Advanced C#

Optional and Named parameters

InfoSupport

Optional and Named parameters

- Two distinct features
- Often useful together

Optional Parameters

- Method declaration:

```
public void M(int a, int b = 3, int c = 5)  { ... }
```

- Method calls:

```
M(2);           // a=2, b=3, c=5
M(2, 2);        // a=2, b=2, c=5
M(2, 2, 2);     // a=2, b=2, c=2
```

- Calls are converted by compiler to:

```
M(2, 3, 5);
M(2, 2, 5);
M(2, 2, 2);
```

Optional Parameters (constraints)

- Optional parameters
  - Must appear after all required parameters
  - May not be a `ref` or `out` parameter
  - May appear in Constructors, Indexers, and Methods
- Default value must be compile-time constant
- Calls may only omit the last parameter(s)

```
M(2, 2);        // a=2, b=2, c=5
M(2, , 2);      // a=2, b=3, c=2   <= NOT ALLOWED
```

Optional Parameters (overloading)

- A method may *not only* be overloaded in the 'optionality' of a parameter:

```
public void M(int a, int b)  { ... }
public void M(int a, int b = 3)  { ... }  //  <= NOT ALLOWED
```

- But it may be overloaded by adding extra optional parameters:

```
public void M(int a, int b = 3)  { ... }
public void M(int a, int b = 3, int c = 5)  { ... }
```

- For resolving, signatures without unnecessary optional parameters are preferred:

```
M(2);          // calls M(int a, int b = 3)
M(2,2);        // calls M(int a, int b = 3)
M(2,2,2);      // calls M(int a, int b = 3, int c = 5)
```

Named Parameters

- Solution for leaving out optional parameters that don't come last

```
public void M(int a, int b = 3, int c = 5)  { ... }
```

```
M(2, , 2);      // a=2, b=3, c=2    <= NOT ALLOWED
M(2, c:2);      // a=2, b=3, c=2    <= OK!!
```

- can also be used for required parameters:

```
M(c:2, a:2);   // a=2, b=3, c=2
```

- is translated on caller-side to:

```
M(2,3,2);
```

Named Parameters (polymorphism)

- Watch out!!

```
public class Base
{
        public virtual void M(int a, int b)  { ... }
}
public class Derived : Base
{
        public override void M(int b, int a)  { ... }
}
```

```
Derived d = new Derived();
d.M(a: 5, b: 7);        // calls M(7,5)

Base b = d;
b.M(a: 5, b: 7);        // calls M(5,7)  !!
```

Named Parameters (constraints)

- All required parameters must appear
- You CAN use ref and out parameters:

```
public void M(ref int a, out int b)  { ... }
```

```
int x = 5, y;
M(b: out y, a: ref x);
```

- Different parameter order means different execution order:

```
public void M(int a, int b)  { ... }
```

```
int x = 7;
M(a: x++,  b: x  );  // calls M(7,8)
x = 7;
M(b: x,    a: x++);  // calls M(7,7)
```

COM Interop

- New features
  - COM type `variant` replaced by `dynamic`
  - COM `ref` parameters replaced by .NET `by value` parameters
  - COM Interop assemblies no longer necessary at run-time

COM variant replaced by dynamic

```
object customProps = doc.CustomDocumentProperties;
Type customPropsType = customProps.GetType();
object myCustomProperty =
    customPropsType.InvokeMember( "Item",
        BindingFlags.Default | BindingFlags.GetProperty,
        null, customProps,
        new object[] {"MyCustomProperty"});
Type myCustomPropertyType = myCustomProperty.GetType();
string myPropertyValue =
    myCustomPropertyType.InvokeMember( "Value",
        BindingFlags.Default | BindingFlags.GetProperty,
        null, myCustomProperty,
        new object[] { }).ToString();
```

- becomes:

```
string propertyValue =
    doc.CustomDocumentProperties["MyCustomProperty"].Value;
```

COM ref parameters

```
object missing = System.Reflection.Missing.Value;
object readOnly = false;
object isVisible = true;
object fileName = "NewTest.doc";
Microsoft.Office.Interop.Word.Document aDoc =
WordApp.Documents.Open(
     ref fileName, ref missing, ref readOnly, ref missing,
     ref missing, ref missing, ref missing, ref missing,
     ref missing, ref missing, ref missing, ref isVisible,
     ref missing, ref missing, ref missing, ref missing);
```

- Compiler inserts `ref` constructs

```
Microsoft.Office.Interop.Word.Document aDoc =
WordApp.Documents.Open(FileName: "NewTest.doc",
                       ReadOnly: false, Visible: true);
```

COM Interop assemblies

- COM Interop Assemblies
  - are not necessary at run-time
  - are only used at compile-time
- Run-time Callable Wrappers are generated at run-time

- Somewhat easier deployment