Agenda

- Method
- Declaring a Method
- Method Calling
- Method Types
- Method Overloading

Method

- A method is:
  - A code block that contains a series of statements. A program causes the statements to be executed by calling the method and specifying any required method arguments. Every executed instruction is performed in the context of a method.
  - Used to access and process data contained in the object.
  - Used to provide responses to any messages received from other objects.
  - The executable code that implements the logic of a particular message for a class.
  - An operation or function that is associated with an object and is allowed to manipulate that object's data.

Creating a Method (1 of 3)

2f4869676820706572666f726d616e63652e2044656c6976657265642e2f4869676820706572666f726d616e63652e2044656c6976657265642e2f4869676820706572666f726d616e63652e2044656c6976657265642e2f4869676820706572666f726d616e63652e2044656c6976657265642e2f4869676820706572666f726d616e63652e2044656c6976657265642e2f48696768207

- The following is a general form of a Method declaration.

Syntax:

```
modifiers type method-name (formal-parameter-list)
{
 method_body
}
```

- Method declaration consists of five components:
  - Methods Modifiers (modifiers).
  - Name of the Method (method-name).
  - Type of value the Method returns (type).
  - List of parameters (formal-parameter-list).
  - Body of the Method (method_body).

Creating a Method (2 of 3)

- Steps in declaring a method:
  - Set the return type.
  - Provide method name.
  - Declare formal parameters.
- Method signature:
  - Consists of the method name and its parameters.
  - Must be unique for each method in a class.
- return statement:
  - Allows the method to return a value to its caller.
  - Also means to stop the execution of the current method and return to its caller.
  - Implicit return at the end of the method.

```
Using System;
class Number
 {
   int multiply(int i, int j)
    {
          return i*j;
     }

   int divide(int i, int j)
    {
          return i/j;
     }

   void printSum(int i, int j)
     {

          Console.WriteLine(i+j
   );

    }

  double getPi()
    {
          return
   3.14159265358979;
    }

}
```

Creating a Method (3 of 3)

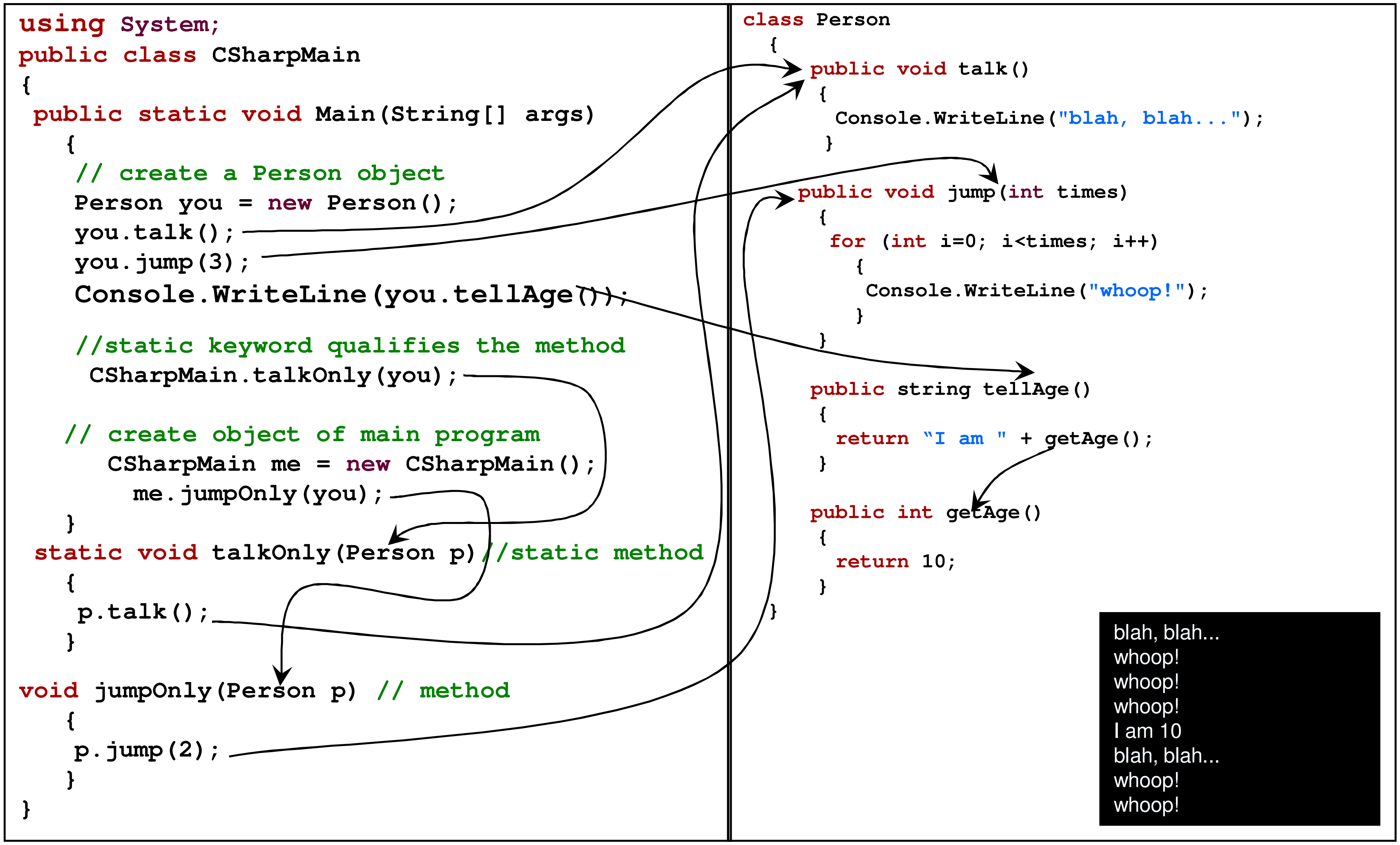
- A method that does not return a value must specify void as its return type.
- A method with empty parameters.

```
Using System;
class Number
 {
    int multiply(int i, int j)
     {
          return i*j;
     }

    int divide(int i, int j)
     {
          return i/j;
     }

    void printSum(int i, int j)
      {
          Console.WriteLine(i+j);

     }

    double getPi()
     {
          return 3.14159265358979;
    }

}
```

Method Calling / Invoking

- The process of activating a method is known as invoking or calling.
- The steps to call a method are:
  1. Method name should match.
  2. Number of parameters should match.
  3. Type of parameters should match.
- Ways of calling a method include:
  1. Calling a method through its object name.
  2. Calling a method within the same class.
  3. Calling a static method through its class name.

Calling a Method



```
using System;
public class CSharpMain
{
 public static void Main(String[] args)
   {
    // create a Person object
    Person you = new Person();
    you.talk();
    you.jump(3);
    Console.WriteLine(you.tellAge());

    //static keyword qualifies the method
     CSharpMain.talkOnly(you);

   // create object of main program
      CSharpMain me = new CSharpMain();
        me.jumpOnly(you);
   }
 static void talkOnly(Person p) //static method
   {
    p.talk();
   }

void jumpOnly(Person p) // method
   {
    p.jump(2);
   }
}
```

```
class Person
   {
    public void talk()
     {
      Console.WriteLine("blah, blah...");
     }
    public void jump(int times)
     {
      for (int i=0; i<times; i++)
        {
         Console.WriteLine("whoop!");
        }
     }
    public string tellAge()
     {
      return "I am " + getAge();
     }
    public int getAge()
     {
      return 10;
     }
   }
```

```
blah, blah...
whoop!
whoop!
whoop!
I am 10
blah, blah...
whoop!
whoop!
```

Types of Methods

- C# employs four kinds of parameters that are passed to methods:
  - Value Type parameters:
    - Used for passing parameters into methods by ***value***.
  - Reference Type parameters:
    - Used to pass parameters into methods by ***reference***.
  - Output parameters:
    - Used to ***pass results back*** from a method.
  - Optional Parameters:
    - Used to pass ***optional*** parameter.
  - Named Argument:
    - Used to pass ***argument by position***.
  - Parameter arrays:
    - Used in a method definition to enable it ***to receive variable number of arguments*** when called.

Passing Value Type Parameters

- A value-type variable contains its data directly:
  - By passing a value-type variable to a method which passes a copy of the variable to the method.
  - Changing the parameter value inside the method does not change the original data stored in the variable.

```
using System;
public class PassingValByVal
 {
  public static void SquareIt(int x) // The parameter x is passed by value.
  {
   x *= x; //Changes to x will not affect the original value of x.
   Console.WriteLine("The value inside the method: {0}", x);
  }

  static void Main()
  {
   int n = 5;
   Console.WriteLine("The Value before calling the method: {0}", n);
   SquareIt(n); // Passing the variable by value.
   Console.WriteLine("The value after calling the method: {0}", n);
  }
 }
```

```
The value before calling the method: 5
The value inside the method: 25
The value after calling the method: 5
```

Reference Parameters

- A variable of a reference type does not contain its data directly.
- It contains a reference to its data.
- Passing a reference-type parameter by reference, it is possible to change the data pointed to, such as the value of a class member.

```
i = 2 j = 3
i = 3 j = 2
```

```
using System;
class PassingValByRef
 {
  static void SwapByRef(ref int x, ref int y)
   {
    int temp = x;
    x = y;
    y = temp;
   }
   public static void Main()
   {
    int i = 2;
        int j = 3;
    Console.WriteLine("i = {0} j = {1}", i, j);
    SwapByRef(ref i, ref j);
    Console.WriteLine("i = {0} j = {1}", i, j);
   }
}
```

Output Parameters

- Out keyword causes arguments to be passed by reference:
  - It is similar to the ref keyword.
  - The difference is ref requires the variable to be initialized before being passed.
- To use an out parameter, both the method definition and the calling method must explicitly use the out keyword.

```
Using System;
class OutReturnExample
{
 static void Method( out int i,out string s1, out
    string s2)
 {
  i = 44;
  s1 = "I've been returned";
  s2 = null;
 }
 static void Main()
 {
  int value;
  string str1;
  string str2;
  Method(out value, out str1, out str2);
  // value is now 44
  // str1 is now "I've been returned"
  // str2 is (still) null;
   Console.WriteLine("{0},{1},’’{2}‘’”,value, str1
    ,str2);
 }
}
```

```
44, I’ve been returned, ‘’  ‘’
```

Optional Parameters

- Methods can declare optional parameters.
- A parameter is optional if it is specified as a default value while declaring.
- Optional parameter can be omitted while calling the method.
- Optional parameter cannot be marked with ref or out keywords.
- Mandatory parameters occur before optional parameters in method declaration and method calling.

```
Using System;
class OptionalExample
{
 static void Method(int x=25)
 {
  Console.WriteLine("{0}",x);
 }
 static void Main()
 {

  Method(); // displays 25

 }
}
```

Named Arguments

- Identifying an argument by name.
- Named arguments can occur in any order.
- Can mix named and positional parameters.
- Positional parameter must come before named arguments.

```
Using System;
class NamedParameterExample
{
 static void Method(int x,int y)
 {

  Console.WriteLine(“x={0}y={1}”,x,y);
 }
 static void Main()
 {

  Method(x:10,y:20); // displays x=10 y=20
  Method(y:11,x:22); // displays x=22 y=11

 }
}
```

Parameter Arrays

- The params keyword allows you to specify a method parameter that takes an argument where the number of arguments is variable.
- No additional parameters are permitted after the params keyword in a method declaration, and only one params keyword is permitted in a method declaration.

Parameter Arrays (cont.)

```
Using system;

public class MyClass
{
 public static void UseParams(params int[] list)
 {
  for (int i=0 ; i < list. Length; i++)
  {
   Console.WriteLine(list[i]);
  }
   Console.WriteLine();
 }

public static void UseParams2(params object[] list)
{
 for (int i = 0 ; i < list.Length; i++)
 {
  Console.WriteLine(list[i]);
 }
  Console.WriteLine();
}
```

```
static void Main()
 {
  UseParams(1, 2, 3);
  // An array of objects can also be passed, as long as
  UseParams2(1, 'a', "test");
  // the array type matches the method being called.
  int[] myarray = new int[] {10,11,12,13};
  UseParams(myarray);
 }
}
```

```
1
2
3

1
a
test

10
11
12
13
```