

JavaFX — GUIs in Java

Prof. K. Coolsaet

Universiteit Gent
2018

Nota's bij de cursus *JavaFX - GUIs in Java* ingericht door het Instituut voor permanente vorming in de wetenschappen van Universiteit Gent ([IPVW](#)) in de reeks *Bijscholing Java voor leerkrachten informatica in het secundair onderwijs*. Deze nota's worden ook gebruikt in het vak *Objectgericht Programmeren*, Bachelor Informatica (1e jaar), Universiteit Gent.

Bijkomend materiaal (broncodefragmenten, links, ...) is ter beschikking op <http://inigem.ugent.be/javafx.html>.



1. Inleiding

JavaFX is een onderdeel van Java waarmee je op een eenvoudige manier grafische gebruikersinterfaces kunt bouwen (*Graphical User Interfaces* — GUIs), enigszins vergelijkbaar met hoe je dit in Visual Basic doet: je bouwt een formulier op met een grafische editor, en je verbindt die met een Java-klasse die dan reageert op gebeurtenissen (het indrukken van een knop, het selecteren van een optie, enz.). Merk op dat JavaFX, net zoals Java, platformonafhankelijk is, en dus werkt op zowel Windows, Linux als macOS.

JavaFX werd reeds in 2008 geïntroduceerd. De eerste versie gebruikte echter een afzonderlijke programmeertaal, naast Java, en is wellicht daarom nooit echt populair geworden. In 2011 besliste Oracle om in de plaats een Java-bibliotheek te ontwerpen met dezelfde functionaliteit, *JavaFX 2* genaamd, en deze bibliotheek werd ook meteen naar voor geschoven als de opvolger van *Swing*, het pakket dat al meer dan 10 jaar de standaard was voor GUI-programma's in Java.

Met de introductie van versie 8 van Java is ook de nummering van de JavaFX-versies veranderd: JavaFX heet nu officieel *JavaFX 8*. We zullen het in het vervolg echter steeds gewoon over JavaFX hebben.

Deze nota's zijn slechts een eerste introductie tot JavaFX. Het pakket biedt nog heel wat meer mogelijkheden, alhoewel dit in sommige gevallen wel een meer diepgaande kennis van Java en objectgericht programmeren vereist dan wat er nodig is voor deze tekst.

1.1 Installatie van de software

Sinds 2013 maakt JavaFX integraal deel uit van Java, althans van de versie die door Oracle wordt verdeeld. (Het is nog even wachten op een *open source* versie, als onderdeel van OpenJDK.) In deze tekst gebruiken we Java 8 maar alle voorbeelden zouden ook moeten werken met recentere versies van Java.

De editor die je gebruikt om formulieren te ontwerpen, de *JavaFX Scene Builder* van Oracle, staat los van de standaard Java-distributie en moet afzonderlijk worden geïnstalleerd.

Tenslotte raden we je ook aan om een professionele programmeeromgeving te gebruiken, zoals *Netbeans*, *IntelliJ IDEA* of *Eclipse*. Dit is niet strikt noodzakelijk

maar maakt het werk wel een heel stuk gemakkelijker. Voor deze nota's gebruiken we IntelliJ IDEA 17.0 (Community Edition).

Je kan Java ophalen vanaf deze URL:

<http://www.oracle.com/technetwork/java/javase/downloads/>.

Gebruik de JAVA-knop (links — Java Platform (JDK)). Het heeft geen zin om ook *NetBeans* te downloaden.

Je moet ook de nieuwste versie van *Scene Builder* installeren. Je vindt die hier:

<http://gluonhq.com/open-source/scene-builder/>.

IntelliJ IDEA vind je op

<http://www.jetbrains.com/idea/download/>.

Klik op DOWNLOAD COMMUNITY (de *Ultimate*-editie is betalend).

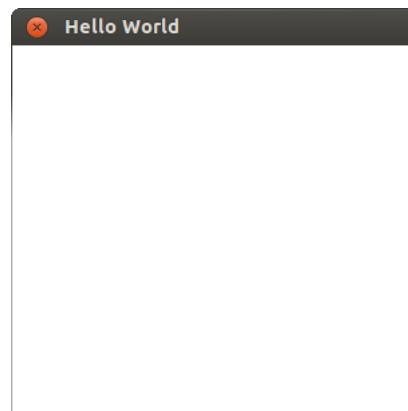
1.2 Een eerste programma

Om de installatie te testen en als eerste voorbeeld van een JavaFX-programma, voer je de volgende stappen uit.

1. Start IntelliJ IDEA op en kies *Create New Project*.
2. Selecteer links *Java FX* en rechts *JavaFX Application*. Zorg dat er bovenaan als *Project SDK* versie '1.8' of '1.9' geselecteerd staat. (De eerste keer is dit misschien nog blanco. Klik dan eerst *New | JDK | Choose*.)
3. Druk *Next* en tik 'Hello' in als naam voor het project in het dialoogvenster dat nu verschijnt. (Laat de andere instellingen voorlopig met rust.)
4. Druk op *Finish*.
5. Sluit het '*Tip of the day*'-venster.
6. Voer het project uit door op het groene driehoekje te klikken in de werk balk rechts bovenaan (*Run Main*) of door op de toetsen SHIFT-F10 te drukken.

Als resultaat verschijnt er een venster zonder inhoud¹. Dit is een zeer elementair programma: het enige wat je kan doen is het venster groter of kleiner maken en het dichtklikken om het programma te beëindigen.

Om dit eenvoudige resultaat te bekomen, heeft IDEA echter al heel wat werk verricht. Een JavaFX-programma bestaat immers uit verschillende onderdelen die nauw met elkaar samenwerken. We bespreken dit in meer detail hieronder.

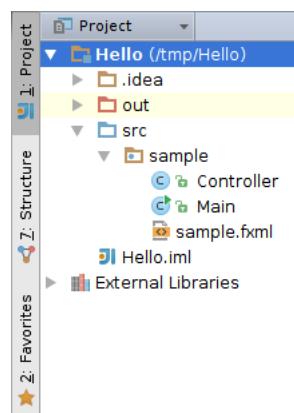


1.3 Bestanden in een IDEA/JavaFX-project

1. Klik in het *Project*-paneel de nodige mapjes open om een overzichtelijk beeld te bekomen zoals hieronder afgebeeld.

De meeste mappen en bestanden zijn bedoeld voor intern gebruik van IDEA. Voor ons zijn enkel de bestanden belangrijk in de deelmap *sample* van de map *src*: twee Java-bestanden en één XML-bestand:

- *Main.java*, het *hoofdprogramma*,
- *sample.fxml*, beschrijft hoe het venster eruit ziet,
- *Controller.java*, definiëert hoe het venster op gebeurtenissen reageert (toetsinvoer, muisklikken, ...).



Het *Project*-paneel toont niet de namen van de Java-bestanden, maar die van de overeenkomstige klassen. En *sample* is eigenlijk geen mapje, maar een *package*. Packages groeperen Java-klassen en zijn onontbeerlijk bij grotere programma's.

De twee Java-klassen in dit voorbeeld behoren beiden tot de package *sample* en hebben dus als 'volledige' naam *sample.Main* en *sample.Controller*. Bovenaan de broncode van die klassen vind je dan ook telkens de volgende **package**-opdracht:

¹De schermafdrukken in deze tekst werden gemaakt op een Linux-toestel. Onder Windows of macOS zullen er kleine verschillen te zien zijn.

```
package sample;
```

Het bestand *sample.fxml* is opgesteld in een XML-formaat dat speciaal is ontworpen voor JavaFX (en *FXML* heet). Dit bestand bevat een beschrijving van het venster in een formaat dat door het programma kan begrepen worden. Ook al kan je een dergelijk FXML-bestand rechtstreeks lezen en editeren, is het in de eerste plaats bedoeld om gebruikt te worden met de *Scene Builder*, de ‘formuliereneditor’ van JavaFX die we zullen bekijken in paragraaf §1.5.

1.4 Een kleine uitbreiding

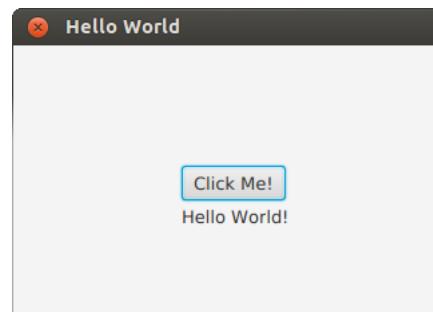
Om ons verhaal toch een klein beetje interessant te maken, bekijken we een kleine uitbreiding van het *Hello*-programma.

1. Download het bestand *Hello2.zip* van de cursuswebsite (zie blz. 1) en pak het uit (bijvoorbeeld onder *Mijn Documenten* in een Windows-omgeving). Als resultaat krijg je een map met de naam *Hello2*.
2. Sluit het huidige project in IDEA. (*File | Close project*)
3. Klik op *Open* en selecteer de map *Hello2* die je daarnet hebt uitgepakt.
4. Voer het project uit, zoals eerder uitgelegd.

Het venster dat nu getoond wordt, heeft een knop *Click Me!*. Wanneer je op die knop klikt, verschijnt eronder de tekst ‘Hello World!’.

Dit project heeft dezelfde bestanden als het vorige, zoals je ziet in het *Project*-venster, alleen zijn er enkele aanpassingen gebeurd aan de inhoud.

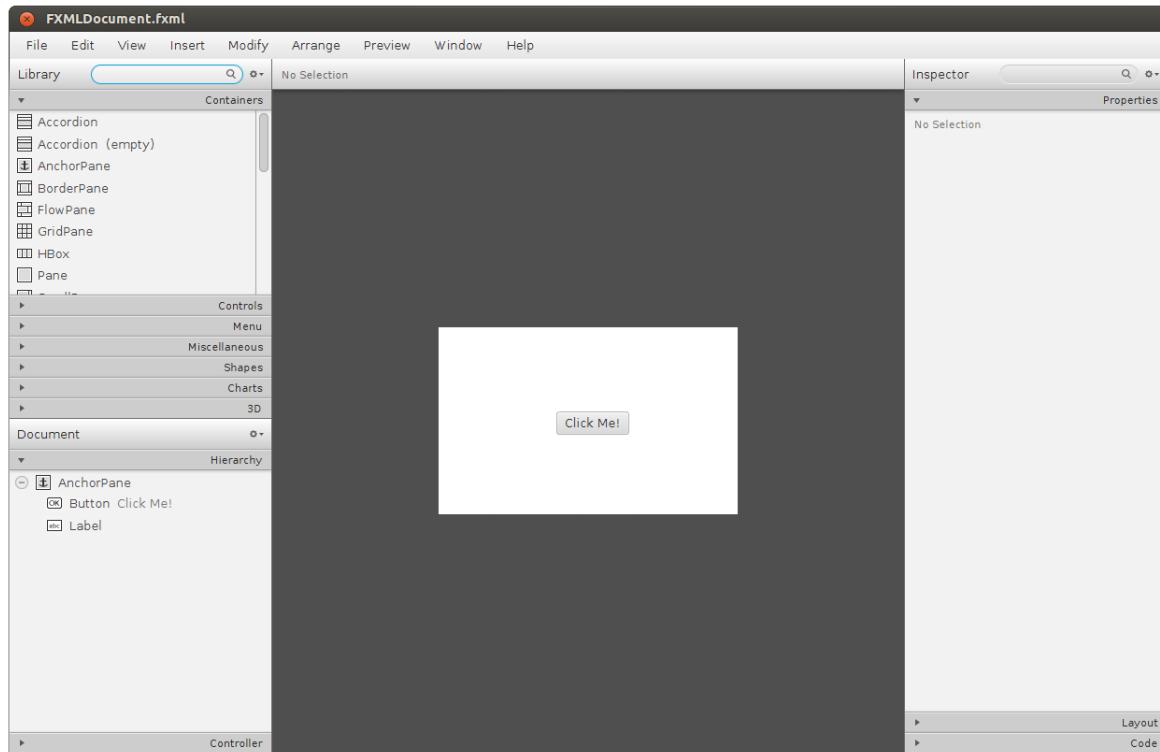
We zullen nu eerst het FXML-bestand bekijken met behulp van de *Scene Builder*.



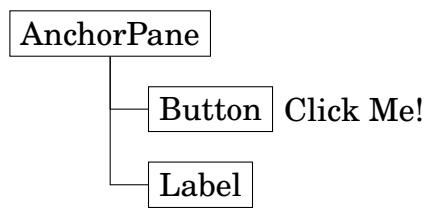
1.5 De Scene Builder

1. Zoek het FXML-bestand *sample.fxml* in het *Project*-paneel van IDEA.
2. Klik met de rechtermuisknop op de naam en selecteer *Open In Scene-Builder* helemaal onderaan in het contextmenu.

Dit opent de *Scene Builder* in een venster zoals hieronder afgebeeld, met in het midden een afbeelding van het bewuste formulier².



Het is ook mogelijk om het *Scene Builder*-venster te openen *binnenin* IDEA door op *sample.fxml* te dubbelklikken en onderaan het paneel waarin het bestand is geopend de *Scene Builder*-tab te selecteren. De mogelijkheden van deze interne *Scene Builder* zijn echter beperkt, en in deze tekst zullen we daarom bij voorkeur de eerste methode hanteren.



Links onder wordt de inhoud van het formulier weergegeven in een boomdiagram. Dit geeft aan dat het formulier uit drie *componenten* bestaat:

- Een ‘ankerpaneel’ (*AnchorPane*), met daarop
- een knop (*Button*), en
- een label (*Label*)

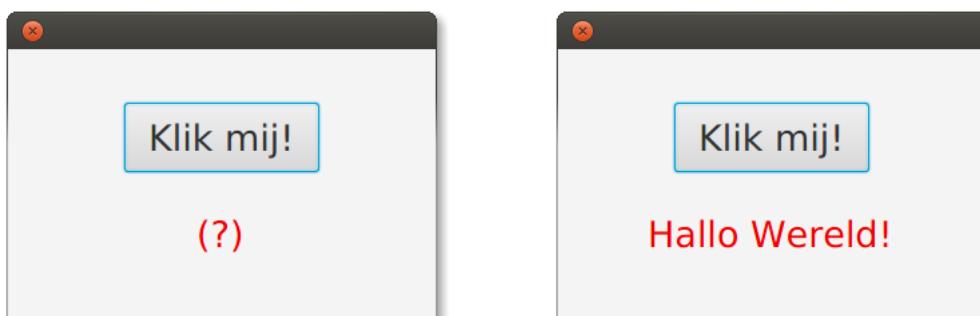
De tekst van het label is oorspronkelijk blanco en wordt later in het programma gewijzigd naar “Hello World!”.

²De eerste keer dat je dit doet, moet je misschien aangeven waar de *Scene Builder* zich bevindt. Aanvaard gewoon de suggestie van IDEA.

1. Klik op het label linksonder in het diagram.
 2. Verander de tekst van het label in “(empty)”.
- Er zijn verschillende manieren om dit te doen: dit kan in het *Text*-veld dat bovenaan rechts is verschenen, of rechtstreeks op de plaats van het label op het formulier, of ook nog onmiddellijk naast het woord ‘Label’ linksonder waarop je zojuist geklikt hebt.
3. Bewaar je veranderingen (*File | Save* of **CTRL-S**).
 4. Voer het programma opnieuw uit. Je merkt hopelijk het verschil op met de eerste versie.

De *Scene Builder* biedt heel wat mogelijkheden om de visuele eigenschappen van componenten aan te passen. Experimenteren is de boodschap!

1. Maak een veiligheidskopie van het FXML-bestand.
2. Pas het formulier zodanig aan dat het eruitziet zoals hieronder links. Het lettertype is 27 pixels hoog en het label onderaan heeft een rode kleur. Merk op dat het label en de knop mooi onder elkaar zijn gecentreerd.
3. Voer het programma uit met dit nieuwe formulier en klik op de knop. Ziet het er precies uit zoals in de rechter afbeelding hieronder?



De tekst “Hallo Wereld!” moet je aanpassen in de programmacode van de klasse *Controller*. Om deze tekst mooi gecentreerd te krijgen, maak je (in de *Scene Builder*) het label (de labelachtergrond) even breed als het ganse venster en verander je zijn *Alignment* van *CENTER LEFT* naar *CENTER*.

1.6 Container-componenten

De *AnchorPane*-component is een zogenaamde *container*³. Een dergelijke component dient om andere componenten te bevatten. Deze worden dan de *kinderen* van de container genoemd.

Er bestaan verschillende soorten container-componenten. Links boven in het *Scene Builder*-venster vind je alvast een hele lijst. Elk type container schikt zijn kinderen op een andere manier. Bij een ankerpaneel onthoudt JavaFX de posities (de coördinaten) waarop je de componenten hebt geplaatst. Bij andere containers (een *VBox*) bijvoorbeeld, bepaalt hij die posities zelf.

Het verschil tussen twee soorten containers is vaak het duidelijkst zichtbaar wanneer je (*at runtime*) het programmavvenster groter of kleiner maakt.

1. Herstel het FXML-bestand vanuit de veiligheidskopie die je hebt gemaakt.
2. Voer het programma uit en klik op de knop.
3. Neem de rechterbenedenhoek van het venster vast met de muis en maak het venster groter en kleiner, breder en smaller.
4. Wat gebeurt er met de knop en het label?

We proberen hetzelfde uit met een ander type container.

1. Gebruik de *Scene Builder* om het ankerpaneel te veranderen in een *VBox* maar zonder de kindercomponenten te veranderen. De gemakkelijkste manier om dit te doen is wellicht om het ankerpaneel eerste te ‘wikkelen’ in een *VBox* (kies *Wrap in ...* in het contextmenu) en het daarna ‘uit te wikkelen’ (*Unwrap*).
2. Selecteer de *VBox*-component en kies in het rechterpaneel (onder *Properties*) voor een gecentreerde uitlijning (*Alignment*). Dit zorgt ervoor dat de kindercomponenten netjes in het midden blijven van het venster.
3. Selecteer rechts het *Layout*-paneel. Stel een marge in voor de *VBox* van 10 pixels bovenaan, onderaan, links en rechts (*Padding*).
4. Kies een spatiëring van 10 pixels tussen de kindercomponenten (*Spacing*).
5. Bewaar het bestand, en voer het programma uit.
6. Wat gebeurt er nu met de kindercomponenten wanneer je het programmavvenster van grootte verandert?

³Componenten zoals labels en knoppen die geen containers zijn, worden ook wel *controls* genoemd. Omdat de woorden *control* en *controller* in de informatica echter zoveel verschillende betekenissen kunnen hebben, zullen we die in deze tekst vermijden.

1.7 Het hoofdprogramma

De klasse *Main* (of dus eigenlijk *sample.Main*) is het feitelijke hoofdprogramma. Het plaatst het formulier in een venster en toont dit op het scherm.

```
public class Main extends Application {  
  
    @Override  
    public void start(Stage primaryStage) throws Exception{  
        Parent root = FXMLLoader.load(  
            getClass().getResource("sample.fxml"))  
        );  
        primaryStage.setTitle("Hello World");  
        primaryStage.setScene(new Scene(root));  
        primaryStage.show();  
    }  
  
    public static void main(String[] args) {  
        launch(args);  
    }  
}
```

We hebben in onze afdruk de **package**- en **import**-opdrachten weggelaten, maar die moeten er wel staan. Gelukkig helpt IDEA ons hierbij:

1. Open (in IDEA) de broncode van de klasse *Main*. (Dubbelklik op de klassennaam in het *Project*-paneel.)
2. Verwijder de **import**-regels. Enkele namen komen in het rood te staan.
3. Er verschijnen blauwe ‘tekstballonnen’ om aan te geven dat er *imports* ontbreken. Druk telkens op ALT-ENTER. Soms krijg je de keuze tussen verschillende opties. Kies dan telkens deze die met ‘javafx.’ begint. (Als de ballonnen niet verschijnen, ga dan met de tekst-cursor op een rode klassennaam staan en druk ALT-ENTER.)

Een hoofdprogrammaklasse voor JavaFX moet aan enkele voorwaarden voldoen:

- Ze moet een uitbreiding zijn van de JavaFX-klasse *Application* (of voluit, *javafx.application.Application*).
- Ze moet de methode *start* implementeren (zie verder).
- Ze moet een *main*-methode bevatten van een bijzondere vorm.

```

public class Main extends Application {

    public void start(Stage stage) throws Exception {
        ...
    }

    public static void main(String[] args) {
        launch(args);
    }
}

```

Als een programma aan deze voorwaarden voldoet, zal JavaFX bij het opstarten een nieuw (programma)venster aanmaken en de methode *start* die je hebt geprogrammeerd oproepen om dit venster te ‘bevolken’.

De JavaFX-bibliotheek gebruikt een terminologie uit de theaterwereld. Een venster wordt een ‘toneel’ genoemd (*stage*) en de inhoud van het venster, het formulier, heet een ‘scène’. Vandaar ook de naam *Scene Builder*. Je vindt de namen *scene* en *stage* ook terug in de implementatie van *start*:

```

public void start(Stage primaryStage) throws Exception{
    Parent root = FXMLLoader.load(
        getClass().getResource("sample.fxml")
    );
    primaryStage.setTitle("Hello World");
    primaryStage.setScene(new Scene(root));
    primaryStage.show();
}

```

Met deze achtergrondinformatie is het niet zo moeilijk te begrijpen wat er hierboven gebeurt, alleen de eerste opdracht vraagt wat bijkomende uitleg. Hier wordt het FXML-bestand ingeladen en omgezet naar een component *root* waarmee de scène uiteindelijk wordt opgebouwd.

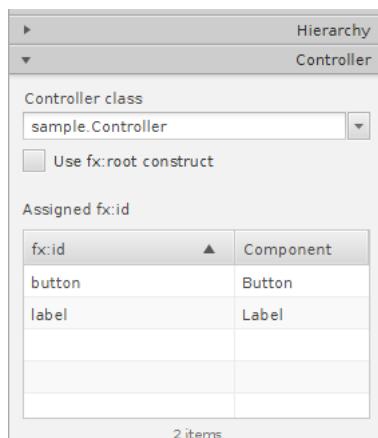
De uitdrukking ‘*getClass().getResource("sample.fxml")*’ geeft aan dat het FXML-bestand ‘*sample.fxml*’ heet en moet gezocht worden op dezelfde plaats waar ook deze klasse (*Main*) zich bevindt⁴, t.t.z. in de package *sample*.

⁴De feitelijke details zijn iets ingewikkelder maar hoeven we hier niet te kennen. We zullen het hoofdprogramma trouwens in de praktijk ook steeds door IDEA zelf laten aanmaken.

1.8 De partnerklasse

Bij elk formulier, of anders gezegd, bij elk FXML-bestand, hoort ook een bijzondere Java-klasse die we zijn *partnerklasse* of *controller-klasse* noemen. Hierin gebeurt het echte programmeerwerk.

De *Scene Builder* heeft helemaal linksonder een *Controller*-paneel dat je kan uitklappen. Daar vind je de naam van de partnerklasse terug (en kan je die eventueel veranderen).



Onze partnerklasse heet dus *sample.Controller*. Voor dit voorbeeld zullen we de broncode van deze klasse niet in detail bespreken maar je mag ze gerust eens bekijken.

In hetzelfde paneel zie je ook dat er aan twee componenten een 'FX-identificatie' is toegewezen (een *fx:id*). Deze dient om de componenten uit het formulier te verbinden met overeenkomstige variabelen in de partnerklasse.

Met elke component komt er immers een Java-object overeen van een bepaalde klasse. Een knop wordt dus een *Button*-object en een label een object van de klasse *Label*. De FX-identificaties zorgen ervoor dat JavaFX aan de gelijknamige velden van de partnerklasse (dus *label* en *button*) het overeenkomstige object automatisch toewijst.

Elke klasse heeft verschillende methoden waarmee je het uitzicht en het gedrag van de betreffende component kunt wijzigen. Het opschrift van een label wijzig je bijvoorbeeld op de volgende manier:

```
label.setText("Hallo Wereld!");
```

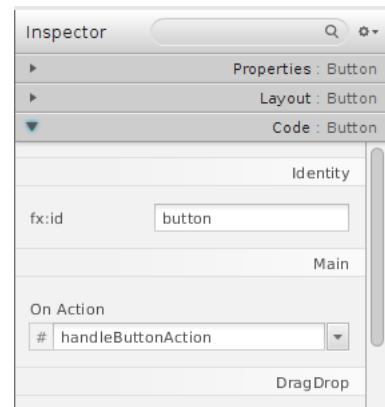
Zoek de elektronische documentatie op van de klasse *Label* en van haar methode *setText*. Er zijn verschillende manieren waarop je dit kunt doen.

1. Gebruik Google. Let wel op dat je de klasse *Label* uit JavaFX bekijkt (dus *javafx.scene.control.Label*) en niet een andere klasse met dezelfde (verkorte) naam. Is het ook de juiste versie van JavaFX?
2. Open de partnerklasse, selecteer in de broncode de klassennaam *Label* en druk CTRL-Q.

1.9 Actiegebeurtenissen

Wanneer je op de knop drukt, wordt een methode opgeroepen uit de partnerklasse. In ons voorbeeld heet die methode `handleButtonAction`. Technisch gezien vangt deze methode de zogenaamde *actiegebeurtenis* van de knop op.

Dat precies deze methode wordt opgeroepen bij het indrukken van de knop, staat aangegeven in het FXML-bestand. Je kan die naam ook terugvinden (en eventueel aanpassen) in de *Scene Builder*.



1. Selecteer de knop in de *Scene Builder*.
2. Kies aan de rechterkant van het *Scene Builder*-venster het paneel *Code*
3. Bij *On Action* zie je de naam staan van de methode die de actiegebeurtenis van de knop zal behandelen.
4. Zoek die methode op in de broncode van de partnerklasse.

Merk op dat het *Code*-paneel dus ook de plaats is waar je de FX-identificatie van een component kan opgeven.

2. Eerste stappen

Als eerste voorbeeldtoepassing die we volledig zelf ontwikkelen, schrijven we een JavaFX-programma waarmee je een gebruikersnummer kunt aanmaken aan de hand van de naam van een persoon.



Vul je een voornaam en een familienaam in in de overeenkomstige velden en druk je op de knop, dan verschijnt er daarnaast een string die kan dienen als een gebruikersnummer voor een computertoepassing.

Het nieuwe gebruikersnummer bestaat uit de eerste letter van de voornaam, en maximum zeven letters van de familienaam — zonder spaties en in kleine letters. De toepassing mag niet vastlopen als één van de velden, of beide velden leeg zijn.

2.1 Starten

De meest gemakkelijke manier om een nieuw project te starten, is te vertrekken van wat IDEA voor ons maakt en enkele bestandsnamen aan te passen.

1. Maak een nieuw JavaFX-project aan in IDEA met als naam ‘User’.
2. Ga naar het *Project*-venster en verander daar de naam van de package van *sample* naar *user*: selecteer het mapje en druk SHIFT-F6.
3. Verander op dezelfde manier de naam van het FXML-bestand in *User.fxml* en de namen van de Java-klassen *Main* en *Controller* naar respectievelijk *User* en *UserController*.
4. Open de broncode van *User* (dubbelklik op de klassennaam in het *Project*-venster) en verander de verwijzing naar het bestand *sample.fxml* naar *User.fxml*¹.

¹Al deze naamsverandering zijn niet strikt noodzakelijk, maar we zullen in deze tekst steeds deze standaardconventies te volgen.

Dit geeft ons een minimaal project. Wanneer je dit uitvoert, verschijnt er een leeg venster zonder inhoud.

We zouden ons formulier het liefst in een ankerpaneel plaatsen. IDEA geeft ons echter een roosterpaneel (*GridPane*).

5. Open het *User.fxml* in de *Scene Builder*. Vervang het roosterpaneel door een ankerpaneel van 140 pixels hoog en 390 pixels breed. Vergeet niet het resultaat te bewaren.(Dit kan bijvoorbeeld met CTRL-S.)

Omdat we expliciet de afmetingen opgeven voor het ankerpaneel, hoeven we dat niet meer voor de scène te doen. Vervang daarom in het hoofdprogramma *User*, de lijn

```
primaryStage.setScene(new Scene(root, 300, 275));
```

door

```
primaryStage.setScene(new Scene(root));
```

Voor de rest kunnen we dit hoofdprogramma min of meer ongewijzigd laten. We plaatsen enkel nog een titel in het programmavenster (“Nieuw gebruikersnummer”) en zorgen ervoor dat het venster niet van grootte kan veranderd worden².

6. Zoek in de documentatie van de klasse *Stage* met welke setter-methode je een titel aan een venster kunt hechten.
7. En hoe je een venster kunt onveranderlijk maken van grootte.
8. Pas het hoofdprogramma *User* op die manier aan.
9. (Voer het programma uit, kijk of het compileert en of het doet wat je verwacht.)

2.2 Het formulier

Als volgende stap ontwerpen we het formulier. De componenten die we op dit paneel plaatsen, zijn van het type *Button*, *Label* en *TextField*.

²Een ankerpaneel ziet er niet altijd even mooi uit wanneer de gebruiker het groter of kleiner maakt — athans niet zonder bijzondere voorzorgen.

1. Open het FXML-bestand met de *Scene Builder* en plaats de nodige componenten erop zodat het resultaat er ongeveer uitziet als in het voorbeeld.
2. Geef de componenten die het nodig hebben een FX-identificatie (*fx:id*)
3. Kies *Preview | Show Preview in Window*, of tik CTRL-P om ondertussen het eindresultaat te bekijken zonder daarom telkens het programma te moeten uitvoeren.

Het is inderdaad goed om op voorhand te bedenken welke componenten er een FX-identificatie nodig hebben. In eerste benadering zijn dit enkel de componenten waarop het programma inwerkt, dus bijvoorbeeld niet het ‘Voornaam’- of ‘Familienaam’-label. Maar ook de knop hoeft geen *fx:id*, omdat de connectie tussen knop en partnerklasse op een andere manier tot stand komt.

Wij kiezen voor de volgende FX-identificaties

<i>fx:id</i>	Component
<i>voornaam</i>	<i>TextField</i>
<i>naam</i>	<i>TextField</i>
<i>resultaat</i>	<i>Label</i>

1. Geef in de *Scene Builder* aan dat de methode *maakNummer* moet opgeroepen worden wanneer de knop wordt ingedrukt.

2.3 De partnerklasse

Bij het invullen van de FX-identificaties heb je wellicht gemerkt dat de *Scene Builder* waarschuwingen geeft (gele driehoekjes bovenaan). Dit is omdat er in de partnerklasse nog geen overeenkomstige velden zijn gedeclareerd. (Je krijgt trouwens ook foutmeldingen wanneer je het programma in IDEA probeert uit te voeren.)

Een minimale partnerklasse die wél kan uitgevoerd worden, ziet er zo uit³:

³Vanaf nu zullen we de **package**- en **import**-opdrachten niet langer afdrukken in de voorbeelden. Vergeet ze dus niet bovenaan toe te voegen.

```
public class UserController {  
  
    public TextField voornaam;  
  
    public TextField naam;  
  
    public Label resultaat;  
  
    public void maakNummer () {  
        ...  
    }  
}
```

Met elke FX-identificatie komt er een veld overeen met dezelfde naam en met als type de overeenkomstige componentklasse. Daarnaast moeten we ook de methode *maakNummer* implementeren die we bij de actiegebeurtenis van de knop hebben geregistreerd. Let goed op dat je hier geen tikfouten maakt. Die worden niet altijd meteen door JavaFX opgemerkt en geven soms aanleiding tot bugs die heel moeilijk op te sporen zijn.

De drie velden moeten **public** zijn, wat eigenlijk indruist tegen een goede softwareontwikkelingspraktijk. Het is ook mogelijk om deze velden **private** te laten, maar dan hebben we zogenaamde Java-annotaties nodig om die toch te kunnen gebruiken, een complicatie die we hier willen vermijden.

1. Neem bovenstaande code over in de broncode van de klasse *UserController*.
2. (Voer het programma uit.)

Dit programma werkt wel, maar er gebeurt niets wanneer we op de knop drukken. Hiervoor moeten we nog de methode *maakNummer* implementeren.

De bedoeling is dat deze methode de strings gebruikt die zich in het *voornaam*- en het *naam*-tekstveld bevinden, deze omzet naar een nieuwe string en dan als opschrift instelt voor het label *resultaat*. Ter informatie: om de inhoud van een tekstveld (of een label) te bekomen, gebruik je *getText()* op het overeenkomstige object. Om de inhoud van een label (of een tekstveld) aan te passen, gebruik je *setText(...)* .

Omwille van een duidelijk ontwerp, schrijf je best eerst een afzonderlijke functie *nummerUitNaam*:

```
private String nummerUitNaam (String voornaam, String naam) {  
    ...  
}
```

1. Voeg de methode *nummerUitNaam* toe aan *UserController* en implementeer die methode.
2. Vervolledig de methode *maakNummer*.

Hiermee is het programma in principe af.

De methode *maakNummer* heb je wellicht als volgt geïmplementeerd:

```
public void maakNummer() {  
    resultaat.setText(  
        nummerUitNaam(voornaam.getText(), naam.getText())  
    );  
}
```

Voor *nummerUitNaam* zijn er verschillende oplossingen mogelijk, ook afhankelijk van wat je wil doen wanneer de voornaam of de familienaam leeg zijn. Zorg er in ieder geval voor dat het programma in die gevallen niet de mist ingaat. Wij kozen voor onderstaande oplossing:

```
private String nummerUitNaam(String voornaam, String naam) {  
    voornaam = voornaam.trim();  
    if (! voornaam.isEmpty()) {  
        naam = voornaam.substring(0,1) + naam;  
    }  
    naam = naam.replaceAll ("\s", " ").toLowerCase();  
    if (naam.isEmpty()) {  
        return "(leeg)";  
    } else if (naam.length() < 8) {  
        return naam;  
    } else {  
        return naam.substring(0, 8);  
    }  
}
```

De `replaceAll`-methode van `String` vervangt een bepaald deelpatroon (reguliere uitdrukking) in een string overal door een andere waarde. Het patroon '\s' dat we in dit voorbeeld gebruiken, correspondeert met elke vorm van spatiëring (blanco's, tabs, ...) en wordt hier telkens vervangen door een lege string⁴.

2.4 Foutbehandeling

Eigenlijk vinden we een lege string als voornaam of familienaam geen zo'n goed idee en willen we dit als een foute invoer beschouwen en de gebruiker hiervan op de hoogte brengen. De schermafdruk hieronder toont hoe we dit willen doen.



We plaatsen hiertoe een label met een foutbericht onder elk tekstveld. In de plaats van de tekst van dit label telkens aan te passen wanneer nodig, zullen we een vaste tekst behouden en het label zelf zichtbaar of onzichtbaar maken.

1. Open het FXML-bestand in de *Scene Builder*.
2. Voeg twee labels toe onder elk tekstveld, in een kleiner lettertype, in het rood en met de foutbericht als tekst.
3. Geef ze als `fx:id` resp. `voornaamError` en `naamError`.
4. Maak beide nieuwe labels onzichtbaar door in het *Properties*-paneel de eigenschap `Visible` uit te vinken.
5. Voeg de twee nieuwe labels ook toe aan de partnerklasse.

In eerste instantie zullen we de foutberichten slechts tonen op het moment dat er op de knop wordt gedrukt.

⁴Dit is een algemene Java-methode die niet specifiek is voor JavaFX. Vind je dit te ingewikkeld, dan kan je ook gemakkelijk zelf een **for**-lus schrijven met hetzelfde effect.

```

public void maakNummer() {
    checkVoornaam();
    checkNaam();
    if (voornaamError.isVisible() || naamError.isVisible()) {
        resultaat.setText(" ");
    } else {
        resultaat.setText(
            nummerUitNaam(voornaam.getText(), naam.getText())
        );
    }
}

```

Je gebruikt de methode *isVisible* om te kijken of een component al dan niet zichtbaar is en de methode *setVisible* (...) om de zichtbaarheid van een component in te stellen.

De methodes *checkNaam* en *checkVoornaam* bekijken het *Naam*- of *Voornaam*-label en maken het corresponderende ‘*Error*’-label zichtbaar als en slechts als de tekst leeg is.

1. Neem bovenstaande code voor *maakNummer* over in de partnerklasse
2. Schrijf een implementatie voor *checkVoornaam* en *checkNaam*.

De implementaties van *checkVoornaam* en van *checkNaam* lijken heel sterk op elkaar. Vanuit het standpunt van goed software-ontwerp is het daarom beter om één methode te schrijven, met twee parameters, die in beide gevallen kan gebruikt worden⁵:

```

private void checkLabel (TextField field, Label errorLabel) {
    if (field.getText().trim().isEmpty()) {
        errorLabel.setVisible (true);
    } else {
        errorLabel.setVisible (false);
    }
}

```

of nog beter:

⁵Later zal echter blijken dat we toch nog een afzonderlijke *checkVoornaam* en *checkNaam* nodig hebben...

```
private void checkLabel (TextField field, Label errorLabel) {  
    errorLabel.setVisible(field.getText().trim().isEmpty());  
}
```

Het zou mooi zijn wanneer de fout al wordt opgemerkt op het moment dat het lege veld wordt ingetikt (of correcter — uitgeveegd), maar het zou ons te ver leiden om dit helemaal naar wens op te lossen. We kiezen daarom voor een middenweg: we voeren de controles ook door wanneer de gebruiker de ENTER-toets indrukt in een tekstveld. Dit veroorzaakt namelijk een actiegebeurtenis die we kunnen opvangen.

1. Verbind, in de *Scene Builder*, de actiegebeurtenis van het eerste tekstveld met de methode *checkVoornaam* uit de partnerklasse.
2. Doe hetzelfde voor *checkNaam*.

Opgave 2.1 Pas het bovenstaande programma aan zodanig dat ook het label aan de linkerkant van een tekstveld rood kleurt wanneer het veld leeg is. Tips: de kleur van een label wordt bepaald door zijn ‘text fill’, en *Color* is een deelklasse van *Paint*.

(In paragraaf §5.4 komen we terug op deze versie van het programma.)

3. Nog meer componenten

Tot nog toe hebben we enkel labels, knoppen en tekstvelden gebruikt. Dit zijn zeer eenvoudige componenten met een eenvoudige Java-interface die heel goed door de *Scene Builder* worden ondersteund.

In dit hoofdstukje bekijken we enkele elementen van JavaFX die wat meer detailwerk vragen.

3.1 De choice-box

We beginnen met een toepassing waarmee een willekeurig wachtwoord kan worden aangemaakt met een druk op de knop. De gebruiker heeft de keuze tussen wachtwoorden van 8, 12 of 16 tekens.



De meest linkse component in dit venster is een *choice-box*. Hiermee maakt de gebruiker een keuze tussen een (klein) aantal voorgeprogrammeerde waarden.

1. Maak een nieuwe JavaFX-toepassing aan met de naam *WWGen*, FXML-bestand *WWGen.fxml* en partnerklasse *WWGenController*.
2. Gebruik de *Scene Builder* om een venster aan te maken zoals in het voorbeeld hierboven. Van links naar rechts bevat dit venster een *ChoiceBox*, een *Button* en een *Label*.
3. Implementeer de partnerklasse zodat een druk op de knop een wachtwoord op het scherm toont bestaande uit willekeurige kleine letters, hoofdletters en/of cijfers.

Opgelet! De *Scene Builder* laat niet toe om de ‘inhoud’ van de choice-box op te geven (de waarden 8, 12 en 16). Laat de choice-box daarom voorlopig leeg en genereer altijd wachtwoorden van lengte 12.

Je kan bijvoorbeeld onderstaande methode gebruiken om een willekeurig wachtwoord aan te maken van een opgegeven lengte:

```

private static final Random RG = new Random();

private static final String LETTERS =
    "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";

private static String randomWachtwoord (int lengte) {
    String str = "";
    for (int i = 0; i < lengte; i++) {
        str += LETTERS.charAt(RG.nextInt(LETTERS.length()));
    }
    return str;
}

```

Met wat we in de vorige hoofdstukken geleerd hebben, is het vrij eenvoudig om deze eerste versie van het programma te bouwen. Het label moet een *fx:id* krijgen (bijv. *resultaat*) en aan de knop moet een methode gehecht worden die zijn actiegebeurtenis opvangt (bijv. *maakWachtwoord*).

```

public Label resultaat;

public void maakWachtwoord () {
    resultaat.setText (randomWachtwoord(12));
}

```

Later zullen we de 12 vervangen door ‘*choiceBox.getValue()*’.

Er is echter al meteen een kleine complicatie. De choice-box die we in dit voorbeeld gebruiken, heeft *gehele getallen* als waarden, maar in een andere toepassing ontmoet je misschien een choice-box die je een *string* laat kiezen, of zelfs een afbeelding.

Voor de *Scene Builder* maakt dit niets uit, maar voor de Java-compiler is dit wel belangrijk. Hij moet namelijk weten dat *getValue()* van het type *Integer* is, en niet van het type *String*. We specifiëren dit in de declaratie van de variabele *choiceBox*:

```
public ChoiceBox<Integer> choiceBox;
```

Let op het type tussen de scheve haken.

3.2 De methode *initialize*

Er zijn twee manieren om de waarden van de choice-box in te stellen, ofwel programmatorisch (als onderdeel van de partnerklasse), ofwel door het FXML-bestand aan te passen.

In het eerste geval voegen we onderstaande code toe aan de partnerklasse:

```
public void initialize () {  
    choiceBox.getItems().add (8);  
    choiceBox.getItems().add (12);  
    choiceBox.getItems().add (16);  
}
```

We gebruiken hierbij het volgende kenmerk van de *FXML-loader*¹: heeft de partnerklasse een methode met de naam *initialize*, dan zal de FXML-loader deze methode automatisch één keer uitvoeren onmiddellijk nadat hij alle variabelen heeft ingevuld die overeenkomen met een FX-identificatie².

1. Voeg bovenstaande code toe aan de partnerklasse van het project.
2. Pas *maakWachtwoord* aan zodat het de waarde gebruikt uit de choice-box, in plaats van 12.
3. Wat ontbreekt er nog om het programma helemaal bruikbaar te maken?

Een choice-box houdt zijn waarden intern bij in de vorm van een lijst die je kan opvragen met de methode *getItems()*. Deze methode heeft een object terug van het type *List<Integer>* waar we op de gebruikelijke manier met *add* elementen kunnen aan toevoegen.

De drie opdrachten in het vorige codefragment kan je trouwens ook afkorten tot één enkele opdracht:

```
choiceBox.getItems().addAll (8, 12, 16);
```

Hoe krijgen we het nu voor elkaar om de choice-box zodanig in te stellen dat hij

¹De FXML-loader is het onderdeel van JavaFX dat een venster opbouwt aan de hand van het FXML-bestand en de componenten ervan met de partnerklasse verbindt. We hebben de naam *FXMLLoader* al in het hoofdprogramma ontmoet.

²Verwar dit niet met de initialisatie die gebeurt in een eventuele constructor van de partnerklasse: de constructor wordt uitgevoerd vóórdat deze variabelen zijn ingevuld.

bij het opstarten meteen de waarde 12 toont? Hiervoor gebruik je de methode `setValue` van de klasse `ChoiceBox` — tijdens `initialize`.

```
public void initialize() {
    choiceBox.getItems().addAll(8, 12, 16);
    choiceBox.setValue(12);
}
```

Misschien heb je gemerkt dat het *Properties*-paneel van de *Scene Builder* voor een choice-box een *Value*-veld bevat? Als je daar 12 invult, dan heeft dit echter niet het verwachte effect. Dit werkt enkel maar bij een choice-box met strings, t.t.z., van het type `ChoiceBox<String>`.

3.3 Het FXML-bestand

Een tweede manier om de waarden van een choice-box reeds op voorhand in te vullen, is door de inhoud van het FXML-bestand handmatig aan te passen. Het wordt dus stilaan tijd dat we een dergelijk bestand eens van dichterbij bekijken.

1. Sluit de *Scene Builder* als die nog open staat.
2. Dubbelklik in het *Project*-paneel van IDEA op de naam van het FXML-bestand. Dit opent het bestand in tekstvorm en laat je toe om er dingen aan te veranderen.

Het FXML-bestand begint met een hoofding van de volgende vorm:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.text.*?>
<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
```

Een lijn zoals de eerste zal je in de meeste XML-bestanden ontmoeten en dient om aan te geven dat het bestand wel degelijk in een XML-formaat is opgesteld en welke codering er gebruikt is. De andere lijnen zijn typisch voor FXML. Ze spelen dezelfde rol als **import**-opdrachten in Java. In het hoofdgedeelte van het bestand zullen namen van klassen voorkomen, zoals `AnchorPane` en `ChoiceBox`,

en de <?import .. ?>-lijnen geven aan in welke packages deze klassen moeten gezocht worden³.

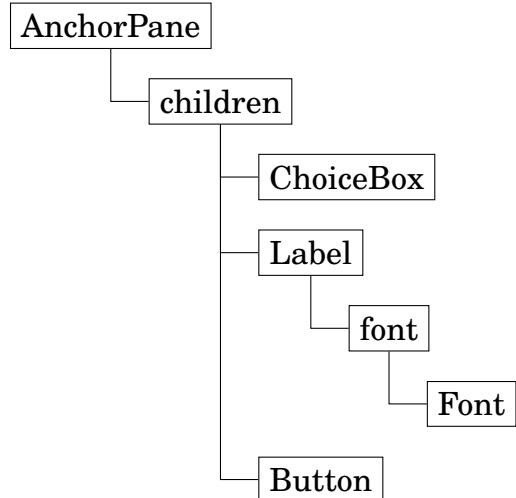
Het hoofdgedeelte van het FXML-bestand bevat een beschrijving van de scène die moet worden opgebouwd:

```
<AnchorPane xmlns:fx="http://javafx.com/fxml/1"
             id="AnchorPane" prefHeight="55.0" prefWidth="396.0"
             xmlns="http://javafx.com/javafx/8"
             fx:controller="wwgen.WWGenController">
    <children>
        <ChoiceBox fx:id="choiceBox" layoutX="27.0" layoutY="14.0"
                   prefHeight="25.0" prefWidth="64.0"/>
        <Label fx:id="resultaat"
               layoutX="148.0" layoutY="16.0"
               prefHeight="21.0" prefWidth="210.0"
               text="(leeg)">
            <font>
                <Font size="18.0" />
            </font>
        </Label>
        <Button layoutX="102.0" layoutY="14.0" mnemonicParsing="false"
               onAction="#maakWachtwoord" text="OK" />
    </children>
</AnchorPane>
```

Het is gemakkelijker om de structuur van dit bestand te begrijpen als je de attributen (zoals `layoutX="27.0"`) voorlopig even wegdenkt.

De structuur komt goed overeen met wat je ook in het *Hierarchy*-paneel van de *Scene Builder* te zien krijgt — met uitzondering van de twee lettertype-elementen en het `children`-element⁴.

De attributen worden gebruikt om allerechte eigenschappen van de componenten in te stellen. Je herkent vanzelfsprekend het `text`-attribuut van het *Label*- en het *Button*-element, en het `onAction`-attribuut van het *Button*-element. Merk op dat gebeurtenisverwerkende methoden een # voor hun naam krijgen.



³In het vervolg zullen we deze hoofding niet meer afdrukken.

⁴Het `children`-element mag trouwens weggelaten worden in de nieuwste versies van JavaFX.

Er zijn ook enkele attributen die een bijzondere notatie gebruiken. Het attribuut *fx:id* behoeft geen uitleg. Het attribuut *fx:controller* bij het buitenste element bevat de (volledige) naam van de partnerklasse.

1. Open het FXML-bestand in de editor.
2. Selecteer het *Structure*-paneel in de linker marge van het IDEA-venster, of gebruik de toetscombinatie ALT-7.

Dit is een andere manier om de structuur van een FXML-bestand te bekijken.

Het is niet nodig om alle details van het FXML-formaat te kennen of te begrijpen. In de praktijk zal men immers nooit een FXML-bestand helemaal met de hand opstellen, maar vertrekt men steeds van wat de *Scene Builder* heeft geproduceerd.

Anderzijds is het soms handig om kleine veranderingen rechtstreeks in het bestand aan te passen⁵, bijvoorbeeld om een tikfout uit de naam van een gebeurtenismethode te halen, of om snel eens een groter lettertype voor een label uit te proberen.

Voer de volgende wijzigingen uit aan het formulier, zonder hiervoor de *Scene Builder* te gebruiken:

1. Verander het opschrift van het label in “(blanco)”
2. Gebruik een groter lettertype voor de knop (18 pixels)
3. Verander de posities van de label en de knop zodat het geheel er weer mooi uitziet

3.4 Items in het FXML-bestand

Door het FXML-bestand rechtstreeks te editeren, kan je vaak meer bereiken dan alleen met de *Scene Builder*:

```
<ChoiceBox fx:id="choiceBox" layoutX="27.0" layoutY="14.0">
    prefHeight="25.0" prefWidth="64.0">
        <Integer fx:value="8" />
        <Integer fx:value="12" />
        <Integer fx:value="16" />
    </ChoiceBox>
```

Bovenstaande XML-code voegt drie gehele waarden toe aan een *ChoiceBox*.

⁵Het opstarten van de *Scene Builder* kan soms toch wel wat lang duren...

1. Vervang het *ChoiceBox*-element in het FXML-bestand door het bovenstaande fragment
2. Verwijder de *initialize*-procedure uit de partnerklasse
3. Voer het programma uit. Wat ontbreekt er nog?

(Veranderingen die je op die manier aan een FXML-bestand maakt, worden gelukkig niet terug weggegooid als je hetzelfde bestand later opnieuw met de *Scene Builder* bewerkt.)

Net zoals bij onze eerste poging uit paragraaf §3.2, krijgt de choice-box geen initiële waarde.

Je kan die zoals voorheen instellen in een *initialize*-methode, maar het is ook mogelijk om dit rechtstreeks in het FXML-bestand te doen. Dit is waarvoor het `<value>`-element dient in onderstaande code.

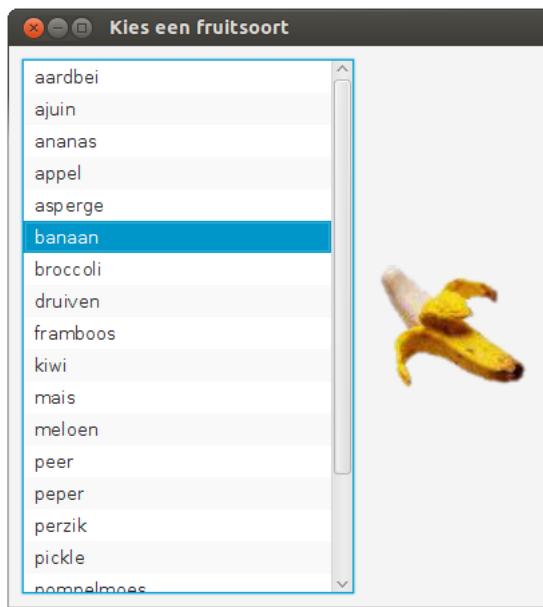
```
<ChoiceBox fx:id="choiceBox" layoutX="27.0" layoutY="14.0"
           prefHeight="25.0" prefWidth="64.0">
    <items>
        <FXCollections fx:factory="observableArrayList">
            <Integer fx:value="8" />
            <Integer fx:value="12" />
            <Integer fx:value="16" />
        </FXCollections>
    </items>
    <value>
        <Integer fx:value="12" />
    </value>
</ChoiceBox>
```

Jammer genoeg is nu ook de notatie voor het instellen van de choice-box-keuzes een stuk ingewikkelder geworden: we moeten nu ook nog een `<items>`-element introduceren met daarbinnen een `<FXCollections>`-element.

Dit is in principe de standaardmanier om de elementen van een choice-box te initialiseren. De techniek die we de eerste keer hebben gebruikt, is hiervan een afkorting die echter alleen toegelaten is wanneer er geen andere eigenschappen van de choice-box moeten worden ingesteld.

3.5 Selecteren uit een keuzelijst

In het volgende voorbeeld kan je een fruitsoort selecteren uit een keuzelijst⁶ om een prent ervan af te beelden, zoals in de volgende schermafbeelding:



1. Creëer een JavaFX-toepassing met de naam *Fruit* (in een package *fruit*).
2. Maak in dit project een nieuwe (sub)package *fruit.images* aan: klik in het *Project*-venster rechts op *fruit* en kies *New | Package*.
3. Kopieer de fruitafbeeldingen naar deze nieuwe package. Dit doe je buiten IDEA. De corresponderende map vind je terug bij de IDEA-projecten onder *Fruit\src\fruit\images*.
4. Maak in de *Scene Builder* een venster dat bestaat uit een *HBox* met daarin een *ListView* (links) en een *ImageView* (rechts). Gebruik *lijst* en *imageView* als *fx:id*. De lijstcomponent bevat strings, dus het corresponderende type is *ListView<String>*.
5. Gebruik *unknown.png* als initiële afbeelding van de *ImageView*. Stel een marge en een tussenruimte in van 10 pixels voor de *HBox*.
6. Vul de *ListView* op met de fruitnamen. Je kan dit doen in een *initialize*-methode of je kan rechtstreeks het FXML-bestand aanpassen. Beide methoden zijn even goed.

⁶Het woord ‘lijst’ heeft in Java heel wat betekenissen. We zullen in deze tekst de term ‘lijst’ meestal alleen gebruiken wanneer we een object van het type *List<...>* bedoelen. Een (visuele) component waarin gegevens worden opgeliist, een object van het type *ListView<...>*, noemen we hier een ‘keuzelijst’ of een ‘lijstcomponent’.

Werk je met een *initialize*-methode, dan kan je de volgende code gebruiken:

```
private final String[] IMAGE NAMES = {  
    "aardbei",  
    "ajuin",  
    ...  
    "watermeloen",  
    "wortel"  
};  
  
public void initialize () {  
    lijst.getItems().addAll(IMAGE NAMES);  
}
```

Geef je de fruitnamen op in het FXML-bestand, dan doe je dit zo:

```
<ListView fx:id="lijst">  
    <String fx:value="aardbei"/>  
    <String fx:value="ajuin"/>  
    ...  
    <String fx:value="watermeloen"/>  
    <String fx:value="wortel"/>  
</ListView>
```

Er zijn twee nieuwe aspecten aan dit project. Hoe reageer je op selectie van een element in een lijst-component en hoe ga je om met afbeeldingsbestanden?

3.6 Selecteren uit een keuzelijst

In tegenstelling tot een *ChoiceBox* reageert een *ListView* niet op actiegebeurtenissen. (Je hebt er misschien al vergeefs naar gezocht in de *Scene Builder*.) In de plaats daarvan moeten we reageren op veranderingen in de selectie. Dit doen we door een zogenaamde *luisteraar* te registeren bij de *selectedItem*-eigenschap van het *selectiemodel* van de lijstcomponent.

Een bijkomend woordje uitleg is hier wellicht gepast.

Een *ListView* is een behoorlijk ingewikkelde component. Om de programmacode overzichtelijk te houden, hebben de ontwerpers daarom bepaalde functionalitei-

ten van een *ListView*-object gedelegeerd naar andere objecten. Het *selectieModel* is daar een voorbeeld van.

Dit is een object (in dit geval van het type *SelectionModel<String>*) dat ten behoeve van de *ListView* alle informatie bijhoudt over welke elementen er al dan niet zijn geselecteerd. Als we dus de geselecteerde elementen willen opvragen of bekijken, dan doen we dit niet rechtstreeks bij de lijstcomponent, maar wel bij zijn selectiemodel.

```
lijst.getSelectionModel()...
```

Het selectiemodel heeft een aantal *eigenschappen*. In JavaFX wordt het woord ‘eigenschap’ (Engels: *property*) gebruikt voor velden van een object waarvan je de waarde kunt opvragen en veranderen met gepaste *getters* en *setters* en waar je bovendien kunt naar ‘luisteren’ (zie verder).

Voor ons zijn twee eigenschappen van het selectiemodel interessant: de eigenschap *selectedItem* en de eigenschap *selectedIndex*. De eerste eigenschap geeft aan welk element er op een gegeven moment is geselecteerd (in dit geval, een string), de tweede eigenschap komt overeen met het *volgnummer* van dit element (in Java beginnen we te tellen vanaf 0).

Dus, om de naam van het geselecteerde fruit op te vragen, schrijven we

```
lijst.getSelectionModel().getSelectedItem()
```

3.7 Luisteraars

Het belangrijkste kenmerk van een JavaFX-eigenschap is dat je kunt reageren op veranderingen in de waarde van die eigenschap. Dit doe je door er een bijzonder object bij te registreren. Een dergelijk object dat reageert op veranderingen van een eigenschap, noemen we een *luisteraar* (Engels: *listener*). In ons programma reageren we op veranderingen in de selectie. Daarom registreren we een luisterraar bij de *selectedItem*-eigenschap van het selectiemodel.

```
lijst.getSelectionModel().selectedItemProperty().addListener(..);
```

Opgelet! We hebben hier de methode *selectedItemProperty* gebruikt, die de ei-

genschap zelf teruggeeft, en niet *getSelectedItem*, die enkel de waarde van de eigenschap retourneert.

Om als luisterraar dienst te doen, moet een object aan enkele voorwaarden voldoen — het moet een bepaalde Java-interface implementeren. Vóór de introductie van zogenaamde ‘lambda’s’ in versie 8 van Java, vroeg het heel wat tikwerk om zelfs de meest eenvoudige luisterraar te implementeren. Nu kan dat echter heel kort:

```
public void initialize () {
    lijst.getSelectionModel().selectedItemProperty().addListener(
        ob -> kiesFruit()           // luisterraar in Java 8
    );
    lijst.getItems().addAll(IMAGE NAMES);
}

private void kiesFruit() {
    ...
}
```

(Registratie van luisterraars gebeurt best in de *initialize*-methode.)

Het pijltje (\rightarrow) is een nieuwe operator uit Java 8 waarmee een *anoniem functie-object* kan worden aangemaakt. Zo’n object kan één specifieke functie uitvoeren — bijvoorbeeld, reageren op een gebeurtenis.

Links van het pijltje staan de *parameters* van die functie — in dit geval een parameter *ob* die meer informatie bevat over het object dat de gebeurtenis heeft gegenereerd (je mag de naam van deze parameter zelf kiezen). In de praktijk heb je deze informatie niet vaak nodig, zoals ook in dit geval.

Rechts van het pijltje staat het *corpus* van de functie, m.a.w., de opdracht die moet worden uitgevoerd wanneer de functie wordt opgeroepen. In ons voorbeeld wordt er gewoon een andere methode opgeroepen. Java laat ook toe om hier meer ingewikkelde code te plaatsen, maar het is een goede gewoonte om dit niet te doen, omwille van de leesbaarheid.

Het effect van dit alles is dat de methode *kiesFruit* wordt opgeroepen telkens wanneer de selectie verandert.

1. Voeg bovenstaande code toe aan de partnerklasse.
2. Implementeer de methode *kiesFruit* zodanig dat ze de naam van het fruit afdrukt. (Hoe je de prent verandert, leer je straks.)

Voor de volledigheid drukken we ook nog eens af hoe dezelfde code er zou uitzien in Java 7:

```
public void initialize () {
    lijst.getSelectionModel().selectedItemProperty().addListener(
        new InvalidationListener() {
            public void invalidated(Observable ob) {
                kiesFruit();
            }
        });
}
```

De Java 8-code is hiervan een exacte afkorting. Merk op dat een luisteraar aan de interface *InvalidationListener* voldoet.

3.8 Afbeeldingen

Afbeeldingen in JavaFX worden geplaatst op een component van het type *ImageView*. De afbeeldingen zelf worden echter voorgesteld als objecten van een klasse *Image*. M.a.w., om een afbeelding op een venster te plaatsen, moet je eerst een *Image*-object aanmaken en dan dit *Image*-object doorgeven aan een *ImageView*-component.

```
imageView.setImage (new Image (...));
```

Je kan deze tweeledige structuur ook herkennen in het FXML-bestand:

```
<ImageView fx:id="imageView" fitWidth="128.0"
           pickOnBounds="true" preserveRatio="true">
    <Image url="@images/unknown.png" />
</ImageView>
```

In de praktijk is een afbeelding bijna altijd afkomstig van een bestand. Om het

corresponderende *Image*-object aan te maken, geef je de naam van dit bestand door aan de constructor⁷.

```
Image image = new Image ("fruit/images/aardbei.png");
```

1. Pas de *initialize*-methode van de partnerklasse aan zodat er bij het opstarten een aardbei wordt getoond in plaats van drie vraagtekens.

Merk op dat we niet alleen de naam van het bestand nodig hebben, maar het ganse pad ernaartoe, inclusief alle packages en deelpackages. Onderdelen van het pad worden gescheiden door gewone schuine strepen (/) en niet door ‘backslashes’ (\), ook in een Windows-omgeving.

Je bezit nu voldoende informatie om het programma volledig af te werken:

1. Pas de methode *kiesFruit* aan zodat de afbeelding van het geselecteerde fruit aan de *ImageView* wordt toegewezen. Bouw de naam van het bestand op aan de hand van het pad van de package, de string die geselecteerd is in de keuzelijst en de extensie “.png”.
2. Voer het programma uit en kies je favoriete fruit.

Opgave 3.1 Implementeer de volgende variant op het fruitprogramma:

- Gebruik *initialize* om de waarden van de keuzelijst in te stellen, en doe dit niet in het FXML-bestand.
- Hou in de partnerklasse een array of lijst bij van alle afbeeldingen — en dus niet van enkel hun namen. Vul deze array of lijst reeds op tijdens *initialize*. Op die manier hoeft een afbeelding die al eerder is getoond niet opnieuw te worden geladen.
- Gebruik de *selectIndex*-eigenschap van het selectiemodel in plaats van *selectedItem*.

⁷In §4.4 ontmoeten we nog een tweede manier om een afbeeldingsbestand te specificeren.

4. Menu's en meer

De schermafbeelding hieronder toont een programma waarmee je afbeeldingen kan bekijken die op je computer zijn opgeslagen. Met dit voorbeeld illustreren we hoe je menu's gebruikt en hoe je een bestandsdialoog oproept.

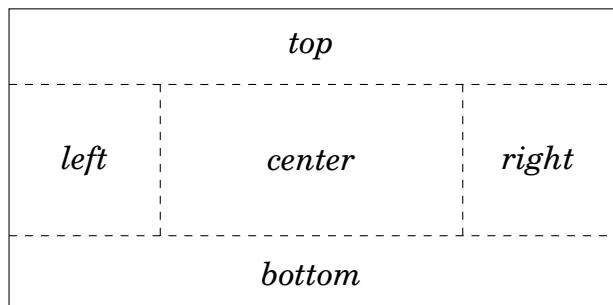


De menubalk in deze toepassing, bevat één menukeuze (*File*) die twee opties biedt als je die openklikt: *Exit* om het programma te beëindigen en *Open...* om een nieuw afbeeldingsbestand te openen. In dit laatste geval toont het systeem een standaarddialoogvenster waarmee je een bestand kan kiezen.

4.1 Het *BorderPane*

1. Start een nieuw JavaFX-project met de naam *Viewer*
2. Maak in de *Scene Builder* een venster aan met een *BorderPane* als container in plaats van een *AnchorPane*.

Een *BorderPane* is een container-component die bijzonder geschikt is om samen met een menubalk te gebruiken. Het paneel is opgedeeld in vijf gebieden — zoals hieronder geschetst.



De gebieden groeien of krimpen automatisch mee met het paneel wanneer het groter of kleiner wordt gemaakt. Het bovenste en onderste deel behouden daarbij hun hoogte, het linker- en rechterdeel behouden hun breedte.

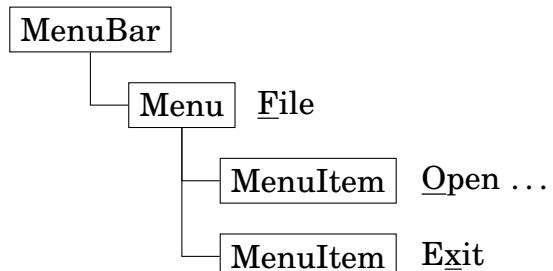
Ons voorbeeld). Op het onderste gebied worden vaak enkele knoppen geplaatst (*OK*, *Cancel*, ...). Het middendeel bevat meestal de essentiële inhoud van het venster. Niet elk van de vijf gebieden hoeft daadwerkelijk ingevuld te zijn.

3. Plaats een menubalk in het bovenste deel van het *BorderPane*. De menubalk (*MenuBar*) vind je bij de *Controls* en niet bij *Menu* zoals je wellicht zou verwachten.
4. Plaats een *ImageView* in het centrale deel. Maak deze component 600 pixels breed en 300 pixels hoog.

4.2 Menu's

De menu's van een toepassing volgen een hiërarchische structuur.

Het programma heeft één enkele menubalk (*MenuBar*), deze balk heeft één of meerdere deelmenu's (*Menu*) en die hebben op hun beurt andere deelmenu's of menu-opties (*MenuItem*).



5. Voeg een menu en twee menu-opties toe aan de menubalk en bezorg ze het juiste opschrift.
6. Stel een sneltoets (*accelerator*) in voor de *Exit*-menu-optie — bijvoorbeeld `CTRL-Q`.

Behalve een sneltoets kan je ook een afkortingstoets (*mnemonic*) hechten aan een menu of een menu-optie. Dit maakt het bijvoorbeeld mogelijk om het *File*-menu te openen door `ALT-F` in te drukken. De afkortingstoetsen worden aangeduid als onderlijnde letters in het menu-opschrift. Deze onderlijning wordt echter enkel zichtbaar zodra (en zolang) je de `ALT`-toets indrukt.

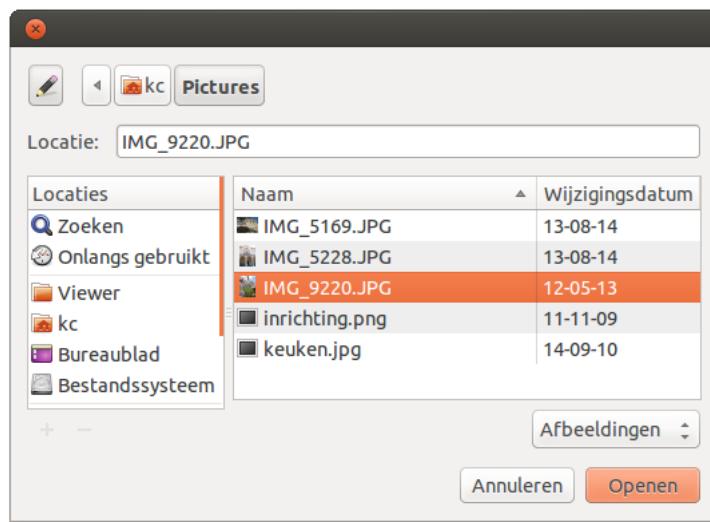
Om een afkortingstoets aan een menu te hechten, schrijf je een *underscore* (_) in het opschrift vlak voor de bewuste letter. Bovendien moet je ook nog de eigenschap *Mnemonic Parsing* aanvinken bij de component.

7. Stel de afkortingsstoeten in voor het menu en de twee menu-opties
8. Voer het programma uit en probeer of alle afkortingsstoeten werken

Het is natuurlijk niet voldoende dat het menu er goed uitziet, we willen ook nog dat er iets gebeurt wanneer we een menu-optie selecteren. Dit is gelukkig heel eenvoudig: menu-opties gedragen zich zoals knoppen en genereren een actiegebeurtenis wanneer ze geactiveerd worden. Je kan er dus een methode aan hechten op dezelfde manier als bij een knop.

1. Zorg ervoor dat de methoden *openFile* en *exitProgram* worden opgeroepen wanneer je de overeenkomstige menu-optie kiest.
2. Voeg dergelijke methoden toe aan de partnerklasse.
3. Laat de implementatie van *openFile* voorlopig leeg, of druk een kort tekstje af zodat je kan zien dat je programma werkt.
4. De opdracht '*Platform.exit()*' beëindigt het programma. Gebruik dit in de implementatie van *exitProgram*.

4.3 De bestandsdialoog¹



¹De afbeelding toont een dialoogvenster zoals het eruit ziet in Ubuntu Linux. Windows zal een ander venster tonen, maar met een gelijkaardige functionaliteit.

Wanneer we de menu-optie ‘*Open ...*’ kiezen, moet er een dialoogvenster verschijnen waarmee we een afbeeldingsbestand selecteren. JavaFX biedt hiervoor de klasse *FileChooser*. Je gebruikt die op de volgende manier:

- Maak een object aan van het type *FileChooser*.
- Roep de methode *showOpenDialog* aan voor dit object. Deze methode geeft een bestand terug in de vorm van een *File*-object, of **null** als er geen selectie is gemaakt.
- Doe iets met dit *File*-object.

(Behalve *showOpenDialog*, die je gebruikt wanneer je een bestand wil inlezen, is er ook *showSaveDialog*, wanneer je een bestand wil opslaan, en *showOpenMultipleDialog*, om in één keer meerdere bestanden te openen. Deze laatste geeft een lijst van *File*-objecten terug, en niet één enkel *File*-object.)

In ons voorbeeld schrijven we dus:

```
private void showImageFromFile (File file) {  
    ...  
}  
  
public void openFile() {  
    FileChooser chooser = new FileChooser();  
    File file = chooser.showOpenDialog(  
        imageView.getScene().getWindow()  
    );  
    if (file != null) {  
        showImageFromFile(file);  
    }  
}
```

Zoals je ziet, neemt de methode *showOpenDialog* één argument, namelijk een venster. Dit venster heet de *ouder* van de bestandsdialoog. Dit heeft het volgende effect:

- Zolang de bestandsdialoog zichtbaar is, aanvaardt de ouder geen invoer.
- Wordt de ouder geminimaliseerd, dan ook de bestandsdialoog.

Je mag lui zijn en **null** opgeven als parameter, maar het is een kleine moeite om het hoofdprogrammavenster op te vragen aan de scène waarin een bepaalde component zich bevindt, zoals we hierboven gedaan hebben.

1. Voeg bovenstaande code toe aan het programma. Gebruik voorlopig nog een lege implementatie voor `showImageFromFile`.

4.4 Werken met bestanden

De naam van de klasse `File` is enigszins misleidend. Een object van die klasse bevat immers alleen maar informatie over de naam van het bestand (en alle bovenliggende mappen) en kan niet gebruikt worden om zijn inhoud op te halen of te wijzigen. Om van een bestand te lezen of ernaar te schrijven, moet je nog andere klassen gebruiken.

Een bijkomend complicatie is dat de klasse `File` in de nieuwere versies van Java werd vervangen door de interface `Path` maar dat dit nog niet tot JavaFX is door gedrongen. Gelukkig heeft de klasse `File` een methode `toPath` waarmee je de omzetting kunt doen naar het nieuwe type.

Het zou ons te ver leiden om de vele verschillende manieren te bespreken waarop er in Java met bestanden kan omgesprongen worden. We zullen ons beperken tot wat we hier nodig hebben, namelijk een `InputStream`. Er bestaat immers een constructor van de klasse `Image` die een dergelijke `InputStream` als argument neemt, en deze willen we gebruiken²:

```
InputStream stream = ...;
Image image = new Image(stream));
```

Om een bestand te gebruiken, moet je het eerst openen, daarna uitlezen en dan weer sluiten.

- Openen doe je door een `InputStream`-object aan te maken met de methode `newInputStream` uit de klasse `Files` — en die neemt een `Path`-object als argument om te weten welk bestand moet geopend worden.
- Uitlezen doet het `Image`-object voor ons automatisch.
- Sluiten kan je doen door de methode `close` van de stream op te roepen, maar zoals je straks zult zien, kan dit ook op een andere manier.

Een bijkomende moeilijkheid bij het werken met bestanden is dat dit ook wel

²De klasse `InputStream` is een (abstracte) klasse met verschillende uitbreidingen. Hier zullen we een stream gebruiken die zijn gegevens uit een bestand haalt, maar er bestaan ook streams voor netwerkverbindingen, of voor gegevens die in het RAM-geheugen zijn opgeslagen.

eens fout kan gaan. Daarom kunnen er uitzonderingen worden opgegooid (van het type *IOException*) en die moeten we opvangen.

Sinds Java 7 bestaat er een constructie (*try with resources of try met bronnen* genaamd) die dit alles in één opdracht combineert:

```
try (InputStream stream = Files.newInputStream(path)) {  
    // doe iets met stream  
} catch (IOException ex) {  
    // vang een eventuele fout op  
}
```

Ook al zie je de methode *close* hier nergens afgedrukt, wordt het bestand toch automatisch afgesloten nadat het **try**-block is uitgevoerd — zelfs wanneer er ondertussen een fout optreedt!

1. Vervolledig het programma door gebruik te maken van wat er hierboven staat uitgelegd.
2. Bekijk je favoriete foto's.

Opgave 4.1 Bij het programma dat we hierboven hebben geschreven, laat de bestandsdialoog toe om eender welk bestand te selecteren. Het is echter ook mogelijk om aan te geven dat een bestandsdialoog alleen bestanden met een bepaalde extensie mag tonen, zoals in de schermafdruk op bladzijde 37.

Zoek in de elektronische documentatie hoe je dit voor elkaar krijgt en pas het programma aan zodat het enkel bestanden toelaat met één van de extensies *.jpg*, *.gif*, *.png*, *.JPG*, *.GIF* of *.PNG*³.

³In Linux en op Max OS/X wordt er bij bestandsnamen een onderscheid gemaakt tussen hoofdletters en kleine letters.

5. JavaFX met stijl

In een moderne softwaretoepassing tracht men verschillende aspecten van een programma netjes in afzonderlijke onderdelen onder te brengen, om het programma gemakkelijker onderhoudbaar en uitbreidbaar te maken en om het werk op een logische manier te kunnen verdelen tussen verschillende ontwikkelaars.

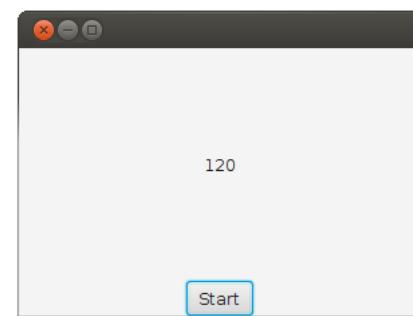
In JavaFX is er daarom een duidelijke scheiding tussen de *weergave* van een programma, zoals vastgelegd in het FXML-bestand, en de *logica*, zoals geprogrammeerd in de partnerklasse. Je kan je inderdaad voorstellen dat de formulieren van een groot project met de *Scene Builder* door één persoon worden gemaakt (die zelfs geen Java hoeft te kennen) en de partnerklassen door iemand anders.

In dit hoofdstuk bekijken we hoe je nog een stap verder kunt gaan en het *ontwerp* van een formulier kunt loskoppelen van het *uitzicht* door gebruik te maken van zogenaamde *stylesheets*. Hiervoor wordt *CSS* gebruikt, dezelfde technologie die ook wordt toegepast bij webpagina's.

5.1 Aftellen tot nul

Als eerste voorbeeld schrijven we een programma dat een teller toont die elke seconde met één vermindert tot hij uiteindelijk 0 bereikt. De teller wordt gestart door op een knop te drukken.

(We concentreren ons eerst op het programmeerwerk en maken pas in een tweede fase het uitzicht iets aantrekkelijker.)



1. Start een nieuw JavaFX-project met de naam *Countdown* in een package *countdown*.
2. Maak in de *Scene Builder* een venster aan met een *BorderPane* als container in plaats van een *AnchorPane*. Maak een paneel van 300 bij 200 pixels.
3. Plaats in het midden een label en onderaan een knop. Maak je geen zorgen over marges en tussenruimtes.
4. Zorg dat de methode *startTimer* van de partnerklasse wordt opgeroepen als je op de knop drukt. Geef het label *label* als *fx:id*.

Aftellen doen we door een luisterraar te registreren bij een zogenaamde *timeline* en aan te geven dat die luisterraar elke seconde moet uitgevoerd worden, in het totaal 120 keer.

De manier waarop je de klasse *Timeline* hiervoor gebruikt, is nogal obscuur en daarom veroorloven we ons om het voorbeeld hieronder af te drukken zonder al te veel uitleg:

```
private Timeline timeline;

public CountdownController() {
    timeline = new Timeline(
        new KeyFrame(Duration.seconds(1.0), e -> puls())
    );
    timeline.setCycleCount(120);
}
```

(We hebben deze code opgenomen in de constructor van *CountDownController*. Als alternatief hadden we dit ook in de *initialize*-methode kunnen doen.)

Je herkent in het tweede argument van *KeyFrame* een functie-object dat de gebeurtenis zal verwerken (zoals in §3.7). De methode *puls*, die we nog moeten schrijven, zal elke seconde één keer opgeroepen worden.

Merk op dat de code hierboven de timeline enkel voorbereidt maar nog niet inschakelt. Dit doe je met de methode *playFromStart*:

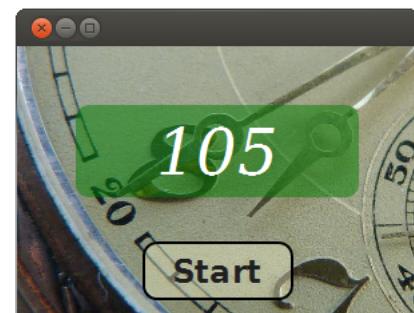
```
public void startTimer () {
    ...
    timeline.playFromStart();
}
```

1. Werk het programma verder af met behulp van de code hierboven. Je zal een gehele tellervariabele nodig hebben om bij te houden wat de huidige waarde is van de teller.
2. Zorg ervoor dat het woord ‘EINDE’ verschijnt, in plaats van 0, wanneer de teller ten einde loopt.
3. Voer het programma uit¹. Wat is het eerste getal dat door de teller wordt aangegeven? En het laatste?

5.2 Stylesheets

Nu het programma werkt, willen we iets doen aan het uitzicht ervan. Rechts zie je identiek hetzelfde programma als in de vorige paragraaf, nadat we aan het venster — of correcter, aan het *BorderPane* — een *stylesheet* hebben gekoppeld.

In dit bestand staan alle stijlveranderingen beschreven: de kleuren van het label en de knop, de afgeronde hoeken, de figuur op de achtergrond, de tussenruimte tussen de knop en de rand van het venster, de lettertypes, ...



Dit bestand is van het type CSS.

1. Klik in het *Project*-paneel rechts op het pakket *countdown* en kies *New | File*.
2. Vul de naam *countdown.css* in (in kleine letters) en druk *Finish*. (De extensie *.css* mag je niet vergeten!)

Dit maakt een nieuw CSS-bestand aan. Om straks te kunnen testen dat de stylesheet haar werk goed doet, plaatsen we nu alvast de volgende code in dit bestand:

```
Label {  
    -fx-text-fill: red;  
}
```

We verbinden dit bestand nu met het formulier:

4. Open het FXML-bestand in de *Scene Builder*.
5. Selecteer het *BorderPane* en zoek in het *Properties*-paneel de eigenschap *Stylesheets*.
6. Klik op de '+'-knop van die eigenschap en selecteer de stylesheet in de bestandsdialoog. Je vindt dit bestand terug bij de IDEA-projecten onder *Countdown\src\countdown\countdown.css*. (Vergeet niet je veranderingen te bewaren.)
7. Voer het programma uit. Welke kleur heeft het label?

¹Als je niet telkens 120 seconden wil wachten terwijl je je programma test, verander dan overal de 120 door een kleiner getal. Beter nog: stel de startwaarde voor door door een constante. Dan hoeft je het programma telkens maar op één plaats aan te passen.

(Misschien kleurde het label ook al rood in de *Scene Builder*. Die doet namelijk zijn uiterste best om ook met stylesheets rekening te houden.)

Een CSS-bestand voor JavaFX volgt dezelfde regels als een CSS-bestand voor het web, alleen zijn de namen van de attributen anders — ze beginnen onder andere allemaal met ‘-fx-’. In een webomgeving zou je ‘color:red’ schrijven om de kleur van de letters te veranderen, bij JavaFX schrijf je in de plaats ‘-fx-text-fill:red’².

Net zoals bij web-CSS bestaat een CSS-bestand in JavaFX uit een aantal blokken die telkens eerst een *locatie* aanduiden en daarna tussen accolades de verschillende *stijlelementen* voor deze locatie.

Als locatie laat JavaFX drie mogelijkheden toe:

- De naam van een Java-klasse, zoals *Label* hierboven. Dit zorgt ervoor dat alle componenten van die klasse de betreffende stijlkenmerken krijgen.
- De naam van een stijlklasse (ook CSS-klasse genaamd), voorafgegaan door een punt. We hadden hier bijvoorbeeld `.label` kunnen gebruiken in plaats van `Label` omdat JavaFX automatisch de stijlklasse *label* toekent aan elk label. Deze optie wordt pas echt interessant als we ze sturen vanuit het programma (zie §5.3).
- De naam van een CSS-identifier, voorafgegaan door een kruis (#). Elke component heeft hiertoe een eigenschap *id* die je kunt instellen met de *Scene Builder* (of rechtstreeks in het FXML-bestand).

1. Geef het label `teller` als CSS-identifier.
2. Gebruik deze CSS-identifier als locatie in het CSS-bestand.
3. Voer het programma uit. Is het label nog steeds rood?

Locaties kunnen ook gecombineerd worden. Zo betekent onderstaand fragment bijvoorbeeld dat elke knop die zich binnen een BorderPane bevindt, een vet lettertype moet gebruiken:

```
BorderPane Button {  
    -fx-font-weight: bold;  
}
```

Ter afsluiting van deze paragraaf drukken we hieronder het CSS-bestand af dat

²De documentatie over welke eigenschappen er met een JavaFX CSS-stylesheet kunnen veranderd worden en wat de correcte notatie is, is nogal moeilijk terug te vinden. Het gemakkelijkste is wellicht om gewoon naar ‘JavaFX 8 CSS Reference Guide’ te googelen.

we voor dit voorbeeld gebruikt hebben. Laat het een inspiratie zijn voor je eigen creativiteit!

```
BorderPane {  
    -fx-background-image: url("clock.jpg");  
    -fx-background-size: cover;  
  
    -fx-padding: 10;  
}
```

Het bestand *clock.jpg* moet in dezelfde map staan als het CSS-bestand. Let ook op dat je geen blanco plaatst vóór het open haakje van `url(...)`.

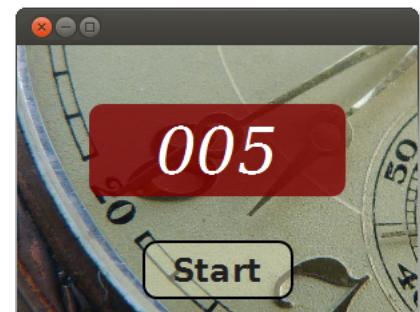
```
Label {  
    -fx-text-fill: white;  
  
    -fx-background-color: rgba(0,127,0,0.5);  
    -fx-background-radius: 10;  
  
    -fx-font-size: 48;  
    -fx-font-style: italic;  
    -fx-font-family: serif;  
  
    -fx-padding: 5 50 5 50;  
}  
  
Button {  
    -fx-background-color: rgba(245,245,205,0.4);  
    -fx-background-radius: 10pt;  
  
    -fx-font-size: 24;  
    -fx-font-weight: bold;  
  
    -fx-border-width: 2;  
    -fx-border-color: black;  
    -fx-border-radius: 10;  
  
    -fx-padding: 5 20 5 20;  
}
```

Let opnieuw op dat er geen blanco staat vóór het open haakje van `rgba(...)`.

5.3 Stijlklassen

We doen nog één aanpassing aan het aftelprogramma: wanneer de laatste 10 seconden beginnen, geven we het label een andere stijl.

Om dit voor elkaar te krijgen gebruiken we *stijlklassen*. We introduceren een stijklasse *alarm* om aan te duiden dat het label een rode kleur moet krijgen en we zorgen ervoor dat het programma deze stijklasse hecht aan het label zodra er nog minder dan 10 seconden op de teller staan.



Aan het CSS-bestand hoeven we nu enkel het volgende blok toe te voegen:

```
.alarm {  
    -fx-background-color: rgba(127, 0, 0, 0.8) ;  
}
```

En ook het programma vergt niet veel aanpassing. Bij elke JavaFX-component kan je met *getStyleClass* de lijst van stijlklassen opvragen³. Het programma hoeft enkel op het juiste moment de string "alarm" aan deze lijst toe te voegen of eruit te verwijderen. (We moeten wel vermijden dat dezelfde stijklasse meer dan één keer in de lijst terechtkomt.)

Toevoegen doen we zo:

```
label.getStyleClass().add ("alarm");
```

en terug verwijderen, zo:

```
label.getStyleClass().remove ("alarm");
```

1. Pas het programma aan zoals we in deze paragraaf hebben geschatst.
2. Test het programma. Wat gebeurt er wanneer je op de startknop drukt terwijl de teller nog aan het lopen is? Wordt het label terug groen?

³De methode had dus beter *getStyleClasses* geheten.

5.4 CSS voor foutbehandeling

Als laatste voorbeeld in dit hoofdstuk keren we terug naar de toepassing uit paragraaf §2.4 waar we een rood foutbericht tonen wanneer een veld leeg is. Bij deze versie tonen we echter niet alleen een foutbericht, maar kleuren we ook de rand van het tekstveld rood en het label links ervan.



We pakken het dit keer anders aan en gebruiken een stijlklasse om een fout te markeren.

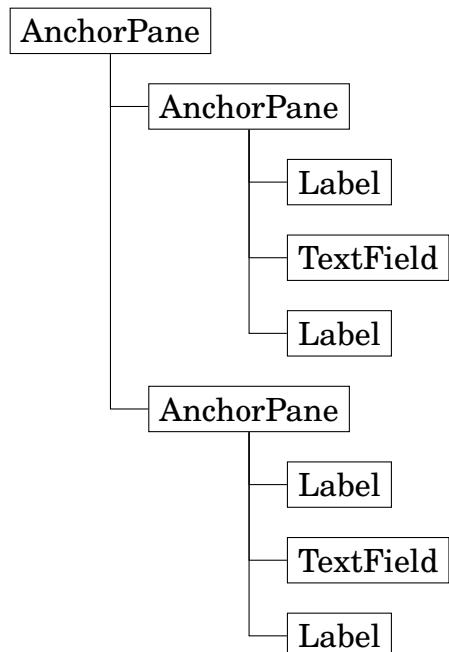
Om dit eenvoudiger te maken, groeperen we elk tekstveld in een afzonderlijk ankerpaneel samen met het label links ervan en het foutbericht eronder. De structuur van het formulier ziet er nu uit zoals hiernaast geschatst.

We verwijderen ook alle *fx:id*'s behalve deze van de tekstvelden.

Om in de stylesheet onderscheid te kunnen maken tussen beide soorten labels, hechten we aan het label met het foutbericht de stijlklasse *errorLabel*. Dit kan je doen in de *Scene Builder* of rechtstreeks in het FXML-bestand.

Wanneer een tekstveld leeg is, moet het programma het omsluitende ankerpaneel de stijlklasse *alarm* geven. We kunnen dan onderstaande CSS-code gebruiken om het gewenste effect te bekomen:

```
.alarm Label {  
    -fx-text-fill: red;  
}
```



```

.alarm Label.errorLabel {
    -fx-text-fill: red;
}

.alarm TextField {
    -fx-border-color: red;
}

Label {
    -fx-text-fill: black;
}

Label.errorLabel {
    -fx-text-fill: transparent;
}

```

Omdat het ankerpaneel geen *fx:id* heeft, kunnen we het niet rechtstreeks aanspreken. We kunnen echter wel aan een tekstveld vragen wat zijn omringende container is, zoals in onderstaande code⁴:

```

public boolean check (TextField field) {
    Parent parent = field.getParent();
    boolean empty = field.getText().trim().isEmpty();
    if (empty) {
        parent.getStyleClass().add ("alarm");
    } else {
        parent.getStyleClass().removeAll ("alarm");
    }
    return ! empty;
}

```

1. Schrijf een nieuwe versie van het project *User* uit hoofdstuk 2 die gebruik maakt van de techniek die we in deze paragraaf beschreven hebben.

We raden je aan met een leeg project te beginnen in plaats van rechtstreeks het oude project aan te passen. Op die manier vermijd je dat er overbodige (Java- en/of FXML-)code blijft staan. Je mag natuurlijk wel fragmenten uit de oude code knippen en plakken, maar probeer dit beperkt te houden.

⁴Het kan gebeuren dat dezelfde stijlklasse meer dan één keer toegevoegd raakt aan een component. Met *removeAll* verwijder je ze allemaal in één keer.