

---

# Single Responsibility

## 150202040 Yasin Emir Kutlu

### 1. Single Responsibility

**Single Responsibility Nedir ?** : SOLID prensipleri arasındaki 'S' prensibini temsil etmektedir. Tek sorumluluk prensibi anlamına gelmektedir. Temelde, projenizdeki bir sınıfın veya fonksiyonun sadece ve sadece tek bir sorumluluğu yerine getirmesi gerektiğini söyler.

**Single Responsibility Nasıl Kullanılır ?** : Bir sınıfta sadece ve sadece o sınıfa ait sorumluluklar bulunmalıdır. Örneğin bahcivan.cs sınıfı içerisinde BudamaYap(), SulamaYap() gibi methodlar bulunabilirken, YemekYap() gibi bir method bu sınıf içerisinde olmamalıdır. Sorumlulukların ait olduğu sınıflara koyularak her sınıfın kendi sorumluluğundaki işleri yapması sağlanır ve Single Responsibility prensibi gerçekleşmiş olur.

**Single Responsibility Neden Kullanılır ?** : Üzerinde çalıştığımız projenin genişletilebilir olması, yapılacak bir değişikliğin farklı yerleri etkilememesi, kolay okunabilir kodlar yazılması ve bileşenler arası bağımlılığın azaltılabilmesi için SOLID prensipleri kullanılır. Single Responsibility prensibinin amacı da bir sınıfın veya bir fonksiyonun sadece ve sadece tek bir sorumluluğa sahip olmasını sağlamaktır.

**Single Responsibility Kullanmanın Avantajları Nelerdir ?** : Kodun okunabilirliğini artırır. Bağımlılığı azaltır (Reduced Coupling). Kod karmaşıklığını azaltır. Yazılmış olan koda genişletilebilirlik ve yönetim kolaylığı sağlar. Tekrardan kullanılabilirlik (Reusability) kavramını gerçekler. Yani fonksiyonlar ayrıştırıldığı için kod içerisinde başka bir yerde o fonksiyona ihtiyaç duyulursa yeniden yazmadan kullanılabilir.

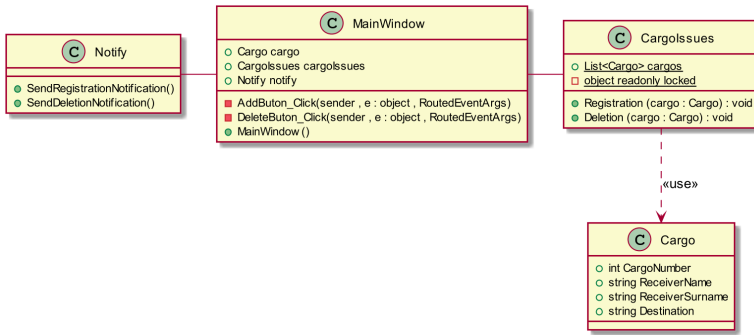


Tüm sorumlulukları  
tek yapıya yüklemek



Sorumlulukları ayırtırmak  
(Single Responsibility Principle)

## 2. Single Responsibility UML Diyagramı



## 3. Kod Açıklaması

```

//Cargo.cs
public class Cargo
{
    public int CargoNumber { get; set; }
    public string ReceiverName { get; set; }
    public string ReceiverSurname { get; set; }
    public string Destination { get; set; }
}

//CargoIssues.cs
public class CargoIssues
{
    public static List<Cargo> cargos = new List<Cargo>();
    private static readonly object locked = new object();
    public void Registration (Cargo cargo) ❶
    {
        lock(locked) cargos.Add(cargo);
    }
}
  
```

```
        public void Deletion (Cargo cargo)
        {
            lock (locked) cargos.Remove(cargo);
        }

    }

// Notify.cs
public class Notify
{
    public void SendRegistrationNotification() ❷
    {
        MessageBox.Show("kargonuz başarıyla eklendi(BİLDİRİM)",
            "İşlem tamamlandı", MessageBoxButton.OK, MessageBoxImage.Information);
    }

    public void SendDeletionNotification()
    {
        MessageBox.Show("kargonuz başarıyla sistemden
            silindi (BİLDİRİM)", "İşlem tamamlandı", MessageBoxButton.OK,
            MessageBoxImage.Information);
    }
}

//MainWindow.cs
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();

    }

    Cargo cargo = new Cargo();
    CargoIssues cargoIssues = new CargoIssues(); ❸
    Notify notify = new Notify(); ❹

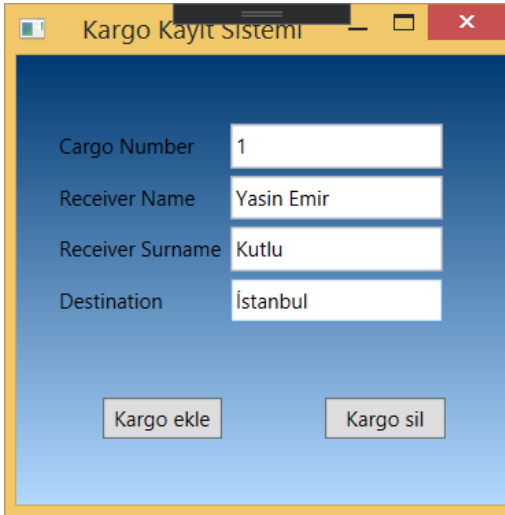
    private void AddButon_Click(object sender, RoutedEventArgs e)
    {
        cargo.CargoNumber = Int32.Parse(txtCargoNumber.Text);
        cargo.ReceiverName = txtReceiverName.Text;
        cargo.ReceiverSurname = txtReceiverSurname.Text;
        cargo.Destination = txtDestination.Text;
        cargoIssues.Registration(cargo); ❺
        notify.SendRegistrationNotification(); ❻
    }
}
```

```
}  
  
private void DeleteButon_Click(object sender, RoutedEventArgs e)  
{  
    cargo.CargoNumber = Int32.Parse(txtCargoNumber.Text);  
    cargo.ReceiverName = txtReceiverName.Text;  
    cargo.ReceiverSurname = txtReceiverSurname.Text;  
    cargo.Destination = txtDestination.Text;  
    cargoIssues.Deletion(cargo);  
    notify.SendDeletionNotification();  
}  
}
```

- ❶ Cargolssues sınıfının sorumluluğundaki görevlerin atanması
- ❷ Notify sınıfının sorumluluğundaki görevlerin belirlenmesi
- ❸ Cargolssues sınıfına ait olan sorumlulukların yerine getirilebilmesi için Cargolssues tipli nesnenin tanımlanması
- ❹ Notify sınıfına ait olan sorumlulukların yerine getirilebilmesi için Notify tipindeki nesnenin tanımlanması
- ❺ Cargolssues sınıfının görevini yapması
- ❻ Notify sınıfının görevini yapması

## 4. Ekran Çıktısı

GUI :



Kargo Kayıt Sistemi

Cargo Number 1

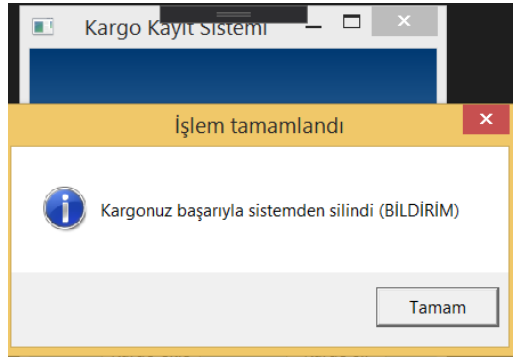
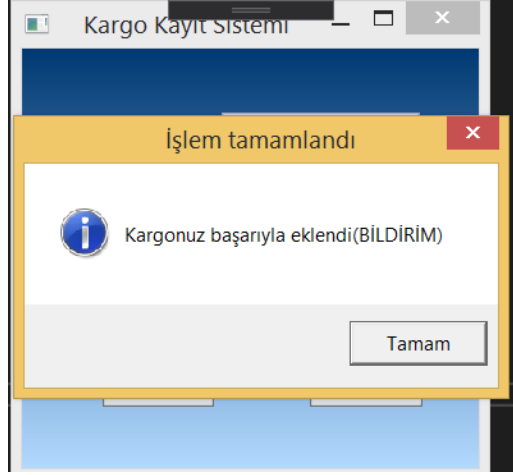
Receiver Name Yasin Emir

Receiver Surname Kutlu

Destination İstanbul

Kargo ekle Kargo sil

Bildirim ekranları :



## 5. Ekran Çıktısının Yorumlanması

Single Responsibility için örnek senaryo bir kargo şirketinin varolan sistemlerine kargo ekleme işlemini canlandırmaktadır. Single Responsibility prensibine göre dizayn edilmiş olan kodda Kargo işlemlerinin (Ekleme Silme) CargoIssues.cs sınıfında, kargo girildiğinde kargocuyu bilgilendirme methodunda Notify.cs sınıfına ayrıştırılarak Single Responsibility prensibi sağlanmıştır. Her method kendi ile alakalı sadece ve sadece bir iş yapmaktadır. Ekran çıktısında da masaüstü uygulamasındaki bildirim ve kargo işlemleri görülmektedir.

