Interfaces
Automobile Class

- Project 2 has been all about the basics of Object Oriented Design
- Let's talk about how to design an automobile class.
- See class exercise.
OF THE POTENTIAL RESPONSES TO MY BRAKES' FAILURE, I DID NOT CHOOSE THE BEST.

HELLO, YOU'RE ON CAR TALK.
Automobile API

- Our Automobile class has the following API:
  - Automobile(String make, String model, int year, String vehicleID)
  - String getMake()   car’s make
  - String getModel()  car’s model
  - int getYear()      car’s model year
  - String getVIN()    car’s vehicle ID
Automobile API

- The API for this (or any) class defines the public methods + constructors available for interacting with instances of the class

- Can use the Automobile API:
  - Automobile car = new Automobile("Toyota", "Camry", 2011, "1GH1234567");
  - System.out.println("car’s year is “+ car.getYear());
What about Trucks?

• Suppose we wanted to create a similar class, called “Truck” with the API:
  – Truck(String make, int truckClass, String vehicleID)  //avoiding “class”
  – String getMake()      truck’s make
  – int    getClass()     truck’s class (1 to 8)
  – String getVIN()       truck’s vehicle ID
Now we can interact with Trucks:

- Truck truck = new Truck("Toyota", 3, "1GH1234567");
- System.out.println("truck’s class is "+ truck.getClass());

There is some commonality here....
Capturing Common Attributes

- Cars and Trucks both have makes and VINs (Vehicle Identification Numbers).
- Java has a construct that allows us to capture this commonality.
- In Java, an `interface` provides us with a way to say that certain classes should share some common set of public methods.
Vehicle Interface

- We can use the following interface in this example:

```java
public interface Vehicle {
    String getMake();
    String getVIN();
};
```

- This declaration creates a new type called Vehicle
Interfaces

• We can now say:
  – An Automobile is-a Vehicle
  – A Truck is-a Vehicle

• This says that an Automobile can do all the actions specified by the Vehicle interface

• Since the getMake() and getVIN() actions define what a Vehicle is, this makes sense in the ordinary use of “is-a”

• Similarly, a Truck is a Vehicle
Classes implement Interfaces

public class Automobile implements Vehicle {
    //... just as before: fields, constructors, 
    // methods
}

public class Truck implements Vehicle {
    //Truck fields, constructor(s), methods
}
Rules for interfaces

• We can't do this:
  – Vehicle vehicle = new Vehicle(...);

• Interfaces never have constructors

• We can create Automobile/Truck instances and call them Vehicles

• That is we declares new instances of Automobiles and Trucks that have type Vehicle
Type Compatibility

• A Truck is-a Vehicle, so there is “type compatibility”:
  − Vehicle truck = new Truck(...);

• Now we can use whatever interface methods are available on auto and truck:
  − System.out.println(“ auto has make “ + auto.getMake());
Power of Interfaces

- This becomes useful if we want an array of vehicles (some might be autos, others trucks)
- We can't make a new Vehicle, but we can make a new array (or ArrayList) of Vehicles:
  - Vehicle[] vehicles = new Vehicle[10];
  - vehicles[0] = auto;
  - vehicle[1] = truck;
  - ...