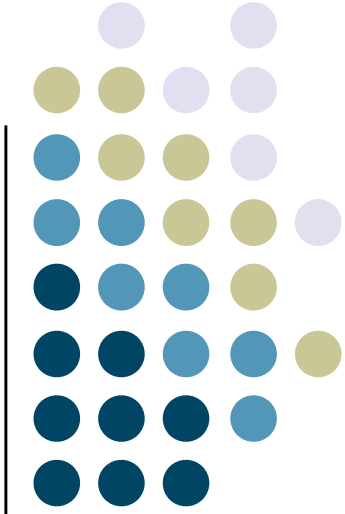
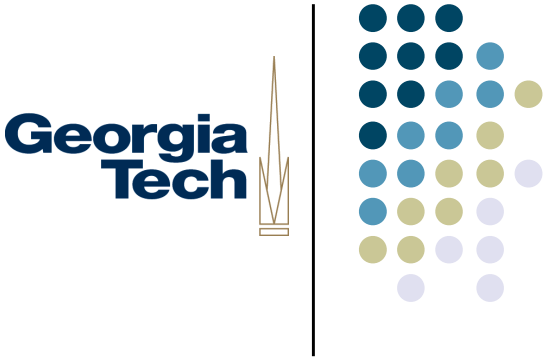
Java Object-oriented Programming 2



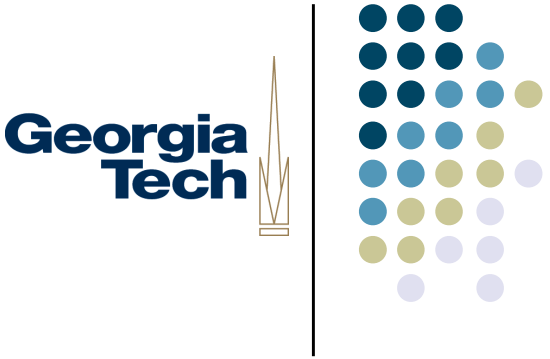
Georgia Tech

Learning Objectives

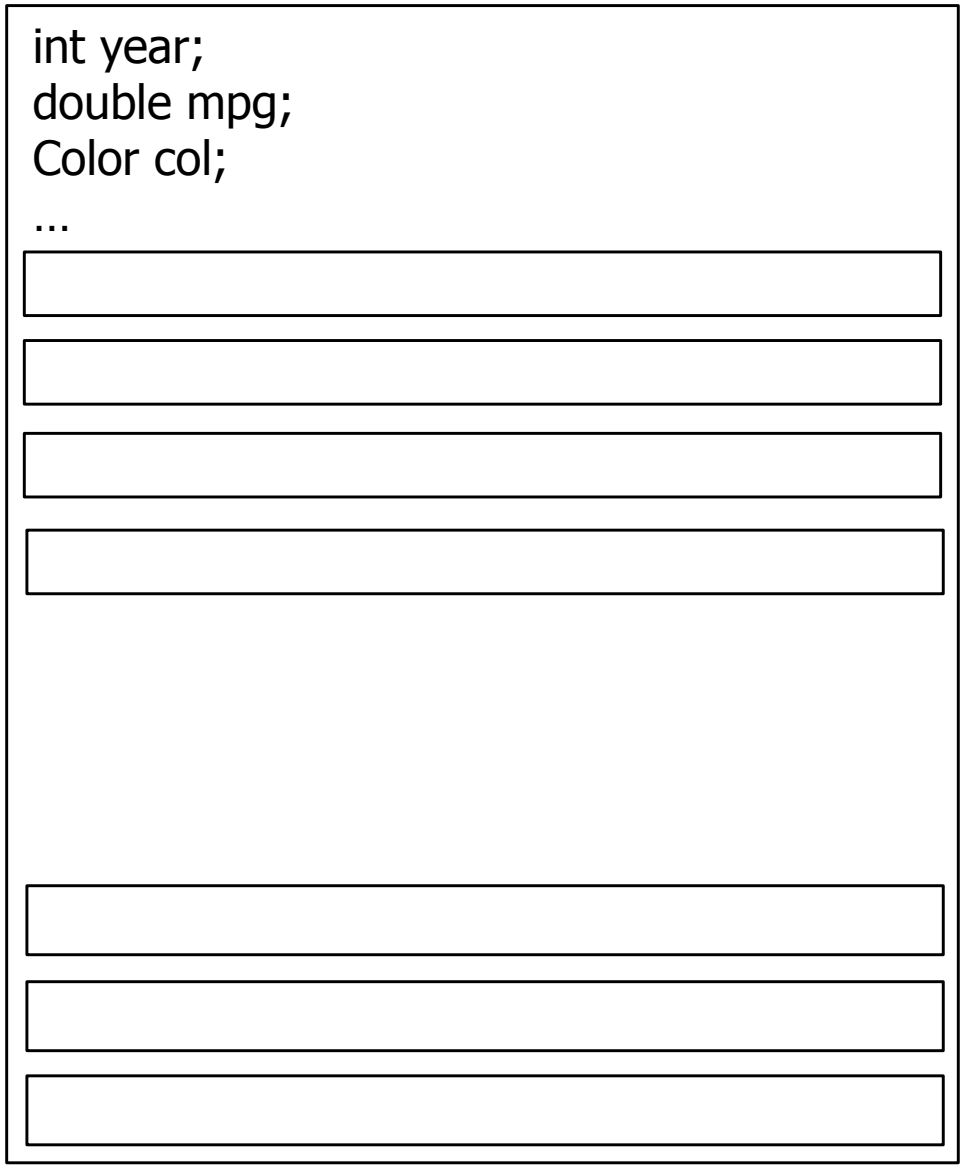


- Java
  - Class design
  - Inheritance
  - Abstract classes
  - Object class
  - Interfaces
  - Dynamic binding

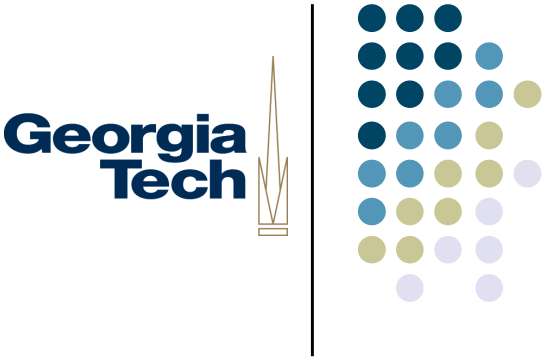
<Review>



class Car


int year;
double mpg;
Color col;
...
instance data
client
interface
externally used methods
internally used methods

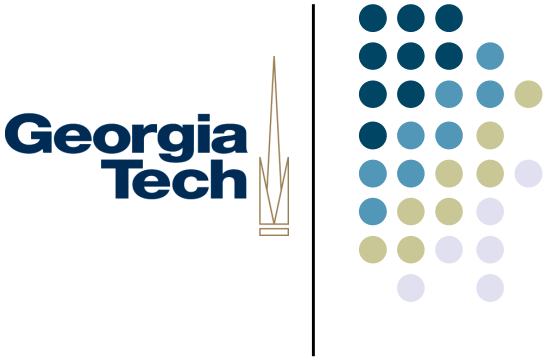
Access



	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

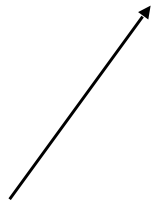
"Modifier" method

Constructor

Internal method

Methods

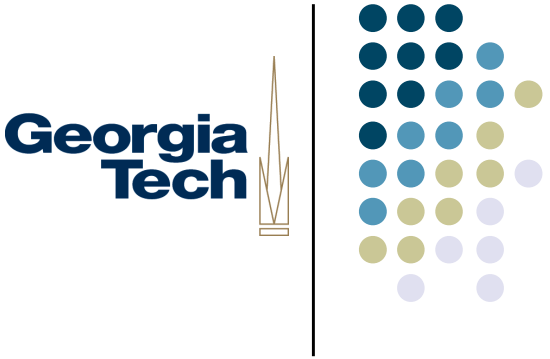
```
public double drive(int time) {
    double distance;

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



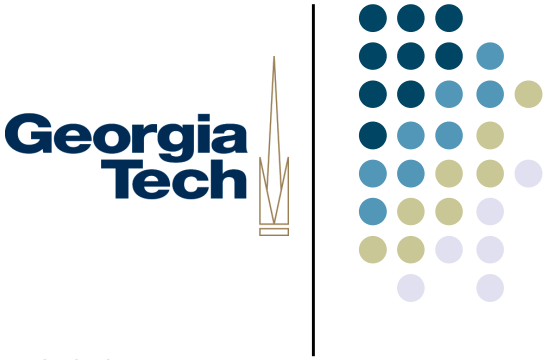
- Nothing in front of `speed`
- Which speed?

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

</Review>



Special Method

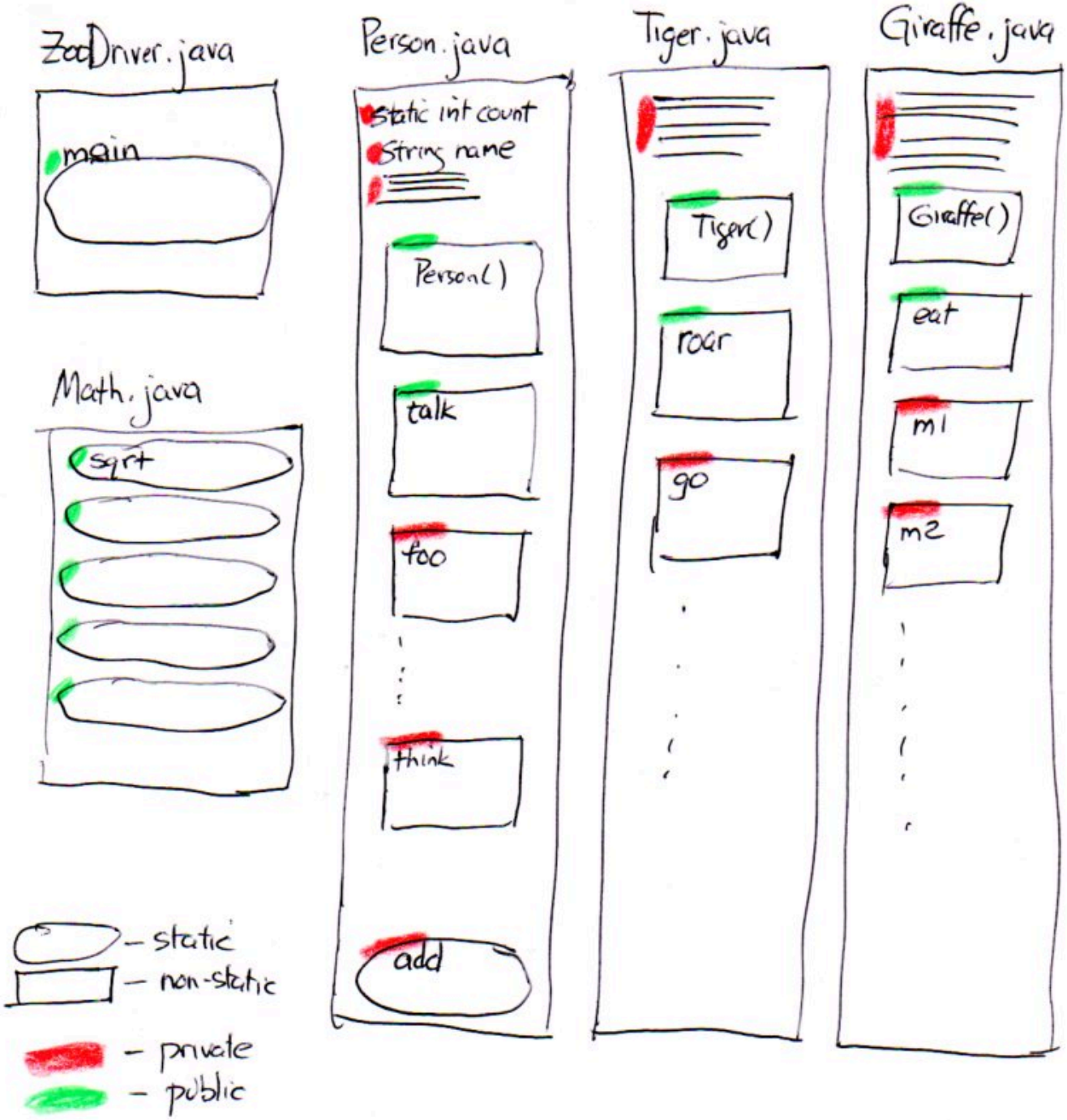


```
Car c1 = new Car(2004, 3247613237, "Audi", "5000");
System.out.println(c1);
```

- When Java tries to print an object, it automatically calls `toString()` method

```
// in Car class
public String toString() {
    return vin + " " + make + " " + model;
}
```

- What is printed is up to you



Sample program layout

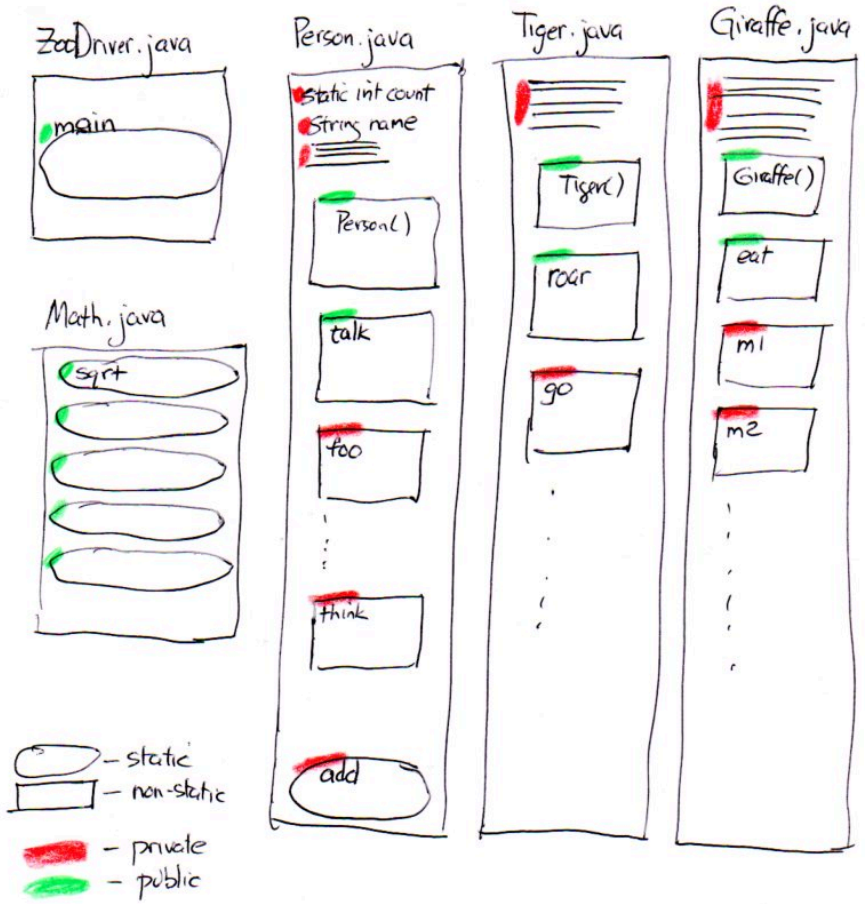
Suppose ZooDriver's main is

```
main(--) {
    Person p = new Person();
    Tiger t = new Tiger();
    t.roar();
```

and Tiger's roar() is

```
roar() {
    go();
```

Why no object in front of go()?



In Person's talk() method, can you
- access name
- access count
- call foo()
- call add()

In Person's static add() method, can you
- access name
- access count
- call foo()
- call add()

Inheritance

- Process of deriving a new class from an existing one
- Automatically contains some or all of the methods of original
- Can add new methods too
- Child can have only one parent class

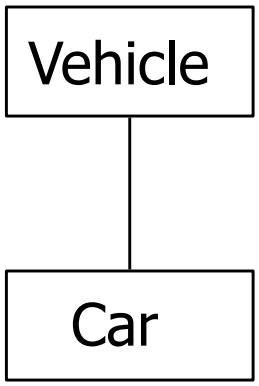
Inheritance

parent/super/base class

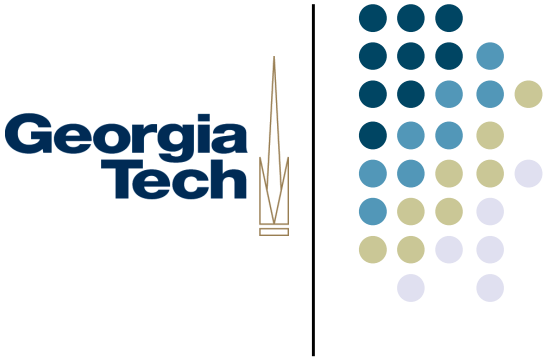
child/subclass

keyword `extends`

is-a hierarchy



Inheritance



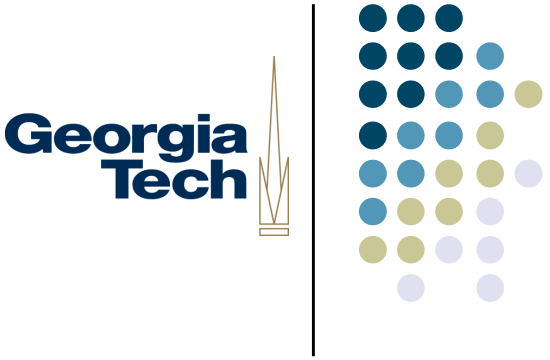

```
public class Vehicle {
   …
}
```

```
public class Car extends Vehicle {
   …
  // Car is-a vehicle
}
```

Vehicle can't access data or methods of Car

```
Car c2 = new Car();  // don't have to instantiate Vehicle
                     // you get Vehicle data & methods too
```

Inheritance

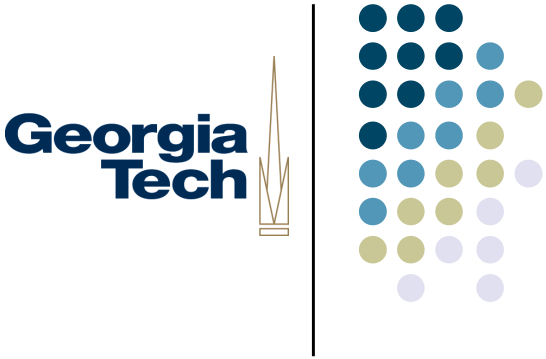


Child class can access public data/methods of parent
Child class cannot access public data/methods of parent
So???

`protected` – Can be accessed by child class but not outside classes

Child class can access parent's instance data and call parent's methods without qualification.
It's like they are yours. They actually are!

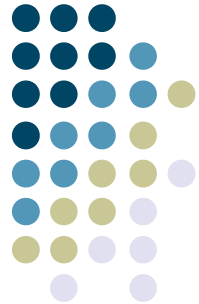
Example



```
public class Person {
   protected int ssn;
   protected String name;
   …
}
```

```
public class GTStudent extends Person {
   protected int gt_id;
   protected int credithrs;
   …

   public void m1(int num) {
      if (num == ssn)
        …
}
```

Inheritance

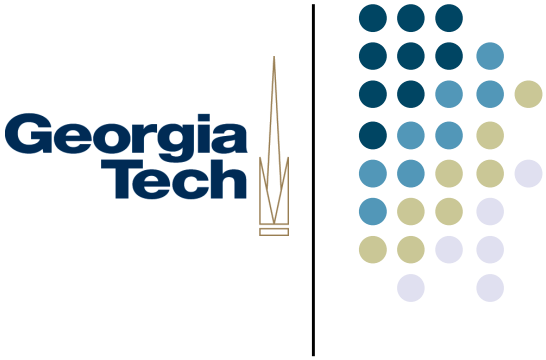


- Overriding – When child class defines a method with same name as one in the parent, child's version overrides the parent's

```
public class Person {
   protected int ssn;
   protected String name;

   public void talk() {
      …
   }
}
```

```
public class GTStudent extends Person {
   protected int gt_id;
   protected int credithrs;
   …

   public void talk() {
      …
   }
}
```

Inheritance



- Quiz
  - Can you do

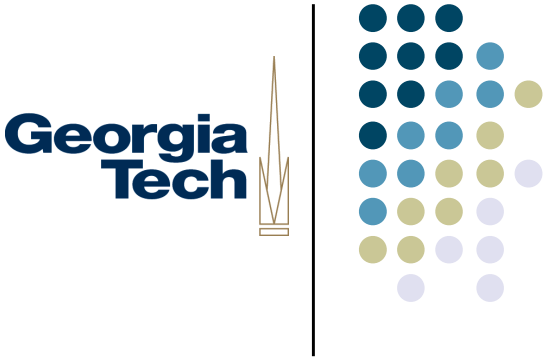
```
Vehicle v1 = new Car();
Person p1 = new GTStudent();
```

  - Yes! (remember is-a)
  - Can you do

```
Car c1 = new Vehicle();
GTStudent g1 = new Person();
```

  - No! (not necessarily true)

Inheritance



- Quiz

```
Person p1 = new GTStudent();
p1.talk();
```

Which talk() method is done, Person's or GTStudent's?

GTStudent's
That's really what's inside the p1 reference

Dynamic binding – Which method to perform is determined dynamically at run-time

Abstract Class

- Vehicle class we showed before might be abstract
  - Not real, ie, you never really make one

```
public abstract class Vehicle {
   protected int year;
   protected Color col;

   public abstract void drive();     // Abstract method
}
```

Abstract method – Subclasses must provide (override) it or be abstract themselves

Abstract Class

- Can abstract class have non-abstract methods?
  - Yes!

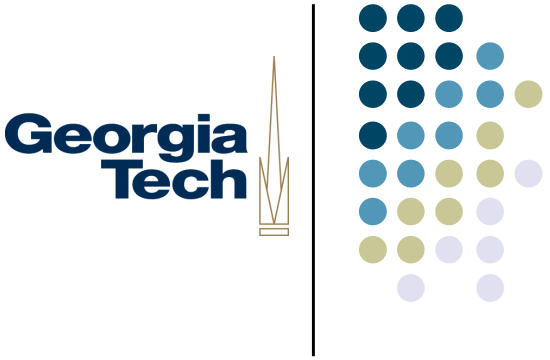
```
public abstract class Vehicle {
   protected int year;
   protected Color col;

   public abstract void drive();    // Abstract method

   public int getYear() {
       return year;
   }
}
```

Abstract class

- Abstract classes cannot be instantiated with `new()`

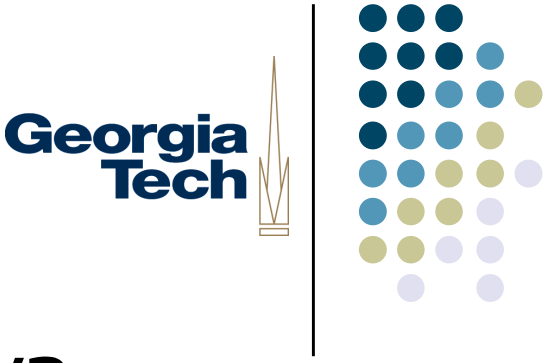
```
Vehicle v1 = new Vehicle();    // Compile error
```

Object class



- Object
  - In java, all classes ultimately derived from it
  - What's in it? Look in API
  - Not an abstract class, a real one

Interface

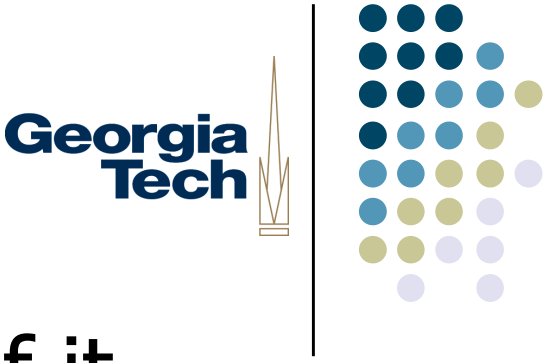


- Different type of construct in Java
  - Not a class
- Set of abstract methods and constants

```
public interface Bank {
   public void deposit(double dep);
   public double withdrawl(double wd);
   public void audit();
}
```

All goes in `Bank.java`

Interface

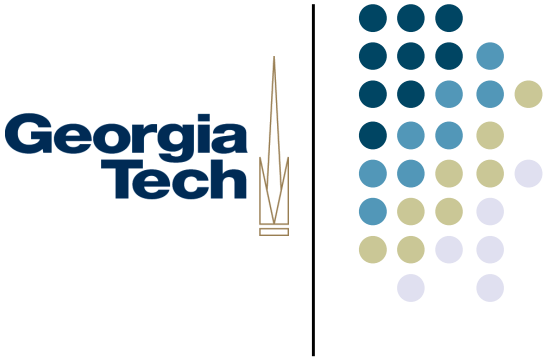


- A class implements an interface if it provides all the methods of the interface
  - Can have other methods too

```
public class MyMutual implements Bank { … }
```

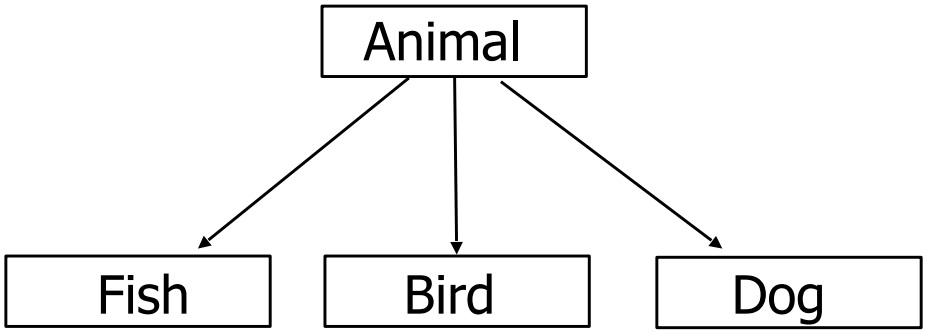
- Interface is a contract or specification
  - Lists set of services
  - If class provides those services, then it implements interface
  - Class can implement more than one interface

Interface



- What for?
  - If a class you want to use implements an interface, you know it has to provide those methods
  - Kind of a "guarantee"

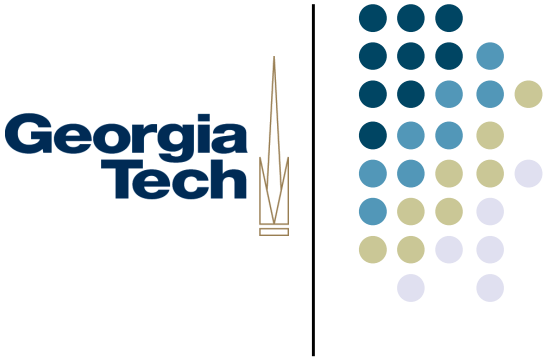
Scenario



Pet Store program

Animal is an abstract class
Others are real

Scenario



```
public abstract class Animal {
   public void makeNoise() {
     System.out.println("I'm an animal");
   }
}
```

```
public class Fish extends Animal {
   public void makeNoise() {
     System.out.println("Glug glug");
   }
}
```

```
public class Bird extends Animal {
   public void makeNoise() {
     System.out.println("Tweet tweet");
   }
}
```

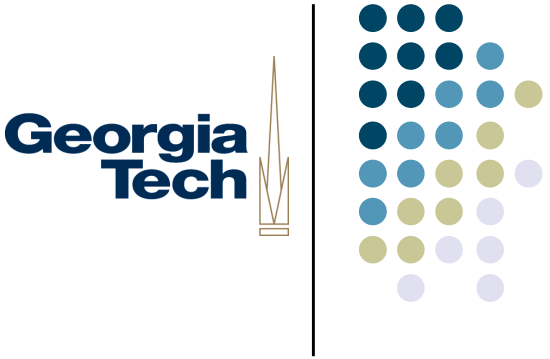
```
public class Dog extends Animal {
   public void makeNoise() {
     System.out.println("Woof woof");
   }

   public void bark() {
     System.out.println("Arf arf");
   }
}
```

Four classes

Scenario

- Want to create a data structure to hold a bunch of these different animals
  - How to do it? What to use?
  - An array – But how?
  - `Animal[] a = new Animal[100];`

Scenario



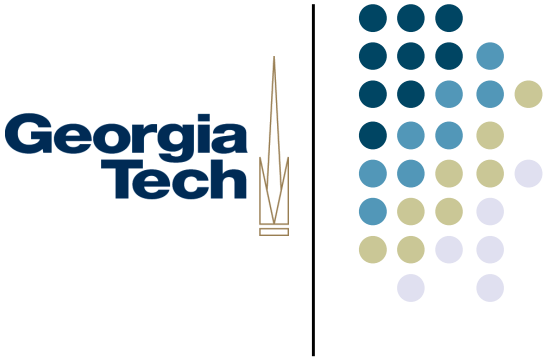
- Can you do

```
a[0] = new Fish();
```

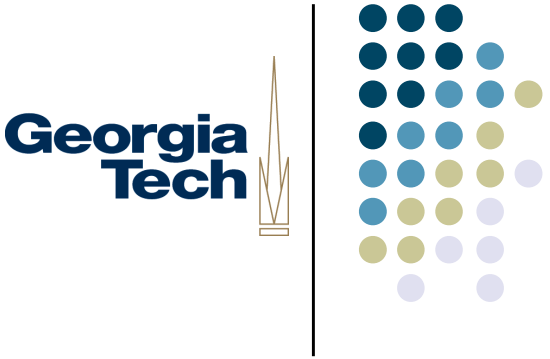
- Yes! A Fish is an Animal
- Same for

```
a[1] = new Bird();
a[2] = new Dog();
```

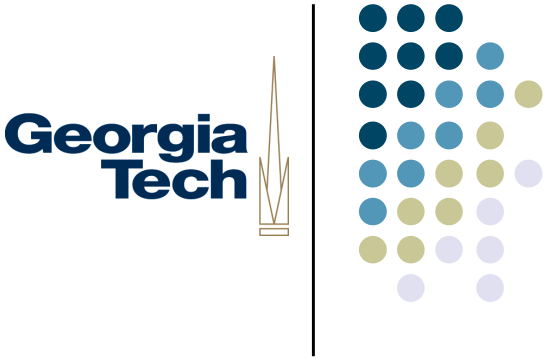
```
public class Driver
{
   public static void main(String[] args) {
     Animal[] a = new Animal[100];
     a[0] = new Bird();
     a[1] = new Fish();
     a[2] = new Dog();
     // …
     // We want to walk through them all
     // and have them make their noise









}
```



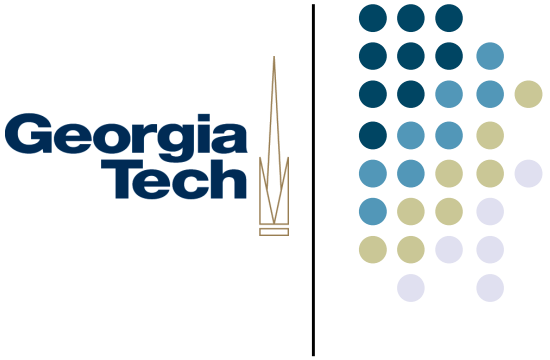
```
public class Driver
{
   public static void main(String[] args) {
     Animal[] a = new Animal[100];
     a[0] = new Bird();
     a[1] = new Fish();
     a[2] = new Dog();
     // …
     // We want to walk through them all
     // and have them make their noise

     for (int i=0; i<a.length; i++) {
        if (a[i] is a Bird) //then

        else if (a[i] is a Fish) //then
     }
     // Will that work?

     // No!




}
```



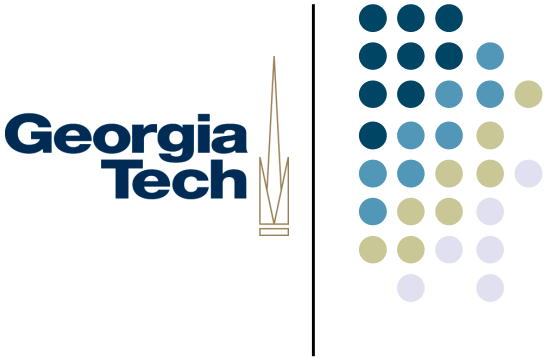
```
public class Driver
{
   public static void main(String[] args) {
     Animal[] a = new Animal[100];
     a[0] = new Bird();
     a[1] = new Fish();
     a[2] = new Dog();
     // …
     // We want to walk through them all
     // and have them make their noise

     for (int i=0; i<a.length; i++) {
        a[i].makeNoise();
     }

     // Will that work?

     // Yes!!!




}
```



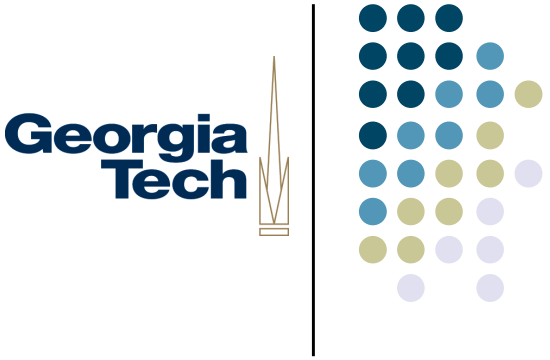
```
public class Driver
{
   public static void main(String[] args) {
     Animal[] a = new Animal[100];
     a[0] = new Bird();
     a[1] = new Fish();
     a[2] = new Dog();
     // …
     // We want to walk through them all
     // and have them make their noise

     for (int i=0; i<a.length; i++) {
        a[i].makeNoise();
     }

     // What if we wanted the Dog ones to bark() too?

     // Can we do
     a[i].bark();

     // No!!!!   Not all a[i] are Dogs




}
```



```
public class Driver
{
   public static void main(String[] args) {
     Animal[] a = new Animal[100];
     a[0] = new Bird();
     a[1] = new Fish();
     a[2] = new Dog();
     // …
     // We want to walk through them all
     // and have them make their noise

     for (int i=0; i<a.length; i++) {
        a[i].makeNoise();
     }

     // What if we wanted the Dog ones to bark() too?

     Dog d;
     if (a[i] instanceof Dog) {
        d = (Dog) a[i];
        d.bark();
     }


}
```



Learning Objectives



- Java
  - Class design
  - Inheritance
  - Abstract classes
  - Object class
  - Interfaces
  - Dynamic binding