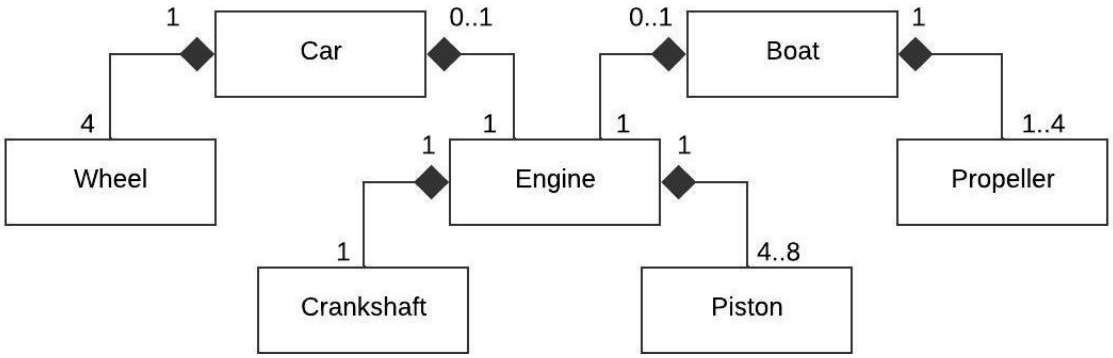
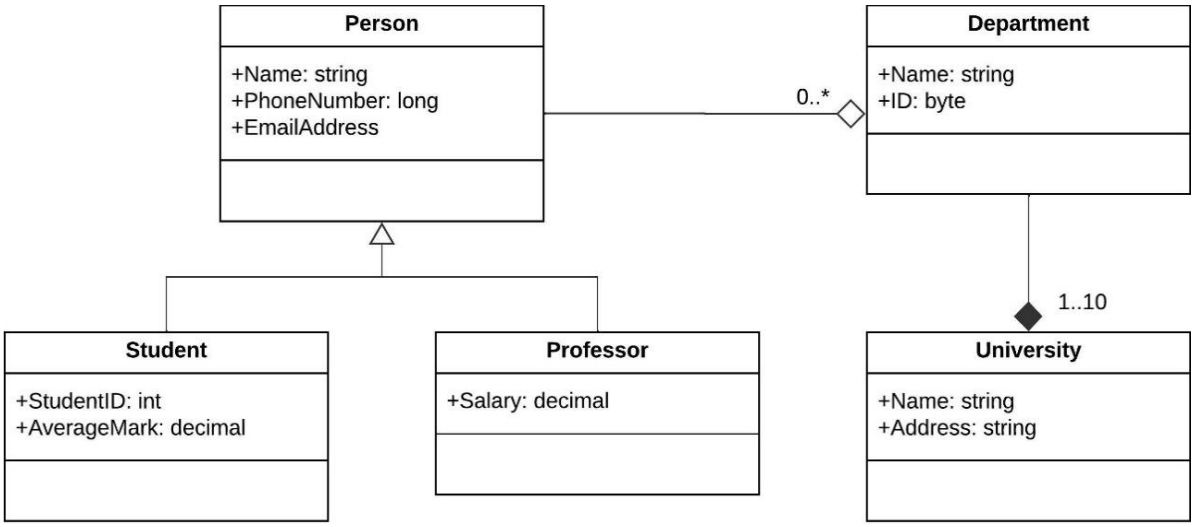
Oefening 89: Vehicles and parts UML-diagram

Implementeer de onderstaande UML-diagram in code.



Oefening 90: University UML-diagram

Implementeer de onderstaande UML-diagram in code.



Oefening 91: Moederbord

Schrijf een programma met de klasse `Moederbord` die een moederbord van een computer voorstelt. Je mag hiervoor kiezen uit een van de vele moederborden die je online vindt. Ik geef je alvast mijn huidige moederbord ter inspiratie.

Ga vervolgens na welke (belangrijke) aansluitingen het gekozen moederbord heeft ('heeft een'). Maak voor iedere aansluiting een aparte klasse die via **compositie** een onderdeel wordt van het moederbord. Denk eraan dat als je bijvoorbeeld meerdere RAM-sloten hebt en hier dus een array of List moet voorzien van het type RAM.

Voorzie de volgende methoden in de klasse `Moederbord`:

- Een methode `AangeslotenOnderdelen()` dat de naam van elk aangesloten onderdeel weergeeft.
- een methode `VrijeAansluitingen()` dat weergeeft welke aansluitingen nog leeg (null) zijn.

Ieder component moet via een property langs buiten ingesteld worden in de methode `Main()`:

```
Moederbord PRIME_X570_P = new Moederbord();
PRIME_X570_P.CPU = new CPUSocket("AMD Ryzen 5 360");
PRIME_X570_P.SSD = new M2Slot("1TB 970 EVO NVMe");
//…
```

Opmerking:
Deze oefening kan erg tijdrovend zijn als je een moederbord tot in detail wilt namaken. Het is aangeraden om slechts enkele computeronderdelen toe te voegen aan het moederbordprogramma. Als je later nog tijd over hebt, kan je deze oefening eventueel uitbreiden. Wees alleszins verstandig met je oefentijd!

(Uitgebreid) voorbeeld uitvoer:

```
Aanroep van de methode AangeslotenOnderdelen():
- Processor: AMD Ryzen 5 360
- SSD: 1TB 970 EVO NVMe.
- Voeding: Corsair RM650 650W.
- RAM-slot 1: DDR4 16GB 3200-16 Vengeance.
- RAM-slot 2: DDR4 16GB 3200-16 Vengeance.
- PCIeX16-slot 1: GeForce RTX 2060
- SATA-slot 1: Seagate BarraCuda 2 TB harde schijf
- Fanheader 1: be quiet! Pure Wings 2
- Fanheader 2: be quiet! Pure Wings 2
- Fanheader 3: be quiet! Pure Wings 2

Aanroep van de methode VrijeAansluitingen():
- Er is geen RAM-geheugen aanwezig in RAM-slot 3.
- Er is geen RAM-geheugen aanwezig in RAM-slot 4.
- Er is geen videokaart aanwezig in PCIeX16-slot 2.
- Er is geen geluids- of netwerkkaart aanwezig in PCIeX1-slot 1.
- Er is geen geluids- of netwerkkaart aanwezig in PCIeX1-slot 2.
- Er is geen drive of HDD aanwezig in SATA-slot 2.
- Er is geen drive of HDD aanwezig in SATA-slot 3.
- Er is geen drive of HDD aanwezig in SATA-slot 4.
```

EXTRA: Voorzie ook klassen voor elk **computeronderdeel** dat je vervolgens via **aggregatie** toevoegt aan de juiste aansluitingen op het moederbord.

```
Moederbord PRIME_X570_P = new Moederbord();

//Processor apart aanmaken en vervolgens toevoegen
CPU cpu = new CPU("AMD Ryzen 5 360");
PRIME_X570_P.CPU = new CPUSocket(cpu);
//…
```

Oefening 92: Politiek

Maak een programma om de politieke situatie van een land te simuleren met de klassen:

- `Minister:`
  - Autoproperty `Naam` van het type `string`.
- `President:`
  - Een President is een minister.
  - Een extra autoproperty `Teller` met private setter van het type `int` met 4 als beginwaarde.
  - Een methode `JaarVerder()` die de `Teller` bij iedere aanroep verlaagt met 1.
- `Land:`
  - Een land heeft 0 of 1 president.
  - Een land heeft 0 of 1 eerste minister.
  - Een land heeft 0 tot 4 ministers (via een `List<Minister>`).

  Al deze compositieobjecten zijn private!

  - Een methode `MaakRegering()` die de volgende parameters aanvaardt:
    - 1 president object die aan de private president variabele wordt toegekend
    - Een `List<Minister>` object waarin tussen de 1 tot en met 5 ministers in staan: de eerste minister in de lijst wordt toegewezen aan de private eerste minister variabele. De overige ministers in de lijst worden aan de private lijst van ministers toegewezen.

  Deze methode zal enkel iets doen indien er geen president in het land is (null). Indien er reeds een regering is dan zal er een foutboodschap verschijnen.

  - Een methode `JaarVerder()` dat de methode `JaarVerder()` van de president aanroept indien deze er is (en dus niet null is). Deze methode controleert ook of de `Teller` van de president na deze aanroep op 0 staat. Als dat het geval is dan worden alle ministers en president in het land op null gezet.
  - Een methode `ToonRegering()` dat een overzicht van de leden van de regering op het scherm toont.

Regeerperiode

Controleer je klasse `Land` door enkele ministers en een president te maken en deze in een object van het type `Land` via de methode `MaakRegering()` door te geven. Test dan wat er gebeurt indien je enkele malen de methode `JaarVerder()` op het land aanroept.

Voorbeeld uitvoer:

```
Gaat niet, want dit land heeft al een regering
President: Kristof Provoost
Eerste minister: Henry Van Damme
Minister 1: Martine De Wit
Minister 2: Marcel De Backer
Minister 3: Olivier Verschueren
Minister 4: Sabine Haegedoorns
Weer een jaar verder
Weer een jaar verder
Weer een jaar verder
Weer een jaar verder
Regering is gedaan
```

EXTRA: Verkiezingen

Maak een klasse `VerkiezingsUitslag`. Deze klasse heeft een default constructor die volgende twee properties random waarden zal geven:

- Een full property met private set `VerkozenPresident` van het type `President`.
- Een full property met private set `VerkozenMinisters` van het type `List<Minister>`.

Maak een `VerkiezingsUitslag`-object aan in de methode `Main()` en gebruik deze om de methode `MaakRegering()` van je `Land` van de nodige informatie te voorzien.

Tip:
Je kan een lijst met namen van kandidaten voor de functie van minister opnemen in deze klasse en de methode `MaakRegering()` hieruit namen laten trekken voor de nieuwe regering. Om je op weg te helpen, krijg je al een lijst met willekeurige namen:

"Henry Van Damme",
"Martine De Wit",
"Marcel De Backer",
"Olivier Verschueren",
"Sabine Haegedoorns",
"Dirk De Winter",
"Philippe Lemaire",
"Olivier Hermans",
"Xavier D'Hondt",
"Lauranne De Boeck",
"Tine Slaegers",
"Tom Vechter",
"Laurant De Champs",
"Nathalie Verbeeck",
"Herman De Puydt",
"Sarah De Maesschalk",
"Cédric De Haene"

Voor de gemakkelijkheid zullen we stellen dat elke kandidaat slechts één termijn in een regering mag zitten.

Voorbeeld uitvoer:

```
President: Tine Slaegers
Eerste minister: Olivier Hermans
Minister 1: Marcel De Backer
Minister 2: Laurant De Champs
Minister 3: Tine Slaegers
Minister 4: Felix Verhofstad
Weer een jaar verder
Weer een jaar verder
Weer een jaar verder
Weer een jaar verder
Regering is gedaan
```

EXTRA 1: Database

Uitdaging:
Maak voor de verandering deze oefening, inclusief commentaarregels, volledig in het Engels.

Waarom? Omdat de kans bestaat dat je later in aanraking zal komen met programmacode die die internationaal werd ontwikkeld. In de IT-wereld is Engels de voertaal, dus het is vanzelfsprekend dat alle communicatie in het Engels verloopt. Veel succes!

Deel 1: Databaseverbinding

Ontwerp een databaseverbinding. Om toegang te krijgen tot een database, moeten we eerst een verbinding openen en deze sluiten zodra ons werk is voltooid. Het verbinden met een database hangt af van het type database en het databasebeheersysteem (Databasemanagementsysteem of DBMS). Het verbinden met bijvoorbeeld een SQL Server database is anders dan het verbinden met een Oracle database. Maar beide verbindingen hebben een enkele zaken met elkaar gemeen:

- Ze hebben een verbindingsreeks (connection string)
- Ze kunnen geopend worden
- Ze kunnen gesloten worden
- Ze kunnen een time-out eigenschap hebben (dus als de verbinding niet binnen een bepaalde tijd geopend kan worden, wordt er een uitzondering gegenereerd).

Jouw taak is om deze gemeenschappelijkheid te representeren in een basisklasse `DbConnection`. Voorzie deze klasse van de volgende eigenschappen:

- `ConnectionString` van het type `string`.
- `Timeout` van het type `TimeSpan`.

Een `DbVerbinding` zal niet in een geldige staat zijn als het geen verbindingsreeks heeft. Dus je moet een verbindingsreeks doorgeven in de constructor van deze klasse. Houd ook rekening met scenario's waarbij null of een lege string wordt doorgegeven als de verbindingsreeks. Zorg er ook voor dat een uitzondering wordt opgeworpen om te garanderen dat je klasse altijd in een geldige staat zal zijn.

Voorzie ook methoden voor het openen en sluiten van een verbinding. We weten niet hoe we een verbinding moeten openen of sluiten in een `DbConnection` en dit moet worden overgelaten aan de klassen die van `DbConnection` afstammen. Deze klassen, zoals `SqlConnection` of `OracleConnection`, zullen de feitelijke implementatie bieden. Het volstaat dus om in de basisklasse deze methoden als abstract te declareren.

Maak hiervan twee afgeleide klassen: `SqlConnection` en `OracleConnection`, en geef een eenvoudige implementatie van het openen en sluiten van verbindingen met behulp van `Console.WriteLine()`.

In de echte wereld biedt SQL Server een API voor het openen of sluiten van een verbinding met een database. Maar daar gaan we ons nu geen zorgen over maken. Een eenvoudige boodschap dat een verbinding geopend of gesloten wordt volstaat voor de oefening.

Test de code door een instantie van beide childklassen te maken en alle methoden aan te roepen in de methode `Main()`.

Voorbeeld uitvoer:

```
Opening SQL Server database connection.
Closing SQL Server database connection.

Opening Oracle Server database connection.
Closing Oracle Server database connection.
```

Deel 2: Databasecommando

Nu we het concept van een `DbConnection` hebben, laten we uitwerken hoe we een `DbCommand` kunnen vertegenwoordigen.

Ontwerp een klasse genaamd `DbCommand` voor het uitvoeren van een instructie in de database. Een `DbCommand` kan niet in een geldige staat zijn zonder een verbinding. Daarom moet een `DbConnection` met een instructie als parameters worden ingegeven in de overloaded constructor van deze klasse. Hou hierbij rekening met het feit dat deze niet null mogen zijn!

Elke `DbCommand` moet ook de instructie hebben die naar de database moet worden gestuurd. In het geval van SQL Server wordt deze instructie uitgedrukt in de T-SQL taal (bijvoorbeeld `SELECT * FROM Users`). Gebruik een string om deze instructie te representeren. Ook hier kan een commando niet in een geldige staat zijn zonder deze instructie. Zorg er dus voor dat je deze ontvangt in de constructor, rekening houdend met een null-referentie of een lege string.

Elke opdracht moet uitvoerbaar zijn. Deze nemen we op in een methode genaamd `Execute()`. In deze methode hebben we een eenvoudige implementatie nodig als volgt:

- Open de verbinding,
- Voer de instructie uit,
- Sluit de verbinding.

Merk op dat we hier, binnen de `DbCommand`, een referentie hebben naar `DbConnection`. Afhankelijk van het type `DbConnection` dat tijdens runtime wordt doorgegeven, zal het openen en sluiten van een verbinding anders zijn. Bijvoorbeeld, als we deze `DbCommand` initialiseren met een `SqlConnection`, zullen we een verbinding openen en sluiten naar een SQL Server database (polymorfisme gedrag). Interessant genoeg, maakt `DbCommand` zich niet druk over hoe een verbinding wordt geopend of gesloten, omdat dit niet zijn verantwoordelijkheid is (abstractie). Het enige wat belangrijk is voor deze klasse, is het versturen van een instructie naar een database.

Ook voor het uitvoeren van de instructie, druk deze eenvoudig af naar de Console.

Initialiseer instanties van `DbCommand` met een string als de instructie die een commando uitvoert in een SQL en een Oracle database. Voer het commando uit en bekijk het resultaat op de console.

Voorbeeld uitvoer:

```
Opening SQL Server database connection.
Executing instruction: SELECT * FROM Users.
Closing SQL Server database connection.

Opening Oracle Server database connection.
Executing instruction: SELECT * FROM Employees.
Closing Oracle Server database connection.
```