CSC104

Adding behavior into our apps

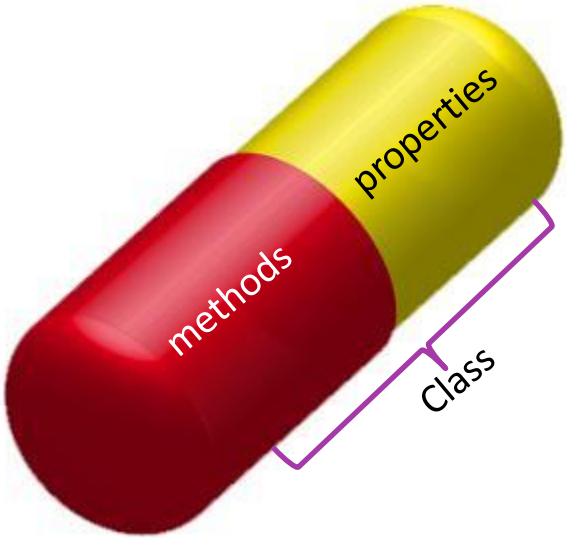
Download class materials from university.xamarin.com

Microsoft

Xamarin University

Objectives

1. Create a class with methods to provide behavior
2. Utilize properties to hide our fields



Create a class with methods to provide behavior

Xamarin University

Tasks

1. Define the usage of methods in a class
2. How to implement methods in a class
3. Call the methods on objects (instances) of the class
4. Passing data into methods
5. Returning data from methods

Reminder: 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

Reminder: what is in a class?

- Classes contain data and behavior bundled together

Class

Fields

Methods

data the class "has"

behavior the class "does"

Reminder: 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*

How do you define a method?

- A method is declared inside a class with a name that indicates what behavior or operation the method performs

method body is contained within open { and close } curly braces

```
public class BankAccount
{
  public double Balance;
  public double InterestRate;

  public void AddInterest()
  {
    double interest = Balance * InterestRate;
    Balance += interest;
  }
  ...
}
```

The method `AddInterest` calculates the interest based on our data fields and adjusts the balance

How do you define a method?

- A method is declared inside a class with a unique name that indicates what behavior or operation the method performs

void indicates it does not return any result value

public indicates it can be used from outside the class

```
public class BankAccount
{
  public double Balance;
  public double InterestRate;

  public void AddInterest()
  {
    double interest = Balance * InterestRate;
    Balance += interest;
  }
  ...
}
```

open (and close) parentheses after the name tells the compiler that this is a method declared for this class

What happens when you call a method?

- Calling a method on an object causes your program to execute the code contained in that method

Main program

```
code statement;
code statement;
AddInterest();
code statement;
...
```

BankAccount

```
public void AddInterest()
{
  double interest = Balance *
                    InterestRate;
   Balance += interest;
}
```

When the method finishes, your program *continues executing* the code that follows the call to the method

How do you call a method?

- Use the *dot operator* and *parentheses* to invoke a method on an object

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

    savings.Balance = 100.00;
    savings.InterestRate = 0.05;

    savings.AddInterest();
}
```

savings

accountNumber	
Balance	105.00
InterestRate	0.05

Calling the method **AddInterest()** will change the Balance to 105.00

Method parameters

- Sometimes methods need additional data in order to perform the logic required – this could be supplied by setting fields in the class
- If the data is only used by the method, a better approach is to pass the data inside the method call – this is called a *parameter*

depositing money would require some $$$ amount to add to our bank account

Passing method parameters

- Method parameters are additional pieces of information passed from the caller into the method (also known as *arguments*)

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

    savings.Balance = 100.00;

    savings.Deposit(50.00);
  }
}
```

```
public class BankAccount
{
  public double Balance;

  public void Deposit(double amount)
  {
    Balance += amount;
  }
  ...
}
```

Parameters act as *local variables* within the method

Method parameter validation

- Method parameters must define the type of value they expect – the compiler will enforce this and not allow unexpected values to be passed in

```
public class BankAccount
{
  public double Balance;

  public void Deposit(double amount)
  {
    Balance += amount;
  }
  ...
}
```

This method expects a double numeric value

```
BankAccount account = ...;

account.Deposit(500.0);

account.Deposit(true);

account.Deposit(500);

account.Deposit("500");
```

Passing multiple parameters

- Methods can take as many parameters as they need to perform their work

```
public class BankAccount
{
  private string accountNumber;
  public  double Balance;
  public  double InterestRate;

  public void Initialize(string account, double balance, double rate)
  {
    accountNumber = account;
    Balance       = balance;
    InterestRate  = rate;
  }
  ...
}
```

parameter #1 — parameter #2 — parameter #3

Each parameter can have a different type

How methods return values

- Methods can compute and return a single value to the caller, each method must declare the type it returns (or **void** to indicate no value)

Declare the return type

```
public void Withdraw()
{
    if (savings.IsOverdrawn() == true)
        return;
}
```

return keyword is used to return a single value, no code is executed after the return

```
public bool IsOverdrawn()
{
    return Balance < 0;
}
```

Individual Exercise

Building a Calculator

Xamarin University

Flash Quiz

Xamarin University

Flash Quiz

① Where do you define a method?

a) Inside the class with a unique name

b) Outside the class with a unique name

c) Inside the class with the class name

Flash Quiz

① Where do you define a method?

a) Inside the class with a unique name

b) Outside the class with a unique name

c) Inside the class with the class name

Flash Quiz

② What is the maximum number of parameters a method can have?

a) 2
b) 4
c) 6
d) As many as is needed

Flash Quiz

② What is the maximum number of parameters a method can have?

a) 2

b) 4

c) 6

d) As many as is needed

Naming your methods

- Methods should have names that reflect what the method *does*
- Don't be afraid of long method names
- The method name and parameters together are called the *method signature*

```
public class BankAccount
{
    ...

double Add(double amount);
bool NotPositive();

double Credit(double amount);
bool AccountIsOverdrawn();

}
```

What is method overloading?

- Sometimes two or more methods perform the same logic but require different parameters
- C# allows you to create more than one method with the *same name* but *different parameters*; this is called method overloading
  - different parameter types
  - different number of parameters

```
double Add(double x, double y);

int Add(int x, int y);

double Add(double x);
```

three variations of an **Add** method on a calculator, each taking different parameters

Return values

- Return values and visibility are not considered in method overloading

```
public class Calculator
{
   public double Add(double x, double y)
   {
      ...
   }

   private int Add(double x, double y)
   {
      ...
   }
}
```

```
error CS111: A member
'Calculator.Add(double,double)' is already defined.
Rename this member of use different parameter types
```

Individual Exercise

Adding onto our Calculator

Xamarin University

Flash Quiz

Xamarin University

Flash Quiz

① Overloaded methods with the same name must differ in

a) return type

b) number or type of parameters

c) a OR b

d) a AND b

Flash Quiz

① Overloaded methods with the same name must differ in

a) return type

b) <u>number or type of parameters</u>

c) a OR b

d) a AND b

Flash Quiz

② The method name and parameters together are called the *method signature*. True or False

a) True

b) False

Flash Quiz

② The method name and parameters together are called the *method signature*. True or False

a) <u>True</u>

b) False

Summary

1. Define the usage of methods in a class
2. How to implement methods in a class
3. Call the methods on objects (instances) of the class
4. Passing data into methods
5. Returning data from methods

Utilize properties to hide our fields

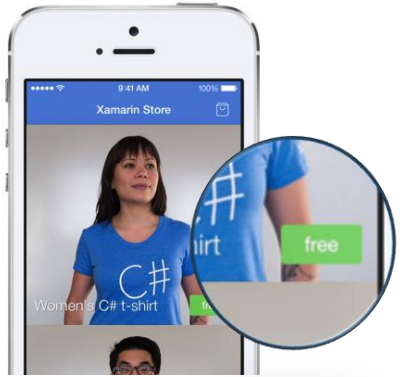
Tasks

1. What are properties?
2. Defining read-only properties
3. Working with auto properties

Reminder: 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*

The problem with fields

- When you make a field **public**, code outside your class can read and alter the value of the field

```
public class BankAccount
{
  public double Balance;
  ...
}
```

```
BankAccount account = ...;

account.Balance = 100.0;
...
account.Balance -= 200.0;
```

Here we are dropping our balance below zero, should that be allowed? How can my class stop this from happening?

The solution – methods!

- We can make fields private and then use methods to read and change the values – this allows our class to ensure the field is always valid

```
public class BankAccount
{
  private double balance;

  public double GetBalance() { return balance; }
  public void SetBalance(double value) {
    if (value >= 0)
       balance = value;
  }
  ...
}
```

```
BankAccount account = ...;

account.SetBalance(100.0);
...
account.SetBalance(
   account.GetBalance()
         - 200.0);
```

This solves our problem, but is more complex .. and ugly

What we really want is..

- Ideally, we could create something that *looks like a field*, but provides methods to get and set the stored value so we can control access to the data

```
BankAccount account = ...;

account.Balance = 100.0;
...
account.Balance -= 200.0;
```

This syntax is very elegant and natural

```
BankAccount account = ...;

account.SetBalance(100.0);
...
account.SetBalance(
  account.GetBalance() - 200.0);
```

... but this provides the *behavior* we want

What is a property?

- A C# property consists of a pair of keywords which provide access to a data value exposed by the class

```
public class BankAccount
{
  private double balance;
  public double Balance
  {
     get { return balance; }
     set {
        if (value >= 0)
           balance = value;
     }
  }
  ...
}
```

typically has a private field to store the value

What is a property?

- A C# property consists of a pair of keywords which provide access to a data value exposed by the class

```
public class BankAccount
{
  private double balance;
  public double Balance
  {
     get { return balance; }
     set {
        if (value >= 0)
           balance = value;
     }
  }
  ...
}
```

Property is **public** and Pascal-cased

Property body is enclosed in curly braces

What is a property?

- A C# property consists of a pair of keywords which provide access to a data value exposed by the class

get method used to retrieve the value

```
public class BankAccount
{
  private double balance;
  public double Balance
  {
    get { return balance; }
    set {
        if (value >= 0)
           balance = value;
    }
  }
  ...
}
```

What is a property?

- A C# property consists of a pair of keywords which provide access to a data value exposed by the class

```
public class BankAccount
{
  private double balance;
  public double Balance
  {
     get { return balance; }
     set {
        if (value >= 0)
           balance = value;
     }
  }
  ...
}
```

set method is used to change the value and provide any necessary validation

Where does the value come from?

- The value assigned to the property is passed in through keyword `value`

```
public class BankAccount
{
  private double balance;
  public double Balance
  {
     get { return balance; }
     set {
        if (value >= 0)
           balance = value;
     }
  }
  ...
}
```

```
account.Balance = 200;
```

200 is passed in as `value`

Using a property

- Code outside the class will use the property to access the data, it looks like a field but acts like a method

```
BankAccount account = ...;

account.Balance = 100.0;
Console.WriteLine(account.Balance);
...
account.Balance -= 200.0;
Console.WriteLine(account.Balance);
```

Here we attempt to change our balance to a negative value, but the property logic stops that and the value remains unchanged

```
100.0
100.0
```

Properties under the covers

- Properties are really methods that are being used – the getter is called whenever code *reads* the value, and the setter is called whenever code attempts to *alter* the value

```
BankAccount account = ...;
account.Balance = 100.0;
Console.WriteLine(account.Balance);
```

We write this nice, natural code which is easy to understand

```
BankAccount account = ...;
account.set_Balance(100.0);
Console.WriteLine(account.get_Balance());
```

… and C# turns that into calls to the defined property methods

Read-only properties

- Normally, we define both a getter and a setter on a property, but you can define a read-only property by leaving off the setter

Here, we only define a getter for the **Balance** property – outside code can only *read* the value

```
public class BankAccount
{
  private double balance;
  public double Balance
  {
    get { return balance; }
  }
  ...
}
```

Calculated values

- Properties are often used to provide access to *calculated* values which do not have an associated field, but are calculated when the getter is called

```
public class BankAccount
{
  public double ExpectedInterest
  {
     get { return balance * interestRate; }
  }
  ...
}
```

We calculate the expected interest each time something reads the property

Flash Quiz

Xamarin University

Flash Quiz

① Why should I prefer properties over fields?

a) They are easier to work with

b) They provide more control over the data

c) They are faster

d) They are the same – you can use either one

Flash Quiz

① Why should I prefer properties over fields?

a) They are easier to work with

b) <u>They provide more control over the data</u>

c) They are faster

d) They are the same – you can use either one

Flash Quiz

② I must supply both a **get** and a **set** accessor with every property

a) True

b) False

Flash Quiz

② I must supply both a **get** and a **set** accessor with every property

a) True

b) False

Flash Quiz

③ In the set method, the assigned value can be accessed with _____

a) parameter
b) Param
c) Value
d) value

Flash Quiz

③ In the set method, the assigned value can be accessed with _____

a) parameter

b) Param

c) Value

d) value

Simplifying our properties

- Properties are often just **simple wrappers around a private field** – no additional logic is provided beyond getting and setting the field's value

```
public class BankAccount
{
    private string id;
    public string Id
    {
        get { return id; }
        set { id = value; }
    }
}
```

Introducing Auto Properties

- When you don't need any additional logic for the property, you can use an *auto property*

```
public class BankAccount
{
    public string Id
    {
        get;
        set;
    }
}
```

You cannot supply method bodies for the getter or setter with auto properties

Read-only auto properties

- Can change the *visibility* of the getter or setter to control how outside code interacts with the property, this is often done to create read-only auto properties

```
public class BankAccount
{
    public string Id
    {
        get;
        private set;
    }
}
```

Visibility keyword is placed right before the get/set keywords – here we make the setter private

Individual Exercise

Working with properties

Xamarin University

Summary

1. What are properties?
2. Defining read-only properties
3. Working with auto properties

Where are we going from here?

- Now you know how to define methods and properties on your classes
- In the next course, we will look at how to create mobile applications with C# and Xamarin Forms

Thank You!

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

Microsoft