FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

# Project Objectgeoriënteerd Programmeren
# **Rubikskubus**

Academiejaar 2022 -2023

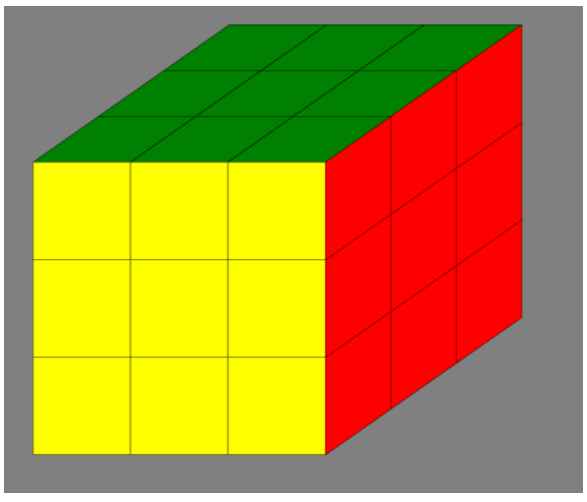
## INLEIDING

Het project draait om de welbekende Rubikskubus. We willen deze structuur in code vatten, zodat hij getekend en gemanipuleerd kan worden op het scherm. Het einddoel is dus een werkende Graphical User Interface (GUI).

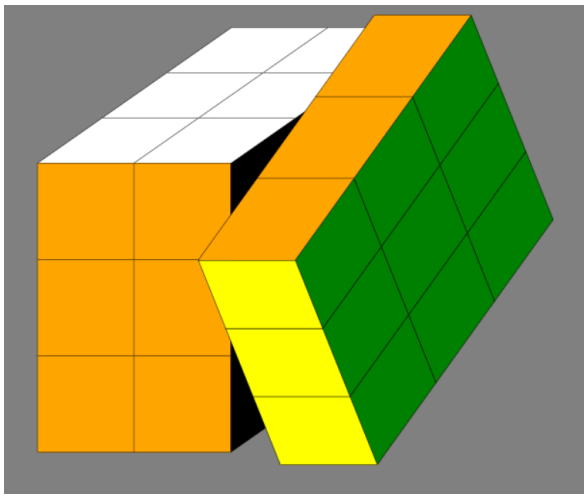
Hieronder zien jullie alvast twee posities van de Rubikskubus: als hij helemaal gemaakt is, en op het moment dat er een zijvlak gedraaid wordt.

Omdat GUI's niet behandeld worden in de cursus OGP, stellen we een duidelijke taakverdeling op: jullie lesgevers zijn leveranciers van de **view** van de toepassing (het tekenwerk); elk duo studenten is verantwoordelijk voor de implementatie van een bijpassend **model** (de data achter de tekening). Voor dat model volstaat de leerstof van de cursus OGP.

Uiteraard moeten beide stukken code op elkaar afgestemd worden. Het team dat het model verzorgt (dat is elk studententeam, hierna het *modelteam* genoemd) produceert best code die beantwoordt aan de vraag van het team dat de view verzorgt (jullie lesgevers, het *showteam*). Daarvoor zullen de nodige afspraken gemaakt worden. Het staat het modelteam in principe vrij om achter de schermen z'n eigen keuzes te maken - zolang tegemoet wordt gekomen aan de uiteindelijke vraag van het showteam.



**Figuur 1:** startpositie



**Figuur 2:** tijdens manipulatie

# EINDDOEL

Een Rubikskubus bestaat uit $3^3$ kleine kubusjes, die elk een aantal gekleurde zijvlakjes hebben (de rest van de zijvlakjes is zwart). Zoek de officiële kleuren van de Rubikskubus op - gebruik het Westerse schema.

We vallen met de deur in huis, en verklappen al wat er op het einde van het project gecodeerd moet zijn. Het showteam verwacht van elk modelteam een klasse RubiksKubus die drie publieke methodes heeft.

- De methode getAlleVlakken() geeft een lijst met alle zijvlakjes van alle kubusjes. (Hoeveel zijn er dat dan?)
- De methode getGekleurdeVlakken() geeft een lijst met enkel de zijvlakjes die niet zwart zijn. (Hoeveel zijn er dat dan?)
- De methode getZichtbareVlakkenInOngemanipuleerdePositie() geeft een lijst met enkel de zijvlakjes die zichtbaar zijn als de Rubikskubus niet gemanipuleerd wordt (hij heeft dus nog de vorm van een kubus), en getekend wordt zoals in figuur 1.

De informatie die het tekenprogramma nodig heeft om een zijvlakje te tekenen is beperkt: de kleur, en de coördinaten van de vier hoekpunten. Beperk u dan ook tot die informatie; meer wil het showteam niet geleverd krijgen.

Uiteraard moet hier eerst wat denkwerk aan vooraf gaan. Het project wordt opgedeeld in sprints. In een sprint wordt normaliter een foutloos werkend project afgeleverd (waarbij elke sprint functionaliteit toevoegt aan de vorige), maar voor sprint 0 houden we het nog kalm.

# SPRINT 0

Wat we verwachten op het einde van sprint 0:

1. een voorstel voor de plaatsing van het assenkruis (X,Y,Z) met argumentatie van deze keuze
2. een handgeschreven schets van één kubusje, nl. dat wat op Figuur 1 de kleuren rood, geel en groen heeft, met coördinaten van de 8 hoekpunten
3. een opsomming van de data die voor dat kubusje bewaard worden (zowel namen van de geheugenplaatsen als de concrete waarden die er voor die kubus in staan)
4. een handgeschreven schets van de hele Rubikskubus, met coördinaten erbij zodat de locatie van elk van de $3^3$ kubusjes gekend is

Allicht zal er druk overleg gepleegd worden over bovenstaande vragen. Luister goed naar de termen die jijzelf en je teamgenoot gebruiken. Elk zelfstandig naamwoord dat aan bod komt bij het beschrijven van de Rubikskubus (zoals *kubusje* en *hoekpunt*) is een kandidaat voor de naam van een klasse. Een klasse beschrijft de blauwdruk van gelijkaardige objecten.

5. Geef voor élk zelfstandig naamwoord dat volgens jullie *er toe doet* (in het kader van dit project) volgende zaken:
   (a) het aantal objecten of instanties dat er van dit type in de Rubikskubus nodig zijn (geef minstens tussenberekeningen, eventueel ook de einduitkomst; dus liever $2 \cdot 4 \cdot 6$ dan 48).

(b) de informatie die elk object van dit type moet bewaren, om zijn positie in de Rubikskubus (en dus in 3D) duidelijk te maken (ook als er tussendoor aan de Rubikskubus gedraaid wordt).

(c) de diensten die elk object van dit type moet kunnen uitvoeren, op vraag van de gebruiker van de Rubikskubus (bijvoorbeeld tijdens het verdraaien van een zijvlak) of om op de drie vragen van het showteam (zie paragraaf EINDDOEL) te kunnen antwoorden.

We willen heel expliciet nog GEEN code zien. Enkel een verslag van jullie denkwerk. Omdat we echter weten dat het heel verleidelijk is om wel meteen in code te duiken, zijn we jullie voor. In Figuur 4 geven wíj een stuk code (in JavaScript), en laten er meteen ons kritisch licht over schijnen. In deze figuur zie je de initialisatie van één kubusje, met dus zes vlakken (enkel de code voor de drie eerste vlakken wordt getoond, de eigenlijke code is dubbel zo lang als getoond).

```
initialiseVlakken(x,y,z){
    return [
        //ondervlak
        [
            new Hoek(x,y+this.zijde,z),
            new Hoek(x+this.zijde, y + this.zijde,z),
            new Hoek(x+this.zijde +this.zijde/2,y + this.zijde/2,z),
            new Hoek(x+this.zijde/2,y+this.zijde/2,z)
            ],
        //Linkervlak
        [
            new Hoek(x,y,z),
            new Hoek(x+this.zijde/2, y-this.zijde/2,z),
            new Hoek(x+this.zijde/2,y + this.zijde/2,z),
            new Hoek(x,y+this.zijde,z)
            ],
        //achtervlak
        [
            new Hoek(x+this.zijde/2,y - this.zijde/2,z),
            new Hoek(x+this.zijde/2 + this.zijde, y - this.zijde/2,z),
            new Hoek(x+this.zijde/2 + this.zijde, y + this.zijde/2,z),
            new Hoek(x+this.zijde/2, y + this.zijde/2,z),
            ],
```

**Figuur 3:** don't try this at home

- Om te beginnen worden er teveel objecten aangemaakt: een kubusje heeft maar 8 hoekpunten, terwijl er in deze code $6 \cdot 4 = 24$ nieuwe `Hoek`-objecten aangemaakt worden. Zorg dat er een 1-op-1-relatie is tussen de objecten in de realiteit en de objecten in het programma.

- Er worden redelijk wat berekeningen uitgevoerd. Kunnen die halveringen niet vermeden worden?
- De berekeningen werden met de hand gedaan: er werd hargecodeerd opgesomd welke vier hoekpunten tot welk zijvlak behoren. Dat is foutgevoelig en zenuwslopend werk. We hopen dat dit in jullie code overgelaten wordt aan de rekenkracht van de computer: geef het programma de juiste formules om te bepalen wanneer een hoekpunt tot een vlak behoort. Onthoud: we vragen nog *geen* code! Maar je mag alvast lopen tobben over formules of manieren om berekeningen uit te spenderen aan het programma.
- Als we de initialisatie van een viertal hoeken bekijken, dan staan de parameters van de opgeroepen constructoren niet netjes onder elkaar.

```
[
    new Hoek(x+this.zijde/2,y - this.zijde/2,z),
    new Hoek(x+this.zijde/2 + this.zijde, y - this.zijde/2,z),
    new Hoek(x+this.zijde/2 + this.zijde, y + this.zijde/2,z),
    new Hoek(x+this.zijde/2, y + this.zijde/2,z),
],
```

**Figuur 4:** schots en scheef

Op zich is dat uiteraard geen ramp, maar de juiste technieken zorgen ervoor dat initialisaties van grote aantallen objecten overzichtelijker gebeurt. Dat zal een medeprogrammeur (een *lezer* van de code) sneller overtuigen van de correctheid van de code. Het is dus een edel streven om de hardgecodeerde initialisering van objecten zo doorzichtig mogelijk te maken.

Vergelijk het hiermee: stel dat de output van een programma visueel gecontroleerd moet worden op fouten. Deze output kan aangeboden worden zoals links, of zoals rechts. In welke output valt de fout het snelst op?

```
grondtal is 2
exponent is 0
macht is 1
grondtal is 2
exponent is 1
macht is 2
grondtal is 2
exponent is 2
macht is 4
grondtal is 3
exponent is 3
macht is 8
grondtal is 2
exponent is 4
macht is 16
grondtal is 2
exponent is 5
macht is 32
```

```
grondtal is 2  exponent is 0   macht is 1
grondtal is 2  exponent is 1   macht is 2
grondtal is 2  exponent is 2   macht is 4
grondtal is 3  exponent is 3   macht is 8
grondtal is 2  exponent is 4   macht is 16
grondtal is 2  exponent is 5   macht is 32
```

# SPRINT 1

Wat we verwachten op het einde van sprint 1:

- Het project `sprint_1` (download zip-bestand van Ufora) aangevuld met de nodige code, zodat het programma uit het bestand `RubiksViewer.java` niet één blokje toont, maar de hele Rubikskubus.
- Daarvoor schrijf je een klasse die een Rubikskubus voorstelt, en die een methode heeft die alle $6 \cdot 3^3$ vlakjes van de kubus teruggeeft. En mogelijks wat hulpklassen :-)

Merk op: er wordt nog geen code gevraagd die met rotaties te maken heeft. Enkel de initiële opstelling van de kubus moet gegeven worden, lees: de constructor van de Rubikskubus moet werken. En, om te kunnen tekenen, uiteraard ook de gevraagde methode `getAllFaces()`.

In het labo van maandag 24 of dinsdag 25 oktober werden de oplossingen van Sprint 0 overlopen. Je kan de gebruikte powerpoint (vanaf dinsdag 17u) terugvinden op Ufora.

Hieronder een aantal tips die van pas kunnen komen.

### Project opstarten

1. Er werd jullie al verteld dat de hoofdzonde *'luiheid'* geen zonde is voor een programmeur (mits oordeelkundig ingezet). Die wordt dus uit het lijstje van zeven hoofdzondes geschrapt. Er komt er echter eentje in de plaats, we gieten het in de vorm van een verbod.

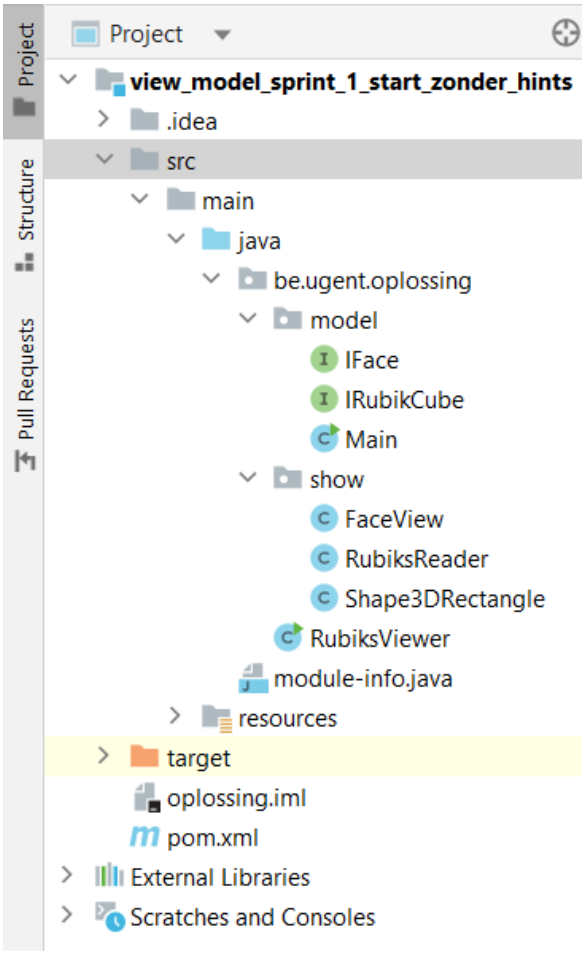
   ```
   Gij zult uw goederen niet bewaren in een map
   waarvan de naam een spatie bevat.
   Gij zult ook alle instanties mijden
   die u hiertoe trachten te verleiden.
   ```

   Ga zorgvuldig na of de hele padnaam van de map waarin straks het project komt, geen spaties bevat. En mijd ook maar meteen OneDrive.

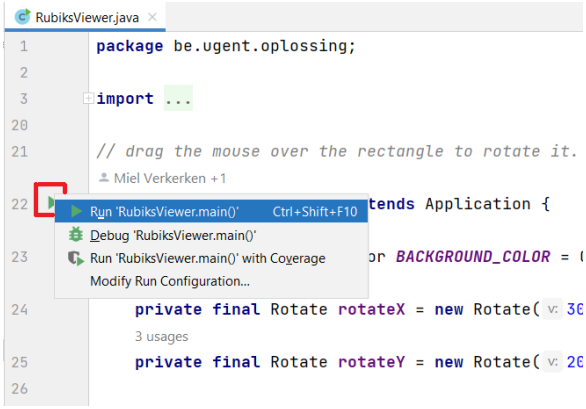
   Toch gezondigd? Uw penitentie bestaat uit het lijdzaam dulden van de `FileNotFoundException`.

2. Bij het **unzippen** van het project, kijk je eerst goed naar de structuur van de verkregen map. Het zou kunnen dat je twee mappen met dezelfde naam in elkaar hebt staan. Dan is de binnenste map de projectmap; dat zie je omdat de projectmap een ("onzichtbare") map bevat met de naam *.idea*. (Let op: van zodra je een niet-projectmap opent in Intellij, voegt Intellij deze *.idea*-map toe. Dus *wacht* met openen in Intellij tot je zeker weet welke map je moet openen. Komt deze tip te laat? Verwijder dan de per ongeluk aangemaakte *.idea*-map uit de bovenste map.)

3. Open het project in Intellij, en **bekijk de projectstructuur** in het venster *Project* (Figuur 5). Open daarna het bestand `RubiksViewer.java` en laat dit programma lopen (rechtermuisknop op groene pijl in linkermarge, Figuur 6). Levert dit problemen op, ga dan naar volgend puntje - anders sla je het over.

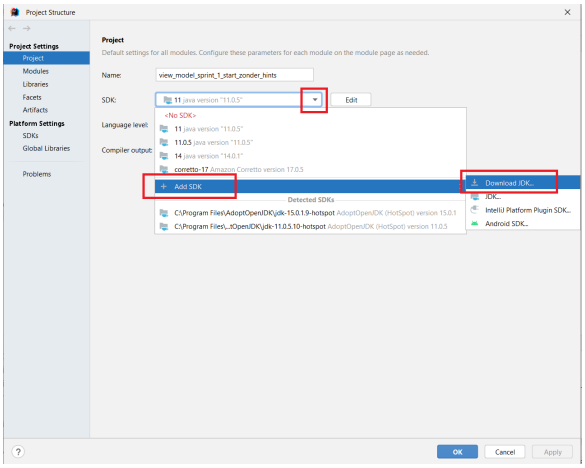



**Figuur 5:** projectstructuur

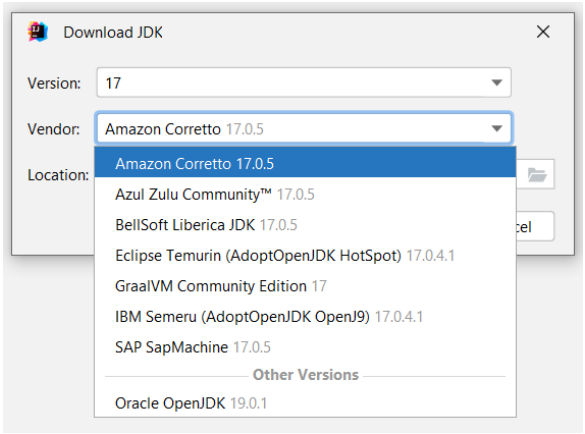



**Figuur 6:** run Viewer

4. Het project werd gemaakt met SDK 17; misschien heb je die nog niet staan. Als er problemen opduiken in verband hiermee, dan open je *File* > *Project Structure* > *Project Settings* > *Project*, en kies je bij het invulveld *SDK* voor *Add SDK* > *Download JDK* (Figuur 7). Stel dan in dat je versie 17 wil, en kies er een uit het lijstje (bvb. *Eclipse Temurin*) (Figuur 8).




**Figuur 7:** add SDK

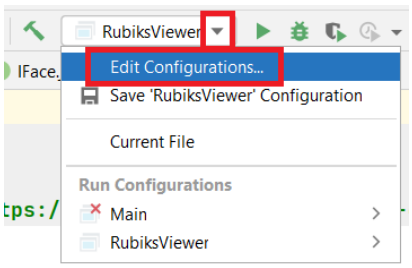



**Figuur 8:** download JDK

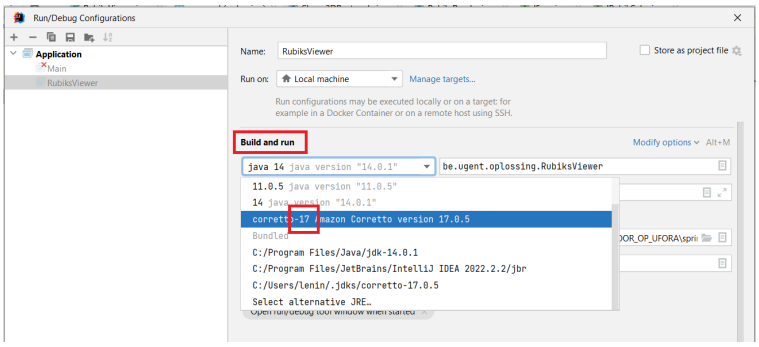
Als het downloaden gelukt is, probeer je het programma opnieuw te runnen.

Krijg je dan nog foutmeldingen zoals `InvalidModuleDescriptorException`, dan neem je nog volgende stappen.

Open het venster *Run/Debug Configurations* aan de hand van de dropdownlist links van de groene run-pijl (bovenaan in Intellij, Figuur 9). Kies daar bij *Build and run* dezelfde JDK-versie (17) als hierboven (Figuur 10).

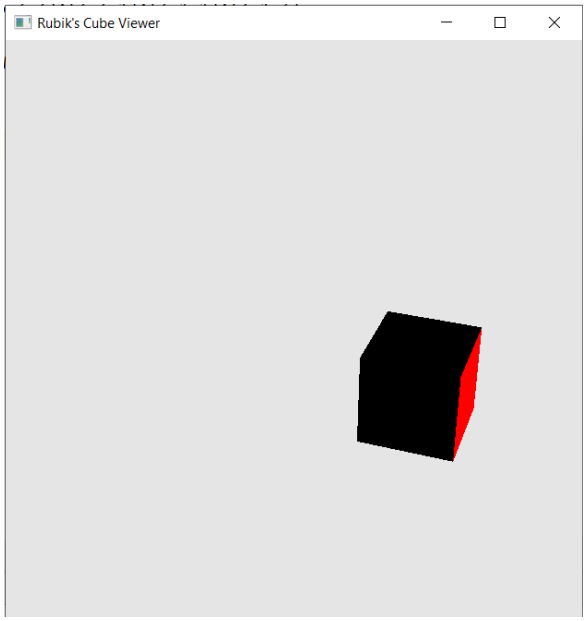



**Figuur 9:** edit configurations

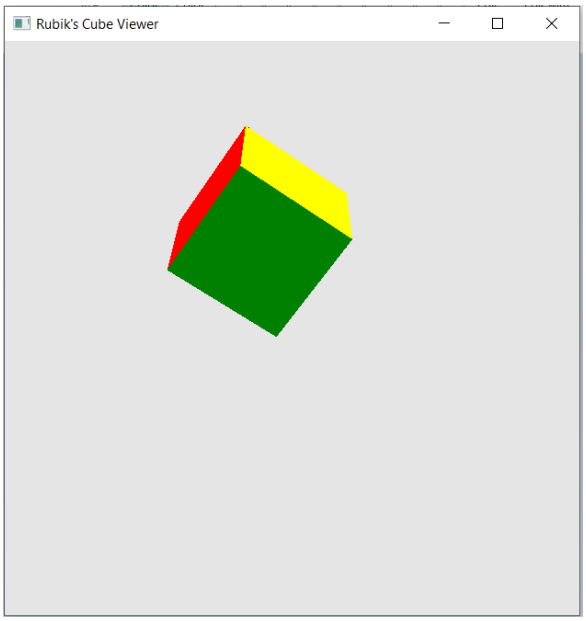



**Figuur 10:** gebruik 17

5. Het programma uit de `RubiksViewer`-klasse geeft een *Graphical User Interface* (GUI) die het beeld uit figuur 11 toont. Neem dit kubusje met de muis vast, en roteer het rond de oorsprong (die ergens buiten het blokje ligt, in het midden van het venster) - dan krijg je figuur 12.




**Figuur 11:** een blokje...




**Figuur 12:** ... geroteerd

### Eigen inbreng: declaratie van klassen

De code die jullie moeten schrijven, komt in de map *model*. Daar staan al drie bestanden klaar.

1. Laat het programma in het bestand `Main.java` lopen. Dat is een 'gewoon' programma dat output produceert in de console. Vul daar code aan, die de klassen die je zelf schrijft uittest. Denk eraan: stapsgewijze testen, niet verder werken op code waarvan je niet zeker weet dat ze correct werkt.

2. In het bestand `IRubikCube` staat een *interface*. Het verschil tussen een klasse en een interface komt nog aan bod in de theorieles. Hier volstaat het om onderstaande stappen te volgen.

   (a) Maak een nieuw bestand aan, waarin jouw klasse die een Rubikskubus voorstelt in komt te staan. (Kies de klassenaam goed; twee letters *volstaan niet*.)

   (b) De hoofding van de klasse vul je aan op deze manier:

   ```
   public class KubusVanVeelKubusjes implements IRubikCube{
   ```

   Met het toevoegen van de code `implements IRubikCube` beloof je letterlijk *deze klasse implementeert de methode(s) uit de interface IRubikCube*.

   (c) Uiteraard moet die belofte ook gehouden worden. Nu is dat nog niet het geval, en komen er dus een rood lijntje onder de klassehoofding. Zet de cursor op die rode lijn, klik *alt-Enter* en kies voor *Implement methods* (gevolgd door *OK*). Nu zal de methode `getAllFaces()` die jullie als modelteam schrijven, naadloos (want zonder tikfouten) aansluiten op de code die het showteam verwacht.

   (d) De implementatie van de methode zal nog niet voor onmiddellijk zijn: allicht moet er eerst een constructor geschreven worden voor de Rubikskubus, zodanig dat er wel degelijk $6 \cdot 3^3$ vlakjes *zijn*. Anders kan je ze ook niet teruggeven in de methode `getAllFaces()`.

3. In het bestand `IFace` staat de interface waaraan jouw klasse die een vlakje van een kubusje voorstelt, moet voldoen. Daar zie je ook dat de view gebruik maakt van de reeds bestaande javafx-klasse `Point3D`. Zoek deze klasse op (google/ecosia) voor meer informatie. Of nog handiger: *Ctrl-klik* op de klassenaam, en je krijgt de broncode van deze klasse te zien.

4. Kleuren worden ook "uit de kast getrokken": zoek informatie over de javafx-klasse `Color` (bekijk eventueel de imports in het bestand `IFace.java` om te weten in welke bibliotheek je moet zoeken).

5. Bekijk nogmaals de powerpoint waarin sprint 0 overlopen werd. Er is nog minstens één klasse die nuttig kan zijn: eentje voor hoekpunten. Nu zijn er verschillende keuzes.

   (a) Je gebruikt meteen de klasse `Point3D`. We hebben immers gezien dat een hoekpunt niet per se meer informatie moet hebben dan zijn 3D-coördinaten.

   (b) Je schrijft een eigen klasse `Hoekpunt`, waarin je een instantievariabele van type `Point3D` bewaart. Het mogelijke voordeel (dat later pas nuttig zal blijken): je kan methode toevoegen om punten bij elkaar op te tellen, of om een punt te roteren rond een bepaalde as en over een bepaalde hoek. (Het roteren wordt nog niet gecodeerd in sprint 1!) Voeg wel een getter toe voor de instantievariabele van type `Point3D`.

   (c) Je leidt een eigen klasse `Hoekpunt` af van de klasse `Point3D`. Maar omdat de onderwerpen *overerving* en *afleiden van klassen* nog niet aan bod kwamen, is deze optie niet erg behulpzaam.

   Valt de keuze op optie (b), dan zijn er allicht aanpassingen nodig (nl. omzettingen van `Hoekpunt`-objecten naar `Point3D`-objecten) om de methode `Point3D[] getFaceCorners()` uit de interface `IFace` te implementeren in de eigen klasse die een vlakje voorstelt. So be it.

Valt de keuze op optie (a), dan zal er een oplossing moeten komen om rotaties uit te voeren op `Point3D`-objecten, maar ook dat is mogelijk. Daarover meer in sprint 2.

6. Heb je al een of enkele klassen geschreven? Test ze tijdig uit in het hoofdprogramma in `Main.java`.

### Eigen inbreng: initialisatie van objecten

Om de rubikskubus (en dus ook alle vlakjes en hoeken) juist te initialiseren, zullen de gepaste kleuren en coördinaten opgesomd moeten worden. Hieronder een mogelijkheid.

```
Point3D centrum = new Point3D(2,2,2);
Kubusje kub = new Kubusje(centrum,Color.RED,Color.GREEN,Color.YELLOW,Color.BLACK,Color.BLACK,Color.BL
kubusjes.add(kub);
centrum = new Point3D(2,0,0);
kub = new Kubusje(centrum,Color.RED,Color.BLACK,Color.BLACK,Color.BLACK,Color.BLACK,Color.BLACK);
kubusjes.add(kub);
centrum = new Point3D(2,2,2);
kub = new Kubusje(centrum,Color.RED,Color.YELLOW,Color.BLACK,Color.BLACK,Color.BLACK,Color.BLACK);
kubusjes.add(kub);
```

Dat is de code om 3 kubusjes aan te maken; er werden dus $24 \cdot 3$ regels weggelaten. Alles tesamen levert dit veel duplicated code. Een tip om dit leesbaarder te maken: toch alle code voor één kubusje op één lijn zetten, zo staan gelijkaardige gegevens recht onder elkaar en is opzoeken van inconsistenties makkelijker (zie blz 4).

Het opstellen van de 27 *juiste* regels code is nu nog monnikenwerk - als we niet opletten. Lees je de juiste coördinaten en kleuren van een tekening af? Dat geeft gegarandeerd hoofdpijn. Kan er niet gesteund worden op het feit dat de coördinaten allemaal in $\{-2, 0, 2\}$ liggen? Dan zou de opsomming vervangen kunnen worden door een driedubbele for-lus (met `i` $\in \{-2, 0, 2\}$).

Maar hoe moet het dan met de bijhorende kleuren? Twee mogelijkheden:

- je blijft bij het gedacht van een driedubbele for-lus en bekijkt of het programma met berekeningen kan beslissen welke kleur de zes vlakjes krijgen (allicht in steeds dezelfde volgorde).
- je stapt af van het gedacht van een driedubbele for-lus met berekeningen, en vult zelf op gestructureerde manier een (excel-)bestandje in. Daar is dan per regel alle informatie van één vlakje te vinden (centrum en kleuren in welbepaalde volgorde). Dat bestand wordt dan (ok, ook met een for-lus) ingelezen.

In ieder geval raden we jullie aan om hier resoluut voor *werkvereenvoudiging* te gaan! En misschien wat documentatie voor de lezer van jullie code...

### Model en view laten samenwerken

Tot nu toe werd het model op zichzelf beschouwd (en getest via output op console of debugger).

Nu is het tijd om de link tussen model en view te maken. Haal daarvoor in `RubiksViewer.java` regels 57 en 58 uit commentaar. Vul ook aan: maak een nieuw object van jouw rubikskubus-klasse. (Vervang het type van de variabele `cube` niet; dat komt wel goed!)

Nu kan je ook visueel controleren of de initialisatie van de Rubikskubus gelukt is.