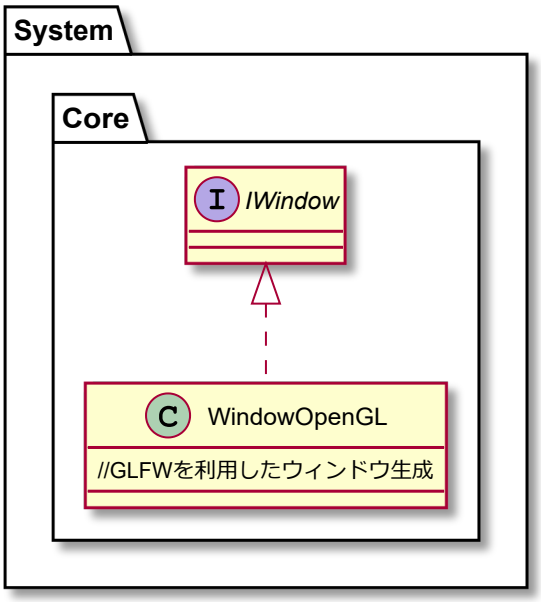
Ulma Game プログラム仕様

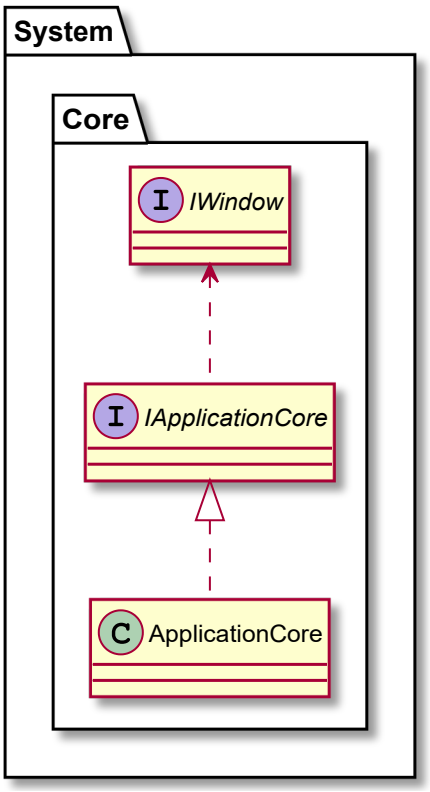
クラス一覧

1. ウィンドウ関連



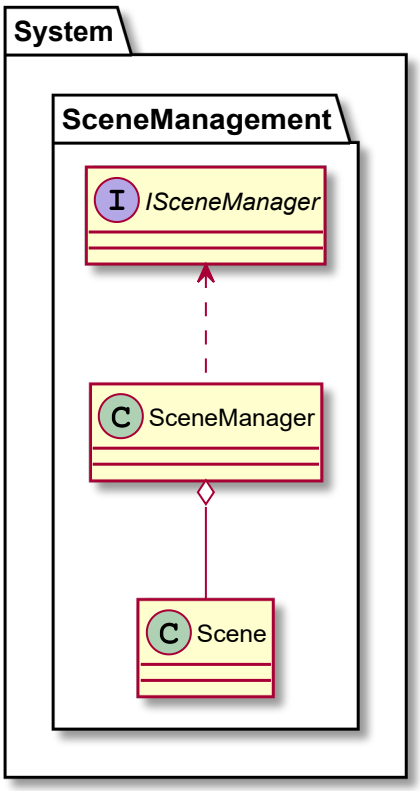

```
//Windowクラス
bool CreateWindow();     //グラフィックスAPIに対応したウィンドウ作成
void ClearDisplayBuffer();  //ディスプレイバッファのクリア
void SwapBuffer();  //ダブルバッファの入れ替え
void Finalize();     //グラフィックスAPIの終了処理（アプリケーションから呼び出される）
void PollEvent();    //イベント処理
bool CanLoop();      //メインループ中か？
float GetCurrentTime(); //経過時間の取得
int GetWindowWidth();    //ウィンドウ幅取得
int GetWindowHeight();  //ウィンドウ縦取得
```

2. アプリケーション




```
//ApplicationCoreクラス
bool Initialize(IWindow& window);    //アプリケーション初期化
void Update();  //メインループでマイフレーム呼ばれる処理
void Finalize();     //アプリケーションの終了処理
```

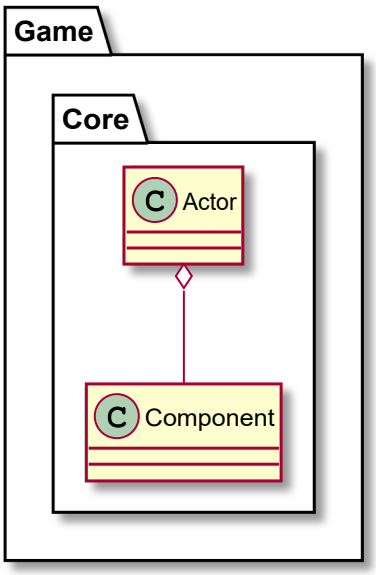
3. シーン関連




```
//SceneManagerクラス（StateMachine）
void UpdateScene(float deltaTime);  //現在のシーンのupdate
void GenerateOutput(ShaderLoaderOpenGL& shader);  //描画処理（シェーダーの渡し方要検討）
void LoadScene(ESceneType type);    //シーンのロード
void AddScene(ESceneType type, Scene& scene);   //シーン追加（引数1:シーン名，引数2:シーン)(引数1要
void RemoveScene(ESceneType type); //シーン削除

//Sceneクラス
void OnEnter{}  //シーンロード時の処理（Actor初期化）
void Update(float deltaTime);   //シーンのupdate
void GenerateOutput(ShaderLoaderOpenGL& shader);    //描画処理
void AddActor(Actor& actor);    //Actor追加
void RemoveActor(Actor& actor); //Actor削除
void AddSprite(SpriteComponent& sprite);    //スプライト追加
```

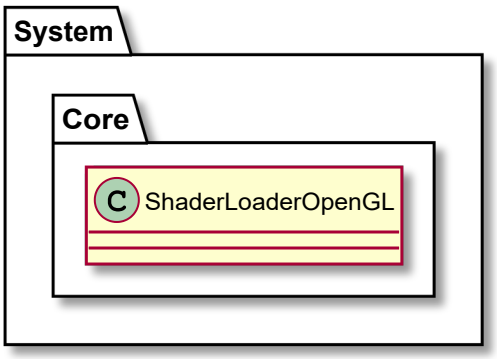
- シーン追加はアプリケーションの初期化で行う

4. アクター、コンポーネント



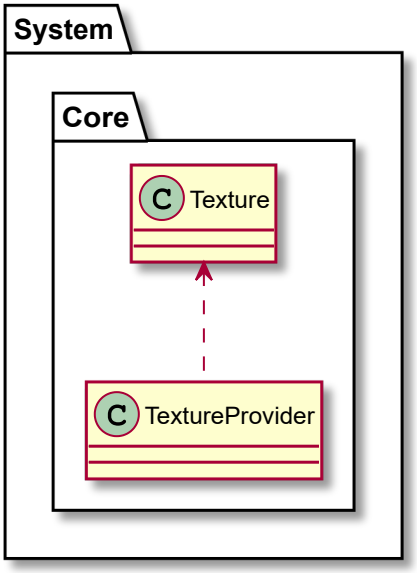

```
//Actorクラス
void Initialize();  //初期化処理（シーンの初期化時呼ばれる）
void Update(float deltaTime);   //Actor全体のupdate（override不可）
void AddComponent(Component& comp); //Component追加
void RemoveComponent(Component& comp);  //Component削除
void UpdateComponents(float deltaTime); //アタッチされているcomponentのupdate
void UpdateActor(float deltaTime);  //Actor自身のupdate（override可）

//Componentクラス
void Update(float deltaTime);   //componentのupdate
```

5. シェーダー



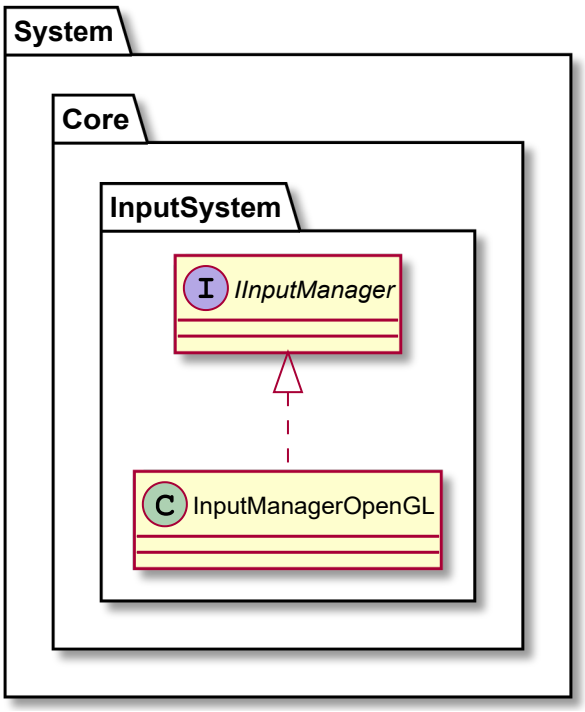

```
//ShaderLoaderOpenGLクラス
bool LoadProgram(const char* vert, const char* frag);   //シェーダーロード
void Activate();    //シェーダーの使用を宣言
void Unload();  //シェーダーのメモリ開放

void SetAttributeVertices(const char* attrib, float vertices[]);    //頂点データの設定
//Uniform変数設定
void SetUniformInt();
void SetUniformFloat();
void SetUniformVec2();
void SetUniformMat4();
GLuint CreateProgram(const char* vert, const char* frag); //プログラムオブジェクト作成
bool ReadShaderSource(const char* name, std::vector<GLchar>& buffer);   //シェーダーソース読み込み
```

6. テクスチャ



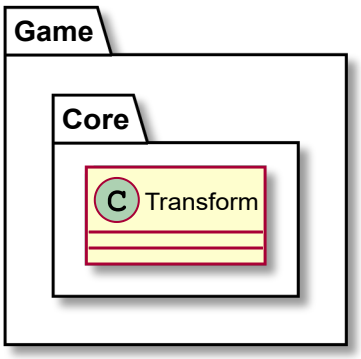

```
//Textureクラス
void Activate();    //テクスチャ有効化
void Inactivate();  //テクスチャ解放
unsigned int GetWidth();    //テクスチャ横幅取得
unsigned int GetHeight();   //テクスチャ縦幅取得
bool CreateTexture();
bool CreateTexture(const char* fileName);   //テクスチャ作成

//TextureProviderクラス
void UseTexture(const std::string& assetName);  //テクスチャの使用宣言
void AddTexture(const std::string& assetName, Texture& texture);    //テクスチャをリストに追加
Texture& GetTexture(const std::string& assetName);  //対象のテクスチャを取得
```

7. 入力




```
//InputManagerクラス
bool GetKey(EKeyCode key);  //キーの入力を取得
bool GetKeyDown(EKeyCode key);  //キーの入力を取得（1度だけ）
bool GetMouseButton(EMouseButton mouse);    //マウスの入力を取得
bool GetMouseButtonDown(EMouseButton mouse);    //マウスの入力を取得（1度だけ）
float GetAxis(EAxisType axis);  //軸の入力を取得
```

8. トランスフォーム




```
//Transformクラス
void ComputeWorldTransform();    //ワールド座標変換を計算
Transform& GetParent();      //親を取得
Matrix4& GetWorldTransform();    //ワールドのTransformを取得
Vector3 GetRight();      //右向きのベクトル取得
Vector3 GetUp();     //上向きのベクトル取得
```