Dynamic Polymorphism

C# allows you to create abstract classes that are used to provide partial class implementation of an interface. Implementation is completed when a derived class inherits from it. **Abstract** classes contain abstract methods, which are implemented by the derived class. The derived classes have more specialized functionality.

Here are the rules about abstract classes:

* You cannot create an instance of an abstract class
* You cannot declare an abstract method outside an abstract class
* When a class is declared **sealed**, it cannot be inherited, abstract classes cannot be declared sealed.

The following program demonstrates an abstract class:

using System;

namespace PolymorphismApplication

{

abstract class Shape

{

public abstract int area();

}

class Rectangle: Shape

{

private int length;

private int width;

public Rectangle( int a=0, int b=0)

{

length = a;

width = b;

}

public override int area ()

{

Console.WriteLine("Rectangle class area :");

return (width \* length);

}

}

class RectangleTester

{

static void Main(string[] args)

{

Rectangle r = new Rectangle(10, 7);

double a = r.area();

Console.WriteLine("Area: {0}",a);

Console.ReadKey();

}

}

}

When the above code is compiled and executed, it produces the following result:

Rectangle class area :

Area: 70

When you have a function defined in a class that you want to be implemented in an inherited class(es), you use **virtual** functions. The virtual functions could be implemented differently in different inherited class and the call to these functions will be decided at runtime.

Dynamic polymorphism is implemented by **abstract classes** and **virtual functions**.

The following program demonstrates this:

using System;

namespace PolymorphismApplication

{

class Shape

{

protected int width, height;

public Shape( int a=0, int b=0)

{

width = a;

height = b;

}

public virtual int area()

{

Console.WriteLine("Parent class area :");

return 0;

}

}

class Rectangle: Shape

{

public Rectangle( int a=0, int b=0): base(a, b)

{

}

public override int area ()

{

Console.WriteLine("Rectangle class area :");

return (width \* height);

}

}

class Triangle: Shape

{

public Triangle(int a = 0, int b = 0): base(a, b)

{

}

public override int area()

{

Console.WriteLine("Triangle class area :");

return (width \* height / 2);

}

}

class Caller

{

public void CallArea(Shape sh)

{

int a;

a = sh.area();

Console.WriteLine("Area: {0}", a);

}

}

class Tester

{

static void Main(string[] args)

{

Caller c = new Caller();

Rectangle r = new Rectangle(10, 7);

Triangle t = new Triangle(10, 5);

c.CallArea(r);

c.CallArea(t);

Console.ReadKey();

}

}

}

When the above code is compiled and executed, it produces the following result:

Rectangle class area:

Area: 70

Triangle class area:

Area: 25