Middlebury

CSCI 201: Data Structures

Spring 2025

Lecture 2W: Objects

Goals for today:

- Start styling our code nicely.
- Create objects using the `new` keyword.
- Define **constructors** which are called when creating objects.
- Reference the current object using the `this` keyword.
- Decide whether member variables (fields) or methods should be declared `public` or `private`.
- Use the `.` "dot" operator to access member variables (fields) or call methods.
- Write **setter** and **getter** methods.
- Make some member variables (fields) **mutable** or **immutable**.
- Reinforce the decision about whether to make a method `static`.

Let's set up a few style guidelines.

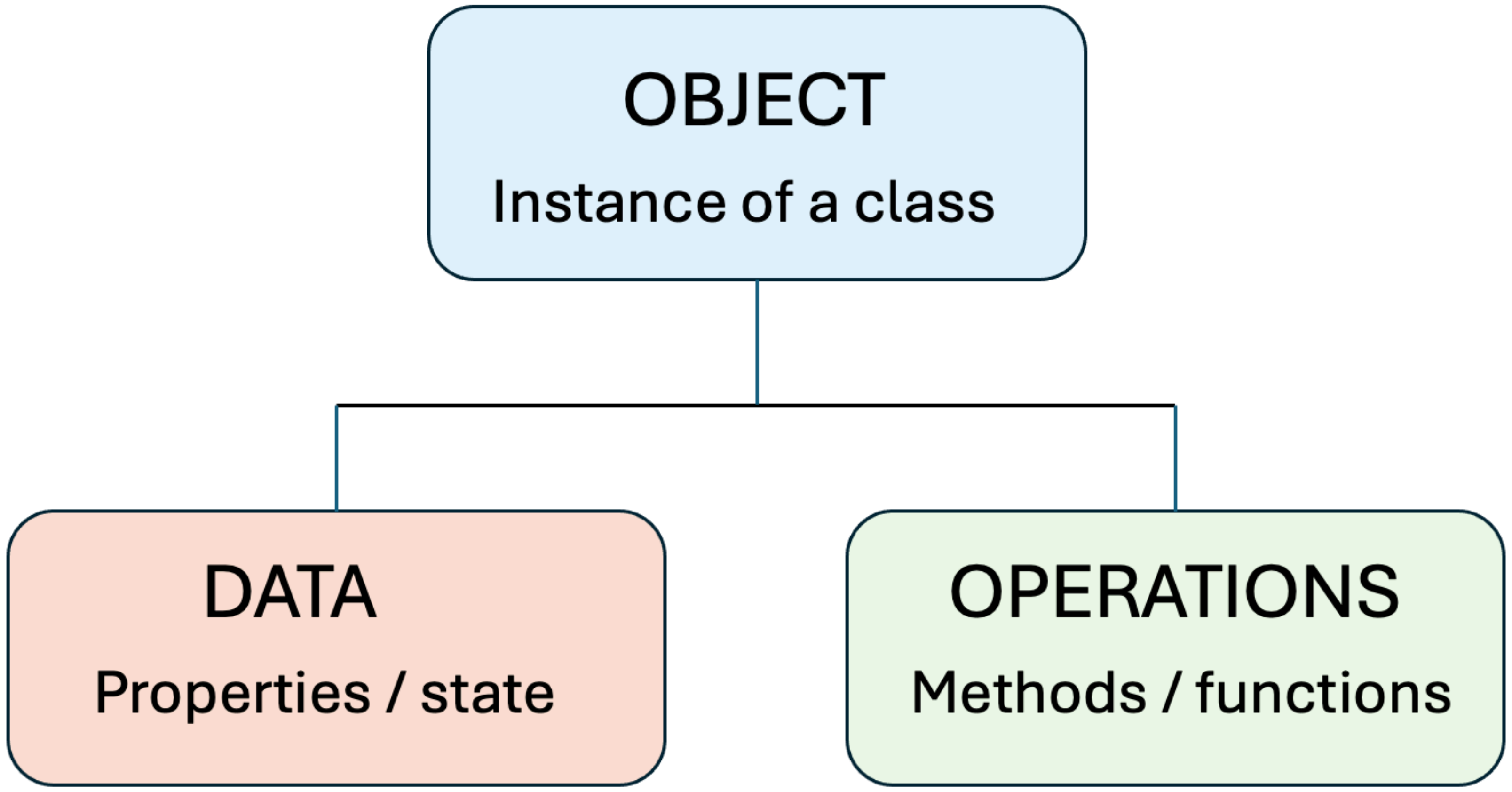
- Opening `{` ends first line of block (`if`, `for`, class, method).
- Indent every line inside block.
- Closing `}` should be on separate line, "undoing" indentation of block.
- Use `CamelCase` for class names, `drinkingCamelCase` for variables/methods, lowercase for single words.
- Use spaces between keywords and other control characters (`()`, `{}`) and operators (`&&`, `<`, etc.).
- No space between semi-colon and **previous** character (`i = 0;`, not `i = 0 ;`).
- Use a space after semi-colon and next statement (in definition of `for`-loop).
- One line, one statement (unless comma-separated, e.g. `int i = 0, j = 0`).

```
int x = 5, y = 20;
String my_name = "Mike Wazowski"; int age = 20;
for(int i = 0; i<values.length;i++){
  if(condition) {

  }
  else
  {

  }
}
```

```
int x = 5, y = 20;
String myName = "Mike Wazowski";
int age = 20;
for (int i = 0; i < values.length; i++) {
  if (condition) {

  } else {

  }
}
```

Java is object-oriented.

- A language is **object-oriented** if programs in that language are organized by the specification and use of **objects**.
- An object consists of (1) some **internal data items** along with (2) **operations** that can be performed on that data.

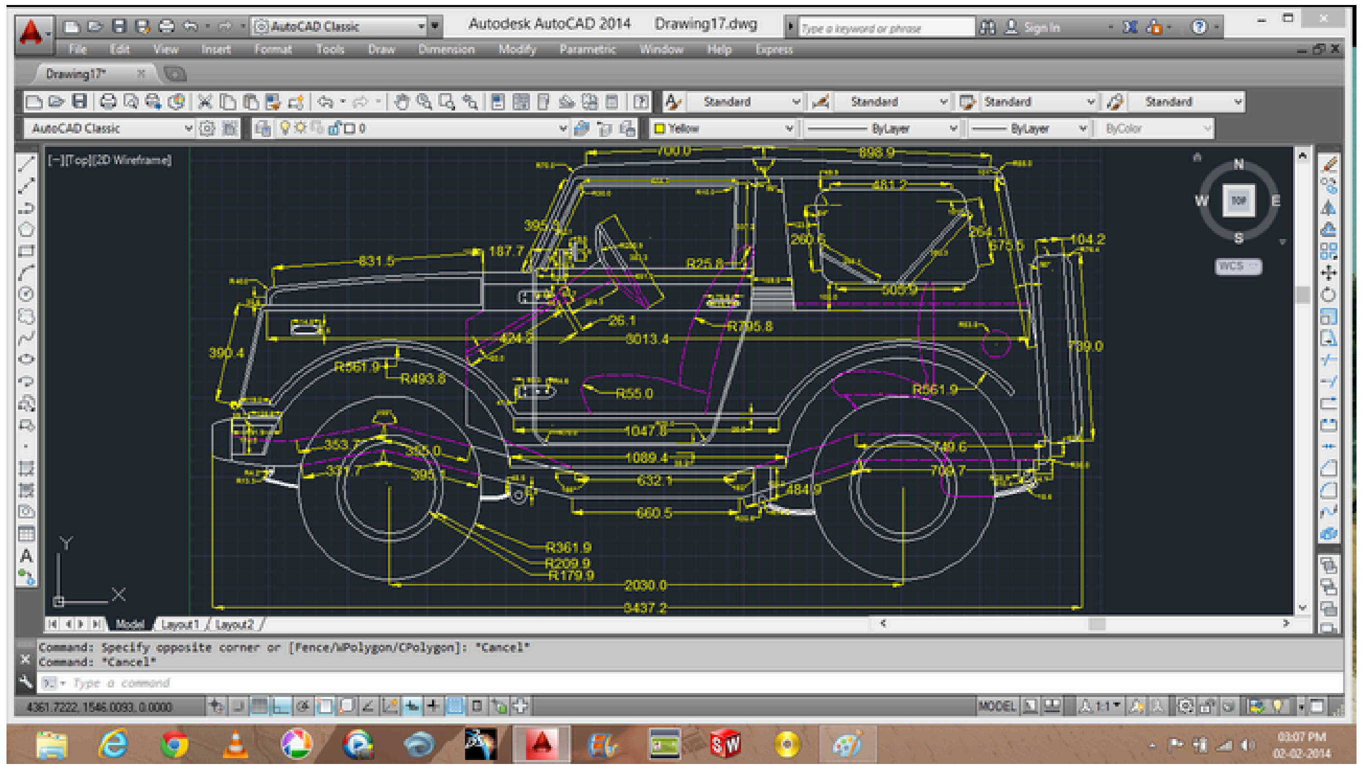


OBJECT
Instance of a class

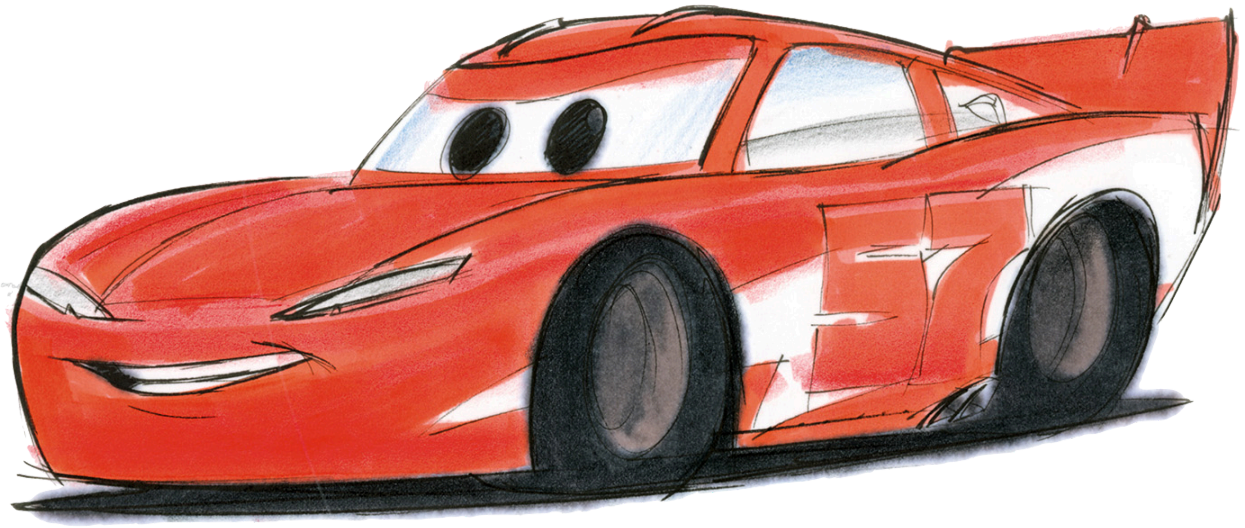
DATA
Properties / state

OPERATIONS
Methods / functions

Let's design and create a car!



```
Command: Specify opposite corner or [Fence/WPolygon/CPolygon]: *Cancel*
Command: *Cancel*
```

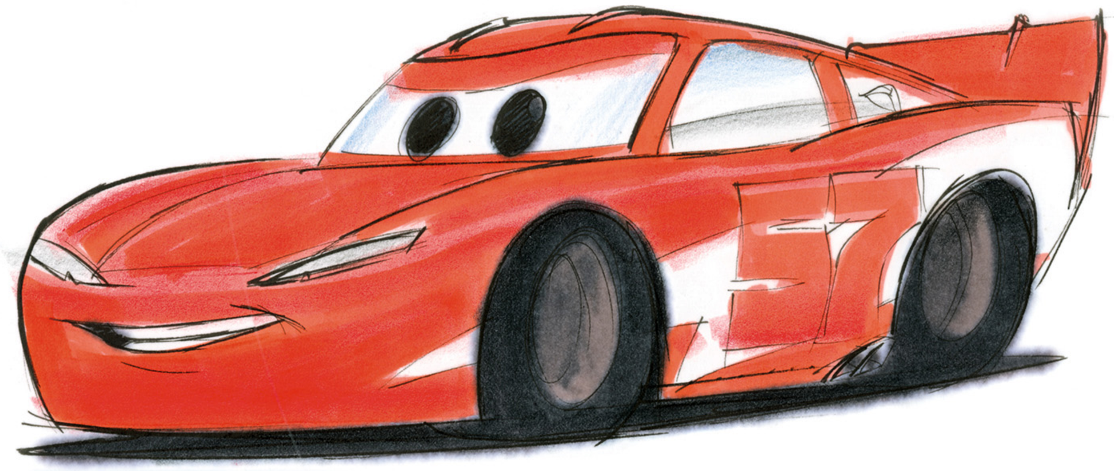


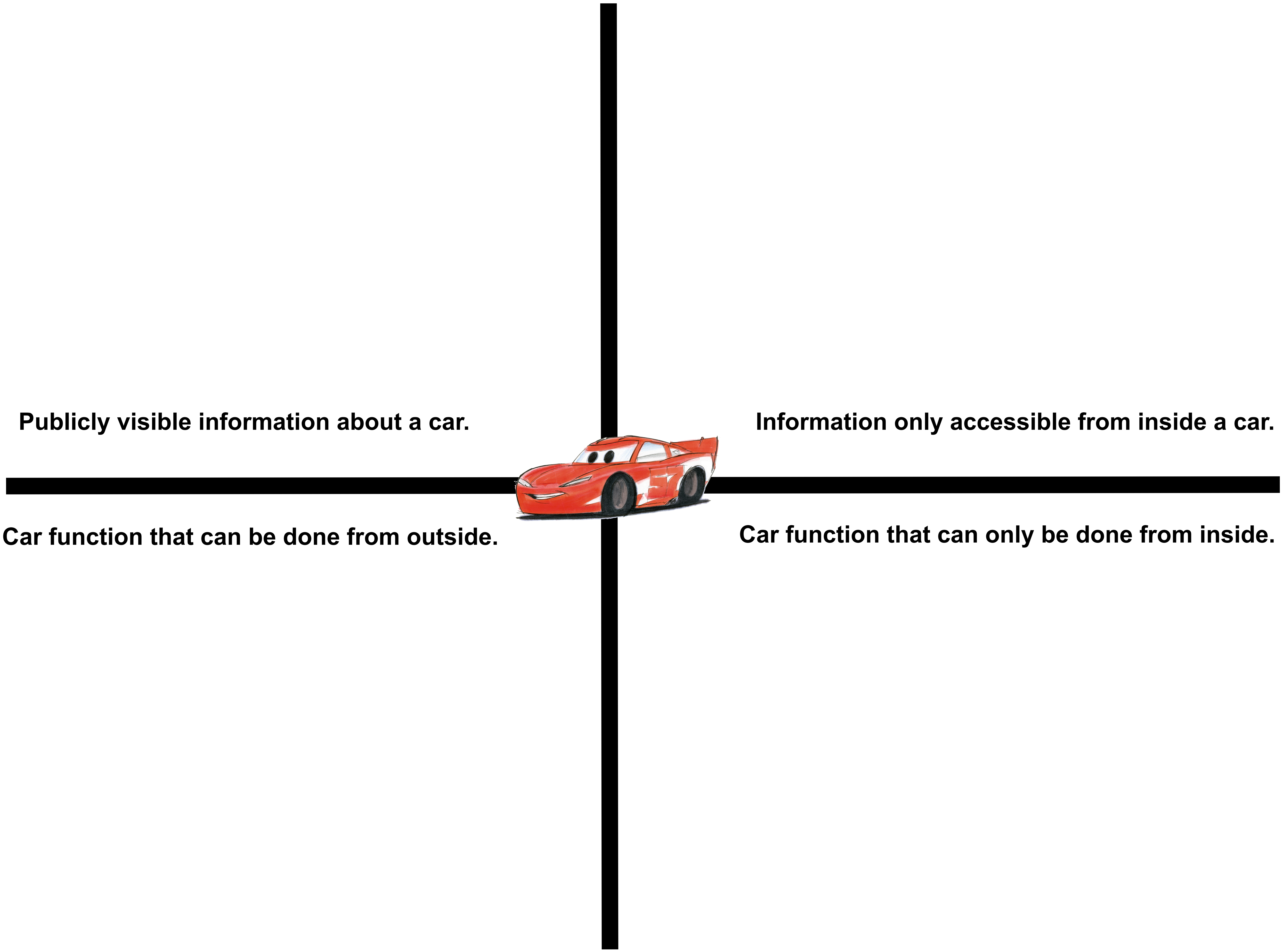
```
class Car {
  ...
}
```

```
Car car = new Car("Subaru", 2019);
```

Think about the following questions in groups.

- What information about a car is publicly visible?
- What information about a car can only be accessed if you're inside the car?
- What is a function of a car that can be done from outside the car?
- What is a function of a car that can only be done from inside the car?
- What is something about a car that can change?
- What is something about a car that cannot change?





Publicly visible information about a car.

Information only accessible from inside a car.

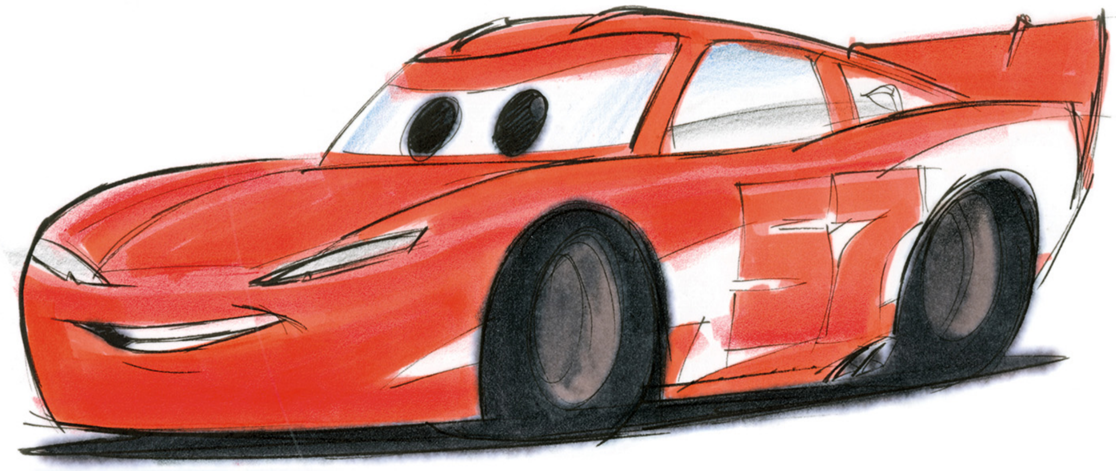
Car function that can be done from outside.

Car function that can only be done from inside.

Anatomy of a Java class definition.

```java
// in a file called CarExample.java
class Car {

  // member variables (fields)
  String make;
  int year;

  // methods
  Car(String make, int year) { // constructor
    this.make = make;
    this.year = year;
  }

  void drive() {
    System.out.println("Starting the " + make + "...");
    System.out.println("Vroom vroom.");
  }
}

public class CarExample {
  public static void main(String[] args) {
    Car car = new Car("Subaru", 2019);
    car.drive();
  }
}
```

Use *access modifiers* to control what is visible and what is not.



- `public`: can be accessed by code *outside* of the class (also inside).
- `private`: can only be accessed by code *inside* the class.

Why do we use these? readability, correctness.

Anatomy of a Java class with *access modifiers*.

```java
// in a file called CarExample.java
class Car {

  // member variables (fields)
  public String make;
  public int year;
  private int gear;

  // methods
  Car(String make, int year) { // constructor
    this.make = make;
    this.year = year;
  }

  public void drive() {
    System.out.println("Starting the " + make + "...");
    setGear(1); // put the car in first gear
    System.out.println("Vroom vroom.");
    honk();
  }

  private void setGear(int gear) {
    this.gear = gear;
  }

  private void honk() {
    System.out.println("beep beep");
  }
}

public class CarExample {
  public static void main(String[] args) {
    Car car = new Car("Honda", 2019);
    car.drive();
  }
}
```

What is this?

```
String make; // fields
int year;
int gear;

Car(String make, int year) { // constructor
  this.make = make;
  this.year = year;
}

private void setGear(int gear) {
  this.gear = gear;
}
```

Primary uses:

- Avoid ambiguity in fields and parameters (notice `make`, `year` and `gear` are **parameters** and **fields**).
- Pass a reference to this object to some other function. For example, imagine we keep a reference to an instance of a `Garage` class called `garage`. Perhaps we need to call `garage.changeOil(this);`

Arrays can be used to hold many `Car` objects.

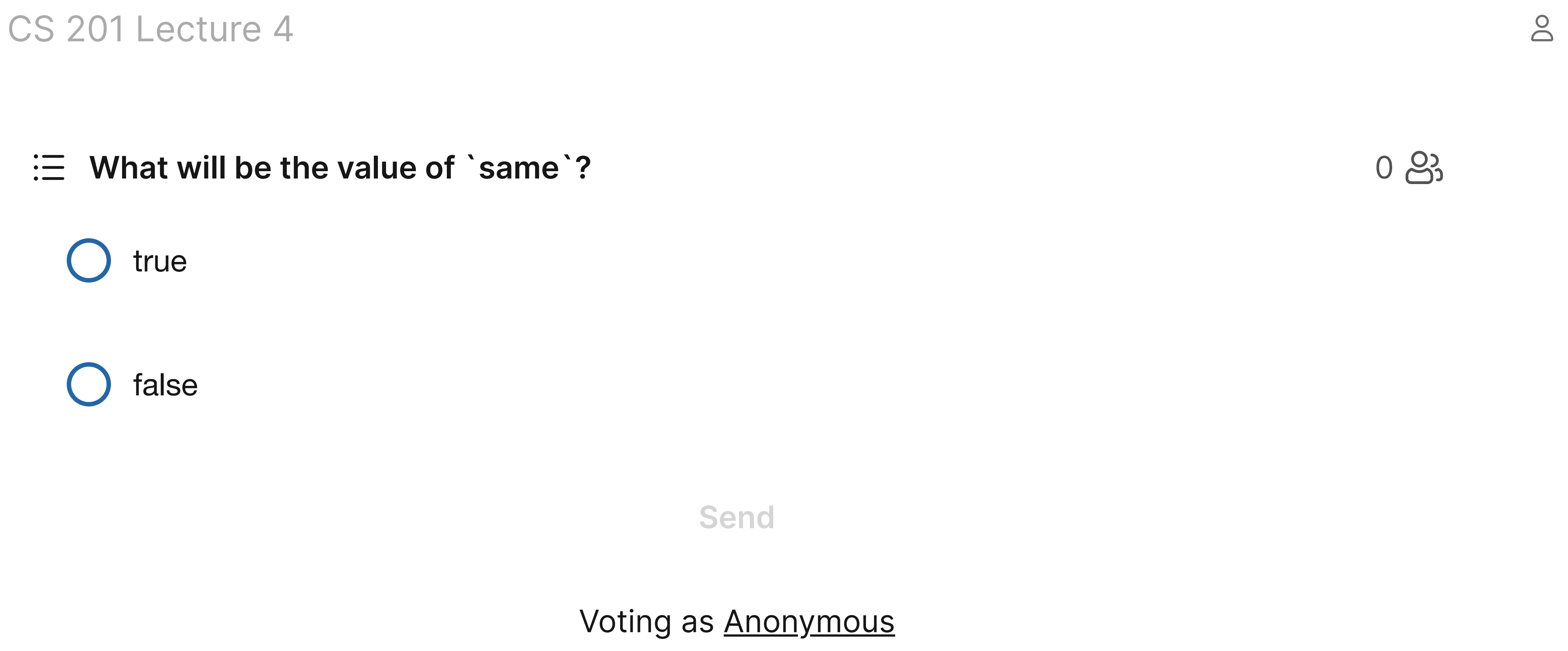
Items are initially `null`: we need to use `new` to actually create `Car` objects.

```
1 int nCars = 5;
2 Car[] cars = new Car[nCars];
3 cars[0] = new Car("Subaru", 2019);
4 cars[1] = new Car("Honda", 2021);
5 cars[2] = new Car("Ford", 2024);
6 // cars[3] and cars[4] are still null
7 System.out.println("Car 3 make = " + cars[3].make); // Exception!!
```

```
Exception in thread "main" java.lang.NullPointerException: Cannot read field
  "make" because "cars[3]" is null at CarExample.main(CarExample.java:7)
```

What will be the value of `same`?
Go to slido.com (event #1424027).

```java
cars[0] = new Car("Honda", 2019);
cars[1] = new Car("Honda", 2019);
boolean same = (cars[0] == cars[1]);
```



CS 201 Lecture 4

What will be the value of `same`? 0

○ true

○ false

Send

Voting as Anonymous

A note about *mutability* and *immutability*.

Think about whether you can "mutate" the object.

- **mutable:** object fields can be modified after creation.
- **immutable:** objects fields cannot be modified after creation.

```java
String s = "Hello";
s += " World"; // append
```

```java
// essentially the same as
String sOld = "Hello";
String sNew = new String(sOld + " World");
```

`String`s are actually **immutable**.

We end up creating a new `String` when appending.

Standard way to expose what can be set and not set, and what can be retrieved: *setter* and *getter* methods.

Recall what we did in our `Car` class.

```java
private void setGear(int gear) {
  this.gear = gear;
}
```

```java
public void setLocked(boolean value) {
  locked = value;
}
```

```java
public int getSpeed() {
  return speed;
}
```

Remember that static keyword means it doesn't require an *instance* (object) - it's part of the class definition.

```java
class Car {
  public static int numberOfWheels() {
    return 4; // we don't need to create a car to ask this question
  }
  ...
}

// somewhere else in the code
System.out.println("The car design has " + Car.numberOfWheels() + "wheels");
```

BUT! Really this should be a *constant*, defined using static final

```java
class Car {
  public static final int NUMBER_OF_WHEELS = 4; // use SNAKE_CASE for constants
  ...
}

// somewhere else in the code
System.out.println("The car design has " + Car.NUMBER_OF_WHEELS + "wheels");
```

Ask yourself: can I use the blueprint/design or do I need a built object?

Group exercise: do something similar for a `Student`.

Things to think about:

- First, think about what a `Student` object should store: name? array of classes? age? dorm? mailbox?
- What kinds of functions should a `Student` be able to do?
- Which fields and methods should be `public`?
- Which fields and methods should be `private`?
- Which fields should be modifiable via a **setter** or retrievable from a **getter**?

Intentionally open-ended!
Brainstorming and designing is part of developing code!

See you Friday!

- Read the Lab 2 writeup and the Homework 2 writeup
- Please visit me (go/briggs) or Noah (go/noah) or Smith (go/smith) for office hours or see the course assistants (schedule at go/cshelp).
- For more information about access modifiers, see Oracle's documentation: https://docs.oracle.com/javase/tutorial/java/javaOO/variables.html