



Reader

TCIF-V10OC1-14

Inhoud

0. Resources	4
1. JavaFX Introductie	5
Inleiding.....	5
Tools en support	5
HelloWorld	6
Theater analogie	7
Handler	9
Innerclasses.....	9
Exit-quiz	10
Practicumopdracht 1-1.....	11
Practicumopdracht 1-2.....	12
Practicumopdracht 1-3.....	12
Practicumopdracht 1-4.....	13
Practicumopdracht 1-5 (Extra uitdagend).....	14
2. Layout en CSS.....	16
CSS-styling.....	16
Layouting.....	20
Achtergronden bij JavaFX, CSS & HTML5	21
Exit-quiz	22
Practicumopdracht 2-1.....	23
Practicumopdracht 2-2.....	24
Practicumopdracht 2-3.....	26
Practicumopdracht 2-4.....	27
Practicumopdracht 2-5.....	27
Practicumopdracht 2-6.....	28
Practicumopdracht 2-7.....	28
Practicumopdracht 2-8 (Extra uitdagend).....	29
3. Canvas	30
Canvas – path.....	33
Canvas – events	33
Exit-quiz	35
Practicumopdracht 3-1.....	36

Practicumopdracht 3-2.....	37
Practicumopdracht 3-3.....	37
Practicumopdracht 3-4.....	39
Practicumopdracht 3-5.....	40
Practicumopdracht 3-6.....	41
Practicumopdracht 3-7 (vervolg van 3-6).....	42
4. Menus, Files en Dialogs	43
MenuBar, Menu & MenuItem	44
MneMonics & Accelerators.....	45
ContextMenu	46
FileChoosers	47
Exit-quiz	49
Practicumopdracht 4-1.....	50
Practicumopdracht 4-2.....	51
Practicumopdracht 4-3.....	52
Practicumopdracht 4-4.....	53
Practicumopdracht 4-5.....	54
Practicumopdracht 4-6.....	55

0. Resources

- Het startpunt van de officiële Oracle JavaFX 8 documentatie:
<http://docs.oracle.com/javase/8/javase-clienttechnologies.htm>
- De JavaFX 8 CSS reference guide:
<http://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html>
- Een overzicht van JavaFX GUI controls met voorbeelden:
http://docs.oracle.com/javase/8/javafx/user-interface-tutorial/ui_controls.htm#JFXUI336
- Het werken met het JavaFX Canvas:
<http://docs.oracle.com/javase/8/javafx/graphics-tutorial/canvas.htm>

1. JavaFX Introductie

Inleiding

Na in het eerste semester kennis te hebben gemaakt met Java en Swing, kijken we in het tweede semester naar JavaFX. De ontwikkeling van JavaFX is in 2007 gestart door Sun Microsystems in een poging om een eenvoudigere Application Programming Interface (API) te ontwikkelen om in Java Graphical User Interfaces (GUIs) te kunnen ontwikkelen. Daarnaast moesten met JavaFX ook meer uitgebreidere userinterfaces gebouwd kunnen worden dan in Swing mogelijk is en was. JavaFX kan gebruikt worden als losstaande techniek, maar kan ook gecombineerd worden met Swing om zodoende oudere userinterfaces te updaten.

Na de overname van Sun door Oracle in 2010 heeft Oracle volop ingezet op de JavaFX technologie. Lange tijd heeft JavaFX bestaan als aparte library naast de standaard Java-installatie. Sinds de introductie van Java 8 in 2014 is JavaFX echter in de standaard Java-installatie geïntegreerd, een teken dat Oracle de techniek volwassen acht.

Hoewel JavaFX officieel geen vervanger is voor Swing (dat daarnaast voorlopig blijft voortbestaan), is de verwachting dat JavaFX de standaard client-technologie voor Java zal worden. De meest recente versie van JavaFX is JavaFX 8. Tijdens deze lessen zal regelmatig verwezen worden naar online-resources waar onderwerpen worden uitgelegd. Daarbij zal soms ook verwezen worden naar materiaal dat handelt over JavaFX 2. Dat zal uiteraard zijn als de informatie ook van toepassing is op JavaFX 8.

Tools en support

Voor deze lessen heb je in principe genoeg aan Textpad. Het kan soms echter erg handig zijn om gebruik te maken van een Integrated Development Environment (IDE) zoals Eclipse (www.eclipse.org) of NetBeans (<https://netbeans.org/>). Een IDE kan je veel typewerk uit handen nemen, maar zorgt er ook voor dat je de API-libraries van Java(FX) misschien minder goed leert kennen. Zeker als je moeite hebt met Java-tentamens is het de overweging waard om nog een lesblok Textpad te blijven gebruiken.

☛ Mocht je gebruik willen maken van Eclipse, dan kun je in voorbereiding op vakken in blok 4 het beste de ‘Eclipse IDE for Java EE Developers’ downloaden. Let op het subtiele verschil in naamgeving met de niet-EE variant! Voor een quickstart introductie, klik op de video rechts.



Naast directe programmeertools zijn er ook fora waar je vragen kunt stellen. Een community speciaal voor JavaFX kun je vinden op <https://www.java.net/community/javafx>. Hier kun je vragen stellen (en antwoorden vinden) met betrekking tot problemen of vragen waar je niet uitkomt. Uiteraard kun je daar ook anderen helpen die wellicht met een probleem kampen waar jij de oplossing voor weet!

HelloWorld

Om met JavaFX een GUI te kunnen bouwen is het noodzakelijk om te begrijpen hoe een JavaFX programma is opgebouwd. We bekijken dat allereerst aan de hand van het JavaFX HelloWorld voorbeeld. Bestudeer onderstaande voorbeeld:

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class HelloWorld extends Application
    implements EventHandler<ActionEvent> {

    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Say 'Hello World'");
        btn.setOnAction(this);

        StackPane root = new StackPane();
        root.getChildren().add(btn);

        Scene scene = new Scene(root, 300, 250);

        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public void handle(ActionEvent event) {
        System.out.println("Hello World!");
    }

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

Listing 1-1: HelloWorld

Net zoals bij ieder ander Java programma is ook hier een main-methode noodzakelijk om het programma te kunnen starten. In semester 1 is steeds gekozen om een aparte Main-klasse met daarin de main-methode te schrijven. Dat is nu nog steeds mogelijk, maar niet noodzakelijk. In dit geval hoeft alleen de applicatie gestart te worden, dus is een aparte Main-klasse ietwat overdreven.

➤ **Probeer bovenstaande voorbeeld uit te voeren op je eigen computer!**

Als je het programma hebt kunnen uitvoeren is het resultaat vergelijkbaar met afbeelding hiernaast. In de volgende paragraaf gaan we dieper in op de gegeven code.



Theater analogie

Om te uit te leggen hoe een JavaFX applicatie is opgebouwd is het handig om de theater-analogie te gebruiken. Het opbouwen van een GUI bestaat namelijk uit een aantal eenvoudige stappen:

1. Maak een podium (stage) waarop jouw programma wordt uitgevoerd.
2. Creëer een scene waarop de acteurs en benodigdheden geplaatst kunnen worden om (visueel aantrekkelijk) te communiceren met het publiek.
3. Richt je scene in met de benodigde acteurs en onderdelen.

De bovengenoemde stappen zijn ook in het HelloWorld voorbeeld toegepast. We lopen de code nu nogmaals langs om dit te constateren. We beginnen met de eerste drie regels:

```
public class HelloWorld extends Application
                                implements EventHandler<ActionEvent> {
    @Override
    public void start(Stage primaryStage) {
```

Elke JavaFX applicatie heeft een klasse die klasse `Application` extend en tegelijk ook methode `start()` overschrijft. In dit voorbeeld is ook de `@Override` annotatie gebruikt. De compiler weet zo dat je een methode van een superklasse wilt overschrijven en controleert of je daarbij de juiste methodenaam, het juiste returntype en de juiste parameters hebt gebruikt. Methode `start()` is als het beginpunt van elke JavaFX GUI.

Klasse `HelloWorld` implementeert daarnaast ook nog de `EventHandler`-interface (vergelijkbaar met de `ActionListener` van Swing). In dit geval hoeven er alleen actionevents verwerkt te worden, daarom is dat tussen de vishaakjes met `<ActionEvent>` aangegeven. We komen hier later nog op terug.

Zoals je ook in de code kunt zien heeft de methode `start()` 1 parameter; `Stage primaryStage`. Dit is het podium waarop je jouw scene kunt vertonen. Dit Stage-object hoef je dus niet zelf te maken maar wordt door Java gecreëerd en aangereikt. We kunnen nu dus beginnen met het opbouwen van een scene. Daarvoor worden in dit voorbeeld eerst de benodigdheden klaargezet:

```
Button btn = new Button();
btn.setText("Say 'Hello World'");
```

Het aanmaken van een button is vrijwel hetzelfde als met Swing, alleen heet de klasse nu geen `JButton` maar `Button` en is de import bovenaan de klasse dus ook `import javafx.scene.control.Button;`

- **In Swing kan je in 1 regel code een button maken en de tekst instellen. Zoek in de API-documentatie of dit in JavaFX ook kan. Zo ja, pas dit toe. Zo niet, leg uit waarom dat niet kan. Zie hiervoor <http://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Button.html>.**

Na het aanmaken van een button kan de handler aan de button gekoppeld worden:

```
btn.setOnAction(this);
```

Nu de button is gemaakt, kunnen we onze scene inrichten. Een scene bestaat altijd uit een parent- of root-node waaraan alle andere GUI-elementen hangen. We moeten dus goed kijken wat onze root-node gaat worden. Een button kan gebruikt worden als root, maar op een button kun je geen andere buttons of labels plaatsen. We hebben dus een element nodig wat als een soort container-element kan fungeren. Vergelijkbaar met een `<div>` tag in HTML of een `JPanel` van Swing. In dit voorbeeld is gekozen voor een `StackPane`:

```
StackPane root = new StackPane();  
root.getChildren().add(btn);
```

Dit is een panel waarbij alle items op een soort stapel worden geplaatst (op de z-as). Het element wat bovenop de stapel ligt is het best zichtbaar. Onderliggende items kunnen, afhankelijk van hun grootte, helemaal aan het zicht onttrokken worden. Uiteraard kun je de z-index van items veranderen.

Je kunt GUI-items aan een panel toevoegen door de lijst met child-nodes op te vragen en hier een nieuw item aan toevoegen zoals in bovenstaande voorbeeld is gedaan.

We hebben het nu diverse keren gehad over een scene, maar we hebben hier nog niets mee gedaan. De volgende regel brengt daar verandering in:

```
Scene scene = new Scene(root, 300, 250);
```

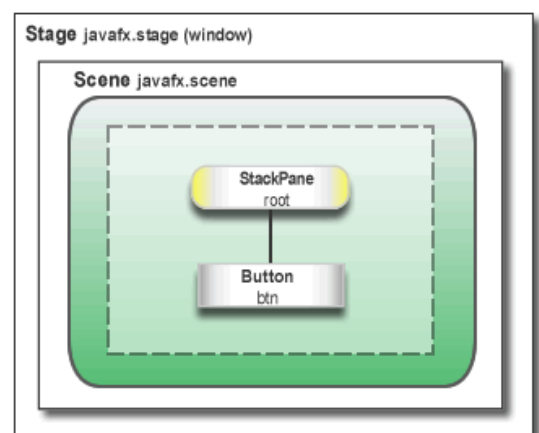
In een scene kan de gebruiker communiceren met de applicatie en een scene kent zijn eigen GUI-items. Net als in een theater kunnen er in een applicatie meerdere scenes zijn. In een game kun je bijvoorbeeld de gameworld tonen (hoofdscene), terwijl je met de Escape-toets teruggaat naar het menu (menuscene). Op die manier kan je logica in je applicatie netjes gescheiden houden en hoeft de processor bij het weergegeven van de hoofdscene geen rekening te houden met de menuscene, dat scheelt rekenkracht en performance.

Hiermee zijn we er nog niet:

```
primaryStage.setTitle("Hello World!");  
primaryStage.setScene(scene);  
primaryStage.show();  
}
```

In deze laatste regels van de methode `start()` wordt de scene aan het venster (`primaryStage`) gekoppeld. Daarnaast is er een titel ingesteld en is de stage zichtbaar gemaakt met de methode `show()`. De applicatie is daarmee opgebouwd zoals in afbeelding hiernaast is weergegeven.

- **Probeer een tweede button op het scherm te plaatsen en probeer te verklaren wat je ziet.**



Handler

Het enige stuk code wat we nog niet hebben bekeken is de handler-methode. De methode `handle(.)` wordt uitgevoerd als de knop is ingedrukt. Net als de methode `actionPerformed(.)` van `ActionListener` heeft ook deze methode een `ActionEvent` parameter. Na indrukken van de knop wordt de tekst "Hello World!" in de console getoond. Er is maar 1 button dus hoeft er niet gecontroleerd te worden welke button is ingedrukt.

```
public void handle(ActionEvent event) {
    System.out.println("Hello World!");
}
```

- **Wijzig de methode `handle()` zodat wanneer je op button 1 klikt, button 2 bovenop de `StackPane` geplaatst wordt. Als je dan op button 2 klikt moet button 1 weer bovenop de geplaatst worden etc.. Je moet dus de z-index van de buton wijzigen. Zoek in de API-documentatie hoe dit moet. Gek genoeg kun je dat niet vinden in de documentatie van `StackPane`, maar in de documentatie van klasse `Node` (een superklasse van `Button`). Zie <http://docs.oracle.com/javase/8/javafx/api/javafx/scene/Node.html> voor informatie.**

Innerclasses

Het koppelen van een handler lijkt op de werkwijze die je kent uit semester 1 met `addActionListener(.)`. In de voorbeelden die je op internet tegen zult komen gebruikt men echter ook veelvuldig innerclasses. Hierop zal in de 3^e Javales uitgebreid op ingegaan worden. Ter illustratie hoe dit eruit ziet is in Listing 1-2 alvast een variant van het HelloWorld voorbeeld opgenomen met een innerclass.

```
// dezelfde imports als Listing 1
public class HelloWorld extends Application {
    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Say 'Hello World'");
        btn.setOnAction(new EventHandler<ActionEvent>() {

            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });

        StackPane root = new StackPane();
        root.getChildren().add(btn);
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

Listing 1-2: HelloWorld met innerclass-handler

In Listing 1-2 is iets merkwaardigs aan de hand. Je hebt namelijk geleerd dat je van een interface geen object kunt aanmaken. Toch staat daar de code `new EventHandler<ActionEvent>()`. Dit is echter alleen toegestaan als je daarna ook direct de implementatie van de interface geeft. De bijbehorende methode `handle(.)` staat er daarom tussen accolades achter. Het verschil is dat nu niet meer klasse `HelloWorld` deze interface heeft geïmplementeerd, maar dat dit door een anonieme innerclass is overgenomen. Meer hierover dus in Java les 3.

Op dit moment is in het voorbeeld nog sprake van uitvoer naar de console. Dit is niet erg netjes. We gaan dit oplossen door een extra label toe te voegen waarin de uitvoer komt te staan.

- **Voeg naast de twee buttons nu ook een label toe (voeg deze als laatste toe aan de `StackPane`).**

Als je het programma nu start zie je het label op een lelijke manier bovenop een button staan. Als je vervolgens op een knop klikt, is het label voorgoed verdwenen. Dat is niet handig. `StackPane` is een subklasse van `Pane`. Er zijn nog meer subklassen van `Pane`, waaronder `FlowPane` en `HBox`.

Zie voor `HBox`: <http://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/HBox.html>.

Zie voor `FlowPane`: <http://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/FlowPane.html>.

- **Vervang de `StackPane` door een `HBox`. Geef in de constructor aan dat de afstand tussen twee GUI-items 10 pixels moet zijn. Wat is nu het effect van de code waarmee je de buttons eerst boven of onder elkaar kon plaatsen?**
- **Vervang de `HBox` nu door een `FlowPane` en zorg ervoor dat beide buttons ook een (verschillende) tekst in het label plaatsen.**
- **Geef de items een voorkeursbreedte met methode `setPrefWidth(.)` zodat ze onder elkaar komen te staan.**

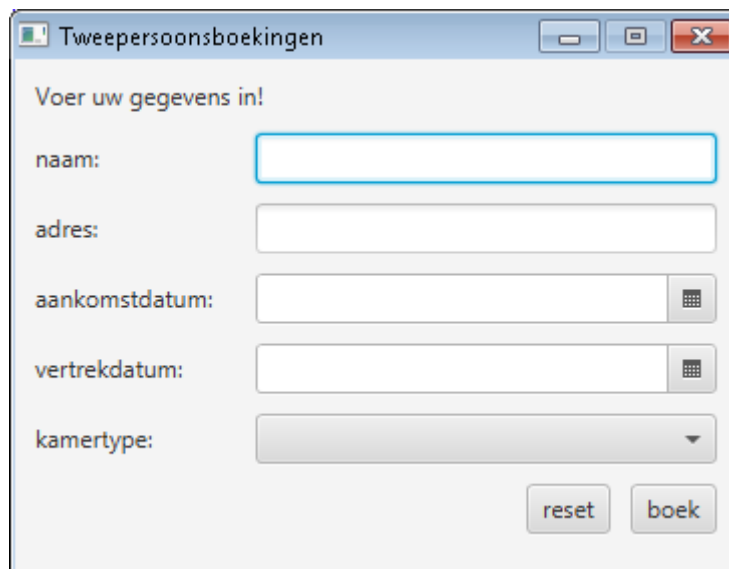
Exit-quiz

- Met welke opdracht start je een JavaFX applicatie altijd?
- Wat is het verschil tussen een `Stage` en een `Scene` object?
- Welk object moet je zelf aanmaken, een `Stage` of een `Scene`?
- Welke klasse moet elk JavaFX programma extenden?
- Welke methode moet elke JavaFX applicatie te hebben?
- Hoe komt het dat een applicatie anders niet gecompileerd kan worden?
- Met welke methode kan je jouw applicatie zichtbaar maken?
- Welke interface gebruik je om onder andere gebruikersinvoer af te vangen?
- Wat is de naam van de methode waarmee je een item in een `StackPane` bovenop kunt plaatsen?
- Hoeveel constructors heeft een `HBox` om de afstand tussen items in een box in te stellen?
- Met welke methode kan je de afstand tussen items in een `HBox` naderhand nog wijzigen?
- Met welke methode kan je in één keer de breedte en hoogte van je labels en buttons instellen?
- Hoe voeg je child-elementen toe aan bijvoorbeeld een `FlowPane`? En hoe aan een `HBox`?

Practicumopdracht 1-1

Deze opdracht is een uitbreiding op de Java-practicumopdracht Inl-1 van les 1. In die opdracht is een klasse `Main` aangeleverd waarin een boeking, een klant, kamertypen en kamers zijn gedeclareerd. In de opdrachten van dit hoofdstuk moet voor het aanmaken van boekingen een GUI worden gerealiseerd.

- **Maak een scherm zoals hieronder is weergegeven in een nieuwe klasse, klasse `BoekingApp`. De buttons hoeven voor dit deel van de opdracht nog niets te doen, dat komt bij de volgende opdracht.**

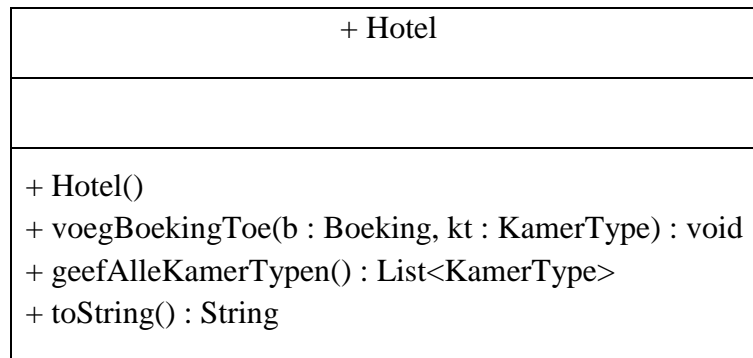


Tips bij dit scherm:

- Er is gebruik gemaakt van een `FlowPane` om alle elementen op te plaatsen.
- Er is 6x een `Label` gebruikt.
- Er is 2x een `TextField` gebruikt.
- Er is 2x een `DatePicker` gebruikt.
- Er is 1x een `ComboBox` gebruikt.
- Er is 2x een `Button` gebruikt.
- De scene heeft een vaste grootte (stage is niet resizable) van 350 x 175.
- Door de voorkeursbreedte van diverse items in te stellen kan deze indeling bereikt worden.
- De buttons staan in een `HBox` die bijna net zo breed is als het scherm en rechts is uitgelijnd.
- Het `FlowPane` heeft een padding van 10 pixels.
- De `ComboBox` gaat items bevatten van het type '`KamerType`' van het eerder gemaakte practicum.
- Zie http://docs.oracle.com/javase/8/javafx/user-interface-tutorial/ui_controls.htm#JFXUI336 voor voorbeelden van de gebruikte items.
- Maak gebruik van het overzicht met resources aan het begin van deze reader.

Practicumopdracht 1-2

Om de GUI van opdracht 1-1 te koppelen aan het boekingen-domein introduceren we een `Hotel`-klasse. Het UML van de klasse `Hotel` is al gegeven. Er liggen echter nog geen associaties naar de andere klassen.



In de constructor komt de code uit de oorspronkelijke klasse `Main`. De daar aangemaakte objecten van het type `Boeking`, `KamerType` en `Kamer` moeten worden opgeslagen in arraylijsten. De `voegBoekingToe()` methode moet de nieuwe boeking toevoegen aan een lijst met boekingen. Voor nu doen we nog niets met het `KamerType`, deze parameter is nodig bij de (optionele) opdracht 1-5. Methode `geefAlleKamerTypen()` geeft de lijst met alle bekende kamertypen. Methode `toString()` geeft een overzichtelijke string met alle boekingen van het `Hotel`.

- **Maak een UML-klassendiagram waarin bovenstaande klasse `Hotel` ook is opgenomen. Zorg ervoor dat je de associaties zo legt dat je de methoden van klasse `Hotel` kunt programmeren. Het UML-diagram dient minimaal de associaties, multiplicititeit en directionaliteit tussen de diverse klassen weer te geven, de attributen mag je achterwege laten.**
- **Overleg met medestudenten over de oplossing en laat je docent jullie uitwerking controleren voordat je begint met programmeren!**

Practicumopdracht 1-3

- **Programmeer de klasse `Hotel`.**

De code van de klasse `Main` kan verplaatst worden naar de constructor van `Hotel` om zodoende alvast een aantal gegevens beschikbaar te hebben als de GUI start. Zorg ervoor dat aangemaakte objecten worden bijgehouden in lijsten (`ArrayList`).

Practicumopdracht 1-4

Als je de opdrachten 1-1 t/m 1-3 hebt afgerond heb je alle benodigde onderdelen om de GUI aan het domein te koppelen. Neem daartoe het volgende attribuut op in je klasse `BoekingApp`:

```
private Hotel hotel = new Hotel();
```

- **Vul de `ComboBox` met de beschikbare `KamerType` objecten van `Hotel`. Gebruik daarvoor methode `geefAlleKamerTypen()`.**
- **Vul de eerste `DatePicker` met de datum van vandaag, dat kan met `datePicker1.setValue(LocalDate.now());`**
- **Vul de tweede `DatePicker` met de datum van morgen, zie de documentatie van `LocalDate` voor handige methoden die je kunnen helpen.**
- **Zorg dat de boeking vastgelegd wordt bij het hotel als er op de knop ‘boek’ is gedrukt. Daar kun je methode `voegBoekingToe()` voor gebruiken. Dit mag alleen als aan de onderstaande voorwaarden is voldaan. In andere gevallen moet er een duidelijke foutmelding in het bovenste label geplaatst worden. Als de boeking succesvol is moet dat gemeld worden. De voorwaarden:**
 - De naam is ingevuld
 - Het adres is ingevuld
 - De aankomst- en vertrekdatum niet in het verleden liggen
 - De aankomstdatum voor de vertrekdatum valt
 - Er is een kamertype geselecteerd

Je zult merken dat de `DatePicker` een `LocalDate` teruggeeft in plaats van een `Calendar`-object. Je mag hiervoor de klasse `Boeking` aanpassen zodat die ook `LocalDate` objecten gebruikt, maar je mag ook de onderstaande code gebruiken om `LocalDate` objecten om te zetten naar `Calendar` objecten (en vice versa):

```
private Calendar localDateToCalendar(LocalDate d) {
    Calendar c = Calendar.getInstance();
    c.set(d.getYear(), d.getMonthValue()-1, d.getDayOfMonth());
    return c;
}

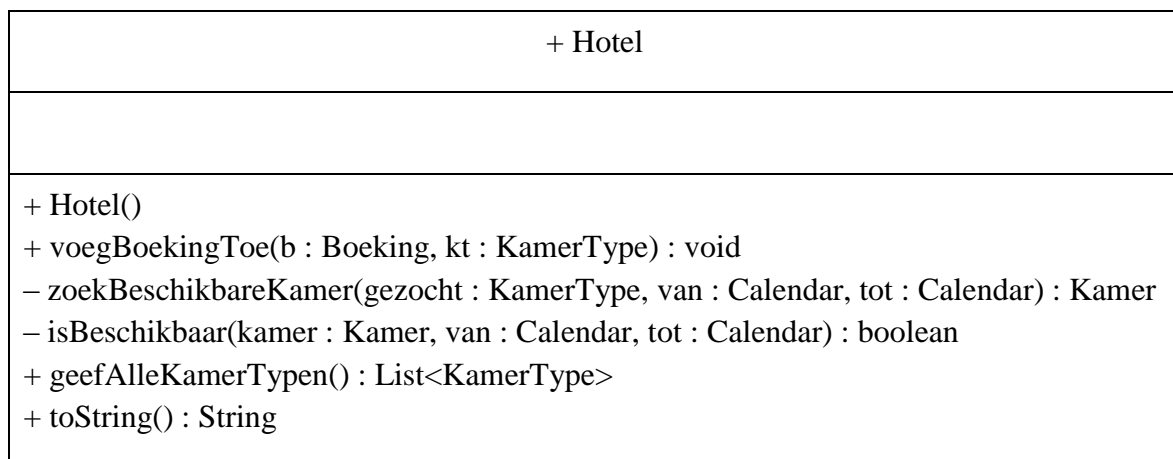
private LocalDate calendarToLocalDate(Calendar cal) {
    int year = cal.get(Calendar.YEAR);
    int month = cal.get(Calendar.MONTH)+1;
    int day = cal.get(Calendar.DAY_OF_MONTH);
    return LocalDate.of(year, month, day);
}
```

- **Zorg dat bij indrukken van de knop ‘reset’ alle velden geleegd worden.**
- **Toon na het succesvol toevoegen van een boeking de `toString()` van `Hotel` in de console en wis de invoervelden (laat nog wel het bericht staan dat de boeking succesvol was).**

Het gewenste eindresultaat van opdracht 1-1 t/m 1-4 is hieronder zichtbaar gemaakt.

Practicumopdracht 1-5 (Extra uitdagend)

- **Controleer bij een boeking of er wel voldoende kamers beschikbaar zijn en of deze op het moment van de boeking nog niet bezet zijn. Programmeer daarvoor in de methode `voegBoekingToe()` in klasse `Hotel` een check en wijs een kamer toe aan de boeking als er nog een kamer vrij is. Je zult hiervoor wellicht een aantal getters moeten toevoegen in de domeinklassen. Om je op weg te helpen staat hieronder een uitgebreider UML-klassendiagram van de klasse `Hotel`:**



Methode `zoekBeschikbareKamer()` doorloopt de lijst met kamers om te kijken of een kamer van het juiste type is. Als een kamer gevonden is, kan methode `isBeschikbaar()` gebruikt worden om te kijken of een `Kamer` beschikbaar is in de gewenste periode. Daartoe worden alle boekingen doorlopen om te controleren of de kamer al in een boeking voorkomt die binnen de gewenste periode valt. **Tip:** maak gebruik van de `before()` en `after()` methoden van `Calendar`.

- **Zorg dat de methode `voegBoekingToe()` een boolean returned: `true` als er nog kamers beschikbaar zijn, `false` als dat niet zo is. Geef in het laatste geval netjes een melding in de GUI.**
- **Test je code met voldoende testcases!**

2. Layout en CSS

In dit hoofdstuk gaan we dieper in op de diverse layout-mechanismen van JavaFX en bijbehorende technieken om een GUI te stylen. In hoofdstuk 1 heb je al kennis gemaakt met een diverse panel-soorten zoals FlowPane en HBox. Daarnaast zijn er nog meer soorten zoals BorderPane, AnchorPane, GridPane etc.

Layouting is een van de moeilijkste onderdelen om te ontwikkelen. Allerlei items hebben een wisselwerking op elkaar en moeten op een fatsoenlijke manier getoond en gemanipuleerd kunnen worden. In Swing heb je tot nu toe het grootste deel van de styling gedaan met methode-aanroepen. In JavaFX heeft men echter sterk ingezet op CSS en HTML5 als extra tool op styling van elementen te kunnen realiseren. Dit hoofdstuk gaan we in op CSS en laten we de HTML-code achterwege.

CSS-styling

Net zoals bij HTML-bestanden, kun je ook aan een JavaFX programma een stylesheet toevoegen. JavaFX ondersteunt op dit moment enkele onderdelen uit de CSS3 specificatie, maar helaas nog niet alles. We moeten ons dus beperken tot CSS2.1, maar dat zal ik de praktijk weinig problemen opleveren. Wat niet mogelijk is in CSS, kan vaak wel met programmacode worden opgelost. Anders dan bij HTML zul je merken dat niet alle styling met CSS gedaan kan worden. Soms wil je namelijk meer flexibel zijn of krachtigere toepassingen maken dan met CSS mogelijk is.

- **Bekijk de volgende video en bestudeer de gegeven voorbeelden nauwkeurig.**



JavaFX Tutorial #2



In de video is sprake van JavaFX 2 en Java 7. Aan de gegeven voorbeelden doet dat niets af, maar de CSS stylesheet waarover aan het eind sprake is, staat nu in de standaard Java-library. De file is gelijk gebleven: `jfxrt.jar`.

Naast de voorbeelden uit de video is hieronder ook het HelloWorld-voorbeeld uit hoofdstuk 1 uitgerust met een stylesheet zodat je het makkelijker kunt gebruiken bij de practicumopgaven.

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.FlowPane;
import javafx.stage.Stage;
import javafx.geometry.Pos;

public class HelloWorld extends Application
    implements EventHandler<ActionEvent> {

    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Say 'Hello World'");
        btn.getStyleClass().add("mybutton");
        btn.setId("special");
        btn.setOnAction(this);

        FlowPane root = new FlowPane(10, 10);
        root.setAlignment(Pos.CENTER);
        root.getChildren().addAll(btn);

        Scene scene = new Scene(root, 300, 250);
        scene.getStylesheets().add("stylesheet.css");

        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public void handle(ActionEvent event) {
        System.out.println("Hello World!");
    }

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

Listing 2-1: HelloWorld met stylesheet

Dit voorbeeld wijkt gedeeltelijk af van de video, want er is hier ook sprake van een id. Een id is specifieker dan een class en moet uniek zijn. Het bestand stylesheet.css, wat in dezelfde directory is opgeslagen, is op de volgende pagina afgedrukt.


```
.mybutton {
    -fx-background-color: lightblue;
}

#special {
    -fx-font-weight: bold;
}
```

In feite worden de classes van een button (of ander item) opgeslagen in een List. Dat komt omdat je meer dan 1 class kunt toepassen op element (in tegenstelling tot het id). Dit kun je al een beetje afleiden aan het toevoegen van de stylesheets met de methode ‘add’.

- **Wijzig de code zodanig dat wanneer je klikt op de button, de class wordt losgekoppeld van de button. Dit moet als effect hebben dat de tekst vetgedrukt blijft, maar de achtergrond wijzigt.**

Zoals je al hebt gezien, moet je voor de gewone CSS eigenschappen ook de prefix ‘-fx-’ toevoegen

Eclipse, JavaFX & CSS stylesheets

Als je gebruik maakt van Eclipse kan je applicatie de stylesheet soms niet goed vinden, dat heeft te maken met het classpath waar Eclipse in zoekt. Een oplossing kan dan zijn om het volledige path op te geven zoals hieronder is gedaan. Je kunt het bestand dan gewoon in de root-folder van je project in Eclipse plaatsen.:

```
File cssSheet = new File("stylesheet.css");
scene.getStylesheets().add(cssSheet.toURI().toString());
```

Uiteraard moet je wel de klasse `java.io.File` importeren om dit te laten werken. Methode `toURI()` levert het volledige path naar het bestand en vervangt daarbij meteen eventuele karakters die een probleem kunnen opleveren. Spaties worden bijvoorbeeld vervangen door ‘%20’, een algemene codering voor spaties waar de meeste systemen mee overweg kunnen. Met methode `toString()` kan je volledige path toevoegen aan de lijst met stylesheets.

- ☛ Zie de eerste paragraaf van hoofdstuk 3 voor een verdere informatie over URI's.

Neem <http://docs.oracle.com/javase/8/javafx/user-interface-tutorial/apply-css.htm#CHDGHCDG> door voor meer voorbeelden en toelichting op dit onderwerp.

Op <http://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html#introscenegraph> kun je een complete JavaFX CSS reference vinden met voorbeelden en informatie over allerlei CSS eigenschappen die standaard bij de UI-controls zijn opgenomen.

- **Een lijst kent ook de methode `clear()`. Daarmee maak je de lijst leeg. Je hebt zelf maar 1 stylesheet toegevoegd, dus je zou de lijst ook met `clear()` kunnen leegmaken. Doe dit en controleer het effect. Wat maak je daaruit op?**

Layouting

Layouting kan lastig zijn, maar als je weet wat de eigenschappen zijn van diverse layouting-technieken kan je daar veel gemak van hebben. Bestudeer daarom de op http://docs.oracle.com/javase/8/javafx/layout-tutorial/builtin_layouts.htm#JFXLY102 de voorbeelden en de informatie die daar gegeven is. Probeer daarna de onderstaande vragen te beantwoorden en test je antwoorden met kleine Java-programma's. Zoek indien nodig in de documentatie van de bijbehorende klassen naar antwoorden.

- In een `FlowPane` kun je eindeloos veel GUI-controls toevoegen. Wat gebeurt er als je twee items in een `BorderPane` plaatst? Worden ze dan beide getoond, komen ze onder elkaar terecht of is er ander gedrag?
- Wat gebeurt er als je op een `BorderPane` items toevoegt met de `getChildren().add(...)` methode?
- Wat is het verschil tussen een `AnchorPane` en een `BorderPane`? Wat zijn de voor- en nadelen en in welke situaties pas je beiden toe?
- Een `TilePane` en `GridPane` werken beiden met 'cellen'. Waarin zijn ze verschillend?

In de practicumopdracht van hoofdstuk 2 moet een GUI voor een damspel gemaakt worden. Zie een mogelijk screenshot hiernaast.

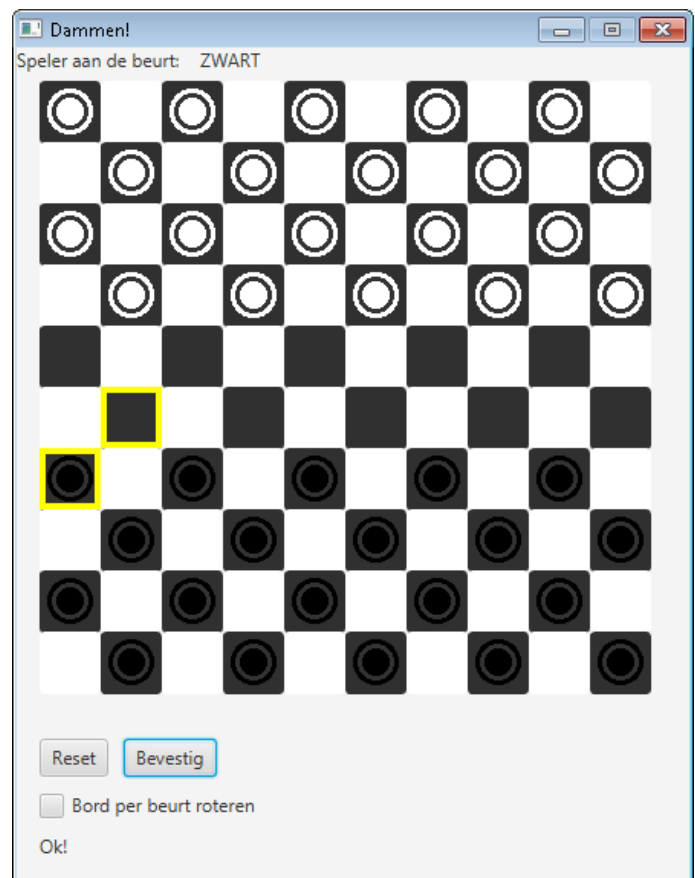
- Maak een schets (bijvoorbeeld zoals op blz. 6 onderaan) waarin je aangeeft hoe jij deze layout zou realiseren. Geef aan welke layout-panels je zou gebruiken en welke UI-controls je zou toepassen voor de damstenen.. Je mag ook een andere layout ontwerpen (met minimaal dezelfde mogelijkheden).

Over het ontwerpen van GUIs zijn al vele boeken geschreven. Er zijn diverse bronnen te vinden waar GUI-ontwerpprincipes staan opgesomd.

- Tegen welke GUI-ontwerpprincipes gaat dit scherm in? Hoe zou dat beter kunnen?

Een aantal principes kun je vinden op:

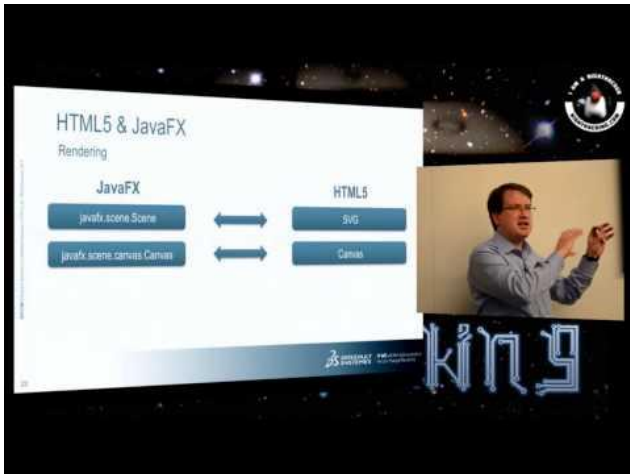
- http://web.cs.wpi.edu/~matt/courses/cs563/talks/smartin/int_design.html
- <http://www.usability.gov/what-and-why/user-interface-design.html>



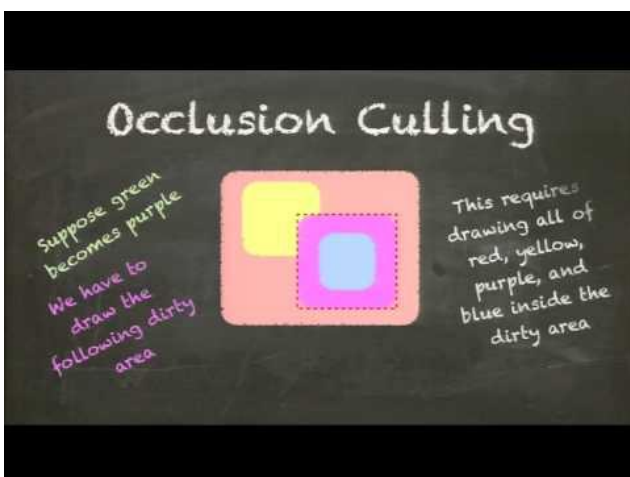
Achtergronden bij JavaFX, CSS & HTML5

Een uitgebreide introductie op de achtergronden van de in dit hoofdstuk besproken technieken in Java kun je beluisteren in de volgende videos. Ze zijn beide vrij lang, maar gaan dieper in op de achtergronden, voor- en nadelen van JavaFX ten opzichte van, en in combinatie met HTML5.

De tweede video bevat in het laatste deel ook een interessant voorbeeld hoe je HTML kunt gebruiken om je interface snel vorm te geven en online content (zoals GoogleMaps) kunt verwerken in JavaFX applicaties.



Zoals je in de videos kunt zien zijn er eindeloos veel mogelijkheden om Java te combineren met Web-technieken. Een voorbeeld waarin je met HTML de GUI van je applicatie kunt opbouwen kun je ook nog eens rustig teruglezen op <http://docs.oracle.com/javase/8/javafx/embedded-browser-tutorial/overview.htm#JFXWV135>.



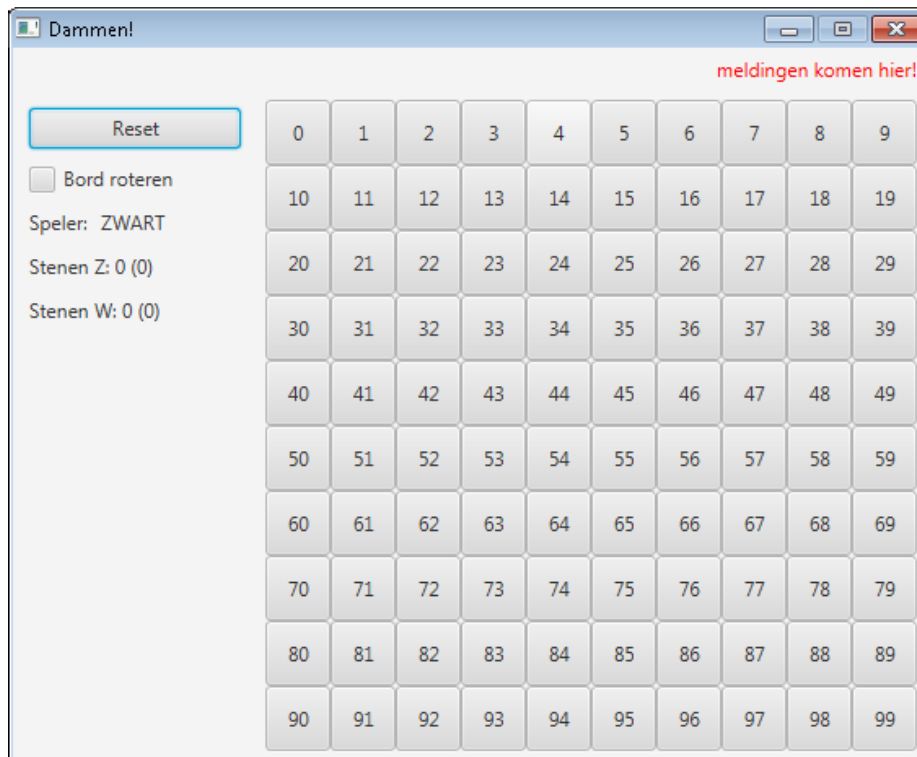
Exit-quiz

- Hoeveel stylesheets kan je aan je programma koppelen?
- Wat is ook al weer het verschil tussen een CSS 'id' en een 'class'?
- Welke methoden moet je gebruiken om een id of class toe te kennen aan een GUI-onderdeel?
- In welke klasse zijn die methoden eigenlijk geprogrammeerd?
- De ene methode is een setter, de andere niet. Waarom is dat verschil?
- Wat is de prefix (voorvoegsel) voor JavaFX CSS eigenschappen?
- Met welke JavaFX CSS eigenschap kan je een Label een 'underline' geven?
- Welke JavaFX CSS eigenschap gebruik je om een lettertype in te stellen op de Scene?
- Hoe stel je de padding-links van een button in?
- In een FlowPane kun je eindeloos veel GUI-controls toevoegen. Wat gebeurt er als je twee items in een BorderPane plaatst? Worden ze dan beide getoond, komen ze onder elkaar terecht of is er ander gedrag?
- Wat gebeurt er als je op een BorderPane items toevoegt met de `getChildren().add(..)` methode? Hoe zou het wel moeten?
- Behalve met CSS kan je ook met programmacode de padding instellen. Hoe moet dat bij een HBox?
- Wat is het verschil tussen een AnchorPane en een BorderPane? Wat zijn de voor- en nadelen en in welke situaties pas je beiden toe?
- Een TilePane en GridPane werken beiden met 'cellen'. Waarin zijn ze verschillend?
- Welke drie userinterface designprincipes vind jij het belangrijkste? Waarom?

Practicumopdracht 2-1

De opdracht van dit hoofdstuk bestaat uit het maken van een GUI voor een damspel. De logica en de regels hoef je niet te programmeren, die kun je vinden op SharePoint (zie ook opdracht 2-2).

- **Maak een GUI zoals hieronder is weergegeven in een nieuwe klasse, klasse `DamspelApp`. De buttons hoeven voor dit deel van de opdracht nog niets te doen, dat komt bij de volgende opdracht. De dambord-buttons moeten in de volgende opdracht met CSS opgemaakt kunnen worden. Geef de buttons daarom een id (zie het HelloWorld-voorbeeld van dit hoofdstuk) en plaats dat ook op de buttons.**



Tips bij dit scherm (het is niet verplicht om het zo aan te pakken):

- Er is gebruik gemaakt van een `BorderPane`
- Er is 4x een `Label` gebruikt.
- Er is 1x een `CheckBox` gebruikt.
- Er is 101x een `Button` gebruikt.
- De buttons staan in een `GridPane` die in het centrum van de `BorderPane` zijn geplaatst.
- Er is een `for-lus` gebruikt om de buttons te plaatsen.
- De scene heeft geen specifieke grootte (stage is niet resizable).
- De grootte van de GUI is afhankelijk van de grootte van de dambord-buttons.
- Er is alleen voor de dam-buttons een voorkeursbreedte van 40 x 40 ingesteld.
- De menuitems staan in een `VBox` (soms gecombineerd met `HBox` panes) aan de 'linkerkant'.
- Maak gebruik van het overzicht met resources aan het begin van deze reader!

Practicumopdracht 2-2

Bij een nette scheiding van verantwoordelijkheden in een programma (separation of concerns) is het belangrijk dat de GUI gescheiden is van de domeinlogica. Domeinlogica is in feite de programmacode die betrekking heeft op alleen het damspel, en geheel losstaat van een GUI. In theorie is het dan zelfs mogelijk om aan hetzelfde domeinmodel zowel een DOS-box, een webapplicatie en een desktop GUI te koppelen. Zover zullen we hier niet gaan.

Om hier kennis mee te maken krijg je bij deze opdracht de klasse `Damspel` mee. Deze klasse bevat de logica die voor het damspel nodig is. Een deel van het UML-klassendiagram van de klasse is hieronder gegeven.

+ Damspel	
+ Damspel() + reset() : void // start het spel opnieuw + getVeldStatus(veldId : int) : String // geeft een string met de veldstatus + getMelding() : String // geeft laatste bericht (foutmelding / informatie) + getSpeler() : String // geeft de naam van de huidige speler + toString() : String // geeft het bord in tekstvorm	

De 'status' van het damspel houden we bij met deze klasse. Met de constructor kun je een spel aanmaken. Vervolgens kun je met de methode `getVeldStatus(.)` de status van een veld als string opvragen. Er zijn 100 velden met de id's 0 t/m 99. De status kan "WIT", "ZWART", "WITDAM", "ZWARTDAM", "NIETSPEELBAAR" of "LEEG" zijn. Deze string kun je gebruiken om de buttons de juiste CSS-class te geven:

```
String cssClassVoorButton10 = spel.getVeldStatus(10);
```

- **Neem in jouw klasse `DamspelApp` het volgende attribuut op:**

```
private Damspel spel = new Damspel();
```

- **Zorg dat elke button dezelfde CSS class krijgt als de veldstatus. Dus button 0 krijgt class "WIT", button 1 "NIETSPEELBAAR", button 2 "WIT" etc. Zorg er met CSS ervoor dat de buttons de juiste `background-color` en, als er een damsteen op staat, het `background-image` van een damsteen krijgen. De afbeeldingen van de stenen kun je op SharePoint vinden. Zie opdracht 2-3 voor een screenshot van een vergelijkbaar resultaat.**

Practicumopdracht 2-3

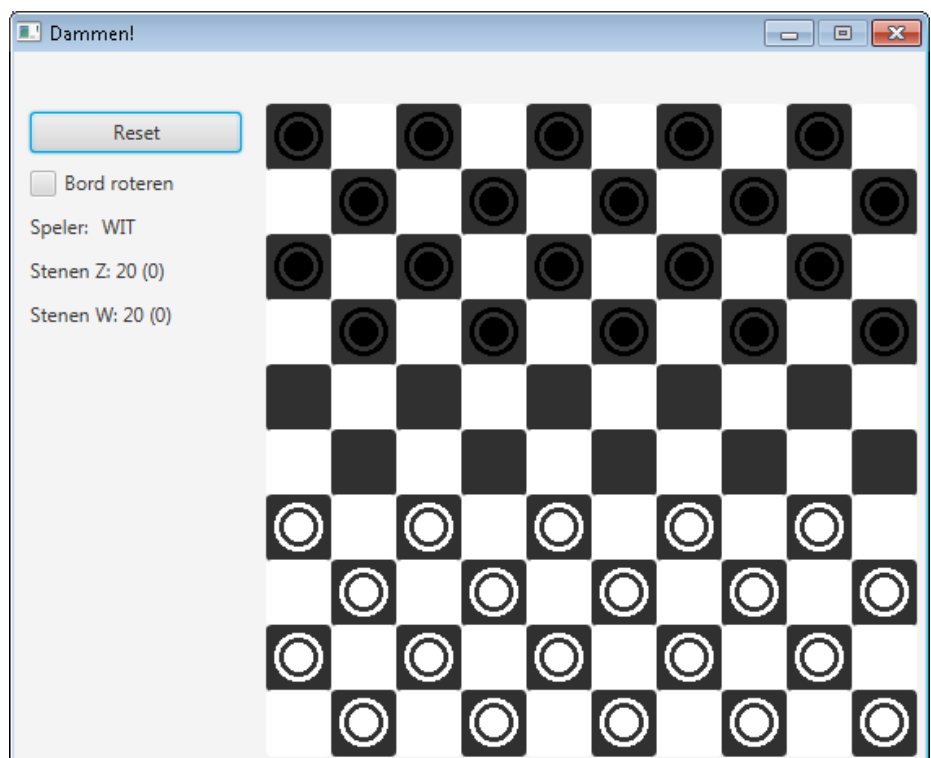
Het spel is nu klaar om gespeeld worden. Daarvoor bevat `Damspel` nog 2 methoden:

+ Damspel
<pre>+ Damspel() + reset() : void + getVeldStatus(veldId : int) : String + isVeldSpeelbaar(veldId : int) : boolean + doeZet(vanVeldId : int, naarVeldId : int) : boolean + getMelding() : String + getSpeler() : String + toString() : String</pre>

Zorg ervoor dat buttons een `EventHandler` krijgen. Klasse `DammenApp` kan daarvoor de `EventHandler` interface implementeren. Als een speler een zet wil doen moet hij een veld aanklikken. Dat mogen niet alle velden zijn. Gebruik methode `isVeldSpeelbaar(..)` om te bepalen of een geschikt veld is aangeklikt. Zo ja, geef het veld een duidelijke border (met behulp van CSS). Bij nog een muisklik wordt de selectie weer opgeheven.

Het tweede veld dat de gebruiker aanklikt moet als (2^e) parameter aan de `doeZet(..)` methode meegegeven worden. Als de zet is toegestaan kan het bord ge-update worden. Anders moet een de laatste melding van klasse `Damspel` in het label rechtsbovenin getoond worden.

Het kan handig zijn om een aparte methode `updateBord()` te maken die ervoor zorgt dat buttons van het bord voorzien worden van de juiste CSS-class, de geselecteerde een CSS style voor de border geeft en tevens de labels van de GUI ververs.

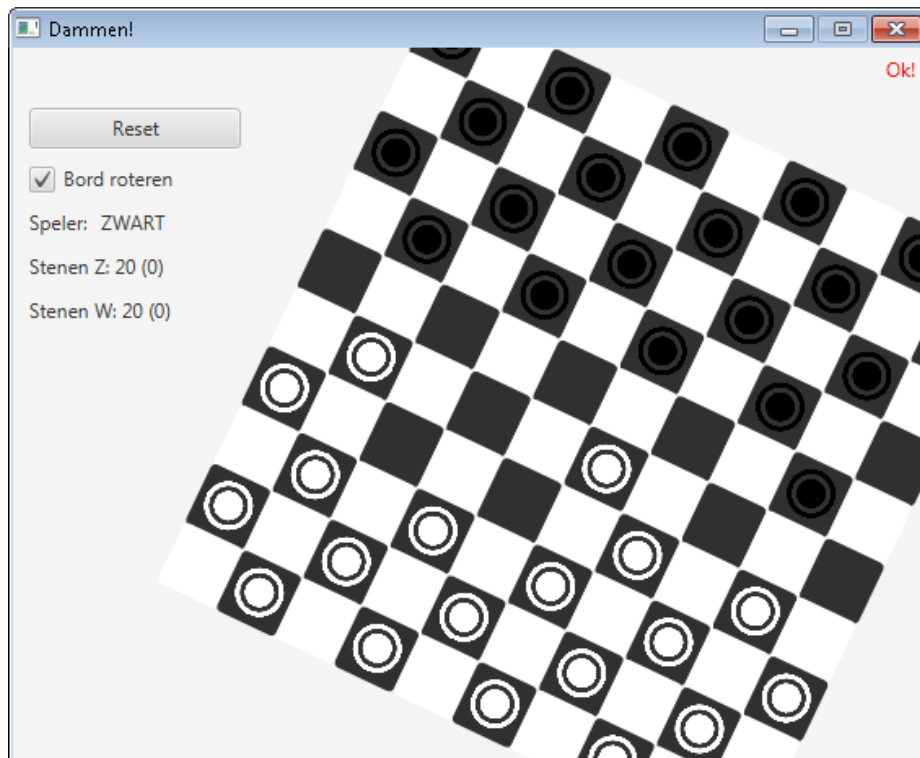


➤ **Programmeer de GUI zoals in deze opdracht is beschreven**

Practicumopdracht 2-4

Net als in CSS kun je ook in JavaFX transformaties en transitie toepassen. Zie voor uitleg en voorbeelden <http://docs.oracle.com/javase/8/javafx/visual-effects-tutorial/basics.htm#BEIIJJJB>.

- Zorg ervoor dat het speelbord omdraait wanneer de beurt wisselt, zodat de andere speler zijn stenen duidelijk voor zich heeft. Dit alles alleen als de rotatie-checkbox geselecteerd is.



Practicumopdracht 2-5

In de klasse `Damspeel` bevinden zich naast de eerder genoemde methoden ook de methoden:

```
public StringProperty getMeldingProperty()  
public StringProperty getSpelerProperty()
```

Properties zijn ideaal om een GUI gesynchroniseerd te houden met het model zonder allerlei lastige code daarvoor te hoeven schrijven. Zo kunnen deze properties aan labels in de GUI gekoppeld worden. De tekst in deze labels worden dan automatisch gewijzigd als de string in het domeinmodel wijzigt.

- Zoek uit hoe JavaFX properties werken (<http://docs.oracle.com/javase/8/javafx/properties-binding-tutorial/binding.htm#JFXBD107>) en 'bind' deze aan labels voor de spelersnaam en het berichtenlabel.

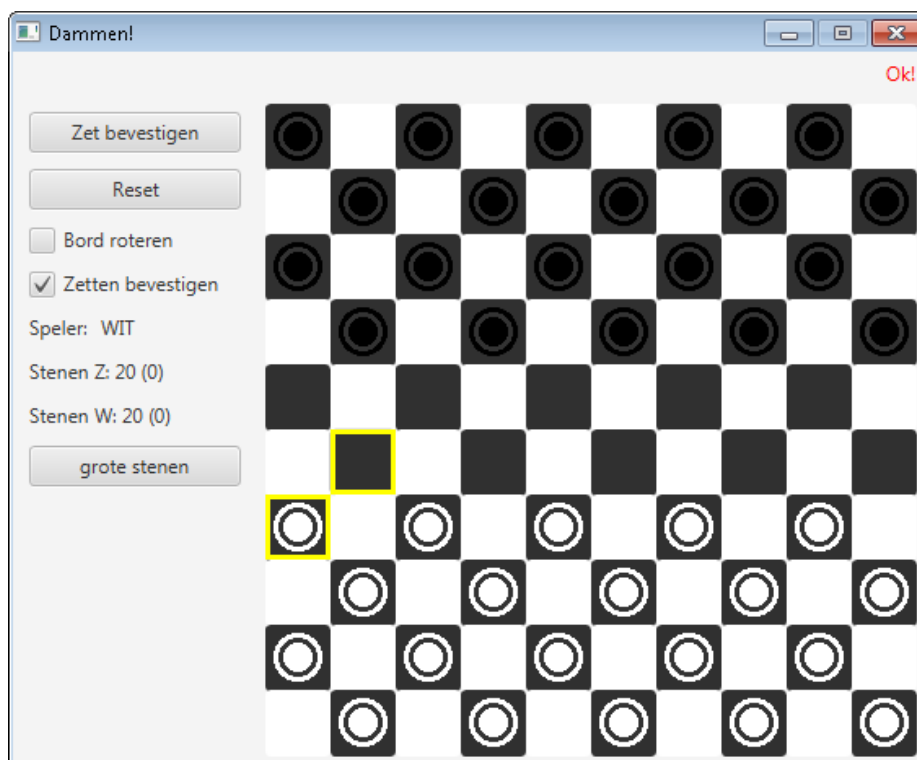
Practicumopdracht 2-6

Als je in je huidige oplossing een zet per ongeluk verkeerd aanklikt, zal de zet meteen uitgevoerd worden. Voor sommige spelers is dat niet prettig. Met methode `isZetMogelijk(int, int)` van klasse `Damspel` kun je controleren of een bepaalde zet of slag mogelijk is. Als dat zo is, moet er een border om het betreffende veld geplaatst worden. De speler kan dan vervolgens de zet bevestigen of het veld weer de-selecteren.

- **Programmeer de bevestigingsmogelijkheid.** Zie ook het screenshot in opdracht 2-7.

Practicumopdracht 2-7

Voeg een `ToggleButton` toe waarmee je de velden van het speelbord kunt vergroten. Zie onderstaande afbeelding voor een voorbeeld. Na het aanpassen van de preferred size van de buttons moet de methode `sizeToScene()` van de klasse `Stage` aangeroepen worden. Hiervoor is het nodig om de `Stage` – parameter in de methode `start()` als private attribuut in de klasse `DammenApp` op te slaan.



- **Programmeer de `ToggleButton`.**

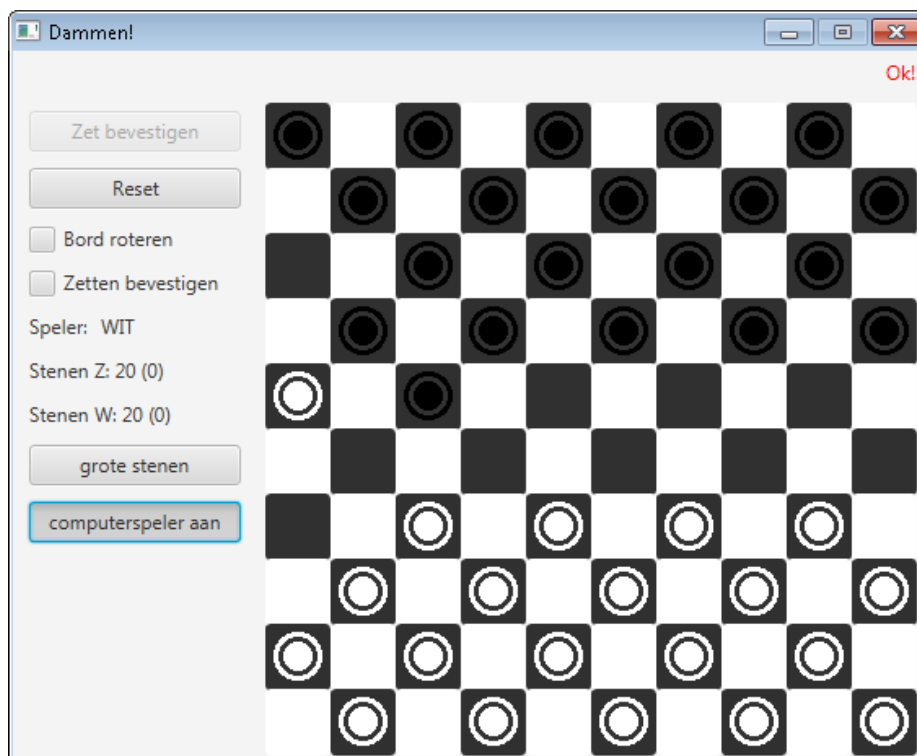
Practicumopdracht 2-8 (Extra uitdagend)

Klasse `Damspel` bevat naast de eerder genoemde methoden ook nog de onderstaande methoden. Met dit arsenaal aan publieke methoden is het mogelijk om een ‘intelligente’ computerspeler te programmeren.

```
public List<Integer> getAlleSlagPositiesVoorSpeler()  
public List<Integer> getAlleZetPositiesVoorSpeler()  
public List<Integer> bepaalMogelijkeZettenVoorVeld(int veldId)  
public List<Integer> bepaalMogelijkeSlagenVoorVeld(int veldId)  
public Damspel clone()
```

Methoden `getAlleSlagPositiesVoorSpeler()` en `getAlleZetPositiesVoorSpeler()` leveren beide een lijst met veldid's waarvandaan een slag of een zet gedaan kan worden. Daarna kan met de methoden `bepaalMogelijkeZettenVoorVeld(int)` en `bepaalMogelijkeSlagenVoorVeld(int)` gekeken worden naar welk veld een zet of slag uitgevoerd kan worden. Methode ‘`clone()`’ kan gebruikt worden om een kopie van het bord op te vragen om zo effecten van een zet te bepalen zonder het echte spel te beïnvloeden.

- **Programmeer een computerspeler. Het handigste is om een aparte klasse `ComputerSpeler` te maken en deze in de constructor van `DammenApp` aan te maken. Geef dit object het `Damspel` object mee. Zorg dat in deze klasse een listener is geprogrammeerd die kan luisteren naar change-events van de `Speler`-property van `Damspel`. Zolang speler "ZWART" aan de beurt is kan er een ‘intelligente’ actie ondernomen worden.**
- **Programmeer een methode `setEnabled(boolean)` in klasse `ComputerSpeler` waarmee de computerspeler middels een `ToggleButton` ‘aan’ of ‘uit’ gezet kan worden:**



3.Canvas

In de practicumopdrachten van hoofdstuk 3 en 4 zullen we een eenvoudig tekenprogramma programmeren. Het belangrijkste onderdeel van een tekenprogramma is het ‘tekenvel’ waarop je afbeeldingen en figuren kunt tekenen. Met JavaFX heb je daarvoor de klasse `Canvas` tot je beschikking. Deze klasse zullen we eerst nader bekijken zodat die gebruikt kan worden bij de opdrachten van dit hoofdstuk.

Een `Canvas` kan eenvoudig aan een GUI worden toegevoegd. In Listing 3.1 staat een zeer eenvoudig voorbeeld waarbij de GUI bestaat uit alleen een `FlowPane` met daarop een `Canvas`. Daarnaast is de gebruikelijke code nodig waarmee een `Stage` wordt getoond.

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.layout.FlowPane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class CanvasApp extends Application {
    public void start(Stage stage) {
        Canvas canvas = new Canvas(200, 200);

        FlowPane flowPanel = new FlowPane();
        flowPanel.getChildren().add(canvas);
        flowPanel.setPrefSize(200, 200);

        Scene scene = new Scene(flowPanel);
        stage.sizeToScene();
        stage.setScene(scene);
        stage.show();
    }

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

Listing 3-1: Leeg Canvas op FlowPane

In het programma (Listing 3-1) is het canvas echter nog helemaal leeg. Om op een canvas te kunnen tekenen moeten we de `GraphicsContext` van het canvas opvragen. Dat kan met de methode `getGraphicsContext()` gedaan worden:

```
GraphicsContext gc = canvas.getGraphicsContext2D();
```

Klasse `GraphicsContext` bevat zeer veel methoden om lijnen, figuren en afbeeldingen op het canvas te tekenen. Die kan je op <http://docs.oracle.com/javase/8/javafx/api/javafx/scene/canvas/GraphicsContext.html> vinden. In Listing 3-2 is het eerder gegeven voorbeeld uitgebreid met een aantal getekende figuren en een afbeelding.

```

import java.io.File;
import javafx.application.Application;
import javafx.scene.canvas.*;
import javafx.stage.Stage;
import javafx.scene.paint.Color;

import java.net.URI;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.layout.FlowPane;

public class CanvasApp extends Application {
    public void start(Stage stage) {
        Canvas canvas = new Canvas(200, 200);

        GraphicsContext gc = canvas.getGraphicsContext2D();
        gc.setFill(Color.BLUE);
        gc.fillRect(10, 10, 100, 25);
        gc.setFill(Color.RED);
        gc.strokeOval(130, 50, 50, 80);

        File bestand = new File("dog.jpg");
        URI uri = bestand.toURI();
        Image afbeelding = new Image(uri.toString());
        gc.drawImage(afbeelding, 10, 105, 100, 85);

        FlowPane flowPanel = new FlowPane();
        flowPanel.getChildren().add(canvas);
        flowPanel.setPrefSize(200, 200);

        Scene scene = new Scene(flowPanel);
        stage.sizeToScene();
        stage.setScene(scene);
        stage.show();
    }

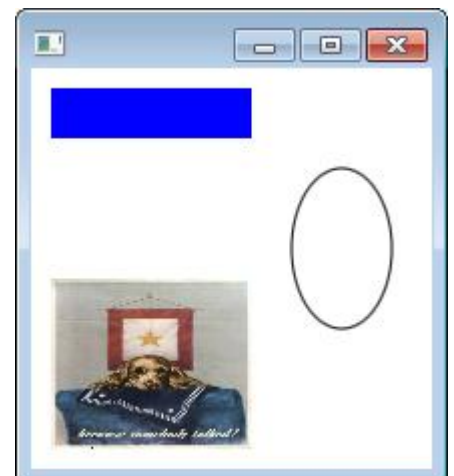
    public static void main(String[] args) {
        Application.launch(args);
    }
}

```

Listing 3-2: Canvas met figuren en afbeelding

Het resultaat is hiernaast weergegeven. De code voor de figuren komt sterk overeen met de code die in Swing gebruikelijk is om rechthoeken en ovals te tekenen. Het instellen van de kleur van een canvas wijkt echter wel iets af. Er zijn twee kleuren: de ‘fill-kleur’ en de ‘stroke-kleur’. De stroke-kleur is niet ingesteld, daarom is de cirkel zwart.

Het plaatsen van een afbeelding is iets ingewikkelder. Daarvoor maken we gebruik van diverse klassen en methoden. We zullen deze code regel voor regel bespreken op de volgende bladzijden.



Met klasse `File` kan gecommuniceerd worden met het bestandssysteem van het besturingssysteem. Je kunt de constructor van deze klasse aanroepen met de bestandsnaam als string. Dit bestand moet dan wel in het classpath van het programma staan:

```
File bestand = new File("dog.jpg");
```

Zodra het `File`-object is aangemaakt zouden we kunnen testen of het bestand wel bestaat.

- **Zoek in de documentatie van klasse `File` welke methode je daarvoor moet gebruiken!**

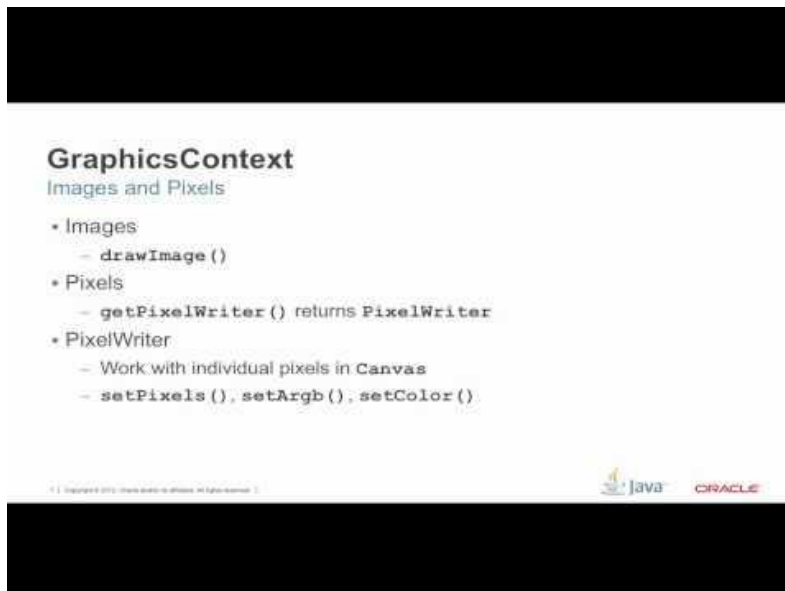
Van het `File`-object kan vervolgens de Uniform Resource Identifier worden opgevraagd. Dit is een adres dat volgens een standaard-protocol is opgebouwd. Met dit adres kan een `Image` object gemaakt worden:

```
URI uri = bestand.toURI();  
Image afbeelding = new Image(uri.toString());
```

Tot slot moet de afbeelding ook nog getoond worden op het canvas. Dat kan op twee manieren. De afbeelding kan op ware grootte getekend worden, maar je kunt ook de grootte op het canvas opgeven:

```
gc.drawImage(afbeelding, 10, 105); // ware grootte  
gc.drawImage(afbeelding, 10, 105, 100, 85); // grootte 100x85
```

- **Oracle heeft een aantal interessante voorbeelden in de JavaFX tutorial opgenomen. Bestudeer de Canvas-documentatie: <http://docs.oracle.com/javase/8/javafx/graphics-tutorial/canvas.htm>.**
- **Bekijk de onderstaande video om een idee te krijgen van de canvas-mogelijkheden.**



Canvas – path

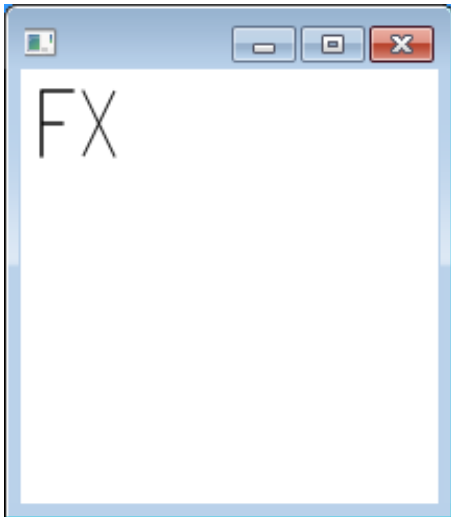
Een interessante mogelijkheid die de Oracle tutorial noemt is het creëren van paden. Het is in de tutorial ondermeer toegepast bij het maken van de letter ‘D’. Daarnaast kun je ook diverse locaties aan een path toevoegen zoals hieronder is gedaan:

```
GraphicsContext gc = canvas.getGraphicsContext2D();

gc.beginPath();
gc.moveTo(10, 10);          gc.lineTo(10, 40);
gc.moveTo(10, 10);          gc.lineTo(25, 10);
gc.moveTo(10, 25);          gc.lineTo(20, 25);
gc.stroke();

gc.moveTo(30, 10);          gc.lineTo(45, 40);
gc.moveTo(30, 40);          gc.lineTo(45, 10);
gc.stroke();

gc.closePath();
```



Wanneer de vetgedrukte code in Listing 3-2 vervangen zou worden door bovenstaande code levert dat de hiernaast getoonde screenshot op.

Nu is het maken van letters natuurlijk veel effectiever met de methode `strokeText(string, double, double)`. Echter, het maken van een pad is erg handig in combinatie met gebruikersinvoer.

Wanneer de gebruiker namelijk op een bepaalde locatie klikt, of naar een locatie sleept, hoeft alleen maar de nieuwe locatie aan het pad toegevoegd worden. Je hoeft dan niet te onthouden waar de vorige keer geklikt werd. Met methode `stroke()` kan vervolgens het gevolgde pad getekend worden. Als een pad volledig is (gebruiker laat bijvoorbeeld de muis los) kan deze afgeloten worden middels `gc.closePath()`.

Canvas – events

De canvas-tutorial bespreekt ook de interactie met de gebruiker van een programma. De interactie kan bestaan uit toetsenbordinvoer, maar kan ook in de vorm van muisacties komen. Al deze invoer van de gebruiker bestaat uit events. We hebben eerder al `ActionEvent`-objecten voorbij zien komen, maar daarnaast zijn er ook `KeyEvent`- en `MouseEvent`-objecten. Daarbij horen ook weer handlers die de code bevatten die uitgevoerd moet worden zodra zo’n event plaats heeft gevonden.

Er zijn verschillende manieren om een handler te koppelen aan een canvas. In de Oracle-tutorial voegt men handlers toe aan het canvas met de methode `addEventHandler(<eventtype>, <handler>)`. Deze werkwijze is prima, maar wijkt af van wat we tot nu toe gezien hebben. We kijken daarom nog naar 2 alternatieven die ook toegepast kunnen worden. Dit sluit ook aan op de derde Javales.

Event-methode

Zoals een button een `setOnAction()` methode heeft, zo hebben de meeste JavaFX control-items een uitgebreide set aan methoden voor elk event dat je voor de betreffende control kunt opvangen. Wanneer je bijvoorbeeld wilt detecteren dat iemand met de rechtermuisknop op het canvas klikt, kan daarvoor de methode `setOnMouseClicked()` gebruikt worden. Wanneer je veel verschillende events wilt kunnen afvangen kan het daarbij handig zijn om een anonieme innerclass toe te passen. Op die manier hoef je in de handler niet steeds met `event.getSource()` te controleren wat de 'source' van het event is:

```
canvas.setOnMouseClicked(new EventHandler<MouseEvent>() {  
    public void handle(MouseEvent event) {  
        if (event.getButton() == MouseButton.SECONDARY) {  
            System.out.println(event.getX() + ", " + event.getY());  
        }  
    }  
});
```

Lambda-functie

Je kunt ook gebruik maken van lambda-functies waarbij je een expliciete constructie van de innerclass achterwege laat:

```
canvas.setOnMouseClicked(event -> {  
    if (event.getButton() == MouseButton.SECONDARY) {  
        System.out.println(event.getX() + ", " + event.getY());  
    }  
});
```

De melding in de console kan vanzelfsprekend vervangen worden door nuttiger code. De methoden `getX()` en `getY()` zijn afkomstig van de klasse `MouseEvent`. De klasse bevat nog veel meer handige methoden voor mouseevent-verwerking: <http://docs.oracle.com/javase/8/javafx/api/javafx/scene/input/MouseEvent.html>.

Let op: Dit betreft de klasse `MouseEvent` in de JavaFX package. Er is ook een `MouseEvent` klasse voor Swing-componenten!

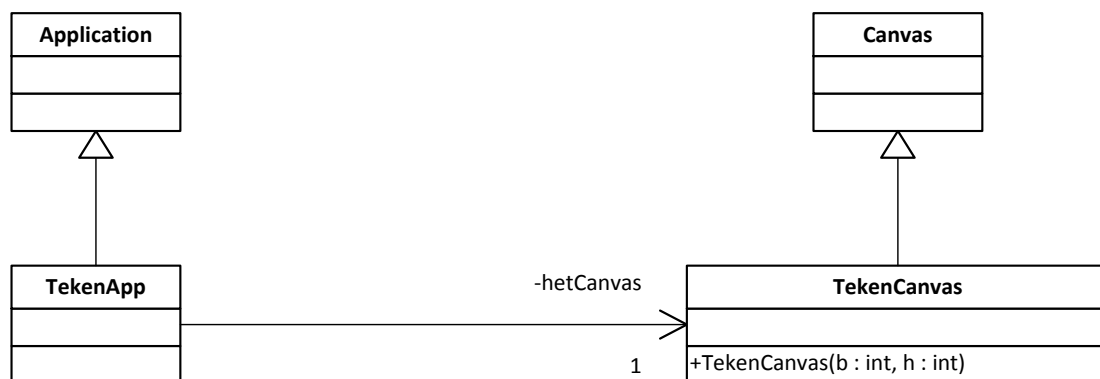
Exit-quiz

- Kan je een Canvas gebruiken als eerste (parent) node en direct op een Scene plaatsen?
- Je kan niet rechtstreeks op een Canvas tekenen. Welk object gebruik je daarvoor?
- Welke methode gebruik je om dat object van het Canvas op te vragen?
- Welke verschillende vormen (zoals 'oval') kan je 'filled' tekenen?
- Hoe kan je diezelfde vormen niet 'filled' tekenen?
- Met welke methode kan je de dikte van een lijn instellen?
- Klasse File heeft een methode om te controleren of een bestand wel bestaat. Welke is dat?
- In de Oracle Canvas-tutorial is in figuur 9-1 een halfopen cirkel getekend. Hoe is dat gedaan?
- Welke verschillende soorten mouse-events kunnen er op het Canvas verwerkt worden?
- Klasse MouseEvent bevat een methode om te achterhalen welke muisknop is gebruikt. Welke is dat?
- Hoe weet je op welke x en y posities een gebruiker heeft geklikt?
- Wat is een 'path' op het Canvas?

Practicumopdracht 3-1

Tijdens de opdrachten van hoofdstuk 3 werken we toe naar een eenvoudig tekenprogramma. Het moet mogelijk worden om een tekening te maken die bestaat uit vrije vormen, lijnen, vierkanten en cirkels. Deze moeten getekend kunnen worden in een kleur die door de gebruiker is geselecteerd. In de eerste opdracht volstaan we met het aanmaken van een GUI waarin later getekend kan worden.

Om te kunnen tekenen op een canvas is er gebruikersinvoer nodig. Om die invoer te verwerken maken we een eigen versie (uitbreiding) van de de klasse `Canvas`. Het UML-diagram ziet er dan als volgt uit:



Je hoeft van deze klassen dus alleen `TekenApp` en `TekenCanvas` zelf te programmeren. Klasse `TekenApp` maakt een `TekenCanvas` aan en plaatst deze in het centrum van een `BorderPane`.

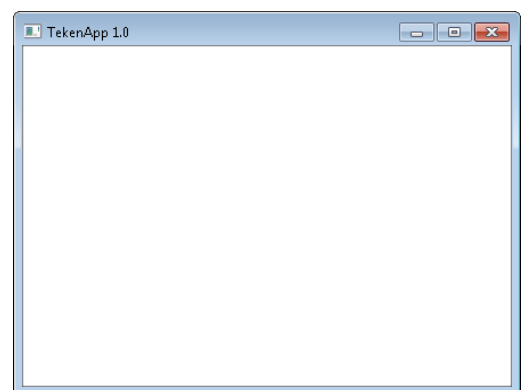
Programmeer in klasse `TekenCanvas` een constructor waarmee je de grootte van een canvas kunt opgeven. Maak gebruik van super-constructing (aanroepen van de constructor van de super-klasse) om de grootte van het canvas in te stellen:

```
public TekenCanvas(int breedte, int hoogte) {
    super(breedte, hoogte);
}
```

Tips bij dit scherm (het is niet verplicht om het zo aan te pakken):

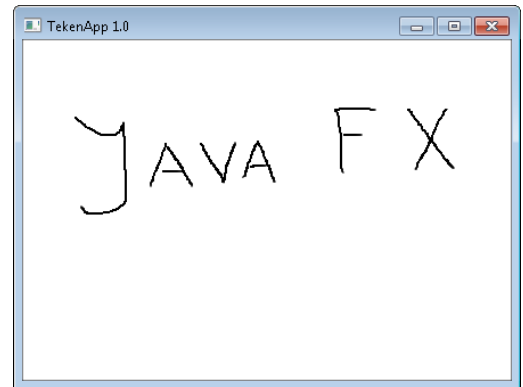
- Er is een `BorderPane` als root-node in de `Scene` gebruikt.
- Het `TekenCanvas` heeft een grootte van 430 x 300.
- Er is een rechthoek getekend op de randen van het canvas.
- Het `TekenCanvas` staat in het 'center' van de `BorderPane`.

➤ **Programmeer de klassen `TekenApp` en `TekenCanvas`.**



Practicumopdracht 3-2

Tot nu toe kan de gebruiker nog helemaal niets met het canvas. Daar brengen we verandering in door het tekenen van vrije vormen mogelijk te maken. Een vrije vorm kan getekend worden door met de linkermuisknop ingedrukt over het canvas te slepen. Daarvoor moeten we een aantal mouse-events af te vangen.



- **Programmeer handlers voor deze events in de klasse `TekenCanvas` en koppel deze aan het canvas (`this`). Doe dit met (anonieme) innerclasses of lambdafuncties.**

We onderscheiden 3 events die daarvoor belangrijk zijn:

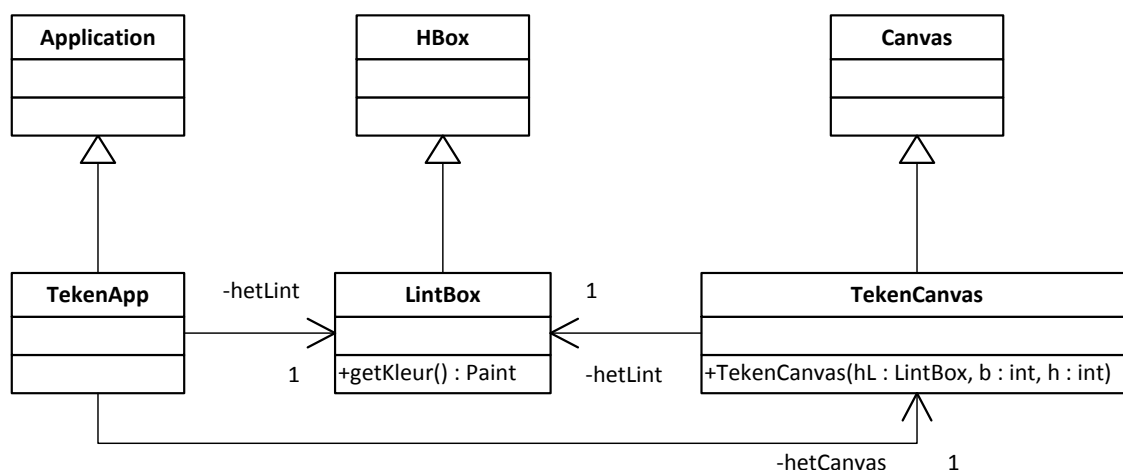
MousePressed: Bij het indrukken van de linkermuisknop moet het pad beginnen.

MouseDragged: Bij slepen van de muis (met linkermuisknop) moet het pad uitbreiden en getekend worden.

MouseReleased: Bij loslaten van de linkermuisknop moet het pad eindigen en getekend worden.

Practicumopdracht 3-3

De gebruiker moet bij deze opdracht ook de mogelijkheid krijgen om kleuren te kiezen waarmee getekend kan worden. Daarvoor maken we speciaal een `LintBox` (vergelijkbaar met het 'lint' in MS Office) waarmee we de kleurkeuze van de gebruiker gaan vastleggen. Het UML-diagram komt er dan als volgt uit te zien:



De klasse `TekenApp` maakt een `LintBox` object aan voordat het `TekenCanvas` gemaakt wordt. Bij het aanmaken van het `TekenCanvas` moet 'hetLint' meegegeven worden aan de constructor van klasse `TekenCanvas`. Hiervoor moet dus de constructor van `TekenCanvas` uitgebreid worden!

`LintBox` extend `HBox` en bevat voor deze opdracht een `ColorPicker` en 1 `Rectangle`. Die kan je gewoon in de constructor van `LintBox` aanmaken en toevoegen aan het `LintBox`. Beiden hebben in de screenshots een breedte van 100. De `Rectangle` is 22 hoog.

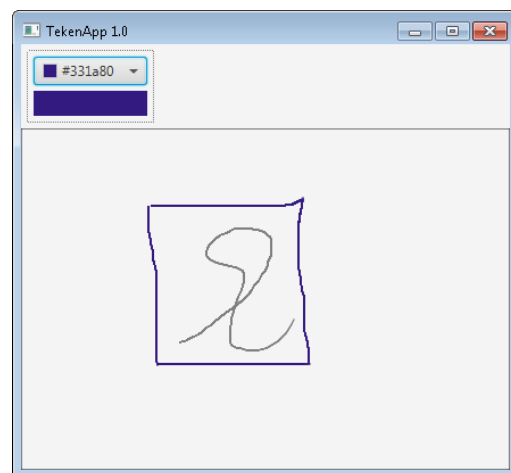
Op <http://docs.oracle.com/javase/8/javafx/user-interface-tutorial/color-picker.htm> kan je meer informatie over de `ColorPicker` vinden. <http://docs.oracle.com/javase/8/javafx/api/javafx/scene/shape/Rectangle.html> geeft meer informatie over de `Rectangle`. Een `Rectangle` kan op dezelfde manier geplaatst worden als een `Label` of `Button`, maar kan eenvoudig een kleur gegeven worden met methode `setFill()`.

Wanneer de gebruiker een nieuwe kleur kiest moet de `Rectangle` ook deze kleur krijgen. Dat kan je heel compact doen met een `ActionEvent`-handler in de vorm van een lambda functie (**niet verplicht**):

```
colorPicker.setOnAction(e ->
    rectangle.setFill(colorPicker.getValue()));
```

Zodra de gebruiker gaat tekenen, moet de huidige kleur aan het `Lint` worden opgevraagd. Hiervoor moet je een methode `getKleur()` opnemen in klasse `LintBox`! Deze kan de fill-kleur van het rechthoek returnen.

- **Breid de applicatie van opdracht 3-2 uit zoals in deze opdracht is beschreven!**
- **Zorg ervoor dat de `LintBox` in klasse `TekenApp` in de 'top' van het `BorderPane` komt te staan.**



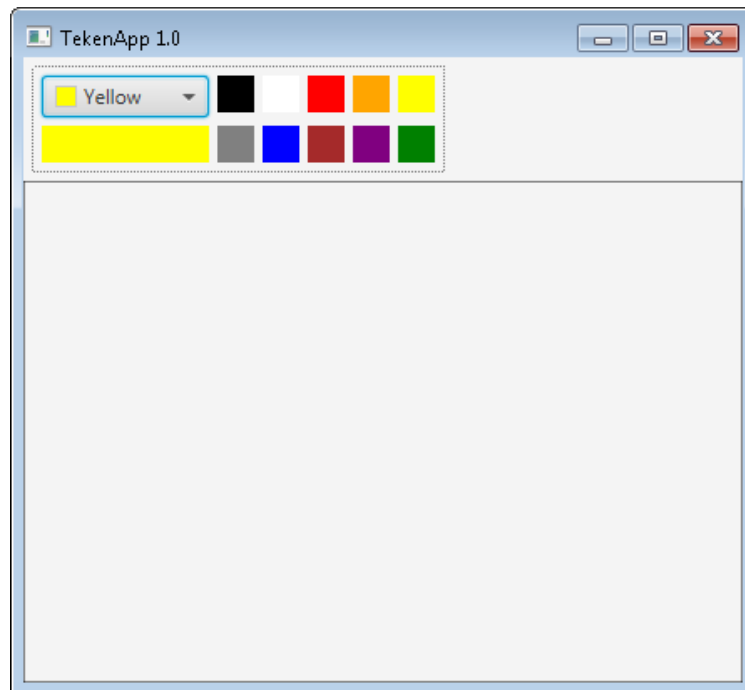
Practicumopdracht 3-4

Voor basiskleuren is het niet zo handig om steeds de `ColorPicker` te gebruiken. Daarom nemen we in het lint 10 `Rectangles` op met basiskleuren. Als de gebruiker op deze rechthoeken klikt, dan moet de kleur van het rechthoek uit opgave 3-3 gewijzigd worden, maar ook de kleur van de `ColorPicker`.

- **Programmeer 10x een `Rectangle` met een aantal basiskleuren. Koppel er een actionhandler aan waarmee je de kleur van de `ColorPicker` en het eerste rechthoek wijzigt.**

Tips bij deze opgave:

- Voor de 10 basiskleuren is in het gegeven screenshot een `TilePane` gebruikt.
- Een `TilePane` is niet de enige mogelijkheid om de basiskleuren in te plaatsen.
- Maak een `Color`-array aan en gebruik een `for`-lus om de rechthoeken te maken.



Practicumopdracht 3-5

Naast vrije vormen moet er ook een mogelijkheid komen om rechthoeken, cirkels en lijnen te tekenen. Daarvoor moet het lint uitgebreid worden met een viertal radiobuttons waarmee een vormkeuze gemaakt kan worden.

- Programmeer in klasse `LintBox` vier `RadioButtons` en voeg deze toe aan een `ToggleGroup`.
- Neem in `LintBox` een enum op waar de vier mogelijke vormen worden gedefinieerd. Zie onder.
- Programmeer in `LintBox` methode `getVorm()` waarmee de huidige vorm gevraagd kan worden.

Enums zijn voorgedefinieerde constanten van een eigen type. Als je een variabele `String s` hebt, dan kan aan 's' elke mogelijke tekst worden toegekend. Maar bij een enum geef je op welke mogelijke waarden aan een variabele van dat type gegeven kunnen worden. Neem bijvoorbeeld deze enum:

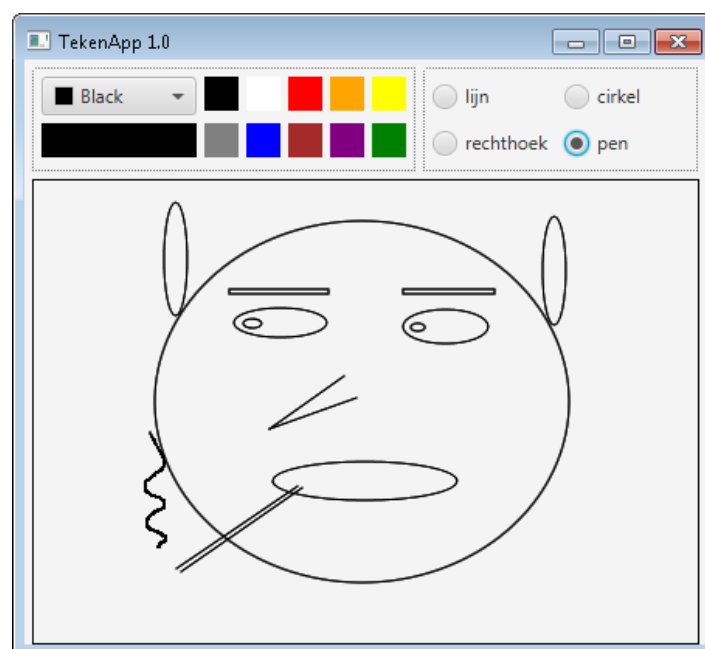
```
public enum Vorm { LIJN, RECHTHOEK, CIRKEL, PEN };  
  
private Vorm vorm1 = Vorm.LIJN;  
private Vorm vorm2 = Vorm.DRIEHOEK;
```

Variabele `vorm2` mag zo niet aangemaakt worden omdat `DRIEHOEK` niet voorkomt in de enum. Met een enum weet je dus altijd zeker dat een variabele maar een bepaald aantal waarden kan hebben, en zo kan je veel fouten in je programma voorkomen. Daarnaast wordt je programma ook veel leesbaarder dan wanneer je bijvoorbeeld integers zou gebruiken (bijvoorbeeld 1 = lijn, 2 = rechthoek etc.).

Voor meer info over enums: <http://docs.oracle.com/javase/tutorial/java/javaOO/enum.html>.

Radiobuttons: <http://docs.oracle.com/javase/8/javafx/user-interface-tutorial/radio-button.htm>.

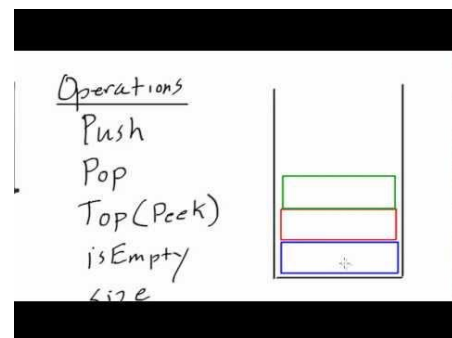
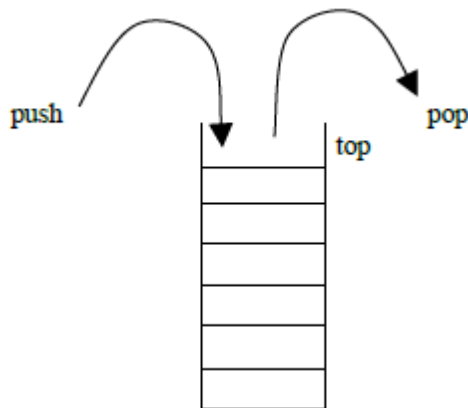
- Pas klasse `TekenCanvas` aan zodat aan de hand van de ingestelde vorm op het lint de juiste vorm getekend zal worden. In deze opdracht is het voldoende als lijnen, rechthoeken en cirkels pas zichtbaar worden zodra de gebruiker de linkermuisknop loslaat.



Practicumopdracht 3-6

Zodra de gebruiker nu een fout maakt bij het tekenen, moet diegene de gehele tekening opnieuw maken. Dat is niet zo handig. We gaan daarom elke keer net voordat de gebruiker gaat tekenen, de afbeelding opslaan. Wanneer de gebruiker na een actie op de rechtermuisknop klikt, moet de laatstopgeslagen afbeelding getoond worden.

In feite is bij een ‘ongedaan maken’ actie sprake van het LIFO principe: Last In, First Out. Dus de laatste opgeslagen wijziging moet als eerste ongedaan gemaakt worden. Daarvoor gebruiken we een `Stack` oftewel stapel. Zie voor een uitgebreide toelichting op de `Stack` de onderstaande video.



Java kent standaard al de `Stack`-structuur: <http://docs.oracle.com/javase/8/docs/api/java/util/Stack.html>. De belangrijkste methoden van een `stack` zijn:

- `empty()`: controleert of de stapel leeg is
- `peek()`: geeft het bovenste item van de stapel, maar deze wordt niet van de stapel verwijderd
- `pop()`: geeft het bovenste item van de stapel, deze wordt ook verwijderd van de stapel
- `push(.)`: plaatst een item bovenop de stapel

Op een `Stack` kan je van alles plaatsen, maar in dit geval moet er een stapel van canvas-snapshots gemaakt worden. Daartoe kan je het volgende attribuut in de klasse `TekenCanvas` opnemen:

```
private Stack<WritableImage> undoStack = new Stack<WritableImage>();
```

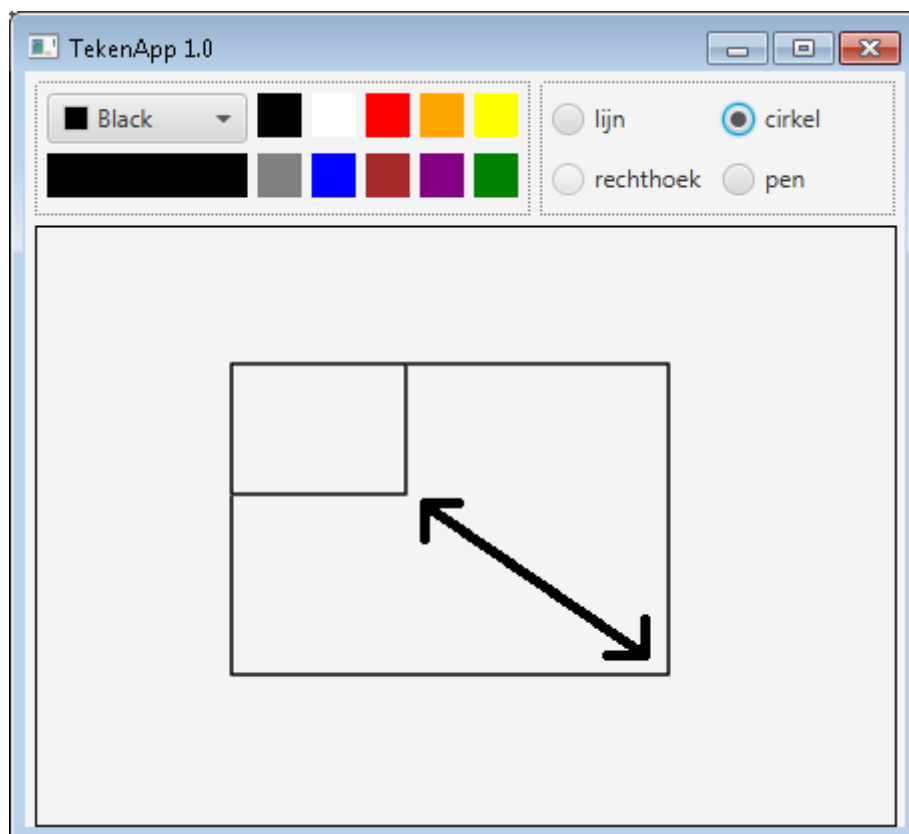
We moeten nu alleen nog afbeeldingen op de stapel plaatsen. Klasse `Canvas` biedt de mogelijkheid tot het nemen van snapshots (bestudeer de methode `snapshot(..)` in de documentatie). Maak daar gebruik van en plaats steeds voorafgaand aan een nieuwe tekenactie een snapshot op de stack. Deze snapshots worden altijd opgeleverd in de vorm van een `WritableImage`-object, die dus op de ‘`undoStack`’ geplaatst kunnen worden.

- **Programmeer de ‘ongedaan maken’ mogelijkheid in klasse `TekenCanvas`. Schrijf een nieuwe handler die bij een `MouseClicked`-event (rechtermuisknop) de laatste afbeelding van de stack op het canvas plaatst. Vergeet niet het canvas eerst leeg te maken!**

Practicumopdracht 3-7 (vervolg van 3-6)

Nu de mogelijkheid bestaat om een vorige afbeelding te herstellen, kan er ook voor gezorgd worden dat cirkels, rechthoeken en lijnen meebewegen bij het tekenen. Zolang er `MouseDown` events optreden moet telkens de laatste afbeelding getekend en overschreven worden. Zo kan de gebruiker zien waar bijvoorbeeld een rechthoek precies in een tekening komt te staan. Wanneer de gebruiker de linkermuisknop loslaat moet de vorm op de definitieve positie getekend worden.

- **Programmeer de hierboven beschreven functionaliteit.**



4. Menus, Files en Dialogs

De meeste applicaties, op vrijwel alle platformen, hebben de mogelijkheid om extra acties uit te voeren die niet direct in het zichtbare gedeelte (GUI) van de applicatie zijn opgenomen. Het kan zijn dat die acties er op dat moment niet toe doen, maar het kan ook zijn dat de interface te druk zou worden als alle mogelijke acties tegelijkertijd toegankelijk zouden zijn. Een veelgebruikte manier om subacties te ontsluiten is via een menu.

Een menu kan in verschillende vormen voorkomen. De menubalk is een menusoort die in de meeste desktop-programma's wel voorkomt, al is die niet altijd permanent meer zichtbaar. Maar de menubalk (menubar) is niet de enige mogelijkheid, je kunt in veel programma's ook een menu oproepen door met de rechtermuisknop te klikken in een programma. Dit zijn de zogeheten 'contextmenu's'. Een voorbeeld uit de Oracle-documentatie die beide vormen combineert is hieronder opgenomen:

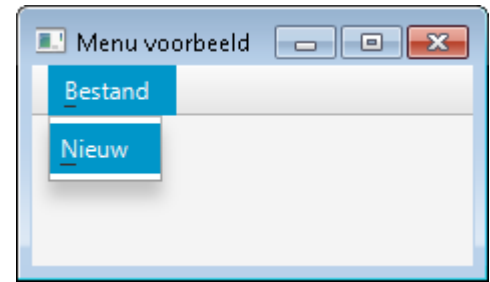


De inhoud van het menu kan afhangen van de staat van het programma. Als je bijvoorbeeld net een Java-klasse in Eclipse hebt geopend kan je het bestand niet opslaan voordat je een wijziging in de klasse hebt aangebracht. Tot die tijd is de menuoptie 'Opslaan' niet beschikbaar.

Zeker bij een contextmenu is niet alleen de status van je programma van invloed op het menu, maar ook de plaats waar je met de rechtermuisknop hebt geklikt. In bovenstaande afbeelding is er bijvoorbeeld de mogelijkheid van het kopiëren van de afbeelding. Wanneer de gebruiker met de rechtermuisknop op de tekst had geklikt had dit menu natuurlijk andere menuitems moeten bevatten die logisch zouden zijn als je tekst wilt bewerken.

MenuBar, Menu & MenuItem

Een menu is in JavaFX opgebouwd uit menu's die menuitems en eventuele submenu's kunnen bevatten. Er ontstaat zo als het ware een boomstructuur. De menu's en menuitems worden geplaatst in een root- of parentobject. Bij een gewone menubalk is de `MenuBar` de root van alle menu's, maar dat kan ook een `ContextMenu` zijn. In Listing 4-1 is een eenvoudig programma gemaakt met als enige menuonderdeel een menubalk en 1 menuitem.



```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class MenuVoorbeeld extends Application {
    public void start(Stage stage) {
        BorderPane panel = new BorderPane();

        MenuBar menuBar = new MenuBar();
        Menu menuBestand = new Menu("_Bestand");

        MenuItem miBestNieuw = new MenuItem("_Nieuw");
        miBestNieuw.setOnAction(e -> nieuwBestandAanmaken() );

        menuBestand.getItems().addAll(miBestNieuw);
        menuBar.getMenus().addAll(menuBestand);

        panel.setTop(menuBar);
        Scene scene = new Scene(panel, 220, 100);
        stage.setScene(scene);

        stage.setTitle("Menu voorbeeld");
        stage.show();
    }

    private void nieuwBestandAanmaken() {
        System.out.println("hier komt functionaliteit");
    }
}
```

Listing 4-1

De menuitems en menu's kunnen op een vergelijkbare wijze worden toegevoegd als gewone UI-controls:

```
menuBestand.getItems().addAll(miBestNieuw);
menuBar.getMenus().addAll(menuBestand);
```

De menubalk (`MenuBar`) kan op diverse plaatsen op het `BorderPane` geplaatst worden. Gebruikelijk is echter om deze het bovenste element van de GUI te laten zijn, met methode `setTop(menuBar)`.

Naast gewone menuitems zijn er bijvoorbeeld ook `RadioMenuItems`. Daarnaast kan je in een menu ook een tweede menu plaatsen. Dit zal dan vervolgens als submenu getoond worden.

- **Welke soorten menuitems kan je in JavaFX toepassen?** Zie ook de documentatie van klasse `MenuItem`: <http://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/MenuItem.html>.
- **Hoe kan je volgens die documentatie een eigen menuitem maken?**
- **Bestudeer de Oracle Menu-documentatie op** http://docs.oracle.com/javase/8/javafx/user-interface-tutorial/menu_controls.htm.

MneMonics & Accelerators

Een menu is met een muis goed te benaderen, maar het is ook erg handig om het menu via het toetsenbord toegankelijk te maken. Als je bepaalde menu's veel gebruikt kan een handeling via het toetsenbord veel tijd schelen omdat je niet steeds je handen naar de muis hoeft te bewegen.

Mnemonics

Met mnemonics kan je menu's en menuitems benaderen via de Alt-toets (Windows). Een mnemonic is zeer eenvoudig aan een menu of menuitem te koppelen door een underscore '_' in de menutekst te verwerken:

```
Menu menuBestand = new Menu("_Bestand");
```

Aan menu 'Bestand' is nu de mnemonic Alt + B toegekend. Hetzelfde kan toegepast worden op menuitems:

```
MenuItem miBestNieuw = new MenuItem("_Nieuw");
```

Als je een mnemonic koppelt aan een menuitem, dan werkt deze alleen maar als er ook aan het menu een mnemonic is gekoppeld. In dit geval is het mnemonic-pad Alt + B, N (zie ook Listing 4-1).

Accelerators

In sommige gevallen gebruik je een menuitem zo vaak dat je eigenlijk een directe sneltoets wilt koppelen aan een actie. Bekende voorbeelden hiervan zijn Ctrl+S (Opslaan), Ctrl+C (Kopiëren), Ctrl+V (Plakken), Ctrl+Z (Ongedaan maken) etc. Dit soort koppelingen zijn accelerators. Deze zijn iets moeilijker te maken dan mnemonics. Om menuitem 'Nieuw' van Listing 4-1 te koppelen aan Ctrl+N is de volgende code nodig:

```
miBestNieuw.setAccelerator(  
    new KeyCodeCombination(KeyCode.N, KeyCombination.CONTROL_DOWN)  
);
```

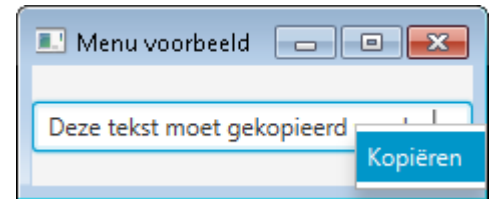
Hiervoor zijn de klasse `KeyCode`, `KeyCombination` en `KeyCodeCombination` nodig. De laatstgenoemde koppelt een `KeyCode` aan een `KeyCombination`. Klasse `KeyCode` bevat voor de toetsen van je toetsenbord een constante (zoals `KeyCode.N`) die je bijvoorbeeld kunt gebruiken om een accelerator te maken. In dit geval wordt de toets 'N' gekoppeld aan de Ctrl-toets.

- **Zoek in de documentatie van `KeyCombination` welke toetsen nog meer gebruikt kunnen worden voor een `KeyCodeCombination`.**

ContextMenu

Een `ContextMenu` kan net als een `MenuBar` ook bestaan uit menu's en menuitems. Een verschil is dat klasse `ContextMenu` geen methode `getMenus()` heeft. Het toevoegen van menu's en menuitems gaat altijd via de methode `getItems()`.

In JavaFX hebben een aantal UI-controls standaard de mogelijkheid om er met methode `setContextMenu()` een contextmenu aan toe te voegen. Een `TextField` is daar een voorbeeld van zoals hiernaast en in Listing 4-2 is weergegeven.



```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.input.*;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class MenuVoorbeeld extends Application {
    private TextField tf = new TextField();

    public void start(Stage stage) {
        BorderPane panel = new BorderPane();
        ContextMenu contextMenu = new ContextMenu();

        MenuItem miCopy = new MenuItem("Kopiëren");
        miCopy.setOnAction(e -> kopieerNaarClipboard());
        contextMenu.getItems().addAll(miCopy);

        tf.setContextMenu(contextMenu);

        panel.setCenter(tf);
        Scene scene = new Scene(panel, 220, 100);
        stage.setScene(scene);

        stage.setTitle("Menu voorbeeld");
        stage.show();
    }

    private void kopieerNaarClipboard() {
        ClipboardContent content = new ClipboardContent();
        content.putString(tf.getText());
        Clipboard.getSystemClipboard().setContent(content);
    }
}
```

Listing 4-2

- Zoek uit (<http://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Control.html>) welke subklassen er van klasse `Control` zijn. Die klassen erven dan de methode `setContextMenu()`. Hoe kan het dat `TextField` geen subklasse van `Control` is maar wel die methode heeft?

Helaas kan je niet overal zomaar een contextmenu aan koppelen. Klasse `BorderPane` biedt die mogelijkheid bijvoorbeeld niet standaard. Je kunt dat echter nog wel zelf programmeren met een mousehandler:

```
panel.setOnMouseClicked(e -> {  
    if (e.getButton() == MouseButton.SECONDARY) {  
        contextMenu.show(panel, e.getScreenX(), e.getScreenY());  
    }  
});
```

- Welke methode in klasse `ContextMenu` kan je gebruiken om het menu weer te verbergen?
- Welke ‘`setOn...()`’ methode kan je gebruiken zodat je niet hoeft te controleren of er wel op de rechtermuisknop is geklikt? Wat heeft dit te maken met het besturingssysteem?

FileChoosers

In veel programma's zijn er in het menu mogelijkheden opgenomen om bestanden te openen of op te slaan. Daarbij is interactie met het besturingssysteem en het bestandssysteem nodig. In Linux zijn er echter andere aanroepen naar het besturingssysteem nodig dan in Windows. Het zou lastig zijn als je in je programma zelf hiervoor de code zou moeten schrijven. Daarom bieden de meeste GUI-libraries een standaard manier aan om te communiceren met het bestandssysteem. JavaFX doet dat via de klasse `FileChooser`.

Het openen van een `FileChooser` dialoogvenster is eenvoudig:

```
FileChooser fileChooser = new FileChooser();  
fileChooser.setTitle("Open");  
File openFile = fileChooser.showOpenDialog(stage);
```

De eerste twee regels van deze code wijzen voor zich, maar de laatste regel heeft wat uitleg. De methode `showOpenDialog(.)` kan aangeroepen worden om de `FileChooser` te tonen. Op dat moment **blokkeert** de methode waarin deze code staat totdat de gebruiker het dialoogvenster weer sluit. Het bestand dat door de gebruiker is geselecteerd wordt toegekend aan variabele ‘`openFile`’. Dat bestand kan dan in het programma verder worden verwerkt (ingelezen). Het kan echter ook zo zijn dat de gebruiker het dialoogvenster heeft gesloten zonder een bestand te selecteren. In dat geval is de waarde van ‘`openFile`’ nog steeds `null`.

- Bestudeer op <http://docs.oracle.com/javase/8/javafx/user-interface-tutorial/file-chooser.htm> de voorbeelden en documentatie van `FileChooser`.
- Welke methode gebruik je om een `Save-FileChooser` te tonen? Zijn er nog meer soorten?
- Hoe kan je ervoor zorgen dat alleen `.PNG` bestanden in de `FileChooser` worden getoond?
- Bestudeer de documentatie van methode `showOpenDialog(.)` en zoek uit wat het effect is van het meegeven van de parameter `stage` als owner van het dialoogvenster.

Dialogvensters

Behalve de `FileChooser` biedt JavaFX in de huidige versie helaas nog geen libraries om snel aangepaste dialogvensters te maken. Wanneer je bijvoorbeeld snel een pop-up melding aan de gebruiker wilt tonen moet je hiervoor een eigen dialogvenster programmeren. Vanaf JavaFX update 40 moet dit in de API beschikbaar worden. In tussentijd programmeren we een eigen klasse `Alert` zoals in Listing 4-3 is te zien.

```
import javafx.geometry.*;          import javafx.scene.Scene;
import javafx.scene.control.*;      import javafx.scene.layout.VBox;
import javafx.stage.*;

public class Alert extends Stage {
    public Alert(Stage owner, String bericht) {
        super(StageStyle.UTILITY);
        initOwner(owner);
        initModality(Modality.WINDOW_MODAL);
        this.setResizable(false);
        this.setTitle(bericht);

        Label melding = new Label(bericht);
        melding.setAlignment(Pos.BASELINE_CENTER);
        melding.setPrefWidth(250);

        Button ok = new Button("Ok");
        ok.setPrefWidth(75);
        ok.setOnAction(e -> this.hide());

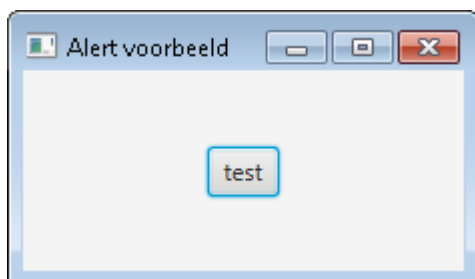
        VBox controlBox = new VBox(10);
        controlBox.setAlignment(Pos.CENTER);
        controlBox.getChildren().addAll(melding, ok);
        controlBox.setPadding(new Insets(15));

        Scene scene = new Scene(controlBox);
        this.setScene(scene);
    }
}
```

Listing 4-3

`Alert` is een subklasse van `Stage`. Daarop kan je net als op een gewone `Stage` een `Scene` met een panel plaatsen. Op dat panel kunnen buttons, labels textfields enz. toegevoegd worden. De button ‘Ok’ zal hier bij aanklikken met de methode `hide()` de `Alert` weer sluiten. Je kunt nu een melding aan de gebruiker geven:

```
Alert al = new Alert(stage, "Wilt u dit bevestigen?");
al.showAndWait();
```



Aanvullende info bij Listing 4-3:

- ☛ De vormgeving (wel of niet een sluit-knop, wel of niet een titelbalk etc.) kan je instellen via de `StageStyle`. Deze is in dit voorbeeld ingesteld op `UTILITY`.
- ☛ Methode `showAndWait()` is een methode die al standaard in klasse `Stage` aanwezig is. Met deze methode kan je ervoor zorgen dat de applicatie waar je die code aanroept **blokkeert** totdat het dialoogvenster wordt gesloten.
- ☛ Door een owner (eigenaar) aan een `Alert` te koppelen met methode `initOwner()`, zorg je ervoor dat beide vensters door je besturingssysteem als een eenheid worden gezien.
- ☛ Als het dialoogvenster een eigenaar heeft, kan je met methode `initModality()` ervoor zorgen dat de gebruiker niets op het onderliggende scherm kan doen zolang het dialoogvenster open staat.

De klasse `Alert` kan je nu gebruiken als basis om ook andere dialoogvensters te maken:

- **Voeg een extra `Button` toe aan `Alert`.**
- **Maak een extra `int` (of `enum`) aan in klasse `Alert` met de naam `keuze` en waarde `0`.**
- **Zorg ervoor dat wanneer de gebruiker op linkerknop klikt, `keuze` de waarde `-1` krijgt. Als de gebruiker op de rechterknop klikt, moet `keuze` de waarde `1` krijgen. Bij indrukken van een knop moet daarna het dialoogvenster gesloten worden.**
- **Programmeer methode `int geefKeuze()` waarmee je de gebruikerskeuze kan opvragen.**
- **Test het nieuwe dialoogvenster.**

Exit-quiz

- Welke soorten menuitems kan je in JavaFX toepassen?
- Hoe kan je een eigen menuitem maken?
- Als je `RadioMenuItems` gebruikt, hoe zorg je dan dat je maar 1 menuitem tegelijk kunt selecteren?
- Je kunt menuitems ook 'uit' zetten. Dan zijn ze wel zichtbaar maar niet aanklikbaar. Hoe moet dat?
- Wat is een `SeparatorMenuItem`?
- Welke `KeyCombination`-toetsen (naast `Ctrl`) kan je voor een `KeyCodeCombination` gebruiken?
- Welke methode in klasse `ContextMenu` kan je gebruiken om het menu weer te verbergen?
- Wat moet je doen om een `FileChooser` in een bepaalde directory te laten starten?
- Welke methode gebruik je om een `Save-FileChooser` te tonen? Zijn er nog meer soorten?
- Hoe kan je ervoor zorgen dat alleen `.PNG` bestanden in de `FileChooser` worden getoond?
- Hoeveel `StageStyle` typen zijn er in te stellen voor een `Stage`? Wat zijn de verschillen?
- Welke soorten modaliteit zijn er naast `WINDOW_MODAL`?
- Naast `showAndWait()` is er methode `show()` om een `Stage` te tonen. Wanneer zou je die gebruiken?

Practicumopdracht 4-1

In de TekenApp van hoofdstuk 3 kan een gebruiker tekeningen maken, maar je kunt de tekening nog niet opslaan of een nieuwe tekening beginnen. Tijdens de opdrachten van hoofdstuk 4 gaan we werken aan een menu(balk) waarin de gebruiker diverse acties kan kiezen.

Om te beginnen moet het mogelijk worden om een nieuwe tekening te starten. Daarvoor dient er een `MenuBar` met een `Menu` 'Bestand' opgenomen te worden. Daarin komt een `MenuItem` 'Nieuw'. Aan een `MenuItem` kan je net zoals aan een gewone button een handler koppelen met de methode `setAction()`. Gebruik hiervoor een anonieme handler! Dat is handig omdat er nog meer menu's geprogrammeerd moeten worden. Door anonieme handlers te gebruiken hoef je niet steeds te controleren welk menuitem is gekozen. Zie ook http://docs.oracle.com/javase/8/javafx/user-interface-tutorial/menu_controls.htm.

Het menu gaan we eenvoudig toegankelijk maken door mnemonics te gebruiken. Hiermee kan je het menu ook goed met het toetsenbord benaderen. Voeg daartoe de volgende mnemonics toe:

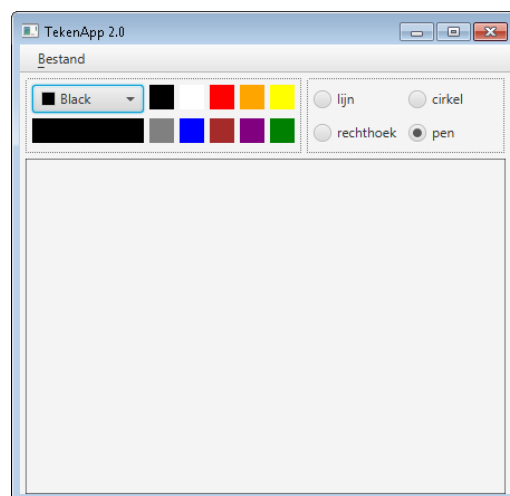
- Alt + B:** opent menu 'Bestand'
- Alt + B, N:** opent menu 'Bestand' en kiest optie 'Nieuw'.

Sommige menuitems gebruik je zoveel dat je daarvoor eigenlijk een aparte sneltoets-combinatie voor wilt hebben. Dat zijn zogeheten accelerators. Koppel de volgende accelerators aan menuitem 'Nieuw':

- Ctrl + N:** kiest zonder het menu te openen voor optie 'Nieuw'

Zoals in de screenshots te zien is, plaatst JavaFX de accelerator achter het menuitem zodra je de toetscombinatie hebt opgegeven. Dit gaat automatisch. Ook mnemonics worden na instellen automatisch met een streepje onder de juiste letter aangewezen.

- **Plaats de `MenuBar` samen met de `LintBox` bovenin de top van het `BorderPanel`. Je kunt altijd maar 1 onderdeel in een `BorderPanel`-positie plaatsen, dus je moet ze samen eerst in een `VBox` zetten en deze `VBox` op het `BorderPanel` plaatsen.**
- **Programmeer de hierboven beschreven functionaliteit. Zorg dat wanneer de gebruiker menuitem 'Nieuw' kiest, er door de handler een nieuw `TekenCanvas` met dezelfde afmetingen in centrum van het `BorderPanel` geplaatst zal worden.**



Practicumopdracht 4-2

Het is wel zo handig als de gebruiker ook de kans heeft om een eerdere tekening op te slaan voordat hij of zij aan een nieuwe tekening begint. We maken daarom in deze opdracht de menuitems ‘Opslaan’ en ‘Opslaan Als’ in het menu ‘Bestand’. Beide menu’s verschillen qua werking vrij weinig van elkaar:

Menu ‘Opslaan’: Slaat een tekening direct op als deze al eens eerder is opgeslagen. Als dat niet het geval is zal er een Opslaan-dialoogvenster getoond worden.

Menu ‘Opslaan Als’: Toont altijd een Opslaan-dialoogvenster.

Neem het volgende attribuut in klasse `TekenApp` op om bij te houden of een bestand al eens is opgeslagen:

```
private File huidigBestand = null;
```

Omdat beide menuacties zo weinig van elkaar verschillen, proberen we zo min mogelijk dubbele code te programmeren. Maak in klasse `TekenApp` methode `afbeeldingOpslaan(boolean dialoogAltijdTonen)`:

```
private boolean afbeeldingOpslaan(boolean dialoogAltijdTonen) {  
    ...  
}
```

De methode zal een Opslaan-dialoogvenster tonen als `huidigBestand` (nog) `null` of `dialoogAltijdTonen` `true` is. Als de gebruiker een bestand kiest moet `huidigBestand` daarnaar verwijzen. In dat geval kan de tekening opgeslagen worden in `huidigBestand`. De methode retourneert een of de afbeelding daadwerkelijk is opgeslagen (`true/false`). **Let op:** de gebruiker kan de `FileChooser` sluiten zonder een bestand te selecteren!

Een tekening van het canvas moet nog omgezet worden naar een afbeelding die eenvoudig naar een bestand geschreven kan worden. Deze code is vrij lastig en is daarom hieronder gegeven. Let op dat deze code nog een `IOException` kan opleveren. Pas daarom een try-catch blok toe in methode `afbeeldingOpslaan()`.

```
SnapshotParameters params = new SnapshotParameters();  
WritableImage img = tekenCanvas.snapshot(params, null);  
RenderedImage img2 = SwingFXUtils.fromFXImage(img, null);  
ImageIO.write(img2, "png", huidigBestand);
```

Koppel de menu’s met de volgende lambda-functies aan de nieuwe methode:

```
menuItemOpslaan.setOnAction(e -> afbeeldingOpslaan(false));  
menuItemOpslaanAls.setOnAction(e -> afbeeldingOpslaan(true));
```

- **Programmeer beschreven functionaliteit en maak een accelerator voor menuitem ‘Opslaan’.**
- **Zorg bij het tonen van de dialoogvensters dat Stage de ‘owner’ is van het venster.**
- **Zie hoofdstuk 4 en <http://docs.oracle.com/javase/8/javafx/user-interface-tutorial/file-chooser.htm>.**

Practicumopdracht 4-3

Wanneer een tekening is opgeslagen, moet deze ook weer geopend kunnen worden. Hiervoor kan opnieuw klasse `FileChooser` gebruikt worden. Zodra de gebruiker een bestand heeft geselecteerd moet er een nieuw `TekenCanvas` gemaakt worden. De afmetingen moeten overeenkomen met de grootte van de afbeelding die geopend moet worden.

Voor het openen en tekenen van een afbeelding op het canvas is hieronder een stuk van de code in **Listing 3-2** nogmaals opgenomen. Zie voor het volledige voorbeeld en een toelichting **Listing 3-2**:

```
File bestand = new File("dog.jpg");
URI uri = bestand.toURI();
Image afbeelding = new Image(uri.toString());
gc.drawImage(afbeelding, 10, 105, 100, 85);
```

In dit voorbeeld is het bestand altijd "dog.jpg" maar dat moet uiteraard vervangen worden door het bestand dat de gebruiker in de `FileChooser` heeft geselecteerd!

Omdat het openen van een bestand slechts bij 1 menuitem hoort, hoeft hiervoor niet perse een aparte methode te worden gemaakt die je voor meerdere menuitems kunt gebruiken zoals bij het opslaan in opdracht 4-2. Je kan het openen van een bestand ook geheel in een handler (innerclass of lambda-functie) programmeren. Beredeneer wat volgens jou het beste is.

➤ **Programmeer menuitem 'Open' in menu 'Bestand' en koppel hieraan accelerator Ctrl+O.**

Tip 1: Je zult merken dat als je een afbeelding opent die groter is dan het scherm, de `Stage` niet meeschaalt naar de juiste grootte. Zodra je een afbeelding hebt geopend kan je de volgende code gebruiken om het scherm automatisch de juiste grootte te geven:

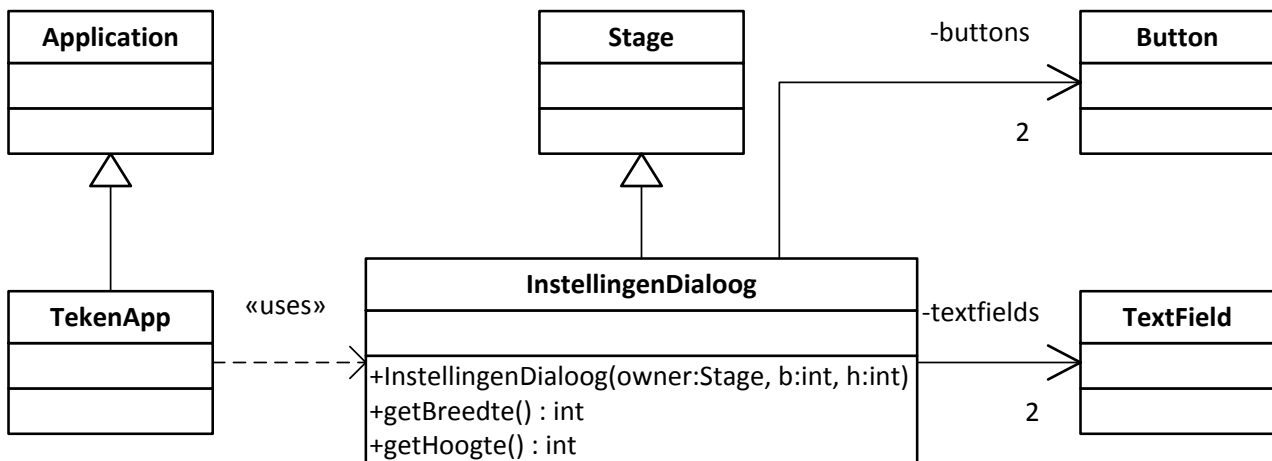
```
stage.sizeToScene();
```

Tip 2: Je kunt voor deze opdracht volstaan met wijzigingen in klasse `TekenApp`. Klassen `TekenCanvas` en `LintBox` kunnen ongewijzigd blijven.

Practicumopdracht 4-4

In hoofdstuk 4 is een `Alert` gemaakt door de klasse `Stage` uit te breiden. In deze opgave maken we een klasse `InstellingenDialog` waarmee de grootte van de afbeelding gewijzigd kan worden.

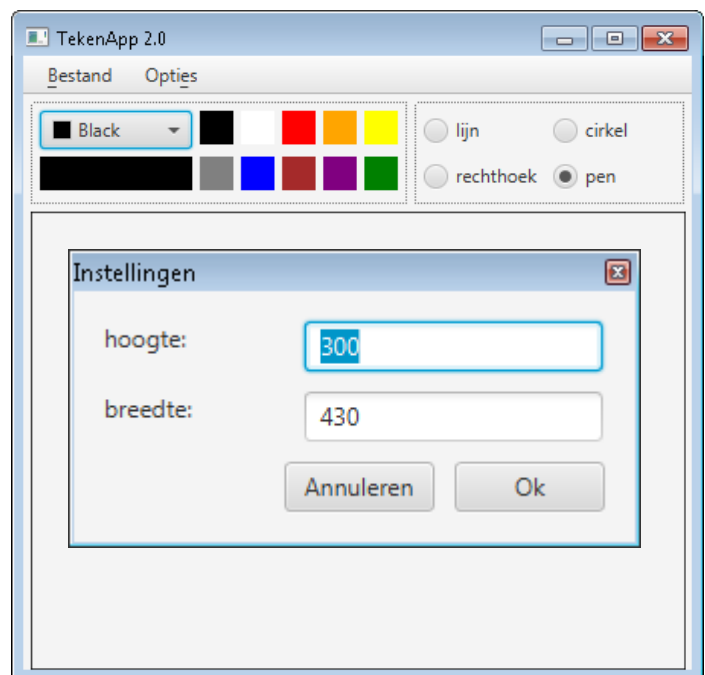
Klasse `InstellingenDialog` bevat 2 buttons; 'Annuleren' en 'Ok'. Daarnaast zijn er ook 2 `TextFields` waarin de grootte en de breedte van de tekening opgeslagen kan worden. Het UML ziet er als volgt uit:



Klasse `TekenApp` creëert een nieuwe `InstellingenDialog` en toont deze middels het (nieuwe) menuitem 'Opties', 'Instellingen'. Methode `showAndWait()` van klasse `Stage` gebruik je om het venster te tonen.

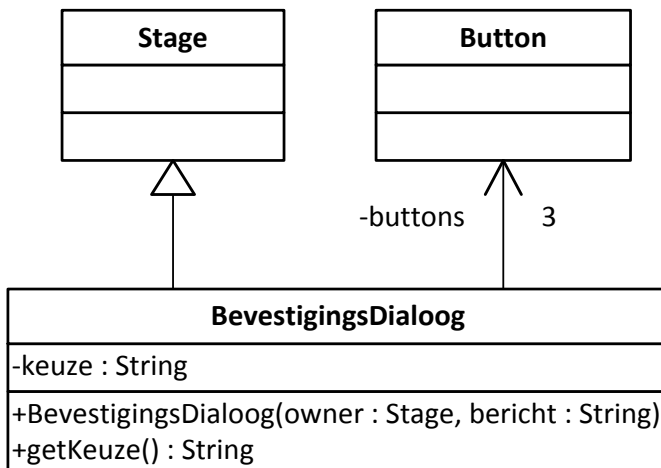
Standaard moeten de breedte en hoogte van de huidige tekening worden getoond in de textfields. De gebruiker kan deze wijzigen en het scherm sluiten met 'Ok' of 'Annuleren'. Daarna kan je de breedte en hoogte opvragen met methoden `getBreedte()` en `getHoogte()`. Klasse `Canvas` bevat al methoden om de hoogte en breedte daarop in te stellen. Zie ook het voorbeeld in hoofdstuk 4.

➤ **Programmeer dit dialoogvenster.**



Practicumopdracht 4-5

In de huidige situatie kan de gebruiker per ongeluk kiezen voor het starten van een nieuwe tekening zonder dat wijzigingen in de oude tekening zijn opgeslagen. Daarvoor gaan we een `BevestigingsDialog` maken. Deze dialog bevat altijd drie knoppen; “Annuleren”, “Nee” en “Ja”. Het dialogvenster kan verschillende vragen of boodschappen aan de gebruiker doorgeven. Daarvoor is er bij de constructor parameter `bericht` opgenomen:



Klasse `TekenApp` krijgt een extra methode, `boolean wijzigingenAfgehandeld()`. Deze methode toont het dialogvenster aan de gebruiker. Dan zijn er diverse mogelijkheden:

- De gebruiker kiest ‘Nee’, dan doet de methode niets en levert `true` op.
- De gebruiker kiest ‘Annuleren’, dan doet de methode niets en levert `false` op.
- De gebruiker kiest ‘Ja’, dan wordt de eerder geprogrammeerde methode `afbeeldingOpslaan(.)` gebruikt om de tekening op te slaan. Het resultaat daarvan wordt ook het resultaat van deze methode.

- **Creëer klasse** `BevestigingsDialog`.
- **Programmeer methode** `wijzigingenAfgehandeld()` **in** `TekenApp`.
- **Gebruik deze methode zodra de gebruiker een nieuwe tekening wil maken.**

Optionele opdrachten:

- **Gebruik deze methode als de gebruiker het programma af wil sluiten** (`setOnCloseRequest`). Hoe kan je afsluiten voorkomen als de gebruiker annuleert? Zie methoden van `WindowEvent`.
- **Pas de methode toe op de momenten dat jij dat handig vindt** (bijv. openen v.e. tekening).
- **Maak in** `TekenCanvas` **een methode waarmee je kunt opvragen of er wijzigingen zijn om te voorkomen dat er ook dialogvensters getoond worden als er nog niets is getekend.**

Practicumopdracht 4-6

Als laatste toevoeging aan de applicatie maken we een `ContextMenu` waarmee het hele canvas gewist kan worden. Een contextmenu is in de meeste applicaties aan de rechtermuisknop gekoppeld. Welk menu je dan te zien krijgt hangt af van waar de muisaanwijzer zich bevindt (de context).

Zie de stof van hoofdstuk 4 voor een voorbeeld van een `ContextMenu`.

- **Creëer een contextmenu en koppel dit aan het `TekenCanvas`. Programmeer dit menu in klasse `TekenCanvas` en niet, zoals de andere menu's, in `TekenApp`! Maak minimaal 1 menuitem waarmee je het canvas kunt leegmaken (methode `clearRect(...)`).**
- **Als je in een eerdere opdracht de 'ongedaan maken' functionaliteit hebt gekoppeld aan de rechtermuisknop. Koppel deze functionaliteit nu aan een menuitem ('Bewerken', 'Ongedaan maken') met als accelerator `Ctrl + Z`. Dat menuitem staat natuurlijk wel in `TekenApp`, dus je hebt in `TekenCanvas` een methode nodig om een actie ongedaan te maken.**

