Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Java FX – FXML & MVC

- A. Laube 2016, adapted by A. Scheidegger

Outline

- Separation of layout and logic
- Observer Design Pattern
- MVC Design Pattern
- Styling

Outline

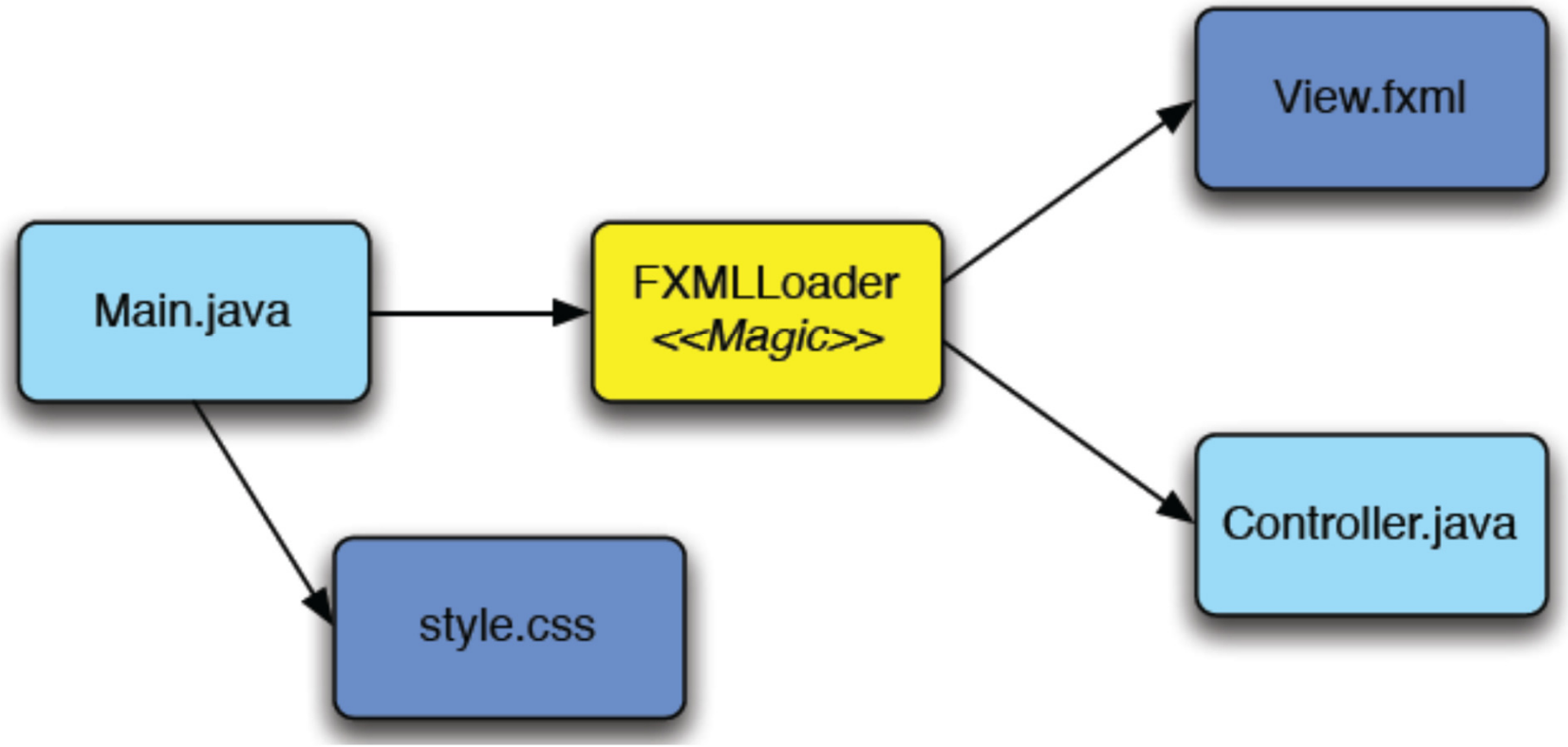
- Separation of layout and logic
- Observer Design Pattern
- MVC Design Pattern
- Styling

Until now

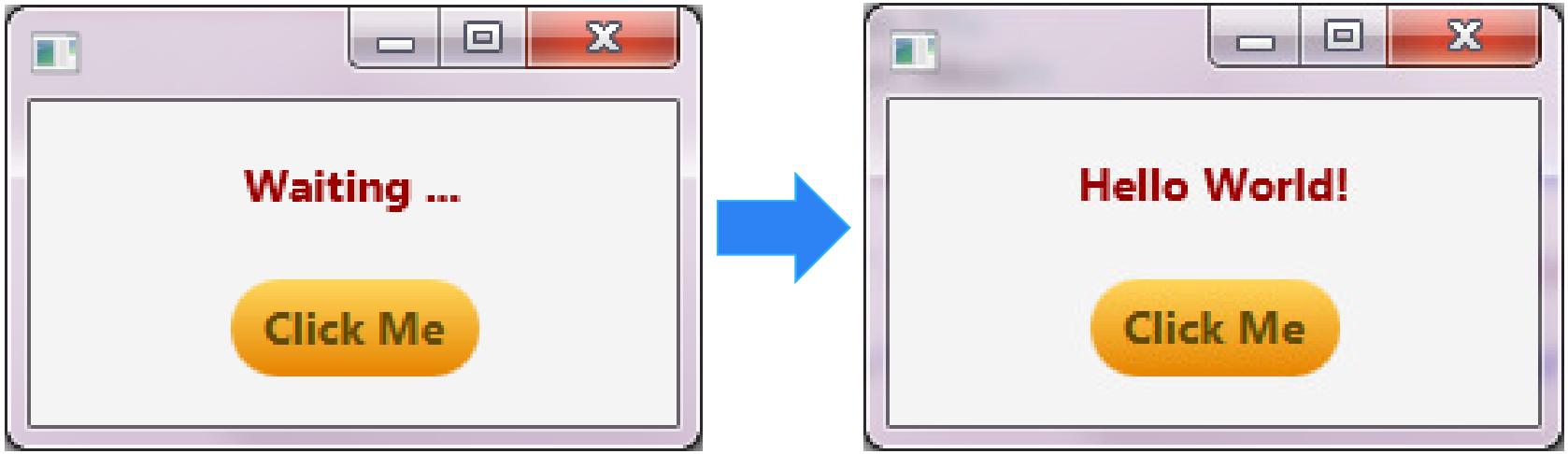
UI elements are explicitly defined

```
public class LabelDemo extends Application {

   public void start(Stage primaryStage) throws Exception {
      final TilePane pane = new TilePane();
      pane.getChildren().add(new Label("I'm a Label"));
      pane.getChildren().add(new Label("I'm another Label"));
      primaryStage.setScene(new Scene(pane, 400, 50));
      primaryStage.setTitle("Label Demo");
      primaryStage.setResizable(true);
      primaryStage.show();
   }
   public static void main(String[] args) {
      launch(args);
   }
}
```

Separation of layout and logic



ClickMe Example



Package clickMeSimple

- *Main.java*
- *SimpleController.java*
- *View.fxml*
- *application.css*

Main.java

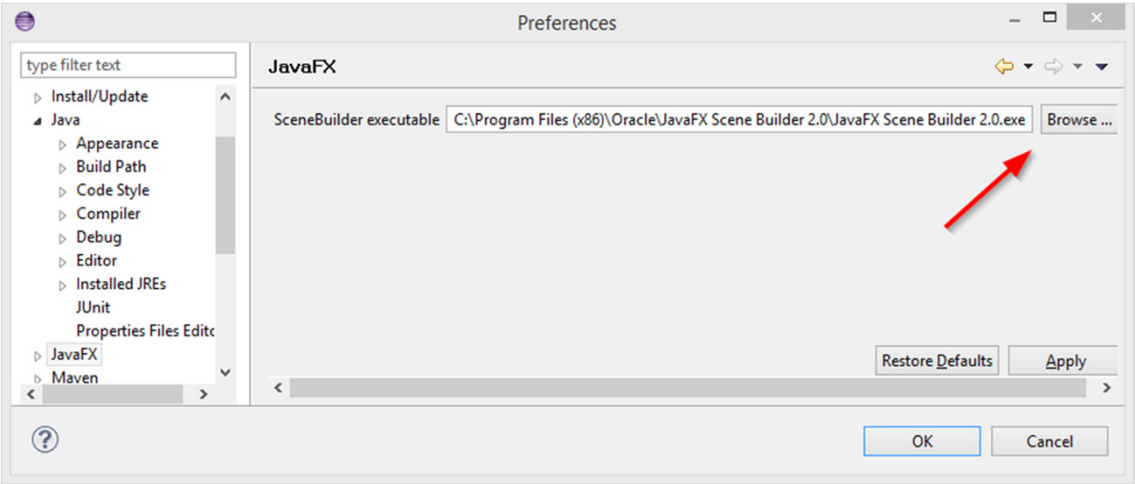
```
public class Main extends Application {
   public void start(Stage primaryStage) {
     try {
       FXMLLoader loader =
           new FXMLLoader(getClass().getResource("View.fxml"));
       Parent root = (Parent) loader.load();
       Scene scene = new Scene(root);
       primaryStage.setScene(scene);
       primaryStage.show();
    } catch(Exception e) {
       e.printStackTrace();
  }}

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

Create an FXMLLoader instance based on the fxml file

Create the node hierarchy by calling method `load()`

JavaFX with SceneBuilder

- Scene Builder needs to be installed separately and configured in Eclipse
  - Download: http://www.oracle.com/technetwork/java/javafxscenebuilder-1x-archive-2199384.html#javafx-scenebuilder-2.0-oth-JPR
  - Configure: Navigate to the *JavaFX* preferences. Specify the path to your Scene Builder executable.



- Complete Tutorial: http://docs.oracle.com/javafx/scenebuilder/1/use_java_ides/sb-with-eclipse.htm

View.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.*?>
<?import javafx.scene.text.*?>
<?import java.lang.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<VBox xmlns="http://javafx.com/javafx/8"
      xmlns:fx="http://javafx.com/fxml/1"
      alignment="CENTER" prefHeight="100.0" prefWidth="200.0"
      fx:controller="ClickMeSimple.SimpleController">
   <children>
      <Text fx:id="message" text="Waiting ..." textAlignment="CENTER"/>
      <Button alignment="CENTER" defaultButton="true"
        onAction="#handleClickMe"
        text="Click Me">
   </children>
</VBox>
```

Imports

Default Namespace

fxml Namespace

Controller class

Name of the element

Event Handler

SimpleController.java

```
public class SimpleController {

   @FXML
   private Text message;

   @FXML
   protected void handleClickMe(ActionEvent event) {
     this.message.setText("Hello World!");
   }
}
```

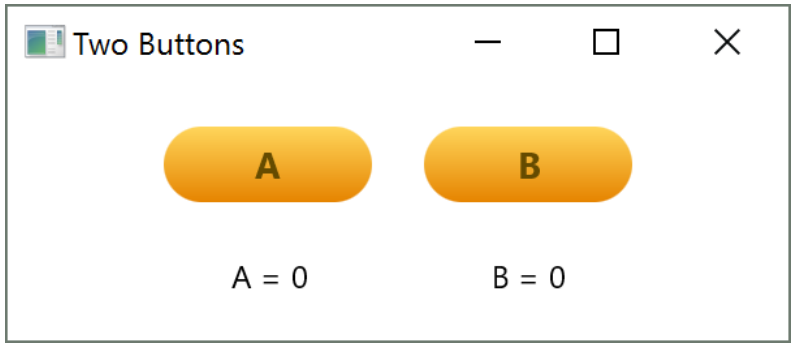
To use an UI element, declare it as private field with the fx:id as name and the type of the UI element

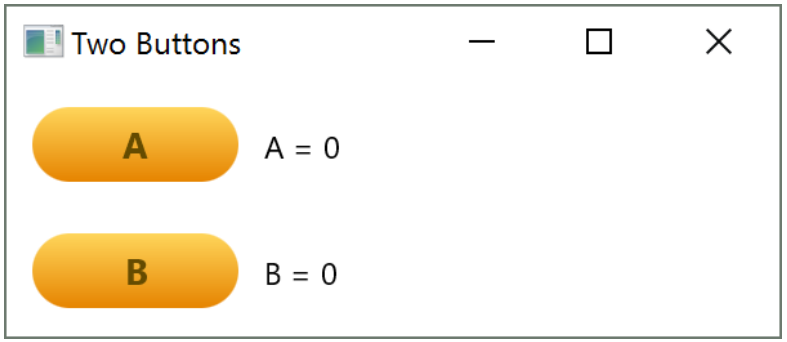
Define a method with a parameter ActionEvent as event handler

Exercise

Write a JavaFX application with two buttons A and B. Each time button A or button B is clicked, a counter is increased and the number of clicks is displayed (separately for button A and B).

Separate the logic from the layout! Provide two different UI descriptions (View.fxml)!

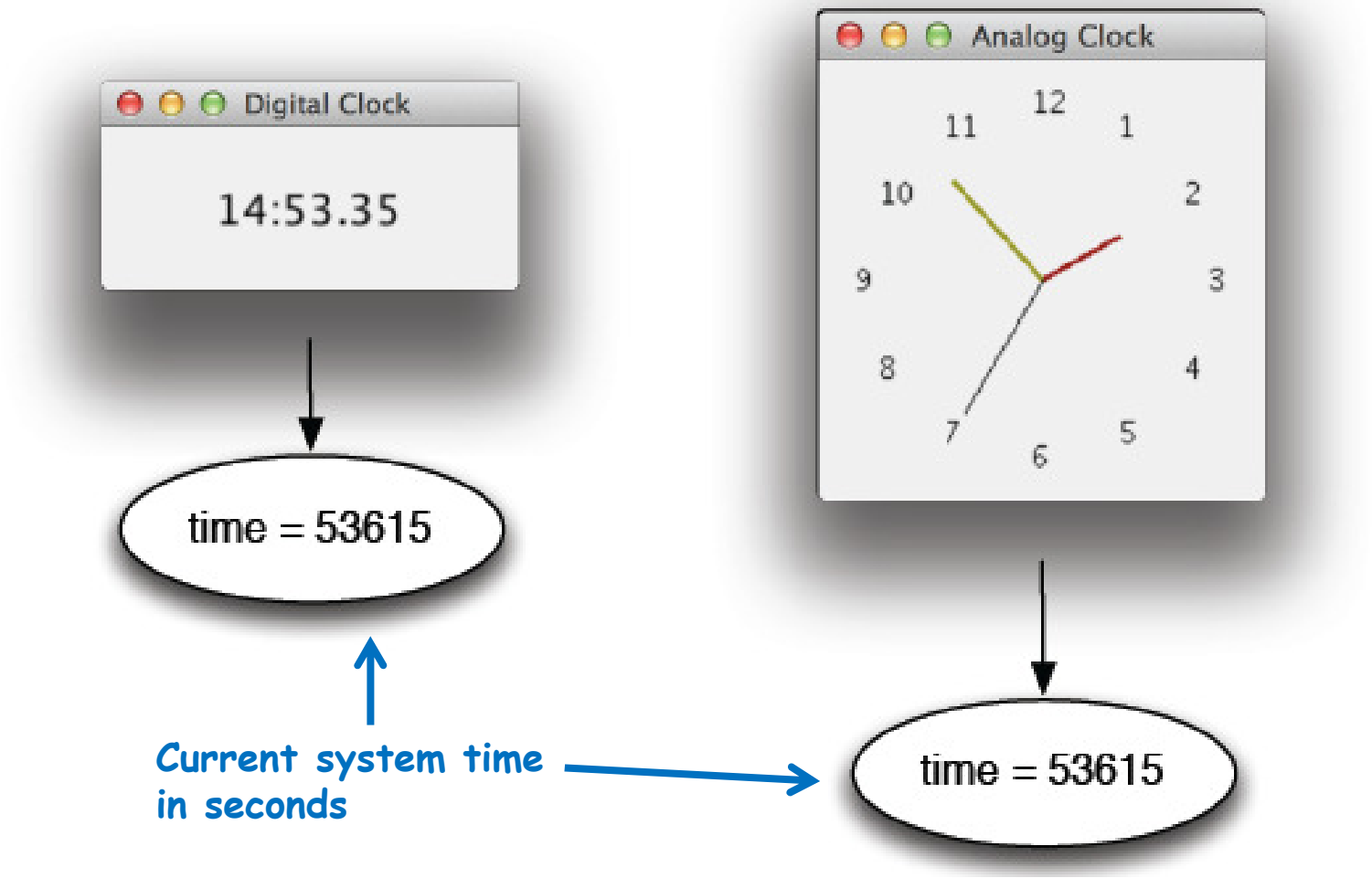




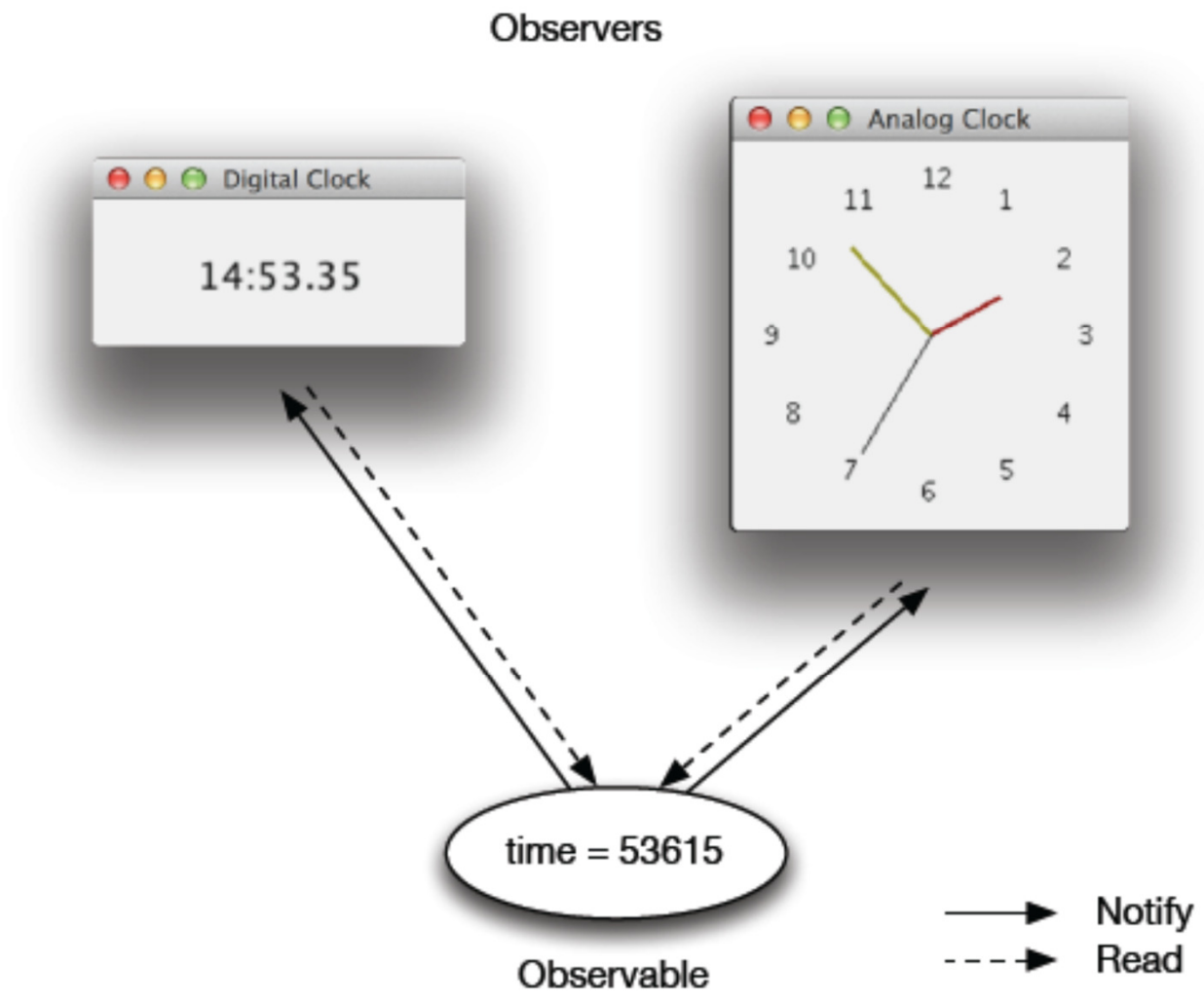
Outline

- Separation of layout and logic
- Observer Design Pattern
- MVC Design Pattern
- Styling

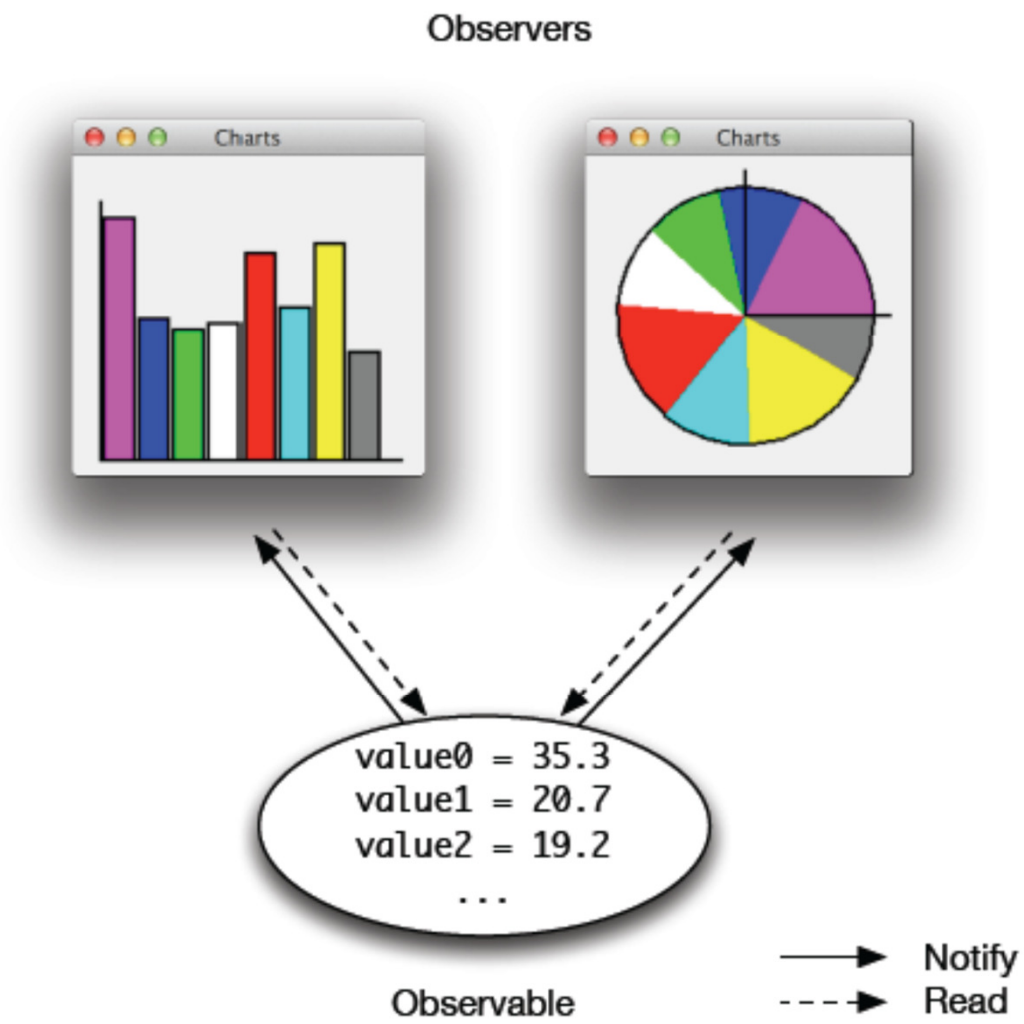
Observer Design Pattern



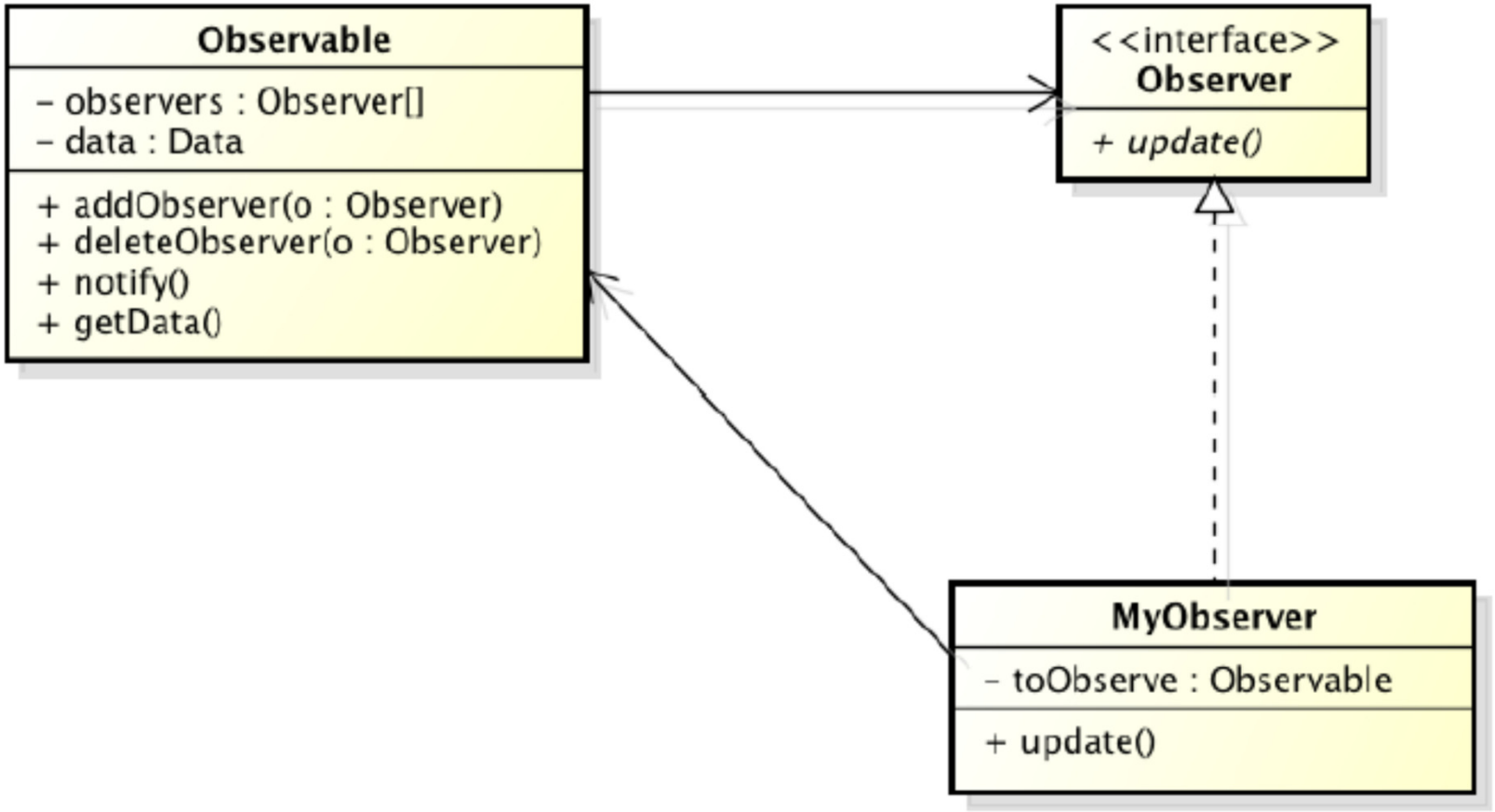
Observer Design Pattern



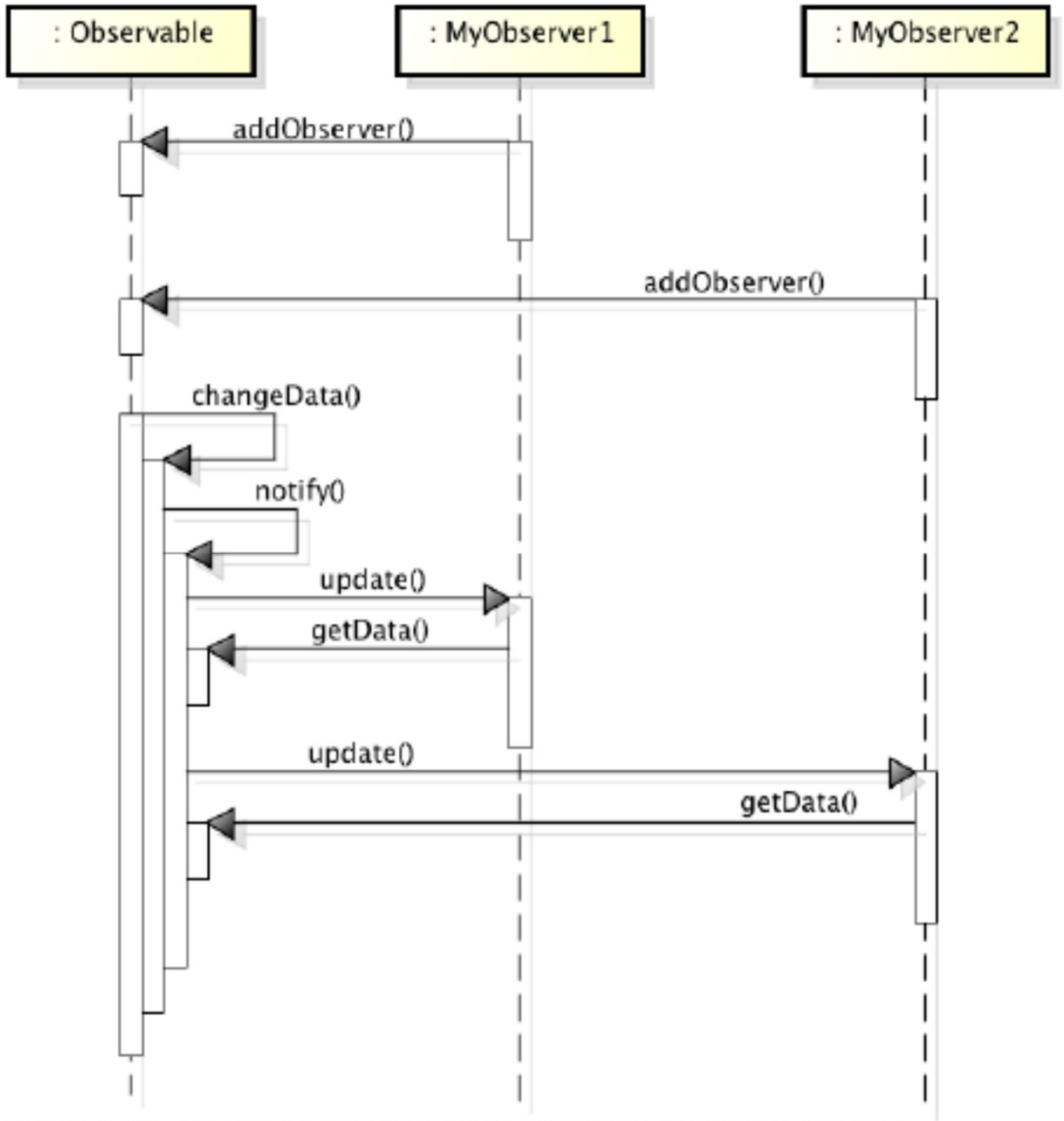
Observer Design Pattern



Observer Class diagramm



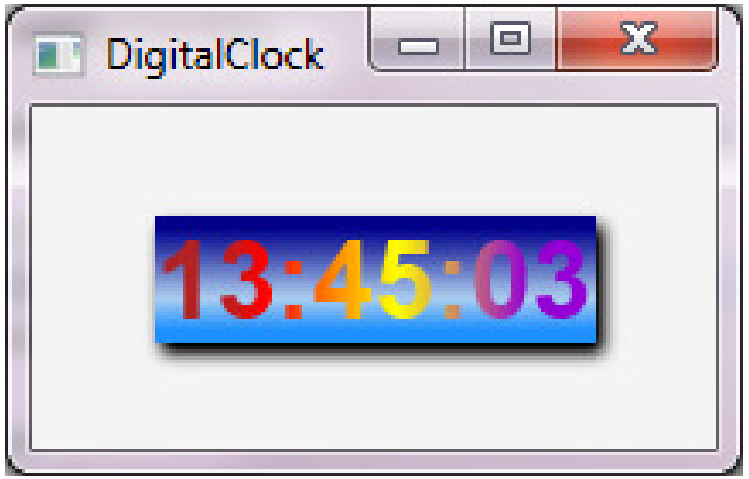
Observer Sequence diagram



Observer Design Pattern in Java

- The Observer Design Pattern is very easy to implement in Java
- Java supports the functionality of the pattern with the following 2 classes:
  - **`java.util.Observable`** - base class for *Observables*
  - **`java.util.Observer`** - Interface for *Observer*
- A concrete *Observable* has to call only the methods `setChanged()` and `notifyObservers()` after a change of its internals
- The base class *Observable* realizes automatically all necessary functionalities.

Example DigitalClock



Package digitalClock

- *Main.java*
- *DigitalClock.java*
- *Time.java*

Observable Time.java 1/2

```
import java.util.Observable;
public class Time extends Observable implements Runnable {
   private long time;

   public Time() {
      this.time =System.currentTimeMillis();
      Thread t = new Thread(this);
      t.setDaemon(true);
      t.start();}

   public void run() {
      while (true) {
        try {
          Thread.sleep(1000);
          this.increaseTime();
          } catch (InterruptedException e) {}
      }}
```

Initialize the object and create a Thread

Update the time every second

Observable Time.java 2/2

```
   private void increaseTime() {
      this.time =System.currentTimeMillis();
      // Important:
      // Call setChanged() before calling notifyObservers()
      this.setChanged();
      this.notifyObservers();
   }

   public long getTime() {
       return this.time;
   }
}
```

Inform the observers

Used by the observers the get the latest values
Corresponds to the `getData()` method

Observer DigitalClock

```
import java.util.Observable;

public class DigitalClock extends Label implements Observer {
  private final SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss a");

  public DigitalClock(final Time time) {
     time.addObserver(this);
  }

  public void update(Observable o, Object arg) {
   // Make sure, the GUI is updated in the JavaFX Application Thread!
   Time time = (Time) o;
   Platform.runLater(() -> {
        this.setText(sdf.format(time.getTime()));   });
  }
}
```

Time formatter

Register the observer

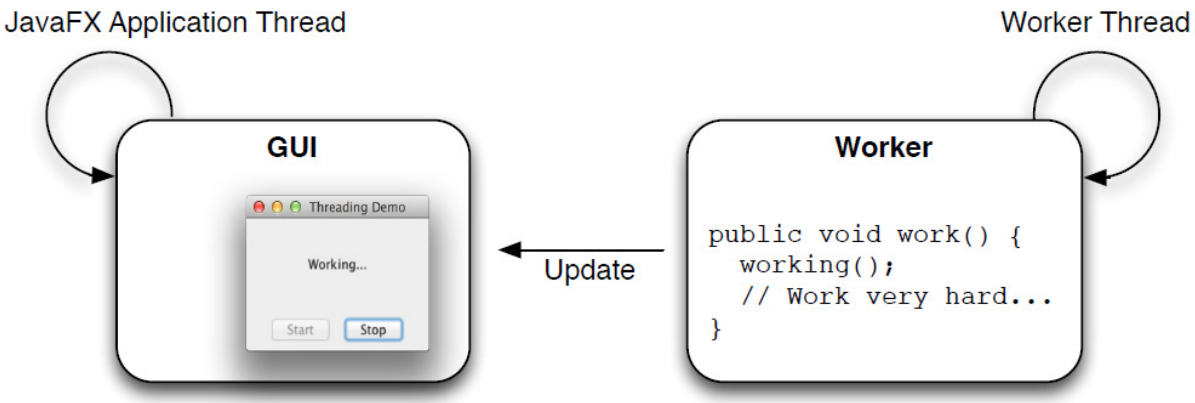
Update() method is called when Observable is changed

Cast the observable

Pull the data

Thread-safety

- **Der Scene Graph von JavaFX ist nicht thread-safe!**
  - Der Scene Graph und somit das GUI darf nur vom JavaFX Application Thread verändert und aktualisiert werden
  - Wird der Scene Graph von einem anderen Thread verändert, wird eine Exception geworfen
  - Mittels `Platform.runLater()` können Ereignisse in die Event-Queue geschrieben werden, welche anschliessend vom UI Thread verarbeitet werden



Main.java

```
public class Main extends Application {

   public void start(Stage primaryStage) throws Exception {
      VBox pane = new VBox(5);
      Time time = new Time();
      DigitalClock clock = new DigitalClock(time);

      pane.getChildren().add(clock);
      primaryStage.setScene(new Scene(pane, 200, 100));
      primaryStage.show();
   }

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

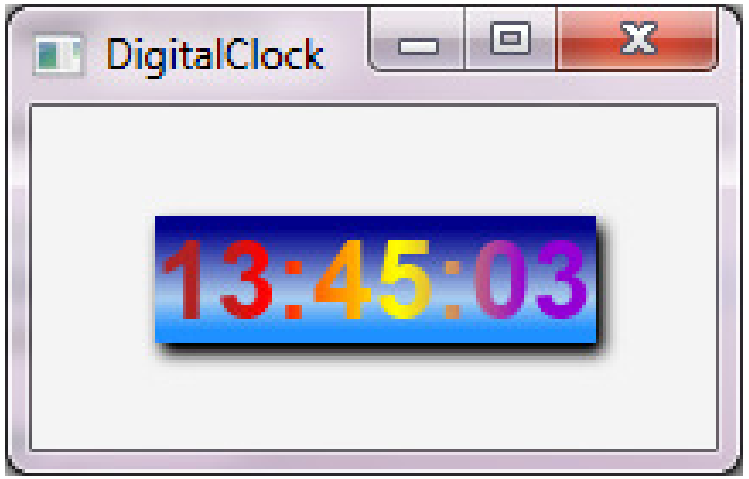
Observable

Observer

Pull or Push

- **Pull**
  - The observer gets the changed data explicit form the observable
  - Ex. `DigitalClock` calls `time.getTime()`
- **Push**
  - The observable hands over the changed data with the update request.
  - A **Data Transfer Object** is used, e.g. the `TimeObject`

Example DigitalClock (push)



Package digitalClockpush

- *Main.java*
- *DigitalClock.java*
- *Time.java*
- *TimeObject.java*

Data Transfer Object

- Stores data which are transferred from the observable to the observer
- Is immutable. (After creation the data can only be read).

```
public class TimeObject {
    private long time;

    public TimeObject(long time) {
        this.time = time;
    }

    public long getTime() {
        return this.time;
    }
}
```

Data Transfer Object

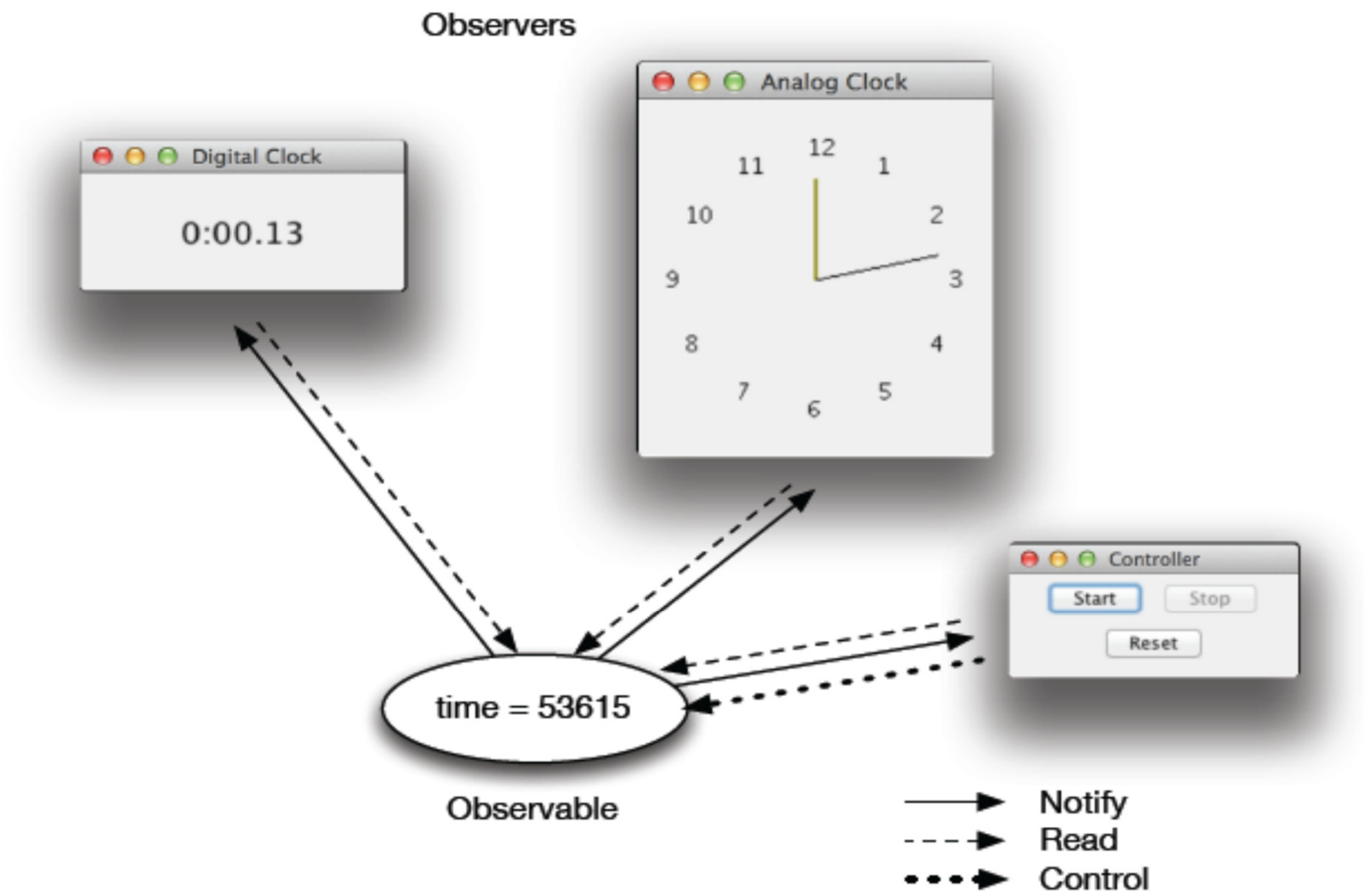
```
public class Time extends Observable implements Runnable {
...
    private void increaseTime() {
        this.time =System.currentTimeMillis();
        this.setChanged();
        this.notifyObservers(new TimeObject(this.time));   }
...
}

public class DigitalClock extends Label implements Observer {
...
    public void update(Observable o, Object arg) {
       TimeObject timeObject = (TimeObject) arg;
       Platform.runLater(() -> {
           this.setText(sdf.format(timeObject.getTime()));
        });
...
}
```

Cast the argument

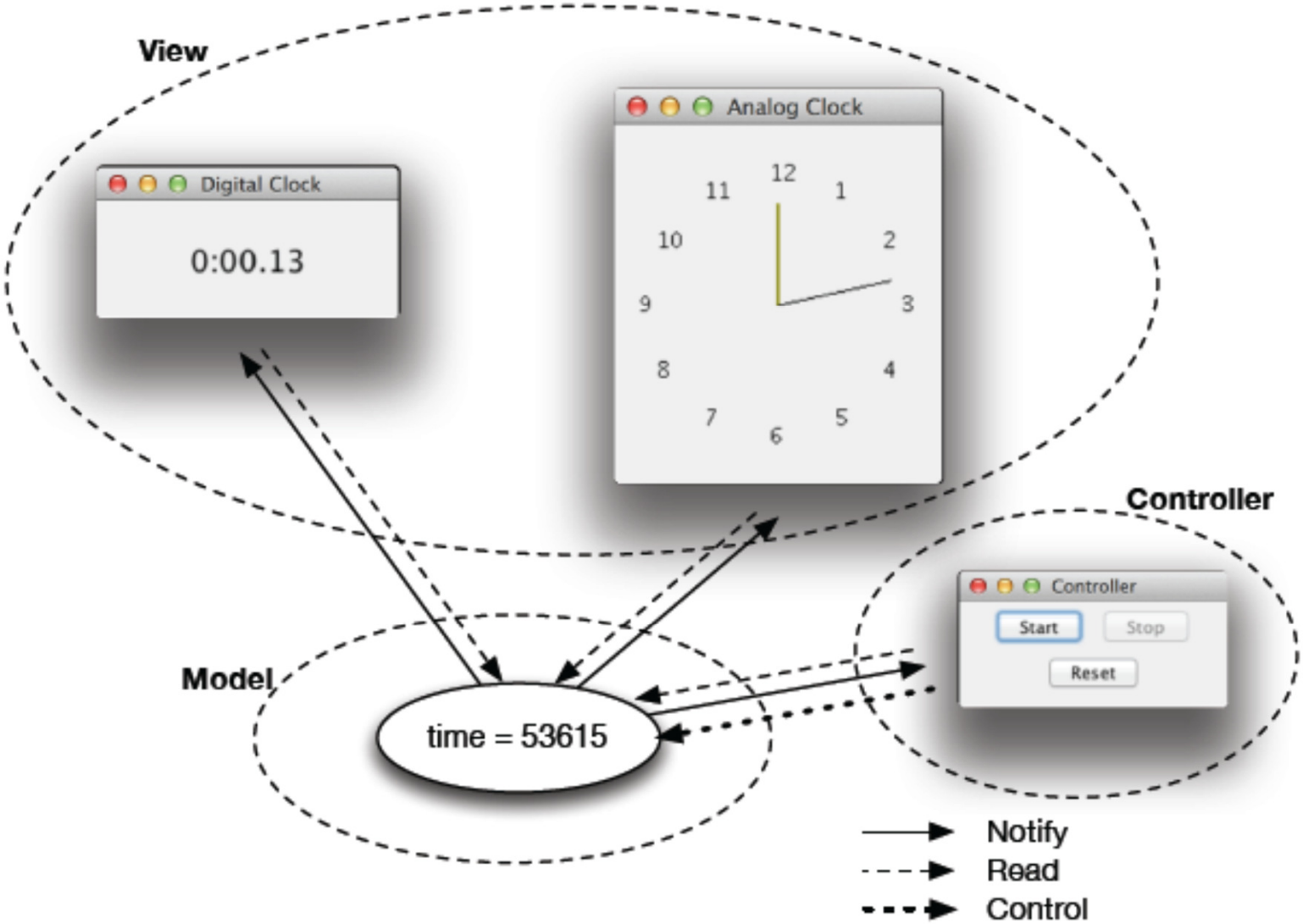
Outline

- Separation of layout and logic
- Observer Design Pattern
- MVC Design Pattern
- Styling

MVC Pattern


Observers
Digital Clock
0:00.13
Analog Clock
12
1
2
3
4
5
6
7
8
9
10
11
Controller
Start
Stop
Reset
time = 53615
Observable
Notify
Read
Control

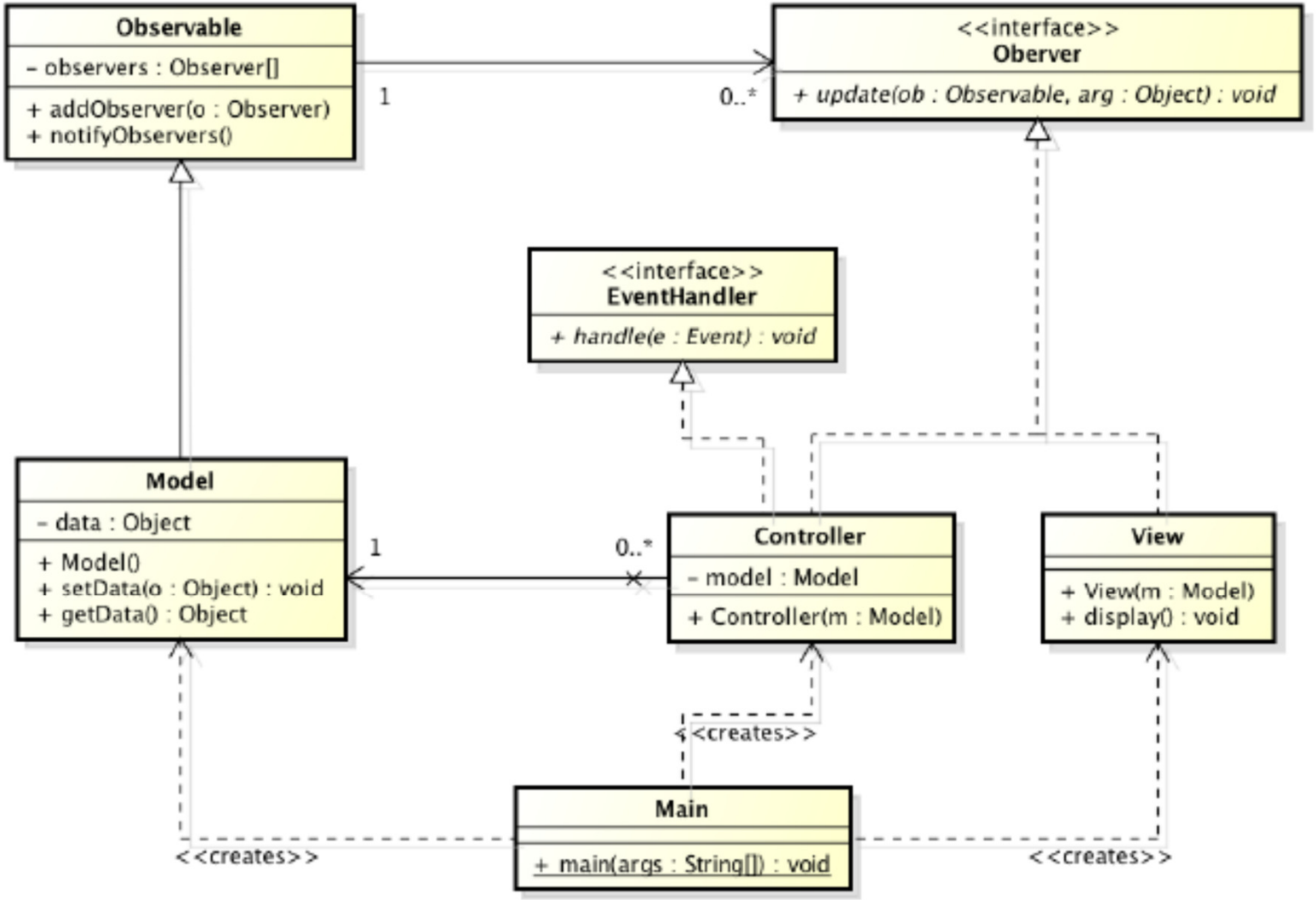
MVC Pattern


View
Digital Clock
0:00.13
Analog Clock
12
1
2
3
4
5
6
7
8
9
10
11
Controller
Controller
Start
Stop
Reset
Model
time = 53615
Notify
Read
Control

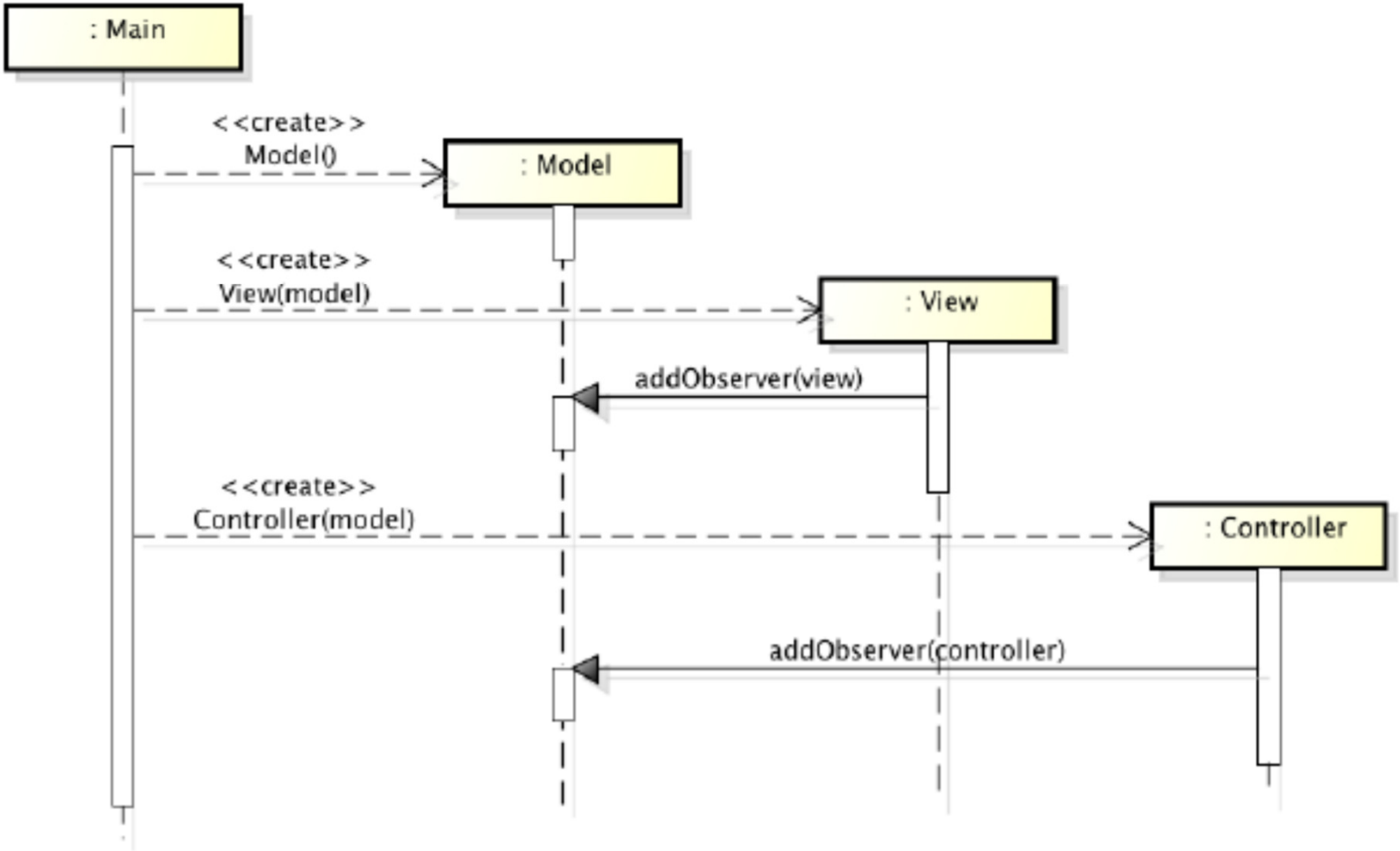
MVC Pattern

- **M**odel
  - Data and data processing
- **V**iew
  - Presentation of the data
- **C**ontroller
  - User Input
- Goal: **Decoupling** of the different parts of the application

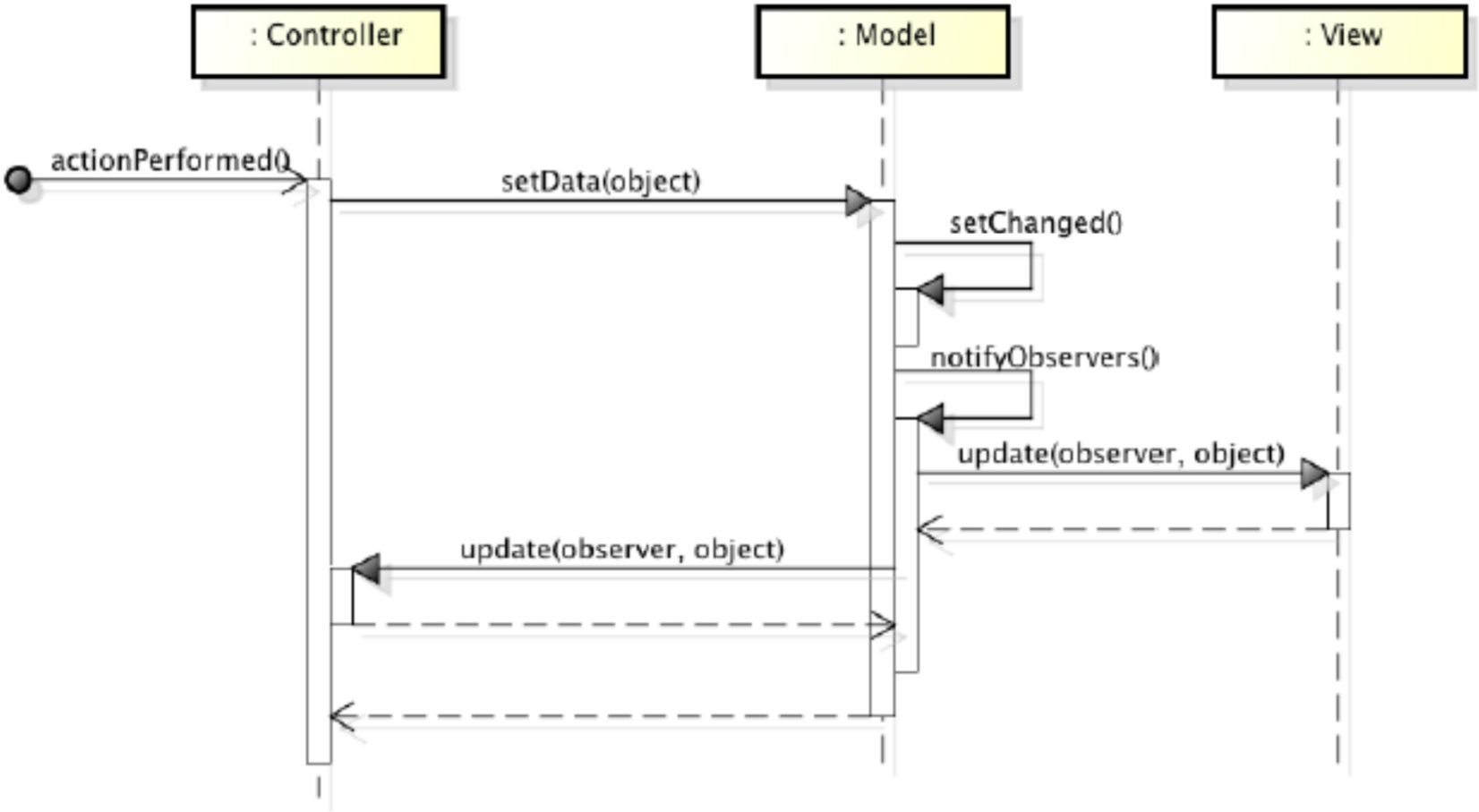
MVC class diagram



MVC Initialization – sequence diagram



MVC event handling – sequence diagram



Example StopWatch MVC

Package stopwatch_mvc

- *Main.java*
- *Controller.java*
- *Timer.java*
- *Stopwatch.java*
- *Stopwatch_blue.fxml / Stopwatch_yellow.fxml*
- *blue.css / yellow.css*

MVC with JavaFX

- How to get to the Model into the controller?
  - static methods
  - Model is implemented as Singleton
  - Model is created and initialized in the `Main` and passed to the `Controller`
  - In bigger projects, it can be reasonable to use
    - Dependency Injection (http://martinfowler.com/articles/injection.html)
    - Event Bus (https://github.com/google/guava/wiki/EventBusExplained)

MVC with JavaFX

Example `Main`: Model is created and initialized in the `Main`

- Variant 1: pass the model to the Controller using an `init` method

```
Model model = new Model();
// Create an FXMLLoader instance based on the FXML
FXMLLoader loader = new FXMLLoader(
  getClass().getResource("View.fxml"));

// Create the node hierarchy by calling load
Parent root = (Parent) loader.load();

// Pass the model to the controller by calling init
loader.<Controller>getController().init(model);
```

Type argument for generic method

Pass the model

MVC with JavaFX

Example `Controller`: Model is passed using an `init` method

```
public class Controller implements Observer {
  private Model model;
  @FXML private Label label;

  // Called by FXMLLoader after the node hierarchy has been created
  @FXML protected void initialize() {   this.label.set...   }

  // Called by the Main after initialize() has been called
  public void init(Model model) {
    this.model = model;
    this.model.addObserver(this);
    this.label.setText(this.model.get... ); }

  @Override public void update(Observable o, Object arg) {
   // Update the gui in response to model changes via runLater()
    Platform.runLater(() -> { this.label.setText(this.model.get... );}); }
}
```

MVC with JavaFX

- Variant 2: pass the Model to the Controller using the constructor

```
Model model = new Model();
 // Create an FXMLLoader instance based on the FXML
 FXMLLoader loader = new FXMLLoader(
   getClass().getResource("View.fxml"));

 // Set Controller explicitly (fx:controller must be
 // removed from FXML!)
 loader.setController(new Controller(model));
  Parent root = (Parent) loader.load();
```

Pass the model

Outline

- Separation of layout and logic
- Observer Design Pattern
- MVC Design Pattern
- Styling

Styling in JavaFX

- Styling and Theming is defined in JavaFX Cascading Style Sheets (CSS)
- Set the main theme with:

```
Application.setUserAgentStylesheet(<stylesheet>);
```

Must be an URL !!!

- There 2 default styles
  - Modena (`STYLESHEET_MODENA`) and
  - Caspian (`STYLESHEET_CASPIAN`)

- You can add an arbitrary number of other stylesheet to your scenes

```
scene.getStylesheets().add(
getClass().getResource("stylesheet.css").toExternalForm());
```

JavaFX CSS

- JavaFX CSS is based on **selectors** and **styling properties**

```
<selector> {
    <property>: <value>;
}
```

- There are 2 selector types: **`id`** and **`class`**
  - Ids are marked with **#**
  - Classes are marked with **.**

```
/* Id selector */
#stop-button {...}
/* Class selector */
.button {...}
```

  - CSS classes have **nothing** to do with Java classes!

JavaFX CSS

- Many controls have default classes
  - Button: `button`
  - Label: `label`
  - CheckBox: `check-box`
  - …
- You can add your own stylesheet classes

```
Button stop = new Button("Stop");
stop.getStyleClass().add("fancy-button");

// FXML
<Button styleClass="fancy-button" ... />
```

JavaFX CSS

- Use pseudo class selectors (marked with :) to style the different states of a control

```
.button:hover {...}
.button:pressed {...}
.button:disabled {...}
...
```

- Use selector patterns to style controls together

```
/* Style all labels and buttons */
.label, .button {...}
/* Style buttons in a BorderPane */
.border-pane .button {...}
/* Style buttons who's parent is an HBox */
.hbox > .button {...}
/* Id selectors are stronger than class selectors */
#stop-button {...}
```

JavaFX CSS

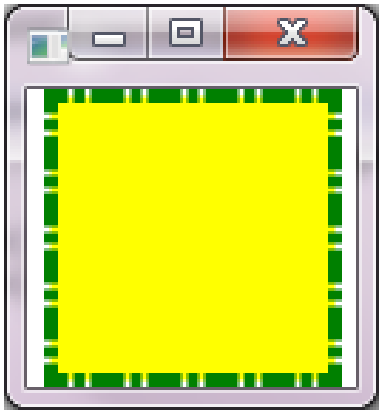
- Styling Properties

```
.button {
-fx-background-color: red;
-fx-text-fill: black;
-fx-font-size: 12px;
...
}
.button:hover {
-fx-backgrond-color: #bb0000;
...
}
```

- JavaFX CSS Reference: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html

Example

- *RectangleExample.java*
- *rectangle.css*
- *rectangle2.css*



Parser Warnings

- When the JavaFX CSS parser encounters a syntax error, a warning message is emitted
- For example
  - `WARNING: com.sun.javafx.css.parser.CSSParser declaration Expected '<percent>' while parsing '-fx-background-color' at ?[1,49]`
- The cryptic '?[1,49]' pertains to the location of the error