Delivery Application

Delivery Application **1**
Sissejuhatus 1
Nõuded 1
Tervik 1
Nõuded osade kaupa 2
Sõidukid ja vedude planeerimine 2
2. Pakiautomaat 4
2.1 Lahendus kasutades Adapter disainimustrit 4
3. Tarnete loendamine 6
3.1 Lahendus kasutades Decorator mustrit 6
4. Sõidukite loomine 9
4.1 Lahendus kasutades Abstract Factory mustrit 10
5. Sõidukite grupeerimine 13
5.1 Lahendus kasutades Composite mustrit 14

Sissejuhatus

Antud ülesande eesmärk on tutvuda tarkvara disainimustritega, arendades kliendi nõuete kohaselt rakendus kulleriteenust pakkuvale ettevõttele. Kuna ülesande mõte ei ole luua võimalikult realistlikut töötavat lahendust, vaid demonstreerida disainimustirite kasulikkust, siis on rakenduses mitmete meetodite sisu asendatud konsooli printimisega. Näiteks, klassi DeliveryCar meetod Deliver() kirjutab konsooli “Car makes a delivery”, kuid reaalse elu infosüsteemis toimuksid selle asemel äriprotsesse toetavad operatsioonid.

Tarkvara disainimustrid

Tarkvara disainimustrid on üldised ja korratavad lahendused tihti esinevatele probleemidele tarkvara disaini juures. Disainimuster ei ole konkreetne lõplik lahendus, mida saab otse lähtekoodi sisestada, vaid see on mudel teatud tüüpi probleemi lahendamiseks.

Käesolevas ülesandes tutvume nelja mustriga raamatust Design Patterns: Elements of Reusable Object-Oriented Software, autoritelt Erich Gamma, Richard Helm, Ralph Johnson ja John Vlissides, keda teatakse üldiselt kui **Gang of Four**. Disainimustrite õppematerjalide koostamisel on tuginetud raamatule Head First Design Patterns. Järgnevalt on välja toodud ülesandes kasutuses olevad mustrid ning nendele vastavad leheküljed Head First Design Patterns raamatus.

- **Adapter** - lk 235-254
- **Facade** - lk 255-274
- **Abstract Factory** - lk 109-168

- **Composite** - lk 356-384

Nõuded

- Rakendusel on DeliveryApp klass koos Main meetodiga
- DeliveryApp klassis on PlanDelivery meetod, mis simuleerib kaubavedu ehk prindib konsooli, kui sõiduk on tarne teostanud.
- Vedude teostamiseks on neli erinevat sõiduki liiki - DeliveryBike, DeliveryCar, DeliveryVan, DeliveryTruck
- Iga sõiduk on omaette klass koos Deliver() meetodiga, mis kirjutab konsooli, kui sõiduk tarnib kaupa. Näiteks DeliveryBike klassi Deliver() meetod kirjutab “Bike makes a delivery”. Tegelikkuses on sõidukitel ka neile omased muutujad, kuid lihtsuse mõttes piirdume vaid ühe meetodiga
- Lisaks sõidukitele on olemas ka pakiautomaat, millel on ProvidePickup() meetod.
- Vedusid ja pakiautomaadist korjatud pakkide koguarvu peab saama loendada. (Printida välja kõik kokku ühe arvuna)
- Sõidukite ja pakiautomaatide objekte saab luua eraldi klassist
- Sõidukeid ja pakiautomaate saab grupeerida ning seejärel teostada vedu juba loodud grupiga. (Ühe käsuga teostatakse mitu vedu)

Ülesande sisuks on arendada kliendi nõuete kohaselt rakendus kulleriteenust pakkuvale ettevõttele.

Lae alla **DeliveryApp-0-initial-state** kaust ning ava **DeliveryApplication.sln**

Projekt sisaldab **DeliveryApp.cs** faili, kuhu läheb kogu antud ülesande jaoks vajalik programmikood. Hetkel on seal DeliveryApp klass koos Main meetodiga.

Ettevõttel on tarne jaoks eri tüüpi sõidukid. Hoolimata sõiduki tüübist, on kõigil võimekus kaupa vedada. Loome sobiva liidese.

```
public interface IDeliveryVehicle
{
   void Deliver();
}
```

Lisame ka sõidukite klassid, mis realiseerivad äsja loodud liidest.

```
public class DeliveryBike : IDeliveryVehicle
{
   public void Deliver()
   {
       Console.WriteLine("Bike makes a delivery");
   }
}
```

```
public class DeliveryCar : IDeliveryVehicle
{
   public void Deliver()
   {
       Console.WriteLine("Car makes a delivery");
   }
}

public class DeliveryVan : IDeliveryVehicle
{
   public void Deliver()
   {
       Console.WriteLine("Van makes a delivery");
   }
}

public class DeliveryTruck : IDeliveryVehicle
{
   public void Deliver()
   {
       Console.WriteLine("Truck makes a delivery");
   }
}
```

Selleks, et sõidukid saaksid pakivedu teostada, täiendame DeliveryApp klassi.

```
class DeliveryApp
{
   public static void Main(string[] args)
   {
       var deliveryApp = new DeliveryApp();
       deliveryApp.PlanDelivery();
   }

   private void PlanDelivery()
   {
       IDeliveryVehicle deliveryBike = new DeliveryBike();
       IDeliveryVehicle deliveryCar = new DeliveryCar();
       IDeliveryVehicle deliveryVan = new DeliveryVan();
       IDeliveryVehicle deliveryTruck = new DeliveryTruck();

       Console.WriteLine("Deliveries:");

       MakeDelivery(deliveryBike);
       MakeDelivery(deliveryCar);
       MakeDelivery(deliveryVan);
       MakeDelivery(deliveryTruck);
   }

   private void MakeDelivery(IDeliveryVehicle vehicle)
```

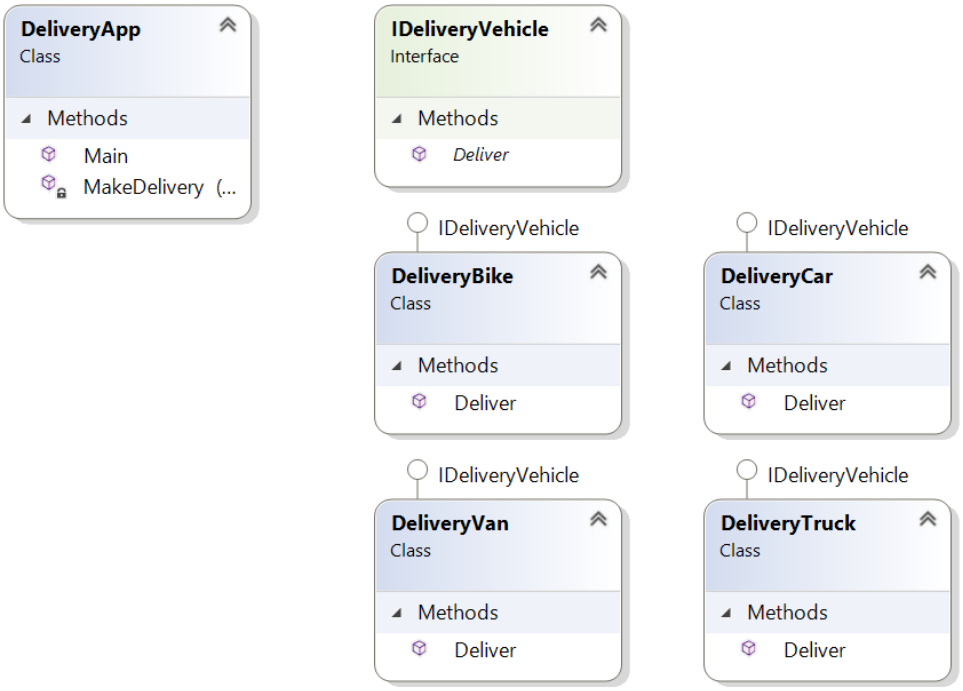
```
    {
        vehicle.Deliver();
    }
}
```

Main meetodis luuakse uus rakenduse isend (ingl.k. *instance)* ja pöördutakse isendi PlanDelivery() meetodi poole. PlanDelivery() meetodis luuakse sõidukite objektid ning teostatakse nendega kauba vedu. Kasutades polümorfismi, on tagatud, et mistahes tüüpi sõiduk MakeDelivery() meetodile argumendina ette anda, pöördutakse vastava sõiduki Deliver() meetodi poole.

Käivitame konsoolirakenduse (*Run 'Default'*). Näeme, et kõik sõidukid on edukalt tarne teostanud:

```
Deliveries:
Bike makes a delivery
Car makes a delivery
Van makes a delivery
Truck makes a delivery
```

Rakenduse seisu illustreeriv klassidiagramm on järgnev.



1. Adapter disainimuster

Ettevõtte tahab kasutusele võtta pakiautomaadid. Erinevalt sõidukitest puudub pakiautomaadil Deliver() meetod, kuid sellel on sarnast funktsiooni täitev ProvidePickup(), mis võimaldab kliendil pakile ise järele tulla. Loo uus klass:

```
public class ParcelLocker
{
   public void ProvidePickup()
   {
       Console.WriteLine("Package is picked up from a parcel locker");
   }
}
```

Selleks, et pakiautomaati saaks infosüsteemis kasutada justkui tarnet teostavad sõidukid, saab kasutada Adapter disainimustrit. Adapter võimaldab koos töötada eri klassidel, mille liidesed ei ühildu. Selle jaoks, et pakiautomaati vedude planeerimisel kasutada, saab luua ParcelLockerAdapter liidese, mis realiseerib IDeliveryVehicle liidest.

```
public class ParcelLockerAdapter : IDeliveryVehicle
{
   private ParcelLocker _parcelLocker;

   public ParcelLockerAdapter(ParcelLocker parcelLocker)
   {
       _parcelLocker = parcelLocker;
   }

   public void Deliver()
   {
       _parcelLocker.ProvidePickup();
   }
}
```

Konstruktor võtab argumendina ParcelLocker objekti, mida kohandatakse sõiduki liidesele sobivaks. Kui programm pöördub Adapteri Deliver() meetodi poole, siis delegeeritakse kutse pakiautomaadi ProvidePickup() meetodile.

Selle jaoks, et pakiautomaati töötamas näha, tuleb täiendada PlanDelivery() meetodit.

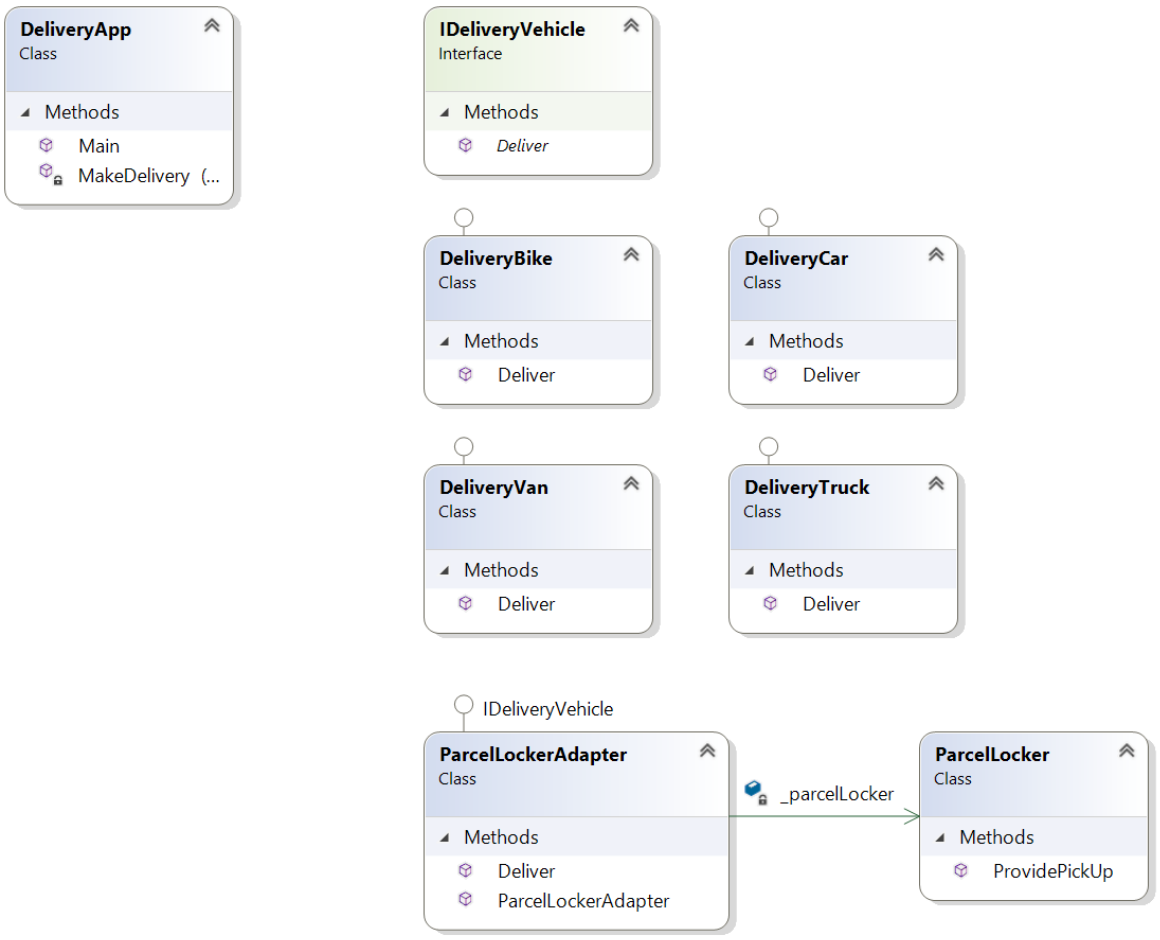
```
class DeliveryApp
{
   public static void Main(string[] args) // main method
   {
       var deliveryApp = new DeliveryApp();
       deliveryApp.PlanDelivery();
   }

   private void PlanDelivery()
```

```
    {
        IDeliveryVehicle deliveryBike = new DeliveryBike();
        IDeliveryVehicle deliveryCar = new DeliveryCar();
        IDeliveryVehicle deliveryVan = new DeliveryVan();
        IDeliveryVehicle deliveryTruck = new DeliveryTruck();
        IDeliveryVehicle locker = new ParcelLockerAdapter(new
ParcelLocker());

        Console.WriteLine("Deliveries:");

        MakeDelivery(deliveryBike);
        MakeDelivery(deliveryCar);
        MakeDelivery(deliveryVan);
        MakeDelivery(deliveryTruck);
        MakeDelivery(locker);
    }

    private void MakeDelivery(IDeliveryVehicle vehicle)
    {
        vehicle.Deliver();
    }
}
```

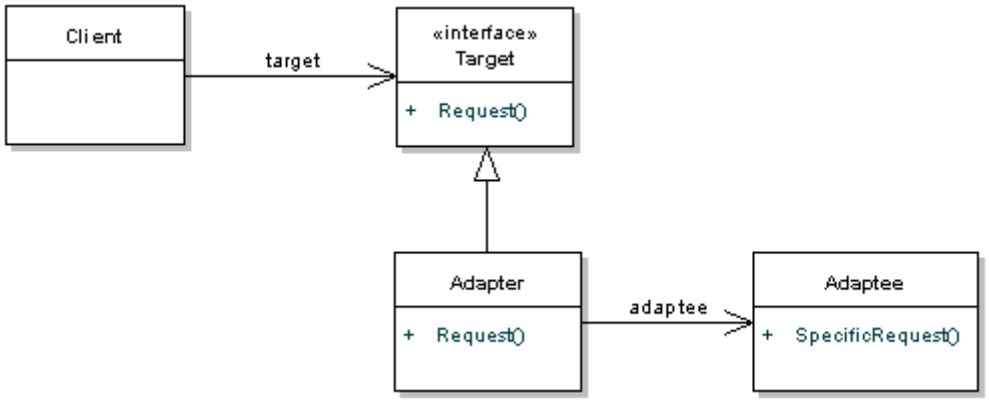
Kõigepealt loome pakiautomaadi ja pakendame (ingl.k. *wrap*) selle ParcelLockerAdapterisse, mille tulemusena käitub see justkui sõiduk. Nüüd, kui pakiautomaat on pakendatud, saab seda kasutada nagu igat teist tarnet teostavat sõidukit. Käivitades rakenduse, on tulemus järgmine:

```
Deliveries:
Bike makes a delivery
Car makes a delivery
Van makes a delivery
Truck makes a delivery
Package is picked up from a parcel locker
```

Rakenduse klassidiagrammile lisandusid ParcelLockerAdapter ja ParcelLocker klassid.



Võrdleme seda Adapter mustri UMLiga.



2. Tarnete loendamine

Ettevõtte leiab, et on tarvilik omada ülevaadet pakivedude (ja pakiautomaatidest võetud saadetiste) arvust. Selle jaoks loome uue klassi DeliveryCounter. Täienda DeliveryCounter

klassi ja DeliveryApp'i PlanDelivery() meetodit nii, et kuvataks tarnete koguarvu (sh pakiautomaat.)

```
public class DeliveryCounter : IDeliveryVehicle
{
   private IDeliveryVehicle _vehicle;
   public static int NumberOfDeliveries { get; private set; }
(võib-olla kahte eelmist rida mitte ette anda, et oleks vabamad käed?)
   //TODO: Your code here
}
```

STOP - iseseisev lahendamine

3.1 Lahendus kasutades Decorator mustrit

```
public class DeliveryCounter : IDeliveryVehicle
{
   private IDeliveryVehicle _vehicle;
   public static int NumberOfDeliveries { get; private set; }

   public DeliveryCounter(IDeliveryVehicle vehicle)
   {
       _vehicle = vehicle;
   }

   public void Deliver()
   {
       _vehicle.Deliver();
       NumberOfDeliveries++;
   }
}
```

Sarnaselt Adapterile, realiseerib DeliveryCounter IDeliveryVehicle liidest. Isendimuutuja _vehicle hoiab endas sõidukit, mida dekoreeritakse. Deklareerime muutuja NumberOfDeliveries, mis olgu staatiline, et loendaksime **kõiki** vedusid. Konstruktoris saame viite dekoreeritavale sõidukile. Kui pöördutakse Deliver() meetodi poole, siis toimub kutse delegeerimine dekoreeritavale sõidukile, pärast mida suurendatakse loendurit ühe võrra.

Teeme ka vajalikud muutused rakenduse klassis.

```
class DeliveryApp
{
   public static void Main(string[] args)
   {
       var deliveryApp = new DeliveryApp();
       deliveryApp.PlanDelivery();
```

```
    }

    private void PlanDelivery()
    {
        IDeliveryVehicle deliveryBike = new DeliveryCounter(new DeliveryBike());
        IDeliveryVehicle deliveryCar = new DeliveryCounter(new DeliveryCar());
        IDeliveryVehicle deliveryVan = new DeliveryCounter(new DeliveryVan());
        IDeliveryVehicle deliveryTruck = new DeliveryCounter(new DeliveryTruck());
        IDeliveryVehicle locker = new DeliveryCounter(new ParcelLockerAdapter(new
ParcelLocker()));

        Console.WriteLine("Deliveries:");

        MakeDelivery(deliveryBike);
        MakeDelivery(deliveryCar);
        MakeDelivery(deliveryVan);
        MakeDelivery(deliveryTruck);
        MakeDelivery(locker);

        Console.WriteLine("\nTotal parcels delivered (including pick-ups): " +
DeliveryCounter.NumberOfDeliveries);
    }

    private void MakeDelivery(IDeliveryVehicle vehicle)
    {
        vehicle.Deliver();
    }
}
```

Iga kord, kui luuakse uus sõiduki objekt, dekoreeritakse see DeliveryCounter'iga. Lõpuks prinditakse välja tarnete koguarv.

```
Deliveries:
Bike makes a delivery
Car makes a delivery
Van makes a delivery
Truck makes a delivery
Package is picked up from a parcel locker

Total parcels delivered (including pick-ups) : 5
```

3. Sõidukite loomine

Hetkel jääb sõidukite loomine rakenduse klassi PlanDelivery() meetodi hooleks. Võimalik, et tulevikus soovib ettevõte objekte luua ka teistes programmi paikades, mis võib tekitada olukorra, kus luuakse objekt ilma DeliveryCounter loenduri pakendisse panemata.

Täienda järgnevat kood, tehes vajalikud muutused DeliveryApp ja CountingDeliveryVehicleFactory klassides. Tulem peaks olema sama, mis varem.

```
public abstract class AbstractDeliveryVehicleFactory
{
   public abstract IDeliveryVehicle CreateDeliveryBike();
   public abstract IDeliveryVehicle CreateDeliveryCar();
   public abstract IDeliveryVehicle CreateDeliveryVan();
   public abstract IDeliveryVehicle CreateDeliveryTruck();
   public abstract IDeliveryVehicle CreateParcelLocker();
}

public class CountingDeliveryVehicleFactory :
AbstractDeliveryVehicleFactory
{
   //TODO: Your code here
}

class DeliveryApp
{
   public static void Main(string[] args)
   {
       AbstractDeliveryVehicleFactory factory = new
CountingDeliveryVehicleFactory();

       var deliveryApp = new DeliveryApp();
       deliveryApp.PlanDelivery(factory);
   }

   private void PlanDelivery(AbstractDeliveryVehicleFactory factory)
   {
       //TODO: Update code here

       Console.WriteLine("Deliveries:");

       MakeDelivery(deliveryBike);
       MakeDelivery(deliveryCar);
       MakeDelivery(deliveryVan);
       MakeDelivery(deliveryTruck);
       MakeDelivery(locker);
```

```
        Console.WriteLine("\nTotal parcels delivered (including pick-ups): "
+ DeliveryCounter.NumberOfDeliveries);
    }

    private void MakeDelivery(IDeliveryVehicle vehicle)
    {
        vehicle.Deliver();
    }
}
```

STOP - iseseisev lahendamine

4.1 Lahendus kasutades Abstract Factory mustrit

Kõigepealt võib luua Factory, mis loob sõidukeid, ilma neid DeliveryCounter'isse pakkimata.

```
public class DeliveryVehicleFactory : AbstractDeliveryVehicleFactory
{
    public override IDeliveryVehicle CreateDeliveryBike()
    {
        return new DeliveryBike();
    }

    public override IDeliveryVehicle CreateDeliveryCar()
    {
        return new DeliveryCar();
    }

    public override IDeliveryVehicle CreateDeliveryVan()
    {
        return new DeliveryVan();
    }

    public override IDeliveryVehicle CreateDeliveryTruck()
    {
        return new DeliveryTruck();
    }

    public override IDeliveryVehicle CreateParcelLocker()
    {
        return new ParcelLockerAdapter(new ParcelLocker());
    }
}
```

DeliveryVehicleFactory laiendab (ingl.k. *extends*) ülesande alguses loodud Abstract Factory klassi. Iga meetod loob eri tüüpi sõiduki. Nüüd, kui rakenduse PlanDelivery() meetod sõiduki

objekte kasutab, siis teeb ta seda teadmata, mis objekt talle tagastatakse. Teada on vaid see, et see on IDeliveryVehicle tüüpi.

Loome nüüd Factory koos loenduriga.

```
public class CountingDeliveryVehicleFactory :
AbstractDeliveryVehicleFactory
{
   public override IDeliveryVehicle CreateDeliveryBike()
   {
       return new DeliveryCounter(new DeliveryBike());
   }

   public override IDeliveryVehicle CreateDeliveryCar()
   {
       return new DeliveryCounter(new DeliveryCar());
   }

   public override IDeliveryVehicle CreateDeliveryVan()
   {
       return new DeliveryCounter(new DeliveryVan());
   }

   public override IDeliveryVehicle CreateDeliveryTruck()
   {
       return new DeliveryCounter(new DeliveryTruck());
   }

   public override IDeliveryVehicle CreateParcelLocker()
   {
       return new DeliveryCounter(new ParcelLockerAdapter(new
ParcelLocker()));
   }
}
```

Täiendame DeliveryApp klassi

```
class DeliveryApp
{
   public static void Main(string[] args)
   {
       AbstractDeliveryVehicleFactory factory = new
CountingDeliveryVehicleFactory();

       var deliveryApp = new DeliveryApp();
       deliveryApp.PlanDelivery(factory);
   }

   private void PlanDelivery(AbstractDeliveryVehicleFactory factory)
   {
```

```
        IDeliveryVehicle deliveryBike = factory.CreateDeliveryBike();
        IDeliveryVehicle deliveryCar = factory.CreateDeliveryCar();
        IDeliveryVehicle deliveryVan = factory.CreateDeliveryVan();
        IDeliveryVehicle deliveryTruck = factory.CreateDeliveryTruck();
        IDeliveryVehicle locker = factory.CreateParcelLocker();

        Console.WriteLine("Deliveries:");

        MakeDelivery(deliveryBike);
        MakeDelivery(deliveryCar);
        MakeDelivery(deliveryVan);
        MakeDelivery(deliveryTruck);
        MakeDelivery(locker);

        Console.WriteLine("\nTotal parcels delivered (including pick-ups): "
+ DeliveryCounter.NumberOfDeliveries);
    }

    private void MakeDelivery(IDeliveryVehicle vehicle)
    {
        vehicle.Deliver();
    }
}
```

Kõigepealt luuakse factory, mis antakse parameetrina PlanDelivery() meetodile. PlanDelivery() meetod aktsepteerib AbstractDeliveryVehicleFactory tüüpi ning kasutab seda sõidukite loomisel, selle asemel, et neid ise luua.

Käivitades rakenduse, saame sama tulemuse, mis varem.

```
Deliveries:
Bike makes a delivery
Car makes a delivery
Van makes a delivery
Truck makes a delivery
Package is picked up from a parcel locker

Total parcels delivered (including pick-ups) : 5
```

4. Sõidukite grupeerimine

Luues rohkem sõiduki objekte, muutub nende haldamine üha keerukamaks. Ettevõte soovib, et töötataks välja moodus, kuidas sõidukeid grupeerida. Lõpeta puuduolevad osad koodis.

```
class DeliveryApp
{
   public static void Main(string[] args)
   {
       AbstractDeliveryVehicleFactory factory = new
CountingDeliveryVehicleFactory();

       var deliveryApp = new DeliveryApp();
       deliveryApp.PlanDelivery(factory);
   }

   private void PlanDelivery(AbstractDeliveryVehicleFactory factory)
   {
       IDeliveryVehicle deliveryBike = factory.CreateDeliveryBike();
       IDeliveryVehicle deliveryCar = factory.CreateDeliveryCar();
       IDeliveryVehicle deliveryVan = factory.CreateDeliveryVan();
       IDeliveryVehicle deliveryTruck = factory.CreateDeliveryTruck();
       IDeliveryVehicle locker = factory.CreateParcelLocker();

       Fleet fleetOfVehicles = new Fleet();

       fleetOfVehicles.Add(deliveryBike);
       fleetOfVehicles.Add(deliveryCar);
       fleetOfVehicles.Add(deliveryVan);
       fleetOfVehicles.Add(deliveryTruck);
       fleetOfVehicles.Add(locker);

       Console.WriteLine("Deliveries:");

       MakeDelivery(fleetOfVehicles);

      //TODO: Using previous code as an example, create a fleet of 3
delivery trucks and then make deliveries

       Console.WriteLine("\nTotal parcels delivered (including pick-ups): "
+ DeliveryCounter.NumberOfDeliveries);
   }

   private void MakeDelivery(IDeliveryVehicle vehicle)
   {
       vehicle.Deliver();
   }
}
```

Siin luuakse kõige pealt kogum sõidukite jaoks ning seejärel lisatakse sellesse olemasolevad sõidukid. Lõpuks pöördutakse MakeDelivery() meetodidi poole, argumendina äsja loodud kogum.

Sarnaselt näitele, **loo kolm uut veoautot ja uus kogum, kuhu neid grupeerida**. Lisa ka puuduolev funktsionaalsus Fleet klassile (Add ja Deliver meetodid). Listi jaoks kasuta teeki System.Collections.Generic

```
public class Fleet : IDeliveryVehicle
{
   List<IDeliveryVehicle> _vehicles = new List<IDeliveryVehicle>();

   //TODO: Create an Add method for adding vehicles to the fleet and a
Deliver method that makes a delivery with each vehicle in the fleet

}
```

STOP - iseseisev lahendamine

5.1 Lahendus kasutades Composite mustrit

```
private void PlanDelivery(AbstractDeliveryVehicleFactory factory)
{
   IDeliveryVehicle deliveryBike = factory.CreateDeliveryBike();
   IDeliveryVehicle deliveryCar = factory.CreateDeliveryCar();
   IDeliveryVehicle deliveryVan = factory.CreateDeliveryVan();
   IDeliveryVehicle deliveryTruck = factory.CreateDeliveryTruck();
   IDeliveryVehicle locker = factory.CreateParcelLocker();

   Fleet fleetOfVehicles = new Fleet();

   fleetOfVehicles.Add(deliveryBike);
   fleetOfVehicles.Add(deliveryCar);
   fleetOfVehicles.Add(deliveryVan);
   fleetOfVehicles.Add(deliveryTruck);
   fleetOfVehicles.Add(locker);

   Console.WriteLine("Deliveries:");

   MakeDelivery(fleetOfVehicles);

   var fleetOfTrucks = new Fleet();

   IDeliveryVehicle truckOne = factory.CreateDeliveryTruck();
   IDeliveryVehicle truckTwo = factory.CreateDeliveryTruck();
   IDeliveryVehicle truckThree = factory.CreateDeliveryTruck();
```

```
    fleetOfTrucks.Add(truckOne);
    fleetOfTrucks.Add(truckTwo);
    fleetOfTrucks.Add(truckThree);

    Console.WriteLine("\nDeliveries made by the new fleet of trucks:");
    MakeDelivery(fleetOfTrucks);

    Console.WriteLine("\nTotal parcels delivered (including pick-ups): " +
DeliveryCounter.NumberOfDeliveries);
}

...

public class Fleet : IDeliveryVehicle
{
    List<IDeliveryVehicle> _vehicles = new List<IDeliveryVehicle>();

    public void Add(IDeliveryVehicle vehicle)
    {
        _vehicles.Add(vehicle);
    }

    public void Deliver()
    {
        foreach (IDeliveryVehicle vehicle in _vehicles)
        {
            vehicle.Deliver();
        }
    }
}
```

Käivitades rakenduse, saame järgneva tulemuse.

```
Deliveries:
Bike makes a delivery
Car makes a delivery
Van makes a delivery
Truck makes a delivery
Package is picked up from a parcel locker

Deliveries made by the new fleet of trucks:
Truck makes a delivery
Truck makes a delivery
Truck makes a delivery

Total parcels delivered (including pick-ups): 8
```