کارگاه برنامه‌نویسی پیشرفته

دستور کار شماره دوازده

اهداف

روش‌های برنامه نویسی موازی در JavaFX
آشنایی با Timeline و TimerTask
مدیریت چند Scene در یک برنامه

فهرست مطالب

توضیح Platform.runLater

از جلسات قبل به یاد دارید که برای برنامه‌نویسی موازی، می‌توانستیم با پیاده‌سازی اینترفیس Runnable و دادن آن به Thread یا Executor Service، یک عمل را موازی با عملیات دیگر انجام دهیم. اما این نکته را توجه کنید که در برنامه‌های JavaFX یک Thread اصلی به اسم FXThread داریم که فقط روی آن می‌توان در root برنامه و Scene تغییرات ایجاد کرد.

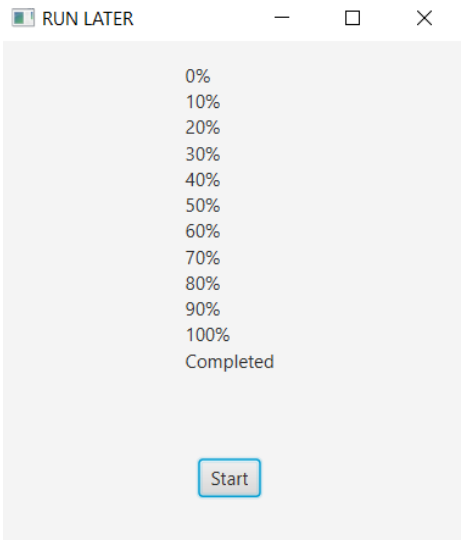
حال، راه‌حل این مشکل توسط خود کتابخانه JavaFX ارائه شده است که Platform.runLater است.

به مثال زیر توجه کنید:

```
import javafx.application.Application;
import javafx.application.Platform;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.stage.Stage;

import java.io.IoException;

class Controller {
    private int progress = 0;
    @FXML
    private Label label;

    @FXML
    public void execute(ActionEvent e) {
        label.setText("0%\n");
        Thread taskThread = new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 0; i < 10; i++) {

                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }

                    progress;

                    Platform.runLater(new Runnable() {
                        @Override
                        public void run() {
                            String text = label.getText() + (progress * 10) + "%" + '\n';
                            label.setText(text);
                            if (progress == 10) {
                                text = label.getText() + "Completed" + '\n';
                                label.setText(text);
                            }
```

```
                        }

                        I
                    });

                }
            }
        });
        taskThread.start();
    }
}

public class Main extends Application {

    @Override
    public void start(Stage primaryStage) {
        try {
            FXMLLoader loader = new FXMLLoader(getClass().getResource("sample.fxml"));
            loader.setController(new Controller());
            Parent root = loader.load();
            primaryStage.setTitle("RUN LATER");
            primaryStage.setScene(new Scene(root));
            primaryStage.show();
        } catch (I0Exception e) {
            e.printStackTrace();
        }
    }

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

خروجی این کد بعد از ۱۰ ثانیه فشردن دکمه start:



آشنایی با TimerTask و انواع آن

شاید شما با واژه فریم بر ثانیه[1] در بازی‌های رایانه‌ای آشنا باشید. آیا ما نیز می‌توانیم با قابلیت‌های جاوا، برنامه‌ای طراحی کنیم که چندین فریم را پشت سر هم اجرا کند؟

جواب مثبت است؛ با دو راه‌حل کاربردی آن آشنا می‌شویم:

راه‌حل اول کلاس‌های Timer و TimerTask است که در پکیج java.util قرار دارند.

Timer

کلاسی است که سازنده[2] بدون آرگومان دارد و شامل متدهای مختلف است که با دوتای آن کار داریم.

برای مثال ما یک شئ از این کلاس می‌سازیم و آن را timer می‌نامیم.

متد اولی که با آن کار داریم متد schedule است. این متد چند overload دارد که پرکاربردترین آنها به عنوان آرگومان اول یک TimerTask و برای آرگومان دوم و سوم long می‌گیرد.

آرگومان دوم مقدار تأخیر[3] بر اساس میلی ثانیه است که تأخیر اجرای TimerTask بعد از رسیدن برنامه به آن خط متد را مشخص می‌کند.

آرگومان سوم period است که بر اساس میلی ثانیه، میزان تأخیر بین هر دوبار اجرای TimerTask را مشخص می‌کند.

[1] frame per second (FPS)
[2] constructor
[3] delay

TimerTask

حال TimerTask را بررسی می‌کنیم. TimerTask یک کلاس انتزاعی است که متد run() آن پیاده‌سازی نشده. برای استفاده از این کلاس دو راه داریم (مثال‌ها خارج از چارچوب JavaFX هستند):

۱) ساخت یک کلاس و ارث‌بری از آن و اُورراید کردن متد run():

```
public class MyTask extends TimerTask {

   private int seconds;

   public MyTask() {
       seconds = 0;
   }

   @Override
   public void run() {
       System.out.println(seconds + "//");
       seconds++;
   }
}
```

۲) پیاده‌سازی به عنوان کلاس ناشناس[1]:

```
public class Main {
   public static void main(String[] args) {
       TimerTask timerTask = new TimerTask() {
           int seconds = 0;

           @Override
           public void run() {
               System.out.println(seconds + "//");
               seconds++;
           }
       };
   }
}
```

نکته: signature کلاس TimerTask به صورت زیر است:

```
public class TimerTask implements Runnable
```

پس هر شئ از آن با یک Thread جدا قابل اجر است و نیاز به توقف دارد، پس متد cancel را داریم که هم برای TimerTask و هم برای Timer نیاز است که بعد از پایان اجرای عملیات آن را صدا بزنیم.

[1] anonymous class

اگر این راه‌حل را در برنامه javafx پیاده‌سازی کنیم و در آن بخواهیم تغییراتی روی root و Scene انجام دهیم به IllegalStateException برخورد می‌کنیم چون Threadای که برای Timer و TimerTask می‌سازیم جدا از FXThread است. راه‌حل این مشکل توسط خود کتابخانه javafx ارائه شده که TimeLine است (البته برای حل این مشکل همچنین می‌توان تغییراتی که می‌خواهیم روی root و Scene انجام دهیم را درون Platform.runLater انجام دهیم).

TimeLine یک کلاس در پکیج javafx.animation است که به ما این اجازه را می‌دهد تا حدودی مانند Timer و TimerTask عمل کنیم. و دیگر نیازی به Platform.runLater نباشد.

نحوه استفاده از TimeLine:

۱) آرگومان سازنده آن یک KeyFrame است. این KeyFrame در سازنده خود مدت زمان تأخیر بین هر دو اجرا و یک actionای که اجرا می‌کند را می‌گیرد (ما اینجا برای پیاده‌سازی آن از lambda استفاده می‌کنیم).

۲) بعد از پیاده‌سازی، تعداد مرتبه‌ای که می‌خواهیم action ما اجرا شود را به وسیله متد setCycleCount() به آن پاس می‌دهیم (برای مقدار بی‌نهایت از Animation.INDEFINITE استفاده کنید).

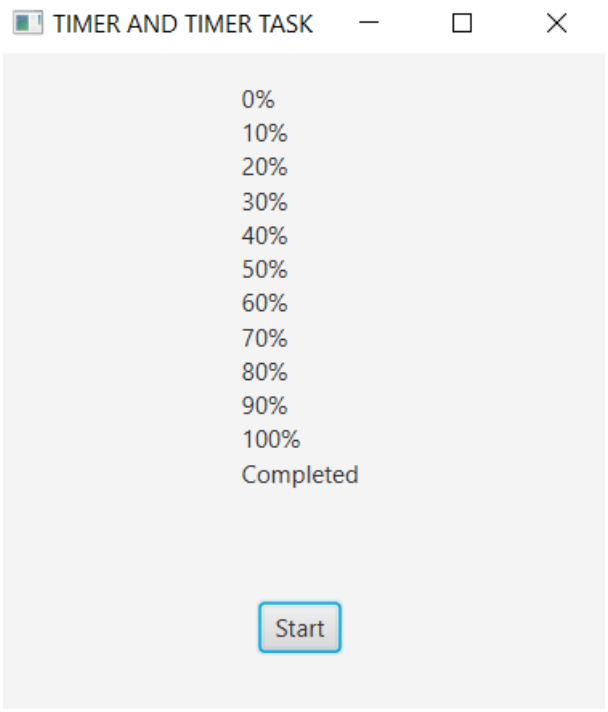
۳) سپس برای شروع اجرا، متد play() را روی آن صدا می‌زنیم.

به مثال زیر توجه کنید:

```
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.stage.Stage;

import java.io.I0Exception;

import javafx.util.Duration;

class Controller {
   private int progress = 0;
   @FXML
   private Label label;

   @FXML
   public void start(ActionEvent e) {
       label.setText("0%\n");
       Timeline timeline = new Timeline(new KeyFrame(Duration.millis(1000),
               actionEvent -> {
                   progress++;
                   String text = label.getText() + (progress * 10) + '%' + '\n';
```

```
                    label.setText(text);
                    if (progress == 10) {
                        text = label.getText() + "Completed" + '\n';
                        label.setText(text);
                    }
                }));
        timeline.setCycleCount(10);
        timeline.play();
    }
}

public class Main extends Application {

    @Override
    public void start(Stage primaryStage) {
        try {
            FXMLLoader loader = new FXMLLoader(getClass().getResource("sample.fxml"));
            loader.setController(new Controller());
            Parent root = loader.load();
            primaryStage.setTitle("TIMER AND TIMER TASK");
            primaryStage.setScene(new Scene(root));
            primaryStage.show();
        } catch (I0Exception e) {
            e.printStackTrace();
        }
    }

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

خروجی این کد بعد از ۱۰ ثانیه از زدن دکمه Start:



نکته: درصورتی که تعداد دفعات اجرای KeyFrame را بی‌نهایت تعریف کنیم، برای متوقف کردن باید متد stop() را روی timeLine صدا بزنیم.

نحوه آدرس‌دهی

برای آدرس‌دهی معمولاً دو راه داریم: آدرس‌دهی نسبی[1] و مطلق[2]. برای مثال فرض کنید یک imageView داریم که می‌خواهیم برای image آن آدرس‌دهی انجام دهیم:

Absolute Path

```
try {
   Image image = new Image(new FileInputStream("C:\\Users\\CE-AUT\\test7\\download.png"));
   imageView.setImage(image);
} catch(FileNotFoundException e) {
   e.printStackTrace();
}
```

توضیح: همانطور که می‌دانید به دلیل کار با کلاس FileInputStream، عبارت موجود در try-catch گذاشته شده است.

در آدرس‌دهی مطلق، مبدأ از پارتیشن‌های اصلی شروع می‌شود تا به فایل مورد نظر برسد. از آنجایی که پارتیشن‌بندی در مک، ویندوز و لینوکس و همچنین در هر دو pc نیز متفاوت است، پیشنهاد می‌شود برای آدرس‌دهی به assetsهای پروژه خود از جمله تصویرهای آن، از این روش استفاده نکنید.

Relative path

```
try {
   Image image = new Image(new FileInputStream("download.png"));
   imageView.setImage(image);
} catch (FileNotFoundException e) {
   e.printStackTrace();
}
```

در آدرس‌دهی نسبی، مبدأ معمولاً همان پوشه‌ای است که پروژه در آن قرار دارد.

برای مثال اگر این عکس در پوشه‌ای به نام assets و در پوشه پروژه قرار داشت، آدرس‌دهی به صورت زیر می‌شد:

```
Image image = new Image(new FileInputStream("assets/download.png"));
```

نکته: از آنجایی که فولدربندی هر دستگاه متفاوت است، سعی کنید از آدرس‌دهی نسبی در پروژه‌های خود استفاده کنید.

[1] relative path
[2] absolute

مراحل اجرا کردن یک فایل صوتی در JavaFX

ابتدا پروژه خود را با maven راه‌اندازی کرده و فایل module-info.java را باز می‌کنیم و اگر javafx.media در آن وجود نداشت به آن اضافه می‌کنیم:

```
module com.example.test7 {
        requires javafx.controls;
        requires javafx.fxml;
        requires javafx.web;
        requires javafx.media;
}
```

سپس به کنترلر خود برگشته و هر کجا که می‌خواهیم از Audio استفاده کنیم، کدی مانند زیر می‌نویسیم:

```
String path = "assets/audio.mp3";
Media media = new Media(Paths.get(path).toUri().toString());
MediaPlayer mediaPlayer = new MediaPlayer(media);
mediaPlayer.play();
```

ساز و کار آن واضح است، ابتدا آدرس‌دهی Media، سپس ایجاد یک MediaPlayer با آن و درآخر اجرای آن media.

همچنین کلاس MediaPlayer بسیاری متدهای دیگر نیز دارد که درصورت علاقه می‌توانید آنها را مطالعه کنید.

علاوه بر فایل محلی[1]، می‌توان برای آدرس‌دهی به media، آدرس یک فایل صوتی موجود در اینترنت را نیز استفاده کرد، برای مثال به تکه کد زیر توجه کنید:

```
String path = "https://www.computerhope.com/jargon/m/example.mp3";
Media media = new Media(path);
MediaPlayer mediaPlayer = new MediaPlayer(media);
mediaPlayer.play();
```

آدرس یک فایل صوتی موجود در اینترنت برای تست:

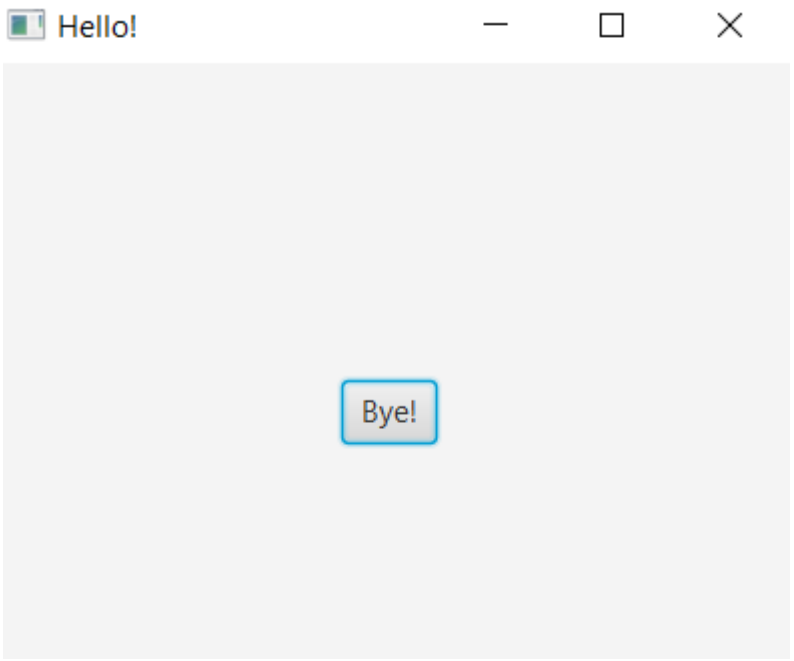
```
"https://www.computerhope.com/jargon/m/example.mp3"
```

[1] local file

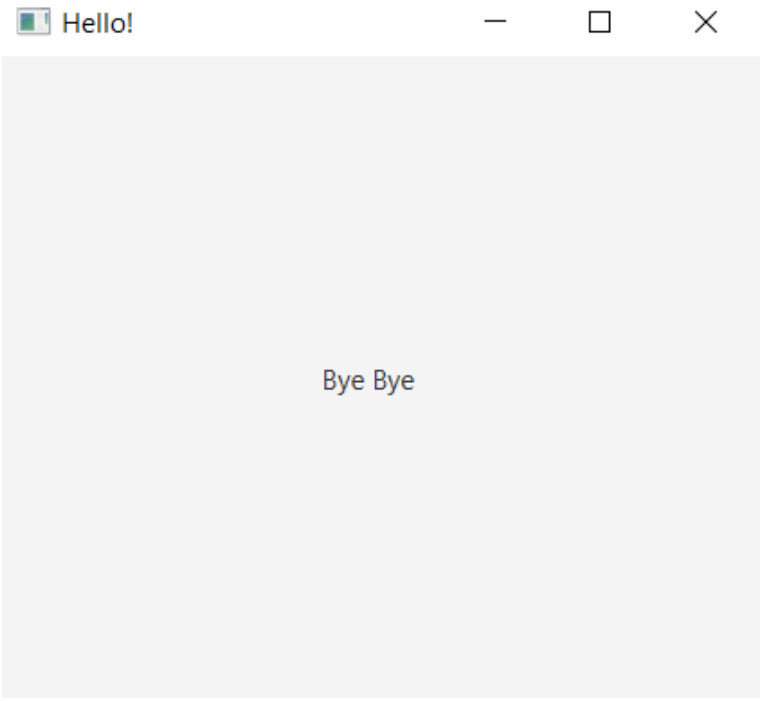
مدیریت چند scene و stage

در پروژه‌های بزرگتر نیاز خواهیم داشت که بتوان چند scene و stage مختلف داشته باشیم و بین آنها جابه‌جایی انجام دهیم.
در یک مثال ساده فرض کنید برنامه‌ای داریم که دارای دو فایل fxml به نام‌های Hello و GoodBye است و می‌خواهیم در ابتدا استیج Hello نمایش داده شود که دارای یک دکمه به اسم bye است. با فشردن دکمه bye توسط کاربر، استیج GoodBye باز شود.

پس ابتدا برنامه ما به این شکل باز می‌شود:



حال با فشردن دکمه bye صفحه‌ای مانند شکل زیر نشان داده می‌شود:



که متن یک label ساده را نشان می‌دهد. روند ساخت این کار را بررسی می‌کنیم.

ابتدا با FXMLLoader فایل fxmlای را که می‌خواهیم نمایش دهیم مانند زیر آدرس‌دهی می‌کنیم:

```
FXMLLoader loader = new FXMLLoader(getClass().getResource("GoodBye.fxml"));
```

سپس مانند تصویر زیر scene را مقداردهی می‌کنیم، stage را ایجاد کرده و scene آن را تنظیم می‌کنیم:

```
Parent root = loader.load();
Stage stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
Scene scene = new Scene(root);
stage.setScene(scene);
```

پس از آن، stage را نمایش می‌دهیم:

```
stage.show();
```

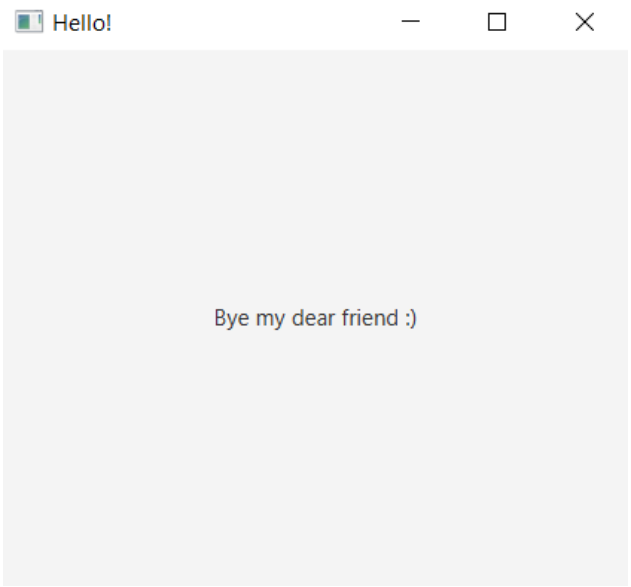
توجه: به دلیل استفاده از FXMLLoader، تمام این فرآیند را در try-catch انجام دهید. دقت کنید که در این حالت پنجره[1] ثابت است و به همین دلیل عنوان آن را نیز تغییر نداده‌ایم. در اینجا فقط Scene فعال آن را عوض کرده‌ایم.

[1] stage

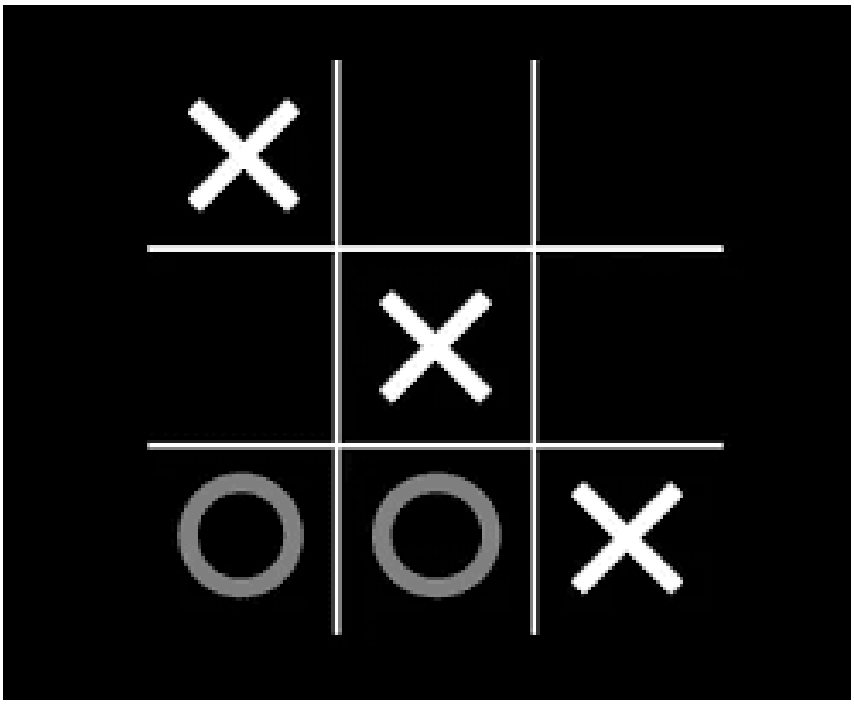
حال فرض کنید هنگام فشردن دکمه bye توسط کاربر و قبل نشان دادن stage بخواهیم متن label را عوض کنیم، برای این کار پس از لود کردن GoodBye.fxml، کنترلر آن را دریافت کرده، سپس روی آن عملیات انجام می‌دهیم:

```
FXMLLoader loader = new FXMLLoader(getClass().getResource("GoodBye.fxml"));
Parent root = loader.load();
GoodByeController goodBye = loader.getController();
goodBye.setLabel("Bye my dear friend :)");
```

پس از آن، خروجی ما بعد فشردن دکمه bye به صورت زیر خواهد بود:



انجام دهید: Tic-Tac-Toe



می‌خواهیم بازی دوز (XO) را با javafx پیاده‌سازی نماییم. قوانین این بازی را می‌توانید از این لینک مشاهده نمایید.

نکات پیاده‌سازی

۱. بازی باید با ربات باشد و ربات به صورت رندوم بازی می‌کند.
۲. در ابتدا باید منوی بازی و سپس صفحه گرفتن نام کاربر نمایش داده شود.
۳. در هنگام بازی، موسیقی پس زمینه باید پخش شود. همچنین این موزیک را از داخل بازی باید بتوان قطع و دوباره ادامه داد.
٤. بازی یک تایمر یک دقیقه ای دارد (در صفحه بازی باید زمان باقی‌مانده نمایش داده شود) و در صورت به پایان رسیدن بازی و مشخص نشدن برنده، بازی مساوی اعلام شود.

روش تحویل

branchها:

در این سؤال تشخیص تعداد branchها بر عهده خودتان است.

pull requestها:

۱. بخش view (فایل‌های fxml) و قابلیت جابه‌جا شدن بین صفحه‌ها
۲. منطق بازی (رعایت نوبت‌بندی صحیح در انتخاب خانه‌های جدول بازی)
۳. منطق بازی (انتخاب رندوم بات بازی)
٤. شرط پایان بازی
٥. نمایش تایمر بازی (در صورت تمام شدن تایمر بازی مساوی اعلام شود)
٦. پخش موسیقی پس‌زمینه

منوی پیشنهادی