INTRODUCTION TO OBJECT ORIENTED PROGRAMMING (OOP)

CS302 – Introduction to Programming
University of Wisconsin – Madison
Lecture 18

By Matthew Bernstein – matthewb@cs.wisc.edu

Purpose of Object Oriented Programming

- You have learned how to structure your programs by decomposing your tasks into methods
- This has made your code more modular and increases code re-use
- **Object Oriented Programming (OOP)** is a style of programming which further decomposes your code into discrete interacting objects

We Have Already Used Objects

- We already have some exposure to objects
- Examples:

```
// Create a new Scanner object
Scanner scan = new Scanner(System.in);

// Create a new Random object
Random rand = new Random();

// Create a new String object
String str = “Hi”;
```

Advantages

- By interacting with discrete objects, we don't need to deal with the underlying complexity within the objects
- For example, when retrieving user input from the keyboard, we do not have to interact with the hardware and deal with all of the event handling necessary to retrieve user input from the keyboard. Instead, we just have to create a Scanner object that does all of that for us.
- This design paradigm is called **encapsulation**

Philosophy of OOP

- Object Oriented Programming is based on the idea of instantiating objects that are of a certain **class**
- A class describes a set of **objects** that have the same behavior
- For example, all objects of the Scanner class all behave the same way
- In the following code:

```
Scanner scan = new Scanner(System.in);
```

The scan object is an **instantiation** of the Scanner class

Conceptual Example of Objects and Classes

- For example, let's say we implement a class called "Vehicle"
- We can create new objects of class Vehicle
- Hypothetical example:

```
Vehicle myCar;
Vehicle momsCar;
```

- Both myCar and momsCar are different objects of the class Vehicle

A Class Defines Each Object's Interface

- Each object has a **public interface** that consists of all methods and variables that are accessible to the user of this object
- This interface is defined by the object's class
- For example, the Scanner's public interface includes the methods **nextInt()**, **hasNextInt()**, **nextLine()**, etc.
- Where are these methods defined? In the Scanner class
- Thus, all **instantiations** of the Scanner class have these methods available to the user

Implementing a Simple Class

- **Each class we write for now must be placed into a .java file of the same name as the class**
- Let's create a class called Car that has just one **instance variable** and two **public methods**

Car Class

```
class Car
{
        // Instance variable
        public int milage;

        // Public method can be accessed by the user of this class
        public void drive(int miles)
        {
                milage += miles;
        }

        // Public method can be accessed by the user of this class
        public int getMilage()
        {
                return milage;
        }
}
```

Instance Variables

- An object stores its data in **instance variables**
- These are also called **fields**
- An instance variable is a storage location that is present in each object of the class
- The value of one object's instance variables may be different from the values of another object's variables even though they are of the same class

Instance Variables

- Instance variables are written inside of the class's code block outside of any method
- Following standard convention, all instance variables should be declared at the top of the class
- Example:

```
class Car
{
        // Instance variables
        private String make;
        private String model;
        private int mileage;
}
```

Declaring Instance Variables

- An instance variable declaration takes the following form:

```
modifier type variable_name;
```

- Example:

```
private String name;
```

Modifiers

- An **access modifier** stipulates whether a field or method can be accessed or called by the user of the class
- Fields/methods denoted as **private** can only be accessed by methods *within* the class
- Fields/methods denoted as **public** can be accessed by methods *either* within the class or outside the class

Composition: Objects as Instance Variables

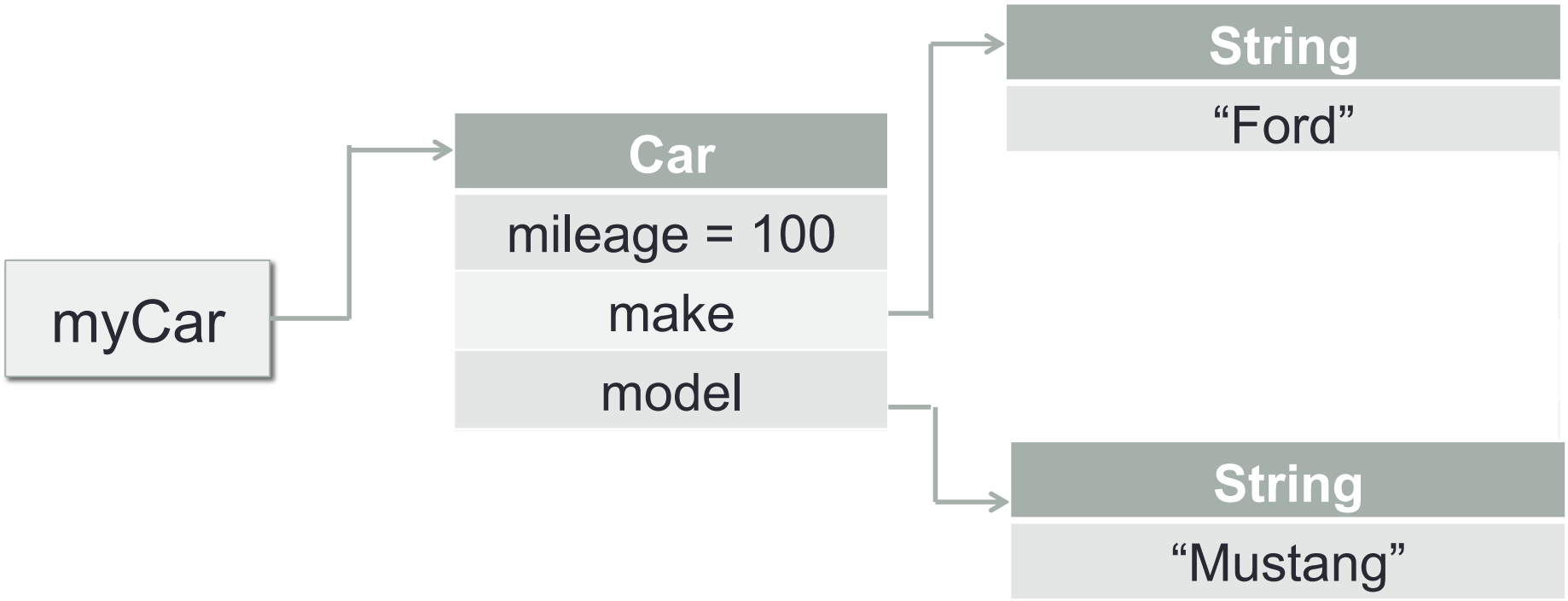
- You may have noticed in the example:

```
class Car
{
        // Instance variables
        private String make;
        private String model;
        private int mileage;
}
```

That we can use objects as instance variables (String is an object). This is called **Composition**

Memory Diagram of Objects

Car myCar = new Car();



More on Instance Variables

- Nearly all of class's instance variables should be **private**
- If we think of an object as a machine, the instance variables represent the gears. We don't want to expose the gears to the user of the machine.
- Instead, if we want to allow the user access to the instance variables, we provide public methods for accessing these instance variables.

Instance Methods

- Similar to instance variables, objects of the same class have the same **instance methods**
- Instance methods are method members that use the object's instance variables
- The code for an object's instance methods are defined in the object's class
- An object's instance methods have access to ALL of its instance variables

Instance Method

```
class Car
{
        // Instance variables
        private String make;
        private String model;
        private int mileage;

        // Instance method
        public void printFullName
        {
                System.out.print(make + " " + model);
        }
}
```

this

- In order for an object to refer to itself, Java provides a reserved word, **this**, that is a reference variable pointing to itself
- A class's instance variable can be referenced as a field of the **this** variable
- The **this** variable's primary use is when an instance variable is overshadowed by a method's parameter

When to use **this**

- This is primarily used when an object's instance variable name is overshadowed by a method's parameter name
- Example:

```
class Car
{
        private String make;

        public void setMake(String make)
        {
                // We set the instance variable
                // "make" to the parameter "make"
                this.make = make;
        }
}
```

Cool CS Link of the Day

- Self Assembling Robots Project at MIT
- https://www.youtube.com/watch?v=6aZbJS6LZbs