1 import java.util.Calendar;  
 2 import java.util.GregorianCalendar;  
 3 import java.util.Random;  
 4 import javafx.util.Duration;  
 5   
 6 import javafx.application.Application;  
 7 import javafx.geometry.Pos;  
 8 import javafx.geometry.Insets;  
 9 import javafx.scene.Scene;  
 10 import javafx.scene.control.Label;  
 11 import javafx.scene.control.Button;  
 12 import javafx.scene.control.CheckBox;  
 13 import javafx.scene.control.RadioButton;  
 14 import javafx.scene.control.ToggleGroup;  
 15 import javafx.scene.control.Slider;  
 16 import javafx.scene.image.ImageView;  
 17 import javafx.scene.layout.BorderPane;  
 18 import javafx.scene.layout.GridPane;  
 19 import javafx.scene.layout.Pane;  
 20 import javafx.scene.layout.StackPane;  
 21 import javafx.scene.layout.HBox;  
 22 import javafx.scene.layout.VBox;  
 23 import javafx.scene.layout.Region;  
 24 import javafx.event.ActionEvent;  
 25 import javafx.event.EventHandler;  
 26 import javafx.scene.text.Font;  
 27 import javafx.scene.text.FontPosture;  
 28 import javafx.scene.text.FontWeight;  
 29 import javafx.scene.paint.Color;  
 30 import javafx.scene.shape.Arc;  
 31 import javafx.scene.shape.ArcType;  
 32 import javafx.scene.shape.Rectangle;  
 33 import javafx.scene.shape.Circle;  
 34 import javafx.scene.shape.Line;  
 35 import javafx.scene.media.Media;  
 36 import javafx.scene.media.MediaPlayer;  
 37 import javafx.scene.media.MediaView;  
 38 import javafx.scene.text.Text;  
 39 import javafx.animation.KeyFrame;  
 40 import javafx.animation.Timeline;  
 41 import javafx.beans.property.DoubleProperty;  
 42 import javafx.scene.layout.ColumnConstraints;  
 43 import javafx.scene.layout.RowConstraints;  
 44 import javafx.scene.layout.Priority;  
 45 import javafx.beans.binding.\*;  
 46 import javafx.stage.Stage;  
 47   
 48 public class n01185608 extends Application  
 49 {   
 50 public static void main(String[] args)  
 51 {  
 52 Application.launch(args);  
 53 }  
 54 @Override  
 55 public void start(Stage primaryStage)  
 56 {   
 57 GridPane gridPaneMain = new GridPane();//Main Grid Pane  
 58   
 59 gridPaneMain.setMaxSize(1000,500);//Pane Size  
 60 gridPaneMain.setMinSize(1000,500);  
 61   
 62 StackPane root = new StackPane(gridPaneMain);//Main Stack Pane  
 63 NumberBinding maxScale = Bindings.min(root.widthProperty().divide(1000),  
 64 root.heightProperty().divide(500));  
 65   
 66 gridPaneMain.scaleXProperty().bind(maxScale);  
 67 gridPaneMain.scaleYProperty().bind(maxScale);  
 68   
 69 //Put the BorderPane inside a grid in 0,0  
 70 MediaDemo obj = new MediaDemo();  
 71 gridPaneMain.add(obj.getPane(), 0,0);  
 72   
 73 RowConstraints row = new RowConstraints();  
 74 row.setVgrow(Priority.ALWAYS);  
 75 gridPaneMain.getRowConstraints().add(row);  
 76   
 77 ColumnConstraints col = new ColumnConstraints();  
 78 col.setHgrow(Priority.ALWAYS);  
 79 col.setPercentWidth(100.0);  
 80 gridPaneMain.getColumnConstraints().add(col);  
 81   
 82 //BouncingRectangle c = new BouncingRectangle();  
 83 Scene scene = new Scene(root, 1000, 500);  
 84 primaryStage.setTitle("Assignment 7"); // Set the stage title  
 85 primaryStage.setScene(scene); // Place the scene in the stage  
 86 primaryStage.show(); // Display the stage  
 87 }  
 88 Text text = new Text(50, 50, "Shyam Rajendren");   
 89 GridPane paneForCenterText = new GridPane();  
 90 protected BorderPane getPane()  
 91 {  
 92 HBox paneForButtons = new HBox(20);  
 93 Button btUp = new Button("Up");  
 94 Button btDown = new Button("Down");   
 95 paneForButtons.getChildren().addAll(btUp, btDown);  
 96 paneForButtons.setAlignment(Pos.CENTER);  
 97 paneForButtons.setStyle("-fx-border-color: green");  
 98   
 99 BorderPane pane = new BorderPane();  
100 pane.setBottom(paneForButtons);  
101   
102 Pane paneForText = new Pane();  
103 paneForText.getChildren().add(text);  
104   
105 //GridPane paneForCenterText = new GridPane();  
106 //paneForText.setPrefWidth(200);  
107 //paneForText.setPrefHeight(200);  
108 //paneForCenterText.setAlignment(Pos.CENTER);  
109 paneForCenterText.add(paneForText,0,0);  
110   
111 RowConstraints row = new RowConstraints();  
112 row.setVgrow(Priority.ALWAYS);  
113 paneForCenterText.getRowConstraints().add(row);  
114   
115 ColumnConstraints col = new ColumnConstraints();  
116 col.setHgrow(Priority.ALWAYS);  
117 paneForCenterText.getColumnConstraints().add(col);  
118 pane.setCenter(paneForCenterText);  
119   
120 btUp.setOnAction(e -> text.setY(text.getY() - 10));  
121 btDown.setOnAction(e -> text.setY(text.getY() + 10));  
122   
123 return pane;  
124 }  
125 }  
126   
127 class Checkbock extends n01185608  
128 {  
129 /\*public static void main(String[] args)  
130 {  
131 launch(args);  
132 }\*/  
133 @Override // Override the getPane() method in the super class  
134 protected BorderPane getPane() {  
135 BorderPane pane = super.getPane();  
136   
137 Font fontBoldItalic = Font.font("Calibri",FontWeight.BOLD, FontPosture.ITALIC, 20);  
138 Font fontBold = Font.font("Calibri",FontWeight.BOLD, FontPosture.REGULAR, 20);  
139 Font fontItalic = Font.font("Calibri",FontWeight.NORMAL, FontPosture.ITALIC, 20);  
140 Font fontNormal = Font.font("Calibri",FontWeight.NORMAL, FontPosture.REGULAR, 20);  
141   
142 text.setFont(fontNormal);  
143   
144 VBox paneForCheckBoxes = new VBox(20);  
145 paneForCheckBoxes.setPadding(new Insets(5, 5, 5, 5));   
146 paneForCheckBoxes.setStyle("-fx-border-color: green");  
147 CheckBox chkBold = new CheckBox("Bold");  
148 CheckBox chkItalic = new CheckBox("Italic");  
149 paneForCheckBoxes.getChildren().addAll(chkBold, chkItalic);  
150 pane.setLeft(paneForCheckBoxes);  
151   
152 EventHandler<ActionEvent> handler = e -> {   
153 if (chkBold.isSelected() && chkItalic.isSelected()) {  
154 text.setFont(fontBoldItalic); // Both check boxes checked  
155 }  
156 else if (chkBold.isSelected()) {  
157 text.setFont(fontBold); // The Bold check box checked  
158 }  
159 else if (chkItalic.isSelected()) {  
160 text.setFont(fontItalic); // The Italic check box checked  
161 }   
162 else {  
163 text.setFont(fontNormal); // Both check boxes unchecked  
164 }  
165 };  
166   
167 chkBold.setOnAction(handler);  
168 chkItalic.setOnAction(handler);  
169   
170 return pane; // Return a new pane  
171 }  
172 }  
173 class Radiobox extends Checkbox  
174 {  
175 /\*public static void main(String[] args)  
176 {  
177 launch(args);  
178 }\*/  
179 @Override // Override the getPane() method in the super class  
180 protected BorderPane getPane() {  
181 BorderPane pane = super.getPane();  
182   
183 VBox paneForRadioButtons = new VBox(20);  
184 paneForRadioButtons.setPadding(new Insets(5, 5, 5, 5));   
185 paneForRadioButtons.setStyle  
186 ("-fx-border-width: 2px; -fx-border-color: green");  
187   
188 RadioButton rbGold = new RadioButton("Gold");  
189 RadioButton rbOrange = new RadioButton("Orange");  
190 RadioButton rbPurple = new RadioButton("Purple");  
191 paneForRadioButtons.getChildren().addAll(rbGold, rbOrange, rbPurple);  
192 pane.setRight(paneForRadioButtons);  
193   
194 ToggleGroup group = new ToggleGroup();  
195 rbGold.setToggleGroup(group);  
196 rbOrange.setToggleGroup(group);  
197 rbPurple.setToggleGroup(group);  
198   
199 rbGold.setOnAction(e -> {  
200 if (rbGold.isSelected()) {  
201 text.setFill(Color.GOLD);  
202 }  
203 });  
204   
205 rbOrange.setOnAction(e -> {  
206 if (rbOrange.isSelected()) {  
207 text.setFill(Color.ORANGE);  
208 }  
209 });  
210   
211 rbPurple.setOnAction(e -> {  
212 if (rbPurple.isSelected()) {  
213 text.setFill(Color.PURPLE);  
214 }  
215 });  
216   
217 return pane;  
218 }  
219 }  
220 class BouncingRectangle extends Radiobox  
221 {  
222 @Override  
223 protected BorderPane getPane()  
224 {  
225 BorderPane pane = super.getPane();  
226 RectanglePane ballPane = new RectanglePane();  
227 Slider slSpeed = new Slider();  
228 slSpeed.setMax(20);  
229 ballPane.rateProperty().bind(slSpeed.valueProperty());  
230   
231 BorderPane pane2 = new BorderPane();  
232 pane2.setCenter(ballPane);  
233 pane2.setBottom(slSpeed);  
234 pane2.setPrefWidth(100);  
235 pane2.setPrefHeight(100);  
236 pane.setTop(pane2);  
237   
238 return pane;  
239 }  
240 }  
241 class RectanglePane extends Pane  
242 {  
243 public final double width = 50;  
244 public final double height = 20;  
245 private double x = width, y = height;  
246 private double dx = 1, dy = 1;  
247 private Rectangle rectangle = new Rectangle(x, y);  
248 private Timeline animation;  
249   
250 public RectanglePane() {  
251 rectangle.setFill(Color.RED); // Set ball color  
252 getChildren().add(rectangle); // Place a ball into this pane  
253   
254 // Create an animation for moving the ball  
255 animation = new Timeline(  
256 new KeyFrame(Duration.millis(50), e -> moveBall()));  
257 animation.setCycleCount(Timeline.INDEFINITE);  
258 animation.play(); // Start animation  
259 }  
260   
261 public void play() {  
262 animation.play();  
263 }  
264   
265 public void pause() {  
266 animation.pause();  
267 }  
268   
269 public void increaseSpeed() {  
270 animation.setRate(animation.getRate() + 0.1);  
271 }  
272   
273 public void decreaseSpeed() {  
274 animation.setRate(  
275 animation.getRate() > 0 ? animation.getRate() - 0.1 : 0);  
276 }  
277   
278 public DoubleProperty rateProperty() {  
279 return animation.rateProperty();  
280 }  
281   
282 protected void moveBall() {  
283 // Check boundaries  
284 if (x < width || x > getWidth() - width) {  
285 dx \*= -1; // Change ball move direction  
286 }  
287 if (y < height || y > getHeight() - height) {  
288 dy \*= -1; // Change ball move direction  
289 }  
290   
291 // Adjust ball position  
292 x += dx;  
293 y += dy;  
294 rectangle.setX(x);  
295 rectangle.setY(y);  
296 }  
297 }  
298 class MediaDemo extends BouncingRectangle  
299 {  
300 private static final String MEDIA\_URL =   
301 "http://www.unf.edu/~n01185608/big\_buck\_bunny.mp4";  
302   
303 @Override   
304 protected BorderPane getPane() {  
305 Media media = new Media(MEDIA\_URL);  
306 MediaPlayer mediaPlayer = new MediaPlayer(media);  
307 MediaView mediaView = new MediaView(mediaPlayer);  
308   
309 mediaView.setFitWidth(600);  
310 mediaView.setFitHeight(300);  
311   
312 Button playButton = new Button(">");  
313 playButton.setOnAction(e -> {  
314 if (playButton.getText().equals(">")) {  
315 mediaPlayer.play();  
316 playButton.setText("||");  
317 } else {  
318 mediaPlayer.pause();  
319 playButton.setText(">");  
320 }  
321 });  
322   
323 Button rewindButton = new Button("<<");  
324 rewindButton.setOnAction(e -> mediaPlayer.seek(Duration.ZERO));  
325   
326 Slider slVolume = new Slider();  
327 slVolume.setPrefWidth(150);  
328 slVolume.setMaxWidth(Region.USE\_PREF\_SIZE);  
329 slVolume.setMinWidth(30);  
330 slVolume.setValue(50);  
331 mediaPlayer.volumeProperty().bind(  
332 slVolume.valueProperty().divide(100));  
333   
334 HBox hBox = new HBox(10);  
335 hBox.setAlignment(Pos.CENTER);  
336 hBox.getChildren().addAll(playButton, rewindButton,  
337 new Label("Volume"), slVolume);  
338   
339 //GridPane paneForCenterText = new GridPane();  
340 BorderPane pane = super.getPane();  
341 BorderPane pane2 = new BorderPane();  
342 //pane1.setPrefWidth(400);  
343 //pane1.setPrefHeight(300);  
344 pane2.setCenter(mediaView);  
345 pane2.setBottom(hBox);  
346   
347 Pane paneForMedia = new Pane(pane2);  
348   
349 /\*String topStringTest = " ";  
350 Label lblTop = new Label (topStringTest);  
351   
352 paneForCenterText.add(lblTop,1,0);\*/  
353 paneForCenterText.add(paneForMedia,1,0);  
354   
355 RowConstraints row = new RowConstraints();  
356 row.setVgrow(Priority.ALWAYS);  
357 paneForCenterText.getRowConstraints().add(row);  
358   
359 ColumnConstraints col = new ColumnConstraints();  
360 col.setHgrow(Priority.ALWAYS);  
361 paneForCenterText.getColumnConstraints().add(col);  
362   
363 pane.setCenter(paneForCenterText);  
364   
365 return pane;  
366   
367 }  
368 }