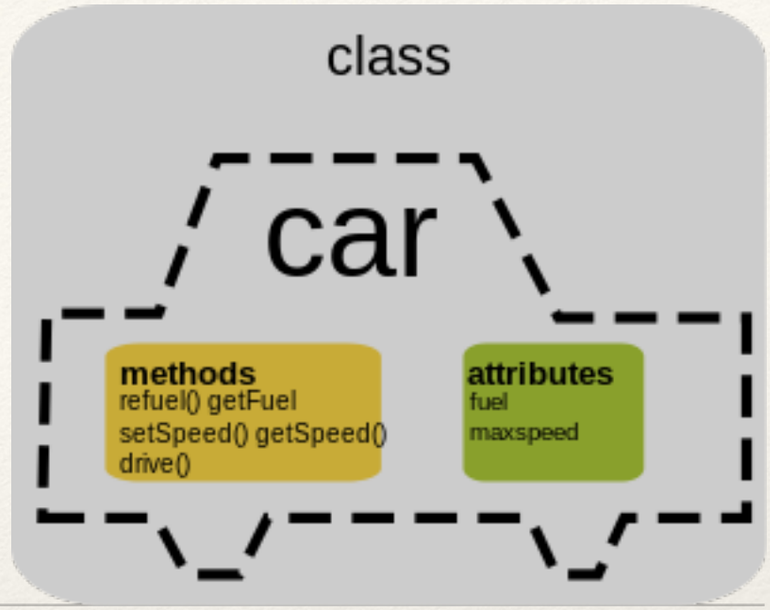
Dr. Sarah Abraham
University of Texas at Austin
Computer Science Department



Components and Inheritance

Elements of Graphics
CS324e

Object Review

- Objects have fields and methods
  - Fields are attributes of that object
  - Methods are functions of that object
- Objects can be fields of other objects

Composite Objects

- Objects that include other objects
  - Build higher levels of abstraction
  - Create greater modularity
- Component-based design allows for an object to be composed of other object instances with desired functionality
- Composition is a “has-a” relationship

Component-based Example

- Components of a Bike object?
- Potential components:
  - Frame
  - Wheels
  - Brakes
  - Drivetrain
  - Handlebars

Consider: Animating a Bike

- To animate:
  - Bike must move
  - Wheels must rotate
- Wheels have the same visual appearance

Theoretical Bike Display

```
//displayBike draws the entire bike

//x, y, and wheelDistance are bike fields

void displayBike() {
  displayFrame(x, y);
  frontWheel.display(x+wheelDistance, y);
  backWheel.display(x-wheelDistance, y);
}
```

Theoretical Bike Move

```
//moveBike moves the entire bike; wheels
rotate based on bike speed

//dx is delta x;

void moveBike(dx) {
  updatePosition(dx);
  frontWheel.rotateWheel(speed);
  backWheel.rotateWheel(speed);
}
```

What If We Have Multiple Types of Bike?

Inheritance

- A class can **inherit** fields and methods from another class
  - An object that inherits from another is a subclass (derived class)
    - The object it inherits from is the superclass (base class)
- A subclass **extends** a superclass
  - Contains all methods and fields of the superclass and more
- Inheritance is a “is-a” relationship

Inheritance in Java

- `class Subclass extends Superclass { }`
- Subclass declares any fields and methods **not** included in the Superclass
- Subclass constructor should call on Superclass constructor
  - `this` refers to an *instance* of a class type
  - `super` refers to to the parent (super) class

What about Re-declarations?

```
class Foo {
  ...
  void printHello() {
     print(“Hello,
  Foo”);
  }
}
```

```
class Bar extends Foo {
  ...
  void printHello() {
     print(“Hello,
  Bar”);
  }
}
```

What about Redeclarations?

- Consider:

```
Foo f = new Foo();
Bar b = new Bar();
f.printHello();
b.printHello();
```

What does `printHello()` do for `f` and `b`?

Why Use Inheritance?

- Inheritance allows for more generalized code
- A general class of behaviors can be extended to a group of more specialized subclasses
- A superclass method can be **overridden** in the subclass to create that specific behavior
- Superclass: Vehicle
- Subclasses: Car, Train, Ship, Plane etc

Vehicle Example

- Consider superclass `Vehicle` and subclasses `Car` and `Train`
- What is a method/field in the `Vehicle` class that would lend itself to use in both the `Car` and `Train` classes?
- What is a method/field in the `Train` class that the `Car` class wouldn't need?

ColorSpot Example

Hands-on: Building with Inheritance

- Today's activities:

1. Get `Spot` subclass, `ColorSpot`, working in your code
2. Create a `Spot` subclass, `TwoSpots`. `TwoSpots` displays two ellipses around a center point
3. Move a `TwoSpots` object across the canvas