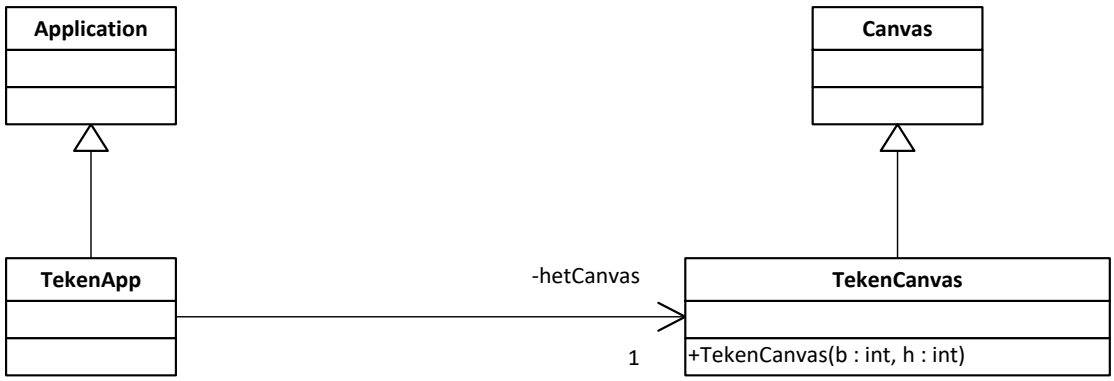
PAINT LES 1

PRACTICUM 1

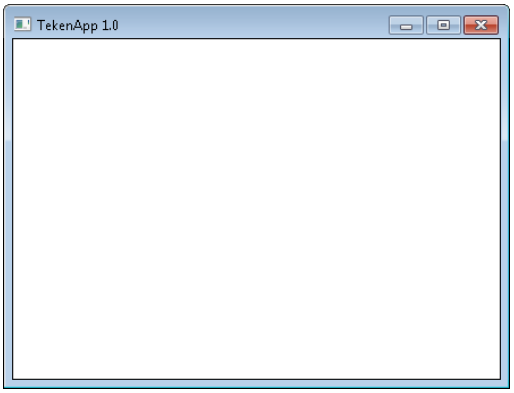
Tijdens de practica van deze les werken we toe naar een eenvoudig tekenprogramma waarin je een tekening moet kunnen maken die bestaat uit vrije vormen, lijnen, vierkanten en cirkels. Dat moet kunnen in een kleur die door de gebruiker is geselecteerd. In de eerste opdracht maken we de GUI waarin later getekend kan worden. Om te kunnen tekenen op een canvas is er gebruikersinvoer nodig. Voor het verwerken van die invoer maken we een eigen versie (uitbreiding) van klasse `Canvas`. Het UML-diagram ziet er zo uit:



Je moet van deze klassen alleen `TekenApp` en `TekenCanvas` zelf programmeren. Klasse `TekenApp` maakt een `TekenCanvas` aan en plaatst deze op het scherm.

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 `Canvas`) om de grootte van het tekengebied in te stellen:

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



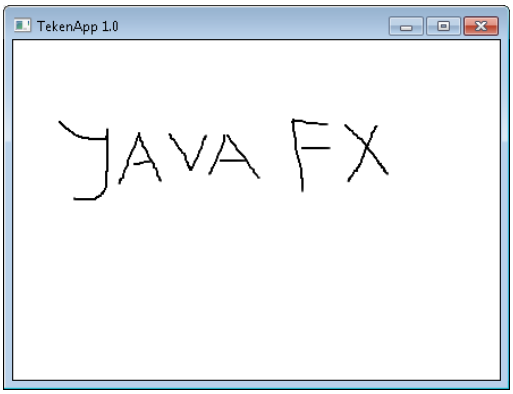
Tips bij dit scherm:

- Het `TekenCanvas` is 430 x 300 pixels groot.
- Er is een `BorderPane` als root-node in de `Scene` gebruikt.
- In het 'centrum' van de `BorderPane` staat een `StackPane` met daarop een `Rectangle` en een `TekenCanvas`. Dit is gedaan zodat de rechthoek bij resizen van het scherm toont wat het tekengebied is. De rechthoek is even groot als het canvas en heeft als stroke-kleur 'black' en als fill-kleur 'transparent'.

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

PRACTICUM 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.



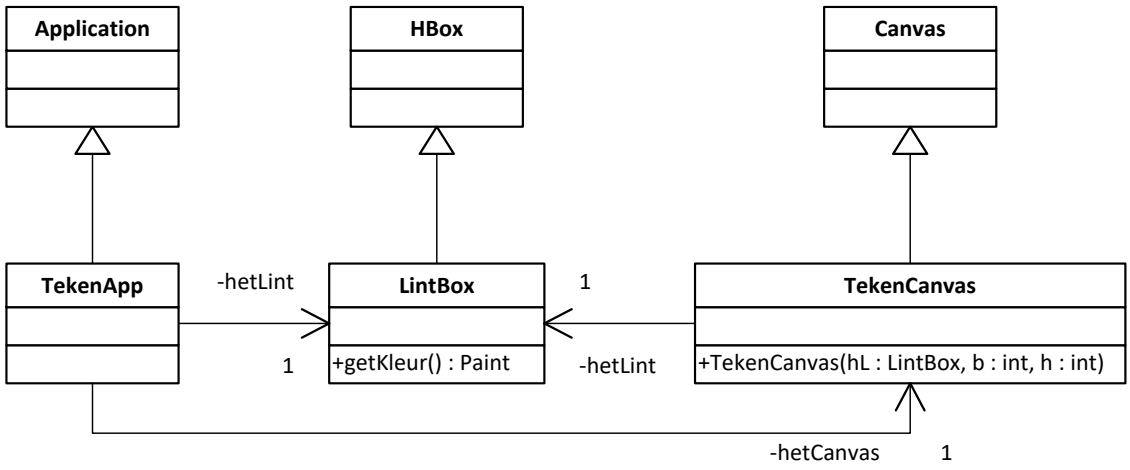
➔ **Programmeer handlers voor deze events in de klasse `TekenCanvas` en koppel deze aan het canvas (`this`). Doe dit met (anonieme) innerclasses of lambdafuncties.**

We onderscheiden 3 events die daarvoor belangrijk zijn:

`MousePressed`:	Bij het indrukken van de linkermuisknop begint het pad.
`MouseDragged`:	Slepen met de muis breidt het pad uit en toont het tussenresultaat.
`MouseReleased`:	Loslaten van de linkermuisknop eindigt het pad en toont het resultaat.

PRACTICUM 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 wordt als volgt uitgebreid:



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` zelf. Beiden hebben in de screenshots een breedte van 100. De `Rectangle` is 22 pixels hoog.

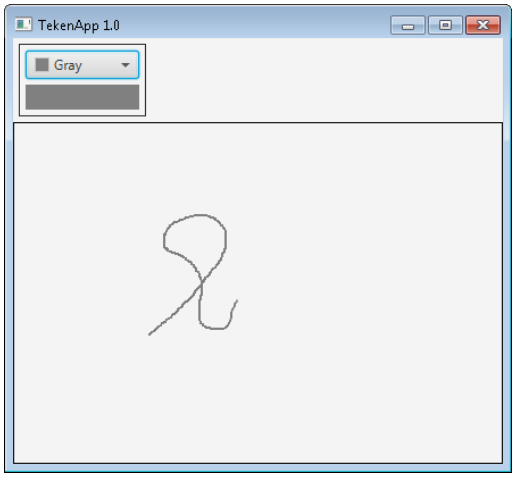
De documentatie bevat meer informatie over de `ColorPicker` en `Rectangle`. Een `Rectangle` kan op dezelfde manier geplaatst worden als een `Label` of `Button`, maar kan eenvoudig een kleur gegeven worden met methode `setFill(.)`.

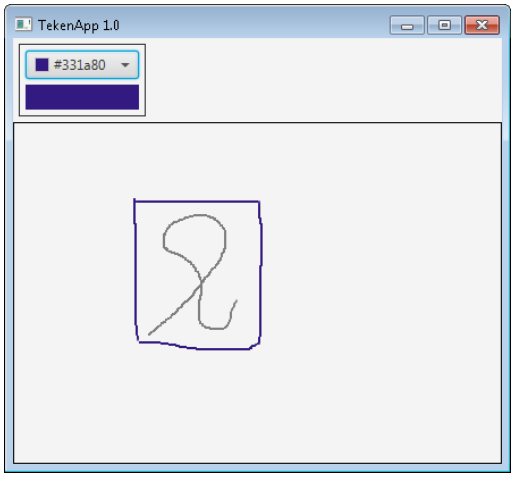
Wanneer de gebruiker een nieuwe kleur kiest moet de `Rectangle` ook deze kleur krijgen. Dat kan je heel compact doen met een `ActionEvent`-handler in de vorm van een lambda functie (**niet verplicht**):

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

Zodra de gebruiker gaat tekenen, moet de huidige kleur aan `hetLint` 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!**
➔ **Plaats de `LintBox` in de 'top' van het `BorderPane` van klasse `TekenApp`.**





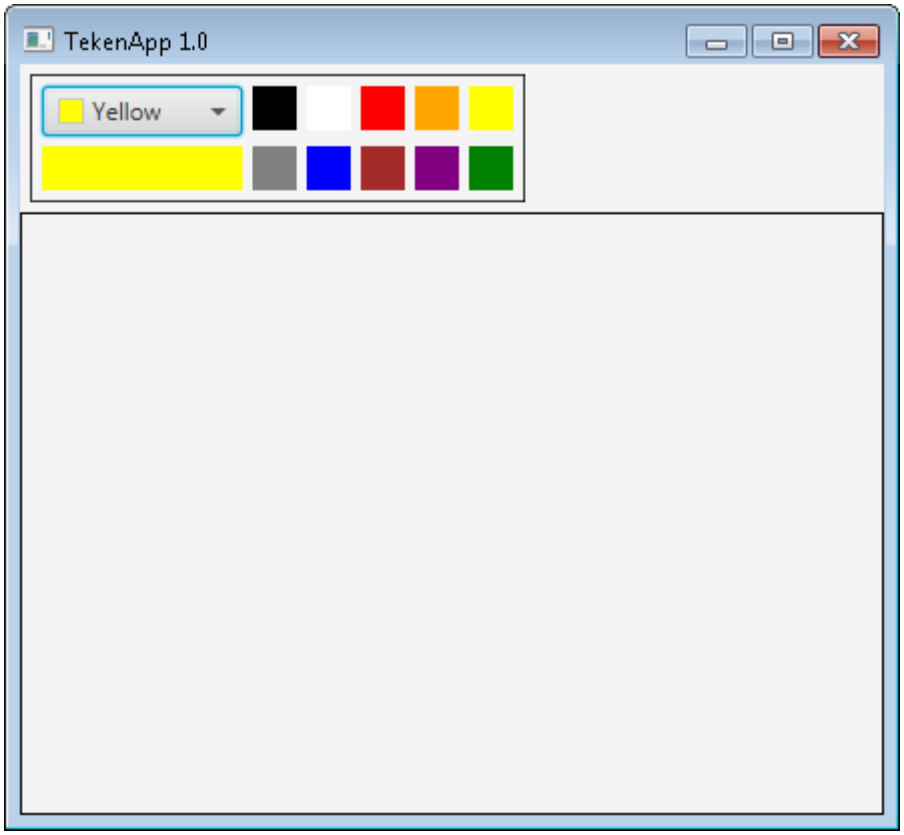
PRACTICUM 4

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

➔ **Programmeer 10 rectangles 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 of lijst en gebruik een for-lus om de rechthoeken te maken.



PRACTICUM 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 togglebuttons waarmee een vormkeuze gemaakt kan worden.

➔ **Programmeer in `LintBox` vier `ToggleButtons` en voeg deze toe aan een `ToggleGroup`.**
➔ **Plaats in `LintBox` een enum met de vier mogelijke vormen. Zie onder.**
➔ **Voeg in `LintBox` methode `getVorm()` toe om de huidige `Vorm` te kunnen opvragen.**

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.).

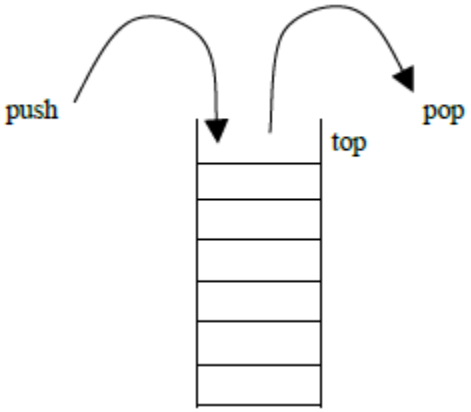
Lees meer over enums en Togglebuttons. Je kunt aan een togglebutton ook een graphic toevoegen met de constructor. Dit kan een bijvoorbeeld een `Rectangle` zijn, zie screenshot!

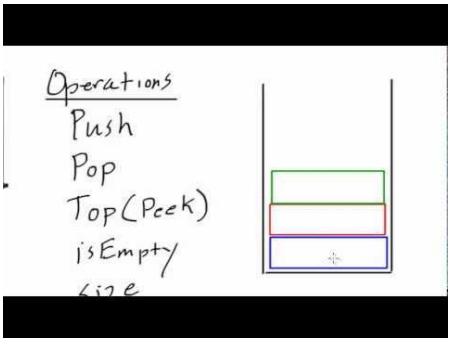
➔ **Pas klasse `TekenCanvas` aan zodat aan de hand van de ingestelde vorm op het lint de juiste vorm getekend zal worden. In deze opdracht is het voldoende als lijnen, rechthoeken en cirkels pas zichtbaar worden als de gebruiker de linkermuisknop loslaat.**

PRACTICUM 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:. De belangrijkste methoden van een stack zijn:

- `push(.)`: plaatst een item bovenop de stapel
- `peek()`: geeft het bovenste item van de stapel, deze blijft op de stapel
- `pop()`: geeft het bovenste item van de stapel, deze verdwijnt van de stapel
- `empty()`: controleert of de stapel leeg is

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.

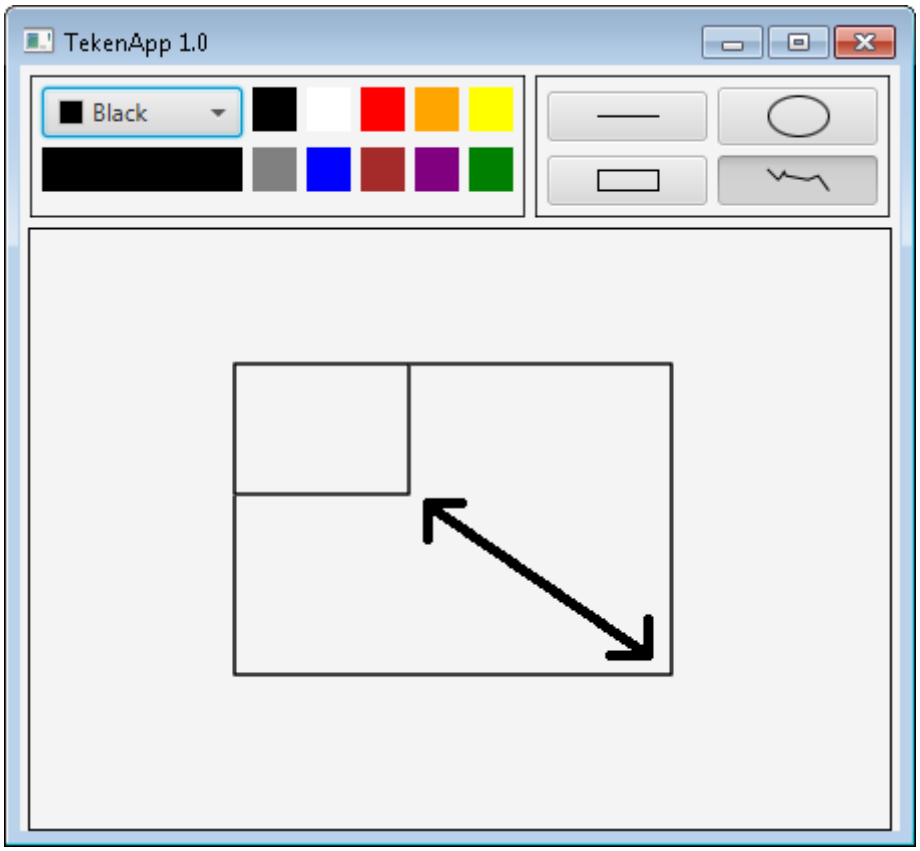
➔ **Programmeer de 'ongedaan maken' mogelijkheid in klasse `TekenCanvas`. Schrijf een nieuwe handler die bij een mouseclicked-event (rechtermuisknop) de laatste afbeelding van de stack op het canvas plaatst. Vergeet niet het canvas eerst leeg te maken!**

PRACTICUM 7

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. Dat kan door zolang er `MouseDragged` events optreden telkens de laatste afbeelding (de situatie voordat de huidige tekenactie begon) opnieuw te tekenen daarop de vorm te tekenen op de positie waar de gebruiker zijn muis dan naartoe heeft bewogen.

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.**



PAINT LES 2

PRACTICUM 1

In de eerder gemaakte TekenApp kunnen tekeningen gemaakt worden, maar je kunt die bijvoorbeeld nog niet opslaan of een nieuwe tekening beginnen. We gaan daarom een menu(balk) maken waarin je diverse acties kunt kiezen.

Als eerste moet het een nieuwe tekening gestart kunnen worden. Daarvoor moet een `MenuBar` met een `Menu` 'Bestand' opgenomen worden. Daarin komt een `MenuItem` 'Nieuw'. Aan een `MenuItem` kan je ook weer een handler koppelen met methode `setAction(.)`. Gebruik een anonieme handler. Dat is handig omdat er nog meer menu's geprogrammeerd moeten worden en zo hoef je niet telkens te kijken welk menuitem er is gekozen. Zie docs!

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

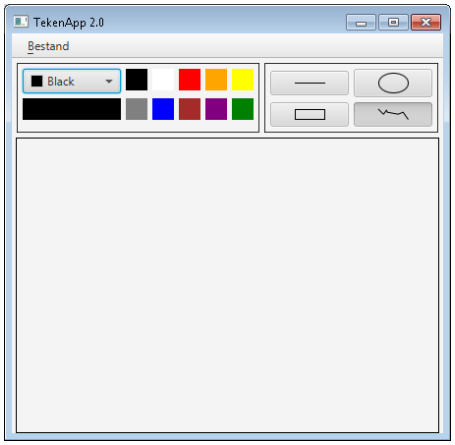
Alt + B: opent menu 'Bestand'
Alt + B, N: opent menu 'Bestand' en kiest optie 'Nieuw'.

Sommige menuitems gebruik je zo vaak dat een aparte sneltoets-combinatie dan erg handig is. Dat zijn zogeheten accelerators. Koppel de volgende accelerator aan menuitem 'Nieuw':

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

De accelerator wordt achter het menuitem geplaatst zodra je de toetscombinatie hebt opgegeven.

Mnemonics worden 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 bovenop de `StackPane` in het centrum van het scherm geplaatst zal worden.**

PRACTICUM 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 zo verschijnt er een Opslaan-dialoogvenster.

Menu 'Opslaan Als': Toont altijd een Opslaan-dialoogvenster.

Gebruik een attribuut in klasse `TekenApp` om bij te houden of een bestand al 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 returnt een of de afbeelding daadwerkelijk is opgelagen (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 deze functionaliteit en maak een accelerator voor menuitem 'Opslaan'.**

➔ **Zorg bij het tonen van de dialoogvensters dat `Stage` de 'owner' is van het venster.**

➔ **Zie de documentatie van `FileChooser`.**

PRACTICUM 3

Wanneer een tekening is opgeslagen, moet deze ook 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 uit de reader (listing 61) nogmaals opgenomen:

```
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 practicum 2. Je kunt het openen van een bestand ook geheel in een handler (innerclass of lambda-functie) programmeren. Beredeneer wat jou het beste lijkt.

➔ **Maak menuitem ‘Open’ in menu ‘Bestand’ en koppel hieraan accelerator Ctrl+O.**

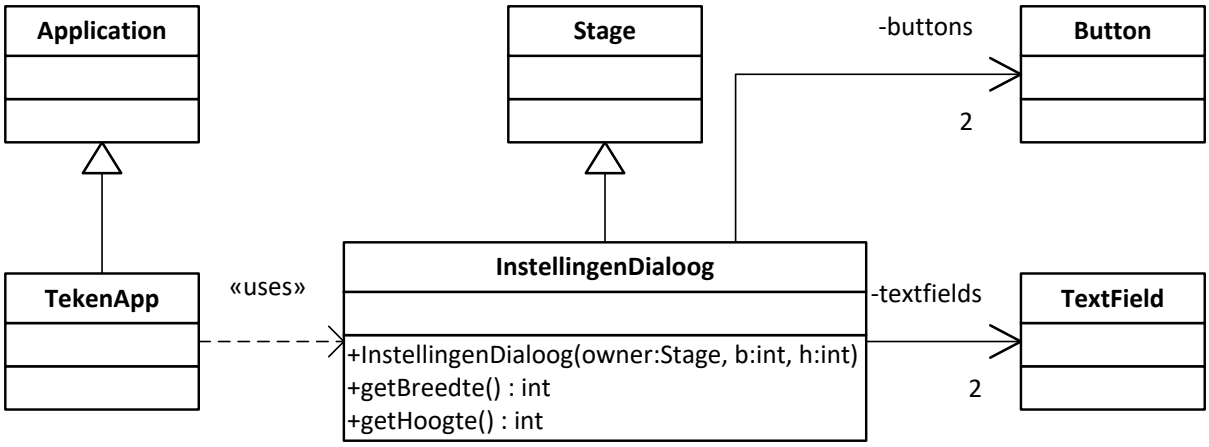
Tip 1: Je zult merken dat als je een afbeelding opent die groter is dan het scherm, de `Stage` niet meeschaalt naar de juiste grootte. Zodra je een afbeelding hebt geopend kan je de volgende code gebruiken om het scherm automatisch de juiste grootte te geven:

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

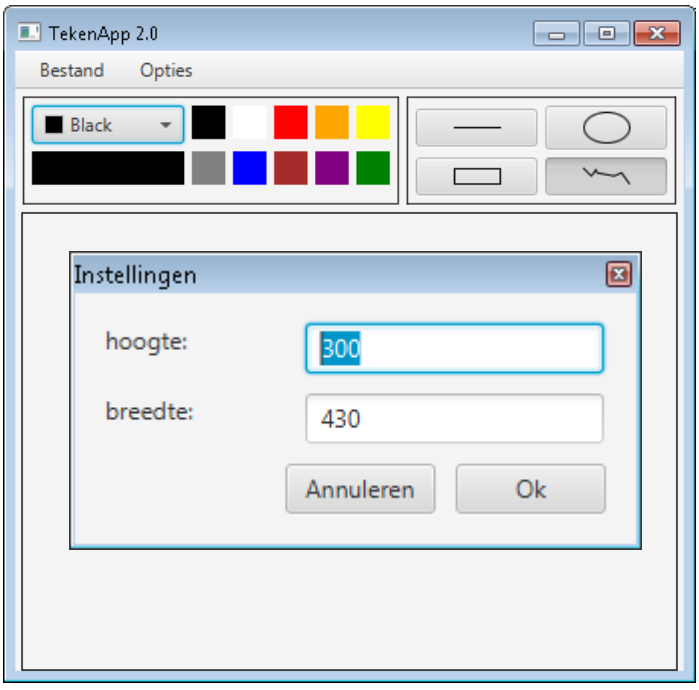
Tip 2: Je kunt voor deze opdracht volstaan met wijzigingen in klasse `TekenApp`. Klassen `TekenCanvas` en `LintBox` kunnen ongewijzigd blijven.

PRACTICUM 4

In deze opgave maken we een klasse `InstellingenDialoog` waarmee de grootte van de afbeelding gewijzigd kan worden. Deze klasse 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 `InstellingenDialoog` 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 de reader.

➔ **Programmeer dit dialoogvenster.**

PRACTICUM 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 moet een Alert getoond worden. Deze alert bevat drie knoppen; “Cancel”, “No” en “Yes”. Het dialoogvenster kan verschillende vragen of boodschappen aan de gebruiker doorgeven.

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

- De gebruiker kiest ‘No’, dan doet de methode niets en levert `true` op.
- De gebruiker kiest ‘Cancel’, dan doet de methode niets en levert `false` op.
- De gebruiker kiest ‘Yes’, dan wordt de eerder geprogrammeerde methode `afbeeldingOpslaan(.)` gebruikt om de tekening op te slaan. Het resultaat daarvan wordt ook het resultaat van deze methode.

➔ Programmeer methode `wijzigingenAfgehandeld()` in `TekenApp`.
➔ Gebruik deze methode zodra de gebruiker een nieuwe tekening wil maken.

<u>Optionele opdrachten:</u>

➔ Gebruik deze methode als het programma afgesloten wordt (`setOnCloseRequest`). Hoe kan je afsluiten voorkomen als de gebruiker annuleert? Zie methoden van `WindowEvent`.
➔ Pas de methode toe op 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 dialoogvensters getoond worden als er nog niets is getekend.

PRACTICUM 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 documentatie van `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.**