Using layouts

- Since JavaFX still is Java, the use of layout managers is a good idea.
- Layouts are used as they make it possible to predict the behaviour of programs on different platforms.
  - The visual relationship with the other components is preserved.
- In JavaFX the layout managers are called panes.
- In essence they work the same way as in "normal" Java.
  - Stretches according to the layout algorithms of the specific pane.
- It is also possible to manually position the visual components, but make a habit of using panes instead.

Panes in JavaFX

- The panes in JavaFX are eight in total and can be used together with each other:
  - Border pane
  - HBox and VBox
  - Stack pane
  - Grid pane
  - Flow pane
  - Tile pane
  - Anchor pane
- Many of them are similar to the Swing versions, but some are developed specially for JavaFX.
- We are going to look more closely on a few of them.
Border pane

- Taken almost directly from the Swing equivalent, the BorderLayout.
- It has five regions:
  - Top
  - Left
  - Right
  - Bottom
  - Center
- The centre region takes the variable space.
- Notice that if space is too small (i.e. the window is too small), then the regions will overlap.
  - Overlapping is subject to the order that the regions have been added.

VBox and HBox

- Two new and very simple to use layouts are VBox and HBox.
  - For vertical and horizontal box.
- With these it is very simple to create user interfaces that look as we like.
- To adjust the space around the layout, there are two methods:
  - setPadding() – for the space around the layout box.
  - setSpacing() – for the space around the controls of the box.
- To adjust the size of the visible components themselves, use the setPrefSize().
  - This sets the preferred size of the component, but it will adjust itself to the surrounding.

Graphics of program

Figure: Original size of program.

Figure: Resized window.

```java
// imports omitted
public class BoxLayout extends Application {
    @Override
    public void start(Stage primaryStage) {
        final TextField left = new TextField();
        left.setPrefSize(100, 20);
        final TextField right = new TextField();
        right.setPrefSize(100, 20);
        Button btn = new Button();
        btn.setText(" -> Move -> ");
        btn.setPrefSize(150, 20);

        HBox root = new HBox();
        root.setPadding(new Insets(10, 10, 10, 10));
        root.setSpacing(5.0);
        root.getChildren().addAll(left, btn, right);

        Scene scene = new Scene(root, 350, 50);
        primaryStage.setTitle("Moving content");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```
Let's create an application in order to create a more life like layout.
The example will show several more interesting parts of JavaFX.
The following is a snapshot of the initial state of the application:

The structure
- At the bottom there is a BorderPane for the five different regions.
- Each of the regions has either a VBox or a HBox for structuring the content.
- Inside of each such pane, a number of controls are used.
- Even if only one control is used, a box has been used in order to get some margins.
- This also helps in keeping the layout when re-sizing the window.

Top pane
- In the top pane a text label and a background is set.
  ```java
  HBox theTop = new HBox();
  Text caption = new Text("Star Wars Knowledge Bank");
  caption.setFont(Font.font("SansSerif", 30));
  theTop.setPadding(new Insets(10, 10, 10, 10));
  theTop.setStyle("-fx-background-color: #666699;"f;
  theTop.getChildren().add(caption);
  ```
  To set the font the Font class is used, but notice that now object is created.
  Padding is using the class Insets which defines the amount of space for top, right, bottom and left (in that order).
- To change the background colour we set a style.
Styles in JavaFX

- Styling in JavaFX follows the CSS 2.1 specification.
- As seen in the previous example, some additions have been made.
- The default style in JavaFX is called Caspian and is available as a loadable css-file.
- This makes it easy to change the style by changing the css-style.
  - To separate style from the content is a good thing, makes it easier to focus.
- As seen in the example, it is also possible to style inline if only a small change is needed.
- Otherwise, a new style can be added to the scene by the getStylesheets() method.

Left pane

- The left pane holds a number of buttons that can be pressed.
- In this example they are statically created, otherwise an array would have been better.
- To get all the buttons to have the same width, the setMaxWidth() method needs to be set.
  - By sending the constant Double.MAX_VALUE all the buttons will get the same width as the widest button.
  - This is the default behaviour if the value sent is too large for the width.
  - An alternative would have been to use a really large number, but if screens get that big, the code will fail.
- Lastly all buttons are added to the box with the addAll() method.

The code

```java
VBox buttons = new VBox();
Button vader = new Button("Darth Vader");
vader.setMaxWidth(Double.MAX_VALUE);
Button tiin = new Button("Saesee Tiin");
tiin.setMaxWidth(Double.MAX_VALUE);
Button bane = new Button("Cad Bane");
bane.setMaxWidth(Double.MAX_VALUE);
Button binks = new Button("Jar Jar Binks");
binks.setMaxWidth(Double.MAX_VALUE);
Button fett = new Button("Boba Fett");
fett.setMaxWidth(Double.MAX_VALUE);
buttons.setPadding(new Insets(10, 10, 10, 10));
businesses.setSpacing(10);
businesses.getChildren().addAll(vader, tiin, bane, binks, fett);
```

Centre pane

- The centre pane is divided into text controls for the heading and the text content of an entry.
- To get them on top of each other, they are put in a VBox.
- The header is getting a nice drop shadow for visual effect.
- The text for the content is entered statically in the code, but in order for it to be wrapped, the setWrappingWidth() method is set.
- Since the content, potentially, can be larger than the height of the text control, we add scroll bars using the class ScrollPane.
- Also, it needs to be put to a preferred height using setPrefHeight.
The code

```java
VBox textContent = new VBox();
textContent.setPadding(new Insets(10, 10, 10, 10));
section.setFont(Font.font("SansSerif", 20));
DropShadow ds = new DropShadow();
ds.setFill(Color.rgb(50, 50, 50, .6));
section.setEffect(ds);
final Text content = new Text("Click the buttons on the left to find out content.setWrappingWidth(500);
ScrollPane forText = new ScrollPane();
forText.setFitToHeight(true);
forText.setContent(content);
textContent.getChildren().addAll(section, forText);
```

Right pane

◮ To the right an image of the character is shown.
◮ In JavaFX, displaying images is a two part task.
◮ First, an `ImageView` needs to be defined which will be the view port of the image.
◮ It is possible to show only a part of an image or to scroll it in different directions.
◮ After that, an `Image` object is needed to hold the image itself.
◮ Use the code above in order to read the file from the `src` directory of the project.
◮ Also notice that we scale the image with preserved ratio to make it fit the pane.

The code

```java
HBox showImage = new HBox();
showImage.setPadding(new Insets(10, 10, 10, 10));
final ImageView theView = new ImageView();
theView.setFitHeight(300);
theView.setPreserveRatio(true);
final Image theImage = new Image(getClass().getResourceAsStream("lightsaber.png"));
theView.setImage(theImage);
showImage.getChildren().add(theView);
```

The bottom and finishing

◮ At the bottom a simple "copyright" message is displayed.
◮ In order to get it with proper margins, it has been put in a box.
◮ Lastly, in the end all the different parts are put into the border pane.

```java
VBox footer = new VBox();
footer.setPadding(new Insets(10, 10, 10, 10));
Text copyright = new Text("No copyright since all material is shamelessly footer.getChildren().add(copyright);
```
And action!

- One part is still missing, the action for the buttons.
- For each button, a setAction() method is used with the same structure:
  ```java
  vader.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
      section.setText("Darth Vader");
      content.setText("Darth Vader (born Anakin Skywalker) is the central
      theatrics. setImage(new Image(getClass().getResourceAsStream("vader")));
    }
  });
  ```
- It sets the specific text and image for the button pressed.
- Notice that the controls need to be declared as final for this to work, since JavaFX needs to be certain that the object stays the same.

GridPane example

- Even though the combination of border pane and boxes gives great power, there are other alternatives.
- Often, a grid-like layout is needed and it is possible to use the GridPane.
- This allows for a very flexible layout, where rows and columns are used — but it is also possible to span multiple rows/columns.
- It is possible to set the grid creating the object, but otherwise it will be calculated by use.
- It is possible to set both the padding and the gap between controls for maximum control.

The application in graphics

![Log in](image)

More about some controls

- In this case, the Label was used instead of Text.
  - The way it is used here does not make it different from Text but additional methods like labelFor() makes it useful for text applications.
- For the first entry the normal TextField is used but for the other a special PasswordField is used.
  - Makes inputted text to appear in hidden format.
- Also notice that we use the method setPromptText() to have a slightly dimmer help text appear in an empty text field.
Creating the grid pane

GridPane theGrid = new GridPane;
theGrid.setPadding(new Insets(5));
theGrid.setHgap(5);
theGrid.setVgap(5);

Label userName = new Label("User Name:");
TextField firstText = new TextField;
firstText.setPromptText("Enter your user name");
Label password = new Label("Password");
PasswordField lastText = new PasswordField;
lastText.setPromptText("Enter your password");
Button okBtn = new Button("OK");

Setting the position

◮ The position is set by specifying the grip coordinate.
◮ Also set the alignment for the controls to make the GUI look professional.

theGrid.setHalignment(userName, HPos.RIGHT);
theGrid.add(userName, 0, 0);
theGrid.setHalignment(password, HPos.RIGHT);
theGrid.add(password, 0, 1);
theGrid.setHalignment(firstText, HPos.LEFT);
theGrid.add(firstText, 1, 0);
theGrid.setHalignment(lastText, HPos.LEFT);
theGrid.add(lastText, 1, 1);
theGrid.setHalignment(okBtn, HPos.RIGHT);
theGrid.add(okBtn, 1, 2);

AnchorPane

◮ The last pane we look at is the, again very powerful, AnchorPane.
◮ This pane allows for attaching other parts to the top, bottom, right, left or centre of the pane.
◮ When the window is re-sized, the controls are anchored to the corners and follows the window.
◮ It is possible to attach a control to more than one anchor.
  ◦ For example, to the lower right corner.
◮ This is also the default pane for the SceneBuilder tool.

Example

◮ The following example will show an image viewer application.
  ◦ It does have some "errors" in it, like no proper error handling and also only able to read images in the src folder.
  ◦ The main part of the application is a image view which is anchored to most of the screen.
  ◦ It also has a ScrollPane to be able to view the entire image if it does not fit.
  ◦ At the lower right there are two buttons that are anchored to that corner even when the window is re-sizing.
In graphics

Everything is put in a HBox.
An view, an image and a scroll pane.
The loading of images works the same as in the previous example, and only from the src folder.
An addition that has been made to the scroll is to set setPannable() to true.
This makes it possible to drag the image around in the pane using the mouse.

HBox theImage = new HBox();
theImage.setPadding(new Insets(10, 10, 10, 10));
final ImageView theView = new ImageView();
final Image theActualImage = new Image(getClass().getResourceAsStream("deathstar.png"));
ScrollPane sb = new ScrollPane();
sb.setPannable(true);
sb.setContent(theView);
theImage.getChildren().add(sb);

The buttons

The two buttons are also added into a HBox.
The action for the exit button is rather simple:

The code

The view of the image

The view of the image
◮ Everything is put in a HBox.
◮ An view, an image and a scroll pane.
◮ The loading of images works the same as in the previous example, and only from the src folder.
◮ An addition that has been made to the scroll is to set setPannable() to true.
◮ This makes it possible to drag the image around in the pane using the mouse.

HBox buttons = new HBox();
buttons.setPadding(new Insets(10, 10, 10, 10));
buttons.setSpacing(10);
Button loadBtn = new Button("Load image...");
Button exitBtn = new Button("Exit");
bMap.getChildren().addAll(loadBtn, exitBtn);

◮ The action for the exit button is rather simple:

◮ The two buttons are also added into a HBox:

◮ Everything is put in a HBox.
◮ An view, an image and a scroll pane.
◮ The loading of images works the same as in the previous example, and only from the src folder.
◮ An addition that has been made to the scroll is to set setPannable() to true.
◮ This makes it possible to drag the image around in the pane using the mouse.

HBox theImage = new HBox();
theImage.setPadding(new Insets(10, 10, 10, 10));
final ImageView theView = new ImageView();
final Image theActualImage = new Image(getClass().getResourceAsStream("deathstar.png"));
ScrollPane sb = new ScrollPane();
sb.setPannable(true);
sb.setContent(theView);
theImage.getChildren().add(sb);

The buttons
◮ The two buttons are also added into a HBox:

◮ Everything is put in a HBox.
◮ An view, an image and a scroll pane.
◮ The loading of images works the same as in the previous example, and only from the src folder.
◮ An addition that has been made to the scroll is to set setPannable() to true.
◮ This makes it possible to drag the image around in the pane using the mouse.

HBox theImage = new HBox();
theImage.setPadding(new Insets(10, 10, 10, 10));
final ImageView theView = new ImageView();
final Image theActualImage = new Image(getClass().getResourceAsStream("deathstar.png"));
ScrollPane sb = new ScrollPane();
sb.setPannable(true);
sb.setContent(theView);
theImage.getChildren().add(sb);
Setting up the anchors

- Lastly the anchors are set up by using static members to the AnchorPane class.
  - On the form AnchorPane.setRightAnchor(sb, 5.0);
  - The first parameter is the control to add to this anchor.
  - The second parameter is for the distance to that edge, in this case five pixel.
  - When all anchors have been set up, it is still necessary to add the controls to the pane itself.
    - That is, to the object.

The code

```java
AnchorPane.setRightAnchor(sb, 5.0);
AnchorPane.setTopAnchor(sb, 5.0);
AnchorPane.setLeftAnchor(sb, 5.0);
AnchorPane.setBottomAnchor(sb, 50.0);
AnchorPane.setRightAnchor(buttons, 5.0);
AnchorPane.setBottomAnchor(buttons, 50.0);
theAnchor.getChildren().addAll(sb, buttons);
```

The last action

- As stated, the loading of images is not working fully.
- When pressed, the button will show the normal file chooser for the system.
- It works similar to the Swing version.
- One method is showOpenDialog() that shows the open dialog and returns the file and path.

```java
loadBtn.setOnAction(
    new EventHandler<ActionEvent>() {
      @Override
      public void handle(ActionEvent event) {
        FileChooser fc = new FileChooser();
        File newImage = fc.showOpenDialog(null);
        theView.setImage(new Image(getClass().getResourceAsStream(newImage.getName())));
      }
    });
```

The End

- This concludes today’s lecture.
- The agenda for next lecture is to show both animations and bindings.
  - And possibly more...