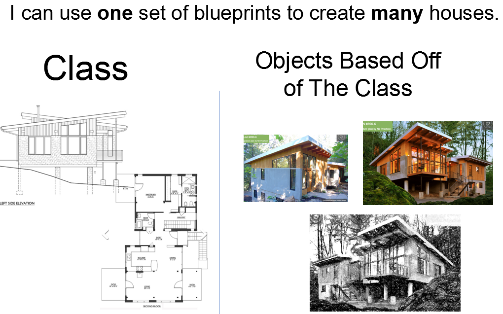
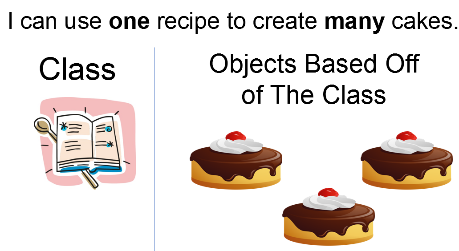
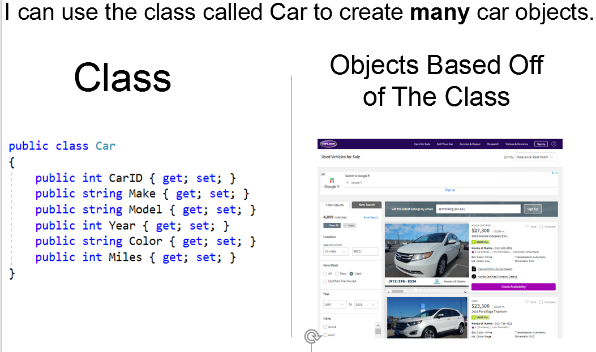
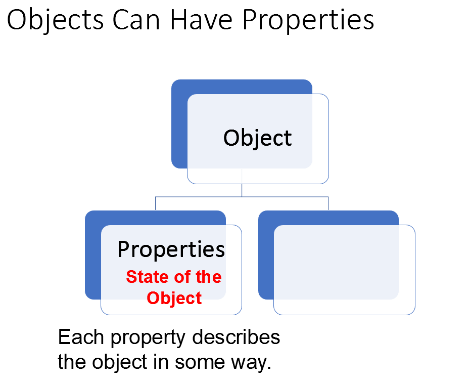
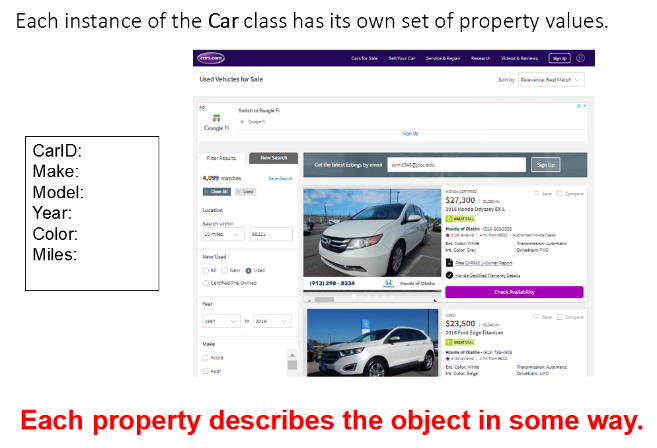
**Working with Classes & Objects in C#**

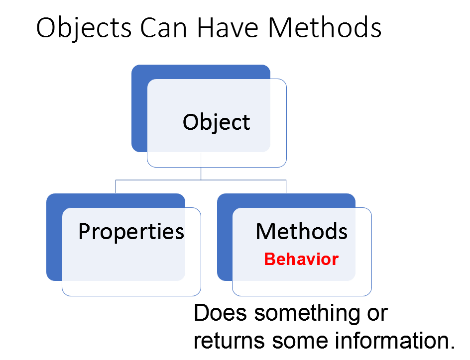
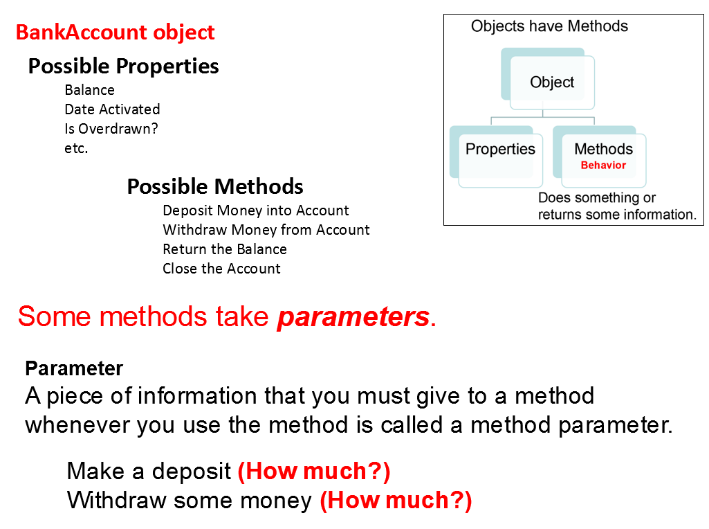
 **Review of Some Object-Oriented Programming Concepts: Classes, Objects, Properties, Method**A **class** is a blueprint that can be used to create an **object**.



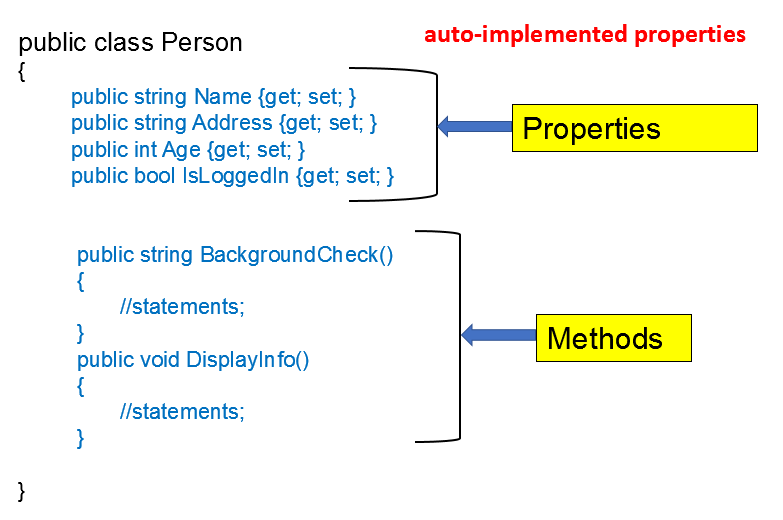
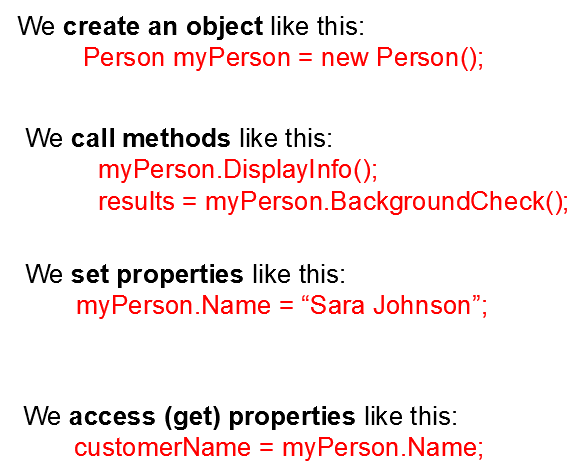


**🡨** Within an MVC application, we   
 would call this a **model class**.



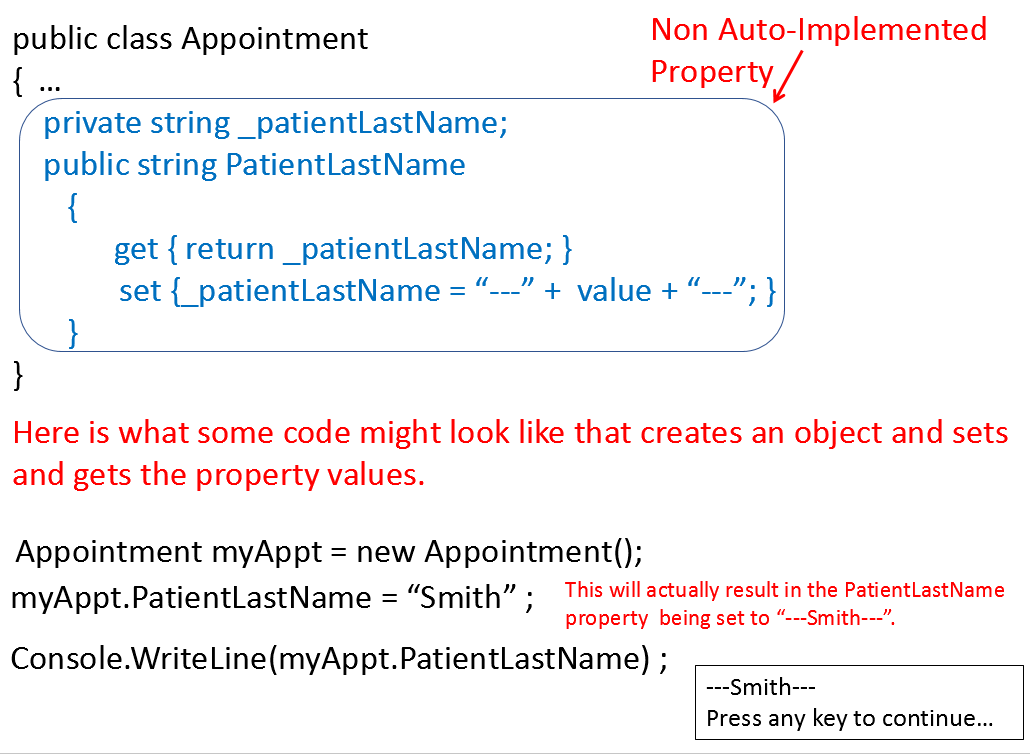


**Creating a Class in C# Working With an Object Based on the Class**

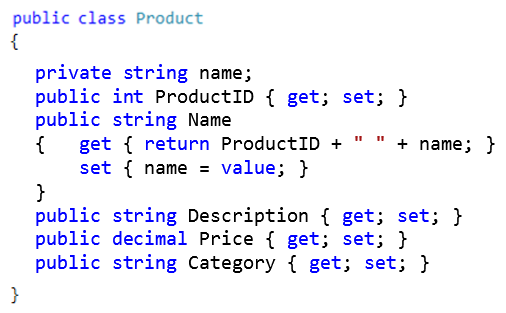


As you read each question below, cover the answer and write or type in your own answer. Then verify that you answered correctly.  
  
Based on the Person class created above, how would you set the Age property of the myPerson object to 25?  
 myPerson.Age = 25;

How would you get the Address of myPerson and put the contents into a string called myString?  
 myString = myPerson.Address;  
  
How would you call the DisplayInfo method on myPerson?  
 myPerson.DisplayInfo();

**Creating Non-Auto-Implemented Properties**

Let’s create a class called Product with one non-auto-implemented property.

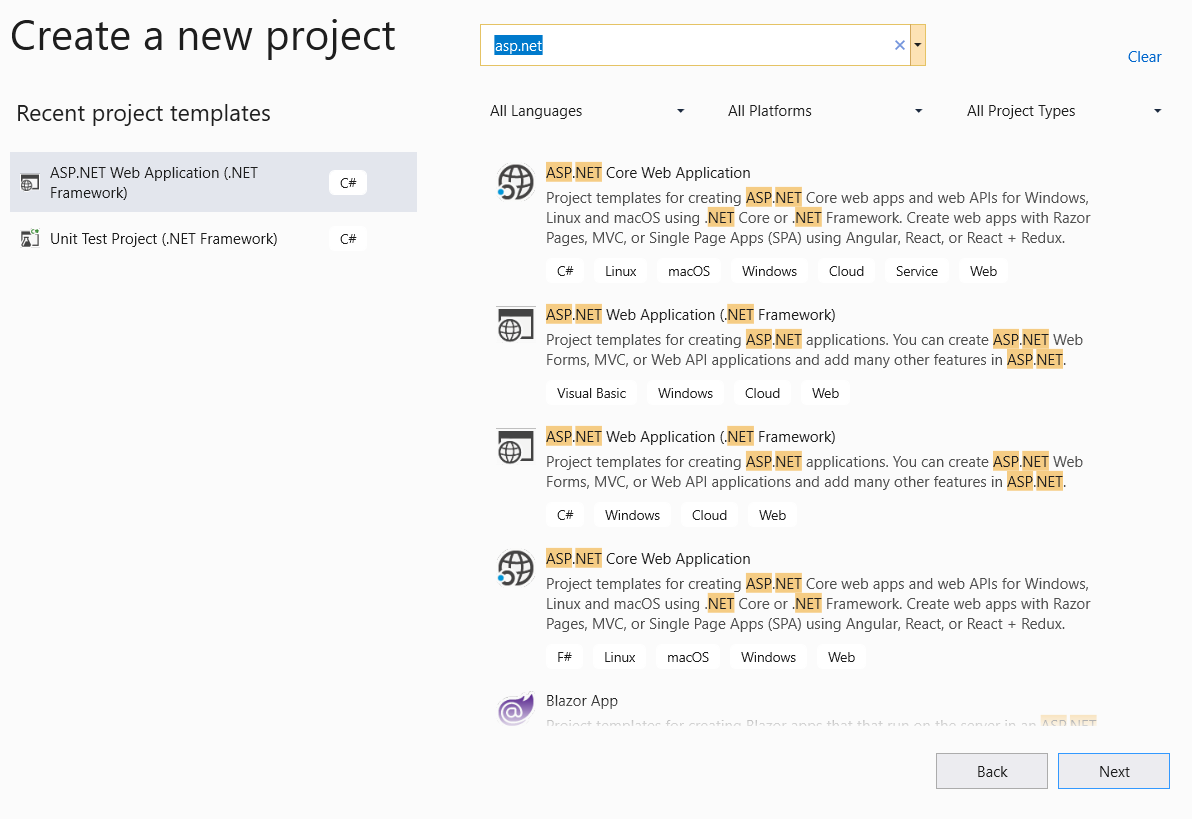


Notice that we have a property called ProductID. When our classes are going to be used to store information from a database, we want to include a property that will be an identifier like this.

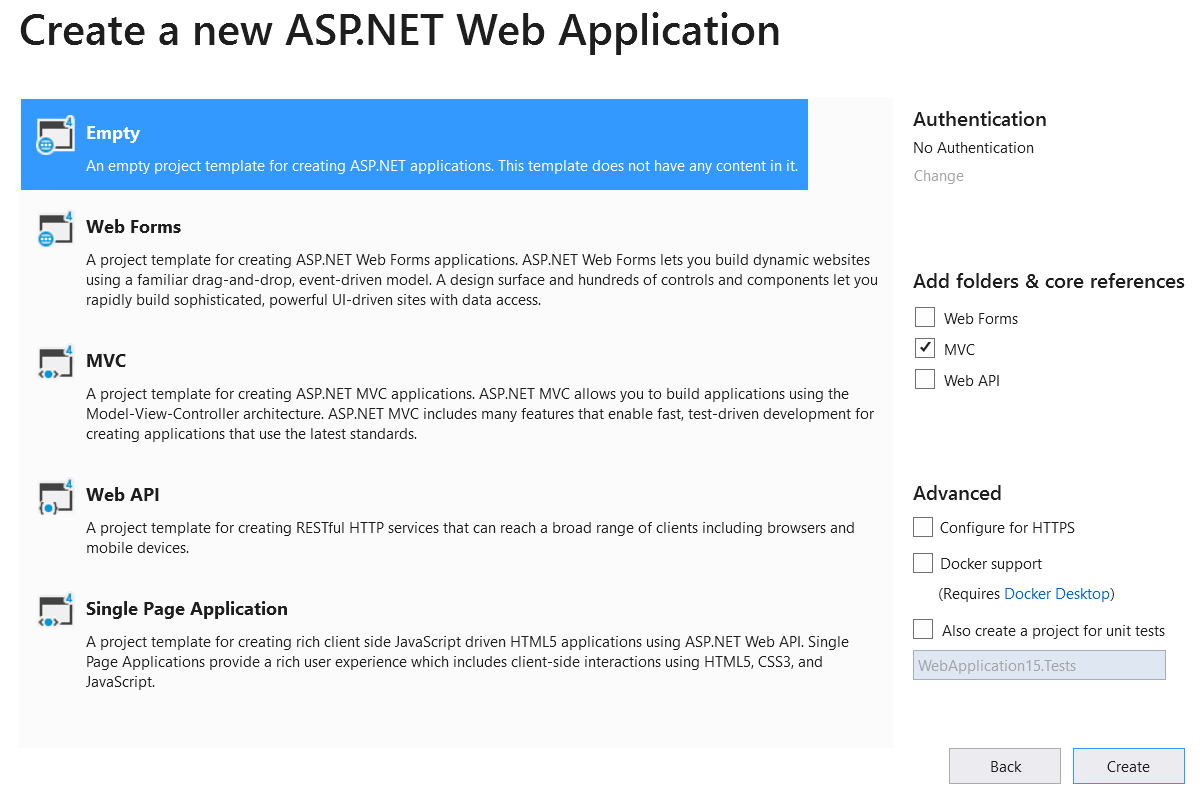
Here, we have set the Name property to always return the ProductID along with the name of the product.

Now let’s create a Model class within an MVC ASP.NET application.

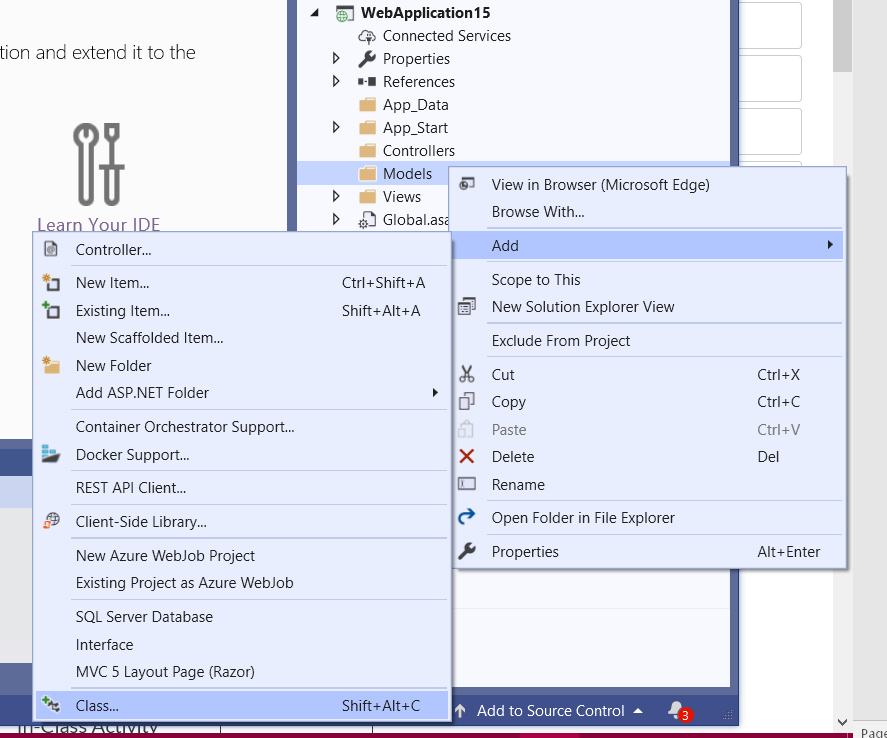
Open Visual Studio 2019. Create a new project.



Be careful not to choose ASP.NET **Core**, and be careful to choose C# and not Visual Basic.



🡨 Uncheck



Highlight the Model folder at the right of your screen, right-click, choose Add > Class.

Name the class and add code that will define the properties. Use prop TAB TAB to get a template for creating a property.

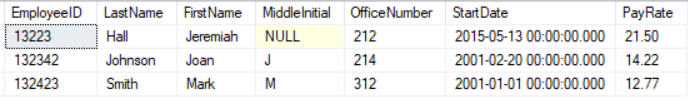
We’ve created a model!

<https://quizlet.com/_8morbj?x=1qqt&i=2z3456>  
Remember to click on Flashcards (even though Flashcards is already underlined) to see the questions first. Then click on the question to see the answer.

**See the Wrap-Up Video for the correct answers.**

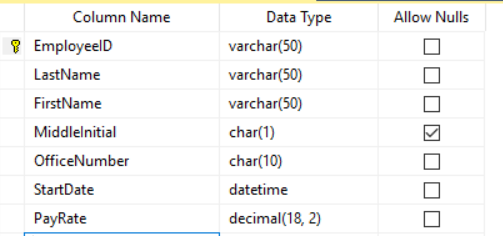
Quizlet:

**Tables & Navigation Properties**

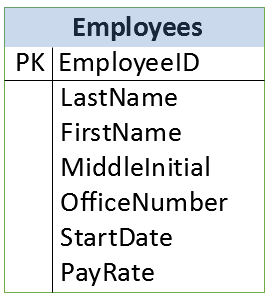
 **What is a table?**  A table is a set of rows and columns, like an Excel spreadsheet.

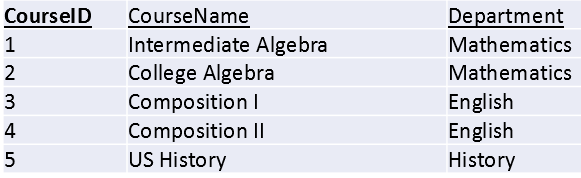
We call each row a record.

In SQL Server (or Oracle, etc.), we would first design the table, stating the column names and their types. We also state whether a column value can be null.

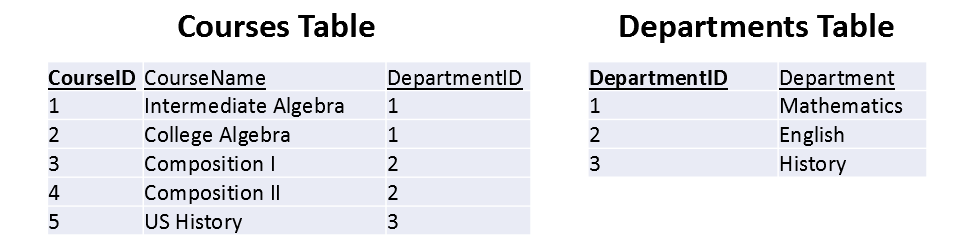


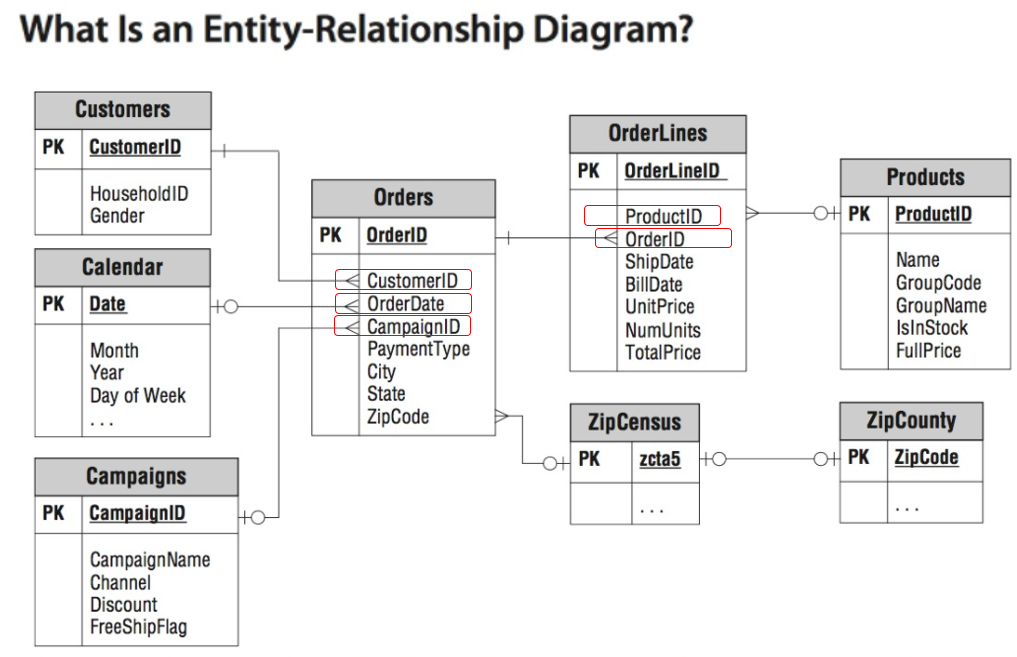
**Primary key columns** uniquely identify each row in the table. That is, no two rows have the same value for the primary key and the primary key is never NULL. We always want to make sure each table has a primary key.

  
Here is another way that you might see this   
same table represented in a diagram.

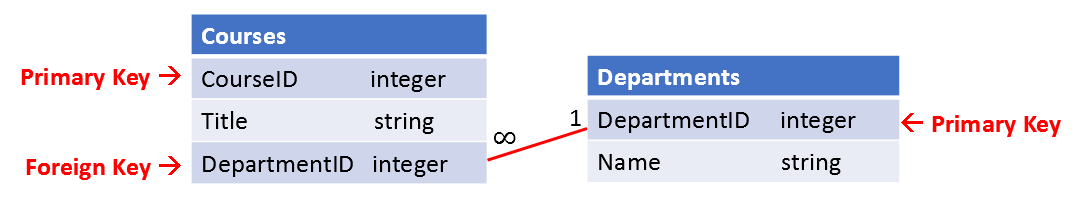
Let’s say that our application will list college courses along with their departments. Our list might look like this:

Imagine if we have 25 courses offered by the Mathematics department. We would have Mathematics repeated 25 times.  
If our records in our table looked just like this, we would be taking up a lot of space to repeat Mathematics over and over.   
Plus, if we ever decided to change the name of the department to Math, we would have many records that needed to be changed.   
We could make some typos or miss some of them.

So instead of having one table in our database with the department name included in the table, we should have two tables that look like this instead:



A **foreign key** is a column whose contents are the primary key of another table . Let’s circle some foreign keys in this Entity-Relationship Diagram.

  
  
Let’s say that we want to create an application that will allow us to view, add, edit, and delete **courses** along with their corresponding **departments**. We would have a table called Courses and another one called Departments.

Our corresponding model classes would look like this:

public class Course

public class Department

{

public int DepartmentID { get; set; }

public string Name { get; set; }

}

{

public int CourseID { get; set; }

public string Title { get; set; }

public int DepartmentID { get; set; }

}

**However,** in order to get everything working correctly and let our application know that there is a connection between the DepartmentIDs in the Course objects and the DepartmentIDs in the Department objects, we need to add some **navigation properties**.

public class Course

public class Department

{

public int DepartmentID { get; set; }

public string Name { get; set; }

**public virtual ICollection<Course> Courses { get; set; }**  
 }

{

public int CourseID { get; set; }

public string Title { get; set; }

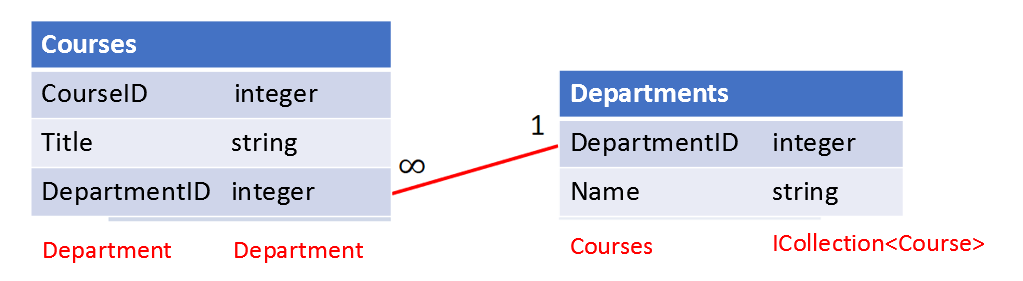
public int DepartmentID { get; set; }

**public virtual Department Department { get; set; }** }

Each course has only **one** department. Each department has **many** courses.

And the Courses entity of the Departments entity will hold a list of all of the Course entities that are related to the Department.

In this case, the Department property of a Course entity will hold the Department entity that is related to the Course.



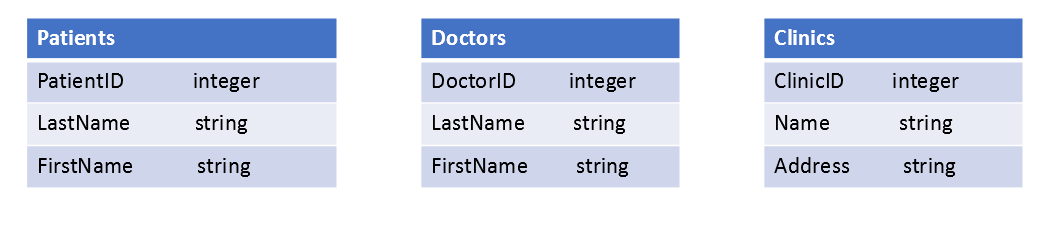
*Getting your navigation properties right* ***is the key*** *to having Code-First work correctly!*

Navigation properties are typically defined as **virtual** so that they can take advantage of certain Entity Framework functionality such as ***lazy loading***.

**Try it Yourself!**  
  
Let’s create a database diagram and some model classes that will show the proper relationships between the following entities.

Assume each patient has only one doctor. Assume doctors don’t work at more than one clinic.

Here’s the start of the Entity Relationship Diagram.

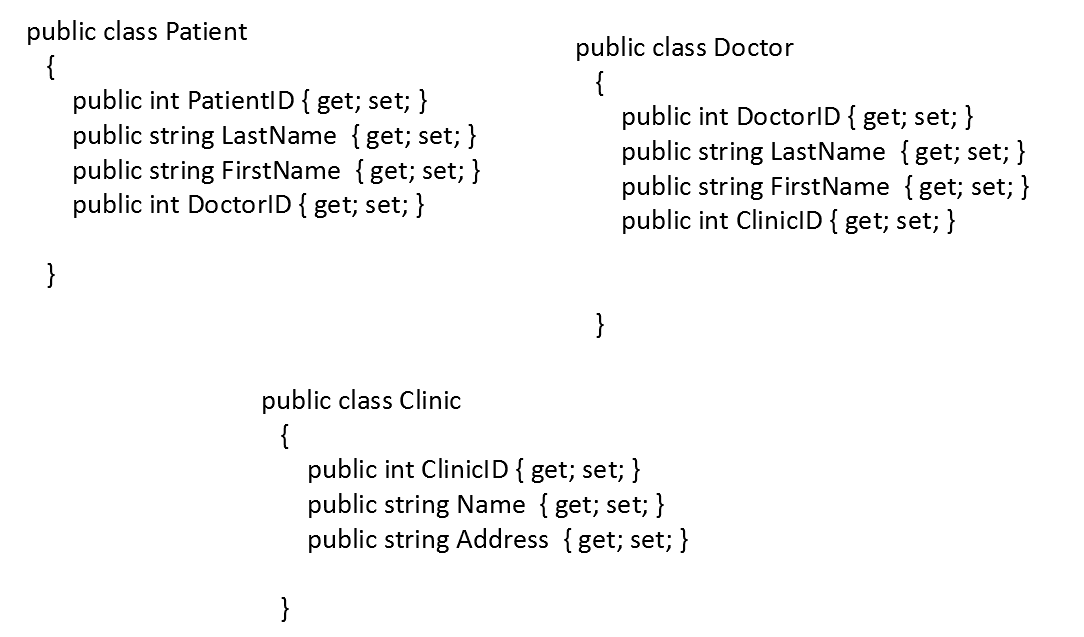


1. Some fields will need to be added to the tables so that our application will know how the entities are related.

Add any fields to the tables that are necessary. (***Some boxes may remain empty.*** )

1. Draw lines between the tables indicating the type of relationship (i.e. one-to-many, etc.).

3. Add the navigation properties to the classes below.



4. Check your work with the solution on the following page.

**Solution to the previous problem:**

