

■課題 1 クリスマスカードを作成しましょう。実行例に示すようにキャンバスの中央にメッセージ「Merry Xmas」の文字列を描画します。さらに、キャンバス上をマウスでクリックすると、クリック座標を中心にして雪の結晶画像が表示されます。

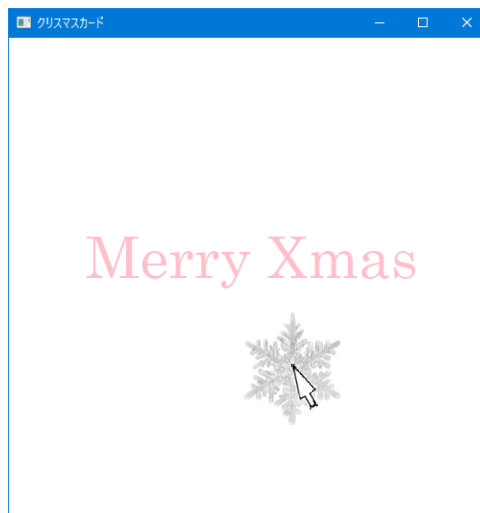
その他の設定は次のとおりです。参考にしましょう。

- キャンバスサイズ → 横 500 ピクセル 縦 500 ピクセル
- メッセージのフォント → "Century" の 60 ポイント [new Font("Century",60);]
- メッセージの座標 → (80,250)
- メッセージの色 → ピンク [Color.PINK]
- ウィンドウのタイトル → クリスマスカード

ヒント：Image クラスのオブジェクト（画像）のサイズは、getWidth()と getHeight()で取得できます。これらを用いてクリック位置に画像の中心が描画されるようにしましょう。

※おおよそ実行例のような画面になれば OK です

〔実行例〕



■課題 2 グリーティングカードアプリを作成しましょう。実行例のように GUI 部品を配置します。ラジオボタンで描画色を選択し、キャンバス上でマウスをドラッグすると指定した色で描画されます。下にあるクリアボタンを押すとキャンバスがクリアされます。また保存ボタンを押すと現在のキャンバス画像が画像ファイルに保存されます。

その他の設定は次のとおりです。参考にしましょう。

HBox レイアウト周りの空白エリア → 10 ピクセル [setPadding(new Insets(10));]
HBox-GUI 部品間の空白エリア → 10 ピクセル [setSpacing(10);]
シーンの色 → オレンジ [Color.ORANGE]
※レイアウト VBox の背景色を透明にします
レイアウト VBox の setBackground(null); を実行
ウィンドウのタイトル → グリーティングカード

ヒント 1: マウスのドラッグイベントが発生したらそのマウス位置に直径 10 ピクセルの円を描画します。毎回描画するとき、キャンバスをクリアしなければ前回描画した円は消えずに残ります。ドラッグイベントが発生する度に円を描画して軌跡を残していきます。

ヒント 2: クリアボタンが押されたときのみ、キャンバスをクリアします。

ヒント 3: 保存ボタンが押されたら、次のコードを実行してファイルに画像を保存します。

〔ファイル CanvasImage.png に画像を保存するコード〕

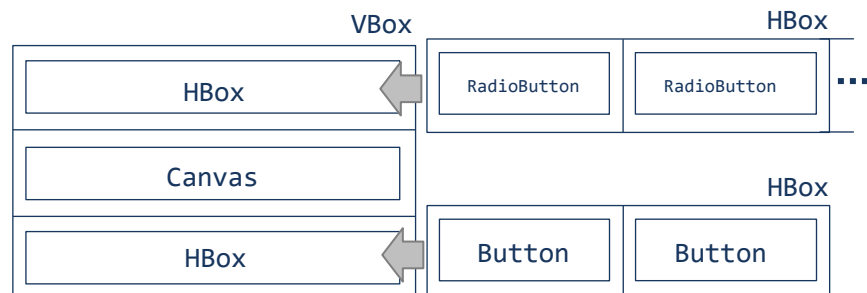
```
WritableImage bi = new WritableImage((int)cv.getWidth(), (int)cv.getHeight());
cv.snapshot(null, bi);
File file = new File("CanvasImage.png");
try{
    ImageIO.write(SwingFXUtils.fromFXImage(bi, null), "png", file);
}catch(Exception ee){}
```

※変数 `cv` はクラス `Canvas` 型の変数です。このキャンバスが画像ファイルに保存されます。
※画像ファイルへの保存を行う各クラスを利用するには次の `import` 文を加えましょう。

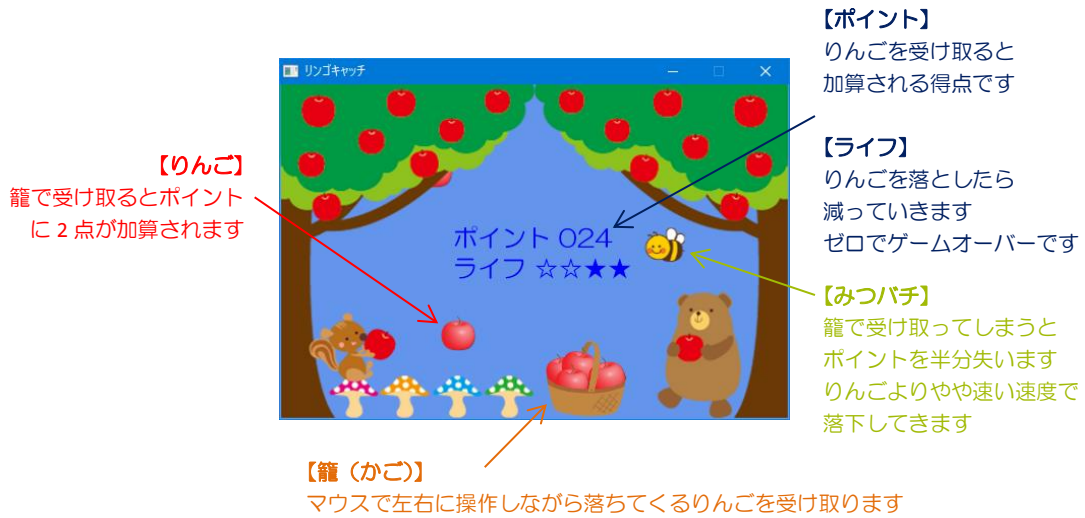
```
import javafx.embed.swing.*;
import javax.imageio.*;
import java.io.*;
```

※画像は HP よりダウンロードしてソースファイルと同じフォルダに保存しておきましょう
※おおよそ実行例のような画面になれば OK です

〔実行例〕



■課題3 リンゴキャッチゲームを作成しましょう。ゲーム内容は以下の通りです。



〔ゲームの内容〕





りんごが次々とランダムな場所から落ちてきますので、マウスで籠を左右に動かしてタイミングよくキャッチします。りんごをキャッチすると得点として2ポイントが加算されます。たまにみつバチが降ってくる場合があります。間違えてキャッチしてしまうと獲得ポイントの半分を失います。気を付けましょう。さらに、りんごを落としてしまった場合はライフが減らされ、ゼロでゲームオーバーです。みつバチは落としても良いです。

〔画面の設計〕

横 500×縦 330 のキャンバスを生成し、レイアウト VBox に配置します。背景の色はシーンで設定します。キャンバスの描画は下記のように奥側から順番に行うことで奥行を表現します。
描画順番：ポイントとライフ → 籠 → りんごとみつバチ → 森の画像



籠やりんご、森の画像のサイズは表のようにキャンバスの大きさに合わせて調整済みでそのまま利用して大丈夫です。籠の動作は左右の平行移動のみですので、縦位置は固定です。実行例ではY座標 290 で固定しています。またりんごとみつバチの出現率は表の通りです。

籠	りんご	みつバチ	森の画像※白部は透明
 100x100	 出現率 80% 35x35	 出現率 20% 40x35	 500x330

〔ゲームの状態〕

ゲームは3つの状態があります。3つの状態とは**スタート**と**ゲーム中**、**ゲームオーバー**です。**スタート**ではマウスクリックをすると**ゲーム中**に移行し、ゲームが開始されます。ゲーム中にりんごを落としライフが無くなると**ゲームオーバー**に移行しゲームを終了します。



〔籠の変化〕

籠の画像はポイントに関係なく受け取ったリンゴの数に応じて、以下の通り0個から5個の6種類を準備しております。5個以上の場合は5個の画像を用います。



〔りんごとみつバチの落下速度について〕

実行例で設定したりんごとみつバチの落下速度の初期値は以下の通りです。みつバチが少々速い速度で落下します。この値を増加することにより難易度を上げます。

落下物	初期落下速度	※1秒間の描画回数に依存して各描画ごとの進み具合を計算します。
りんご	150 pixel/s	※例えば1秒間に30回の描画する時、33ミリ秒毎の描画となります。 150pixel/sの速度 == 毎描画毎に150*33/1000 pixel ほど進む速度
みつバチ	200 pixel/s	200pixel/sの速度 == 毎描画毎に200*33/1000 pixel ほど進む速度

獲得ポイントが増えると落下速度を上げて難易度を上げます。20ポイント毎に速度を20%加算します。取得ポイント40~59で40%の加算、取得ポイント100~119で100%の加算（開始時の2倍の落下速度）となります。

〔落下中のりんごやみつバチを管理する ArrayList クラス〕

りんごやみつバチは上から落下して、籠に入るかまたは画面外に出るまで描画します。このとき、画面内のりんごやみつバチの位置を保持するために便利なクラスが ArrayList クラスです。ArrayList は与えられたクラスのインスタンスをリスト構造で管理します。新しいインスタンスの追加や指定位置のインスタンスの削除がとても簡単に実行できます。

ArrayList クラスのインスタンス lst



以下のコードは使用例です。

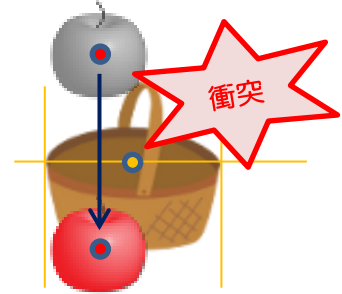
りんごやみつバチの位置情報をクラス Apple(りんごとみつバチはメンバ変数の値で区別) で表現しています。

```
ArrayList<Apple> lst = new ArrayList<Apple>();
lst.add(new Apple()); // 最後尾に新しい Apple インスタンスを追加
Apple ap=lst.get(2); // 添え字 2 の Apple インスタンスの取得
lst.remove(2); // 添え字 2 の Apple インスタンスの削除
lst.size(); // 格納されている Apple インスタンスの数 (上の例では 10)
```

〔衝突判定について〕

りんごやみつばちは籠で受け取ることができます。受け取れたと判断するためにアルゴリズム例を紹介します。衝突の判定に用いるりんごやみつばち、籠の基準点をそれぞれの画像の中心とし、りんごを例に説明します。

赤色のりんごは現在の位置(x_1, y_1)で、灰色のりんごは1つ過去の位置(x_2, y_2)を表します。このように各りんごの位置は現在と1つ過去のものを保持しておきましょう。 y_1 と y_2 の間に籠のY軸値があった場合、衝突した可能性があります。さらに、X軸値を考えます。ここでりんごはまっすぐ落下しますので $x_1 = x_2$ です。 x_1 の値が籠の画像の幅に以内であれば衝突となります。



〔アニメーションの実現〕

これまではマウスやキーボード、ボタンなどからなんらかのイベントが発生したタイミングで処理を実行する方法を学習してきました。ここでは、このようなイベントが発生しない場合でも処理を実行するアニメーションの実現方法を概説します。マウスを動かさなくてもりんごやみつばちが落下してくる処理がこれに該当します。

スレッド (Thread) クラスまたはタイマー (Timer) クラスを用いて並行処理を行います。この並行処理が逐一実行され、アニメーションが実現されます。

- スレッド → 一度だけ並行処理を実行します
- タイマー → 指定間隔で繰り返し並行処理を実行します

※スレッドについて Java プログラミング 1 にて既に学習しております

いずれを用いてもアニメーションを実現できますが、タイマーを用いた方がよりスマートに記述できますので、タイマーの利用方法について紹介します。

手順は Thread の場合と同様です。ここでは 2 つのクラス Timer と TimerTask を用います。

- ① TimerTask クラスのサブクラスを宣言する
- ② 継承される run() メソッドをオーバーライドする

```
class MyTimer extends TimerTask{
    public void run(){
        // ここに指定間隔で実行したい処理を記述します
        // りんごやみつばちの落下による位置の更新
        // 籠との衝突判定 (ポイントをリンゴであれば加算、みつばちであれば半減)
        // 落ちた (画面外に出た) りんごやみつばちの後処理 (ライフの更新や削除)
        // 新たなりんごやみつばちの落下準備 (乱数による位置の決定)
        // 更新された情報をもとに再描画の要求 など
    }
}
```

- ③ Timer クラスのインスタンスを生成する
- ④ MyTimer クラスのインスタンスを schedule() メソッドに渡してタイマーを開始する

```
Timer tm = new Timer();
tm.schedule(new MyTimer(), 0, 33); // 0 ミリ秒後から 33 ミリ秒間隔
```

- ⑤ タイマを止めるには Timer や MyTimer インスタンスの cancel() メソッドを実行する
- ⑥ インポート文に 『import java.util.*;』 を追加します

〔GUI 操作と JavaFX アプリケーションスレッドについて〕

JavaFX アプリケーションを動かす主なスレッドは 2 つあります。

- **Main スレッド** `main()`メソッド
 `launch()`メソッド
- **JavaFX アプリケーションスレッド** `start()`メソッド
 ボタンやチェックボックス、テキストフィールドなど
 GUI 部品のイベントハンドラ

起動処理は **Main スレッド** が実行し、実行中の処理は **JavaFX アプリケーションスレッド** が実行します。どちらのスレッドが実行していかを知るためには `Thread` クラスのクラスメソッド `currentThread()` を用いて次のように画面出力すればわかります。

```
■System.out.println(Thread.currentThread());
```

JavaFX では次の約束があります。

『**GUI 部品へのアクセスは JavaFX アプリケーションスレッドからのみ許可されます**』

ボタンやチェックボックスなどのイベントハンドラは JavaFX アプリケーションスレッドから実行されるので、テキストフィールドの文字列読み込みやプロント文字列設定ができます。

一方、スレッドやタイマーにより起動された並行処理は JavaFX アプリケーションスレッドとは別のスレッドにより実行されます。このため、**並行処理のなかで GUI 部品へアクセスしてはいけません**。 ※具体的には上記 `MyTimer` クラスの `run()` メソッドの中に GUI 部品へアクセスするコードを書く
と実行時に例外が発生します。

この並行処理の中で記述できることは、ゲームの状態を表す変数（例えば、ポイントやライフ、各リングの座標、籠の座標を表す変数）など GUI 部品とは関係のない変数へのアクセスのみです。

しかしながら、再描画（必ず `Canvas` クラスなど GUI 部品へのアクセスが発生）などどうしても実行したい場合には、`Platform` クラスのクラスメソッド `runLater()` を用いて次のように**実行要求**を出せば良いです。

```
■Platform.runLater(Runnable r);
```

`r` `Runnable` インタフェースを実装したクラスのインスタンス

継承した `run()` メソッドをオーバーライドし、中に実行したいコードを記述

実行要求を出した後、JavaFX アプリケーションスレッドにより `run()` が実行されます