Grafiska användargränssitt och händelsehantering

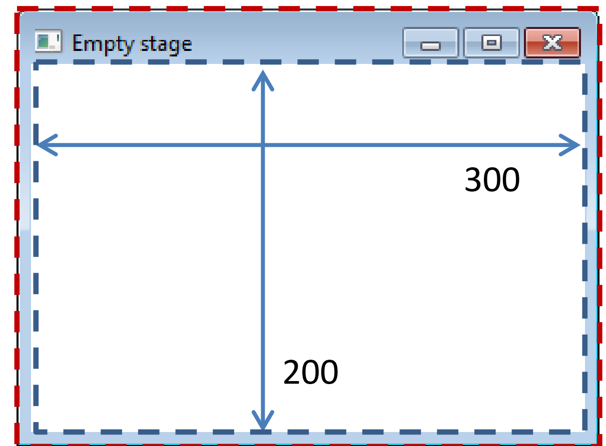
…i Java

GUI i Java

- AWT, Abstract Window Toolkit, java.awt
  - Använder systemets GUI-komponeneter
- Swing, javax.swing
  - Komponenter, förutom fönster, renderas av Java
- JavaFX, javafx och underpaket
  - Komponenter, förutom fönster, renderas av Java
- JavaFX
  - Utvecklas kontinuerligt
  - Desktop/browser/pad: support för touch events, CSS-styling, …
  - Enklare hantering av layout
  - Bättre grafik: rotationer, gradienter, …
  - Enklare och säkrare animering
- NB! Många klasser i JavaFX har samma namn som i AWT – håll koll på vilket paket du importerar från

Ett tomt GUI

```
public class EmptyStage extends Application {

    @Override
    public void start(Stage primaryStage) {
        Pane pane = new Pane();
        Scene scene = new Scene(pane, 300, 200);

        primaryStage.setTitle("Empty stage");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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



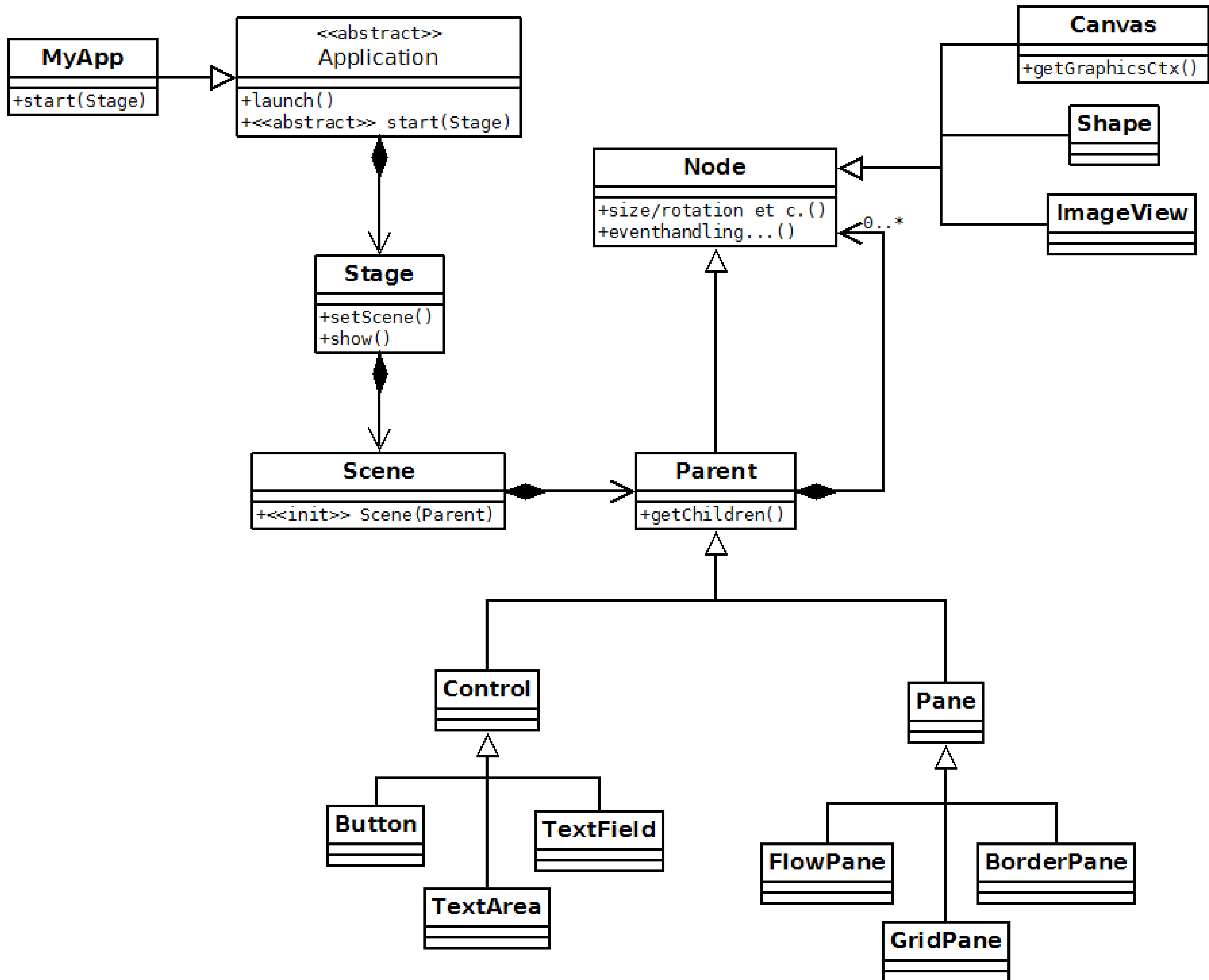
GUI i Java FX

```
public class EmptyStage extends Application {

    @Override
    public void start(Stage primaryStage) {
        Pane pane = new Pane();
        Scene scene = new Scene(pane, 300, 200);

        primaryStage.setTitle("Empty stage");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

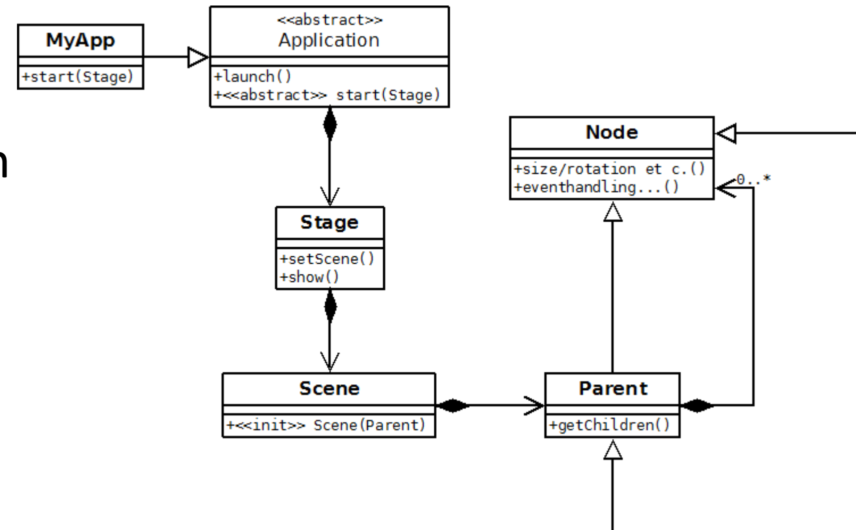
- Ärv klassen javafx.application.Application
- Omdefiniera metoden start
  - Skapa ui-komponenter och placera dessa i ett Stage-objekt (fönster)
  - Anropa stage.show
- Exekveringen startar genom att metoden Application.launch anropas

- En JavaFX-applikation exekveras i en separat tråd, ”UI thread”, “JavaFX Application thread”
  - All uppdatering av ui-komponenter måste göras från denna tråd
  - All kod för händelshantering (event handlers) exekveras på denna tråd
  - Långa beräkningar bör exekveras på separat tråd för att inte “frysa” UI

- Exempel: EmptyStage.java, HelloJavaFXWorld.java


MyApp
+start(Stage)
<<abstract>>
Application
+launch()
+<<abstract>> start(Stage)
Stage
+setScene()
+show()
Scene
+<<init>> Scene(Parent)
Node
+size/rotation et c.()
+eventhandling...()
0..*
Canvas
+getGraphicsCtx()
Shape
ImageView
Parent
+getChildren()
Control
Button
TextField
TextArea
Pane
FlowPane
BorderPane
GridPane

Stage, Scene*



- Stage (extends Window)
  - Det yttersta fönstret i applikationen
  - Ytterligare fönster?
    - PopupWindow
    - Alert – tillfällig meddelande/input
    - Ytterligare Stage? Sätt ”modal”, använd show/hide
- Scene
  - En container för allt innehåll i ett scen-träd (”scene graph”)
  - Applikationen måste ange en rot-nod för scenen, som i sin tur innehåller alla eventuella övriga komponenter
  - Applikationen kan byta mellan olika scener för en och samma stage

* ”Scene” i svenska betydelsen en scen eller tablå i en pjäs

Tillfälliga dialogfönster

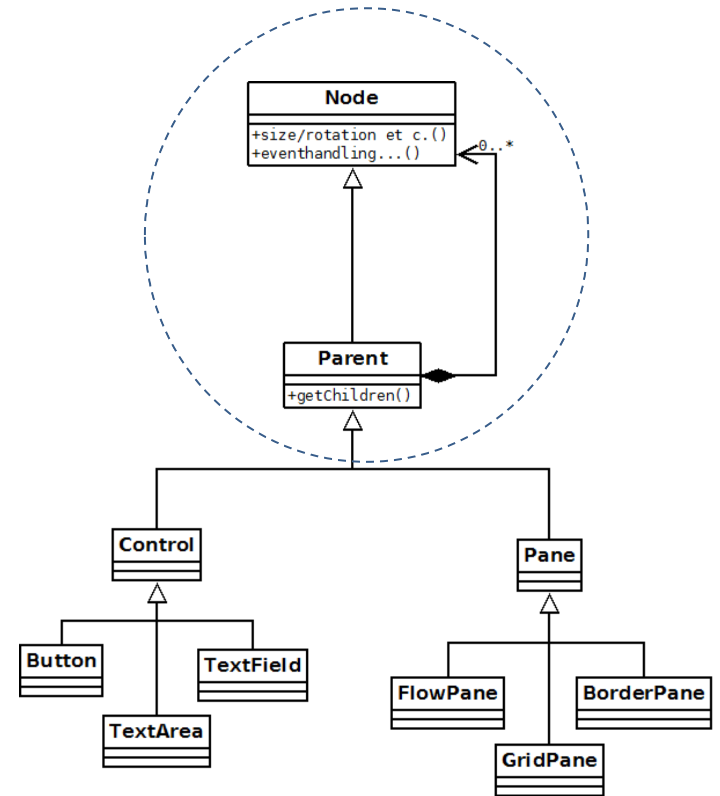
- Måste skapas på UI-tråden

```
Alert alert = new Alert(Alert.AlertType.INFORMATION);
alert.setHeaderText("Message");
alert.setTitle("Hello!");
alert.setContentText(message);
alert.show();
```

- Default är ett Alert-objekt
  - Modalt – användaren kan inte interagera med bakomliggande UI (efter alert.show)
  - Blockerande – exekveringen av huvudapplikationen pausar till dess objektet inte visas längre (ex. via alert.hide() eller användarinteraktion)
- Exempel: HelloJavaFXWorld.java

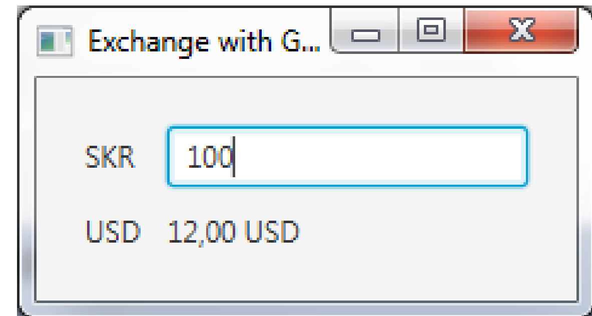
Node, Parent, Pane, Control

- Node
  - Superklass för alla noder i ett scen-träd
  - En ui-komponent som kan visas och agera händelsekälla
- Parent
  - Subtyp till Node som kan innehålla andra noder, ”children” (en container)
- Pane
  - Subtyp till Parent som även definierar en layout-strategi, dvs. hur noder ska placeras i vyn
  - Exempel: FlowPane, GridPane, BorderPane
- Control
  - Superklass för alla komponenter för användarinteraktion, som knapp, inmatningsfält, …

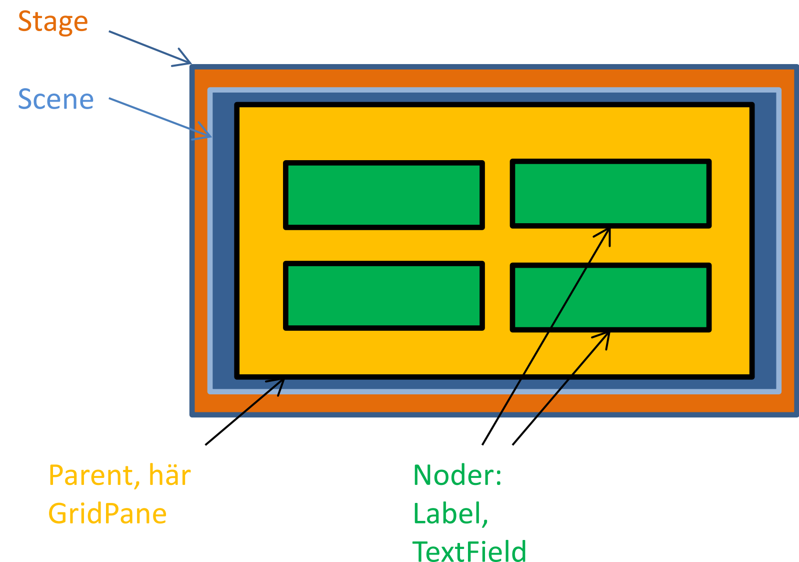


”Scene graph”

- Exempel – UI för en enkel växlingsapplikation



- Motsvarande komponenthierarki, inklusive ”scene graph”



”Scene graph” (Application)

```
private TextField skrInput;
private Label usdValue;

@Override
public void start(Stage primaryStage) {

    skrInput = new TextField("Enter value...");
    usdValue = new Label("- - -");

    FlowPane root = new FlowPane();

    Collection children = root.getChildren();
    children.add(new Label("SKR"));
    children.add(skrInput);
    children.add(new Label("USD"));
    children.add(usdValue);

    Scene scene = new Scene(root);
```

- Scene(root, 300, 300) - ger angiven storlek i pixels
  Scene(root) – ger en storlek beräknad efter noderna som palcerats i root
- Kodexempel: ExchangeFlowPane.java, ExchangeGridPane.java

Layout med Pane(s)

- Pane (superklass)
  - pane.getChildren().add(node)
- FlowPane
  - Noderna placeras från vänster till höger så långt de får plats, sedan ny ”rad”
  - Inbördes placering kan ändras om användaren ändrar fönstrets dimensioner
  - Hack:
    stage.setScene(new Scene(pane, 300, 300);
    stage.setResizable(false);
- GridPane
  - Noder laceras ut i rader och kolumner:
    gridPane.add(node, *col, row*)
  - gridPane.setH/Vgap(pixels);
- Hbox, Vbox
  - Noder placeras ut på en enda rad
  - Box.setMargin(pixels)

- Kodexempel: ExchangeFlowPane.java, ExchangeGridPane.java, BorderPaneExample.java

Layout med Pane, forts.

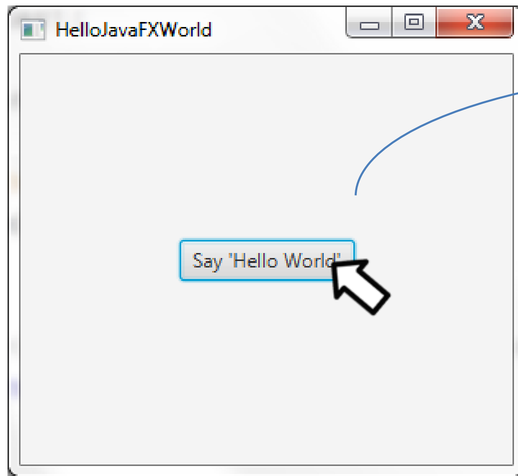
- BorderPane
  - Noder placeras ut med borderPane.setTop(node/parent) setBottom / setLeft / setRight / setCenter
  - När fönstret ändrar storlek påverkar detta främst center-regionen
  - Flexibelt: var och en av de fem regionerna kan i sig innehålla en parent-komponent

- Kodexempel: BorderPaneExample.java

Layout med Pane, forts

```
BorderPane border = new BorderPane();

Label topLabel = new Label("This is the top area");
border.setTop(topLabel);

FlowPane bottomPane = new FlowPane();
bottomPane.getChildren().add(new Button("Hit me!"));
bottomPane.getChildren().add(new Button("No, hit me!"));
border.setBottom(bottomPane);

TextArea textArea = new TextArea("Bla bla bla...");
border.setCenter(textArea);

Scene scene = new Scene(border);
```

Händelsehantering



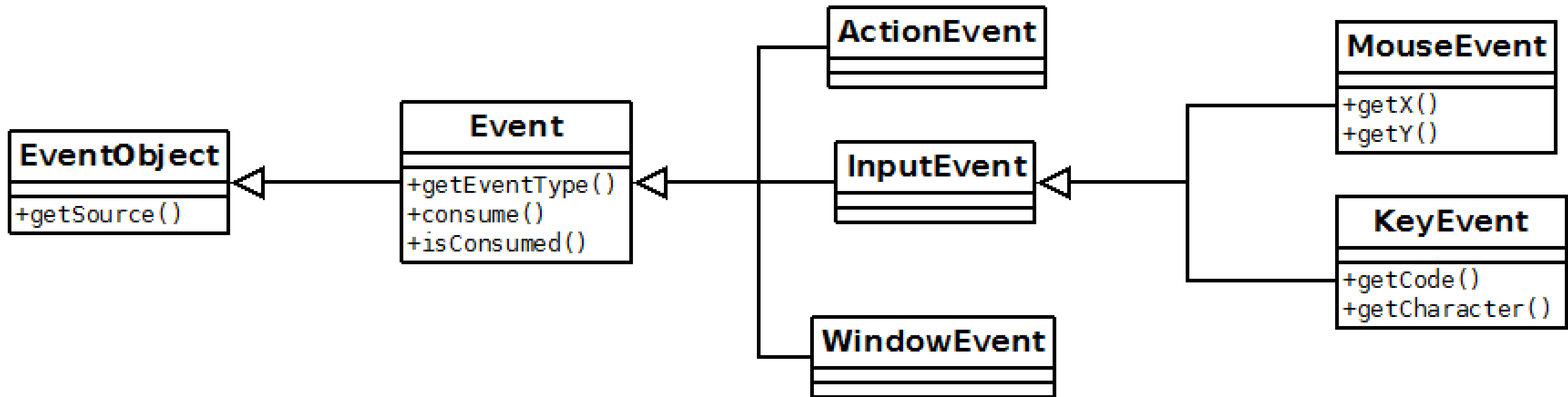
Event-objekt

EventHandler-object

```
class ButtonHandler implements
   EventHandler<ActionEvent> {

   @Override
   public void handle(ActionEvent me) {
      // Take some action…
   }
}
```

- Händelsekälla, ex knapp eller inmatningsfält
- Händelse, ex knapptryck eller ENTER i inmatningsfält – ett Event-objekt skapas
- Händelsehanterare – implementerar EventHandler.handle(Event) – definierar vad som ska göras

Event-typer



- getSource –returnerar en referens till källan till händelsen
- getEventType – returnerar en händelses (under-) typ, ex. MouseEvent.RELEASED
- consume/isConsumed – flera komponenter kan registrera sig att hantera en händelse, consume flaggar att händelsen inte ska hanteras vidare

Applikationsprogrammerarens jobb…

Skriv en klass som implementerar interfacet EventHandler<…> (som en separat klass eller en inre klass)

```
1. class ButtonHandler
       implements Eventhandler<ActionEvent>  {

        @Override
        public void handle(ActionEvent event) {
            // Handle the event…
        }
   }
```

Applikationsprogrammerarens jobb…

I Application.start eller konstruktor i någon subklass till Node:

```
2.  Button button = new Button("Red");      // source
3.  ButtonHandler handler =
       new ButtonHandler(…);                // handler
4.  button.addEventHandler(
       ActionEvent.ACTION, handler);        // associate
```

Alternativ till 4:

```
4. button.setOnAction(handler);
```

Allmänt:

component.setOnXXX, ex. canvas.setOnMouseClicked(…)

Kodexempel: EventHandlerExample1.java

Inre klasser (ej specifikt för EventHandler)

```
class Outer {
   private int x, y;

   class Inner {
      private int x;
      void foo() { Outer.this.x++; y++; x++; }
   }
}
```

- Inner-objekt kan endas skapas av ett Outer-objekt
- Objekt av den inre klassen har direkt insyn i den yttre klassen, Outer - även privat data och metoder
- I inre klassen:
  - this refererar till Inner-objektet självt
  - Outer.this refererar till objektet av den yttre klassen

EventHandler som inre klass

```
public class EventHandlerExample2 extends Application {

    private VBox root;

    public void start(Stage primaryStage) {

        root = new VBox(10);
        Button redButton = new Button("Red");

        redButton.setOnAction(new ButtonHandler("red"));
        …
    }

    private class ButtonHandler implements EventHandler<ActionEvent> {
        …
        @Override
        public void handle(ActionEvent event) {
            root.setStyle(styleStr); // CSS style
        }
    }
```

Kodexempel: EventHandlerExample2.java

EventHandler som anonym och inre klass

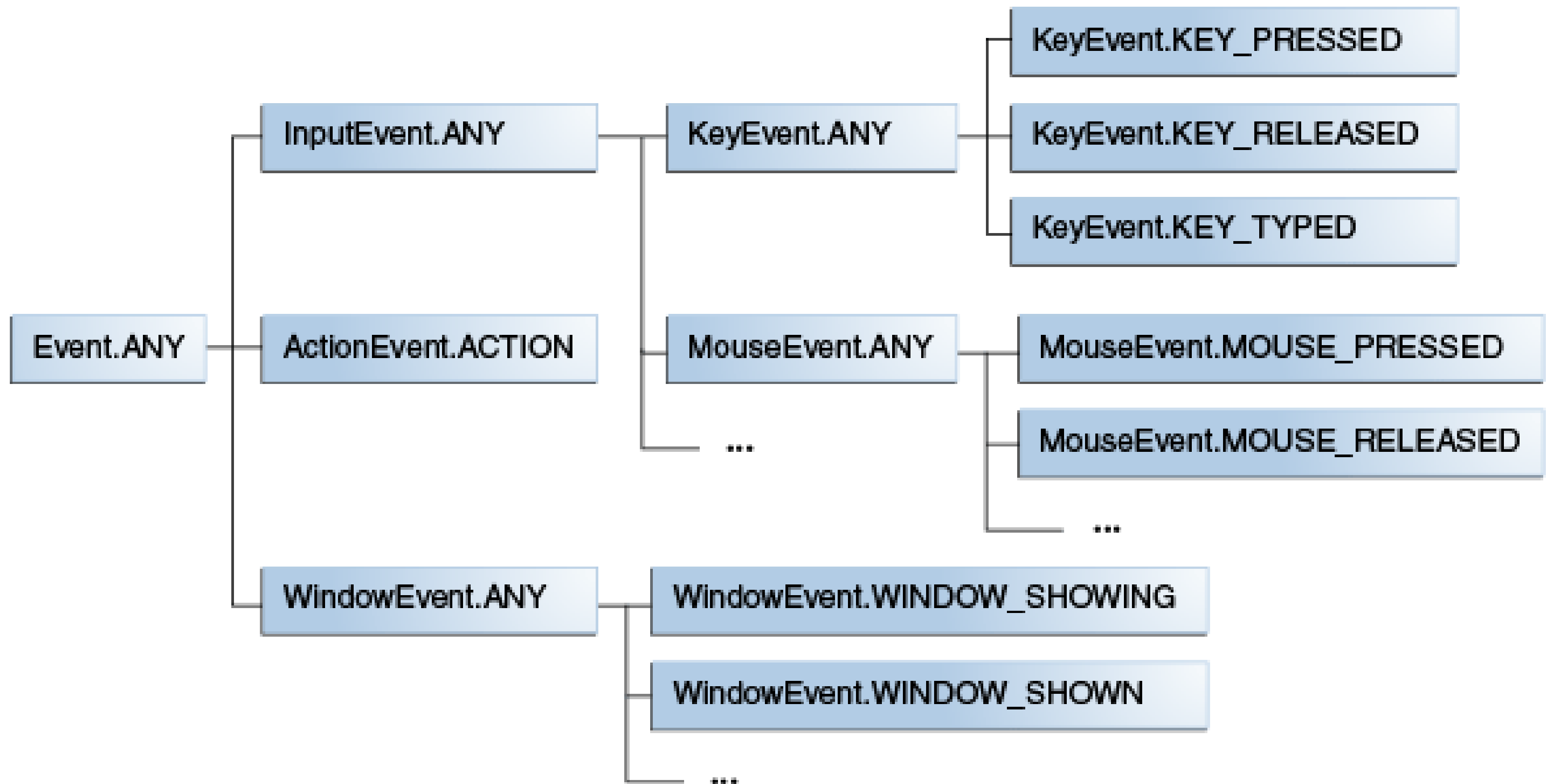
```
public class EventHandlerExample3 extends Application {

    private VBox root;

    public void start(Stage primaryStage) {

        root = new VBox(10);
        Button redButton = new Button("Red");
        …
        // An anonymous inner class
        EventHandler buttonHandler = new EventHandler<ActionEvent>() {
             …
            @Override
            public void handle(ActionEvent event) {
                 root.setStyle(styleStr);
            }
        };

        redButton.setOnAction(buttonHandler);
        …
```

Kodexempel: EventHandlerExample3.java

Händelsetyper och källor (ej komplett)

Användarinteraktion	Händelsetyp	Klass som kan registrera motsv. EventHandler
Tangent tryckt	KeyEvent	Node, Scene
Mus rörs eller musknapp används	MouseEvent	Node, Scene
Mus ”släpas”	MouseDragEvent	Node, Scene
Komponent ”rörs”	TouchEvent	Node, Scene
Knapp trycks, ENTER i inmatningsfält, menyalternativ väljs, …	ActionEvent	ButtonBase, ComboBoxBase, ContextMenu, MenuItem, TextField
Fönster stängs, visas eller döljs	WindowEvent	Window (Stage är en subklass)
…		

Hierarki för händelsetyper



Källa: Oracle: JavaFX Documetation

MouseEvent

```
public void start(Stage primaryStage) {

        canvas = new Canvas(300, 300);
        canvas.setOnMouseClicked(
                new MouseClickHandler());
        …
}

private class MouseClickHandler implements
            EventHandler<MouseEvent> {

        @Override
        public void handle(MouseEvent event) {
            // …
        }
```

- Kodeexempel: MouseEventExample.java

KeyEvent

```
public void start(Stage primaryStage) {

    …
    KeyHandler keyHandler = new KeyHandler();
    root.setOnKeyPressed(keyHandler);
    root.setOnKeyReleased(keyHandler);
    root.setOnKeyTyped(keyHandler);
    …
    // NB! The component receiving key events must be in focus
    root.setFocusTraversable(true);
    root.requestFocus();
}

private class KeyHandler implements EventHandler<KeyEvent> {
    @Override
    public void handle(KeyEvent event) {
        if (KeyEvent.KEY_TYPED.equals(event.getEventType())) {
            text.setText(ch);
        } …
```

- Kodeexempel: KeyEventExample.java, KeyCodeExample.java

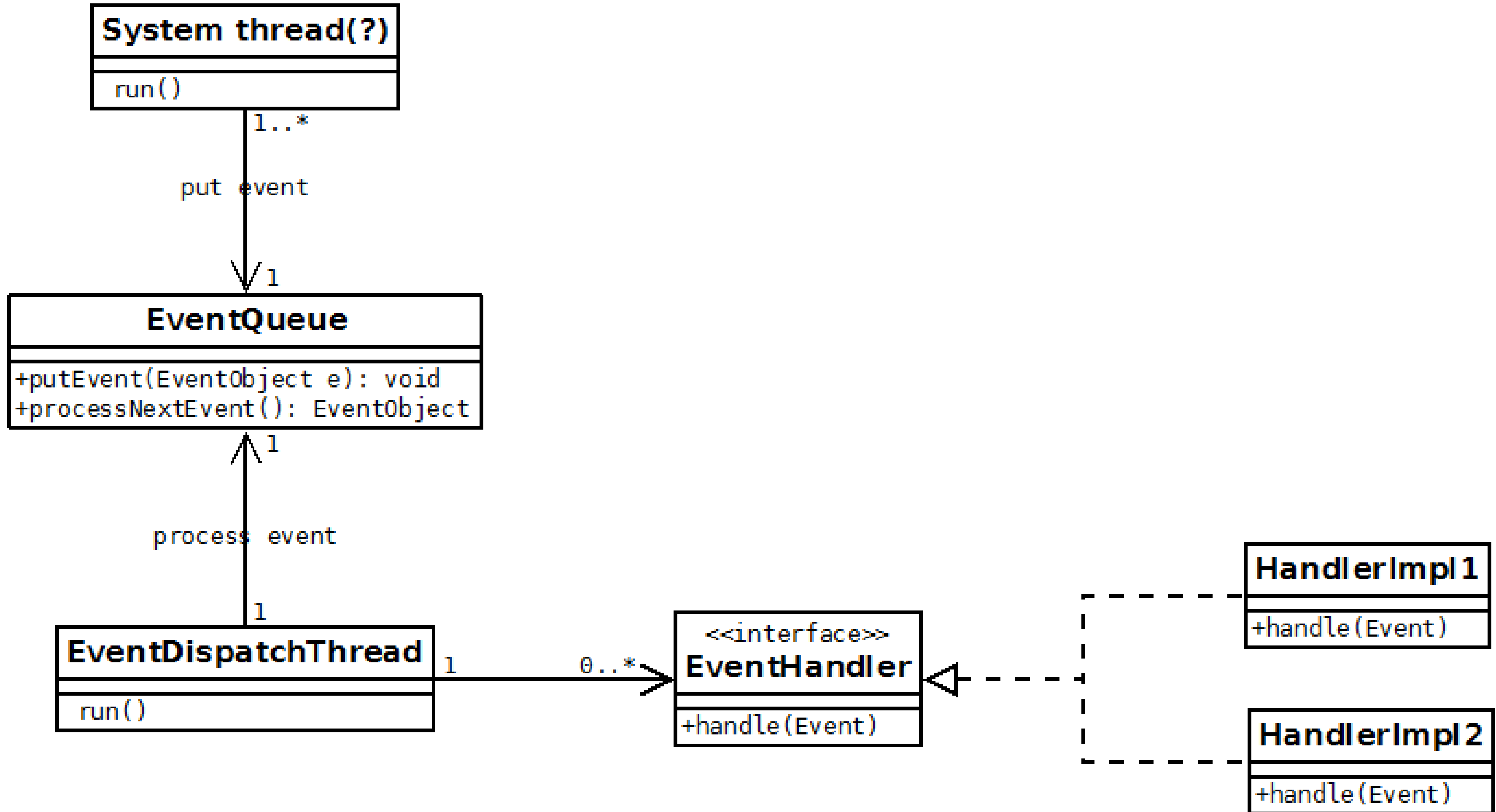
WindowEvent

```
stage.setOnCloseRequest(
        new EventHandler<WindowEvent>() {

        @Override
        public void handle(WindowEvent event) {
             // E.g. save data before exit
        }
});
```

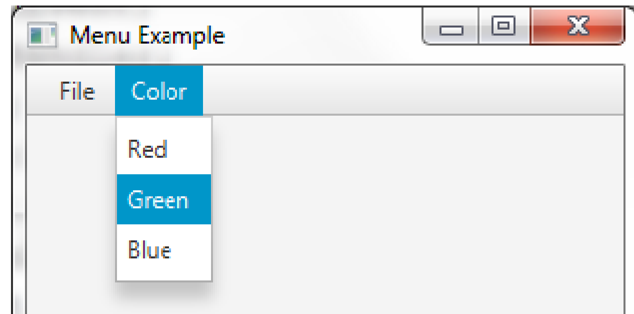
- Ex. definiera vad som ska ske innan applikationens huvudfönster stängs, via ”krysset”, dvs. innan applikationen avslutas

- Kodexempel: WindowEventExample.java

Händelsehantering, bakom kulisserna(?)



Menyer



- MenuBar
- Menu
  - kan innehålla MenuItem-objekt eller Menu-objekt (sub-meny)
- MenuItem
  - källa för ActionEvent
- Lägg till menyraden till root-objektet för Scene
  - root.getChildren.add(menuBar);

Menyer

```
Menu fileMenu = new Menu("File");

MenuItem exitItem = new MenuItem("Exit");
fileMenu.getItems().add(exitItem);
…

MenuBar menuBar = new MenuBar();
menuBar.getMenus().addAll(fileMenu,
colorMenu);

root.getChildren().add(menuBar);
```