Module 06

"Interfaces"

TEKNOLOGISK
INSTITUT

TEKNOLOGISK
INSTITUT

Agenda

- Introducing Interfaces
- Using Interfaces
- Building Comparable Objects with `IComparable`
- Building Enumerable Types with `IEnumerable`

TEKNOLOGISK
INSTITUT

What is an Interface?

- An *interface* is a reference-type containing a named set of abstract members

```
public interface IDropTarget
{
   void OnDragDrop( DragEventArgs e );
   void OnDragEnter( DragEventArgs e );
   void OnDragLeave( EventArgs e );
   void OnDragOver( DragEventArgs e );
}
```

- Interface names start with a capital `I`
- Interfaces can contain methods, properties, events declarations only
  - Cannot contain member variables, method bodies or implementation
- Interface methods are implicitly public, so access modifiers are disallowed

Defining Custom Interfaces

TEKNOLOGISK INSTITUT

- You can easily define your own interface types

```
public interface IPointy
{
   int Points{ get; }
}
```

```
static void Main()
{
   IPointy p = new IPointy();
}
```

```
public interface IPointy
{
   public int numberOfPoints;
   public IPointy()
   {
      numberOfPoints = 0;
   }
   int GetNumberOfPoints( )
   {
      return numberOfPoints;
   }
}
```

- Interfaces does not really provide any substance until they're implemented by a concrete class or struct

Contrasting Interfaces to Abstract Base Classes

- Differences
  - Interfaces cannot contain implementation
  - Abstract classes are used for partial implementation
  - Interface members are all public
  - Interfaces can derive only from other interfaces
  - Interfaces are for types unrelated by inheritance – abstract classes enforce inheritance relationship

- Identical aspects
  - Reference types
  - Cannot be instantiated
  - Not allowed to be sealed
  - Can be derived from by classes

Implementing an Interface

TEKNOLOGISK INSTITUT

- The implementing method or property must be public and have the same signature as the interface method or property being implemented

```
public class Triangle : Shape, IPointy
{
   public Triangle( ) { }
   public override void Draw()
   {
      Console.WriteLine( "Drawing {0}", PetName );
   }
   public int Points
   {
      get { return 3; }
   }
}
```

- Using Visual Studio 2012 eases interface implementation

TEKNOLOGISK
INSTITUT

Agenda

- Introducing Interfaces
- **Using Interfaces**
- Building Comparable Objects with `IComparable`
- Building Enumerable Types with `IEnumerable`

Invoking Members at the Object Level

TEKNOLOGISK INSTITUT

- Invoke methods and properties directly from the object level (*)

```
Triangle tri = new Triangle();
Console.WriteLine("Points: {0}", tri.Points );
```

- Alternatively, you could explicitly convert to the interface type to check whether type implements the interface

```
Triangle tri = new Triangle();
try
{
   IPointy pointy = (IPointy) tri;
   Console.WriteLine( pointy.Points );
}
catch( InvalidCastException e )
{
   Console.WriteLine( e.Message );
}
```

The `is` and `as` Keywords for Interfaces

TEKNOLOGISK INSTITUT

- If the object can be treated as implementing the interface, as returns a reference to such an interface

```
Triangle tri = new Triangle();
IPointy pointy = tri as IPointy;
if( pointy != null )
{
   Console.WriteLine( pointy.Points );
}
else { // Does not implement Ipointy }
```

- `is` can be used to check directly for implementation of a specific interface

```
if( tri is IPointy )
{
   Console.WriteLine( ( (IPointy) tri).Points );
}
else { // Does not implement Ipointy }
```

Interfaces as Parameters and Return Values

TEKNOLOGISK INSTITUT

- Interfaces are reference types and behave exactly like other reference types with respect to methods
- They can be passed to methods as parameters

```
static void WritePointy( IPointy pointy )
{
   Console.WriteLine( pointy.Points );
}
```

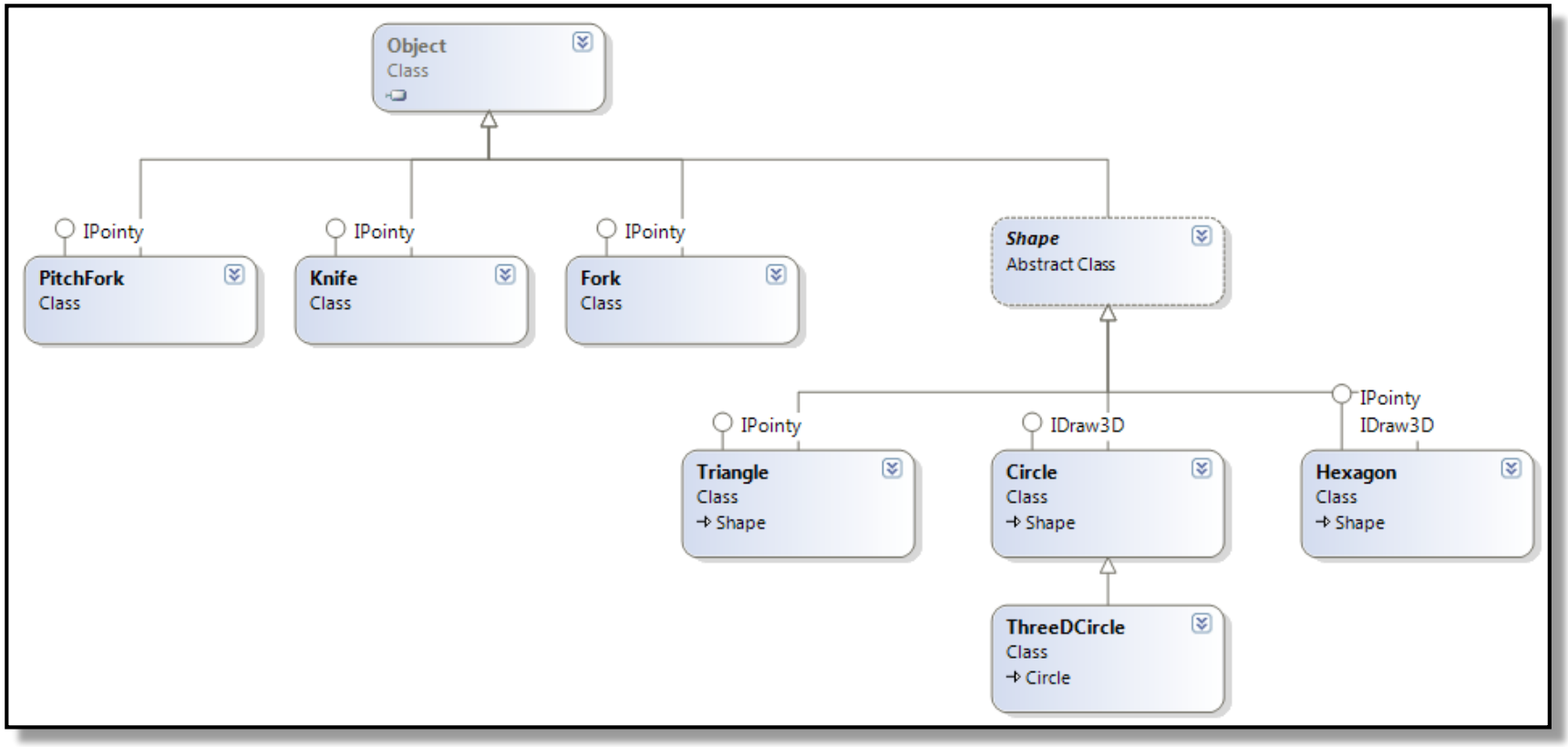
- Similarly, they can be returned from methods as return values

```
static IPointy ExtractPointyness( object o )
{
   return o as IPointy;
}
```

Arrays of Interface Types

TEKNOLOGISK INSTITUT

- Even if the interface is implemented by multiple distinct types, you can iterate through an array of interfaces and treat each item identically



Multiple Inheritance with Interface Types

TEKNOLOGISK INSTITUT

- A class can implement an arbitrary number of interfaces
  - But only have one superclass!

```
interface IDrawable
{
   void Draw();
}
```

```
interface IPrintable
{
   void Print();
   void Draw();
}
```

```
interface IRenderToMemory
{
   void Render();
}
```

```
class SuperShape : IDrawable, IPrintable, IRenderToMemory
{
   public void Draw() { ... }
   public void Print() { ... }
   public void Render() { ... }
}
```

- Potential name clash!

TEKNOLOGISK
INSTITUT

Name Clashes

- But what in the methods are not at all the same?

```
interface IArtist
{
   void Draw();
}
```

```
interface IGunslinger
{
   void Draw();
}
```

```
class ArtisticCowboy : IArtist, IGunslinger
{
   public void Draw(); // ???
}
```

- How do we signal which method we refer to?

Explicit Interface Implementation

TEKNOLOGISK INSTITUT

- Interfaces can be implemented *explicitly* to resolve name clashes

```
interface IArtist
{
   void Draw();
}
```

```
interface IGunslinger
{
   void Draw();
}
```

```
class ArtisticCowboy : IArtist, IGunslinger
{
   void IArtist.Draw() { ... }
   void IGunslinger.Draw() { ... }
}
```

- Can only be accessed through the corresponding interface
- No access modifier on method
- Cannot be virtual or overridden!

```
ArtisticCowboy ac = new ArtisticCowboy();
ac.Draw();
```

Designing Interface Hierarchies

TEKNOLOGISK INSTITUT

- An interface can extend an arbitrary number of interfaces
- Arrange your related interfaces into interface hierarchies!
- This has been done extensively through the .NET Framework classes
  - E.g. **`IList`**, **`ICollection`**, …

```
public interface IList : ICollection, IEnumerable
{
    ...
}
```

- An interface cannot be more accessible than it's base interface!

Quiz: Designing Interfaces – Right or Wrong?

TEKNOLOGISK INSTITUT

```
interface IDrawable
{
   void Draw();
}
```

```
class WyattEarp : IDrawable
{
   void Draw() { ... }
}
```

```
class Circle : IDrawable
{
   public void Draw() { Console.WriteLine("Drawing..."); }
}
```

```
class Artist : IDrawable
{
   public void Draw( Canvas c ) { ... }
}
```

TEKNOLOGISK
INSTITUT

Agenda

- Introducing Interfaces
- Using Interfaces
- **Building Comparable Objects with `IComparable`**
- Building Enumerable Types with `IEnumerable`

The `IComparable` Interface

TEKNOLOGISK INSTITUT

- Implement **`IComparable`** to compare objects to each other

```
public interface IComparable
{
   int CompareTo( object obj );
}
```

`CompareTo()` Return Value	Indicating…
< 0	This instance is before **`obj`**
0	This instance is equal to **`obj`**
> 0	This instance is after **`obj`**

- Built into .NET

Implementing IComparable

TEKNOLOGISK INSTITUT

- You can implement **IComparable** in your own types

```
public class Car : IComparable
{  ...
   public int ID { get; set; }

   public int CompareTo( object obj
   {
      Car other = obj as Car;
      if( this.carID < other.carID ) { return -1; }
      else if( this.carID > other.carID ) { return 1; }
      return 0;
   }
}
```

```
Car c1 = ...;
Car c2 = ...;
if( c1.CompareTo( c2 ) < 0 )
{
   // c1 is less than c2
}
```

- **IComparable** types can be sorted e.g. in arrays

The IComparer Interface

TEKNOLOGISK INSTITUT

- Multiple sort orders can be obtained using the generic **IComparer**

```
interface IComparer
{
   int Compare( object o1, object o2 );
}
```

- In **System.Collections** namespace

```
public class PetNameComparer : IComparer
{
   int IComparer.Compare( object o1, object o2 )
   {
      Car c1 = o1 as Car;
      Car c2 = o2 as Car;
      return String.Compare( c1.PetName, c2.PetName );
   }
}
```

```
Array.Sort( cars, new PetNameComparer() );
```

TEKNOLOGISK
INSTITUT

Agenda

- Introducing Interfaces
- Using Interfaces
- Building Comparable Objects with `IComparable`
- **Building Enumerable Types with `IEnumerable`**

The `IEnumerable` Interface

TEKNOLOGISK INSTITUT

- The **`IEnumerable`** interface states that the items of a class can be enumerated

```
using System.Collections;

public interface IEnumerable
{
   IEnumerator GetEnumerator();
}
```

```
public interface IEnumerator
{
   bool MoveNext ();
   object Current { get; }
   void Reset ();
}
```

- The **`IEnumerator`** interface provides an enumerator mechanism for the class
- Both are built into the .NET Framework base classes in the **`System.Collections`** namespace
- Arrays and collection types implement **`IEnumerable`** out-of-the-box

Implementing `IEnumerable`

TEKNOLOGISK INSTITUT

- You can implement **`IEnumerable`** in your own types

```
Garage garage = new Garage();
foreach( Car c in garage )
{
   Console.WriteLine( c.PetName );
}
```

```
public class Garage : IEnumerable
{
   private Car[] carArray = new Car[ 4 ];
   public Garage()
   {
      carArray[ 0 ] = new Car( "FeeFee", 200 );
      carArray[ 1 ] = new Car( "Clunker", 90 );
      carArray[ 2 ] = new Car( "Zippy", 30 );
      carArray[ 3 ] = new Car( "Fred", 30 );
   }

   public IEnumerator GetEnumerator() { ... }
}
```

Building Iterators with `yield`

TEKNOLOGISK INSTITUT

- C# provides powerful mechanisms for creating *iterator methods*

```
public IEnumerator GetEnumerator()
{
   foreach( Car c in carArray )
   {
      yield
   }
}
```

```
public IEnumerator GetEnumerator()
{
   yield return carArray[ 0 ];
   yield retur
   yield retu
   yield retu
}
```

```
public IEnumerator GetEnumerator()
{
   int i = 0;
   while( true )
   {
      yield return carArray[ i++ ];
      if( i == 4 ) { yield break; }
   }
}
```

Named Iterators

TEKNOLOGISK INSTITUT

- Multiple iterators can be built for a class with named iterators

```
public IEnumerable GetTheCars( bool returnReversed )
{
   if( returnReversed )
   {
      for( int i = carArray.Length; i != 0; i-- )
      { yield return carArray[i-1]; }
   }
   else
   {
      foreach( Car c in carArray ) { yield return c; }
   }
}
```

```
Garage garage = new Garage();
foreach( Car c in garage.GetTheCars( true ) )
{
   Console.WriteLine( c.PetName );
}
```

TEKNOLOGISK
INSTITUT

Summary

- Introducing Interfaces
- Using Interfaces
- Building Comparable Objects with `IComparable`
- Building Enumerable Types with `IEnumerable`

Question

TEKNOLOGISK INSTITUT

You are developing a program containing the following code segments.

```
interface INumberStorage
{
    int Number { get; set; }
}
```

You need to ensure that the ExtractNumber() method does not throw an exception if o is not of type INumberStorage.

```
int? ExtractNumber( object o )
{
    [                                        ]
    if( numberStorage != null )
    {
        return numberStorage.Number;
    }
    return null;
}
```

Which code segment should be inserted into the box?

a) `throw new InvalidCastException();`
b) `var numberStorage = o as INumberStorage;`
c) `var numberStorage = (INumberStorage) o;`
d) `var numberStorage = o is INumberStorage;`

TEKNOLOGISK
INSTITUT