```
#include <iostream>
#include <sstream>

#include <Kinect.h>
#include <opencv2¥opencv.hpp>

#include <atlbase.h>

// 次のように使います
// ERROR_CHECK( ::GetDefaultKinectSensor( &kinect ) );
// 書籍での解説のためにマクロにしています。実際には展開した形で使うことを検討してください。
#define ERROR_CHECK( ret )  ¥
    if ( (ret) != S_OK ) {    ¥
        std::stringstream ss;   ¥
        ss << "failed " #ret " " << std::hex << ret << std::endl;           ¥
        throw std::runtime_error( ss.str().c_str() );           ¥
    }

class KinectApp
{
private:

    ComPtr<IKinectSensor> kinect = nullptr;

    ComPtr<IBodyFrameReader> bodyFrameReader = nullptr;
    IBody* bodies[6];

public:

    ~KinectApp()
    {
        // Kinectの動作を終了する
        if ( kinect != nullptr ){
            kinect->Close();
        }
    }

    // 初期化
    void initialize()
    {
        // デフォルトのKinectを取得する
        ERROR_CHECK( ::GetDefaultKinectSensor( &kinect ) );
        ERROR_CHECK( kinect->Open() );

        // ボディリーダーを取得する
        ComPtr<IBodyFrameSource> bodyFrameSource;
        ERROR_CHECK( kinect->get_BodyFrameSource( &bodyFrameSource ) );
        ERROR_CHECK( bodyFrameSource->OpenReader( &bodyFrameReader ) );

        for ( auto& body : bodies ){
            body = nullptr;
        }
    }

    void run()
    {
        while ( 1 ) {
            update();
            draw();

            auto key = cv::waitKey( 10 );
            if ( key == 'q' ){
                break;
            }
        }
    }

private:

    // データの更新処理
    void update()
    {
        updateBodyFrame();
```

```
    }

    // ボディフレームの更新
    void updateBodyFrame()
    {
        // フレームを取得する
        ComPtr<IBodyFrame> bodyFrame;
        auto ret = bodyFrameReader->AcquireLatestFrame( &bodyFrame );
        if ( FAILED( ret ) ){
            return;
        }

        // 前回のBodyを解放する
        for ( auto& body : bodies ){
            if ( body != nullptr ){
                body->Release();
                body = nullptr;
            }
        }

        // データを取得する
        ERROR_CHECK( bodyFrame->GetAndRefreshBodyData( 6, &bodies[0] ) );
    }

    void draw()
    {
        drawBodyIndexFrame();
    }

    void drawBodyIndexFrame()
    {
        // 関節の座標をDepth座標系で表示する
        cv::Mat bodyImage = cv::Mat::zeros( 424, 512, CV_8UC4 );

        for ( auto body : bodies ){
            if ( body == nullptr ){
                continue;
            }

            BOOLEAN isTracked = false;
            ERROR_CHECK( body->get_IsTracked( &isTracked ) );
            if ( !isTracked ) {
                continue;
            }

            // 関節の位置を表示する
            Joint joints[JointType::JointType_Count];
            body->GetJoints( JointType::JointType_Count, joints );
            for ( auto joint : joints ) {
                // 手の位置が追跡状態
                if ( joint.TrackingState == TrackingState::TrackingState_Tracked ) {
                    drawEllipse( bodyImage, joint, 10, cv::Scalar( 255, 0, 0 ) );
                }
                // 手の位置が推測状態
                else if ( joint.TrackingState == TrackingState::TrackingState_Inferred ) {
                    drawEllipse( bodyImage, joint, 10, cv::Scalar( 255, 255, 0 ) );
                }
            }
        }

        cv::imshow( "Body Image", bodyImage );
    }

    void drawEllipse( cv::Mat& bodyImage, const Joint& joint, int r, const cv::Scalar& color )
    {
        // カメラ座標系をDepth座標系に変換する
        ComPtr<ICoordinateMapper> mapper;
        ERROR_CHECK( kinect->get_CoordinateMapper( &mapper ) );

        DepthSpacePoint point;
        mapper->MapCameraPointToDepthSpace( joint.Position, &point );

        cv::circle( bodyImage, cv::Point( point.X, point.Y ), r, color, -1 );
```

```
    }
};

void main()
{
    try {
        KinectApp app;
        app.initialize();
        app.run();
    }
    catch ( std::exception& ex ){
        std::cout << ex.what() << std::endl;
    }
}
```