الجامعة المستنصرية
كليـــــــــــة العـــــــلوم
قسم علوم الحاسبات

البرمجة الكيانية / المرحلة الثانية / حسن قاسم محمد

Object Oriented Programming

Polymorphism

Polymorphism is a Greek word, meaning "one name many forms". In other words, one object has many forms or has one name with multiple functionalities. "Poly" means many and "morph" means forms. Polymorphism provides the ability to a class to have multiple implementations with the same name.

This allows us to perform a single action in different ways.

Polymorphism is the ability of an entity to take several forms. It allows a common data-gathering message to be sent to each class. Polymorphism encourages called as ‘extendibility’ which means an object or a class can have its uses extended.

ذهب

عين

Bank

Can

حية

حب

MAN is only one, but he takes multiple roles like

DAD
EMPLOYEE
SALESPERSON

and many more.

you have a SMARTPHONE

for communication.

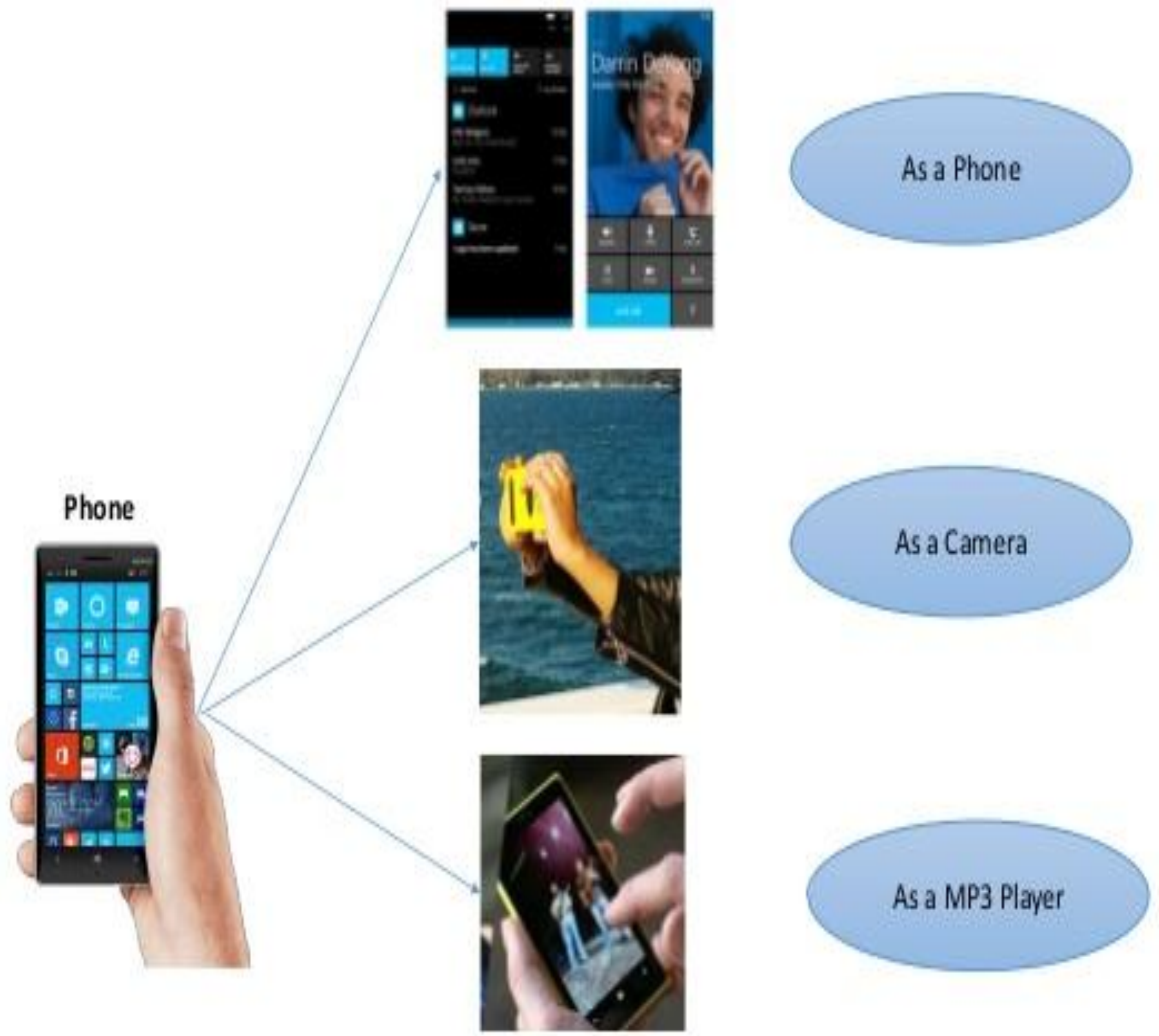
The communication could be :

CALL

TEXT MESSAGE

PICTURE MESSAGE

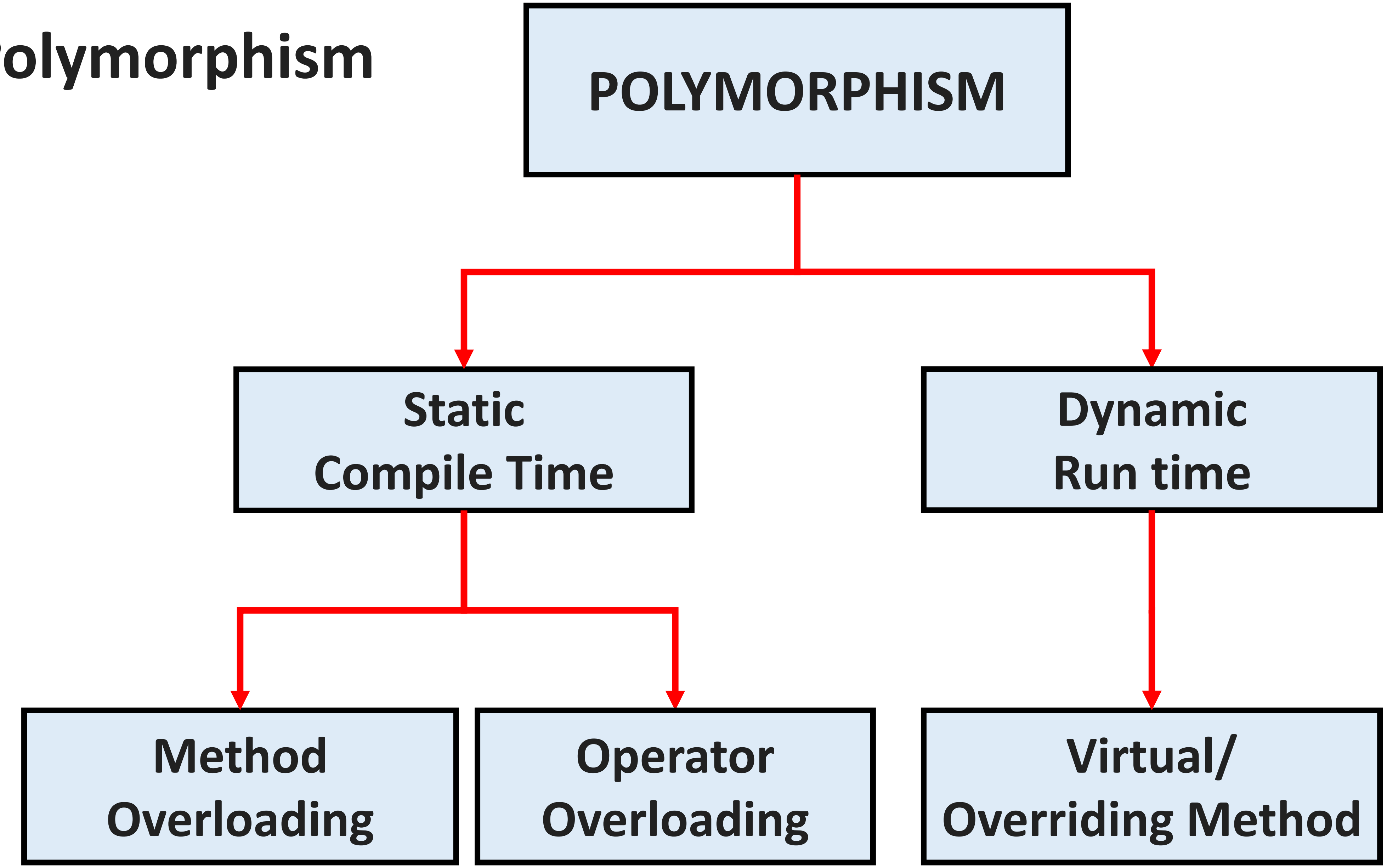
MAIL



So, the goal is communication, but their approach is different.

We can decide the correct call at runtime

Types of Polymorphism



1 Static or Compile Time Polymorphism

It is also known as Early Binding.

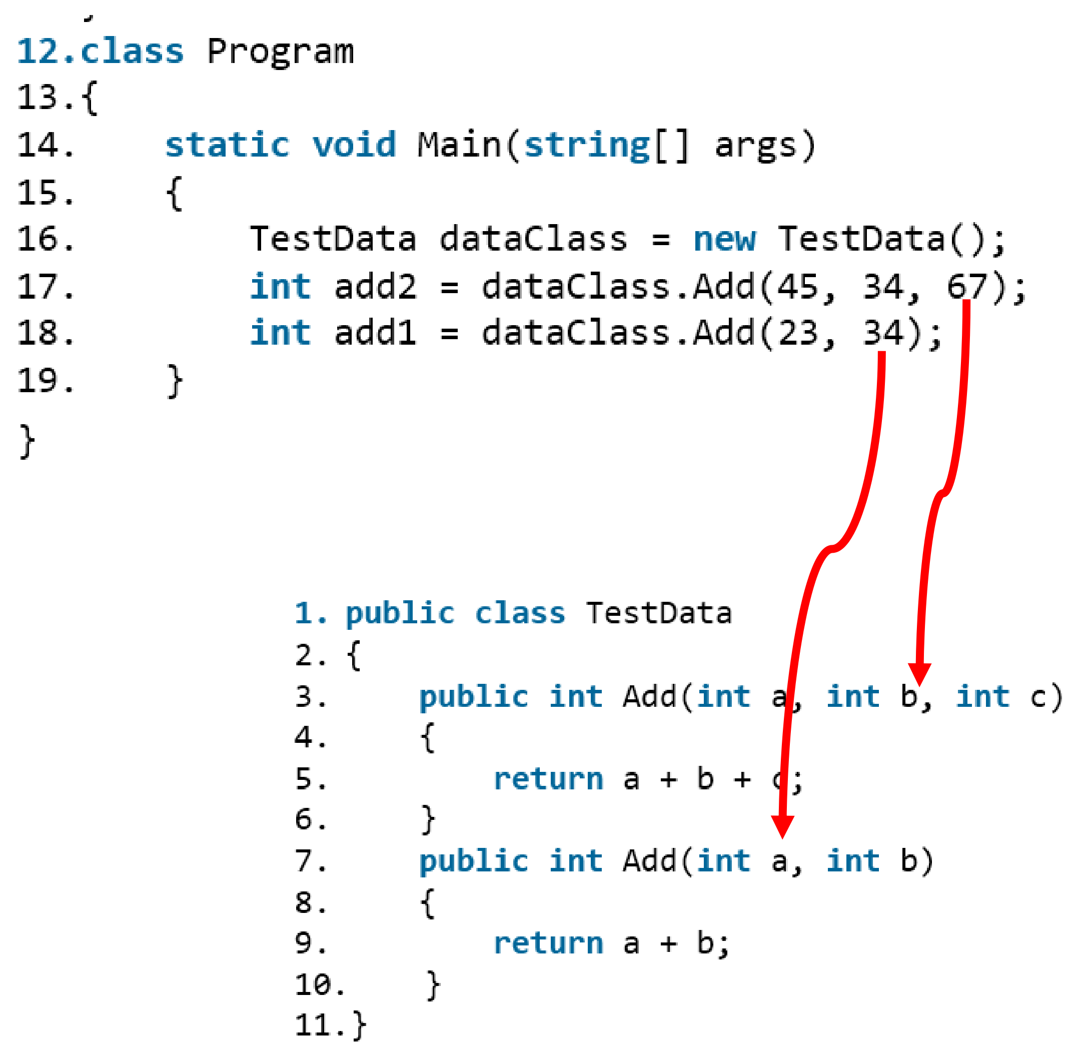
Method overloading is an example of Static Polymorphism.

In overloading, the method / function has a same name but different signatures(set of parameters). It is also known as Compile Time Polymorphism because the decision of which method is to be called is made at compile time.

1 Static or Compile Time Polymorphism Example

```
public class TestData
{
    public int Add(int a, int b, int c)
    {
        return a + b + c;
    }
    public int Add(int a, int b)
    {
        return a + b;
    }
}
class Program
{
    static void Main(string[] args)
    {
        TestData dataClass = new TestData();
        int add2 = dataClass.Add(45, 34, 67);
        int add1 = dataClass.Add(23, 34);
    }
}
```

same name but different signatures(set of parameters)



```
class Program
{
    static void Main(string[] args)
    {
        TestData dataClass = new TestData();
        int add2 = dataClass.Add(45, 34, 67);
        int add1 = dataClass.Add(23, 34);
    }
}

public class TestData
{
    public int Add(int a, int b, int c)
    {
        return a + b + c;
    }
    public int Add(int a, int b)
    {
        return a + b;
    }
}
```

2 Dynamic / Runtime Polymorphism

Dynamic / runtime polymorphism is also known as late binding. Here, the method name and the method signature (number of parameters and parameter type must be the same and may have a different implementation). **Method overriding** is an example of dynamic polymorphism.

Method overriding can be done using inheritance. With method overriding it is possible for the base class and derived class to have the same method name and same something. The compiler would not be aware of the method available for overriding the functionality, so the compiler does not throw an error at compile time. The compiler will decide which method to call at runtime and if no method is found then it throws an error.

2 Dynamic / Runtime Polymorphism Example

```
public class Drawing
{
    public virtual double Area()
   {
         return 0;
   }
}
```

```
public class Circle : Drawing
{
    public double Radius { get; set; }
    public Circle()
    {
        Radius = 5;
    }
    public override double Area()
    {
        return (3.14) * Math.Pow(Radius, 2);
    }
}
```

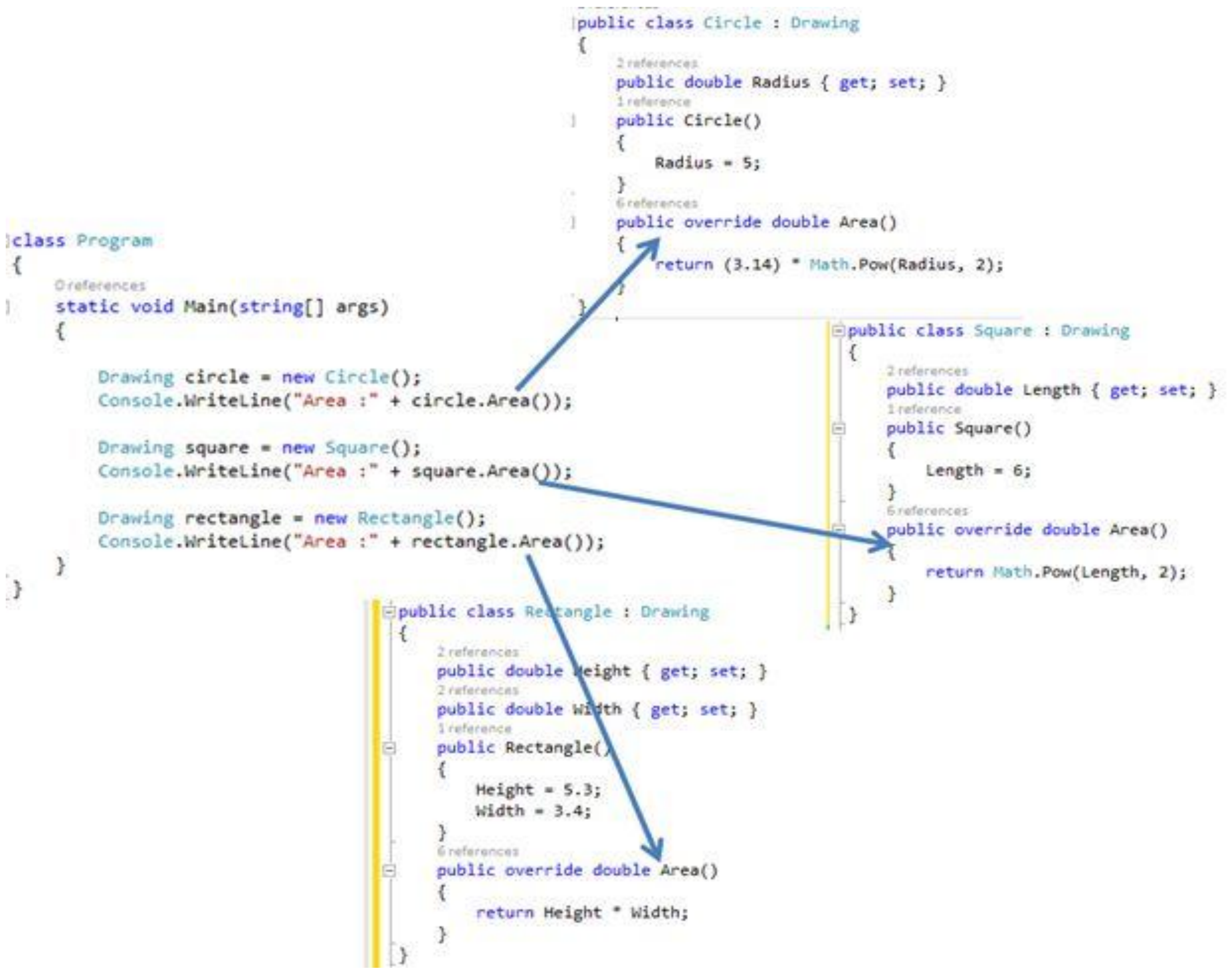
```
public class Square : Drawing
{
    public double Length { get; set; }
    public Square()
    {
        Length = 6;
    }
    public override double Area()
    {
        return Math.Pow(Length, 2);
    }
}
```

```
public class Rectangle : Drawing
{
    public double Height { get; set; }
    public double Width { get; set; }
    public Rectangle()
    {
        Height = 5.3;
        Width = 3.4;
    }
    public override double Area()
    {
        return Height * Width;
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {

        Drawing circle = new Circle();
        Console.WriteLine("Area :" + circle.Area());

        Drawing square = new Square();
        Console.WriteLine("Area :" + square.Area());

        Drawing rectangle = new Rectangle();
        Console.WriteLine("Area :" + rectangle.Area());

    }
}
```

The compiler requires an Area() method and it compiles successfully but the right version of the Area() method is not being determined at compile time but determined at runtime. Finally the overriding methods must have the same name and signature (number of parameters and type), as the virtual or abstract method defined in the base class method and that it is overriding in the derived class.

A method or function of the base class is available to the child (derived) class without the use of the "overriding" keyword. The compiler hides the function or method of the base class. This concept is known as shadowing or method hiding.



```
class Program
{
    0 references
    static void Main(string[] args)
    {

        Drawing circle = new Circle();
        Console.WriteLine("Area :" + circle.Area());

        Drawing square = new Square();
        Console.WriteLine("Area :" + square.Area());

        Drawing rectangle = new Rectangle();
        Console.WriteLine("Area :" + rectangle.Area());
    }
}
```

```
public class Circle : Drawing
{
    2 references
    public double Radius { get; set; }
    1 reference
    public Circle()
    {
        Radius = 5;
    }
    6 references
    public override double Area()
    {
        return (3.14) * Math.Pow(Radius, 2);
    }
}
```

```
public class Square : Drawing
{
    2 references
    public double Length { get; set; }
    1 reference
    public Square()
    {
        Length = 6;
    }
    6 references
    public override double Area()
    {
        return Math.Pow(Length, 2);
    }
}
```

```
public class Rectangle : Drawing
{
    2 references
    public double Height { get; set; }
    2 references
    public double Width { get; set; }
    1 reference
    public Rectangle()
    {
        Height = 5.3;
        Width = 3.4;
    }
    6 references
    public override double Area()
    {
        return Height * Width;
    }
}
```

Overriding

Method overriding is an important feature of OOP that allows us to re-write a base class function or method with a different definition. Overriding is also known as "Dynamic polymorphism" because overriding is resolved at runtime. Here the signature of the method or function must be the same. In other words both methods (base class method and child class method) have the same name, same number and same type of parameter in the same order with the same return type. The overridden base method must be virtual, abstract or override.

Overloading vs. Overriding

- Overloading deals with multiple methods in the same class with the same name but different signatures
- Overloading lets you define a similar operation in different ways for different data

- Overriding deals with two methods, one in a parent class and one in a child class, that have the same signature
- Overriding lets you define a similar operation in different ways for different object types

A method cannot be overridden if:

- Methods have a different return type
- Methods have a different access modifier
- Methods have a different parameter type or order
- Methods are non virtual or static

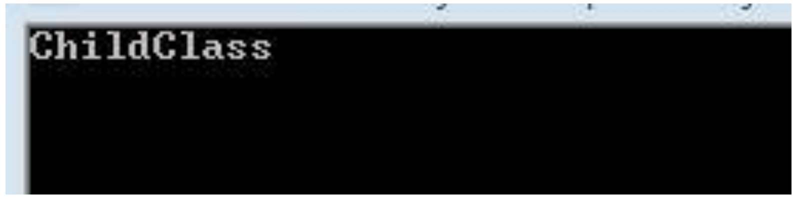
ANY DIFFERENCE

Shadowing (method hiding)

A method or function of the base class is available to the child (derived) class without the use of the "overriding" keyword. The compiler hides the function or method of the base class. This concept is known as shadowing or method hiding. In the shadowing or method hiding, the child (derived) class has its own version of the function, the same function is also available in the base class.

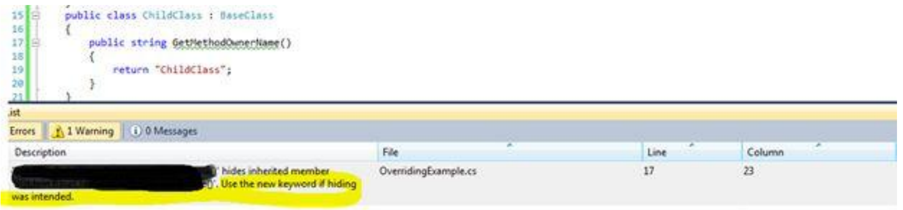
```
Public class BaseClass
{
    public string GetMethodOwnerName()
    {
        return "Base Class";
    }
}
public class ChildClass : BaseClass
{
    public new string GetMethodOwnerName()
    {
        return "ChildClass";
    }
}
```

Output



```
ChildClass
```

If we do not use the new keyword the compiler generates the warning:



```
public class ChildClass : BaseClass
{
    public string GetMethodOwnerName()
    {
        return "ChildClass";
    }
}
```

ist

Errors | 1 Warning | 0 Messages

Description	File	Line	Column
[redacted]' hides inherited member [redacted]'. Use the new keyword if hiding was intended.	OverridingExample.cs	17	23

Test Code

```
static void Main(string[] args)
{
    ChildClass c = new ChildClass();
    Console.WriteLine(c.GetMethodOwnerName());
}
```