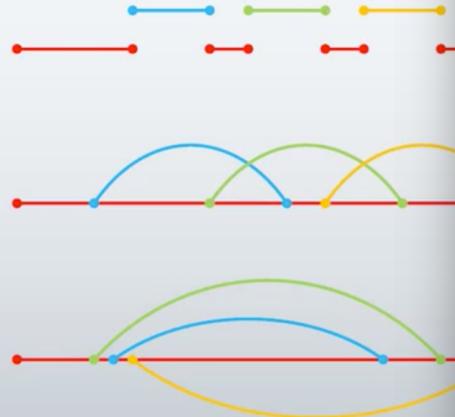


# 委托的高级使用

- 多播 ( multicast ) 委托
- 隐式异步调用
  - 同步与异步的简介
    - 中英文的语言差异
    - 同步：你做完了我（在你的基础上）接着做
    - 异步：咱们两个同时做（相当于汉语中的“同步进行”）
  - 同步调用与异步调用的对比
    - 每一个运行的程序是一个进程（process）
    - 每个进程可以有一个或者多个线程（thread）
    - 同步调用是在同一线程内
    - 异步调用的底层机理是多线程
    - 串行==同步==单线程，并行==异步==多线程
  - 隐式多线程 v.s. 显式多线程
    - 直接同步调用：使用方法名
    - 间接同步调用：使用单播/多播委托的Invoke方法
    - 隐式异步调用：使用委托的BeginInvoke
    - 显式异步调用：使用Thread或Task
- 应该适时地使用接口（interface）取代一些对委托的使用
  - Java完全地使用接口取代了委托的功能，即Java没有与C#中委托相对应的功能实体



namespace 委托指针的升级版

```
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            IProductFactort pizza = new PizzaFactory();  
            IProductFactort toycar = new ToyCarFactory();  
  
            WrapFactory wrapFactory = new WrapFactory();  
  
            Box box1= wrapFactory.WrapProuduct(pizza);  
            Box box2= wrapFactory.WrapProuduct(toycar);  
  
            Console.WriteLine(box1.Product.Name) ;  
            Console.WriteLine(box2.Product.Name) ;  
        }  
        interface IProductFactort  
        {  
            Product Make();  
        }
```

```
}

class PizzaFactory : IProductFactort
{
    public Product Make()
    {
        Product product = new Product();
        product.Name = "Pizza";
        return product;
    }
}

class ToyCarFactory : IProductFactort
{
    public Product Make()
    {
        Product product = new Product();
        product.Name = "ToyCar";
        return product;
    }
}

class Product
{
    public string Name { get; set; }
}

class Box
{
    public Product Product { get; set; }
}

class WrapFactory
{
    public Box WrapProuduct(IProductFactort productFactort)
    {
        Box box = new Box();
        Product product = productFactort.Make();
        box.Product = product;
    }
}
```

```
    return box;  
}  
  
}  
  
}
```