

# Programação Orientada por Objetos

---

## **JavaFX – Animações**

Prof. José Cordeiro,

Prof. Cédric Grueau,

Prof. Laercio Júnior

Departamento de Sistemas e Informática

Escola Superior de Tecnologia de Setúbal – Instituto Politécnico de Setúbal

2019/2020

- Sessão 4 – Animações com Linhas de Tempo (**Timeline**)
  - Timeline básica
  - Timeline com eventos
  - Interpoladores (**Interpolators**)
  
- Sessão 5 – Transições Paralelas (**Transitions**)
  - Simultâneas (**ParallelTransition**)
  - Sequenciais (**SequentialTransition**)
  
- Sessão 6 – Exemplo de animação de formas
  - Problema e classes utilizadas
  - Estrutura e principais passos
  - Cálculos envolvidos



Módulo 15 – JavaFX – Animações

# SESSÃO 4 – ANIMAÇÕES COM LINHA DE TEMPO

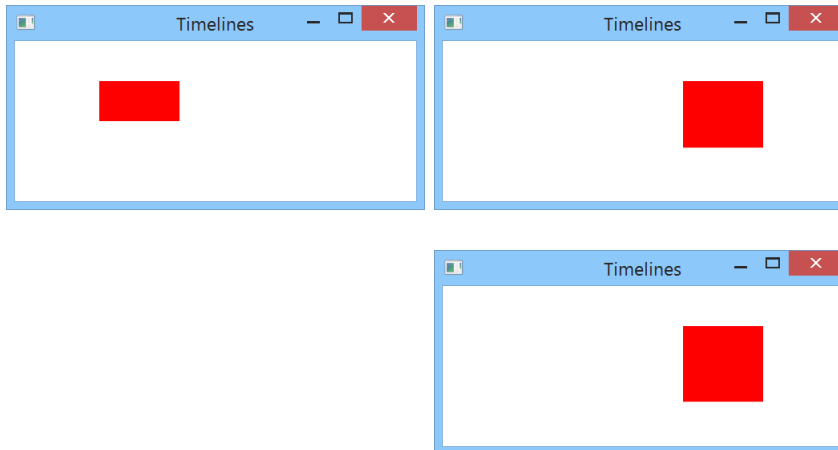
# Animações — Timeline e Transition

- As animações incorporadas no JavaFX são de dois tipos:
  - **Transitions** (transições) - Simples transições entre dois estados (última aula)
  - **Timeline** (linhas de tempo) - para as quais podemos definir diferentes
    - **KeyFrames** (quadros chave) para os quais podemos definir diferentes durações
    - **KeyValues** (valores chave) que irão definir o valor das propriedades do nó animado nos **KeyFrames** que os usam.
- Vamos estudar as potencialidades das animações que recorrem a uma Timeline.

# Animação — TimeLine

@Override

```
public void start(Stage primaryStage) {  
    Rectangle retangulo =  
        new Rectangle(100,50,100,50);  
    retangulo.setFill(Color.RED);  
  
    Group root = new Group();  
    root.getChildren().add(retangulo);  
    Scene scene = new Scene(root, 500, 200);  
  
    primaryStage.setTitle("Timelines");  
    primaryStage.setScene(scene);  
    primaryStage.show();  
}
```



- ❑ Uma **animação** é realizada através de alterações das propriedades do objeto, como por exemplo dimensão, localização, cor, etc.
- ❑ O JavaFX permite controlar como essas alterações são realizadas
- ❑ Criar uma animação em que:
  - A coordenada x, que começa em 100, avance, num 1s, até 300
  - A altura, que começa em 50, aumente, em 2s, até 100

De 0s até 1s:

- Coordenada X: 100 → 300
- Altura: 50 → 75 (metade)

De 1s até 2s:

- Altura: 75 → 100

# Animação — Timeline

```
Timeline timeline = new Timeline();
timeline.setCycleCount(Timeline.INDEFINITE);
timeline.setAutoReverse(true);
```

```
KeyValue keyValue1 = new KeyValue(
    retangulo.xProperty(),
    300.0);
```

```
KeyFrame keyFrame1 = new KeyFrame(
    Duration.millis(1000),
    keyValue1);
```

```
KeyValue keyValue2 = new KeyValue(
    retangulo.heightProperty(),
    100.0);
```

```
KeyFrame keyFrame2 = new KeyFrame(
    Duration.millis(2000),
    keyValue2);
```

```
timeline.getKeyFrames().addAll(
    keyFrame1, keyFrame2);
timeline.play();
```

```
}
```


## □ Timeline

javafx.animation.Timeline

javafx.animation.KeyFrame

javafx.animation.KeyValue

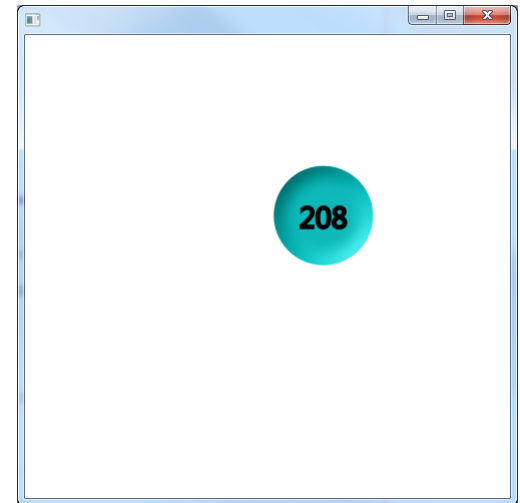
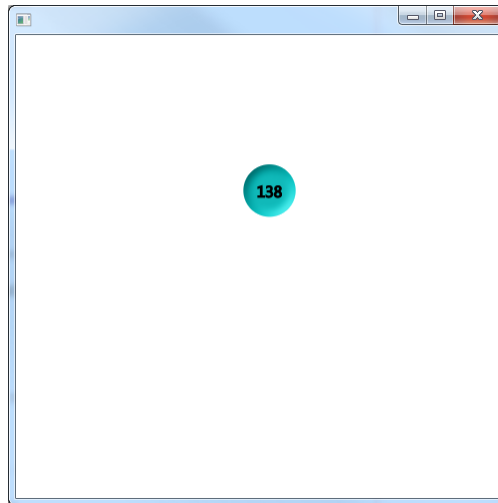
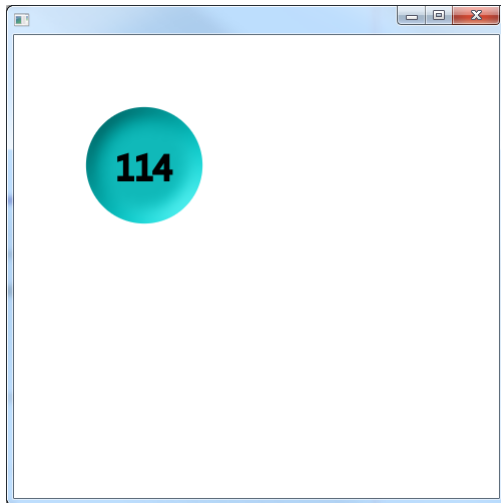
- As **Timeline** permitem atualizar as suas propriedades ao longo do tempo.
- O JavaFX suporta animação por **key frame** (quadros chave).
- Na animação por key frame as transições entre estados de uma animação são marcadas por quadros iniciais e finais (key frames ou quadros chave).
- O Sistema executa a animação automaticamente e pode terminá-la, pará-la (**pause**), recomeçá-la (**resume**) ou repeti-la (**repeat**).

Coord. X:	100	300	
Altura:	50	75	100
Instante:	0s	1s	2s
			

# Animação — Eventos numa Timeline

## □ Timeline Events

- O JavaFX permite incorporar eventos que são despoletados no decorrer da animação com **Timeline**.
- Criar animações
  - Escrever um número, crescente, dentro do círculo (a laranja no programa) – **AnimationTimer**
  - Alterar o raio de um círculo (a verde no programa) – **Timeline**
  - Alterar, aleatoriamente, o centro de um círculo (a azul no programa) – **onFinished Event**



# Animação — Eventos numa TimeLine (cont.)

```
public class JavaFXAnimationTime extends Application {  
  
    public static void main(String[] args) {  
        launch(args);  
    }  
  
    //Valor do interior do circulo  
    private Integer valor = 0; //Integer em vez de int para se fazer o toString()  
  
    @Override  
    public void start(Stage stage) {  
        //Circulo com efeito  
        final Circle circle = new Circle(20, Color.CYAN);  
        circle.setEffect(new Lighting());  
  
        //Texto dentro do circulo  
        final Text text = new Text(valor.toString());  
        text.setStroke(Color.BLACK);  
    }  
}
```



# Animação — Eventos numa TimeLine (cont.)

```
//Painel para conter os elementos (StackPane = centrados)
final StackPane painel = new StackPane();
painel.getChildren().addAll(circle, text);
painel.setLayoutX(30); //X inicial (devia ser uma constante)
painel.setLayoutY(30); //Y inicial (devia ser uma constante)

Scene scene = new Scene(painel, 500, 500);
stage.setScene(scene);
stage.show();

//AnimationTimer executa o método handle em cada "frame"
AnimationTimer timer = new AnimationTimer() {
    @Override
    public void handle(long now) {
        text.setText(valor.toString());
        valor++;
    }
};

//AnimationTimer é uma classe abstrata
//com dois métodos concretos: start, stop
//e um método abstrato, que tem de ser definido: handle
//Por ter mais de um método não pode ser utilizada lambda expression
```

# Animação — Eventos numa TimeLine (cont.)

```
//Criar os keyValue que aumentem 3 vezes a escala, em X e Y
KeyValue keyValueX = new KeyValue(painel.scaleXProperty(), 3);
KeyValue keyValueY = new KeyValue(painel.scaleYProperty(), 3);

//Duração para atingir os valores anteriores: 2s
Duration duration = Duration.millis(2000);

//O que fazer ao atingir o fim da animação: mudar o centro e reset do valor
EventHandler onFinished = e -> {
    //Mudar o centro
    painel.setTranslateX(java.lang.Math.random() * 300 - 100);
    painel.setTranslateY(java.lang.Math.random() * 200 - 100);
    //Reset do valor
    valor = 0;
};

//Criar o keyFrame com base na informação anterior
KeyFrame keyFrame = new KeyFrame(duration, onFinished, keyValueX, keyValueY);
```

# Animação — Eventos numa TimeLine (cont.)

```
//Criar a timeline e indicar as suas características
Timeline timeline = new Timeline();
timeline.setCycleCount(Timeline.INDEFINITE);
timeline.setAutoReverse(true);
timeline.getKeyFrames().add(keyFrame);

//Iniciar as animações
timeline.play();
timer.start();
}
```

# Animação — Interpolators

```
Rectangle rectangle = new Rectangle(100, 50, 100, 50);  
rectangle.setFill(Color.BROWN);
```

```
Timeline timeline = new Timeline();  
timeline.setCycleCount(Timeline.INDEFINITE);  
timeline.setAutoReverse(true);
```

```
KeyValue keyValue = new KeyValue(  
    rectangle.xProperty(),  
    300.0,  
    Interpolator.EASE_BOTH);
```

```
KeyFrame keyFrame = new KeyFrame(  
    Duration.millis(2000),  
    keyValue);
```

```
timeline.getKeyFrames().add(keyFrame);  
timeline.play();
```

## ❑ Interpoladores

- A interpolação define (calcula) as posições intermédias de um objeto entre um ponto inicial e um ponto final do seu movimento.
- Podemos recorrer a diferentes implementações de interpoladores já existentes na classe **Interpolator**.
- O exemplo ao lado mostra a utilização de um **Interpolator.EASE\_BOTH** que cria um efeito de "mola" quando o objeto atinge os pontos final e inicial.

# Animação — Interpolators

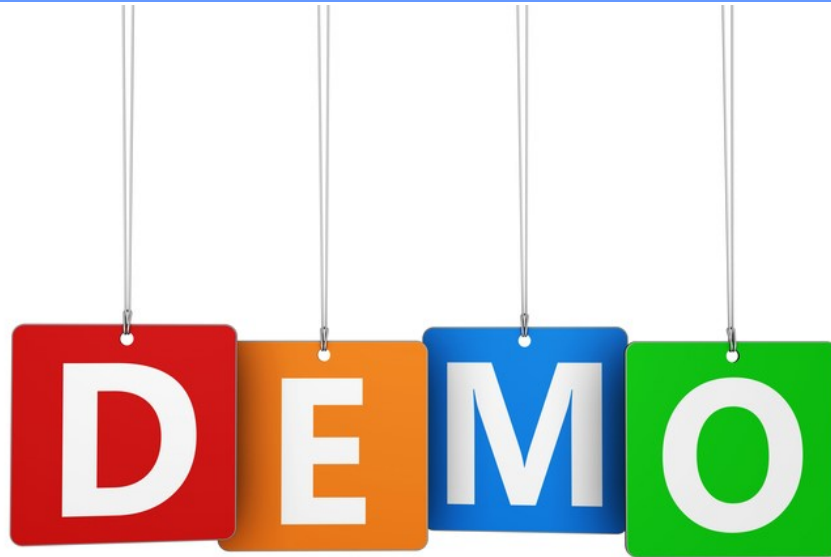
```
public class InterpolatorAnima extends Interpolator {  
    @Override  
    protected double curve(double t) {  
        return Math.abs(0.5-t)*2 ;  
    }  
}
```

```
final KeyValue keyValue = new KeyValue(  
    rectangle.xProperty(),  
    300.0,  
    new InterpolatorAnima());
```

## □ Interpoladores personalizados

- Por omissão o JavaFX usa interpolação linear (**Interpolator.LINEAR**) para calcular as coordenadas intermédias.
- Mas podemos criar os nossos próprios interpoladores que recorram a outras formas de cálculo.
- O exemplo ao lado mostra a definição de uma classe **InterpolatorAnima** que herda de **Interpolator** e redefina o método **curve** (que deve devolver valor entre [0.0,1.0]), indicando o "fator multiplicativo".

# JavaFX- Eventos : Exemplo





Módulo 15 – JavaFX – Animações

# SESSÃO 5 – TRANSIÇÕES PARALELAS

# Animação — Transições Simultâneas

```
public class Paralelo extends Application { // método main omitido
    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Parallel Transition");
        Group root = new Group();
        Scene scene = new Scene(root,500,300);
        Rectangle rectangle = new Rectangle(100,100,50,50);
        rectangle.setFill(Color.DARKBLUE);
        root.getChildren().add(rectangle);
        primaryStage.setScene(scene);
        primaryStage.show();

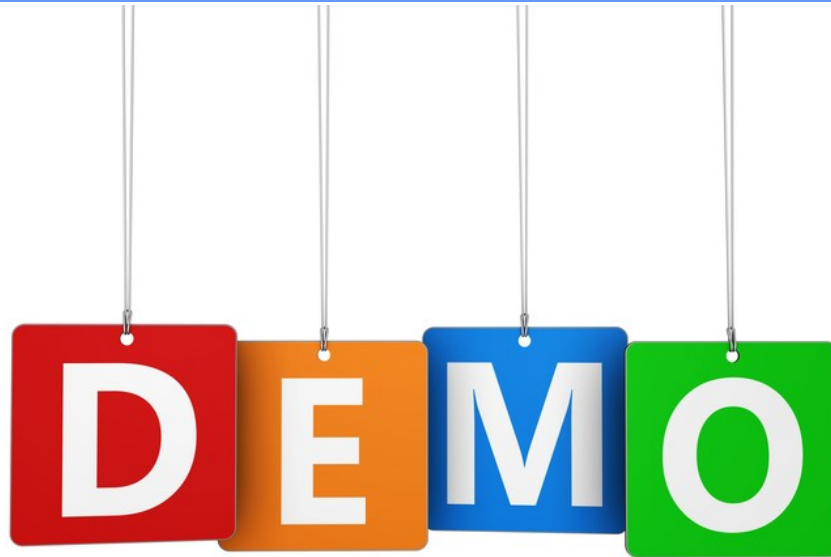
        RotateTransition animaRoda = new RotateTransition(
            Duration.seconds(3), rectangle);
        animaRoda.setByAngle(180f);
        animaRoda.setCycleCount(4);
        animaRoda.setAutoReverse(true);
        ScaleTransition animaTamanho = new ScaleTransition(
            Duration.seconds(2), rectangle);
        animaTamanho.setToX(2f);
        animaTamanho.setToY(2f);
        animaTamanho.setCycleCount(2);
        animaTamanho.setAutoReverse(true);
        ParallelTransition parallelTransition = new ParallelTransition();
        parallelTransition.getChildren().addAll(animaRoda, animaTamanho);
        parallelTransition.setCycleCount(Timeline.INDEFINITE);
        parallelTransition.play();
    }
}
```

## □ ParallelTransition

- Executa várias transições em simultâneo.
- O exemplo mostra a aplicação de uma **ParallelTransition** que executa a rotação e o redimensionamento simultâneos de um retângulo



# JavaFX- Eventos : Exemplo



# Animação — Transições Sequenciais

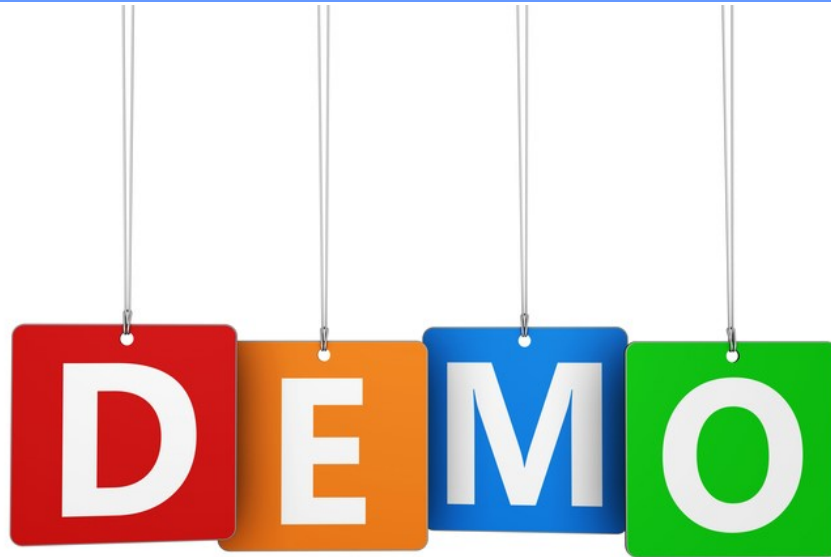
```
@Override
public void start(Stage primaryStage) {
    primaryStage.setTitle("Sequential Transition");
    Group root = new Group();
    Scene scene = new Scene(root, 500, 300);
    Rectangle rectangle = new Rectangle(100, 100, 50, 50);
    rectangle.setFill(Color.DARKBLUE);
    root.getChildren().add(rectangle);
    primaryStage.setScene(scene);
    primaryStage.show();

    RotateTransition animaRoda = new RotateTransition(
        Duration.seconds(3), rectangle);
    animaRoda.setByAngle(180f);
    animaRoda.setCycleCount(1);
    animaRoda.setAutoReverse(true);
    ScaleTransition animaTamanho = new ScaleTransition(
        Duration.seconds(2), rectangle);
    animaTamanho.setToX(2f);
    animaTamanho.setToY(2f);
    animaTamanho.setCycleCount(2);
    animaTamanho.setAutoReverse(true);
    SequentialTransition animaSequential = new SequentialTransition();
    animaSequential.getChildren().addAll(animaRoda, animaTamanho);
    animaSequential.setCycleCount(Timeline.INDEFINITE);
    animaSequential.setAutoReverse(true);
    animaSequential.play();
}
```

## □ SequentialTransition

- Executa várias transições uma a seguir à outra.
- O exemplo ao lado ilustra a criação de uma **SequentialTransition** que executa uma rotação e em seguida um redimensionamento de um retângulo

# JavaFX- Eventos : Exemplo



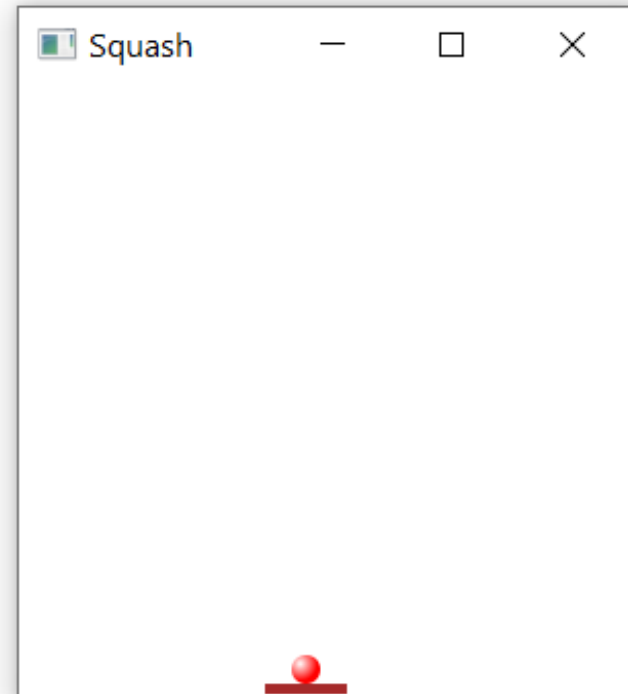


Módulo 15 – JavaFX – Animações

## **SESSÃO 6 – EXEMPLO SQUASH**

# Exemplo de Animação

- Fazer um jogo que simula o "Squash":
  - Existe, na base, uma raquete que é movida pelo rato
  - A bola é lançada (com o clique do rato) num determinado angulo
  - Ao bater nas paredes (laterais e de topo) é redirecionada para a parede seguinte
  - Quando chega à base se não estiver sobre a raquete sai do jogo e é perdida



# Exemplo de Animação — Squash

```
public class Squash extends Application {  
  
    @Override  
    public void start(Stage primaryStage) {  
        final Jogo jogo = new Jogo();  
  
        Scene scene = new Scene(jogo, Jogo.LARGURA, Jogo.ALTURA);  
  
        scene.setOnMouseClicked(e -> jogo.iniciarMoverBola());  
  
        scene.setOnMouseMoved(e -> jogo.moverRaqueteBola(e.getX()));  
  
        scene.setCursor(Cursor.NONE);  
  
        primaryStage.setTitle("Squash");  
        primaryStage.setScene(scene);  
        primaryStage.show();  
    }  
  
    public static void main(String[] args) {  
        launch(args);  
    }  
}
```

## □ Classe principal

- Cria-se um **Jogo** (será um **Group** com os elementos gráficos).
- A cena é montada com o Jogo e as respetivas dimensões.
- Prepara-se, na cena (para ser ativado globalmente), o **EventHandler** de clique do rato, para lançar a Bola.
- Prepara-se, na cena (para ser ativado globalmente), o **EventHandler** de mover o rato, para mover a Raquete (em função da coordenada X do rato).
- Esconde-se o rato

# Exemplo de Animação — Squash (cont.)

```
public class Jogo extends Group {

    public static final int LARGURA = 300;
    public static final int ALTURA = 300;
    public static final int PADDING = 2;

    private boolean bolaParada;
    private Bola bola;
    private Raquete raquete;

    public Jogo() {
        bolaParada = true;
        raquete = new Raquete();
        bola = new Bola(this);
        getChildren().addAll(raquete, bola);
    }

    public Bola getBola() {
        return bola;
    }

    public Raquete getRaquete() {
        return raquete;
    }
}
```

## □ Classe Jogo

- Define-se as constantes características do Jogo.
- Guarda-se a indicação se a Bola está parada ou em movimento, a Bola e a Raquete.
- Disponibiliza-se acesso à Bola e à Raquete do Jogo.

# Exemplo de Animação — Squash (cont.)

```
public void moverRaqueteBola(double x) {  
    raquete.mover(x);  
    if (bolaParada) {  
        bola.moverComRaquete(raquete);  
    }  
}  
  
public void iniciarMoverBola() {  
    if (bolaParada) {  
        bolaParada = false;  
        bola.iniciarMover(this);  
    }  
}  
  
public void pararMoverBola() {  
    bolaParada = true;  
}  
}
```

## □ Classe Jogo

### ■ Ações disponíveis:

- Mover a Raquete e, eventualmente (se não estiver em movimento) a Bola.
- Iniciar o movimento da Bola, se ainda não estiver em movimento.
- Parar o movimento da Bola (voltará a andar associada à Raquete).



# Exemplo de Animação — Squash (cont.)

```
public class Raquete extends Rectangle {

    public static final int LARGURA = 40;
    public static final int MEIO_RAQUETE = Raquete.LARGURA / 2;
    public static final int ALTURA = 5;
    public static final int X_ORIGINAL = Jogo.LARGURA / 2
        - Raquete.LARGURA / 2;
    public static final int Y_ORIGINAL = Jogo.ALTURA - Jogo.PADDING
        - Raquete.ALTURA;
    public static final int X_MIN = Jogo.PADDING
        + Math.max(0, Bola.RAIO - Raquete.LARGURA / 2);
    public static final int X_MAX = Jogo.LARGURA - Jogo.PADDING
        - Raquete.LARGURA - Math.max(0, Bola.RAIO - Raquete.LARGURA / 2);

    public Raquete() {
        setWidth(Raquete.LARGURA);
        setHeight(Raquete.ALTURA);
        setX(Raquete.X_ORIGINAL);
        setY(Raquete.Y_ORIGINAL);
        setFill(Color.BROWN);
    }

    public void mover(double x) {
        setX(Math.max(Raquete.X_MIN, Math.min(x, Raquete.X_MAX)));
    }

    public boolean dentro(double x) {
        return (x >= getX()) && (x <= getX() + Raquete.LARGURA);
    }

    public double getMeioRaquete() {
        return getX() + Raquete.MEIO_RAQUETE;
    }
}
```

## □ Classe Raquete

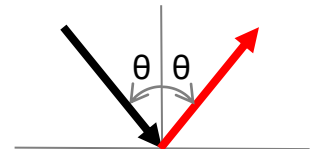
- Define-se as constantes características da Raquete.
- Ações disponíveis:
  - Mover a Raquete, em função da coordenada X do rato (veio do **EventHandler**) dentro dos limites definidos.
  - Verificar se uma coordenada X (do centro da Bola) está incluída dentro da Raquete.
  - Obter a posição do meio da Raquete

# Exemplo de Animação — Squash (cont.)

```
public class Bola extends Circle {  
  
    public static final int RAIIO = 7;  
    public static final int X_ORIGINAL = Jogo.LARGURA / 2;  
    public static final int Y_ORIGINAL = Raquete.Y_ORIGINAL - Bola.RAIIO;  
  
    public static final int X_MIN = Jogo.PADDING + Bola.RAIIO;  
    public static final int X_MAX = Jogo.LARGURA - Jogo.PADDING  
        - Bola.RAIIO;  
    public static final int Y_MIN = Jogo.PADDING + Bola.RAIIO;  
    public static final int Y_MAX = Raquete.Y_ORIGINAL - Bola.RAIIO;  
  
    public static final int LARGURA_DISPONIVEL = Bola.X_MAX  
        - Bola.X_MIN;  
    public static final int ALTURA_DISPONIVEL = Bola.Y_MAX  
        - Bola.Y_MIN;  
  
    public static final double VELOCIDADE = 3; //Quanto maior, mais lento  
  
    private Random aleatorio;  
    private boolean cimaParaBaixo;  
    private boolean esquerdaParaDireita;  
    private double tangenteAngulo;
```

## □ Classe Bola

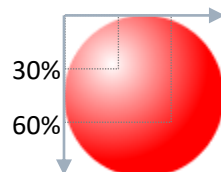
- Define as constantes características da Bola.
- Guarda os valores necessários para o cálculo dos ângulos de reflexão das paredes:
- Gerador aleatório para simular os dados do lançamento da Bola.
- Direções do movimento (cima ↔ baixo e esquerda ↔ direita).
- Tangente do ângulo de "ataque" (explicado mais à frente).



# Exemplo de Animação — Squash (cont.)

```
public Bola(final Jogo jogo) {
    //Definir características da bola:
    setRadius(Bola.RAIO);
    setCenterX(Bola.X_ORIGINAL);
    setCenterY(Bola.Y_ORIGINAL);
    setFill(gradienteBola());
    //Preparar a geração de números aleatórios:
    aleatorio = new Random();
}

private RadialGradient gradienteBola() {
    return new RadialGradient(
        0, //Angulo de focus
        0, //Distância de focus
        0.3, //Centro X: 30%
        0.3, //Centro Y: 30%
        0.6, //Raio: 60%
        true, //as coordenadas anteriores são indicadas em [0,1],
            // representando percentagens, em relação ao "contentor"
        CycleMethod.NO_CYCLE,
        new Stop(0, Color.WHITE),
        new Stop(1, Color.RED));
}
```



## □ Classe Bola

- O construtor da Bola define as suas características.
- A Bola é preenchida com um gradiente que lhe dá um aspeto tridimensional.
- Define-se o gradiente em função das coordenadas do objeto envolvente.
- Usa-se um gradiente radial cujo o foco está descentrado em relação ao objeto 30% e com um raio de 60%.
- O gradiente começa com a cor branca (simular luz) e termina na cor vermelha.

# Exemplo de Animação — Squash (cont.)

```
public void iniciarMover(Jogo jogo) {
    cimaParaBaixo = true;
    esquerdaParaDireita = aleatorio.nextBoolean();
    tangenteAngulo = Math.tan(aleatorio.nextDouble() * (Math.PI / 2.0));
    fazerTranslacao(jogo);
}

public void pararMover(Jogo jogo) {
    setTranslateX(0);
    setTranslateY(0);
    jogo.pararMoverBola();
    moverComRaquete(jogo.getRaquete());
}

public void moverComRaquete(Raquete raquete) {
    setCenterX(raquete.getMeioRaquete());
}

public int xAtual() {
    return (int) Math.round(getBoundsInParent().getMinX() + Bola.RAIO);
    //ou (int) Math.round(getBoundsInParent().getMaxX() - Bola.RAIO);
}

public int yAtual() {
    return (int) Math.round(getBoundsInParent().getMinY() + Bola.RAIO);
    //ou (int) Math.round(getBoundsInParent().getMaxY() - Bola.RAIO);
}
```


## □ Classe Bola

- As operações relacionadas com o movimento da Bola são:
  - Iniciar o movimento da Bola (definindo as suas características, de forma aleatória) e depois realizar a translação.
  - Parar o movimento da Bola.
  - Mover a Bola em conjunto com a Raquete (para quando a Bola não está em movimento).
  - Determinar as coordenadas X,Y atuais da Bola (em função do `getBoundsInParent()`).
  - Fazer a translação (apresentada adiante)

# Exemplo de Animação — Squash (cont.)

```
public boolean naLinhaDaRaquete() {  
    return (yAtual() >= Bola.Y_MAX);  
}  
  
public boolean naMargemTopo() {  
    return (yAtual() <= Bola.Y_MIN);  
}  
  
public boolean naMargemEsquerda() {  
    return (xAtual() <= Bola.X_MIN);  
}  
  
public boolean naMargemDireita() {  
    return (xAtual() >= Bola.X_MAX);  
}
```

## □ Classe Bola

- É necessário determinar em que parede a Bola se encontra.
- Esta determinação é feita em função da posição atual e das coordenadas X,Y limites (o sistema de coordenadas do JavaFX é  ).

# Exemplo de Animação — Squash (cont.)

```
public void fazerTranslacao(final Jogo jogo) {  
    //Determinar as distâncias a percorrer:  
    int x = xAtual();  
    int y = yAtual();  
    double dX;  
    double dY;  
  
    <algoritmo de cálculo, em função das direções do movimento e da  
    tangente do ângulo de ataque (cimaParaBaixo, esquerdaParaDireita  
    e tangenteAngulo), da posições de destino na próxima parede>  
  
    //Criar, parametrizar e executar a translação:  
    ...próximo slide...
```

## □ Classe Bola

- A animação do movimento é feita através de translações sucessivas de "parede a parede".
- Para a criação da translação é necessário determinar a distância percorrida nas coordenadas X,Y (**dX** e **dY**).
- Com esses valores é possível criar a translação.

# Exemplo de Animação — Squash (cont.)



```
//Criar, parametrizar e executar a translação:
TranslateTransition translacao =
    new TranslateTransition(Duration.millis(
        Math.sqrt(dX * dX + dY * dY) * Bola.VELOCIDADE),
        this);
translacao.setOnFinished(e -> {
    if (naLinhaDaRaquete()
        && !jogo.getRaquete().dentro(xAtual())) {
        //Saiu pelo fundo: termina
        pararMover(jogo);
    } else {
        //Não saiu pelo fundo: avança com nova translação
        fazerTranslacao(jogo);
    }
});
translacao.setByX(dX);
translacao.setByY(dY);
translacao.play();
}
```

## □ Classe Bola

- A duração da translação é função da distância percorrida (teorema de Pitágoras) vezes uma velocidade pré-estabelecida.
- Para além da duração, é necessário indicar o objeto a mover (a Bola – **this**).
- No final da translação (**EventHandler OnFinished**) verifica-se, caso esteja na base, se não há sobreposição com a Raquete. Se sim, então termina, caso contrário executa uma nova animação até à outra parede.


# Exemplo de Animação — Squash (cont.)

## ❑ Determinação da translação

- O algoritmo de cálculo das posições de destino na próxima parede depende das direções que o movimento da Bola trás e da tangente do ângulo de ataque.
- Será necessário perceber as diferentes alternativas possíveis (slides seguintes)
- Com a análise das várias alternativas possíveis sistematizam-se duas situações  e 

- ❑ Para testar o funcionamento de toda a aplicação (movimento da Raquete, iniciar/parar o movimento da Bola, visualizar a translação, etc.) é possível implementar um movimento, muito simples – a Bola apenas se deslocar na vertical (cima ↔ baixo):

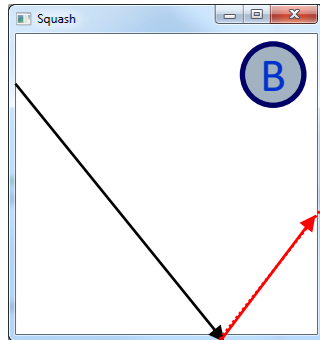
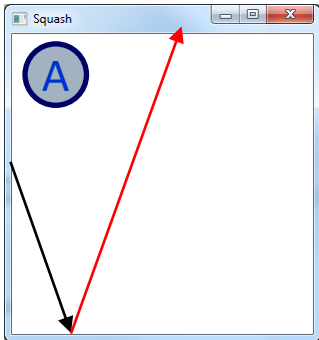
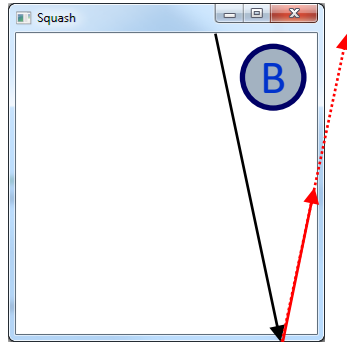
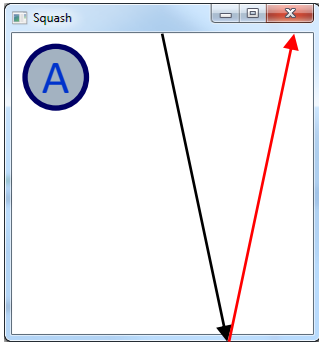
```
//Algoritmo para movimentar a bola, apenas na vertical:  
dx = 0.0;  
if (naLinhaDaRaquete()) {  
    dy = -Bola.ALTURA_DISPONIVEL;  
} else {  
    dy = Bola.ALTURA_DISPONIVEL;  
}
```

- ❑ Nota: em JavaFX o sistema de coordenadas é . Assim, quando está na linha da Raquete, a Bola sobe ao diminuir os valores em y ( $dy = -...$ ) e quando está no topo, a Bola desce ao aumentar os valores em y ( $dy = +...$ ).

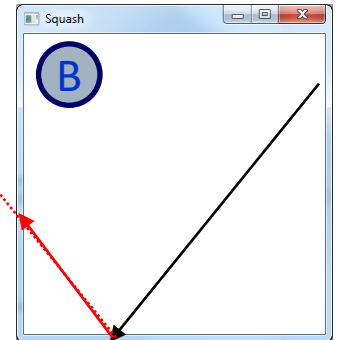
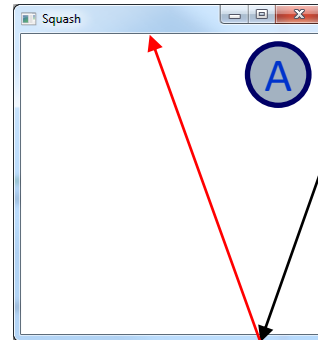
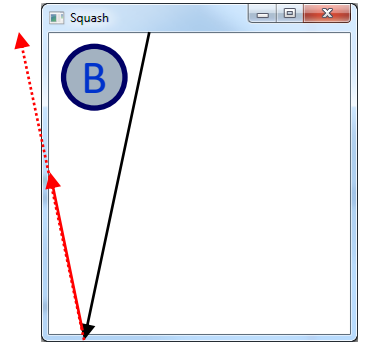
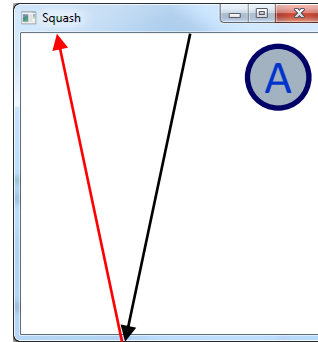


# Bola chega à Base (cima → baixo)

esquerda → direita

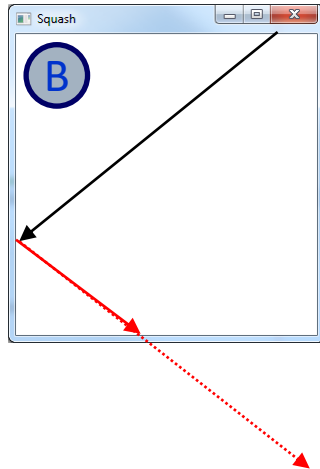
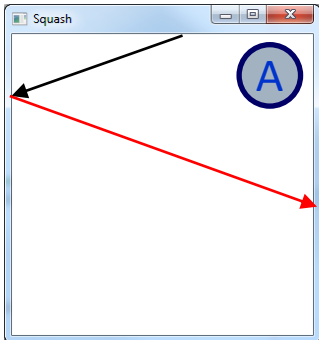
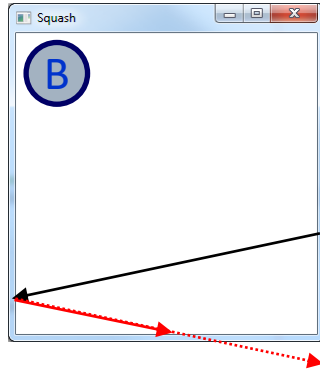
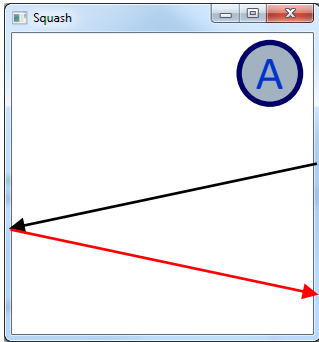


direita → esquerda

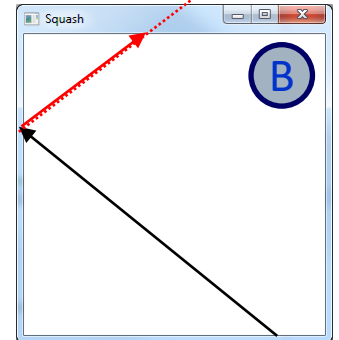
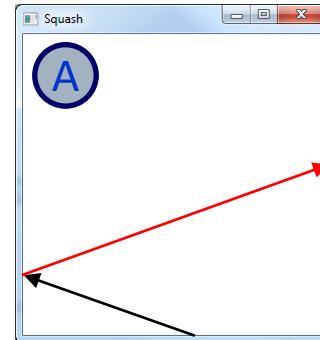
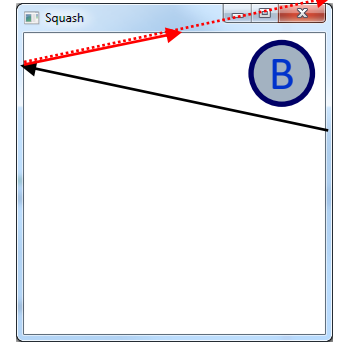
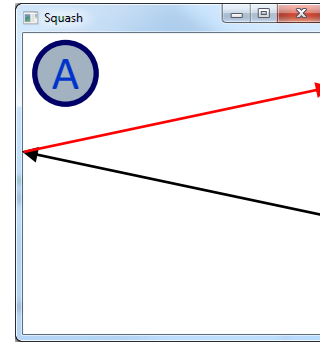


# Bola chega à Esquerda (direita → esquerda)

cima → baixo

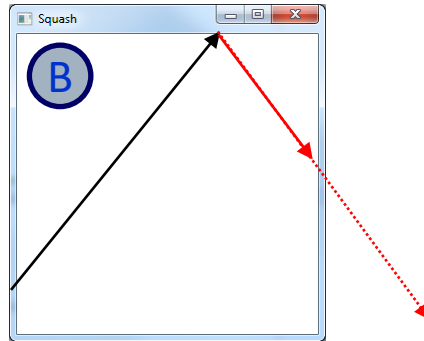
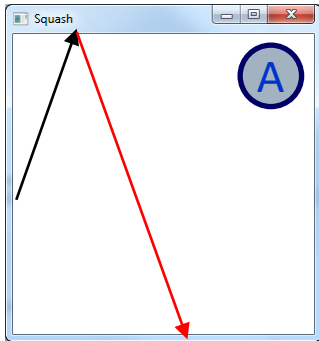
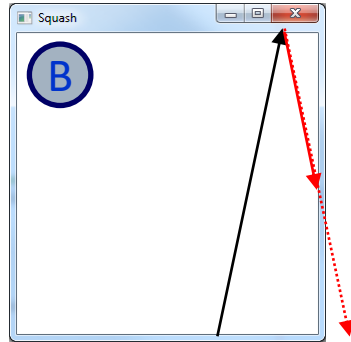
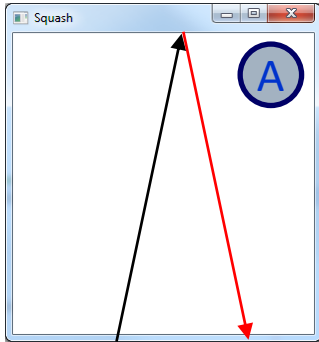


baixo → cima

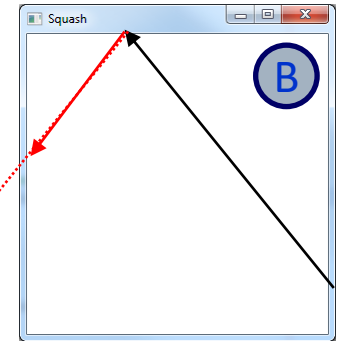
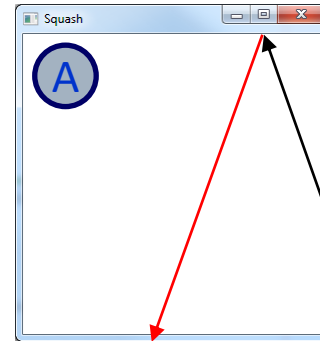
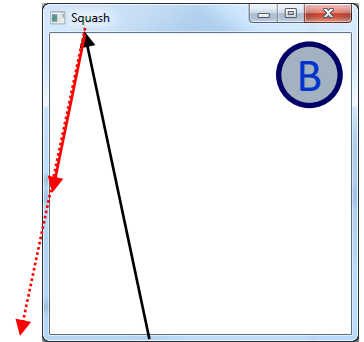
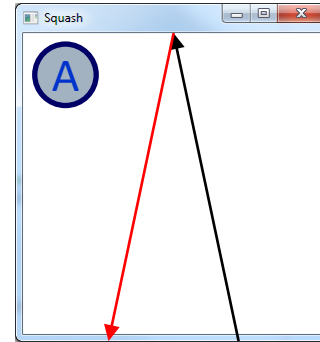


# Bola chega ao Topo (baixo → cima)

esquerda → direita

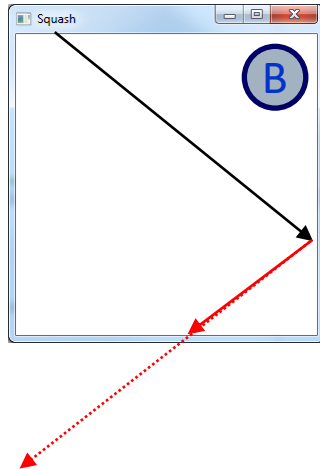
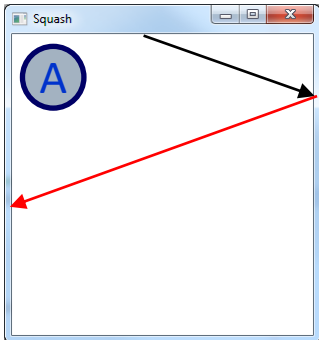
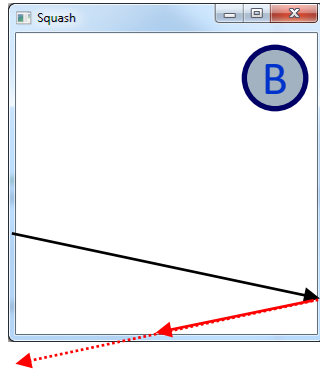
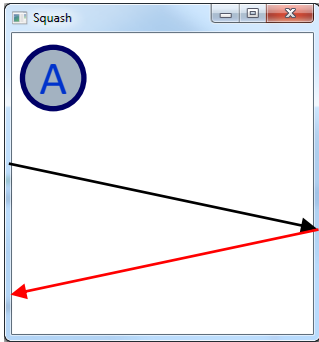


direita → esquerda

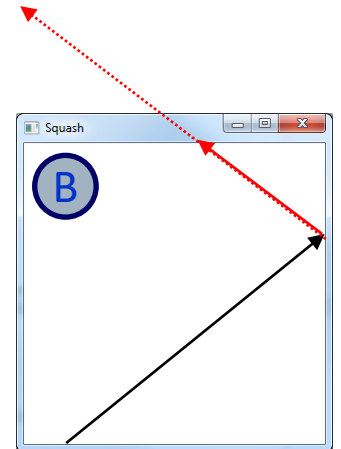
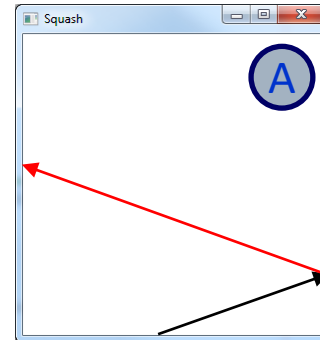
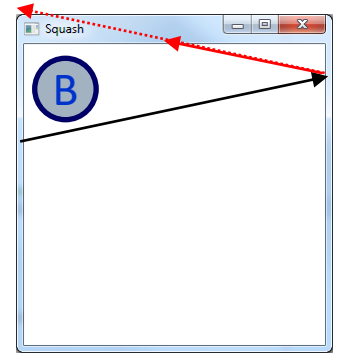
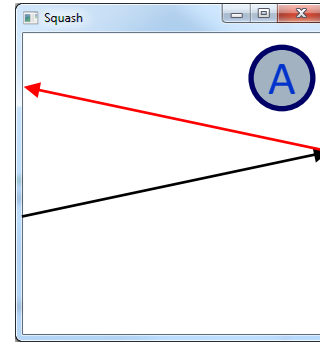


# Bola chega à Direita (esquerda → direita)

cima → baixo



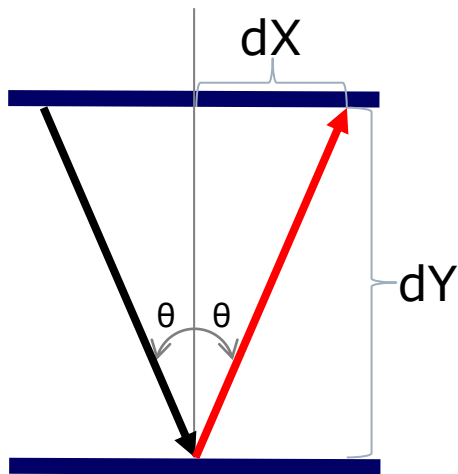
baixo → cima



# Situações Possíveis

## □ Situação A

- A Bola, ao bater ( $\leftarrow$ ) na parede, sairá ( $\rightarrow$ ) para um caminho completo até à parede oposta.
- A distância percorrida, numa das coordenadas, será igual a **LARGURA\_DISPONIVEL** ou **ALTURA\_DISPONIVEL** (consoante o movimento).
- A distância percorrida, pela outra coordenada, será calculada através de (foi escolhida uma das situações A). Nos cálculos aqui apresentados assume-se o sistema de coordenadas tradicional da matemática  $y \uparrow x \rightarrow$ ):

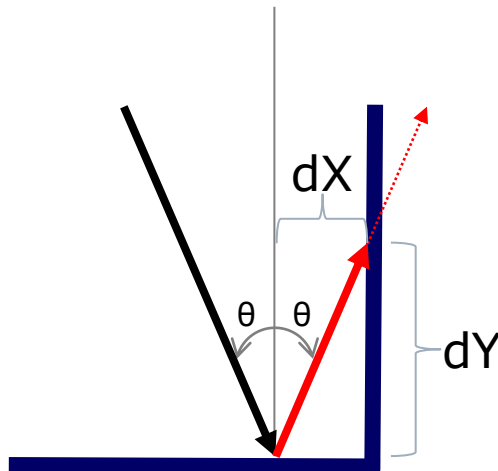


$$\begin{aligned} dY &= \text{ALTURA\_DISPONIVEL} \\ dX &= dY \times \text{tg}(\theta) \end{aligned}$$

# Situações Possíveis

## □ Situação B

- O movimento para a parede oposta não será possível, atingindo-se, antes, a parede adjacente.
- Não será possível ter uma distância percorrida, numa das coordenadas, igual a **LARGURA\_DISPONIVEL** ou **ALTURA\_DISPONIVEL** (consoante o movimento): somando a coordenada do ponto de embate (na parede) com essa distância percorrida, teremos um valor que excede o limite máximo (na situação apresentada teríamos  $x + dX > X\_MAX$ ).
- As distâncias percorridas, em cada coordenada, passaram a ser calculadas através de:



Se  $X\_MAX$  for igual a  $x$  (no vértice)  
 $dX = X\_MAX$  (ou outro  $\neq$  de zero)  
senão  
 $dX = X\_MAX - x$   
 $dY = dX / \text{tg}(\theta)$

# Exemplo de Animação — Squash (cont.)

## □ Determinação do movimento da Bola

- Os cálculos são feitos em função de:
  - parede onde a Bola se encontra;
  - direção do movimento (atributos **cimaParaBaixo** e **esquerdaParaDireita**);
  - da tangente do ângulo de "ataque".
- Ao estar numa das paredes, haverá uma das direções do movimento (atributo **cimaParaBaixo** ou **esquerdaParaDireita**) que será afetada (modificada).
- Começa-se por assumir que toda a distância até à parede oposta será percorrida e determina-se a distância da outra coordenada em função da multiplicação pela tangente (situação **A**).
- Se não for possível percorrer toda a distância até à parede oposta (a outra coordenada sairia da área disponível) então recalculam-se as distância com o limite possível e novamente utilizando a tangente do ângulo de "ataque" (situação **B**).

# Exemplo de Animação — Squash (cont.)

//Algoritmo para movimentar a bola:

```
if (naLinhaDaRaquete()) {
    cimaParaBaixo = false;
    modificarTangenteAngulo (x, jogo.getRaquete());
    dY = -Bola.ALTURA_DISPONIVEL;
    if (esquerdaParaDireita) {
        dX = -dY * tangenteAngulo;
        if (x + dX > Bola.X_MAX) {
            dX = ((Bola.X_MAX == x) ? Bola.X_MAX : Bola.X_MAX - x);
            dY = -(dX / tangenteAngulo);
        }
    } else {
        dX = dY * tangenteAngulo;
        if (x + dX < Bola.X_MIN) {
            dX = -((Bola.X_MIN == x) ? Bola.X_MIN : x - Bola.X_MIN);
            dY = dX / tangenteAngulo;
        }
    }
}
```

## □ Na linha da Raquete:

- Para introduzir um pequeno grau de incerteza será necessário modificar, ligeiramente, o ângulo de "ataque", quando a Bola toca na Raquete (método **modificarTangenteAngulo**)



# Exemplo de Animação — Squash (cont.)

```
} else if (naMargemEsquerda()) {  
    esquerdaParaDireita = true;  
    dX = Bola.LARGURA_DISPONIVEL;  
    if (cimaParaBaixo) {  
        dY = dX * tangenteAngulo;  
        if (y + dY > Bola.Y_MAX) {  
            dY = ((Bola.Y_MAX == y) ? Bola.Y_MAX : Bola.Y_MAX - y);  
            dX = dY / tangenteAngulo;  
        }  
    } else {  
        dY = -(dX * tangenteAngulo);  
        if (y + dY < Bola.Y_MIN) {  
            dY = -((Bola.Y_MIN == y) ? Bola.Y_MIN : y - Bola.Y_MIN);  
            dX = -dY / tangenteAngulo;  
        }  
    }  
}
```

□ Na parede da esquerda

# Exemplo de Animação — Squash (cont.)

```
} else if (naMargemTopo()) {  
    cimaParaBaixo = true;  
    dY = Bola.ALTURA_DISPONIVEL;  
    if (esquerdaParaDireita) {  
        dX = dY * tangenteAngulo;  
        if (x + dX > Bola.X_MAX) {  
            dX = ((Bola.X_MAX == x) ? Bola.X_MAX : Bola.X_MAX - x);  
            dY = dX / tangenteAngulo;  
        }  
    } else {  
        dX = -(dY * tangenteAngulo);  
        if (x + dX < Bola.X_MIN) {  
            dX = -((Bola.X_MIN == x) ? Bola.X_MIN : x - Bola.X_MIN);  
            dY = -dX / tangenteAngulo;  
        }  
    }  
}
```

□ Na parede do topo

# Exemplo de Animação — Squash (cont.)

```
} else if (naMargemDireita()) {
    esquerdaParaDireita = false;
    dX = -Bola.LARGURA_DISPONIVEL;
    if (cimaParaBaixo) {
        dY = -dX * tangenteAngulo;
        if (y + dY > Bola.Y_MAX) {
            dY = ((Bola.Y_MAX == y) ? Bola.Y_MAX : Bola.Y_MAX - y);
            dX = -dY / tangenteAngulo;
        }
    } else {
        dY = dX * tangenteAngulo;
        if (y + dY < Bola.Y_MIN) {
            dY = -((Bola.Y_MIN == y) ? Bola.Y_MIN : y - Bola.Y_MIN);
            dX = dY / tangenteAngulo;
        }
    }
} else { //Por segurança
    return;
}

dX = Math.round(dX);
dY = Math.round(dY);
```

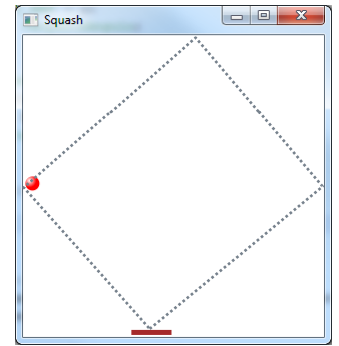
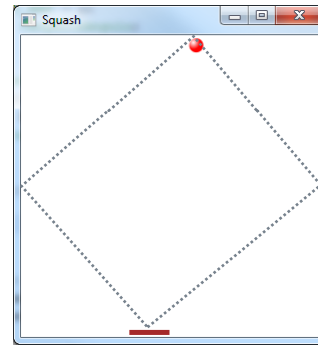
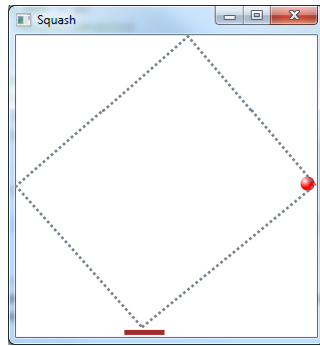
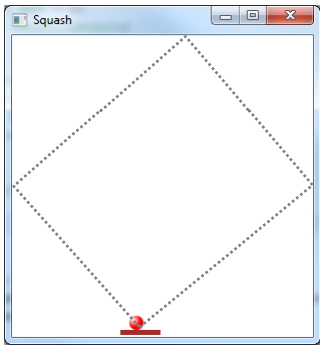
❑ Na parede da direita

❑ Após tratar todas as alternativas possíveis e determinar-se o dX e dY a aplicar, arredondam-se esses valores obtidos.

# Exemplo de Animação — Squash (cont.)

## ❑ Modificação do ângulo de "ataque"

- Se nada for feito, ao ângulo de "ataque", o sistema manterá a inclinação do movimento da Bola indefinidamente.
- É necessário alterar, ligeiramente, a tangente do ângulo de "ataque"



# Exemplo de Animação — Squash (cont.)

```
public void modificarTangenteAngulo(int x, Raquete raquete) {  
    double meioRaquete = raquete.getMeioRaquete();  
    double percentagemVariacao = 0.25; //25%  
    if (tangenteAngulo == 0) {  
        tangenteAngulo = percentagemVariacao;  
    } else if (esquerdaParaDireita) {  
        if (x > meioRaquete) {  
            //Aumenta ângulo:  
            tangenteAngulo = (1 + percentagemVariacao) * tangenteAngulo;  
        } else {  
            //Diminuí ângulo:  
            tangenteAngulo = (1 - percentagemVariacao) * tangenteAngulo;  
        }  
    } else {  
        if (x > meioRaquete) {  
            //Diminuí ângulo:  
            tangenteAngulo = (1 + percentagemVariacao) * tangenteAngulo;  
        } else {  
            //Aumenta ângulo:  
            tangenteAngulo = (1 - percentagemVariacao) * tangenteAngulo;  
        }  
    }  
}
```

## □ Melhoramentos

- Contador de pontos (ex.: paredes tocadas)
- Contador de bolas
- Alteração do tamanho da Bola
- Alteração do tamanho da Raquete
- Alteração da velocidade de Jogo
- *High Scores*
- Criar tipos específicos de Bolas, por herança da classe Bola
- Refinar o algoritmo de modificação do ângulo de "ataque"
- Introduzir blocos ("tijolos") no meio do campo
- Etc.

## Chapter 5 – Graphics with JavaFX Pgs 139 a 150

- ❑ <http://docs.oracle.com/javase/8/javafx/visual-effects-tutorial/animations.htm>
- ❑ <http://docs.oracle.com/javase/8/javafx/api/javafx/animation/package-summary.html>

