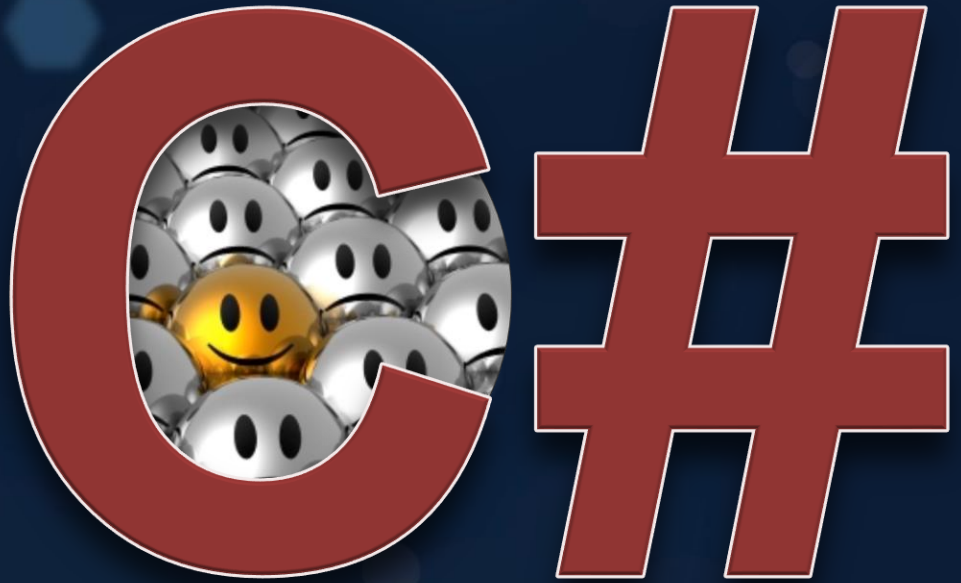
CSC103



Object Oriented Programming

Download class materials from
university.xamarin.com

Microsoft

Xamarin University

Objectives

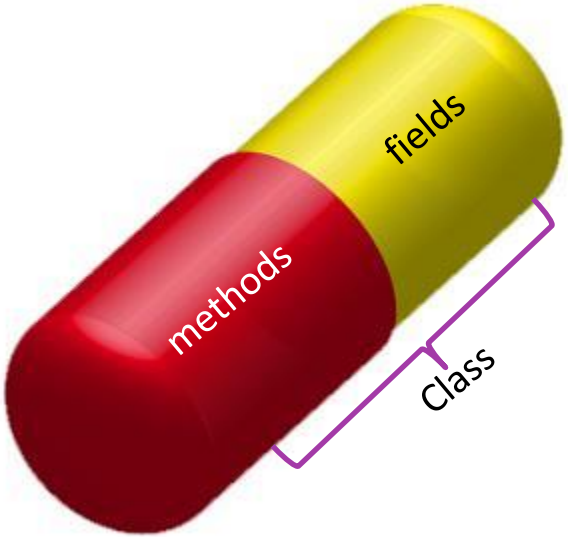
1. Determine classes and relationships in a program
2. Create a class with fields to maintain state
3. Use enumerations to define constant values

Determine classes and relationships in a program

Xamarin University

Tasks

1. Define Object Oriented Programming (OOP)
2. Work with classes and OOP in C#
3. Define fields in C#



Motivation

- You should strive to write software that is easy for you and other programmers to understand

Why is organization important?

- The way your program code is organized makes a huge difference in how easy it is to debug and maintain

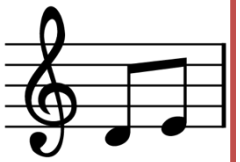
Faster to fix errors

Faster to add new features

What is Object-Oriented Programming?

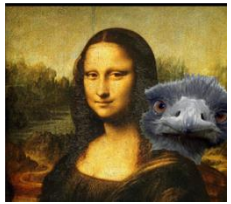
- *Object-Oriented Programming* (OOP) is a design philosophy invented to handle increasingly complex programs where we create *objects* to model things in the problem we are trying to solve (programmers call this the "problem domain")

Music app might have:
- Album
- Song
- Playlist
- …



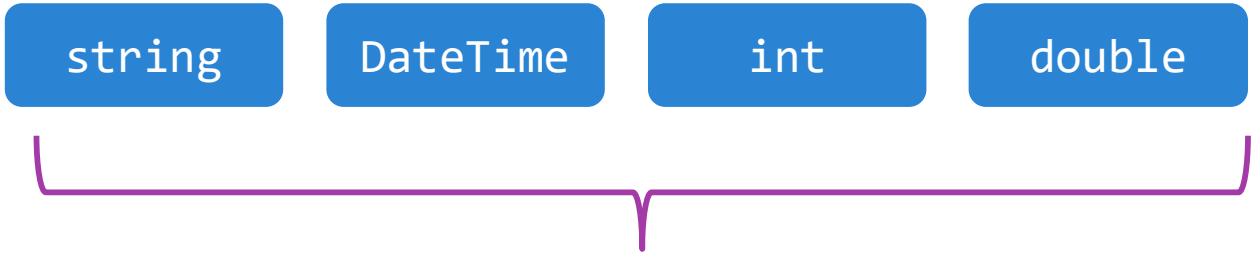
Nike app might have:
- Runner
- Workout
- Route
- …

Drawing app might have:
- Circle
- Square
- Eraser
- …



Defining new types

- The key idea of OOP is that the programmer can create new types to better model the world



remember, a *type* tells you the kind of object you are working with, these are examples of built-in types

What is a class?

- A class is a software model that defines a new type representing some concept or real-world element in your program

Just as this model represents an airplane and has many of the same elements

What is in a class?

- Classes contain data and behavior bundled together

Class

Fields

data the class "has"

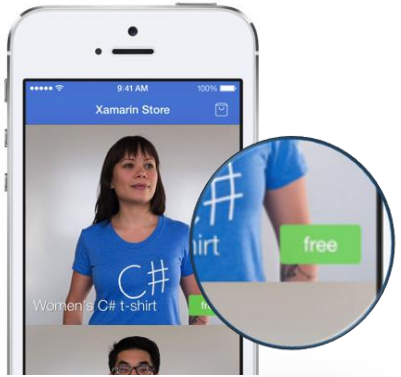
Methods

behavior the class "does"

What are fields?

- A field is a variable owned by the class that holds data

For a dog, the fields might include *name*, *age*, *weight*, and *breed*



For a button, the fields might include *width*, *height*, *position*, and *text*

What are methods?

- Methods are code blocks, containing C# statements, that provide logic to perform work related to the class

For a dog, the methods might include *bark*, *eat*, *walk*, *lick*, and *sniff*

For a button, the methods might include *show*, *hide*, *click* and *resize*

Flash Quiz

Xamarin University

Flash Quiz

① A class is a type that combines both data and behavior?

a) True

b) False

Flash Quiz

① A class is a type that combines both data and behavior?

a) True

b) False

Flash Quiz

② A field defines behavior associated with the class

a) True

b) False

Flash Quiz

② A field defines behavior associated with the class

a) True

b) False

Flash Quiz

③ You can only have one method in a class

a) True

b) False

Flash Quiz

③ You can only have one method in a class

a) True

b) False

How to identify classes?

- Classes are *models of things in the real world*; we can often identify the potential classes by examining what we need the app to do

What are the **potential classes** we might need for this mapping application to support route planning?

How to identify how to make a class

- Here are some possibilities – remember we want to accurately reflect the real world "things" we are working with

  - ✓ Map
  - ✓ Current Position
  - ✓ Calculated Route
  - ✓ Street
  - ✓ Turn

Flash Quiz

Xamarin University

Flash Quiz

① Name 3 potential classes that might be defined for a chat application

A chat application that allows users to converse using text messages

Flash Quiz

① Name 3 potential classes that might be defined for a chat application

1. Message
2. User
3. Conversation
4. Connection
5. History
6. ????

A chat application that allows users to converse using text messages

Flash Quiz

② Name three pieces of data (fields) we might want in a chat message

Flash Quiz

② Name three pieces of data (fields) we might want in a chat message

1. Date/Time
2. Text
3. Emoticon(s)
4. To User
5. From User
6. Color
7. ???

Flash Quiz #2

③ Name three methods (behavior) we might need in a chat message

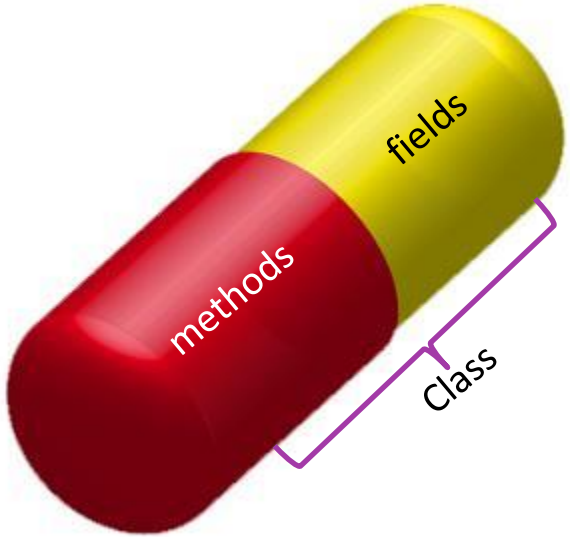
Flash Quiz

③ Name three methods (behavior) we might need in a chat message

1. Send
2. Take picture
3. Delete
4. Show keyboard
5. Reply
6. Add another person
7. Remove a person
8. Add attachment
9. Dial phone
10. ???

Summary

1. Define Object Oriented Programming
2. Define classes and OOP
3. Define fields and methods



Create a class with fields to maintain state

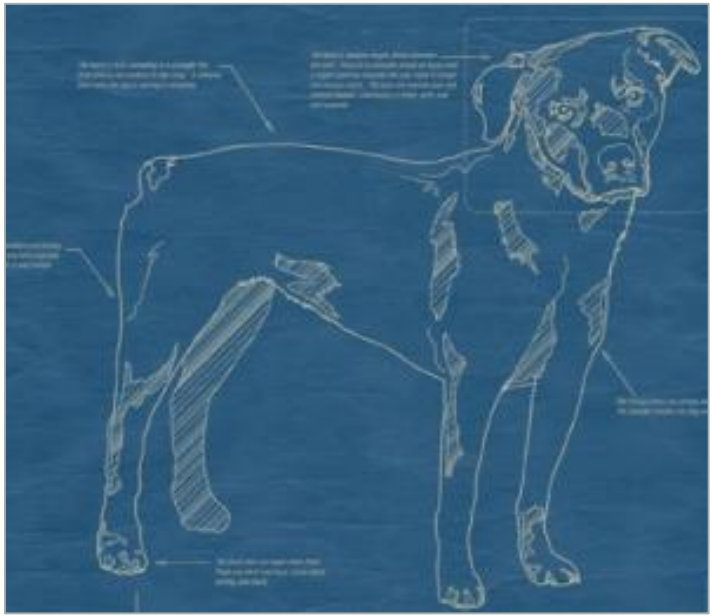
Xamarin University

Tasks

1. Create a class with fields
2. Instantiate objects of the class
3. Access the fields

What are classes and objects?

- A *class* defines a template and *objects* are instances of that template

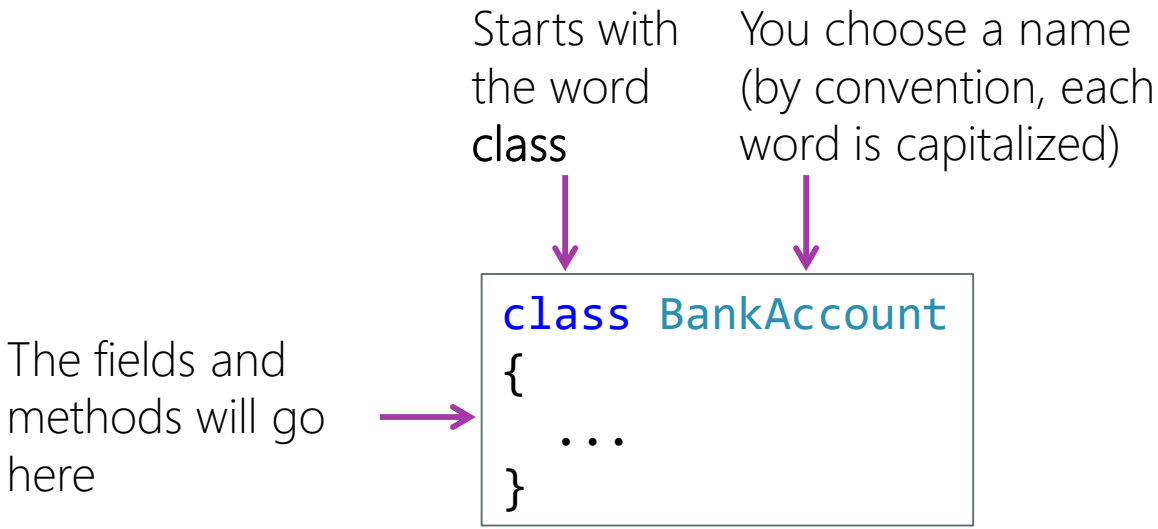


class **Dog**

instances of class **Dog**

How to define a class?

- A C# class has a name and a body, the body is delimited with **{** and **}**



Classes and files

- You should put each class in its own file (not required, but recommended)

File name should be the same as the class name with .cs extension

BankAccount.cs

```
class BankAccount
{
   ...
}
```

Classes and namespaces

- A *namespace* groups related classes together, useful for organization

BankAccount.cs

```
namespace Finance
{
  class BankAccount
  {
    ...
  }
}
```

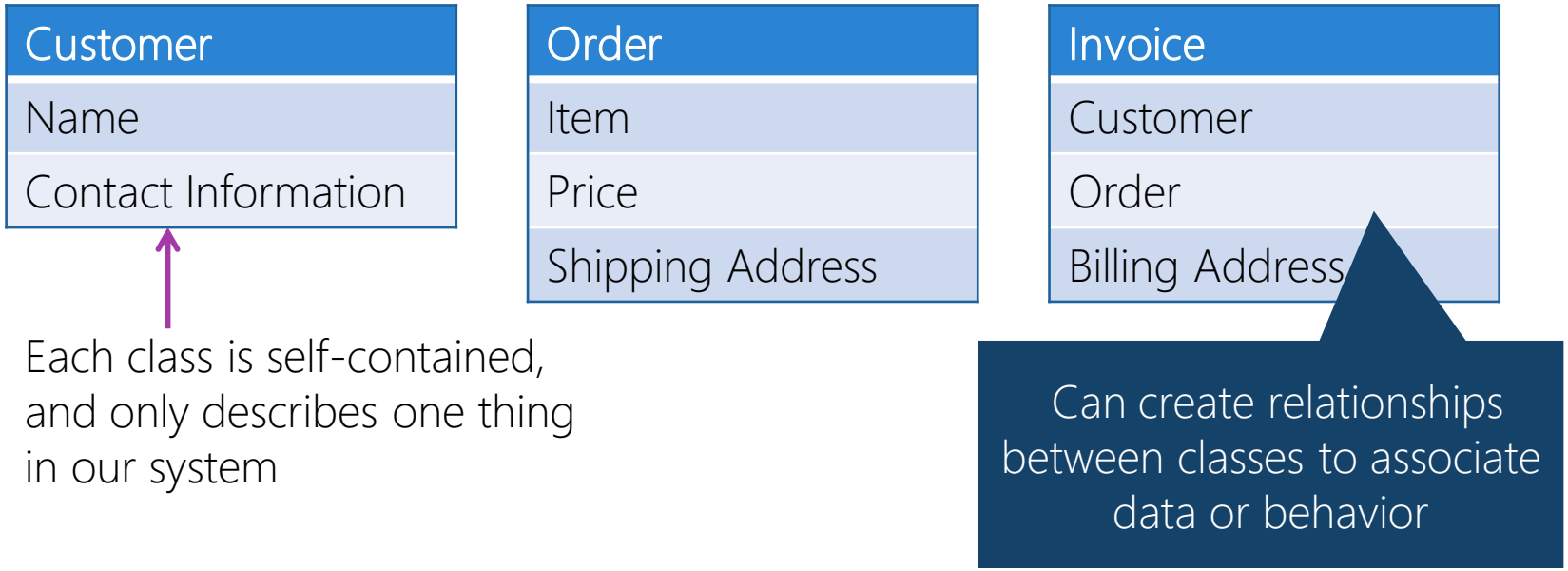
Typical to put classes inside a `namespace` to help describe their purpose

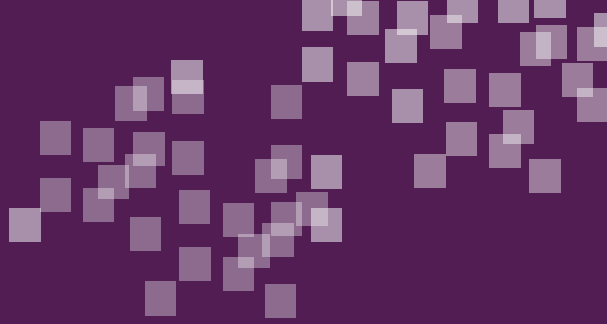
CreditCard.cs

```
namespace Finance
{
  class CreditCard
  {
    ...
  }
}
```

Class granularity

- Classes should be counted on to do one, well-understood thing





Group Exercise

Define a class and namespace to manage a road trip

Xamarin University

Declaring fields

- Variables declared inside the class define the fields

Three independent fields are included as part of the `BankAccount` class

```
class BankAccount
{
  string accountNumber;
  double balance;
  double interestRate;

  ...
}
```

Visibility

- Some things in a class are **public** – can and should be seen by other classes
- Other things in a class are **private** and should only be visible inside the class

Access modifiers

- *Access modifiers* limit what fields and methods contained in the object can be used from the outside

public

- anyone can use this field or method

private

- only the class can use this field or method
- this is the *default!*

Well-designed classes (public vs. private)

- Should carefully decide what visibility to make each field and method
  - Do other classes need access to this field, or is this field internal information that only needs to be used inside the class itself to make decisions?
  - Is this method useful by other classes, or is it a method that helps other methods inside the class do their job?

Remember that the *default* visibility for classes, fields and methods is always private

Declaring fields (for real)

private fields are only accessible from inside this class

public fields are values that other classes can access from outside the class, e.g. from **Main**

```
public class BankAccount
{
  private string accountNumber;

  public double Balance;
  public double InterestRate;

  ...
}
```

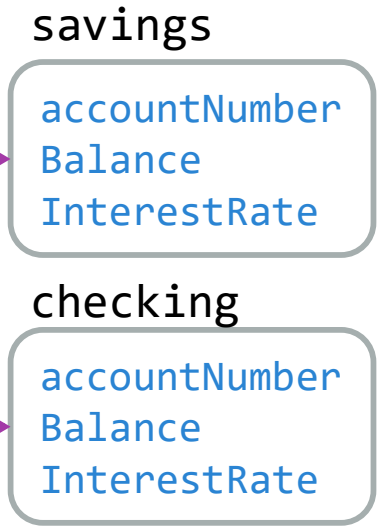
Typical to use different capitalization for **public** vs. **private**

private is used to "hide" implementation details, we will see more in a later course
public fields are not common, we will see why in a later course

How to create objects?

- Use **new** to create instances (objects) of a class, each object gets its own copy of the fields

```
public class Program
{
  public static void Main()
  {
    BankAccount savings  = new BankAccount();
    BankAccount checking = new BankAccount();
    ...
  }
}
```



How to access fields?

- Use the "*dot operator*" to access (read or change) the fields of an object

```
public class Program
{
  public static void Main()
  {
    BankAccount savings  = new BankAccount();
    BankAccount checking = new BankAccount();

    savings.Balance  = 100.00;
    checking.Balance = 500.00;

    double netWorth = savings.Balance + checking.Balance;
  }
}
```

Apply the "dot operator" to an object to access its members

savings

```
accountNumber
Balance          100.00
InterestRate
```

checking

```
accountNumber
Balance          500.00
InterestRate
```

Flash Quiz

Xamarin University

Flash Quiz

① An object defines an instance of a class

a) True

b) False

Flash Quiz

① An object defines an instance of a class

a) True

b) False

Flash Quiz

② To make a field accessible, you must add the _____ keyword

a) private
b) visible
c) public
d) None of the above

Flash Quiz

② To make a field accessible, you must add the _____ keyword

a) private
b) visible
c) public
d) None of the above

Flash Quiz

③ Select the missing code to complete the following program

```
public class Program
{
  public static void Main()
  {
    BankAccount savings  = new BankAccount();
    BankAccount checking = new BankAccount();

    savings.Balance  = 100.00;
    checking.Balance = 500.00;

    double netWorth = ??????????
  }
}
```

a) `savings.Balance + checking.Balance;`

b) `savings->Balance + checking->Balance;`

c) `savings.Balance + checking.Balance`

d) `savings.Balance & checking.Balance;`

Flash Quiz

③ Select the missing code to complete the following program

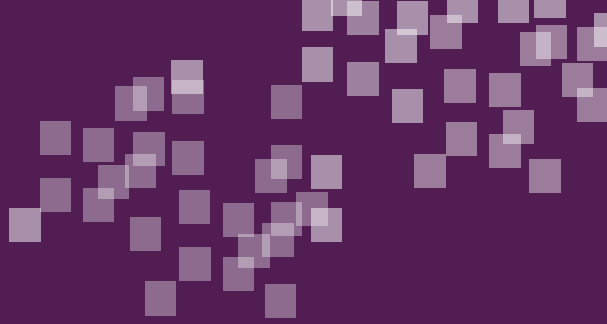
```
public class Program
{
  public static void Main()
  {
    BankAccount savings  = new BankAccount();
    BankAccount checking = new BankAccount();

    savings.Balance  = 100.00;
    checking.Balance = 500.00;

    double netWorth = ??????????
  }
}
```

a) **savings.Balance + checking.Balance;**

b) savings->Balance + checking->Balance;

c) savings.Balance + checking.Balance

d) savings.Balance & checking.Balance;



Individual Exercise

Planning a Road Trip

Summary

1. Create a class with fields
2. Instantiate objects of the class
3. Access the fields

Use enumerations to define constant values

Xamarin University

Tasks

1. Declare enumeration type by using the `enum` keyword
2. Use `enum` to define a set of named integral constants assigned to a variable

```
public enum Colors
{
    Red,
    Orange,
    Yellow,
    Green,
    Blue
    Indigo,
    Violet
}
```

What is an enumeration?

- An enumeration is a special C# data type that defines a set of integral constants as a group

Declare valid *values* to use with this enumeration, here we declare the valid English months

```
public enum Months
{
    January, February,
    March, April,
    May, June,
    July, August,
    September, October,
    November, December
}
```

Looks a bit like a class definition, except we use the keyword `enum` instead of class

How enum works

- C# assigns sequential integer values to each enumeration value, starting with zero (0) for the first one, *every defined value has a numeric value*

```
public enum Months
{
    January, February,
    March, April,
    May, June,
    July, August,
    September, October,
    November, December
```

January = 0
February = 1
...
December = 11

C# actually works with the numeric values – so when you use an enumeration, it's like using an integer value except that you are locked into a specific set of valid values

Changing the enum values

- Can assign specific values to each enumeration
- Unassigned values are assigned the prior value + 1
- Allows for duplicate numeric values to define *synonyms* to an enumeration value

```
public enum Months
{
    January = 1, February,
    March, April,
    May, June = 6,
    July, August,
    September, October,
    November, December = 12
}
```

Now our months would match

We really only need to assign the *first* value if we just want them to run sequentially – otherwise we can assign any number, in any order to any value

Using enums

- Enumerations are validated by the compiler – this ensures we pass in a correct value to a method or property that uses the **enum**

```
public int CalculateAgeFromBirthday(
                Months month, int day, int year)
{
    if (month == Months.January) {
       ...
    }
    ...
}
```

We get *named* values to compare against – and C# ensures that it's a Month, not just a number

Printing an enumeration

- Using **`ToString()`** on an enumeration will return the *text* value

```
Months december = Months.December;  // really 12

Console.WriteLine("{0} is {1}",
                  december.ToString(), (int) december);
```

December is 12

Can also *cast* the enumeration back to the underlying number – this is then printed as the numeric value

Converting a number to an enum

- C# allows you to cast numeric values to the appropriate `enum` type using an explicit cast – this can be useful when converting back and forth between numbers and enumerated values

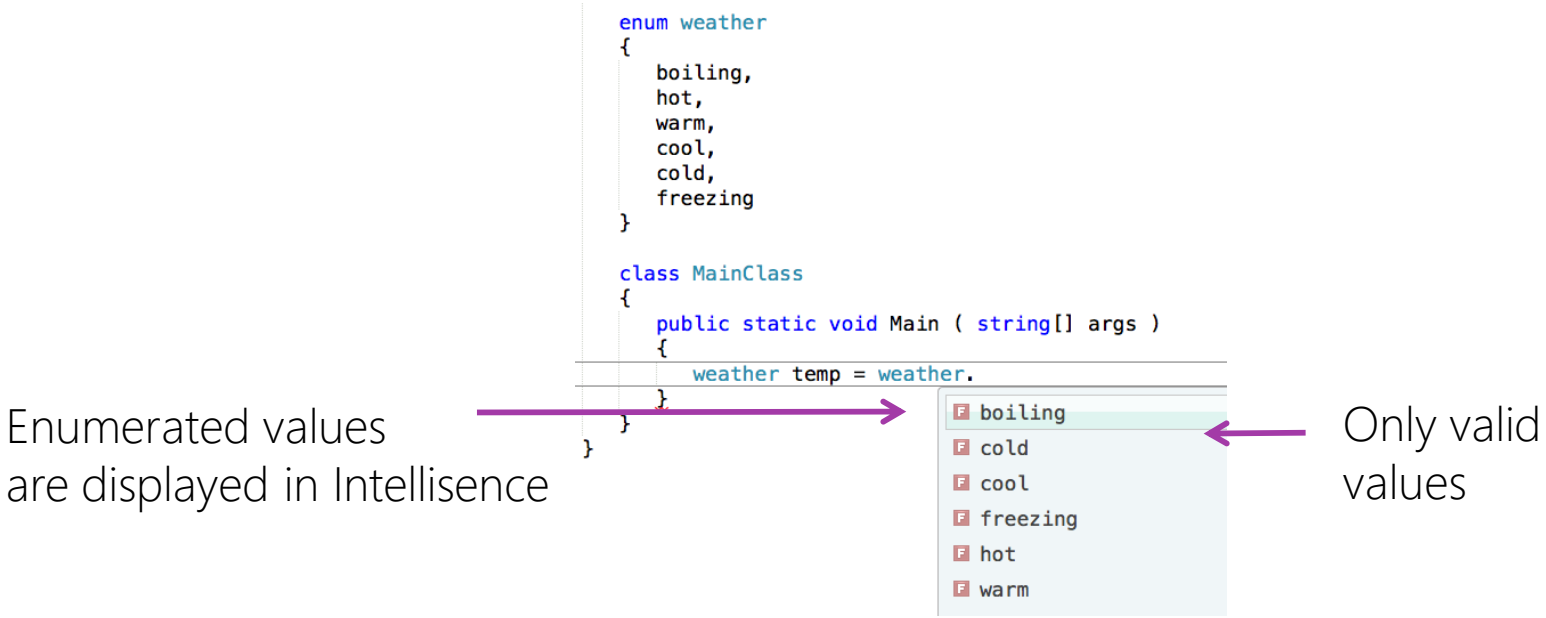
Tell C# to assign the `Months` enumeration for "6"

```
Months june = (Months) 6;

Console.WriteLine("{0} is {1}",
                  june.ToString(), (int) june);
```

```
June is 6
```

What are enums good for?

- Excellent for intellisense and auto-completion



Flash Quiz

Xamarin University

Flash Quiz

① If you don't assign a numeric value to the first enumeration, what will the default value be?

a) 0

b) 1

c) Undefined

Flash Quiz

① If you don't assign a numeric value to the first enumeration, what will the default value be?

a) 0

b) 1

c) Undefined

Flash Quiz

② Calling the `ToString()` method on an enumeration will print the numeric value

a) True

b) False

Flash Quiz

② Calling the `ToString()` method on an enumeration will print the numeric value

a) True

b) False

Summary

1. Declare enumeration type by using the `enum` keyword
2. Use `enum` to define a set of named integral constants assigned to a variable

```
public enum Colors
{
    Red,
    Orange,
    Yellow,
    Green,
    Blue
    Indigo,
    Violet
}
```

Where are we going from here?

- You now know how to define a new class with data in C# to represent an aspect of your program
- In the next course, we will look at how to define behavior for that class through methods

Thank You!

Please complete the class survey in your profile:
university.xamarin.com/profile