CMPINF0401 Recitation

TUESDAYS 11:00-12:50

MICHAEL BARTLETT

Overview

- Object Oriented Programming (OOP)
- Lab 9

OOP: What’s an object?

- Technically speaking, an object is a bundle of state and behavior:
  - State: The data contained in the object (the object’s fields)
  - Behavior: The actions supported by the object (its methods)

OOP: What’s a Class?

- Every object has a class
  - A class defines the object’s methods and fields
- A class defines both type and implementation
  - Type → Where the object can be used (What datatypes is the constructor going to accept?)
  - Implementation → How the object does things (its methods)

OOP: Class vs. Objects Example

- Example of a class:
  - Fruit
  - Car
- Example of corresponding objects:
  - Apple, Banana, Mango
  - Volvo, Audi, Toyota
- So, a class is a template for objects, and an object is an instance of a class

OOP: Class vs. Objects Example

- When we create individual objects, they inherit all the variables and methods from the class
  - We’ve seen this with Strings:
    - String name = “Michael”
      - name is a String Object where String is a class
    - name = name.toUpperCase();
      - Now, name == ”MICHAEL”
      - name was able to use the predefined method for the String object

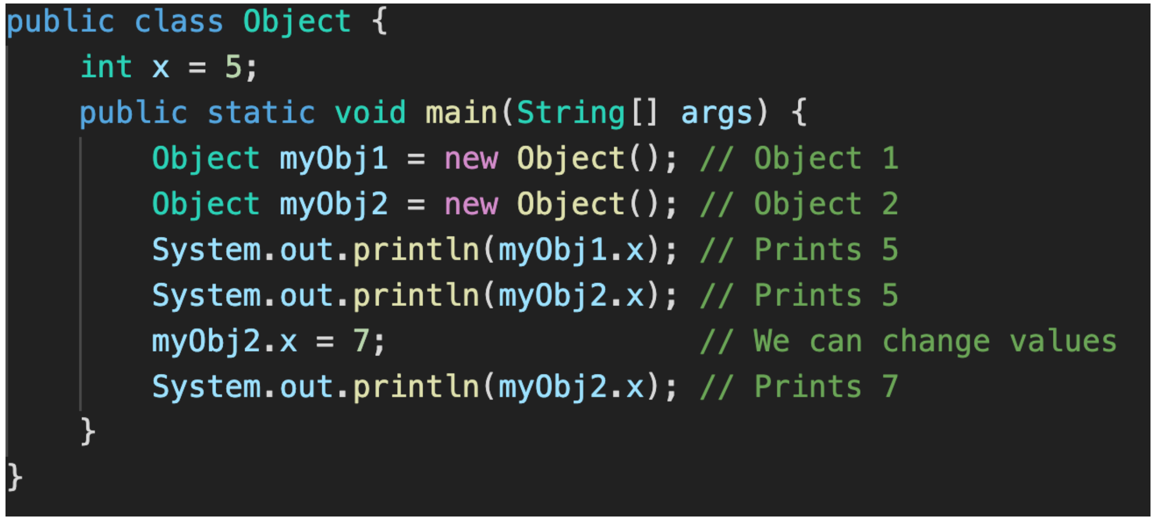
OOP: Interfaces vs Classes

- Interfaces can be used to define the methods that the class must contain
  - For example, an interface might define a Car like this:
    - interface Car { public String getColor();}
  - The class for a type of Car could look like this:

    -
      ```
      class Volkswagen implements Car
      {
          String color = “blue”; // Normally done with an init method
          public String getColor()
          {
                  return(this.color);
          }
      }
      ```

How do we make objects?

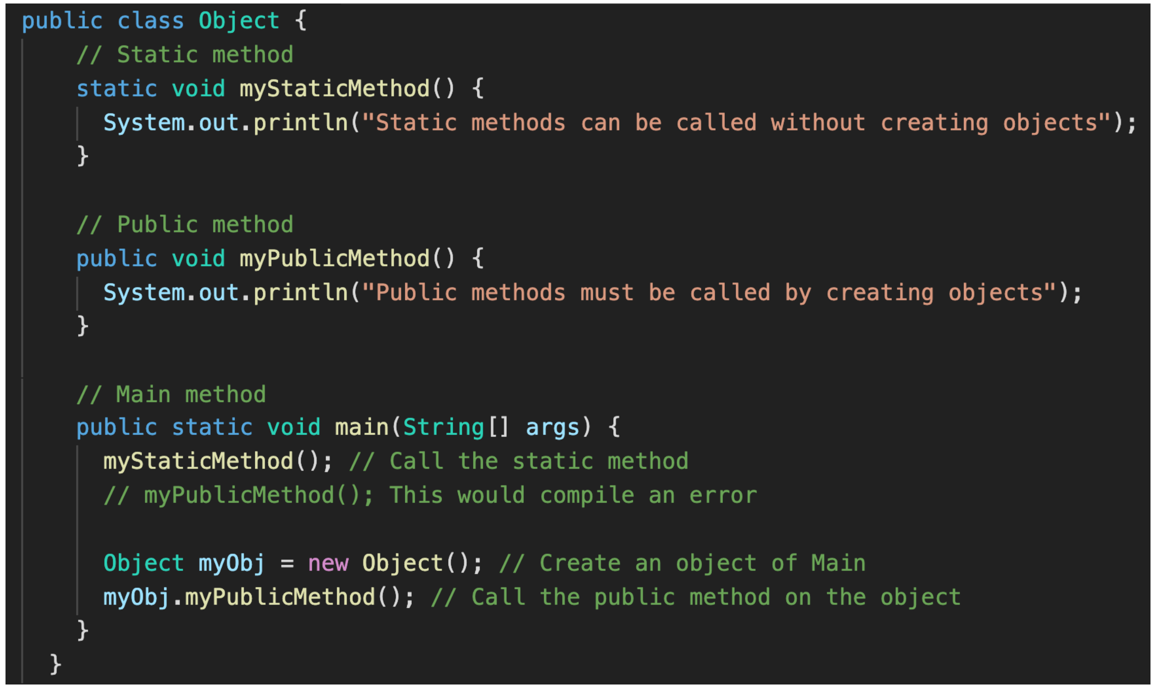
- We saw the previous String example, but how do we define our own and use them?



```java
public class Object {
    int x = 5;
    public static void main(String[] args) {
        Object myObj1 = new Object(); // Object 1
        Object myObj2 = new Object(); // Object 2
        System.out.println(myObj1.x); // Prints 5
        System.out.println(myObj2.x); // Prints 5
        myObj2.x = 7;                 // We can change values
        System.out.println(myObj2.x); // Prints 7
    }
}
```

Classes can have methods

- As shown before, we know that Classes can have methods. There are two types, static and public.
- Static can be accessed without creating an object of the class, public needs an instance of the class to be created.

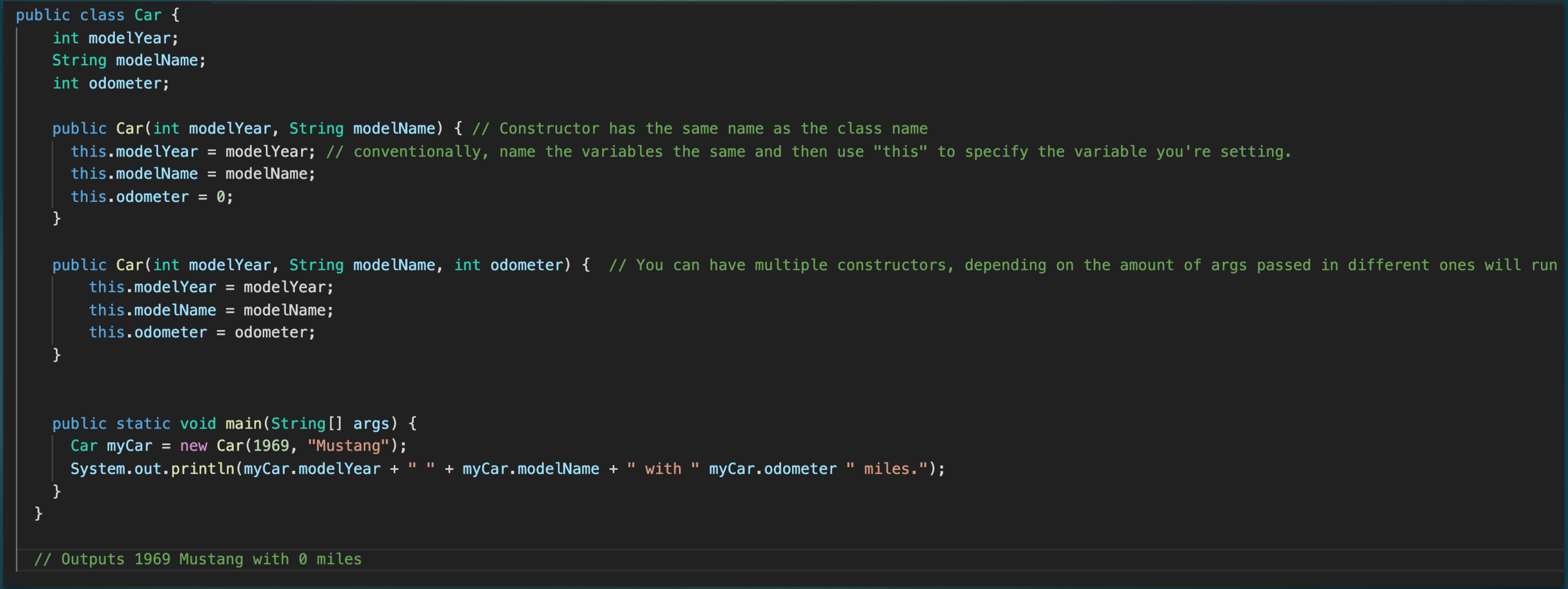


```
public class Object {
    // Static method
    static void myStaticMethod() {
      System.out.println("Static methods can be called without creating objects");
    }

    // Public method
    public void myPublicMethod() {
      System.out.println("Public methods must be called by creating objects");
    }

    // Main method
    public static void main(String[] args) {
      myStaticMethod(); // Call the static method
      // myPublicMethod(); This would compile an error

      Object myObj = new Object(); // Create an object of Main
      myObj.myPublicMethod(); // Call the public method on the object
    }
  }
```

Constructors

- A constructor is a special method that is used to initialize objects.
- The constructor is called when an object of a class is created.
- It can be used to set initial values for object attributes

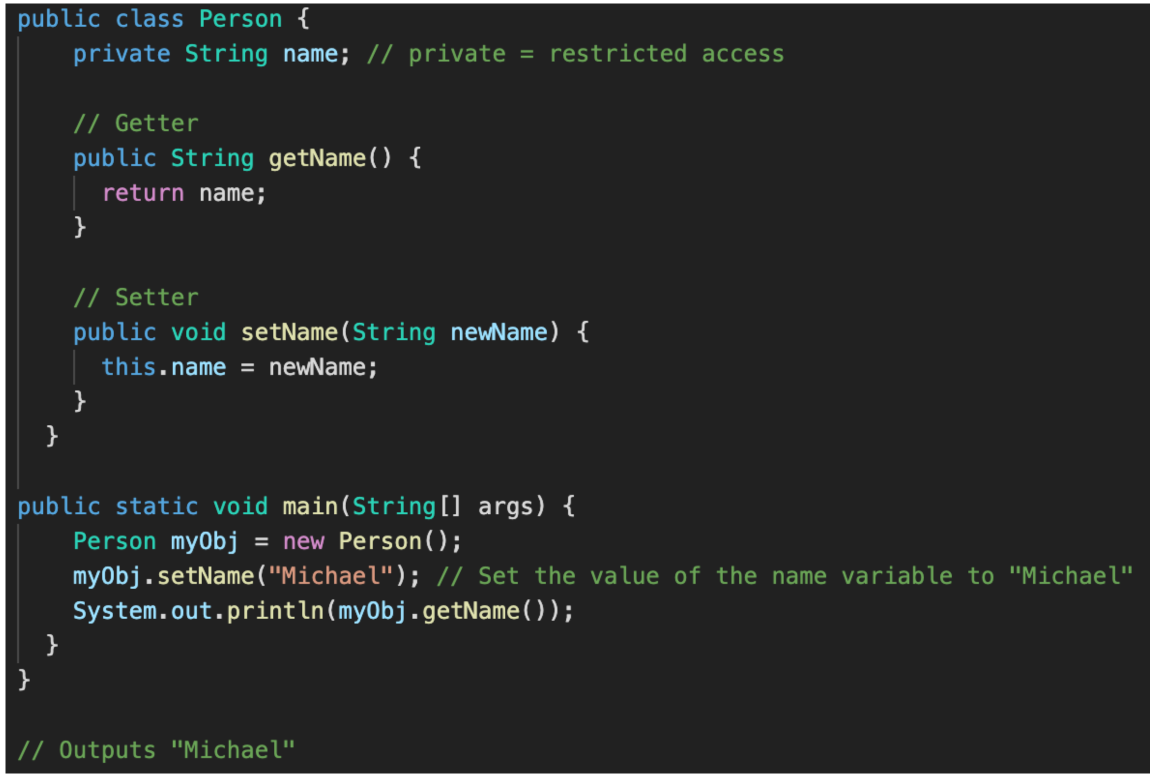
Constructors Example



```
public class Car {
    int modelYear;
    String modelName;
    int odometer;

    public Car(int modelYear, String modelName) { // Constructor has the same name as the class name
      this.modelYear = modelYear; // conventionally, name the variables the same and then use "this" to specify the variable you're setting.
      this.modelName = modelName;
      this.odometer = 0;
    }

    public Car(int modelYear, String modelName, int odometer) {  // You can have multiple constructors, depending on the amount of args passed in different ones will run
        this.modelYear = modelYear;
        this.modelName = modelName;
        this.odometer = odometer;
    }


    public static void main(String[] args) {
      Car myCar = new Car(1969, "Mustang");
      System.out.println(myCar.modelYear + " " + myCar.modelName + " with " myCar.odometer " miles.");
    }
  }

  // Outputs 1969 Mustang with 0 miles
```

Encapsulation

- Encapsulation: make sure that “sensitive” data is hidden from users. To do this:
  - Declare class variables/attributes as private
  - Provide public get and set methods to access and update the value of a private variable.



```java
public class Person {
    private String name; // private = restricted access

    // Getter
    public String getName() {
      return name;
    }

    // Setter
    public void setName(String newName) {
      this.name = newName;
    }
  }

public static void main(String[] args) {
    Person myObj = new Person();
    myObj.setName("Michael"); // Set the value of the name variable to "Michael"
    System.out.println(myObj.getName());
  }
}

// Outputs "Michael"
```

Why Encapsulation?

- Better control of class attributes and methods
- Class attributes can be read-only (if they can only be accessed through a get method), or write-only (if they can only be accessed through a set method).
- Flexible: The programmer can change one part of the code without affecting others
- Increased Security: hide data that shouldn’t be available to all users, test to see if the value being set is valid (i.e., a Person’s weight shouldn’t be 0)

Lab 9

- Due 4/4
  - https://canvas.pitt.edu/courses/127916/files/8050409?module_item_id=2735328

Lab 9

- Vehicle class:
  - Add a private attribute for the ‘make’ of the car.
  - Should use encapsulation which means that you need getter (return make) and setter (store make as an uppercase String) methods for it
- Fleet class:
  - Get rid of the hardcoded values for car1 and car2.
  - Make car1 by using the BASE CONSTRUCTOR (no args)
    - Use a Scanner to prompt the user for one variable at a time and use car1’s setters to set the variables
  - Make car2 by using the OVERLOAD CONSTRUCTOR (provide args)
    - Get values from the user with a Scanner (store as variables in your main), and then pass those variables into the constructor
- Call .stats for both cars and screenshot it as part of your submission