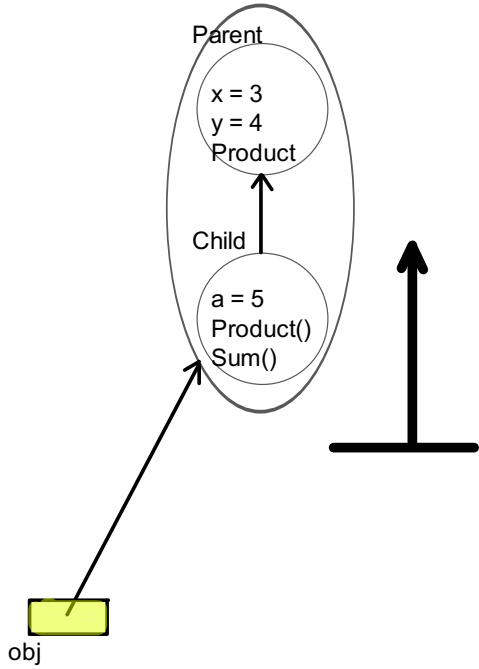
```
class Parent
{           protected int x, y;
            public Parent()
            {x = y = 0;}
            public Parent(int m)
            {x = y = m;}          3       4
            public Parent(int m, int n)
            {x = m;     y = n;}
            public int GetX()
            { return x;}
            public int GetY()
            {return y;}
            public void SetX(int m)
            {x = m;}
            public void SetY(int n)
            {y = n;}
            public virtual int Product()
            {return (x * y);}
}

class Child : Parent
{           int a;
            public Child()
            {a = 0;}            3       4       5             3  4
            public Child(int l, int m, int n) : base(l, m)
            { a = n;}
            public int GetA()
            {return a;}
            public void SetA(int m)
            {a = m;}
            public override int Product()
            {return (x * y * a);}
            public int Sum()
            {return (x + y + a);}
}
```

Method Overriding:
More than one method having the same name and same parameter (same signature) implemented in different classes having an inheritance relation

```
class Test
{
    public static void Main()
    {
            Child obj;
            obj = new Child(3, 4, 5);
            int val = obj.Product();          //3 * 4 * 5
    }
}
```

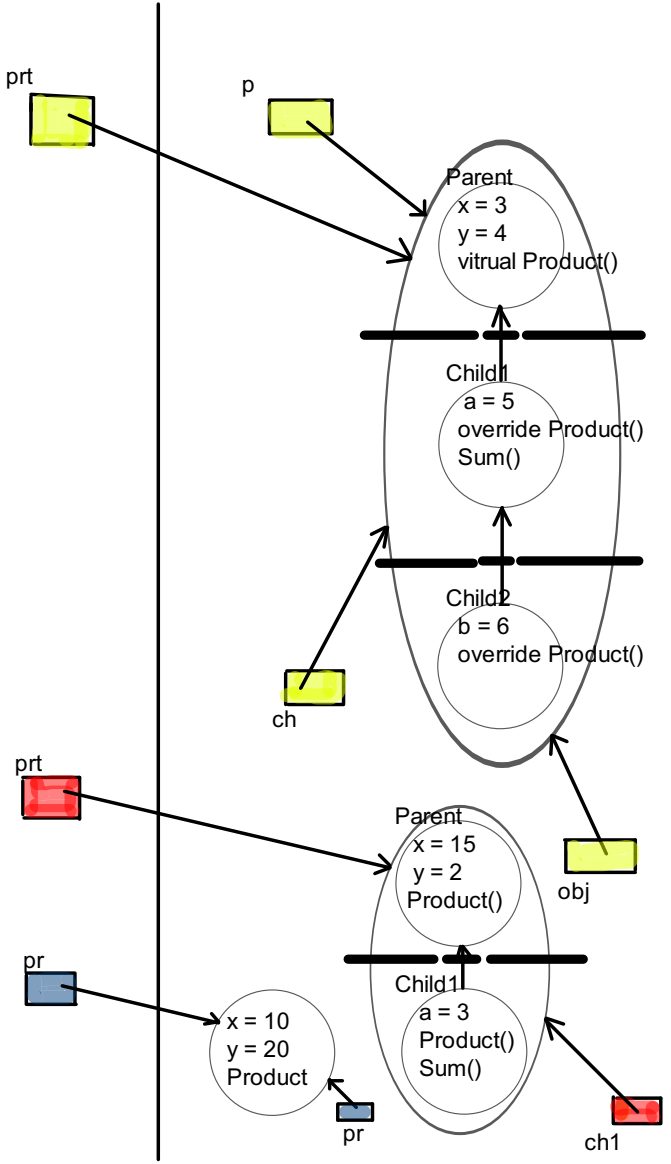


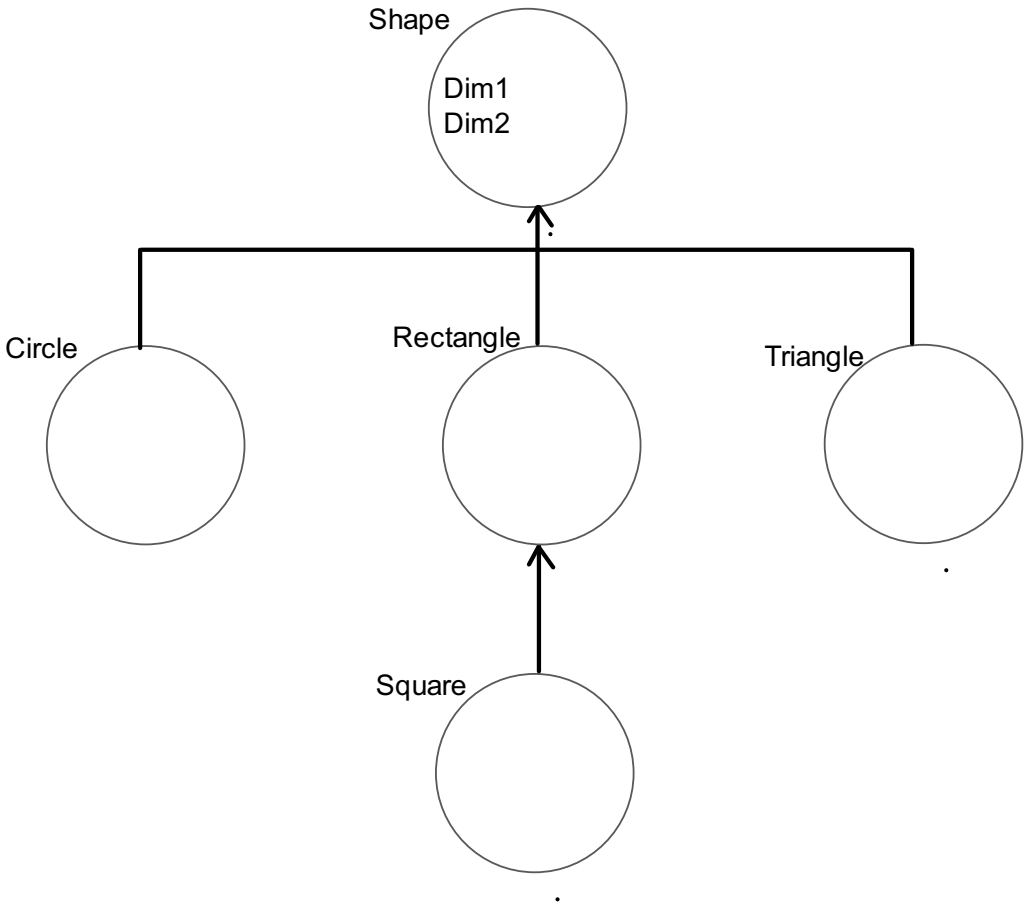
```
class Parent
{           protected int x, y;
            public Parent()
            {x = y = 0;}
            public Parent(int m)
            {x = y = m;}          3      4
            public Parent(int m, int n)
            {x = m;    y = n;}
            public int GetX() { return x;}
            public int GetY() {return y;}
            public void SetX(int m) {x = m;}
            public void SetY(int n)   {y = n;}
            public virtual int Product()
            {return (x * y);}
}
class Child1 : Parent
{           int a;
            public Child1()
            {a = 0;}              3     4      5           3  4
            public Child1(int l, int m, int n) : base(l, m)
            { a = n;}
            public int GetA() {return a;}
            public void SetA(int m) {a = m;}
            public override int Product()
            {return (x * y * a);}
            public int Sum()
            {return (x + y + a);}
}
class Child2 : Child1
{  int b;
   public Child2()
   {b = 0;}             3      4      5      6           3 4   5
   public Child2(int l, int m, int n, int p) : base(l, m, n)
   { b = p;}
   public int GetB() {return b;}
   public void SetB(int m) {b = m;}
   public override int Product()
   {return (x * y * a * b);}
}
```

```
class Test
{
    public static void Main()
    {
            Child2 Obj;
            Obj = new Child2(3, 4, 5, 6);
            int val = Obj.Product();          //x * y * a * b
            int m = Obj.Sum();                //x + y + a

            Child1 ch;
            ch = Obj;
            int n = ch.Sum();                 //x + y + a
            int l = ch.Product();             //x * y * a * b

            Parent p;
            p = Obj;
            int z = p.Product();              //x * y * a * b
            Print(Obj);             //3 * 4 * 5 * 6

            Child1 ch1 = new Child1(15, 2, 3);
            Print(ch1);             //15 * 2 * 3

            Parent pr = new Parent(10, 20);
            Print(pr);              //10 * 20
    }

    public static void Print(Parent prt)
    {
            Console.WriteLine(prt.Product());
    }
}
```

Reference to the base class:
we can assign reference of the parent to an object from the child class

*) Dynamic Binding (late Binding)
*) Open Ended Hierarchy

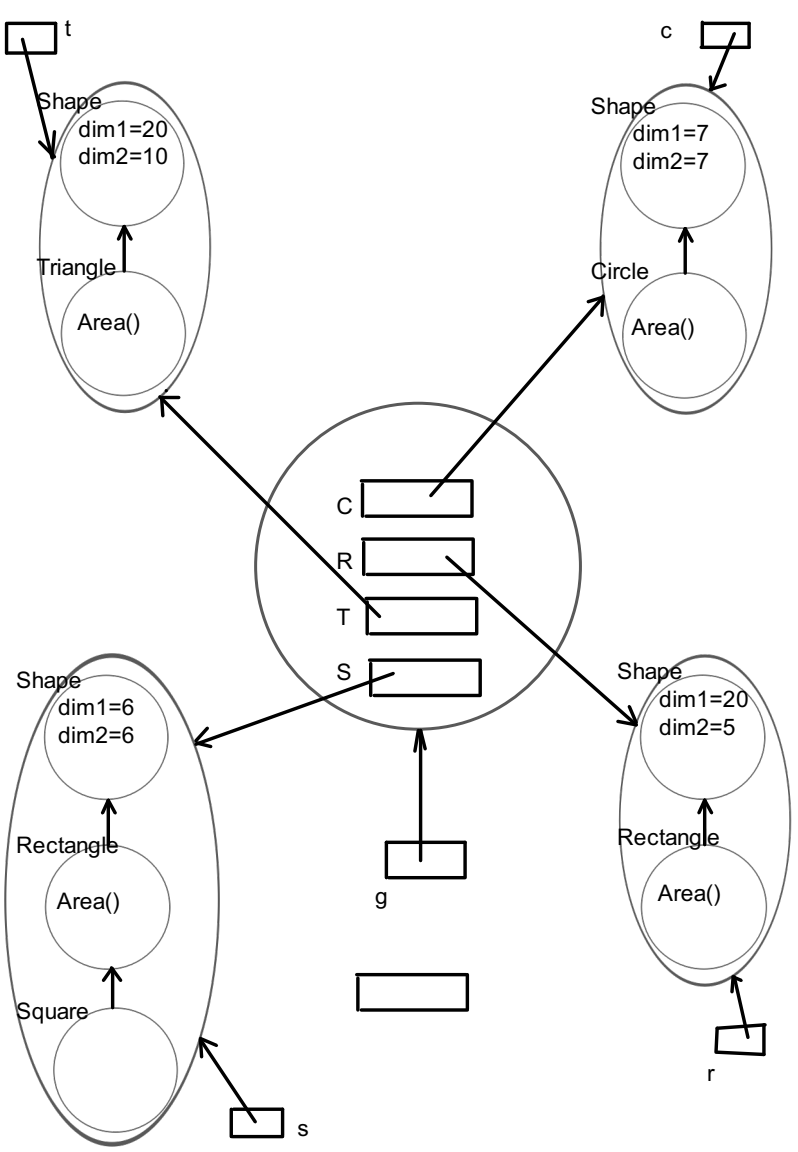



Shape
Dim1
Dim2
Circle
Rectangle
Triangle
Square

```
class Shape
{  protected int dim1, dim2;
   public  Shape(){dim1=dim2=0;}
   public Shape(int m){dim1=dim2=m;}
   public Shape(int m, int n)
   {dim1 = m;       dim2 = n;}
   public void SetD1(int m){dim1 = m;}
   public void SetD2(int n){dim2 = n;}
   public int GetD1(){return dim1;}
   public int GetD2(){return dim2;}

}
class Circle : Shape
{
    public Circle(){}
    public Circle(int r):base(r){}
    public float Area()
    {return (3.14 * dim1 * dim2);}
}
class Rectangle : Shape
{
    public Rectangle(){}
    public Rectangle(int l, int w):base(l,w){}
    public float Area()
    {return (1.0 * dim1 * dim2);}
}
class Triangle : Shape
{
    public Triangle(){}
    public Triangle(int w, int h):base(w,h){}
    public float Area()
    {return (0.5 * dim1 * dim2);}
}
class Square : Rectangle
{
    public Square(){}
    public Square(int s):base(s,s){}
}
```

```
class GeoShape
{
    Circle C;
    Rectangle R;
    Triangle T;
    Square S;
    public GeoShape(Circle c1, Rectangle r1,
                    Triangle t1, Square s1)
        {
            C = c1;
            R = r1;
            T = t1;
            S = s1;
        }
    public float TotalArea()
        {
            float total;
            total = C.Area() + R.Area() +
                    T.Area() + S.Area();
            return total;
        }
}
```

```
class Test
{
public static void main()
{
        Circle c = new Circle(7);
        Rectangle r = new Rectangle(20, 5);
        Triangle t = new Triangle(20, 10);
        Square s = new Square(6);
        GeoShape g = new GeoShape(c, r, t, s);

        Console.WriteLine(g.TotalArea(;)
}
}
```



```
abstract class Shape
{  protected int dim1, dim2;
   public  Shape(){dim1=dim2=0;}
   public Shape(int m){dim1=dim2=m;}
   public Shape(int m, int n)
   {dim1 = m;        dim2 = n;}
   public void SetD1(int m){dim1 = m;}
   public void SetD2(int n){dim2 = n;}
   public int GetD1(){return dim1;}
   public int GetD2(){return dim2;}
   public abstract float Area();
}
class Circle : Shape
{
    public Circle(){}
    public Circle(int r):base(r){}
    public override float Area()
    {return (3.14 * dim1 * dim2);}
}
class Rectangle : Shape
{
    public Rectangle(){}
    public Rectangle(int l, int w):base(l,w){}
    public override float Area()
    {return (1.0 * dim1 * dim2);}
}
class Triangle : Shape
{
    public Triangle(){}
    public Triangle(int w, int h):base(w,h){}
    public override float Area()
    {return (0.5 * dim1 * dim2);}
}
class Square : Rectangle
{
    public Square(){}
    public Square(int s):base(s,s){}
}
```

```
class GeoShape
{
    Shape C;
    Shape R;
    Shape T;
    Shape S;
    public GeoShape(Shape c1, Shape r1,
                    Shape t1, Shape s1)
        {
            C = c1;
            R = r1;
            T = t1;
            S = s1;
        }
    public float TotalArea()
        {
            float total;
            total = C.Area() + R.Area() +
                    T.Area() + S.Area();
            return total;
        }
}
```

```
class Test
{
public static void main()
{
        Circle c = new Circle(7);
        Rectangle r = new Rectangle(20, 5);
        Triangle t = new Triangle(20, 10);
        Square s = new Square(6);
        GeoShape g = new GeoShape(c, r, t, s);

        Console.WriteLine(g.TotalArea();)
}
}
```