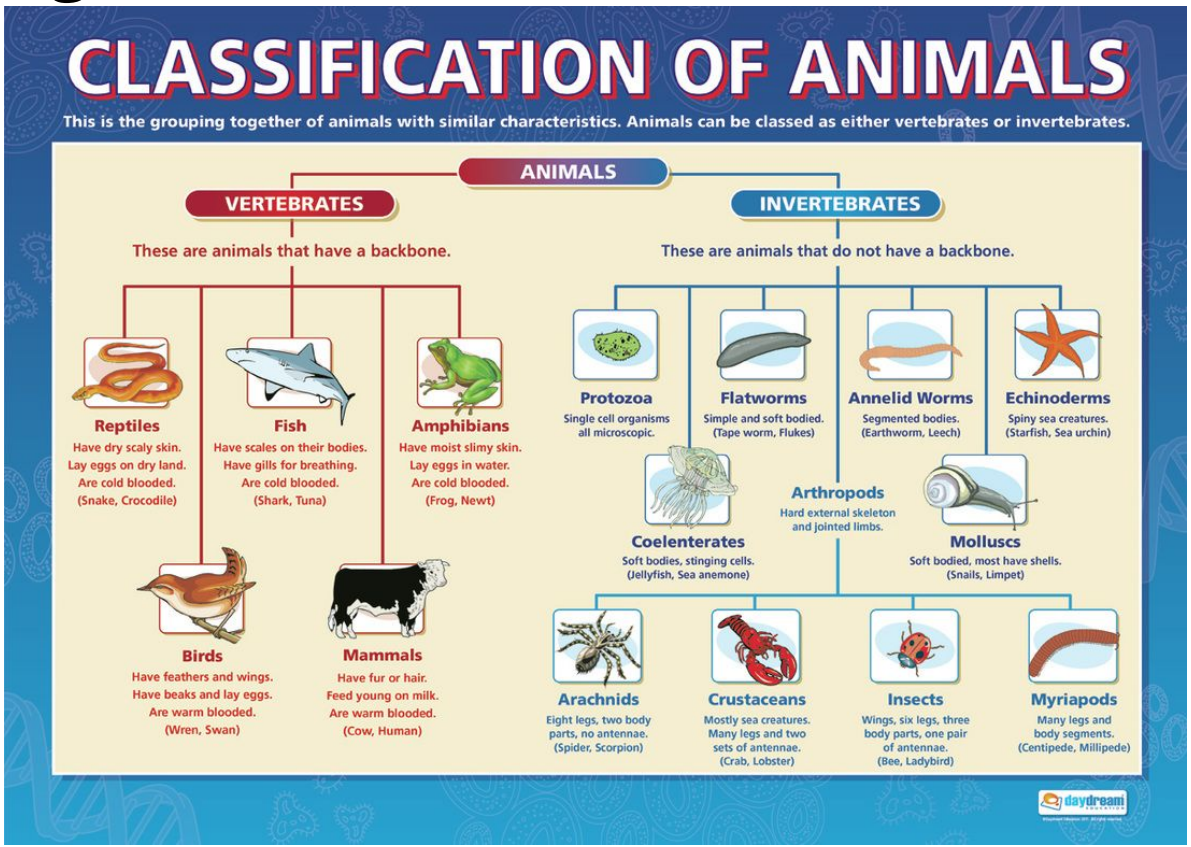
Full Stack C# .NET

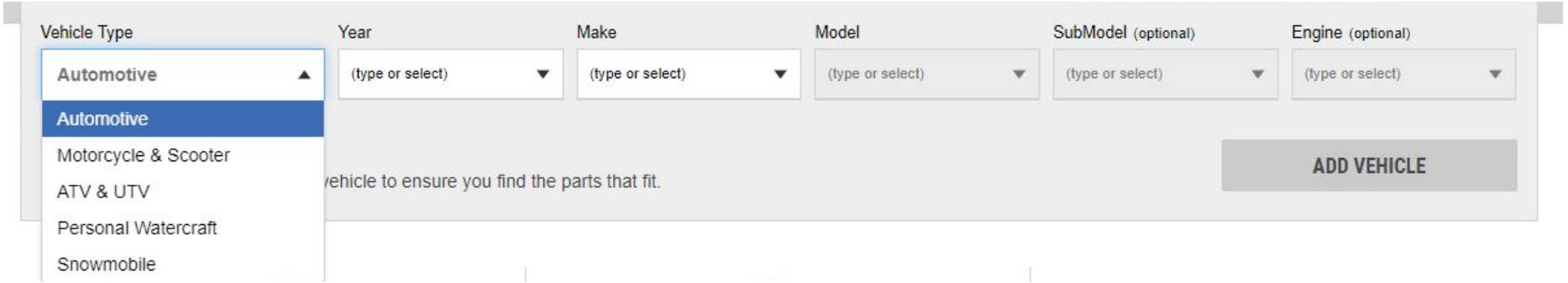
Intro to Classes and Objects

Download PDF Copy

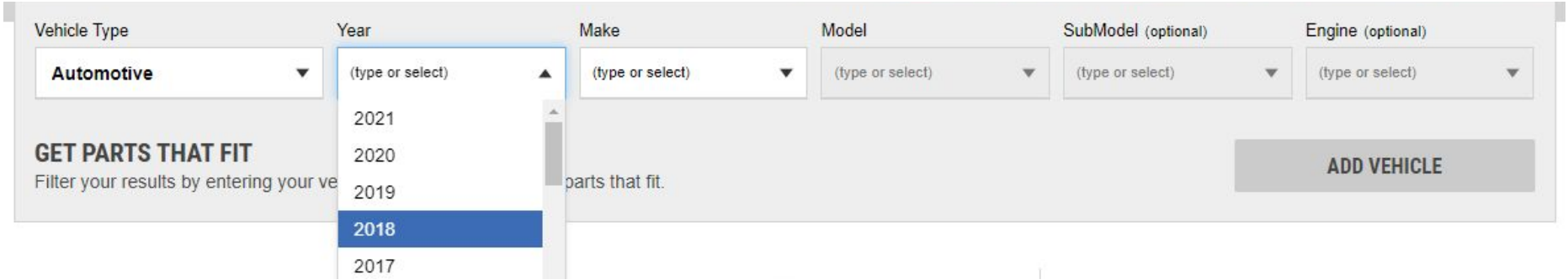
We organize information with classifications



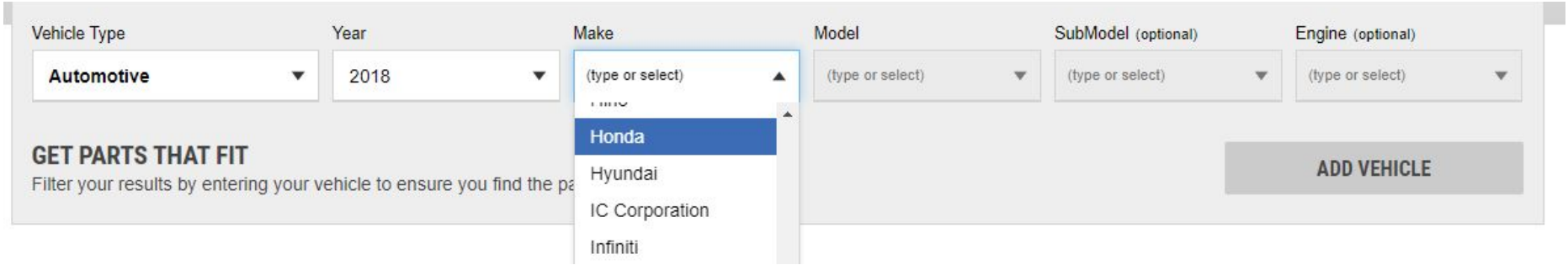
Classifying vehicles



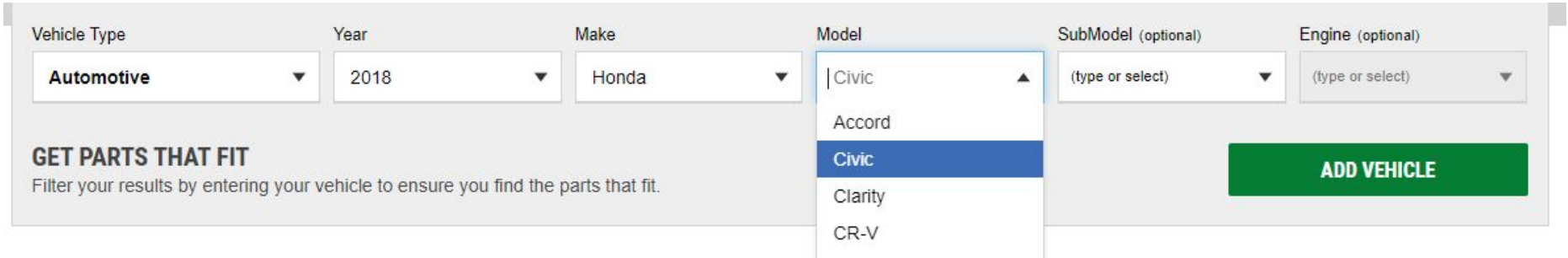
Classifying vehicles



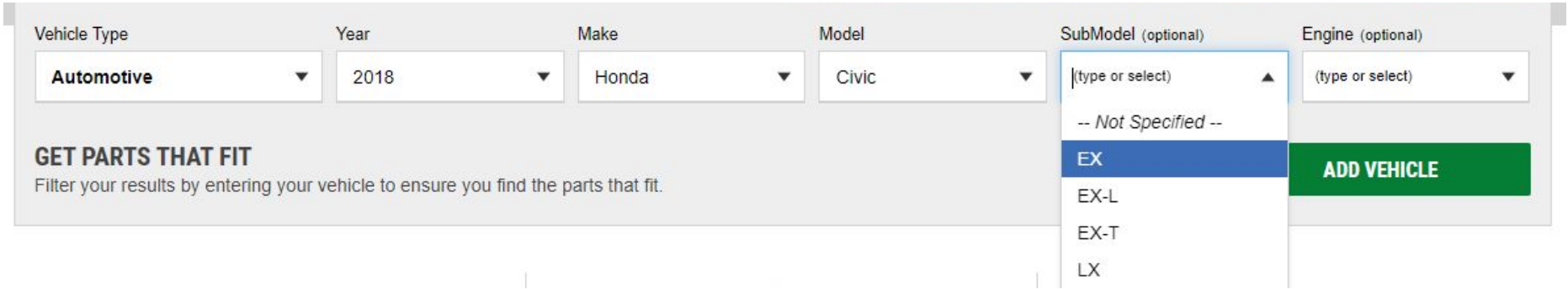
Classifying vehicles



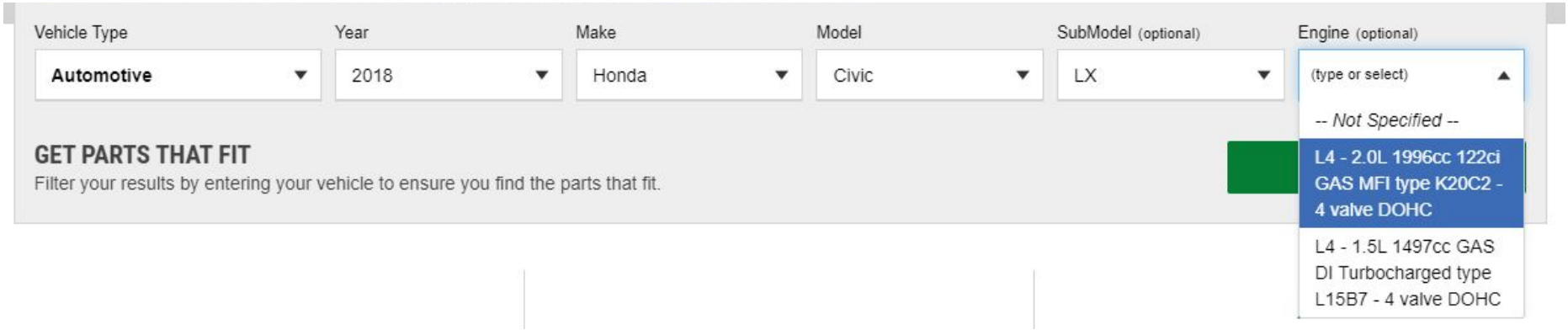
Classifying vehicles



Classifying vehicles



Classifying vehicles



Class is short for Classification

Class == Classification

A class is a category, or a group, or a taxonomy.

It’s a way of grouping data

Instances

Individual examples or individual items are called instances.

They’re also called objects.

Instances

What is this object an example of? What is this an instance of?

Instances

What’s different about these two pens?

Instances

What’s different about these two pens?

One is blue

One is red

Instances

They have attributes that describe them and distinguish them.

Color: Blue

Color: Red

Instances

We can add more attributes, such as brand and type of ink.

Color: Blue
Brand: Pilot
Ink Type: Gel

Color: Red
Brand: Pilot
Ink Type: Gel

Instances

Now let’s add a third instance.

Color: Blue
Brand: Pilot
Ink Type: Gel

Color: Red
Brand: Pilot
Ink Type: Gel

Color: Blue
Brand: Uline
Ink Type: Ballpoint

Instances

...and even more attributes!

Color: Blue
Brand: Pilot
Ink Type: Gel
Price: $1.50
Length: 5 inches

Color: Red
Brand: Pilot
Ink Type: Gel
Price: $1.50
Length: 5 inches

Color: Blue
Brand: Uline
Ink Type: Ballpoint
Price: $0.99
Length: 4.5 inches

Instances

Let’s formalize it into actual C#. These “attributes” will become variables that we call “member variables.”

In C# it might look like this. Notice we’re only providing a type and a name at this point. (That is, a pen has a Color, but we’re not assigning “blue” to it yet here.)

```
{
        string Color;
        string Brand;
        string InkType;
        decimal Price;
        float Length;
};
```

Note: This isn’t yet full C# code. It won’t run yet until the next slide…

Instances

Let’s give our classification a name. We’ll call it Pen.

```
class Pen {
        String Color;
        String Brand;
        String InkType;
        Decimal Price;
        Float Length;
};
```

And now we have true C# code.

Instances

Let’s create an instance of class Pen. Here are the attributes this pen will have.

Color: Black
Brand: Bick
Ink Type: Ballpoint
Price: $0.50
Length: 5.3 inches

Instances

Let’s print out some attribute, i.e. member variables, of this new pen

```
Pen mypen = new Pen() {
        Color = "Black",
        Brand = "Bick",
        InkType = "Ballpoint",
        Price = 0.50m,
        Length = 5.3f
};
Console.WriteLine(mypen.Color);
Console.WriteLine(mypen.Price);
```

Classes in C#

Here's a full example. First you create the class definition. And then you can create instances of the class.

This code creates a new instance of Pen, sets its member variables to various values, and then prints out the pen's color and price.

```
class Pen
    {
        public string Color;
        public string Brand;
        public string InkType;
        public Decimal Price;
        public float Length;
    };

Pen mypen = new Pen()
            {
                Color = "Black",
                Brand = "Bick",
                InkType = "Ballpoint",
                Price = 0.50m,
                Length = 5.3f
            };

Console.WriteLine(mypen.Color);

Console.WriteLine(mypen.Price);
```

Classes are Types

Notice the similarity!

int x; // This declares a variable that can hold an integer

string y; // This declares a variable that can hold a string

Pen p; // This declares a variable that can hold a pen

(Technically there are more differences, including the fact that we use the word “new,” but that’s a good start.)

Whereas int and string come with C#, we’re free to create more types. We do so by creating classes. Classes are user-defined types.

Classes in C#: Try it!

Think of a classification, it can be anything.

- Cars
- Buildings
- Musical Instruments
- Food

etc. Just pick one. It can be anything.

Classes in C#: Try it!

List some attributes of your classification, and give each one a type.

Example:

Classification pasta

- Shape is a string
- Number of pounds is a float

Classes in C#: Try it!

List some attributes of your classification, and give each one a type.

Example:

Classification pasta

- Shape is a string
- Number of pounds is a float

And then let's create two instances.

```
class Pasta
{
    public string Shape;
    public float Pounds;
};

Pasta dish = new Pasta()
{
    Shape = "Spirals",
    Pounds = 1.0f
};

Pasta dish2 = new Pasta()
{
    Shape = "Shells",
    Pounds = 2.0f
};

Console.WriteLine(dish.Shape);
Console.WriteLine(dish2.Shape);
```

Real-life Objects can do things

Objects don’t just have attributes that describe them.

Objects can do things.

Real-life Objects can do things

Objects don’t just have attributes that describe them.

Objects can do things.

Pens can draw on paper.

Cars can move forward.

Real-life Objects can do things

Objects don’t just have attributes that describe them.

Objects can do things.

Pens can draw on paper.

Cars can move forward.

We can refill pens… but…

Real-life Objects can do things

We can refill pens but…

Think of it from the perspective of the pen.

Then pen receives a refill

The pen refills

In a sense, we don’t refill the pen; we ask the pen to refill it.

In C# we “ask” an object to do something by calling its method:

mypen.refill();

Real-life Objects can do things

These things that objects do are called... methods!

Objects have methods and properties

Member variables describe the object

Member methods are the things the object can do

We treat methods as from the perspective of the object itself.

I don’t start the car’s engine; I give the car a command to start and it starts itself.

Methods can give us information about the object

In addition to just “doing things,” methods can give us information about an object.

We can make multiple objects

Pasta dish = new Pasta();

Pasta dish2 = new Pasta();

We can store our objects in arrays

Pasta[] mydishes = new Pasta[20];

Mydishes[0] = new Pasta();

Mydishes[1] = new Pasta();

Think in terms of objects and classifications

The world is made up of classifications (classes!) and instances of those classes!

It’s a natural way to write software

Video games will have a class for character, and each character will have member variables (strength, health, hair color, etc.)

Video games will have a class for weapon, and each weapon will have member variables (type, damage level, etc.)

Think in terms of objects and classifications

In a video game, a character object might have a member variable that's an array of weapons and other items, perhaps called "inventory."

Objects can have member variables that are objects

Objects can have member variables that are arrays of objects

Constructing an object

When we create a new object, we typically need to initialize it with information.

A pen might need a brand, color, ink level, etc.

A pen might start with an empty amount of ink as as default.

We include functions in our classes called “constructors” where we do such initialization. These are just methods, but they get the same name as a class.

Keeping the properties private

A common approach to Object Oriented Programming is that we usually don’t modify the attributes directly in an object.

Instead we call methods.

- One method might change a member variable
- Another method might give us back the value of a member variable

The method for changing a member variable might do some validation and return an error.

It might also adjust the data -- round numbers, adjust the casing for words, etc.

Keeping the properties private

By restricting access to the member variables through methods, we keep control of our member variables so they can't be damaged.

By declaring the member variables "private" the compiler will stop coders from even being able to access the member variables directly.

We access member variables through methods that are "public."

We restrict member variables by making them "private."

Restricting Access

Any member (variable or method) can be either public or private.

Private methods can only be called from within other methods inside the class.

Private member variables can only be accessed from within any method inside the class.

Getters and setters

Methods that set a member variables are called setters.

Methods that get a member variables are called getters.

C# lets us connect these getters and setters directly to a member variable using an approach called “properties” that we’ll look at another day.

Static Members

When we refill a pen, we increase the ink in that particular pen.

The ink amount and the refill method are attached to a particular pen.

Sometimes, however, member variables and methods might need to be shared among all of the instances of a class.

For example, the pens might all live inside an array or a list. And we might have a method that scans through all the pens and refills any that are empty.

Static Members

Such methods aren’t unique to individual pens. They are shared among all the pens.

We call these methods *static*.

(That word is a holdover from the old C++ language and doesn’t have much meaning regarding the non-technical language meaning.)

Recap

What should you know:

- What are objects?
- How to build classes
- How to call methods on objects
- How to call constructors with the new keyword
- Public vs Private