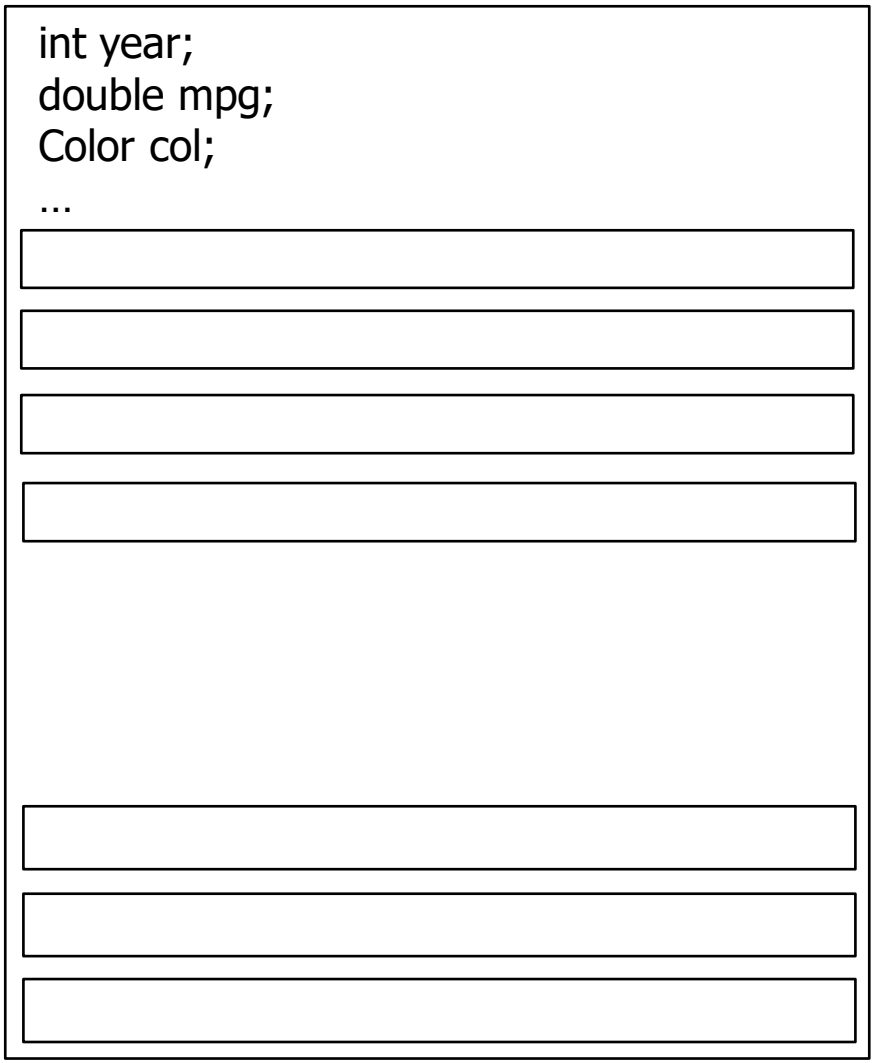



- RollingDice
  - chap 4

# Encapsulation


- Objects should be responsible for themselves
- Don't want outsiders modifying instance data
- Specify certain methods for outsiders (other classes) to use
  - Called the *class interface*

# class Car




instance data

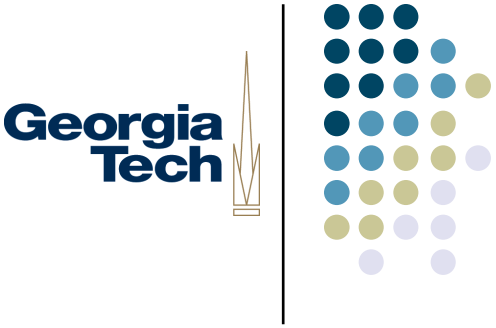
# Visibility


- How do we specify what is externally visible?
  - Use modifiers

- Visibility modifiers – Control access
  - `public`, `private`, `protected`

| `public` | `private` | `protected` |
|---|---|---|
| outsiders | only in class | (later) |

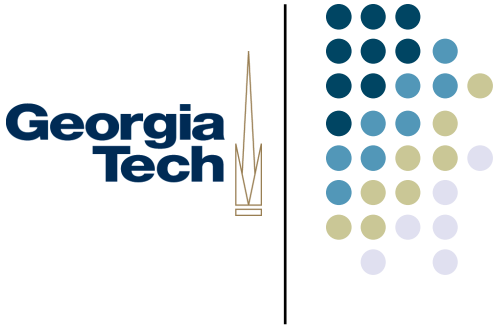
# Access



| | public | private |
|---|---|---|
| variables | X | natural |
| methods | service to clients | internal class support |

Class has access to all private members

```
public class Car
{
   private int vin,year;
   private double speed, mpg;

   public void drive() {
     …
   }

   public int getYear() {
      return year;
   }

   public void setYear(int y) {
      year = y;
   }

   public Car() {
      …
   }


   private void diagnose() {
     …
   }

   public static void main (String[] args)
   { …  }
}
```




"Accessor" method

"Modifier" method

Constructor

Internal method

# Constructor


- Special method called when objects are instantiated
- Same name as the class
- Their primary use is to initialize instance variables

# Constructors

```
public Car(int y, double s, double m) {
    year = y;
    speed = s;
    mpg = m;
}
```

- What if we did

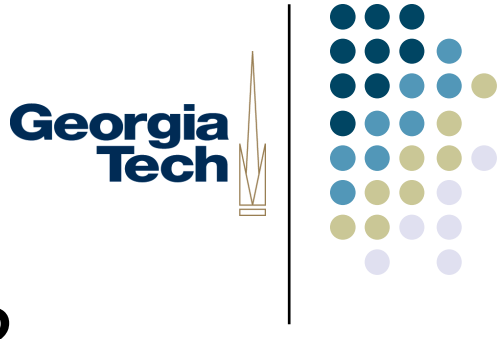
```
public Car(int year, double speed, double mpg) {
    year = year;
    speed = speed;
    mpg = mpg;
}
```

# Constructors

- How to correct that?

```
public Car(int year, double speed, double mpg) {
    this.year = year;
    this.speed = speed;
    this.mpg = mpg;
}
```

# Variable Scope




- What is the *scope* of a variable?
  - Region of a program where it's visible
- Formal parameter
  - The method in which it is a parameter
- Local variable
  - The method in which it is defined
- Instance variable
  - Entire class

# Questions

- Legal?

```
public void foo(int a) {
    int a;
    …
}
```

- No, compile error
  - Two local variable declarations of a

# Method Overloading

- Use of same method name with different parameter lists to create multiple versions of method

```
public int drive(int a) {
    …
}
```

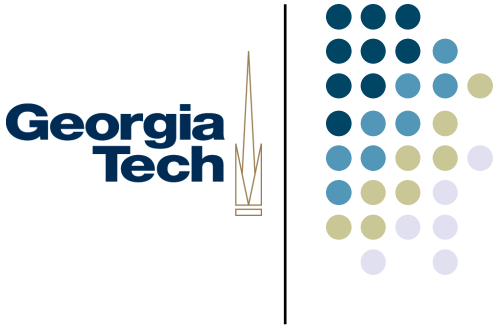
```
public int drive(int a, double d) {
    …
}
```

- How does it know which is called?
  - Looks at call and matches
  - `c1.drive(5, 23.4);`

# Example Program

- Account & Transactions
  - chap 4

# Static Variables




- Another modifier
- So far, seen local vars and instance vars
- Another kind: static (class) variable
  - One copy shared by all instances of class
  - `private static int count = 0;`
  - Memory space for it is in class, not instances
  - Useful for object counters

# Example Program

- Slogan
  - chap 6

# Static Methods


- Do not operate in the context of a particular object (no `this`)
  - So they cannot reference instance variables
  - Typically worker functions, often mathematical

- Look at Slogan again
  - static `getCount()` cannot access `phrase`

# null

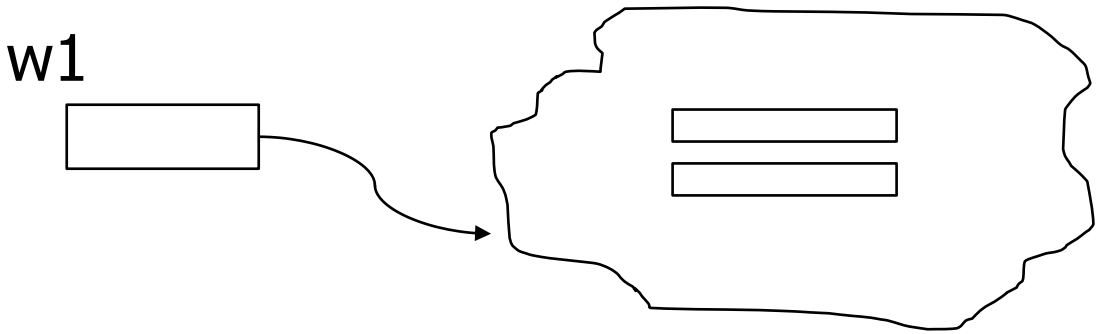
```
public class Worker {
   private String name;
   private int id;

   public Worker(name, id) {
      this.name = name;
      this.id = id;
   }
}
```

```
Worker w1;
w1 = new Worker("Mary", 12);
```

After declaration, what is w1?

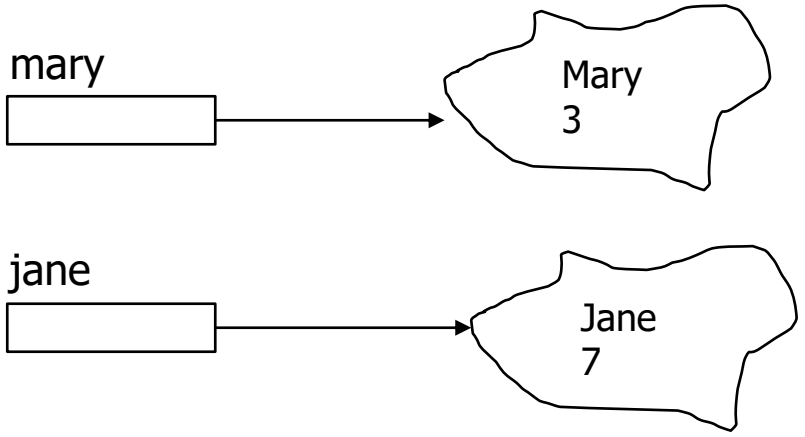
`null`

After instantiation:



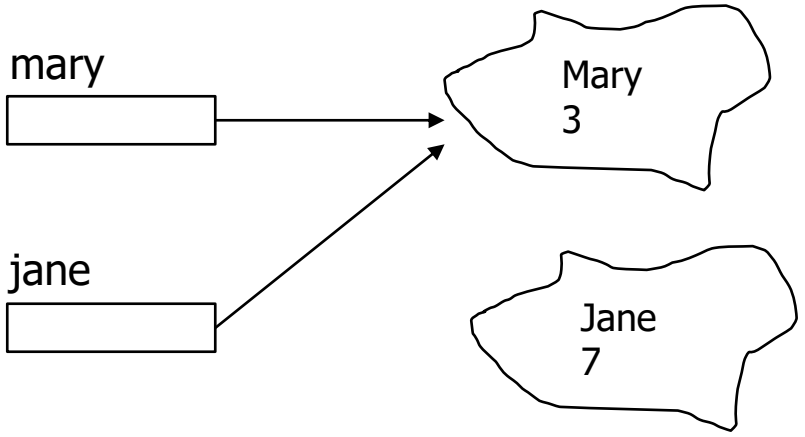
# Quiz

```
int x = 3;
int y = 7;
y = x;

Worker mary = new Worker("Mary", 3);
Worker jane = new Worker("Jane", 7);
jane = mary;


jane.id = 33;

System.out.println(mary.id);
```



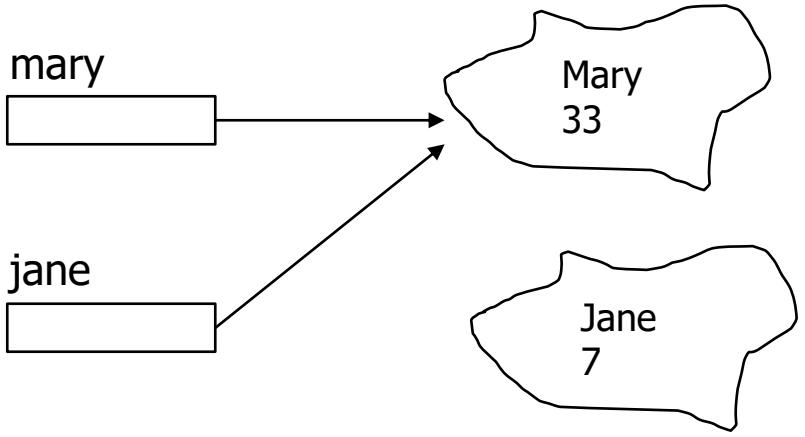
# Quiz

```
int x = 3;
int y = 7;
y = x;

Worker mary = new Worker("Mary", 3);
Worker jane = new Worker("Jane", 7);
jane = mary;


jane.id = 33;

System.out.println(mary.id);
```



# Quiz

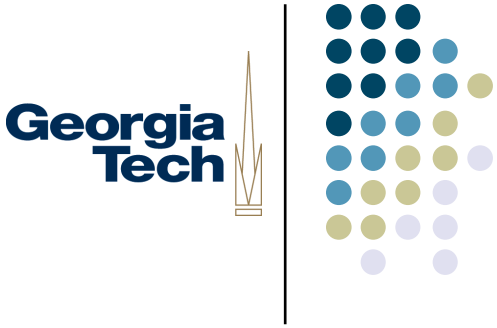
```
int x = 3;
int y = 7;
y = x;

Worker mary = new Worker("Mary", 3);
Worker jane = new Worker("Jane", 7);
jane = mary;


jane.id = 33;

System.out.println(mary.id);
```



# Example Program

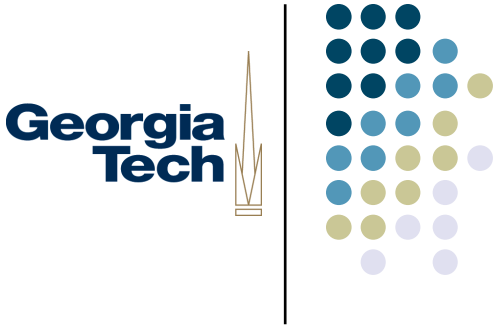
- RationalNumber
  - chap 6

# Learning Objectives



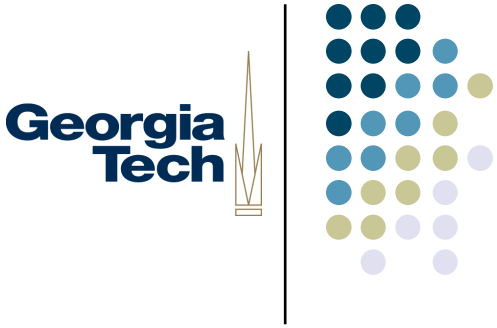

- Java classes and objects
  - Instance data
  - Methods
  - Constrcutors
  - Visibility
  - Scope
  - Static

# HW 5



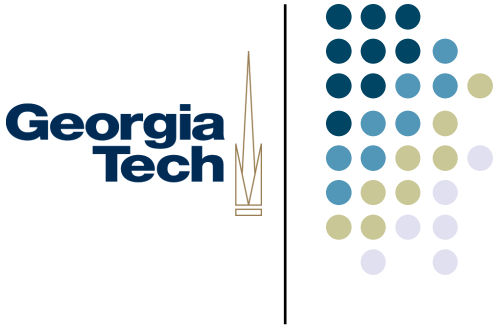

- Coming later tonight
- Will require a few classes
- Due Tuesday @ 1:30pm
  - Sakshi and I aren't here next week

# Java Books




- Come with me

# Next Time




- More with classes and OOP
  - inheritance & hierarchies
  - interfaces
  - abstract classes
  - dynamic binding