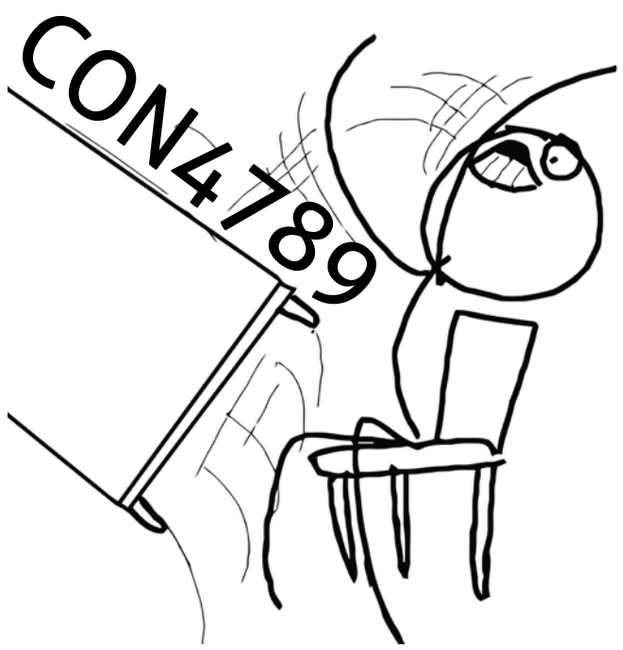
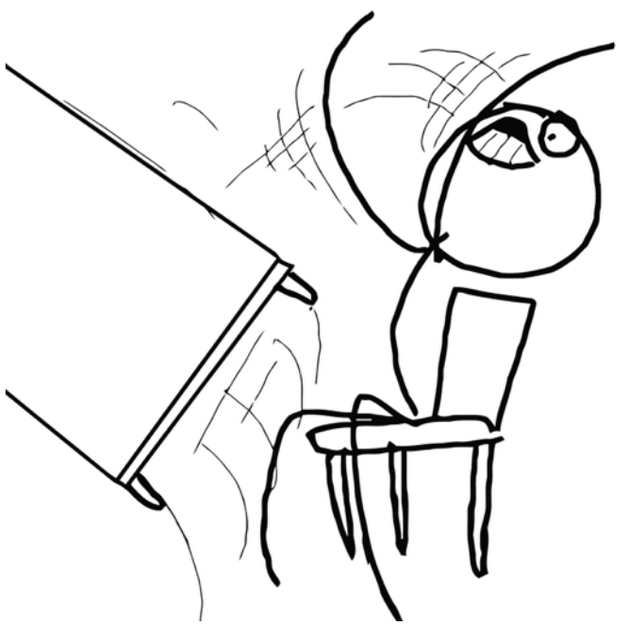
JavaFX Pitfalls

Know them, or you will get angry.



Warning

This presentation contains a heavy load of code



Speaker

Alexander Casall

Software Architect & Scrum Product Owner

alexander.casall@saxsys.de

Saxonia Systems

So geht Software.

m v v m FX

@sialcasa

Custom Software Development Company in Germany
Dresden, Munich, Berlin, Hamburg, Leipzig…
225 employees
3 Scrum Teams in JavaFX Projects
±**7** consultants supporting JavaFX projects on customer side

Speaker

Andy Moncsek

JACPFX

Senior Consultant Application Development

trivadis

makes IT easier.

Trivadis AG Switzerland - Zürich
600 employees in Switzerland, Germany and Austria
consulting, development, BI, infrastructure, operation

andy.moncsek@trivadis.com

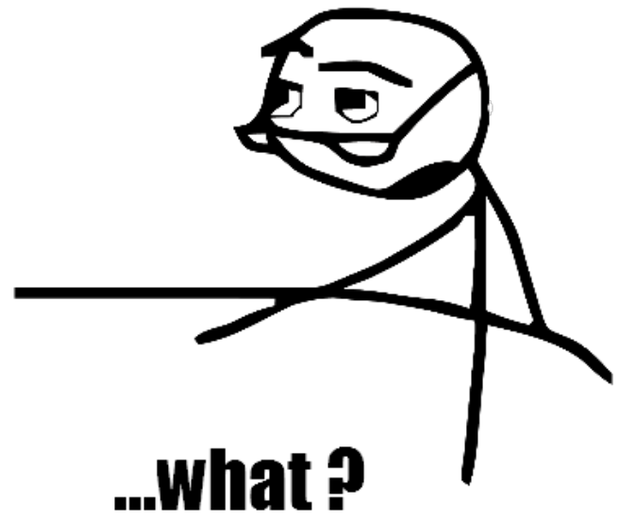
FXML / Caching

Memory

Threading and Services

Pixel Operations

Examples
https://github.com/sialcasa/javafx-pitfalls



Before we start:

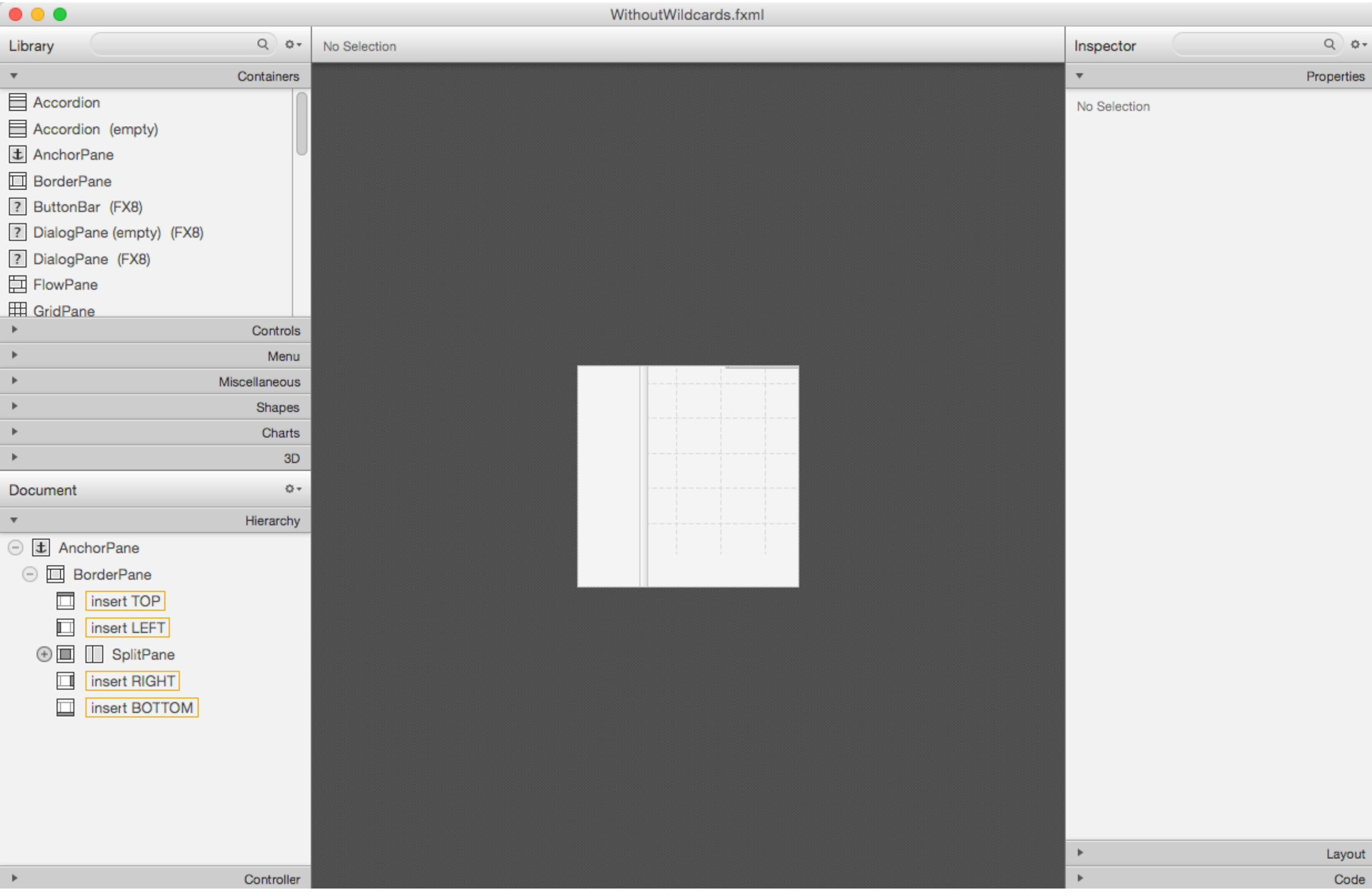
If you are looking for common performance tipps regarding JavaFX

https://wiki.openjdk.java.net/display/OpenJFX/Performance+Tips+and+Tricks

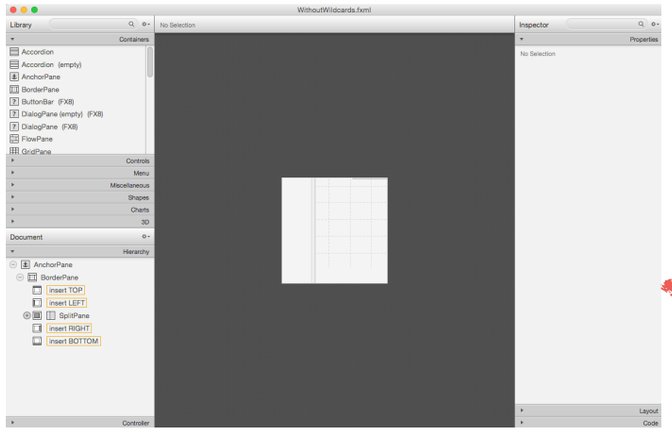
FXML / Caching

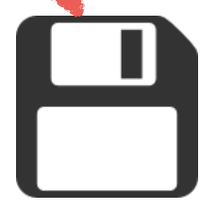
FXML / Caching

FXML-Loading Performance

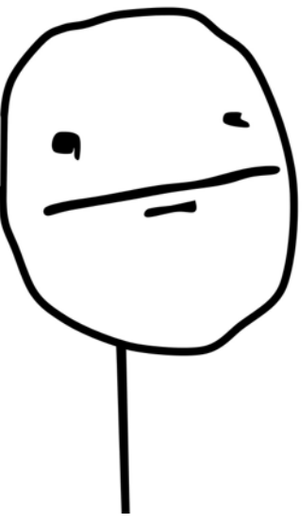

WithoutWildcards.fxml
Library
Containers
Accordion
Accordion (empty)
AnchorPane
BorderPane
ButtonBar (FX8)
DialogPane (empty) (FX8)
DialogPane (FX8)
FlowPane
GridPane
Controls
Menu
Miscellaneous
Shapes
Charts
3D
Document
Hierarchy
AnchorPane
BorderPane
insert TOP
insert LEFT
SplitPane
insert RIGHT
insert BOTTOM
Controller
No Selection
Inspector
Properties
No Selection
Layout
Code

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

<?import javafx.scene.chart.*?>
<?import javafx.scene.shape.*?>
<?import javafx.scene.text.*?>
<?import javafx.scene.control.*?>
<?import java.lang.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.layout.AnchorPane?>
```





```
<AnchorPane xmlns:fx="http://javafx.com/fxml/1" xmlns="http://javafx.com/javafx/8.0.40">
   <children>
      <BorderPane prefHeight="200.0" prefWidth="200.0">
         <center>
            <SplitPane dividerPositions="0.2979797979797796" prefHeight="160.0" prefWidth="200.0" Bord
              <items>
                <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="160.0" prefWidth="100.0" />
                <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="160.0" prefWidth="100.0">
                     <children>
                        <TabPane layoutX="-32.0" layoutY="-31.0" prefHeight="200.0" prefWidth="200.0" t
                          <tabs>
                            <Tab text="Untitled Tab 1">
                              <content>
                                <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="180.0" prefWidth
                                      <children>
                                         <TextFlow layoutX="-86.0" layoutY="-30.0" prefHeight="200.0"
                                           <children>
                                              <TitledPane animated="false" text="untitled">
                                                <content>
                                                  <AnchorPane minHeight="0.0" minWidth="0.0" prefHeig
                                                       <children>
                                                          <Hyperlink layoutX="127.0" layoutY="24.0" t
```



DEMO

Conclusion

wildcard imports are bad - check your FXML files

IntelliJ IDEA has import optimisation

We'll fix this issue on Scene Builder

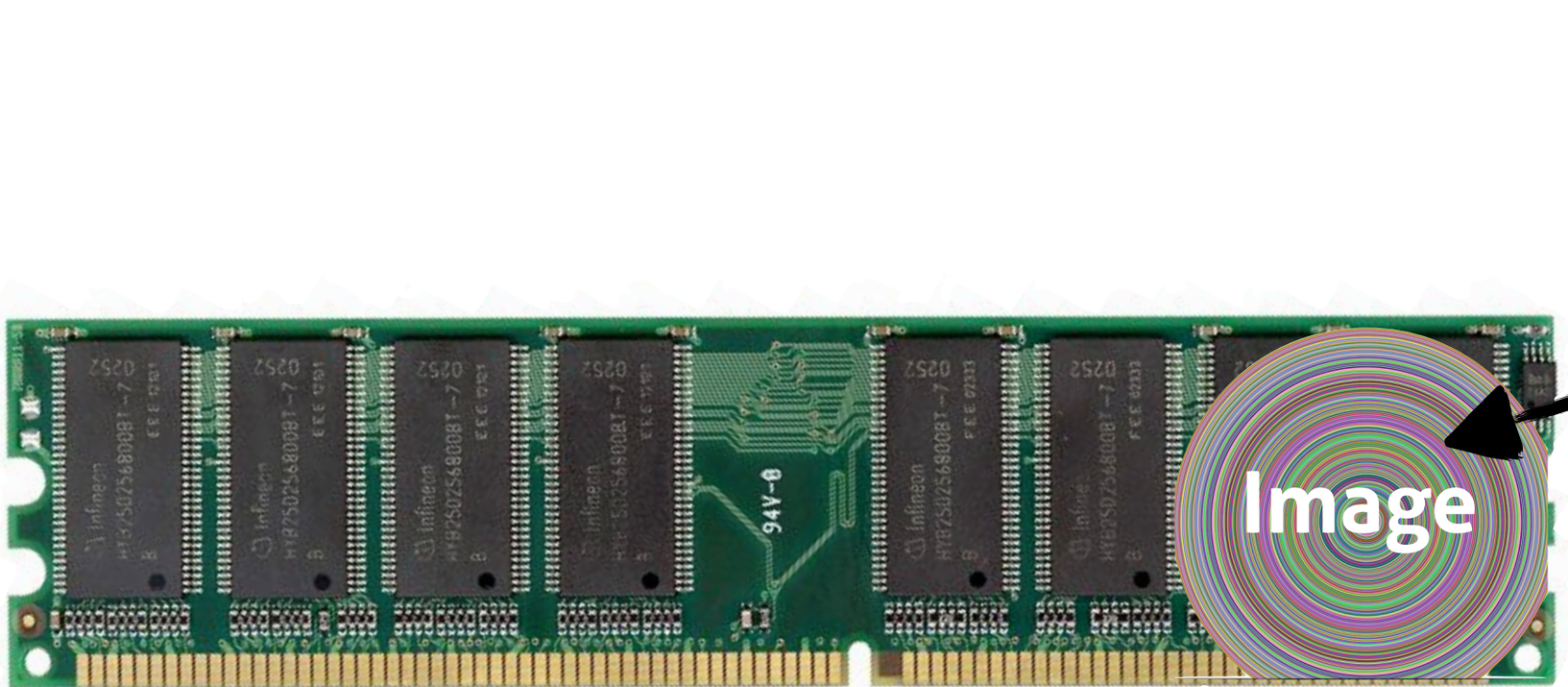
FXML / Caching

Why you don’t use caching?

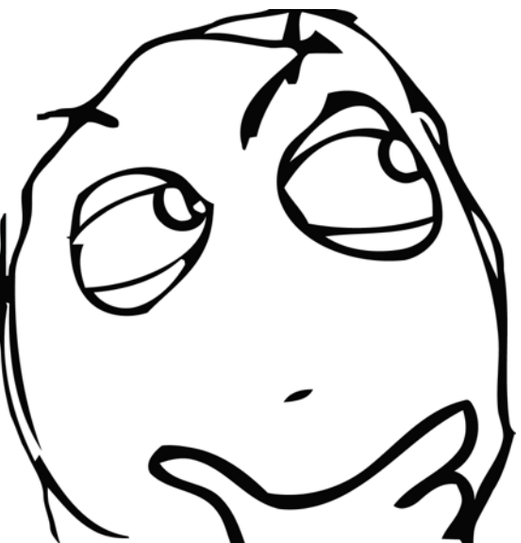
What caching does

```
extends Node
```

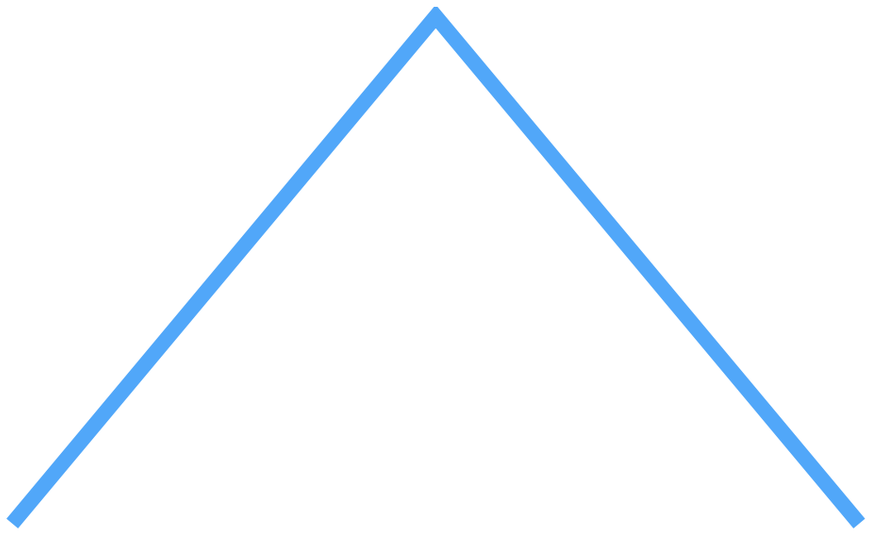
```
myMagicalCircle.setCache(true);
```



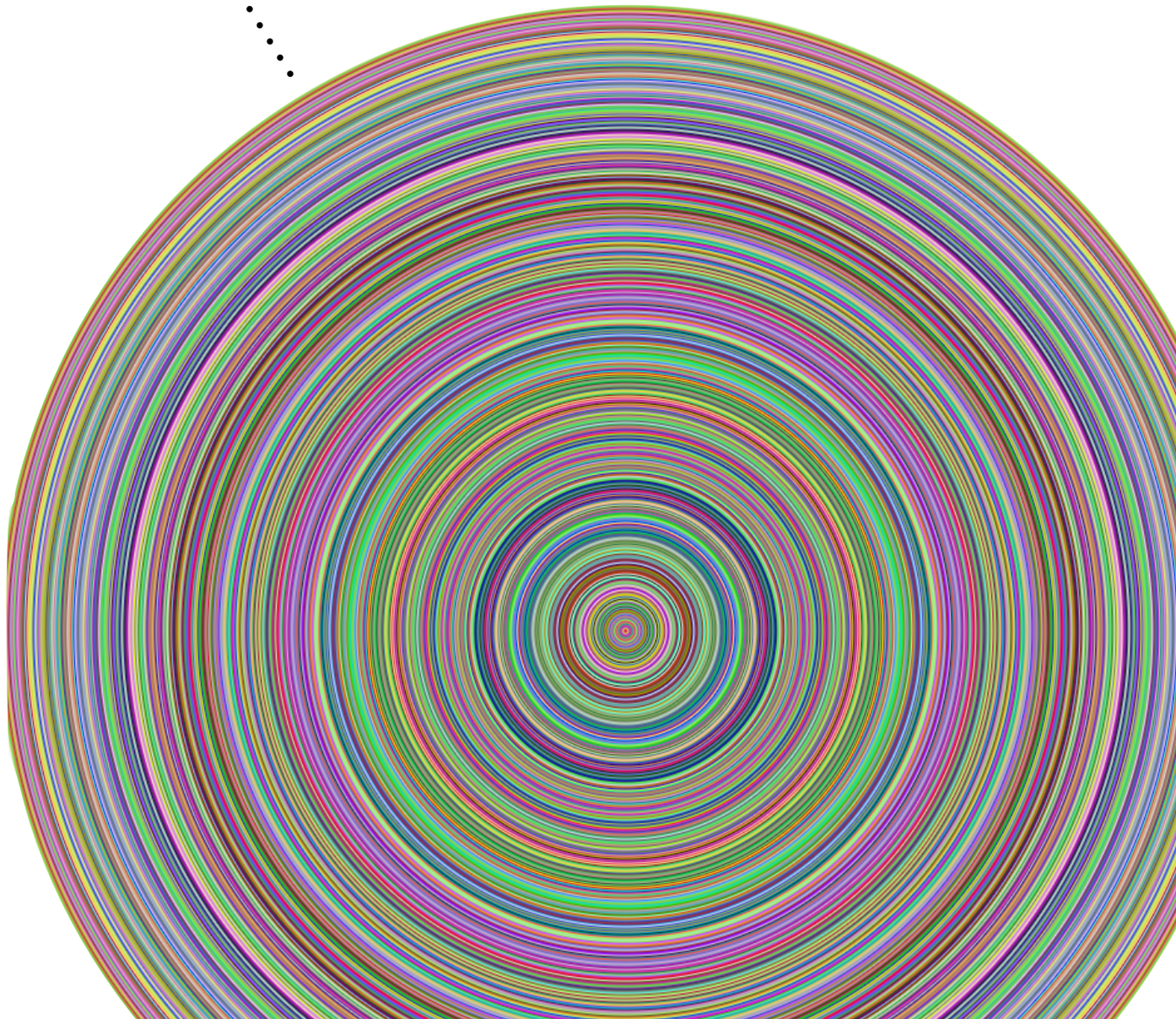
Why should you use it?



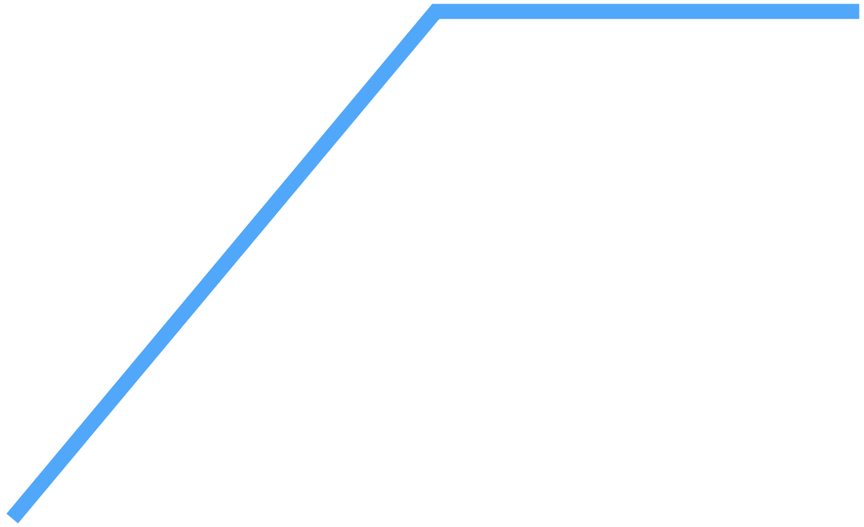
magical circles are expensive to render



GPU

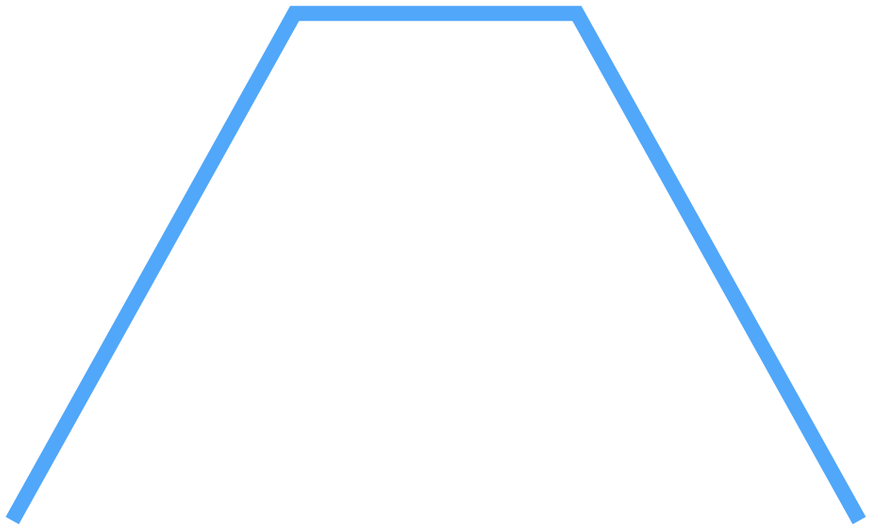


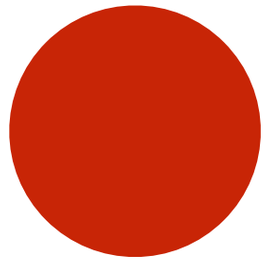
The movement of the element causes rendering effort

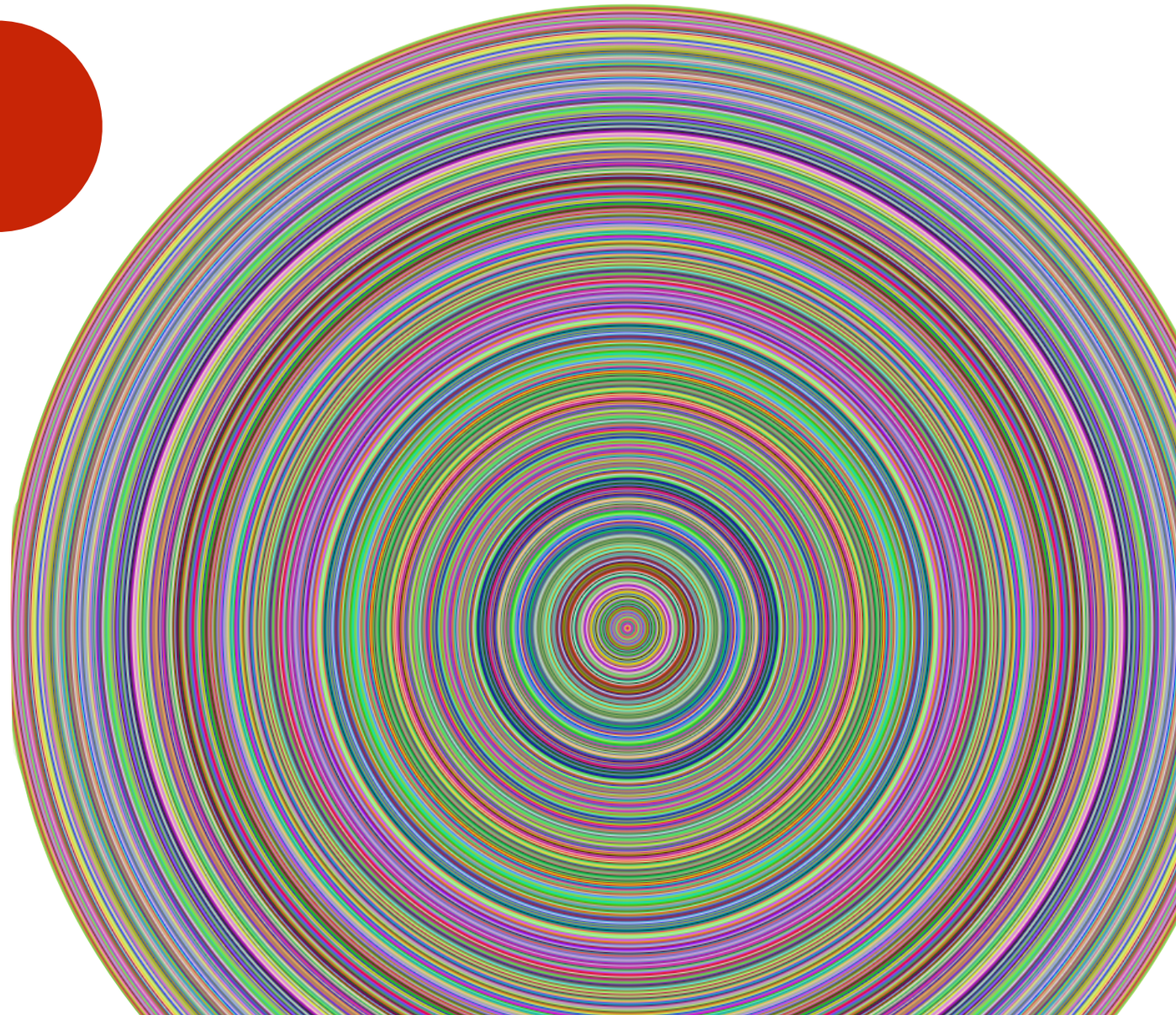


GPU

Other nodes can also cause rendering



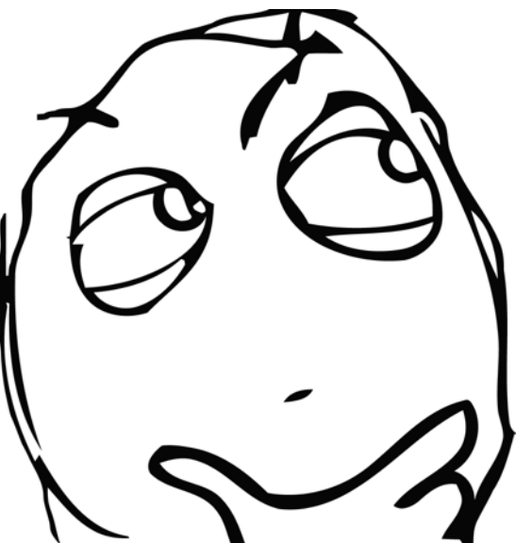




GPU

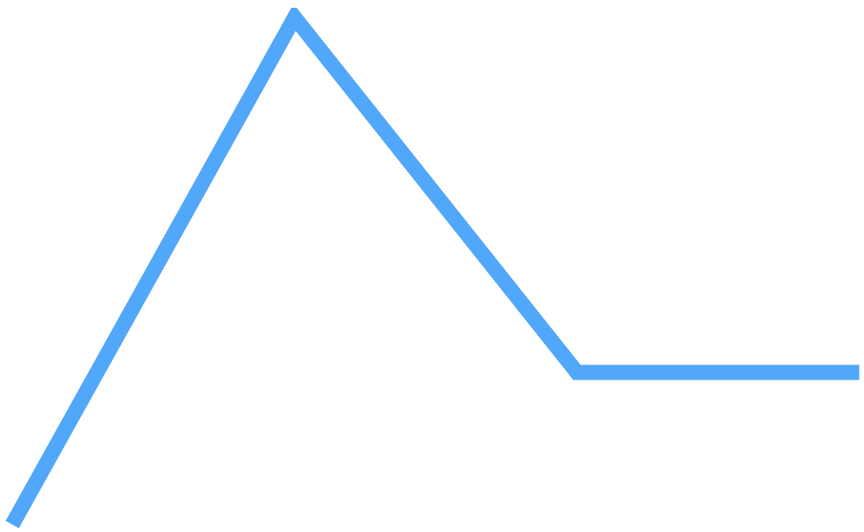
Why should you use it?

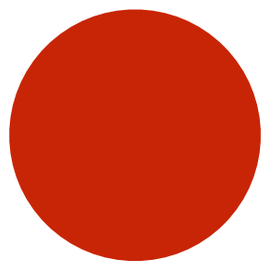
Caching avoids unnecessary rendering of nodes

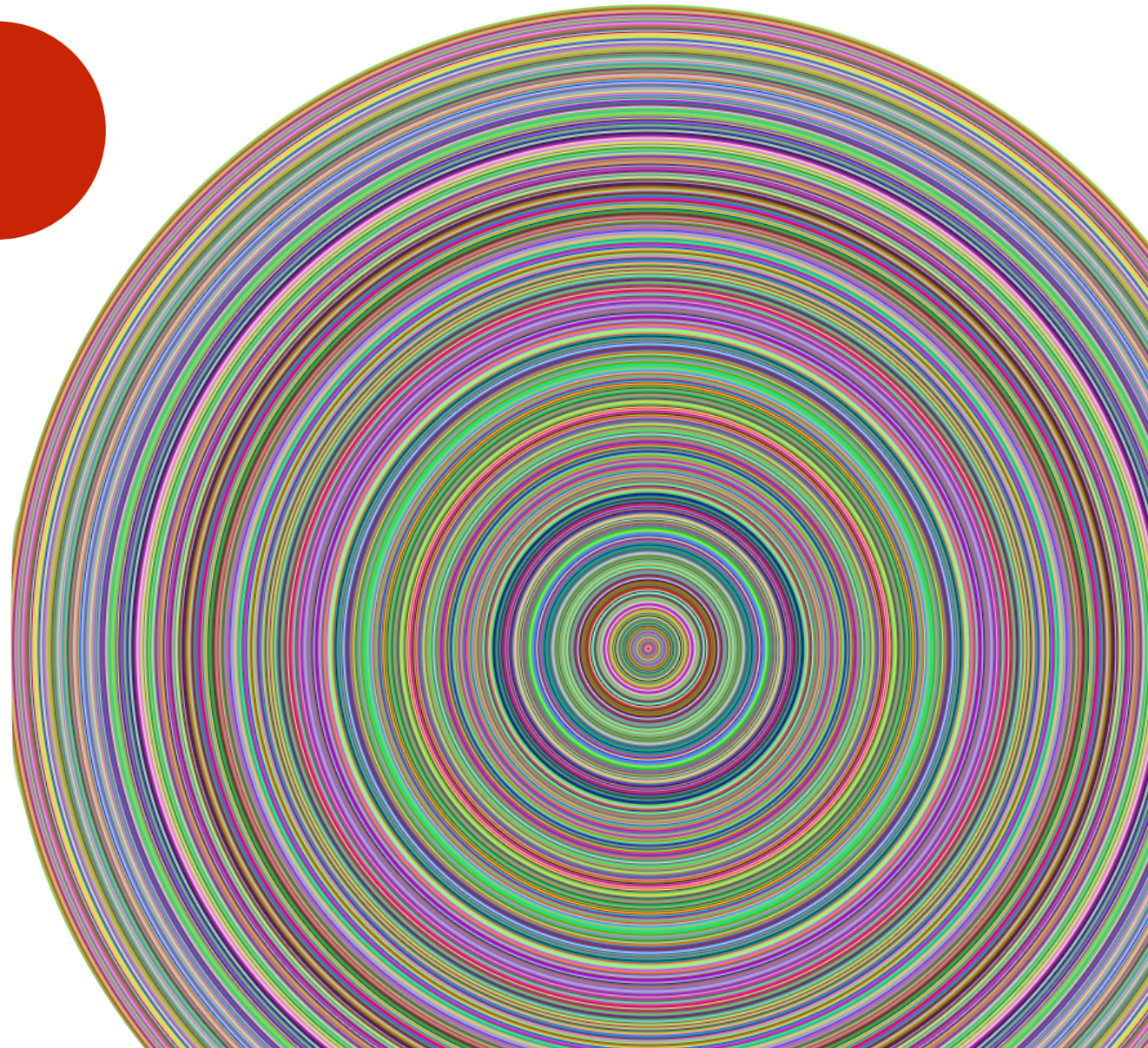


`myMagicalCircle.setCache(`**`true`**`)`

no more rendering caused e.g. by movement or overlapping nodes







GPU

Demo

Conclusion

(Don’t) cache…

simple objects which change often

complex objects which don’t change

Memory

Memory

Listener can cause memory leaks

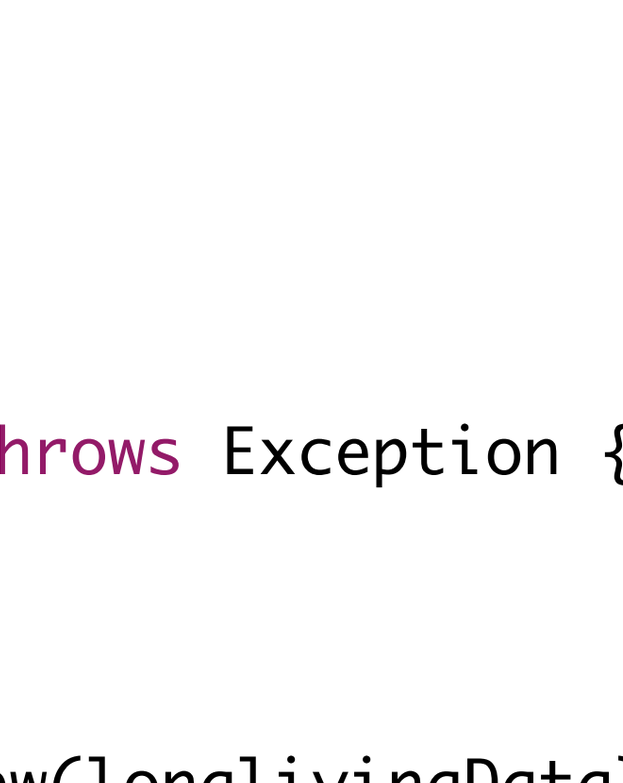
Instances of anonymous inner classes hold on to a reference to their enclosing instances even if these references are never actually used.

```
public class TemporaryView extends StackPane {

   public TemporaryView(Car dataModel) {
      dataModel.name.addListener(new ChangeListener<String>() {
         @Override
         public void changed(ObservableValue<? extends String> observable,
            String oldValue, String newValue) {
            //Do Something
         }
      });
   }
}
```

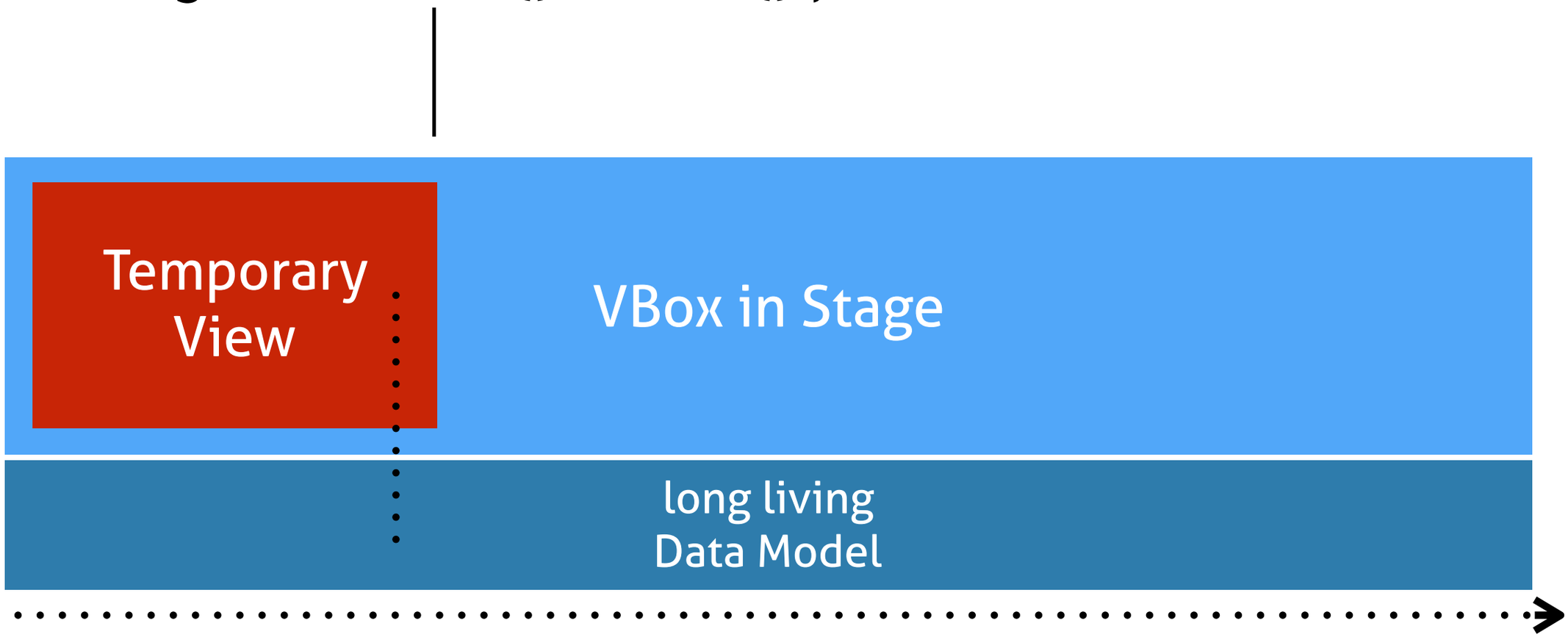
```
public class Car {
   StringProperty name = new SimpleStringProperty("Audi");
}
```

references

```
@Override
public void start(Stage primaryStage) throws Exception {

   this.longlivingData = new Car();

   this.root = new VBox(new TemporaryView(longlivingData));

   Scene scene = new Scene(root);

   primaryStage.setScene(scene);
   primaryStage.show();
}
```

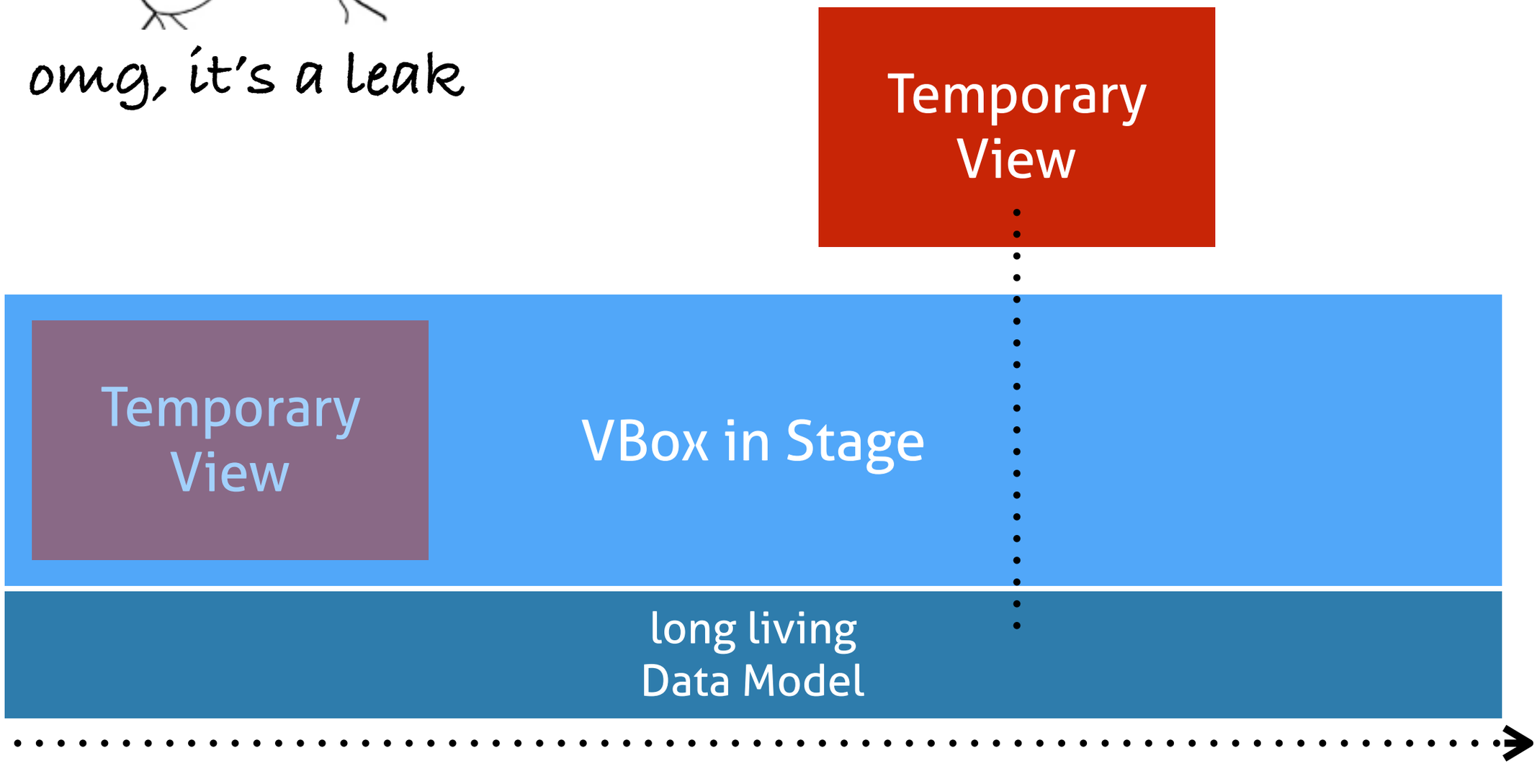


```
root.getChildren().clear();
```



omg, it's a leak

retains in memory



How to avoid it?

Use `WeakListener` (dangerous)

Do manual Housekeeping

Demo

Conclusion

Listeners can leak

WeakListeners are dangerous

manage your Listeners manually

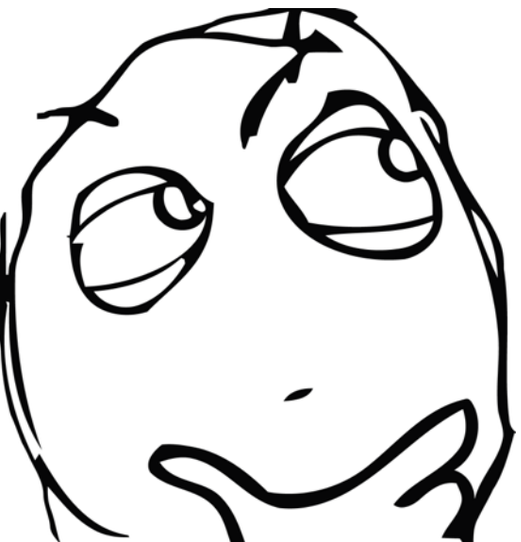
avoid anonymous implementations

Memory

Be aware of lost listeners!

The listener get's garbage collected

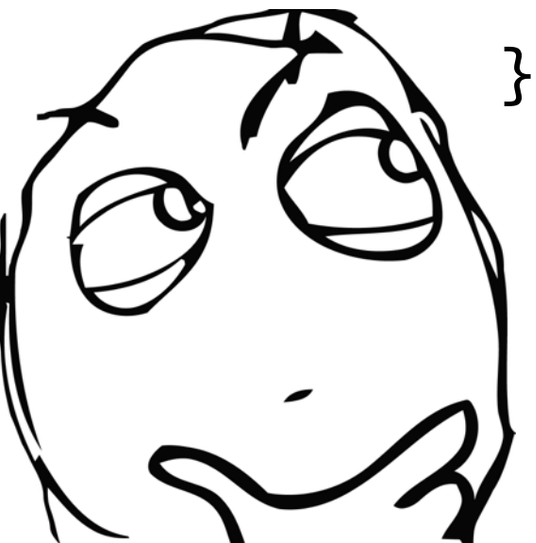
```
DoubleProperty one=…;
DoubleProperty two=…;

this.one.add(this.two).
   addListener((x, y, z) -> {
            if (z != null && z.intValue() == 10) {
              Alert alert = new Alert(…);
              alert.showAndWait();
            }
   });
```



Same as:

The listener get's garbage collected

```
DoubleProperty one=…;
DoubleProperty two=…;

NumberBinding add = Bindings.add(one, two);

add.addListener((x, y, z) -> {
   if (z != null && z.intValue() == 10) {
      Alert alert = new Alert(…);
      alert.showAndWait();
   }
});
```



DEMO

Conclusion - reference the binding

```
DoubleProperty one=…;
DoubleProperty two=…;

this.add = Bindings.add(one, two);
this.add.addListener((x, y, z) -> {
   if (z != null && z.intValue() == 10) {
      Alert alert = new Alert(…);
      alert.showAndWait();
   }
});
```

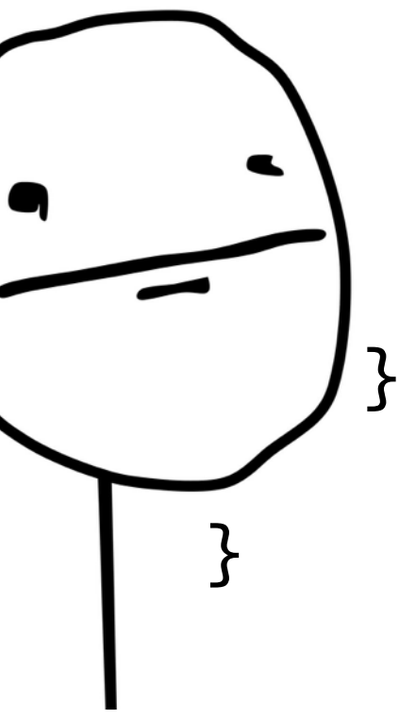
Threading and Services

Threading and Services

It’s hard to test JavaFX Services!

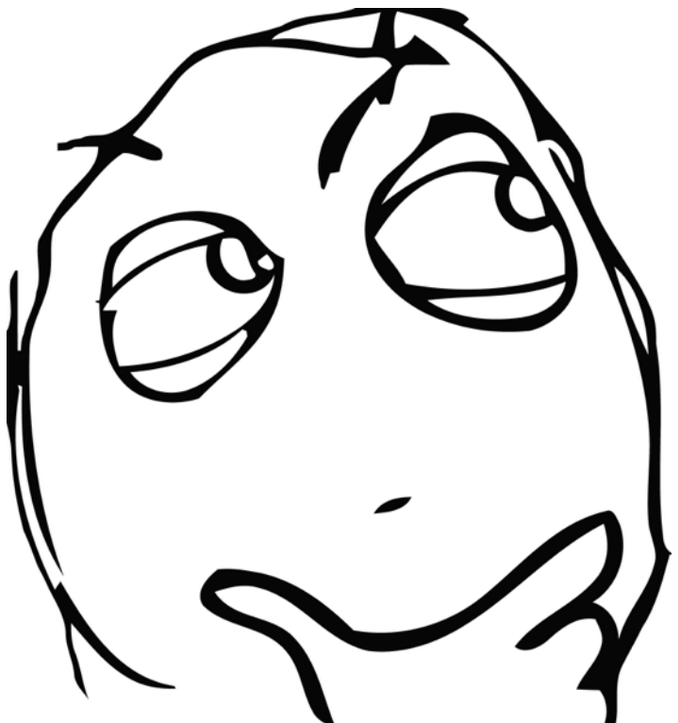
How to test this?

```
public class TestService extends Service<String> {

    @Override
    protected Task<String> createTask() {
        return new Task<String>() {
            @Override
            protected String call() throws Exception {
                Thread.sleep(5000);
                return "I'm an expensive result.";
            }
        };
    }
}
```



Not FX specific:

How to test async behaviour?



Green or Red?

```
@Test
public void testLongLastingOperation() throws … {
   TestService service = new TestService();

   CompletableFuture<String> future = new CompletableFuture<>();

   service.valueProperty().addListener((b, o, n) -> {
      if (n != null) {
         future.complete(n);
      }
   });

   service.restart();

  assertEquals("I'm an expensive result.", future.get(5, TimeUnit.SECONDS));
}
```

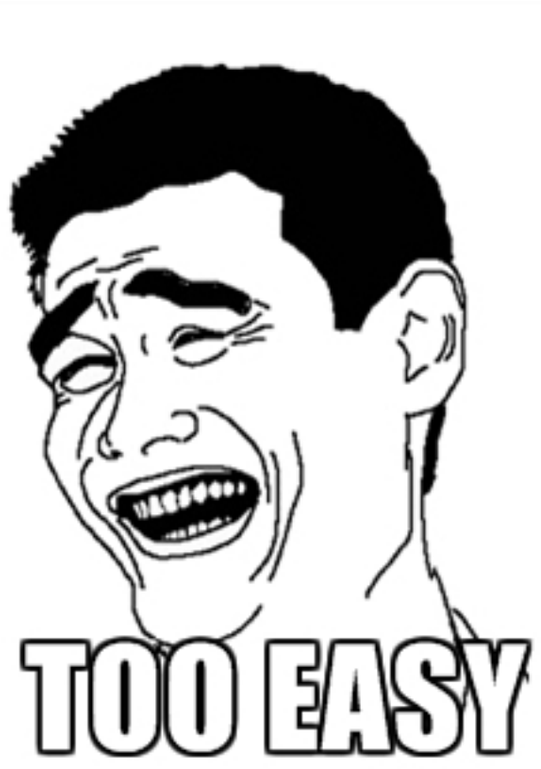
```
java.lang.IllegalStateException: Toolkit not initialized
    at com.sun.javafx.application.PlatformImpl.runLater(PlatformImpl.java
    at com.sun.javafx.application.PlatformImpl.runLater(PlatformImpl.java
    at javafx.application.Platform.runLater(Platform.java:83)
    at javafx.concurrent.Service.runLater(Service.java:913)
    at javafx.concurrent.Service.start(Service.java:676)
…
```

Green or Red?

```
@Test
public void testLongLastingOperation() throws … {
   TestService service = new TestService();

   CompletableFuture<String> future = new CompletableFuture<>();

   service.valueProperty().addListener((b, o, n) -> {
      if (n != null) {
         future.complete(n);
      }
   });

   service.restart();

  assertEquals("I'm an expensive result.", future.get(5, TimeUnit.SECONDS));
}
```

Needs started FX-Toolkit

Ok… let’s fix this



```
                                    <dependency>
                                        <groupId>de.saxsys</groupId>
                                        <artifactId>jfx-testrunner</artifactId>
                                        <version>1.2-SNAPSHOT</version>
                                    </dependency>
```

```
@RunWith(JfxRunner.class)
public class ServiceNotTestableTest {

 @Test
  public void testLongLastingOperation() throws … {
     TestService service = new TestService();

     CompletableFuture<String> future = new CompletableFuture<>();

     service.valueProperty().addListener((b, o, n) -> {
        if (n != null) {
           future.complete(n);
        }
     });

  service.restart();
   assertEquals("I'm an expensive result.", future.get(5, TimeUnit.SECONDS));
}
```

But...

The Service has the intention to be reused multiple times. Let's test this!

Let’s do a small refactoring

```
@RunWith(JfxRunner.class)
public class TestableServiceTest {

   @Test
   public void testLongLastingOperation() throws … {
      TestService service = new TestService();
      CompletableFuture<String> future = new CompletableFuture<>();

      service.valueProperty().addListener((b, o, n) -> {
         if (n != null) {
            future.complete(n);
         }
      });

      service.restart();
      assertEquals("I'm an expensive result.", future.get(5, TimeUnit.SECONDS));
   }
```

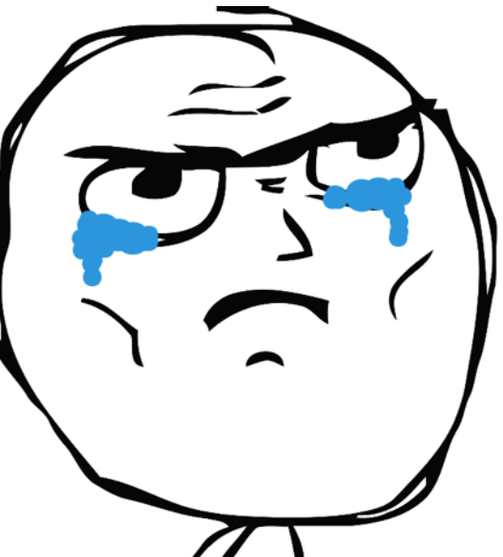
```
@RunWith(JfxRunner.class)
public class TestableServiceTest {

  @Test
  public void testLongLastingOperation() throws … {
   TestService service = new TestService();

   //refactored the Completable future stuff with the listeners in a method
   CompletableFuture<String> future = createFutureWithListener(service);

   service.restart();
   assertEquals("I'm an expensive result.", future.get(5, TimeUnit.SECONDS));
  }
}
```

Green or Red?

```
@RunWith(JfxRunner.class)
public class TestableServiceTest {

  @Test
  public void testLongLastingOperation() throws … {
   TestService service = new TestService();

   //refactored the Completable future stuff with the listeners in a method
   CompletableFuture<String> future = createFutureWithListener(service);

   service.restart();
   assertEquals("I'm an expensive result.", future.get(5, TimeUnit.SECONDS));

   CompletableFuture<String> future = createFutureWithListener(service);
   service.restart();
   assertEquals("I'm an expensive result.", future.get(5, TimeUnit.SECONDS));

  }
}
```

java.lang.IllegalStateException: Service must only be used from the FX Application Thread

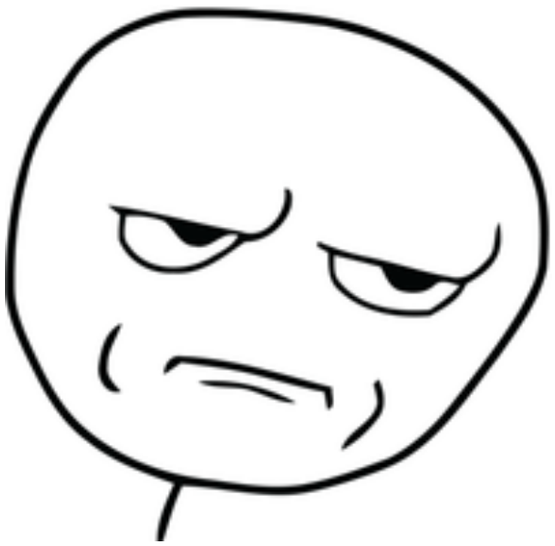
```
    at javafx.concurrent.Service.checkThread(Service.java:906)
    at javafx.concurrent.Service.valueProperty(Service.java:205)
    at
de.saxsys.pitfalls.services.ClassToTestTest.testLongLastingOperation(ClassToTestTest.java:
40)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:
```



```java
@RunWith(JfxRunner.class)
public class TestableServiceTest {

  @Test
  public void testLongLastingOperation() throws … {
   TestService service = new TestService();

   //refactored the Completable future stuff with the listeners in a method
   CompletableFuture<String> future = createFutureWithListener(service);

   service.restart();
   assertEquals("I'm an expensive result.", future.get(5, TimeUnit.SECONDS));

   CompletableFuture<String> future = createFutureWithListener(service);
   service.restart();
   assertEquals("I'm an expensive result.", future.get(5, TimeUnit.SECONDS));

  }
}
```

Let's have a look on the Service Implementation

```
@Override public final V getValue() { checkThread(); return value.get(); }
@Override public final ReadOnlyObjectProperty<V> valueProperty() { checkThread(); return value;}

/**
 * Cancels any currently running Task, if any, and restarts this Service. The state
 * will be reset to READY prior to execution. This method should only be called on
 * the FX application thread.
 */
public void restart() {
    checkThread();
    …
}
```



```
void checkThread() {
    if (startedOnce && !isFxApplicationThread()) {
    throw new IllegalStateException("Service must only be
    used from the FX Application Thread");
    }
}
```

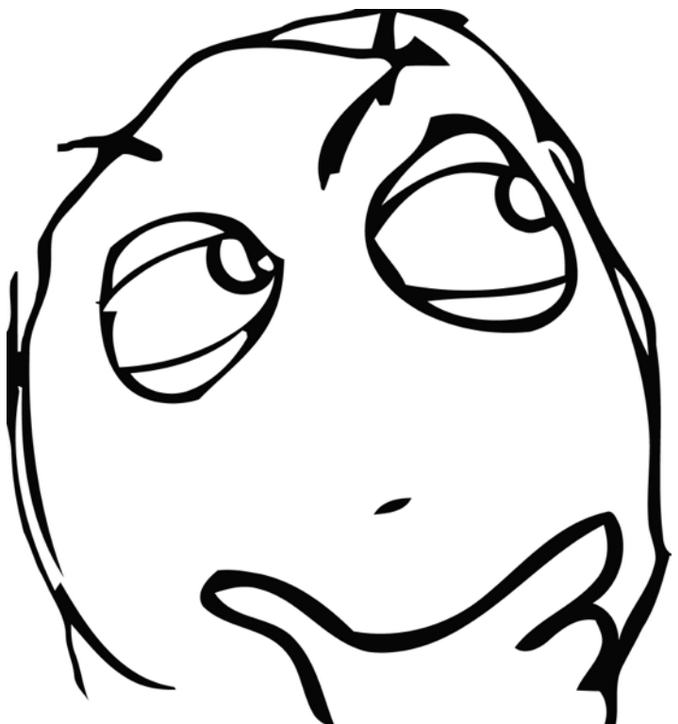
first restart()

second restart()

But I wan’t to call my service multiple times in my test - because this is it’s use-case.

CHALLENGE ACCEPTED

Let’s test this in the FX-Thread



```
@RunWith(JfxRunner.class)
public class TestableServiceTest {

  @Test
  public void testLongLastingOperation() throws … {
   TestService service = new TestService();

   //refactored the Completable future stuff with the listeners in a method
   CompletableFuture<String> future = createFutureWithListener(service);

   service.restart();
   assertEquals("I'm an expensive result.", future.get(5, TimeUnit.SECONDS));

   CompletableFuture<String> future = createFutureWithListener(service);
   service.restart();
   assertEquals("I'm an expensive result.", future.get(5, TimeUnit.SECONDS));

  }
}
```

```java
@RunWith(JfxRunner.class)
public class TestableServiceTest {

  @Test
  @TestInJfxThread ..............................
  public void testLongLastingOperation() throws … {
   TestService service = new TestService();

   //refactored the Completable future stuff with the listeners in a method
   CompletableFuture<String> future = createFutureWithListener(service);

   service.restart();
   assertEquals("I'm an expensive result.", future.get(5, TimeUnit.SECONDS));

   CompletableFuture<String> future = createFutureWithListener(service);
   service.restart();
   assertEquals("I'm an expensive result.", future.get(5, TimeUnit.SECONDS));

  }
}
```

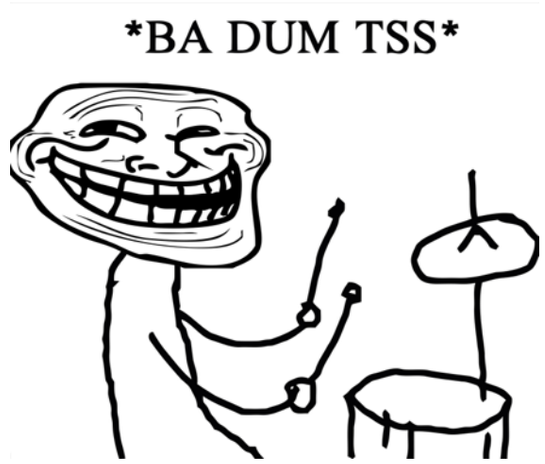
jfx-testrunner annotation

Green or Red?

```
@RunWith(JfxRunner.class)
public class TestableServiceTest {

  @Test
  @TestInJfxThread
  public void testLongLastingOperation() throws … {
   TestService service = new TestService();

   //refactored the Completable future stuff with the listeners in a method
   CompletableFuture<String> future = createFutureWithListener(service);

   service.restart();
   assertEquals("I'm an expensive result.", future.get(5, TimeUnit.SECONDS));

   CompletableFuture<String> future = createFutureWithListener(service);
   service.restart();
   assertEquals("I'm an expensive result.", future.get(5, TimeUnit.SECONDS));

  }
}
```

java.util.concurrent.TimeoutException

at java.util.concurrent.CompletableFuture.timedGet(CompletableFuture.java:1763)
at java.util.concurrent.CompletableFuture.get(CompletableFuture.java:1907)
at de.saxsys.pitfalls.services.nottestable.ServiceNotTestableTest.testLongLastingOperationInFXThread(
TestServiceTest.java:**85**)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:497)
at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:50)
at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:47)
at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:325)
at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:78)
at de.saxsys.javafx.test.JfxRunner.access$0(JfxRunner.java:1)
at de.saxsys.javafx.test.JfxRunner.lambda$0(JfxRunner.java:47)
at de.saxsys.javafx.test.JfxRunner$$Lambda$56/1865127310.run(Unknown Source)
at com.sun.javafx.application.PlatformImpl.lambda$null$170(PlatformImpl.java:295)
at com.sun.javafx.application.PlatformImpl$$Lambda$52/1724593206.run(Unknown Source)
at java.security.AccessController.doPrivileged(Native Method)
at com.sun.javafx.application.PlatformImpl.lambda$runLater$171(PlatformImpl.java:294)
at com.sun.javafx.application.PlatformImpl$$Lambda$51/580016451.run(Unknown Source)
at com.sun.glass.ui.InvokeLaterDispatcher$Future.run(InvokeLaterDispatcher.java:95)

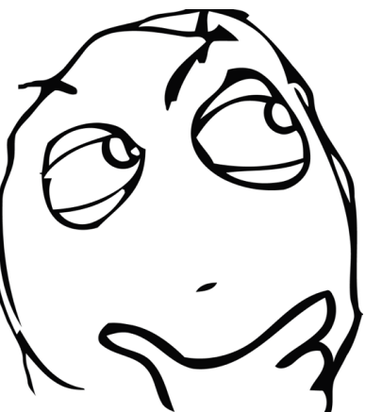


Line 85: .get()
blocks the UI Thread

```
@RunWith(JfxRunner.class)
public class TestableServiceTest {

  @Test
  @TestInJfxThread
  public void testLongLastingOperation() throws … {
   TestService service = new TestService();

   //refactored the Completable future stuff with the listeners in a method
   CompletableFuture<String> future = createFutureWithListener(service);

   service.restart();
   assertEquals("I'm an expensive result.", future.get(5, TimeUnit.SECONDS));

   CompletableFuture<String> future = createFutureWithListener(service);
   service.restart();
   assertEquals("I'm an expensive result.", future.get(5, TimeUnit.SECONDS));

  }
}
```

Ok, correct solution:
start with Test Thread and move all Service accesses to FX Thread

```java
@Test
public void testLongLastingOperationSimple() throws … {
    TestService service = new TestService();
    CompletableFuture<String> future = new CompletableFuture<>();
    Platform.runLater(() -> {
        service.valueProperty().addListener((b, o, n) -> {
            if (n != null) {
                future.complete(n);
            }
        });
        service.restart();
    });
    assertEquals("I'm an expensive result. 1", future.get(5, TimeUnit.SECONDS));
}
```

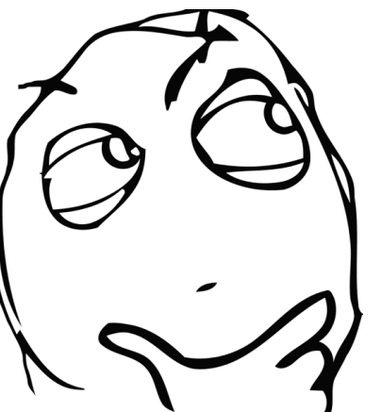


FX Thread

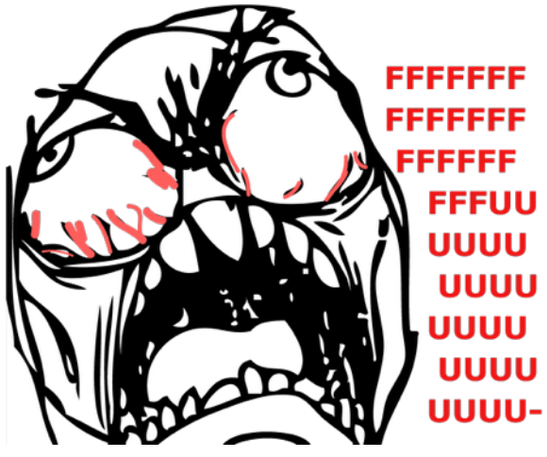
Ok, let’s run and test the Service multiple times

For each execution of the Service you have to double the amount of the (boilerplate)-code.

```
@Test
public void testLongLastingOperationSimple() throws … {
      TestService service = new TestService();
      CompletableFuture<String> future = new CompletableFuture<>();
      Platform.runLater(() -> {
            service.valueProperty().addListener((b, o, n) -> {
                  if (n != null) {
                        future.complete(n);
                  }
            });
            service.restart();
      });
      assertEquals("I'm an expensive result. 1", future.get(5, TimeUnit.SECONDS));

      CompletableFuture<String> future = new CompletableFuture<>();
      Platform.runLater(() -> {
            service.valueProperty().addListener((b, o, n) -> {
                  if (n != null) {
                        future.complete(n);
                  }
            });
            service.restart();
      });
      assertEquals("I'm an expensive result. 1", future.get(5, TimeUnit.SECONDS));

}
```



This is too much boilerplate code for testing Services



get help with `jfx-testrunner`

https://github.com/sialcasa/jfx-testrunner

reduces
platform issues

no more
checkThread()
problems for asserts

```
@RunWith(JfxRunner.class)
public class ServiceTestableTest {

  @Test
    public void startServiceMultipleTimesTest() throws …{
       TestService service = new TestService();
       ServiceWrapper wrapper = new ServiceWrapper(service);
       wrapper.startAndWait(6000);
       assertEquals("I'm an expensive result. 1", wrapper.getValue())
       wrapper.restartAndWait(6000);
       assertEquals("I'm an expensive result. 2", wrapper.getValue())
    }
}
```

blocks test thread

Conclusion

there are problems testing services

you need a lot of boilerplate code to test them

use libraries

Threading and Services

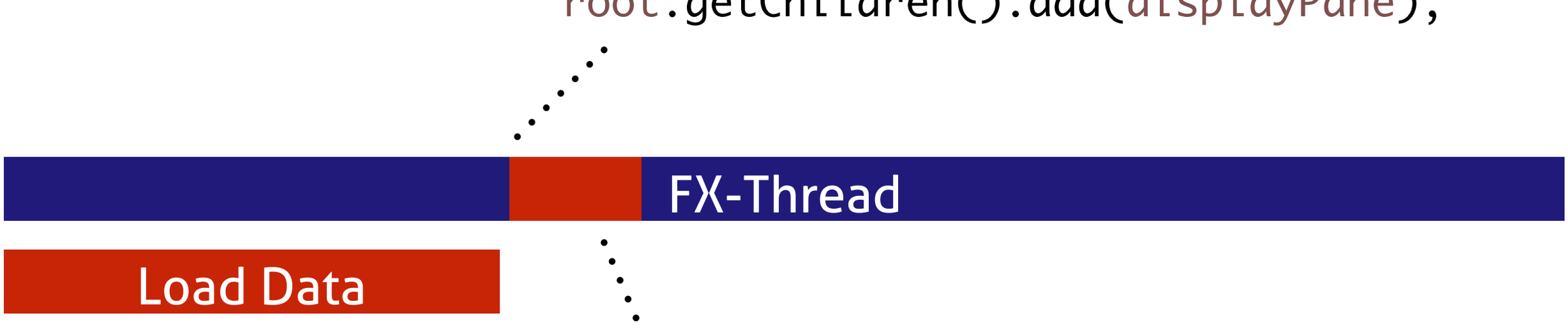
Don’t create a complex node graph in the UI Thread!

Attach **Nodes** to the scene graph on the UI Thread only,

but you should create **Nodes** in background threads!

Display Loaded Data

```
ArrayList<String> data = getBigData();
Pane displayPane = new Pane();
for (int i = 0; i < data.size(); i++) {
   Label label = new Label(data.get(i));
   displayPane.getChildren().add(label);
}
root.getChildren().add(displayPane);
```



UI-Freezes until the
Graph has been build

```
ArrayList<String> data = new ArrayList<>();
Pane displayPane = new Pane();
for (int i = 0; i < data.size(); i++) {
   Label label = new Label(data.get(i));
   displayPane.getChildren().add(label);
}
Platform.runLater(() -> root.getChildren().add(displayPane));
```

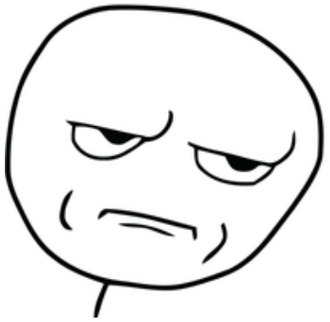
Pixel Performance

Pixel Performance

Preprocess Metadata of Images

How to preprocess the Metadata of Images?

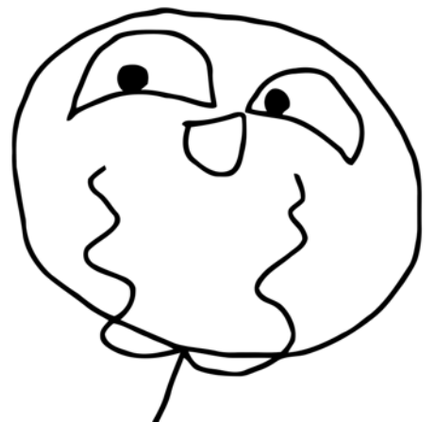
Loading the whole image to get the Metadata is bad.



```
Image image = new Image("/50mp.jpg", true);
double width = image.getWidth();
double height = image.getHeight();
```

Uses meta-data in header [1]

```
SimpleImageInfo info = new SimpleImageInfo(file);
double width  = info.getWidth();
double height = info.getHeight();
```



Pixel Performance

Read- / Write Pixels

Read / Write

`javafx.scene.image.`**`PixelWriter`**

writing the pixel data of a WritableImage

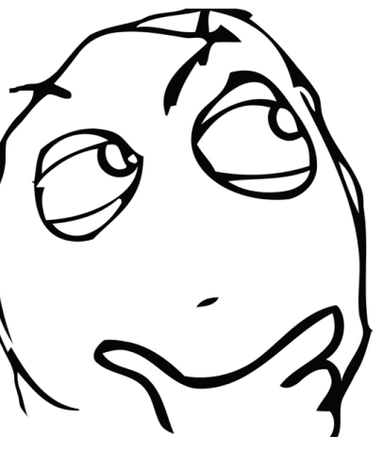
`javafx.scene.image.`**`PixelReader`**

retrieving the pixel data from an Image

Many methods, same result…

e.g. cropping

PixelReader	`getArgb(…)`	`getColor(…)`	`getPixels(…)`
PixelWriter	`setArgb(…)`	`setColor(…)`	`setPixels(…)`



DEMO

Pixel Performance

Choose the right pixel format!

What is the right pixel format for Quantum Toolkit?

```
myPixelReader.getPixels(…,pixelformat ,…);
```

BYTE_BGRA_PRE

```
myPixelReader.setPixels(…,pixelformat ,…);
```

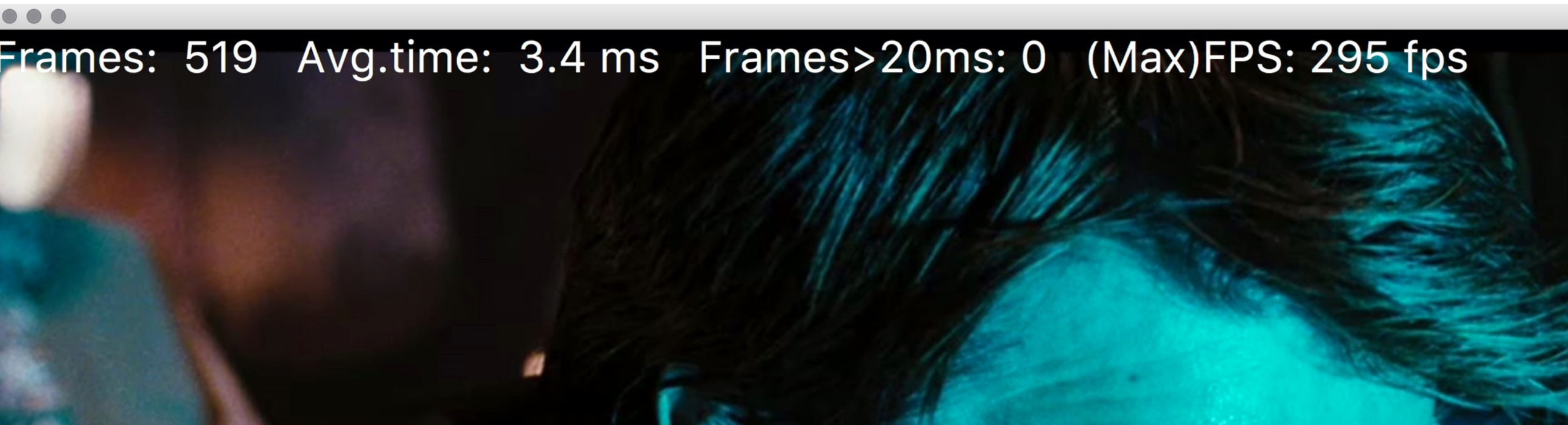
calls

```
@Override
public PlatformImage createPlatformImage(int w, int h) {
    ByteBuffer bytebuf = ByteBuffer.allocate(w * h * 4);
    return com.sun.prism.Image.fromByteBgraPreData(bytebuf, w, h);
}
```

Demo - Videoplayer using **VLC** & setPixels()

VLC solves 2 additional pitfalls

1. FX-Mediaplayer does not supports all codecs
2. there is no `javafx.scene.media.*` on ARM



Pixel Performance

Many complex nodes? Use Canvas!

Rule: use fewer nodes (>20.000 Nodes)!

picking, sync, compute dirty regions, apply CSS,....

Use Canvas / Image when you have a lot to show!

Demo

Canvas: + higher fps

+ lower heap
+ (~128MB - 250 pic.)
+ Node count 2

Node: - lower fps

- higher heap
-(~200MB - 250 pic.)
- Node count >250

Pixel Performance Conclusion

be aware of complex/many nodes

Node or Images? —> depends

choose correct PixelFormat & methods

Questions?

Hint: JavaFX Real-World Apps [CON3907]
Alexander Casall, Dirk Lemmermann
Wednesday, Oct 28, 1:00 p.m.

Links

1. http://max-wielsch.blogspot.de/2015/10/testing-asynchronous-javafx-logic.html
2. https://jaimonmathew.wordpress.com/2011/01/29/simpleimageinfo/
3. http://tomasmikula.github.io/blog/2015/02/10/the-trouble-with-weak-listeners.html
4. https://blog.codecentric.de/en/2015/04/tweaking-the-menu-bar-of-javafx-8-applications-on-os-x/

BACKUP

leaking Bindings

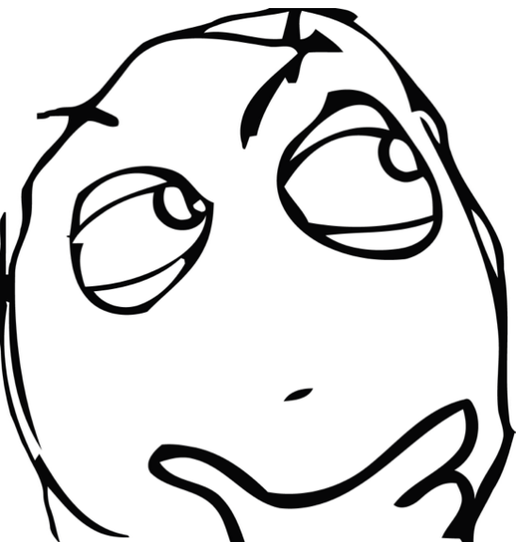
- JavaFX bindings uses WeakListeners to observe dependencies
- **dispose root binding should be enough**

leaking Bindings

- **Memory Leaks in Bindings**

SimpleDoubleProperty **prop** = new SimpleDoubleProperty(0.0);

```
for (int i = 0; i < 1000000; ++i) {
    prop.add(5).multiply(10).dispose();
}
prop.unbind(); // will it work?
```



NO!

count WeakListener = 1000000 !
count WeakReference = 1000000 !

leaking Bindings

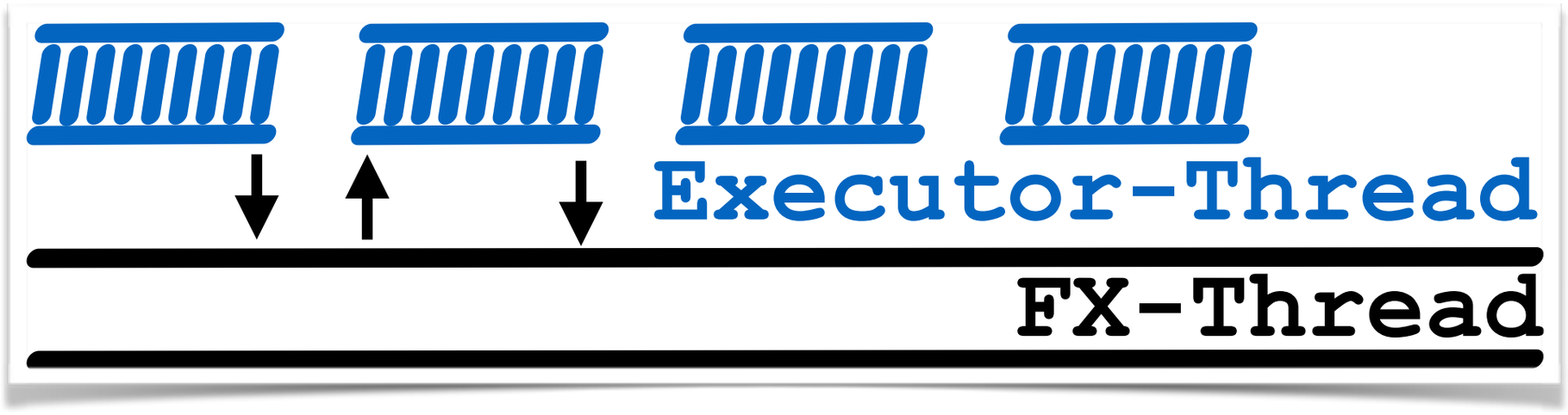
- **Memory Leaks in Bindings**

SimpleDoubleProperty **prop** = new SimpleDoubleProperty(0.0);

```
for (int i = 0; i < 1000000; ++i) {
    prop.add(5).multiply(10).dispose();
}
```

prop.add(10); // invalidates all ref. !!

chain your service execution

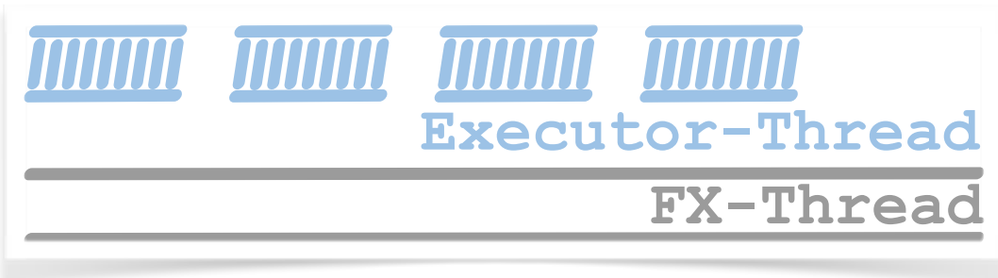


chain your Service execution

- Solution 1 - Task & Platform.runLater

```
new Task<String>() {
 protected String call() throws Exception {
    Result r = service.get();
    Platform.runLater(()-> .....);
    Result r2 = service2.get();
    Platform.runLater(()-> .....);
    return "finished - step 2";
  }
};
```

- **Platform.runLater - keeps order**
- **bad error handling & more !!**



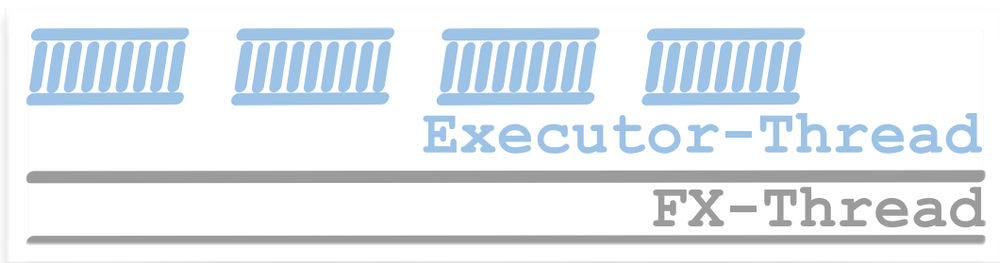
chain your Service execution

- Solution 2 - event handler

```
A.addEventHandler(SUCCEEDED, (s) -> {
   updateUI();
   // start next service
   B.start();
});

A.addEventHandler(FAILED, (e) -> {
   doThat();
   updateUI();
});
```

```
B.setOnSucceeded((state) -> {
updateUI();
});

B.setOnFailed((event) -> {
doThat();
updateUI();
});
```

- **better error handling**
- **lots of code!**

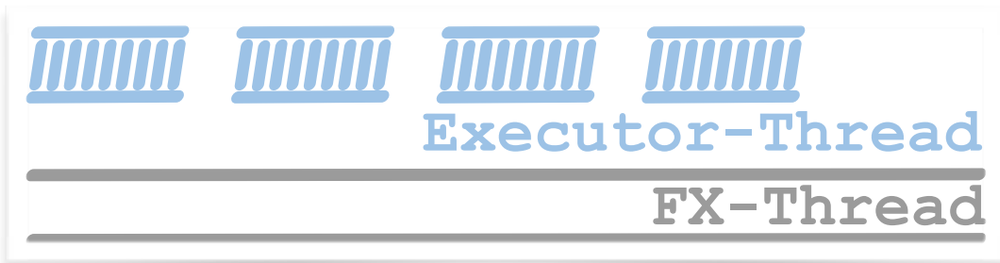


chain your Service execution

- Solution 3 - fluent API


```
handler
.supplyAsync(()->longRunningTask1())
.onError(this::updateErrorMessage)
.consumeOnFXThread(stringVal ->
        vbox.getChildren().
        add(new Button(stringVal))
)
.supplyAsync(()->longRunningTask2())
.onError(this::updateErrorMessage)
.execute(this::updateUI); //non-blocking!
```

- **good error handling**
- **easy to read**



Test service with fluent API

```
final TestService service = new TestService();

IntStream.range(0, 5).forEach(v -> {

            handler.supplyOnFXThread(() -> {
                registerListenerAndStart(service, waitLatch);
                return service;
            })

    });
```

main thread **FX thread**

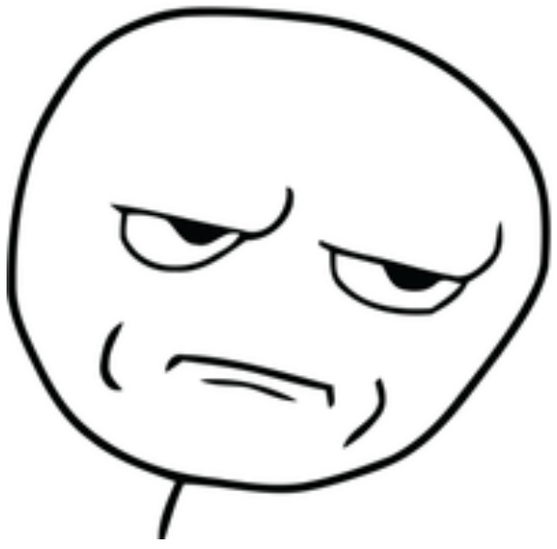
```
final TestService service = new TestService();

IntStream.range(0, 5).forEach(v -> {

            handler.supplyOnFXThread(() -> {
                registerListenerAndStart(service, waitLatch);
                return service;
            }).functionOnExecutorThread((service) -> {
                // wait outside FX application thread !!!
                awaitService(waitLatch);
                return service;
             }).execute(serv -> {
                assertEquals("I'm an expensive result.", serv.getValue());
                stop.countDown();
             }); // Non-blocking !!!

             awaitServiceResult(stop);
            });
```

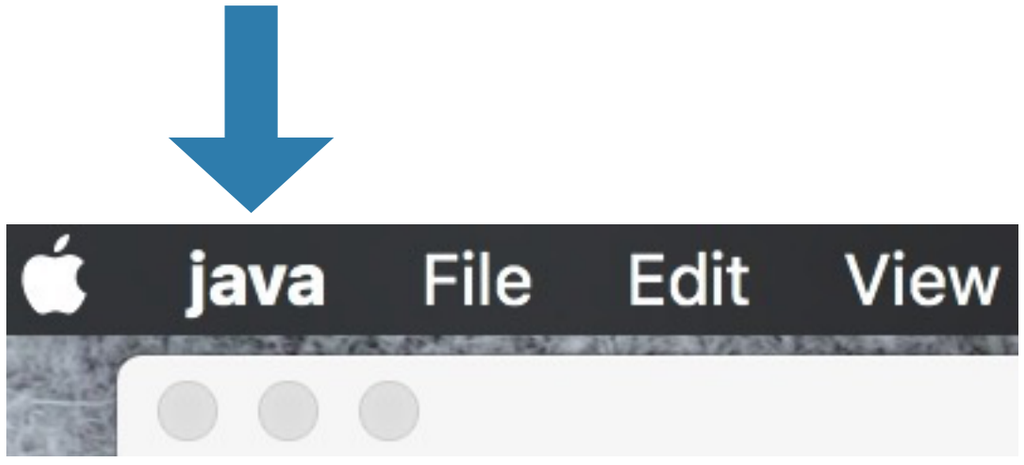
main thread **FX thread** **worker thread**

OS X Menu Bar

OS X Menu Bar

- JavaFX is platform independent
- **no direct access to OS X application menu**

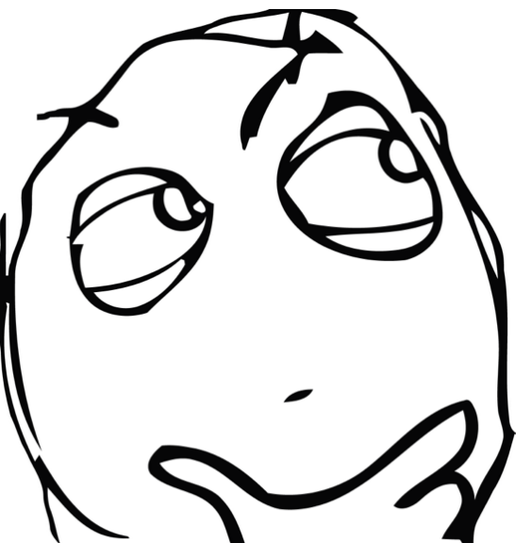


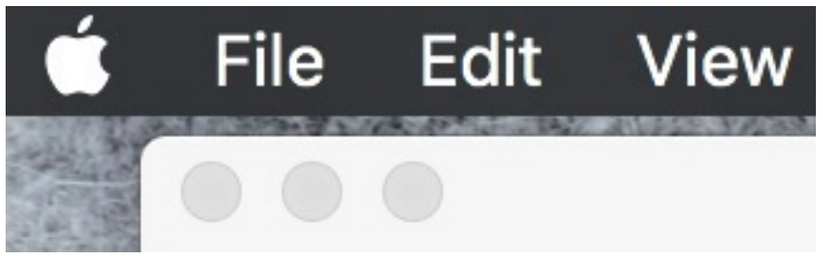


OS X Menu Bar

- Solution: **Eclipse SWT -> Cocoa native binding**

  JavaFX wrapper: NSMenuFX [3]





better