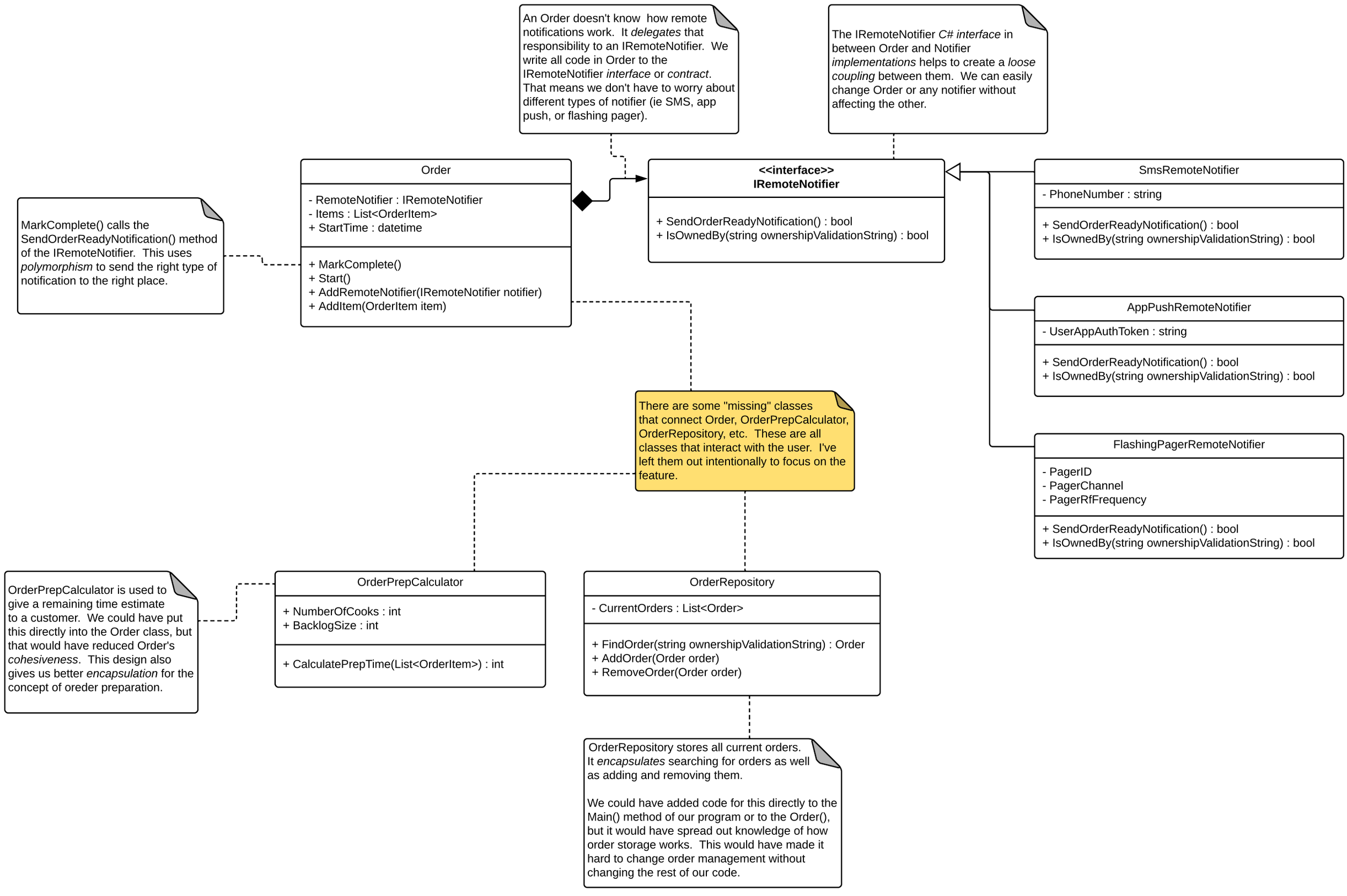
An Order doesn't know how remote notifications work. It *delegates* that responsibility to an IRemoteNotifier. We write all code in Order to the IRemoteNotifier *interface* or *contract*. That means we don't have to worry about different types of notifier (ie SMS, app push, or flashing pager).
The IRemoteNotifier *C# interface* in between Order and Notifier *implementations* helps to create a *loose coupling* between them. We can easily change Order or any notifier without affecting the other.
Order
- RemoteNotifier : IRemoteNotifier
- Items : List<OrderItem>
+ StartTime : datetime
+ MarkComplete()
+ Start()
+ AddRemoteNotifier(IRemoteNotifier notifier)
+ AddItem(OrderItem item)
MarkComplete() calls the SendOrderReadyNotification() method of the IRemoteNotifier. This uses *polymorphism* to send the right type of notification to the right place.
<<interface>>
IRemoteNotifier
+ SendOrderReadyNotification() : bool
+ IsOwnedBy(string ownershipValidationString) : bool
SmsRemoteNotifier
- PhoneNumber : string
+ SendOrderReadyNotification() : bool
+ IsOwnedBy(string ownershipValidationString) : bool
AppPushRemoteNotifier
- UserAppAuthToken : string
+ SendOrderReadyNotification() : bool
+ IsOwnedBy(string ownershipValidationString) : bool
FlashingPagerRemoteNotifier
- PagerID
- PagerChannel
- PagerRfFrequency
+ SendOrderReadyNotification() : bool
+ IsOwnedBy(string ownershipValidationString) : bool
There are some "missing" classes that connect Order, OrderPrepCalculator, OrderRepository, etc. These are all classes that interact with the user. I've left them out intentionally to focus on the feature.
OrderPrepCalculator is used to give a remaining time estimate to a customer. We could have put this directly into the Order class, but that would have reduced Order's *cohesiveness*. This design also gives us better *encapsulation* for the concept of oreder preparation.
OrderPrepCalculator
+ NumberOfCooks : int
+ BacklogSize : int
+ CalculatePrepTime(List<OrderItem>) : int
OrderRepository
- CurrentOrders : List<Order>
+ FindOrder(string ownershipValidationString) : Order
+ AddOrder(Order order)
+ RemoveOrder(Order order)
OrderRepository stores all current orders. It *encapsulates* searching for orders as well as adding and removing them.
We could have added code for this directly to the Main() method of our program or to the Order(), but it would have spread out knowledge of how order storage works. This would have made it hard to change order management without changing the rest of our code.