7 回目　イベント処理

今日の講義で学ぶ内容

・ウィンドウイベントの利用
・マウスイベントの利用
・スクロールイベントの利用

ウィンドウイベントの利用

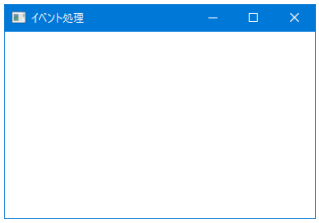
§1　ウィンドウイベント を受け取ってみましょう

ウィンドウイベントは、ウィンドウを開いたり、閉じたりするときに発生します。

ソースファイル名：Sample7_1.java

```
// ※HP よりインポート文をここへ貼り付けてください

// ウィンドウイベントのイベント処理
public class Sample7_1 extends Application
{
   public void start(Stage stage) throws Exception
   {
      // レイアウト VBox を生成／設定します
      VBox vb = new VBox();

      // シーンを生成／設定します
      Scene scene = new Scene(vb);

      // ステージを設定します
      stage.setScene(scene);
      stage.setTitle("イベント処理");

      // イベントハンドラを設定します
      MyEventHandler windowhandler = new MyEventHandler();
      stage.addEventHandler(WindowEvent.ANY, windowhandler);

      // ステージを表示します
      stage.show();
   }

   // イベントハンドラ（イベント処理）クラスの宣言
   private class MyEventHandler implements EventHandler<WindowEvent>
   {
      public void handle(WindowEvent e)
      {
         EventType<WindowEvent> type = e.getEventType();
         if(type == WindowEvent.WINDOW_SHOWING){
            System.out.println("これからウィンドウの表示処理を開始します");
         }
         if(type == WindowEvent.WINDOW_SHOWN){
            System.out.println("これでウィンドウの表示処理を終了します");
         }
```

```
        if(type == WindowEvent.WINDOW_CLOSE_REQUEST){
           System.out.println("終了の要求がありました");
        }
        if(type == WindowEvent.WINDOW_HIDING){
           System.out.println("これからウィンドウを閉じる処理を開始します");
        }
        if(type == WindowEvent.WINDOW_HIDDEN){
           System.out.println("これでウィンドウを閉じる処理を終了します");
        }
     }
  }

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



実行結果

```
これからウィンドウの表示処理を開始します
これでウィンドウの表示処理を終了します
終了の要求がありました
これからウィンドウを閉じる処理を開始します
これでウィンドウを閉じる処理を終了します
```

■ウィンドウイベントとは

ウィンドウイベントは、ウィンドウを開いたり、閉じたりするときに発生するイベントです。これらのイベントが発生したタイミングで、各処理を実行させることができます。

■ウィンドウイベントを表現するクラス WindowEvent

クラス WindowEvent により表現され、以下の種類があります。

- WindowEvent.WINDOW_SHOWING → ウィンドウが表示される直前に発生します
- WindowEvent.WINDOW_SHOWN → ウィンドウが表示された直後に発生します
- WindowEvent.WINDOW_CLOSE_REQUEST → ウィンドウを閉じる命令があったとき発生します
- WindowEvent.WINDOW_HIDING → ウィンドウが閉じられる直前に発生します
- WindowEvent.WINDOW_HIDDEN → ウィンドウが閉じられた直後に発生します

これらのイベントは次の順番で発生します。

この他、すべてのイベントを表現するイベントがあります。実際に発生するイベントではなく、すべてのイベントを受け取りたいときに利用します。

- WindowEvent.ANY → 上記すべてのイベントを表現します

■ウィンドウイベントを処理するイベントハンドラインタフェース EventHandler<WindowEvent>

ウィンドウイベントはイベントハンドラクラスで受け取り、対応する処理を行います。

1. EventHandler<WindowEvent>インタフェースを実装してイベントハンドラクラスを宣言

2. 継承される void handle(WindowEvent e);メソッドをオーバーライドして処理を記述

※発生したイベントがメソッドの引数 e に渡されて呼び出されます

〔コード例〕

```
1. class MyEventHandler implements EventHandler<WindowEvent>{
2.    public void handle(WindowEvent e)
3.    {
4.       // ここにイベントに対応する処理を記述します
5.    }
6. }
```

■インナークラスとは

クラスの中に宣言されるクラスを**インナークラス**といいます。外側のクラスを**アウタークラス**といいます。アウタークラス内でのみインナークラス型の変数の宣言やそのオブジェクトの生成ができます。

```
class Outer{
   void func(int i){
      Inner in = new Inner();
      in.a = i;
   }
   class Inner{
      int a;
   }
}
```

また、インナークラスはアウタークラスのメンバーです。従って、インナークラス内でアウタークラスのフィールドを参照したり、メソッドを実行したりすることができます。

```
class Outer{
   int b;
   class Inner{
      void func(int n){
         b = n;
      }
   }
}
```

■ステージへイベントハンドラを登録

GUI 部品やシーン、ステージは様々なイベントを発生します。これらを受け止めるためにイベントハンドラをそれぞれに登録する必要があります。ウィンドウイベントはウィンドウを表現するステージで受け取ることができます。ステージにイベントハンドラを登録します。

〔コード例〕

```
1. MyEventHandler wh = new MyEventHandler();
2. stage.addEventHandler(WindowEvent.ANY, wh);
```

※オブジェクト wh をイベントハンドラとして Stage クラスのオブジェクト stage に登録します

■利用したクラス／インタフェースの一覧

WindowEvent クラス←Event クラス←EventObject クラス←Object クラス

WindowEvent.WINDOW_SHOWING	ウィンドウ表示の直前に発生するイベントです。
EventType<WindowEvent> getEventType(){…}	イベントの種類を取得します。

EventHandler<WindowEvent>インタフェース←EventListener インタフェース

void handle(WindowEvent e);	イベントが発生したときに実行されます。

Stage クラス←Window クラス←Object クラス

void addEventHandler(EventType<WindowEvent> e, EventHandler<WindowEvent> h){…}	イベント e を受け取るハンドラ h を登録します。

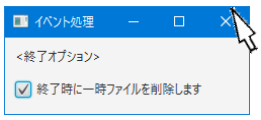
§2　ウィンドウイベントで終了処理をしてみましょう

ウィンドウが閉じるときのイベントを利用して、終了処理を実行することができます。

ソースファイル名：Sample7_2.java

```
// ※HP よりインポート文をここへ貼り付けてください

// ウィンドウイベントの使用例
public class Sample7_2 extends Application
{
   private CheckBox cb;

   public void start(Stage stage) throws Exception
   {
      // ラベル／チェックボックスを生成／設定します
      Label lb = new Label("<終了オプション>");
      cb = new CheckBox("終了時に一時ファイルを削除します");

      // レイアウト VBox を生成／設定します
      VBox vb = new VBox();
      ObservableList<Node> lst = vb.getChildren();
      lst.add(lb);
      lst.add(cb);
      vb.setPadding(new Insets(10));
      vb.setSpacing(15);

      // シーンを生成／設定します
      Scene scene = new Scene(vb);

      // ステージを設定します
      stage.setScene(scene);
      stage.setTitle("イベント処理");

      // イベントハンドラを設定します
      MyEventHandler windowhandler = new MyEventHandler();
      stage.addEventHandler(WindowEvent.ANY, windowhandler);

      // ステージを表示します
      stage.show();
   }

   // イベントハンドラ（イベント処理）クラスの宣言
   private class MyEventHandler implements EventHandler<WindowEvent>
   {
      public void handle(WindowEvent e)
      {
         EventType<WindowEvent> type = e.getEventType();
         if(type == WindowEvent.WINDOW_CLOSE_REQUEST)
         {
            if(cb.isSelected()==true)
                System.out.println("一時ファイルを削除しました");
```

```
        }
      }
   }

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



実行結果

```
一時ファイルを削除しました
```

■イベントハンドラから GUI 部品へのアクセス

イベントが発生したら、GUI 部品の状態を読み取ったり、GUI 部品の状態を変更したりすることがよくあります。たとえば、チェックボックスのチェック状態を読み取ったり、ボタンを無効状態にしたりすることがあります。

start()メソッド内で宣言したボタンなどの GUI 部品を格納している変数は、このメソッドが終了すると変数のスコープを外れるため、その変数は消えてしまいます。イベントハンドラが実行されたときには存在しないことになり、これらの GUI 部品へのアクセスができません。そこで、イベントハンドラからアクセスしたい変数は、クラスのメンバとして宣言しておきます。

例題では、イベントハンドラからクラス CheckBox 型の変数にアクセスするため、これをクラスのメンバとして宣言しています。

```
private CheckBox cb;
```

■チェックボックスの状態を確認するには

クラス CheckBox により管理され、チェック状態を取得するメソッドが準備されています。

・チェック状態の取得　→ isSelected();
　・戻り値が true であればチェックされています。
　・戻り値が false でればチェックされていません。

■利用したクラスの一覧

CheckBox クラス←ButtonBase クラス←Labeled クラス←Control クラス←Region クラス←Parent クラス←Node クラス←Object クラス

boolean isSelected(){…}　チェック状態を返します。

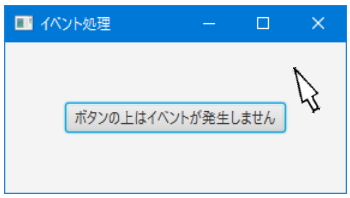
§3　マウスイベントを受け取ってみましょう

マウスを動かしたり、クリックしたりするときに、マウスイベントが発生します。

ソースファイル名：Sample7_3.java

```
// ※HP よりインポート文をここへ貼り付けてください

// マウスイベントのイベント処理
public class Sample7_3 extends Application
{
   public void start(Stage stage) throws Exception
   {
      // ボタンを生成／設定します
      Button bt = new Button("ボタンの上はイベントが発生しません");

      // レイアウト VBox を生成／設定します
      VBox vb = new VBox();
      ObservableList<Node> lst = vb.getChildren();
      lst.add(bt);
      vb.setPadding(new Insets(50));
      vb.setSpacing(15);

      // シーンを生成／設定します
      Scene scene = new Scene(vb);

      // イベントハンドラを設定します
      MyEventHandler mousehandler = new MyEventHandler();
      scene.addEventHandler(MouseEvent.ANY, mousehandler);

      // ステージを設定します
      stage.setScene(scene);
      stage.setTitle("イベント処理");

      // ステージを表示します
      stage.show();
   }

   // イベントハンドラ（イベント処理）クラスの宣言
   private class MyEventHandler implements EventHandler<MouseEvent>
   {
      public void handle(MouseEvent e)
      {
         EventType<? extends MouseEvent> type = e.getEventType();
         if(type == MouseEvent.MOUSE_MOVED){
            System.out.println("マウスが動きました");
         }
         if(type ==  MouseEvent.MOUSE_PRESSED){
            System.out.println("マウスボタンが押されました");
         }
```

```
        if(type ==  MouseEvent.MOUSE_RELEASED){
          System.out.println("マウスボタンが離されました");
        }
        if(type ==  MouseEvent.MOUSE_CLICKED){
          System.out.println("マウスがクリックされました");
        }
        if(type ==  MouseEvent.MOUSE_DRAGGED){
          System.out.println("マウスがドラッグされました");
        }
        if(type ==  MouseEvent.DRAG_DETECTED){
          System.out.println("マウスのドラッグを検出しました");
        }
      }
    }

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



実行結果

```
マウスが動きました
マウスが動きました
マウスボタンが押されました
マウスボタンが離されました
マウスがクリックされました
        :
```

■マウスイベントとは

マウスイベントは、マウスカーソルを動かしたり、マウスボタンをクリックしたりするときに発生するイベントです。これらのイベントが発生したタイミングで、各処理を実行させることができます。

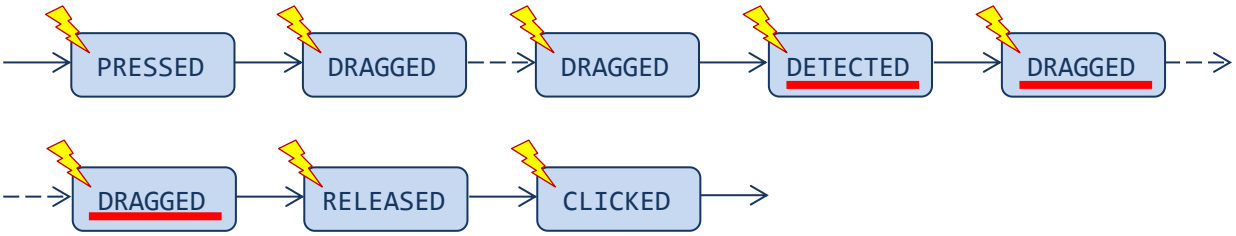
■マウスイベントを表現するクラス MouseEvent

クラス MouseEvent により表現され、以下の種類があります。

- MouseEvent.MOUSE_MOVED → マウスカーソルが動いたときに発生します
- MouseEvent.MOUSE_PRESSED → マウスボタンが押されたときに発生します
- MouseEvent.MOUSE_RELEASED → マウスボタンが離されたときに発生します
- MouseEvent.MOUSE_CLICKED → マウスボタンがクリックされたときに発生します
- MouseEvent.MOUSE_DRAGGED → マウスボタンを押しながらマウスカーソルが動いたときに発生します
- MouseEvent.DRAG_DETECTED → ドラッグジェスチャと認識されたときに発生します

これらのイベントは次の順番で発生します。
□クリックした場合

□ドラッグした場合



この他、すべてのイベントを表現するイベントがあります。実際に発生するイベントではなく、すべてのイベントを受け取りたいときに利用します。

・MouseEvent.ANY　　→ 上記すべてのイベントを表現します

■マウスイベントを処理するイベントハンドラインタフェース EventHandler<MouseEvent>

マウスイベントはイベントハンドラクラスで受け取り、対応する処理を行います。

1. EventHandler<MouseEvent>インタフェースを実装してイベントハンドラクラスを宣言

2. 継承される void handle(MouseEvent e);メソッドをオーバーライドして処理を記述

※発生したイベントがメソッドの引数 e に渡されて呼び出されます

イベントハンドラインタフェースは、宣言に引数を用いるジェネリクスを利用しています。引数に対応するイベントクラスを指定することで、該当のイベントを処理するイベントハンドラを作成できます。

〔コード例〕

```
1. class MyEventHandler implements EventHandler<MouseEvent>{
2.    public void handle(MouseEvent e)
3.    {
4.       // ここにイベントに対応する処理を記述します
5.    }
6. }
```

■シーンへイベントハンドラを登録

GUI 部品やシーン、ステージは様々なイベントを発生します。これらを受け止めるためにイベントハンドラをそれぞれに登録する必要があります。マウスイベントはクライアント領域を表現するシーンで受け取ることができます。シーンにイベントハンドラを登録します。

〔コード例〕

```
1. MyEventHandler wh = new MyEventHandler();
2. scene.addEventHandler(MouseEvent.ANY, wh);
```

※オブジェクト wh をイベントハンドラとして Scene クラスのオブジェクト scene に登録します

■マウスイベントの透過性

ボタンの上でマウスカーソルを動かすまたマウスボタンをクリックしてもマウスイベントを受け取ることができません。これをイベントの透過性といいます。ボタンはマウスイベントの透過性がありません。

一方、レイアウトはマウスイベントの透過性があります。従って、例題のようにレイアウトの下に配置してあるシーンでマウスイベントを受け取ることができるのです。

□マウスイベントを透過しないもの

ボタン（Button）、チェックボックス（CheckBox）…

□マウスイベントを透過するもの

レイアウト（FlowPane, HBox…）、ラベル（Label）、イメージビュー（ImageView）…

■利用したクラス／インタフェースの一覧

MouseEvent クラス←InputEvent クラス←Event クラス←EventObject クラス←Object クラス

`MouseEvent.MOUSE_MOVED` マウスカーソルが動いたら発生するイベントです。

`EventType<? extends MouseEvent> getEventType(){…}`
イベントの種類を取得します。

※ `<? extends MouseEvent>` はジェネリクスの指定の 1 つです。この場合「MouseEvent クラスをスーパークラスにもつクラスを引数とする EventType クラス」を意味します。

EventHandler<MouseEvent>インタフェース←EventListener インタフェース

`void handle(MouseEvent e);` イベントが発生したときに実行されます。

Scene クラス←Object クラス

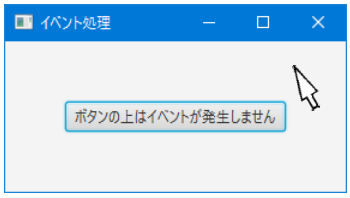
`void addEventHandler(EventType<MouseEvent> e, EventHandler<MouseEvent> h){…}`
イベント e を受け取るハンドラ h を登録します。

§4　マウスのクリック座標を取得してみましょう

マウスボタンをクリックすると、クリックしたボタンや座標などの詳細情報を受け取ることができます、

ソースファイル名：Sample7_4.java

```
// ※HP よりインポート文をここへ貼り付けてください

// クリックボタンとクリック座標
public class Sample7_4 extends Application
{
   public void start(Stage stage) throws Exception
   {
      // ボタンを生成／設定します
      Button bt = new Button("ボタンの上はイベントが発生しません");

      // レイアウト VBox を生成／設定します
      VBox vb = new VBox();
      ObservableList<Node> lst = vb.getChildren();
      lst.add(bt);
      vb.setPadding(new Insets(50));
      vb.setSpacing(15);

      // シーンを生成／設定します
      Scene scene = new Scene(vb);

      // イベントハンドラを設定します
      MyEventHandler mousehandler = new MyEventHandler();
      scene.addEventHandler(MouseEvent.ANY, mousehandler);

      // ステージを設定します
      stage.setScene(scene);
      stage.setTitle("イベント処理");

      // ステージを表示します
      stage.show();
   }

   // イベントハンドラ（イベント処理）クラスの宣言
   private class MyEventHandler implements EventHandler<MouseEvent>
   {
      public void handle(MouseEvent e)
      {
         double mx = e.getX();
         double my = e.getY();
         boolean pb = e.isPrimaryButtonDown();
         boolean mb = e.isMiddleButtonDown();
         boolean sb = e.isSecondaryButtonDown();
         System.out.println("("+mx+","+my+") "+pb+"/"+mb+"/"+sb);
      }
   }
```



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

実行結果

```
(54.0,7.0) false/false/false
(54.0,7.0) true/false/false
(57.0,8.0) true/false/false
(58.0,10.0) true/false/false
(58.0,10.0) false/false/false
(58.0,10.0) false/false/true
(56.0,10.0) false/false/false
(56.0,10.0) false/true/false
(56.0,10.0) false/false/false
          :
```

■クリックボタンやその座標を取得するには

イベントハンドラが呼び出されるとき、マウスイベントの詳細情報が MouseEvent クラスのオブジェクトに格納されて、void handle(MouseEvent e){…}メソッドの引数 e に渡されます。MouseEvent クラスにはこれらの情報を取り出すメソッドが準備されています。

- シーン上のマウスカーソルの座標（X 軸）　→ getX();
- シーン上のマウスカーソルの座標（Y 軸）　→ getY();
- 左ボタンの押下状態（true/false）　→ isPrimaryButtonDown();
- 中ボタンの押下状態（true/false）　→ isMiddleButtonDown();
- 右ボタンの押下状態（true/false）　→ isSecondaryButtonDown();

■利用したクラスの一覧

MouseEvent クラス←InputEvent クラス←Event クラス←EventObject クラス←Object クラス

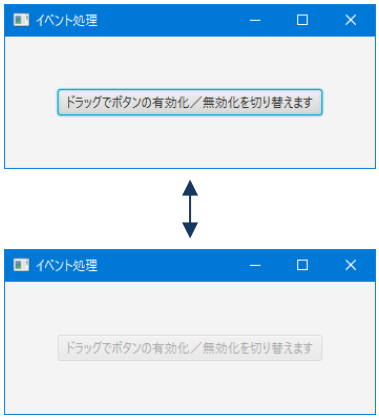
double getX(){…}	マウスカーソルのシーン座標（X 軸）を取得します。
double getY(){…}	マウスカーソルのシーン座標（Y 軸）を取得します。
boolean isPrimaryButtonDown(){…}	左ボタンが押されているとき true を返します。
boolean isMiddleButtonDown(){…}	中ボタンが押されているとき true を返します。
boolean isSecondaryButtonDown(){…}	右ボタンが押されているとき true を返します。

§5　ドラッグジェスチャを使ってみましょう

マウスのドラッグジェスチャで、ボタンの有効／無効を切り替えます。

ソースファイル名：Sample7_5.java

```
// ※HP よりインポート文をここへ貼り付けてください

// マウスイベントの使用例
public class Sample7_5 extends Application
{
   private Button bt;

   public void start(Stage stage) throws Exception
   {
      // ボタンを生成／設定します
      bt = new Button("ドラッグでボタンの有効化／無効化を切り替えます");
      bt.setDisable(true);

      // レイアウト VBox を生成／設定します
      VBox vb = new VBox();
      ObservableList<Node> lst = vb.getChildren();
      lst.add(bt);
      vb.setPadding(new Insets(50));
      vb.setSpacing(15);

      // シーンを生成／設定します
      Scene scene = new Scene(vb);

      // イベントハンドラを設定します
      MyEventHandler mousehandler = new MyEventHandler();
      scene.addEventHandler(MouseEvent.ANY, mousehandler);

      // ステージを設定します
      stage.setScene(scene);
      stage.setTitle("イベント処理");

      // ステージを表示します
      stage.show();
   }

   // イベントハンドラ（イベント処理）クラスの宣言
   private class MyEventHandler implements EventHandler<MouseEvent>
   {
      public void handle(MouseEvent e)
      {
         EventType<? extends MouseEvent> type = e.getEventType();
         if(type ==  MouseEvent.DRAG_DETECTED){
            if(bt.isDisable())
                 bt.setDisable(false);
            else
                 bt.setDisable(true);
```



```
        }
      }
   }

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

■ボタンの有効／無効を切り替えるには

Button クラスにボタンの有効／無効を設定する、その状態を読み出すメソッドが準備されています。

・ボタンを有効に設定　→ setDisable(false);
・ボタンを無効に設定　→ setDisable(true);
・ボタンの有効／無効の取得（true/false）　→ isDisable();

□有効の状態

□無効の状態

■利用したクラスの一覧

Button クラス←ButtonBase クラス←Labeled クラス←Control クラス←Region クラス←Parent クラス←Node クラス←Object クラス

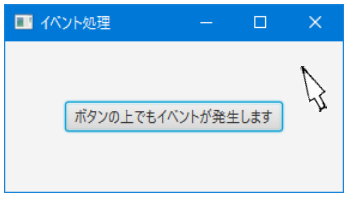
void setDisable(boolean v){…}	ボタンの状態を設定します。 ・引数が true のとき、無効にします。 ・引数が false のとき、有効にします。
void isDisable(){…}	ボタンの有効（false）／無効（true）を取得します。

§6　スクールイベントを受け取ってみましょう

マウスのホイールを回すと、スクロールイベントが発生します。

ソースファイル名：Sample7_6.java

```
// ※HP よりインポート文をここへ貼り付けてください

// スクロールイベントのイベント処理
public class Sample7_6 extends Application
{
   public void start(Stage stage) throws Exception
   {
      // ボタンを生成／設定します
      Button bt = new Button("ボタンの上でもイベントが発生します");

      // レイアウト VBox を生成／設定します
      VBox vb = new VBox();
      ObservableList<Node> lst = vb.getChildren();
      lst.add(bt);
      vb.setPadding(new Insets(50));
      vb.setSpacing(15);

      // シーンを生成／設定します
      Scene scene = new Scene(vb);

      // イベントハンドラを設定します
      MyEventHandler scrollhandler = new MyEventHandler();
      scene.addEventHandler(ScrollEvent.ANY, scrollhandler);

      // ステージを設定します
      stage.setScene(scene);
      stage.setTitle("イベント処理");

      // ステージを表示します
      stage.show();
   }

   // イベントハンドラ（イベント処理）クラスの宣言
   private class MyEventHandler implements EventHandler<ScrollEvent>
   {
      public void handle(ScrollEvent e)
      {
         EventType<ScrollEvent> type = e.getEventType();
         if(type == ScrollEvent.SCROLL){
            System.out.println("スクロールしています");
         }
         if(type == ScrollEvent.SCROLL_STARTED){
            System.out.println("スクロールジェスチャが始まりました");
         }
```



```
        if(type == ScrollEvent.SCROLL_FINISHED){
          System.out.println("スクロールジェスチャが終わりました");
        }
      }
    }

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

実行結果

```
スクロールしています
スクロールしています
スクロールしています
スクロールしています
スクロールしています
        :
```

■スクロールイベントとは

スクロールイベントはマウスのホイールを回したり、タッチスクリーンを指でドラッグしたりすると発生するイベントです。これらのイベントが発生したタイミングで、各処理を実行させることができます。

■スクロールイベントを表現するクラス ScrollEvent

クラス ScrollEvent により表現され、以下の種類があります。

- ScrollEvent.SCROLL → マウスホイールを回したときや、タッチスクリーン上で指をドラッグしたときに発生します
- ScrollEvent.SCROLL_STARTED → スクロールジェスチャを認識したときに発生します
- ScrollEvent.SCROLL_FINISHED → スクロールジェスチャが終了したときに発生します

※マウスホイールで発生するイベントは、ScrollEvent.SCROLL のみです。

この他、すべてのイベントを表現するイベントがあります。実際に発生するイベントではなく、すべてのイベントを受け取りたいときに利用します。

- ScrollEvent.ANY → 上記すべてのイベントを表現します

■スクロールイベントを処理するイベントハンドラインタフェース EventHandler<ScrollEvent>

スクロールイベントはイベントハンドラクラスで受け取り、対応する処理を行います。

1. EventHandler<ScrollEvent>インタフェースを実装してイベントハンドラクラスを宣言

2. 継承される void handle(ScrollEvent e);メソッドをオーバーライドして処理を記述

※発生したイベントがメソッドの引数 e に渡されて呼び出されます

イベントハンドラインタフェースは、宣言に引数を用いるジェネリクスを利用しています。引数に対応するイベントクラスを指定することで、該当のイベントを処理するイベントハンドラを作成できます。

〔コード例〕

```
1. class MyEventHandler implements EventHandler<ScrollEvent>{
2.   public void handle(ScrollEvent e)
3.   {
4.     // ここにイベントに対応する処理を記述します
5.   }
6. }
```

■シーンへイベントハンドラを登録

GUI 部品やシーン、ステージは様々なイベントを発生します。これらを受け止めるためにイベントハンドラをそれぞれに登録する必要があります。スクロールイベントはマウスイベントと同様にクライアント領域を表現するシーンで受け取ることができます。シーンにイベントハンドラを登録します。

〔コード例〕

```
1. MyEventHandler wh = new MyEventHandler();
2. scene.addEventHandler(ScrollEvent.ANY, wh);
```

※オブジェクト wh をイベントハンドラとして Scene クラスのオブジェクト scene に登録します

■スクロールイベントの透過性

スクロールイベントは比較的に広い透過性があります。

※クラス宣言の説明に、透過性に関する明確な記述が見当たりませんでしたので詳細は分かりません。

■利用したクラス／インタフェースの一覧

ScrollEvent クラス←GestureEvent クラス←InputEvent クラス←Event クラス←EventObject クラス←Object クラス	
ScrollEvent.SCROLL	マウスホイールを回す、タッチスクリーンを指でドラッグするときに発生するイベントです。
EventType<ScrollEvent> getEventType(){…}	イベントの種類を取得します。
EventHandler<ScrollEvent>インタフェース←EventListener インタフェース	
void handle(ScrollEvent e);	イベントが発生したときに実行されます。

§7　マウスホイールの回転方向を取得してみましょう。

マウスホイールの上下の回転方向や、左右の傾き方向などの詳細情報を受け取ることができます。

ソースファイル名：Sample7_7.java

```
// ※HP よりインポート文をここへ貼り付けてください

// スクロール（ホイール）の回転方向
public class Sample7_7 extends Application
{
   public void start(Stage stage) throws Exception
   {
      // ボタンを生成／設定します
      Button bt = new Button("ボタンの上でもイベントが発生します");

      // レイアウト VBox を生成／設定します
      VBox vb = new VBox();
      ObservableList<Node> lst = vb.getChildren();
      lst.add(bt);
      vb.setPadding(new Insets(50));
      vb.setSpacing(15);

      // シーンを生成／設定します
      Scene scene = new Scene(vb);

      // イベントハンドラを設定します
      MyEventHandler scrollhandler = new MyEventHandler();
      scene.addEventHandler(ScrollEvent.ANY, scrollhandler);

      // ステージを設定します
      stage.setScene(scene);
      stage.setTitle("イベント処理");

      // ステージを表示します
      stage.show();
   }

   // イベントハンドラ（イベント処理）クラスの宣言
   private class MyEventHandler implements EventHandler<ScrollEvent>
   {
      public void handle(ScrollEvent e)
      {
         EventType<ScrollEvent> type = e.getEventType();
         if(type == ScrollEvent.SCROLL){
            double sx = e.getDeltaX();
            double sy = e.getDeltaY();

            if(sx > 0) System.out.println("左スクロール");
            else if(sx < 0) System.out.println("右スクロール");

            if(sy > 0) System.out.println("上スクロール");
```

```
            else if(sy < 0) System.out.println("下スクロール");
        }
    }
  }

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

実行結果

```
下スクロール
上スクロール
上スクロール
下スクロール
下スクロール
   :
```

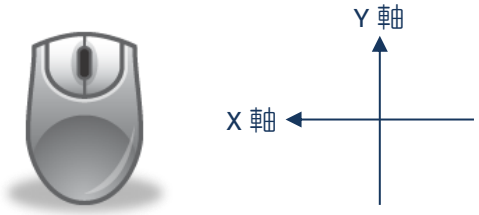
■マウスホイールの回転方向や左右の傾きを取得するには

イベントハンドラが呼び出されるとき、スクロールイベントの詳細情報が ScrollEvent クラスのオブジェクトに格納されて、void handle(ScrollEvent e){…}メソッドの引数 e に渡されます。ScrollEvent クラスにはこれらの情報を取り出すメソッドが準備されています。

・水平方向のスクロール量（ピクセル）　→ getDeltaX();
　左スクロールが正で、右スクロールが負です

・垂直方向のスクロール量（ピクセル）　→ getDeltaY();
　上スクロールが正で、下スクロールが負です

座標系は以下のとおりです。



■利用したクラスの一覧

ScrollEvent クラス←GestureEvent クラス←InputEvent クラス←Event クラス←EventObject クラス←Object クラス

double getDeltaX(){…}	マウスホイールの水平スクロール量（ピクセル）を取得します。
double getDeltaY(){…}	マウスホイールの垂直スクロール量（ピクセル）を取得します。