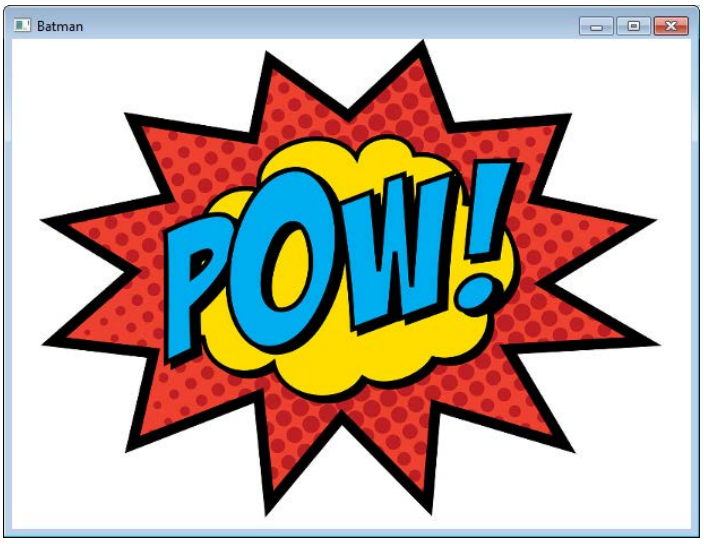
Karel de Grote Hogeschool

BATMAN



In deze opdracht oefenen we de overgang tussen verschillende views. Doorheen de applicatie blijven we werken met één en hetzelfde venster terwijl we de inhoud van het venster aanpassen.

De view en het model zijn gegeven, je hoeft dus enkel de presenter te implementeren.

De voornaamste JavaFX klassen die we voor deze oefening nodig hebben zijn:

- `javafx.scene.Scene`
- `javafx.event.EventHandler`
- `javafx.scene.input.MouseEvent`

Raadpleeg in eerste instantie de JavaFX documentatie als je ergens vast zit!

1 WIREFRAME

De wireframe die bij deze oefening hoort is triviaal. Je mag deze als oefening bouwen met behulp van een tool naar keuze.

2 HOOFDSCHERM AANMAKEN – MVP

De *model* klasse is **`Game`**. De **`enum GamePhase`** duidt aan in welke fase het “spel” zich bevindt (*pow*, *wham* of *zap*).

Er zijn **drie** *view* klassen: **`PowPane`**, **`WhamPane`** en **`ZapPane`**. Deze klassen zijn gegeven en hoef je niet aan te passen.

De *presenter* klasse is **`BatmanPresenter`**. Deze klasse dien je volledig uit te werken in punt 4.

De **`start`** methode van de **`Main`** klasse moet aangevuld worden:

- Maak een instantie aan van het model (**`Game`**).
- Maak een instantie aan van elk van de drie view klassen.
- Maak een instantie aan van **`BatmanPresenter`**. Gebruik hiervoor een nog te implementeren constructor (zie punt 4) waar je het model en de drie aangemaakte views als parameter aan meegeeft.
- Maak een **Scene** aan waar je je **`PowPane`** object op plaatst.
- Plaats je **Scene** op je **`Stage`**.

3 UI OPBOUWEN

De view is reeds uitgewerkt voor deze oefening.

Er is echter een treffende gelijkenis tussen de klassen **`PowPane`**, **`WhamPane`** en **`ZapPane`**. Je kan de dubbele code wegwerken door gebruik te maken van *inheritance*. Probeer deze klassen te *refactoren* wanneer je klaar bent met de oefening.

4 AFHANDELEN EVENTS

De **BatmanPresenter** klasse zorgt voor de afhandeling van events. Telkens de gebruiker ergens in het venster klikt moet er overgegaan worden naar de volgende fase van het spel.

De volgorde is: pow → wham → zap → pow → wham → zap → pow → wham → zap → …

Met bijhorende view klassen: **PowPane → WhamPane → ZapPane → PowPane →** …

- Zorg voor vier attributen: een attribuut voor het model en een attribuut elk van de drie views waar deze presenter mee werkt.
- Implementeer de constructor: zorg er voor dat elk van de vier attributen een waarde krijgt. Hiervoor gebruiken we vier parameters.

4.1 De methode addEventHandlers

Voeg de methode **addEventHandlers** toe aan **BatmanPresenter** en zorg er voor dat ze opgeroepen wordt in de constructor.

In de methode **addEventHandlers**:

- Hang een event handler aan elk van de drie views. We hangen m.a.w. een event handler aan de views zelf (dit zijn **BorderPanes**) en niet aan controls binnenin de views.
- We willen dat onze event handler opgeroepen wordt wanneer de gebruiker met de muis klikt in een view, dus we werken met **setOnMouseClicked** en **niet** met **setOnAction**!
- De event handler doet het volgende:
  1. Het model wordt naar de volgende spelfase gebracht (**nextGamePhase**).
  2. De **updateView** methode wordt opgeroepen (zie 4.2).
- Extra: Je kan werken met één instantie van de event handler en deze ene instantie meegeven als parameter bij elke oproep van **setOnMouseClicked**. Op deze manier hoef je geen **nieuwe** event handler aan te maken voor elke **setOnMouseClicked**.

4.2 De methode updateView

Voeg de methode **updateView** toe aan **BatmanPresenter**, we roepen deze methode **niet** op in de constructor.

In de methode **updateView**:

- Raadpleeg het model om te achterhalen wat de **huidige** spelfase is.
- Op basis van de **huidige** spelfase bepaal je de **vorige** spelfase.
- Gebruik de view die hoort bij de **vorige** spelfase om een **Scene** object te verkrijgen:
  ```
  Scene scene = someView.getScene();
  ```
- Op de **Scene** plaats je de view die hoort bij de **huidige** spelfase:
  ```
  Scene.setRoot(anotherView);
  ```