UNIVERSITY OF Hull

Constructing Class Instances

Rob Miles

Department of Computer Science

29b Programming 2

Using New

```
Account test = new Account();
```

- We have seen that when we want to make an instance of a class we have to use `new`
- This creates an object in memory which the tag is connected to
- Now we are going to find out how `new` really works

Calling a Constructor

```
Account test = new Account();
```

- The code after the `new` keyword looks a lot like a method call
  - Although the method has no parameters
- This is because it actually is a method call
- The method that is called is known as the *constructor*

What is the constructor for?

- The constructor is provided so that programmers can get control at the point in the program where an instance of a class is being created
- The constructor method is called automatically during object creation
- It means that we can create code to set up an object before it is used in the program
- It is how we ensure that objects have integrity from the start of their lifetime

Why have we not had to make constructors before?

- If you don't provide a class with a constructor the C# compiler will create one for you
- This "default" constructor has no parameters and does nothing
- It is called when you create a new instance

```
Account a; // declare the reference
a = new Account(); // constructor
                   // called
```

Making our own constructors

```
public class Account
{
    public Account ()
    {
       Console.WriteLine ("Account made");
    }
}
```

- We can add a constructor method to any class
- The constructor method has the same name as the class
- It does not return anything
- Each time we make a new Account the method is called

Useful Constructors

- A constructor is a method like any other
- When the constructor finishes the object is ready for use in the program
- A constructor can be supplied with parameters that can be used during object construction to set initial values
- In the case of the bank we might want to set the name, address and initial balance of an account when we create it

Account Constructor

```
class Account
{
  // private member data
  private string name;
  private string address;
  private decimal balance;

  // constructor
  public Account (string inName, string inAddress,
    decimal inBalance)
  {
    name = inName;
    address = inAddress;
    balance = inBalance;
  }
}
```

Making an Account

```
Account robsAccount;
robsAccount = new Account("Rob Miles", "Hull", 0);
```

- The constructor will run when the account is created and set up the account values
- Our program never calls the constructor method directly, it is called for us when an object is created
- Constructors have been running every time we have created an object

The Default Constructor

- Once we create our own constructor the compiler stops giving us the "free" one
- For the Account class above the only way to construct an instance is to provide the name, address and balance values
  - i.e. we must give these parameters each time we use new to create an Account instance
- If we don't the program won't compile
- This is good, because it forces other programmers to create objects “our” way

Overloading Constructors

- C# lets you *overload* methods in a class
- When you overload a method you create a new method with the same name but a different *signature*
- This is useful if you have a particular task that can be done in several ways
  - There are lots of ways to define the value of a date:
    - DD/MM/YY – 26/02/2010
    - DD MMM YY – 26 Feb 2010
  - We could provide multiple SetDate methods depending on the parameters to be used

Signatures and Overloading

```
public void silly (int i)
{
}

public void silly (int i, int j)
{
}
```

- The method name `silly` has been overloaded
- There are two different method signatures for `silly`
  - A single integer
  - Two integers
- The compiler can work out which to use from the context of the call of the method

Multiple Account Constructors

```
Account rob = new Account ( "Rob", "Hull" );
Account jim = new Account ( "Jim", "Beverly", 100 );
```

- We might want to provide more than one constructor for the Account class:
  - Sometimes we don't have a balance value, so the balance should automatically be set to 0
  - We could provide a constructor that only accepts the name and address information
- We can use overloading to achieve this

Signatures and Overloading

```
public Account (string inName, string inAddress,
  decimal inBalance)
{
  name = inName;
  address = inAddress;
  balance = inBalance;
}


public Account (string inName, string inAddress)
{
  name = inName;
  address = inAddress;
  balance = 0;
}
```

- There are now two ways an Account can be created

Using `this` in Constructors

- There is some code duplication in the constructors we have written
  - I hate code duplication
  - It means that if I fix a bug I might have to fix it in lots of different places in my code
  - I try and write my code once, and once only
- It would be easier if we could make one "master" constructor and then use that from all the others
- You can do this by using the keyword `this`

Account construction

```
public Account (string inName, string inAddress,
  decimal inBalance)
{
  name = inName;
  address = inAddress;
  balance = inBalance;
}

public Account (string inName, string inAddress) :
  this (inName, inAddress, 0 )
{
}

public Account (string inName) :
   this (inName, "Not Supplied")
{
}
```

Constructors Chaining

- This technique is called *constructor chaining*
- We design a set of constructors which are all linked back to the "master" constructor which sets all the properties
- How your constructors work is something you should consider when you design your objects

Object Integrity

- Whenever we set a value in our object we are supposed to be very careful that the new value is valid
  - Don't want to set an empty name or a stupid balance value on our Account
- We solve this by using validation on the values that the object is given
- If the new value is invalid we reject it

Constructors and Validation

- A constructor can validate the values being supplied to set up an object, but if it decides a value is incorrect it cannot reject it

A constructor cannot fail

- When you create a new instance, even if the constructor doesn't like the values it has been given, when the constructor finishes the objects will be created

Constructing Invalid Objects

- We don't want to create invalid objects so we need a way to handle this
- Since the constructor will create an instance if it completes the only way to resolve this is to have the constructor fail to complete
- It can do this by throwing an exception
- This will transfer execution to an exception handler or stop the program

Throwing Exceptions

- We have seen exceptions before
- They have been thrown at our code
  - When Parse fails it throws an exception
  - When we fall off the end of an array an exception is thrown
- An exception should be "the weapon of last resort"
- Only throw an exception when you can't do anything else

Exceptions in a constructor

```
public Account (string inName, string inAddress){
  if ( SetName (inName) == false ) {
    throw new Exception ("Bad name " + inName) ;
  }
  if ( SetAddress (inAddress) == false ) {
    throw new Exception ("Bad address " + inAddress);
   }
}
```

- This version of the constructor uses the Set methods to validate the supplied values
- If either of the methods fail the constructor throws an exception

Exception Etiquette

- Only throw an exception if you have no other way of resolving the situation
- Make sure that people who use your objects know that the constructor might throw an exception
- They can then use try – catch to recover
- Make your exceptions as useful as possible

Assembling an error message

```
public Account (string inName, string inAddress)
{
   string errorMessage = "";
   if ( SetName (inName) == false )
   {
      errorMessage = errorMessage + "Bad name " + inName;
   }
   if ( SetAddress (inAddress) == false )
   {
      errorMessage = errorMessage + " Bad addr " +
                     inAddress;
   }
   if ( errorMessage != "" )
   {
     throw new Exception ("Bad account" + errorMessage) ;
   }
 }
```

Construction Summary

- A constructor method gets control each time a new instance of a class is created
- The compiler provides a default constructor
- You can create your own constructor method and use overloading to provide multiple versions
- Constructors cannot fail, but they can throw exceptions so that they don't complete