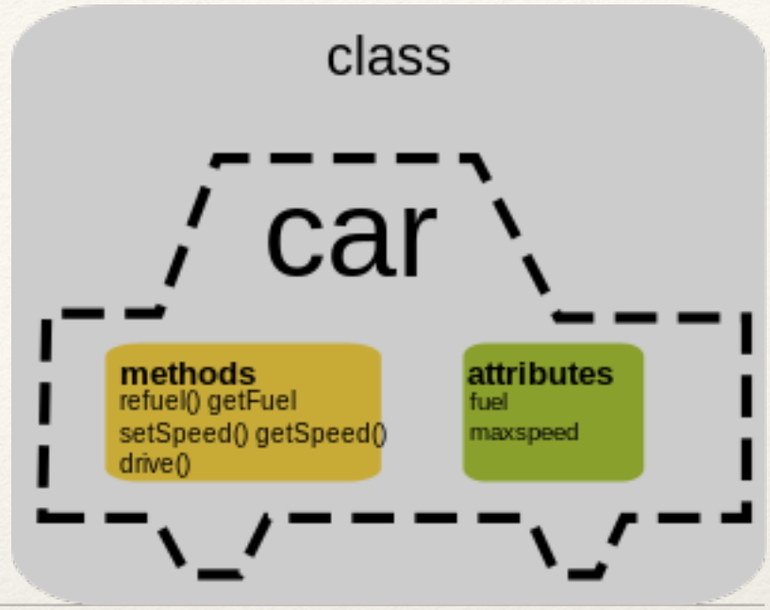
*Dr. Sarah Abraham*
*University of Texas at Austin*
*Computer Science Department*




# 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
//position and wheelDistance are bike fields
void displayBike() {
  frame.display(position);
  frontWheel.display(position.X+wheelDistance,
position.Y);
  backWheel.display(position.X-wheelDistance,
position.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 DerivedClass : BaseClass { }`
- Derived declares any fields and methods **not** included in the BaseClass
- DerivedClass constructor can call on BaseClass constructor
  - `this` refers to an *instance* of a class type
  - `base` refers to to the parent (base) class

# What about Re-declarations?

```
class Foo {

   ...

   void printHello() {
      Console.WriteLine(“Hello, Foo”);
   }

}


class Bar : Foo {

   ...

   void printHello() {
      Console.WriteLine(“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
- Base class: Vehicle
- Derived classes: 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?

# Hands-on: Building with Inheritance

- Today's activities:

1. Create a `Spot` derived class, `TwoSpots`. `TwoSpots` displays two spots that are around a center point

2. Create a `TwoSpots` object that moves across the screen