CSC106

Method parameters and constructors

Download class materials from
university.xamarin.com

Microsoft

Xamarin University

Objectives

1. Pass information into methods
2. Receive information from methods
3. Initialize objects using constructors

Pass information into methods

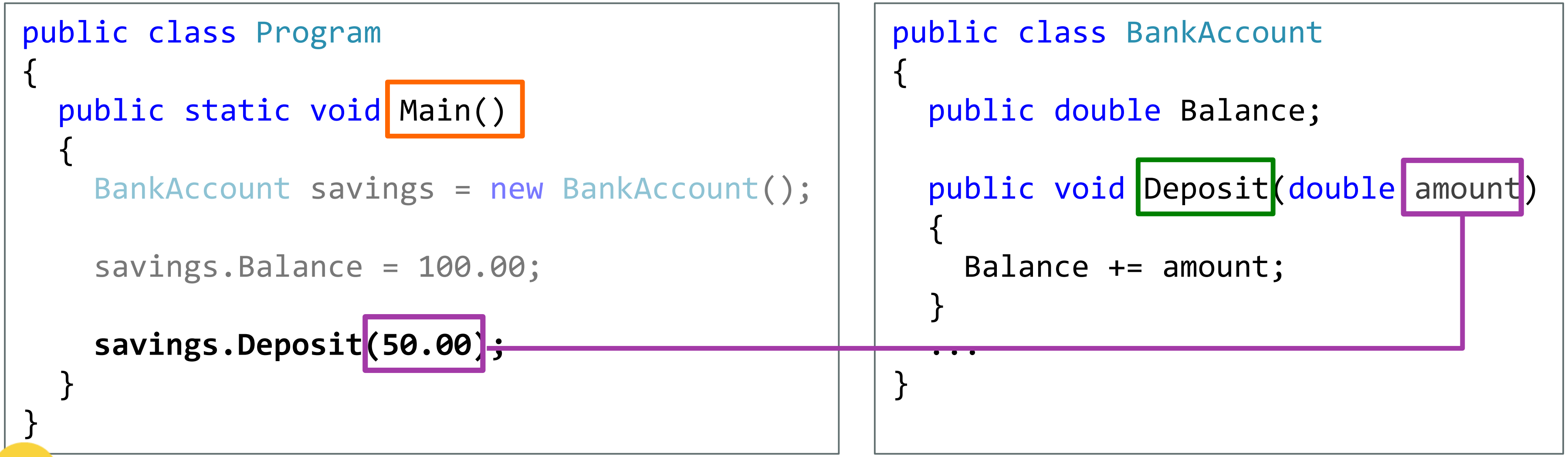
Xamarin University

Tasks

1. Pass method parameters
2. Pass optional parameters

Reminder: 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;
  }
  ...
}
```

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

Xamarin University

Method parameters are local variables

❖ When you pass parameters to a method, they **act as a local variable** inside the method

We use amount directly as if it were declared in our method →

```
public class BankAccount
{
  public void Deposit(double amount)
  {
     if (amount > 10000)
        ReportLargeDeposit(amount);
     ...
  }
}
```

Xamarin University

Optional Parameters

- C# supports *optional parameters* where the method declares a default value which is used if parameter not passed

```
double CalculateTax( double amount, double rate = 15.0 )
{
    return amount * rate / 100;
}
```

optional parameter(s) must be at the **end** of the parameter list

```
double tax = CalculateTax(2000);            // Rate = 15.0
```

```
double tax = CalculateTax(2000, 20);        // Rate = 20.0
```

Individual Exercise

Use optional parameters

Summary

1. Pass method parameters
2. Pass optional parameters

Receive information from methods

Xamarin University

Tasks

1. Return results from a method
2. Return multiple results from a method
3. Use out parameters
4. Call methods with *out* parameters

Xamarin University

Motivation

- We have used the `int.Parse()` method to do numeric conversions
- What happens when you pass in bad data (as shown in the code)?
- What we really want is **both** a conversion **and** a success result (`true` or `false`)

```
string text = "1K";

int num = int.Parse(text);
```

You get a runtime failure!

Reminder: returning 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;
}
```

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

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

What if I need more return values?

- Methods can return **zero** or **one** return value.. what if I need more?

```
public ??? CalculateMortgage(double loanAmount)
{
   double monthlyPayment = ...;
   double totalInterest = ...;

   return ???
}
```

How can we return *both* the monthly payment and total interest?

What if I need more return values?

❖ One solution would be to create a new class to hold our return data

```
public class MortgageInfo
{
   public double Payment { get; set; }
   public double TotalInterest { get; set; }
   ...
}
```

Add properties to hold the data we want to return

```
public MortgageInfo  CalculateMortgage(double loanAmount)
{
   ...
   MortgageInfo payment = new MortgageInfo {
      Payment = ...,
      TotalInterest = ...
   };
   return payment;
}
```

Return single object with all the data

Getting multiple values out of a method

- Methods can return more than one value by using `out` parameters

`out` is a keyword indicating that **the called method assigns a value** to the parameter **which updates the variable** that the caller passed in

```
public double CalculateMortgage(
                      double loanAmount, out double totalInterest)
{
   double monthlyPayment = ...;
   totalInterest = 10240;

   return monthlyPayment;
}
```

Out parameters *must* have a value assigned in the method. The caller's variables will have the assigned values.

Xamarin University

Getting multiple values out of a method

- Must add the **out** keyword to the parameter when *calling* the method as well – this make sure you know the value will be changed

```
public double CalculateMortgage(
                  double loanAmount, out double totalInterest) ...
```

Variable does not need to have an initial value before passing to the method

```
double totalInterest;
double monthlyPayment = bank.CalculateMortgage(10000,out totalInterest)
```

Xamarin University

Getting multiple values out of a method

```
double totalInterest;
double monthlyPayment = bank.CalculateMortgage(10000,out totalInterest)
```

fields start out with no value (uninitialized) →

totalInterest	n/a
monthlyPayment	n/a

```
public double CalculateMortgage(
                double loanAmount, out double totalInterest)
{
   totalInterest = loanAmount * 0.20;
   double monthlyPayment = (loanAmount + totalInterest) / 36;
   return monthlyPayment;
}
```

Getting multiple values out of a method

```
double totalInterest;
double monthlyPayment = bank.CalculateMortgage(10000,out totalInterest)
```

totalInterest	n/a
monthlyPayment	n/a

```
public double CalculateMortgage(
                double loanAmount, out double totalInterest)
{
   totalInterest = loanAmount * 0.20;
   double monthlyPayment = (loanAmount + totalInterest) / 36;
   return monthlyPayment;
}
```

Xamarin University

Getting multiple values out of a method

```
double totalInterest;
double monthlyPayment = bank.CalculateMortgage(10000,out totalInterest)
```

totalInterest	2000
monthlyPayment	n/a

```
public double CalculateMortgage(
                double loanAmount, out double totalInterest)
{
   totalInterest = loanAmount * 0.20;
   double monthlyPayment = (loanAmount + totalInterest) / 36;
   return monthlyPayment;
}
```

Xamarin University

Getting multiple values out of a method

```
double totalInterest;
double monthlyPayment = bank.CalculateMortgage(10000,out totalInterest)
```

totalInterest	2000
monthlyPayment	333.333333

```
public double CalculateMortgage(
                double loanAmount, out double totalInterest)
{
   totalInterest = loanAmount * 0.20;
   double monthlyPayment = (loanAmount + totalInterest) / 36;
   return monthlyPayment;
}
```

Xamarin University

Back to Int.Parse

- Built in method `int.TryParse()` uses this technique to provide a **safer numeric conversion**
- Returns `true` or `false` result based on the success of the conversion
- Uses an `out` parameter to give you the converted value or zero

```
string text = "1K";

int num;
if (int.TryParse(
        text, out num)) {
   // success, use number
}
else {
   Console.WriteLine(
    "Invalid number.");
}
```

This is the **preferred** way to convert strings to numeric values

Individual Exercise

Create a method that returns multiple values

Summary

1. Returning results from a method
2. Returning multiple results from a method
3. Using out parameters
4. Calling methods with *out* parameters

Initialize objects using constructors

Xamarin University

Tasks

1. Creating valid instances with a constructor
2. Passing values into constructors
3. Creating multiple constructors

Xamarin University

Motivation

- When we create an object (instance of a class), we need to make sure it's ready to be accessed

Ensure that fields are assigned appropriate values

Create (instantiate) child member objects

Xamarin University

What is a constructor?

- A constructor is a special piece of code that is called automatically by C# when the object is created, it is responsible for **initializing** the object

```
Dog lassie = new Dog();

// Use lassie here
```

C# does two things when we call `new` on a class – it **allocates** memory to hold the object, and then **builds** the object by calling the constructor method

How do I define a constructor?

- Constructors look like methods, but they have two unique characteristics in how they are defined

```
public class Dog
{
    public Dog( )
    {
       ...
    }
}
```

Constructor will not have a return type

Constructor always has the same name as the class

Why use a constructor?

- Constructors let you assign **default values** and ensure the object is correctly set up before it is used

We can assign reasonable values to any fields and properties, so clients do not get unexpected results when they use the object

```
public class Dog
{
    public string Breed { get; set; }

    public Dog()
    {
        Breed = "Unknown";
    }
}
```

Xamarin University

Default constructors

- A default constructor is any constructor that takes no parameters

```
public class Dog
{
   public int Age { get; set; }
   public string Breed { get; set; }
   public bool Pure { get; set; }

   public Dog()
   {
      Age = 0;
      Breed = "Unknown";
      Pure = false;
   }
}
```

Default constructors

- If you do not declare any constructors, the compiler will give you an invisible default constructor that does nothing

fields and properties are initialized to default values

```
public class Dog
{
    public int Age { get; set; }
    public string Breed { get; set; }
    public bool Pure { get; set; }
    // Age = 0;
    // Breed = null;
    // Pure = false;
}
```

`null` is a special value assigned to `string` and other non-numeric types that indicates it has *no value*, this is different from an empty string (`""`) which is a string with no data

Assigning values to object

- Can assign properties after the object is constructed

```
public class Dog
{
    public string Breed { get; set; }

    public Dog() // Default constructor
    {
        Breed = "Unknown";
    }
}
```

```
Dog lassie = new Dog ();
lassie.Breed = "Collie";
...
```

To use the dog, we assign the **`Breed`** property *after* we create the object.

Xamarin University

Passing parameters to constructors

❖ We can pass parameters to the constructor so we can set fields and properties in the constructor itself

```
public class Dog
{
    public string Breed { get; set; }

    public Dog(string breed)
    {
        Breed = breed;
    }
}
```

Define parameters to be passed into the constructor

```
Dog dog = new Dog();
dog.Breed = "Collie";


Dog dog = new Dog("Collie");
```

Passing parameters to constructors

- If you declare a constructor with parameters you must supply the parameters when you create the object

```
public class Dog
{
   public Dog(string breed) { ... }
   ...
}
```

❌

```
Dog dog = new Dog();  // not valid
```

This can be very beneficial if the class *must* have the supplied information to make the object valid, but what if it's not required?

Multiple constructors

- You can *overload* the constructor (create multiple constructors) to support different requirements

Declare a default constructor to allow for our simpler creation scenario

```
public class Dog
{
    public string Breed
        { get; set; }

    public Dog(string breed)
    {
        Breed = breed;
    }

    public Dog()
    {
        Breed = "Unknown";
    }
}
```

Xamarin University

Chaining constructors

- One constructor can **call another** to share the initialization code

Here, the default constructor calls the first constructor with a parameter

```
public class Dog
{
    public string Breed
        { get; set; }

    public Dog(string breed)
    {
        Breed = breed;
    }

    public Dog()
      : this("Unknown") {}
}
```

Group Exercise

Adding constructors to your class

Xamarin University

Flash Quiz

Flash Quiz

① What are characteristics of a constructor?

a) A constructor has the same name as the class

b) A constructor never has a return type, including void

c) Both

d) Neither

Flash Quiz

① What are characteristics of a constructor?

a) A constructor has the same name as the class

b) A constructor never has a return type, including void

c) Both

d) Neither

Flash Quiz

② True or False: When you create an object, you can initialize some or all the fields or properties

a) True

b) False

Flash Quiz

② True or False: When you create an object, you can initialize some or all the fields or properties

a) True

b) False

Flash Quiz

③ True or False: You use optional parameters with constructors

a) True

b) False

Flash Quiz

③ True or False: You use optional parameters with constructors

a) [✓ True]

b) False

Summary

1. Creating valid instances with a constructor
2. Passing values into constructors
3. Creating multiple constructors

Thank You!

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

Microsoft