CHAPTER 12

Redefining Members

A member in a derived class can redefine a member in its base class. This can be done for all kinds of inherited members, but it is most often used to give instance methods new implementations. To give a method a new implementation, the method is redefined in the child class with the same signature as it has in the base class. The signature includes the name, parameters, and return type of the method.

```
class Rectangle
{
  public int x = 1, y = 10;
  public int GetArea() { return x * y; }
}

class Square : Rectangle
{
  public int GetArea() { return 2 * x; }
}
```

Hiding Members

It must be specified whether the method is intended to *hide* or *override* the inherited method. By default, the new method will hide it, but the compiler will give a warning that the behavior should be explicitly specified.

To remove the warning, the new modifier needs to be used. This specifies that the intention was to hide the inherited method and to replace it with a new implementation.

```
class Square : Rectangle
{
  public new int GetArea() { return 2 * x; }
}
```

Overriding Members

Before a method can be overridden, the virtual modifier must first be added to the method in the base class. This modifier allows the method to be overridden in a derived class.

```
class Rectangle
{
  public int x = 1, y = 10;
  public virtual int GetArea() { return x * y; }
}
```

The override modifier can then be used to change the implementation of the inherited method.

```
class Square : Rectangle
{
  public override int GetArea() { return 2 * x; }
}
```

Hiding and Overriding

The difference between `override` and `new` is shown when a `Square` is upcast to a `Rectangle`. If the method is redefined with the `new` modifier, then this allows access to the previously hidden method defined in `Rectangle`. On the other hand, if the method is redefined using the `override` modifier, then the upcast will still call the version defined in `Square`. In short, the `new` modifier redefines the method down the class hierarchy, while `override` redefines the method both up and down in the hierarchy.

Sealed Keyword

To stop an overridden method from being further overridden in classes that inherit from the derived class, the method can be declared as `sealed` to negate the `virtual` modifier.

```
class MyClass
{
  public sealed override int NonOverridable() {}
}
```

A class can also be declared as `sealed` to prevent any class from inheriting it.

```
sealed class NonInheritable {}
```

Base Keyword

There is a way to access a parent's method even if it has been redefined. This is done by using the base keyword to reference the base class instance. Whether the method is hidden or overridden, it can still be reached by using this keyword.

```
class Triangle : Rectangle
{
  public override GetArea() { return base.GetArea()/2; }
}
```

The base keyword can also be used to call a base class constructor from a derived class constructor. The keyword is then used as a method call before the constructor's body, prefixed by a colon.

```
class Rectangle
{
  public int x = 1, y = 10;
  public Rectangle(int a, int b) { x = a; y = b; }
}
class Square : Rectangle
{
  public Square(int a) : base(a,a) {}
}
```

When a derived class constructor does not have an explicit call to the base class constructor, the compiler will automatically insert a call to the parameterless base class constructor in order to ensure that the base class is properly constructed.

```
class Square : Rectangle
{
  public Square(int a) {} // : base() implicitly added
}
```

Note that if the base class has a constructor defined that is not parameterless, the compiler will not create a default parameterless constructor. Therefore, defining a constructor in the derived class, without an explicit call to a defined base class constructor, will cause a compile-time error.

```
class Base { public Base(int a) {} }
class Derived : Base {} // compile-time error
```