LOOT 기본 설명서

Version 1.8  
racin@hanyang.ac.kr  
최종 갱신: 2016. 5. 9

# 개요

LOOT는 Library Of OOP Term-project의 줄임말이며 LOOT에는 객체지향설계및프로그래밍 수업을 듣는 학생들이 텀 프로젝트를 수행할 때 사용할 수 있는 여러 기능들이 포함되어 있습니다. 이 참조 문서에서는 LOOT에 포함되어 있는 각 요소들에 대한 설명과 함께 해당 요소를 사용할 때 알아 두어야 할 점 및 배경 지식을 서술합니다.

# 문서 구성

이 참조 문서에는 다음과 같은 내용들이 포함되어 있습니다:

* '시작' 단원: 텀 프로젝트를 시작하기 위해 먼저 수행해야 할 사항들을 순서대로 설명합니다.
* '기본' 단원: LOOT를 사용하여 프로그램을 만들기 위한 기본적인 기법들을 설명합니다.

이러한 두 단원은 처음부터 순서대로 진행하는 것을 가정하고 작성되어 있습니다(실습 수업에서도 이들 내용을 순서대로 다룰 예정입니다).

# 문서를 읽을 때 주의해야 할 사항

이 참조 문서를 읽기 전에 다음 내용들을 꼭 먼저 확인하세요:

* 이 문서는 여러분이 Eclipse의 'Ctrl + Space'를 충분히 사용한다는 것을 전제로 작성되었으며 특히 package 관련 부분( import loot.~~ 등 )에 대한 설명은 대부분 생략되어 있습니다. 그러므로, 텀 프로젝트를 시작하기 전에 '시작' 단원의 순서에 따라 LOOT project에 대한 **build path를 확실하게 설정해** 둔 다음 새 클래스를 사용할 때마다 **Ctrl + Space를 눌러** 해당 클래스에 대한 import 구문을 자동으로 추가하게 하세요.
* LOOT가 여러분의 텀 프로젝트에 필요한 많은 기능을 제공하고 있는 것은 사실이지만, LOOT에 있는 각 기능을 프로그램에 어떻게 적용할 것인지 판단하는 것은 LOOT 자체를 이해하는 것과는 또 별개의 문제입니다. 여러분의 프로그램을 만들 때 막히는 부분이 발생하는 경우 **반드시 조교의 도움을 구하도록** 하세요.

# 시작 1 - 새 프로젝트 만들기

LOOT를 사용하면 5분 안에 여러분의 텀 프로젝트에 필요한 기본 골격을 세울 수 있습니다. GameFrame은 Java Swing의 JFrame class(실제 모니터에 띄울 수 있는 '창' 클래스)를 상속받아 게임에 필요한 여러 핵심 요소들을 추가로 정의합니다.

텀 프로젝트 시작을 위한 과정은 다음과 같습니다:

* 먼저 최신 버전의 LOOT project를 다운로드하여 Eclipse에 import하세요.  
  필요한 경우 import하기 전에 이미 Eclipse에 있는 LOOT project를 '완전 제거'하세요.
* 여러분의 텀 프로젝트를 만들기 위한 새 Java project를 만드세요.
* 새 Java project가 LOOT project를 참조하도록 build path를 설정하세요.
  + 새 Java project를 우클릭한 다음 Build Path 🡪 Configure Build Path… 메뉴를 클릭하세요.
  + Projects 탭을 선택하고 Add… 버튼을 누른 다음 목록에 표시된 LOOT project를 선택하세요.
  + OK OK를 누르면 build path 설정이 완료됩니다.
* 새 Java project에 GameFrame을 상속받아 게임을 구성할 새로운 클래스를 만드세요.
  + 새 클래스 창에서 Superclass를 loot.GameFrame으로 지정하세요.
  + 아래에 있는 체크박스 중 'Constructors from superclass'와 'Inherited abstract methods'를 체크하세요.
  + 이외에는 평소 새 클래스를 만들던 방법과 동일한 과정을 수행하면 됩니다.
  + 생성된 .java 파일에 총 네 개의 메서드가 미리 정의되어 있다면 성공입니다.
* 새 Java project에 main()을 정의할 새로운 클래스를 만드세요.
  + 이 작업은 평소 main()이 들어 있는 클래스를 만들던 방법과 동일한 과정을 수행하면 됩니다.
* main()에 아래의 코드를 추가하세요:

GameFrameSettings settings = new GameFrameSettings();

/\* 여기서 settings. 을 입력하여 게임 화면 관련 설정 가능 \*/

SampleFrame window = new SampleFrame(settings); //SampleFrame 대신 여러분이 만든 클래스 이름 넣기

window.setVisible(true);

* 이제 Ctrl + F11을 눌러 실행하면 모니터에 게임 화면이 표시됩니다!
  + 표시되는 게임 화면은 GameFrameSettings에 있는 기본값에 맞추어져 있으며  
    다음 페이지에서 설명할 '게임 화면 설정'을 통해 여러분의 텀 프로젝트에 적합한 게임 화면을 만들 수 있게 됩니다.

# 시작 2 - 게임 화면 설정하기

GameFrameSettings에는 게임 화면 설정을 위한 필드가 여럿 들어 있으며 새 게임 화면을 만들기 전에 이들의 값을 재설정함으로써 여러분의 텀 프로젝트에 적합한 게임 화면을 만들 수 있습니다.

GameFrameSettings에는 다음과 같은 필드들이 포함되어 있습니다:

* String window\_title
  + 초기 창 제목입니다. 실행 도중 창 제목을 바꾸려면 setTitle()을 사용하면 됩니다.
  + 기본값은 "개발중!"입니다.
* int canvas\_width / int canvas\_height
  + 게임 화면의 가로 / 세로 길이(픽셀 수)입니다.
  + 기본값은 800 / 600입니다.
* java.awt.Color canvas\_backgroundColor
  + 게임 화면의 배경색입니다.
  + 기본값은 Color.WHITE입니다. Color class에는 흰 색 외에도 자주 사용되는 몇몇 색이 정의되어 있습니다.

위의 세 필드를 적절히 바꾸면 여러분이 만들 게임 화면에 대한 기본적인 설정이 끝납니다.

이외에도 GameFrameSettings에는 게임 루프 설정, 입력 설정에 필요한 필드들이 추가로 정의되어 있습니다. 이 필드들은 다른 단원에서 자세히 설명되어 있고 지금은 크게 신경쓰지 않아도 됩니다.

* long gameLoop\_interval\_ns
  + 게임을 진행하는 각 프레임 사이의 간격을 나노초 단위로 설정합니다.
  + 기본값은 16666666이며 이를 FPS(초당 프레임 수)로 환산하면 약 60이 됩니다.
* boolean gameLoop\_use\_virtualTimingMode
  + 게임 루프를 가상 타이밍 모드로 동작시킬지 여부를 설정합니다.
  + 기본값은 true (가상 타이밍 모드 사용) 입니다.
* int numberOfButtons
  + 게임 내에서 총 몇 개의 가상 버튼을 bind하여 사용할 것인지 설정합니다.
  + 기본값은 8입니다.

# 기본 1 - 게임 화면에 요소 직접 그리기

이 단원에서는 LOOT를 사용하여 게임 화면에 요소를 직접 그리는 방법을 설명합니다. 요소를 그릴 때 사용할 그림 파일 하나를 준비한 다음 아래의 순서에 따라 프로그램을 구성해 보세요.

* '시작' 단원의 내용을 참고하여 새 Java project를 만들고 GameFrame을 상속받는 새 클래스를 만드세요.  
  모든 작업을 성공적으로 마쳤다면 새 클래스에는 총 네 개의 메서드가 정의되어 있을 것입니다.
* 새 Java project를 우클릭한 다음 New 🡪 Folder를 클릭하여 'Images'라는 이름의 새 폴더를 추가하세요.  
  이 폴더는 그리기 작업에 사용할 그림 파일을 담아 두는 곳으로 활용됩니다.
* Images 폴더 안에 미리 준비한 그림 파일을 드래그 & 드롭으로 추가하세요. 이 때 뜨는 창에서 Copy files를 선택하면 해당 파일을 Images 폴더에 복사해 줍니다.
* **새 클래스** **안**에 아래의 코드를 추가하세요:

class MyObject extends DrawableObject //여기까지 적고 Ctrl + Space 한 번 입력(자동 import)

{

public MyObject()

{

super(350, 250, 100, 100, images.GetImage("img"));

}

}

MyObject myObject;

* 새 클래스 안에 있는 Initialize()의 내용을 아래의 코드로 바꾸세요:

images.LoadImage("Images/그림\_파일\_이름", "img");

myObject = new MyObject();

return true;

* 새 클래스 안에 있는 Update()의 내용을 아래의 코드로 바꾸세요:

return true;

* 새 클래스 안에 있는 Draw()의 내용을 아래의 코드로 바꾸세요:

BeginDraw();

myObject.Draw(g);

EndDraw();

* 이제 main()이 잘 정의되어 있나 다시 한 번 확인한 다음에 Ctrl + F11을 눌러 프로그램을 실행해 보세요.  
  게임 화면의 크기가 800x600일 때 화면 중앙에 그림이 그려져 있으면 성공입니다!
  + 그림이 안 보이는 경우 각 설정값(특히 그림 파일 경로 및 이름)을 다시 한 번 확인해 보세요.  
    그래도 안 보이는 경우, 해당 그림을 다른 형식으로 변환한 다음 사용해 보세요. (png, jpg 등)

## 주목할 점

* GameFrame 안에는 여러분의 프로그램에서 반드시 사용하게 될 요소들이 많이 포함되어 있습니다.  
  그 중 하나인 images는 ImageResourceManager class의 인스턴스로서 이름처럼 image들을 관리해 줍니다.
  + LoadImage()를 호출하면 그림 파일에서 image를 읽어 와 여러분이 지정한 이름으로 보관해 줍니다.
  + GetImage()를 호출하면 보관중인 해당 이름의 image를 반환해 줍니다.
* DrawableObject는 '2차원 화면에 그릴 수 있는 요소'를 나타내는 클래스입니다.
* Initialize()는 GameFrame 초기화가 모두 끝나고 게임이 시작되기 직전에 단 한 번 호출됩니다.  
  요소 초기화 작업, 각종 불러오기 작업은 보통 Initialize() 안에서 수행하게 됩니다.
* Draw()는 여러분이 설정한 시간 간격에 따라 게임 화면을 다시 그리기 위해 호출됩니다.
  + Draw()의 시작과 끝에서는 **반드시** BeginDraw()와 EndDraw()를 호출해 주어야 합니다.
  + 요소를 그릴 때는 myObject.Draw(g);처럼 g라는 필드를 담아 해당 요소의 Draw(g)를 호출하면 됩니다.  
    (이 문서에서는 GameFrame.Draw()와 구분하기 위해 각 요소를 그리는 메서드는 Draw(g)로 표현합니다)
  + 당연하게도, Draw(g) 호출을 하지 않으면 해당 요소는 화면에 그려지지 않습니다.

## 추가로 해 볼 만한 것들

* MyObject class의 생성자에 있는 super() 호출 부분에서 각 숫자를 바꾸면  
  화면에 보이는 요소의 위치 및 크기가 바뀝니다.
  + 심지어 크기가 음수일 때는 그림이 반전되어 그려집니다.
* 예제 코드에서는 요소를 하나만 그렸지만 필드를 여러 개 선언하여 여러 요소를 같이 그릴 수도 있습니다.
  + 두 요소의 위치가 서로 겹치는 경우 Draw(g)가 나중에 호출된 요소가 더 위에 있는 것처럼 보입니다.  
    Draw(g)는 항상 '덧그리기' 방식으로 요소를 그립니다.
* 이 문서에 있는 코드를 복사하여 여러분의 .java 파일로 가져간 경우 Ctrl + Shift + F를 누르면 가져온 코드를 적절하게 다듬어 줍니다.

# 기본 2 - 시간 흐름에 따라 요소의 값이 달라지게 만들기

이 단원에서는 요소에 '물리'를 추가한 다음 Update()를 통해 요소의 각 값을 바꾸어 보고 이러한 변화가 게임 화면에 어떻게 적용되는지 확인해 봅니다. 먼저 '기본 1' 단원의 내용을 수행한 다음 아래의 순서에 따라 프로그램을 변경해 보세요.

* '기본 1' 단원의 내용을 모두 수행한 경우 새 클래스에는 MyObject class, myObject 필드, 새 클래스의 생성자, 그리고 Initialize(), Update(), Draw()가 있게 됩니다. 잘 확인해 보고 실행해 보고 이상이 없으면 다음 단계로 진행하세요.
* MyObject class에 다음 필드를 추가하세요:

double pos\_x, pos\_y; //pos는 요소의 위치를 나타냅니다.  
double vel\_x, vel\_y; //vel은 요소의 속도를 나타냅니다.  
double acc\_x, acc\_y; //acc은 요소에 걸린 가속도를 나타냅니다.

* MyObject class의 생성자 내용을 아래의 코드로 바꾸세요:

public MyObject()  
{  
 super(350, 50, 100, 100, images.GetImage("img"));  
 pos\_x = 350;  
 pos\_y = 50;  
 vel\_x = 6.4; //초기 속도(현재 오른쪽 방향으로 이동 중)  
}

* 이 단원에서는 '장력'을 사용하여 요소의 가속도를 결정합니다. 장력은 기본적으로 두 점 사이의 길이에 비례하며 여기서는 힘의 크기를 쉽게 계산하기 위해 다른 변수들을 축약한 상수 계수, 장력이 걸릴 원점 좌표를 지정해 두고 사용합니다. MyObject class **바로** **위(새 클래스 안)에** 아래의 코드를 추가하세요:

final double coef\_tension = 0.001f;  
final int origin\_x, origin\_y;

* 새 클래스의 생성자 안에 아래의 코드를 추가하세요:

origin\_x = settings.canvas\_width / 2 - 50; //화면 중앙에서 요소 크기의 절반만큼 왼쪽 위에 있는 좌표  
origin\_y = settings.canvas\_height / 2 - 50;

* 물리값을 눈으로 직접 확인하기 위해 TextBox를 사용해 봅시다. 새 클래스 안에 아래의 코드를 추가하세요:

TextBox tb\_physics; //TextBox까지 입력하고 Ctrl + Space를 눌러 import 구문 자동 추가

* Initialize()에 아래의 코드를 추가하세요:

tb\_physics = new TextBox(10, 10, 110, 70);

* 시간 흐름에 따라 myObject 필드에 있는 물리값을 바꾸기 위해 Update()의 내용을 아래의 코드로 바꾸세요:

myObject.acc\_x = ( origin\_x - myObject.pos\_x ) \* coef\_tension;  
myObject.acc\_y = ( origin\_y - myObject.pos\_y ) \* coef\_tension;  
  
myObject.vel\_x += myObject.acc\_x;  
myObject.vel\_y += myObject.acc\_y;  
  
myObject.pos\_x += myObject.vel\_x;  
myObject.pos\_y += myObject.vel\_y;  
  
myObject.x = (int)myObject.pos\_x;  
myObject.y = (int)myObject.pos\_y;  
  
tb\_physics.text = String.format(  
 "pos: (%d, %d)\n" +  
 "vel: (%+.2f, %+.2f)\n" +  
 "acc: (%+.2f, %+.2f)",  
 myObject.x, myObject.y, myObject.vel\_x, myObject.vel\_y, myObject.acc\_x, myObject.acc\_y);  
  
return true;

* 이번엔 Draw()의 내용을 아래의 코드로 바꾸세요:

BeginDraw();  
  
ClearScreen();  
  
myObject.Draw(g);  
  
tb\_physics.Draw(g);  
  
EndDraw();

* 이제 Ctrl + F11을 눌러 프로그램을 실행해 보세요.  
  게임 화면의 크기가 800x600일 때 화면 중앙을 기준으로 그림이 천천히 원 운동을 하면 성공입니다!
  + 아직까지도 이 코드들을 전부 손으로 옮겨 적고 있는 학생은 없으리라 생각되지만, 만약 그림이 안 보이는 경우 위의 내용을 각 위치에 잘 적었는지 여부를 다시 한 번 확인해 보세요.

## 주목할 점

* 일반적으로 컴퓨터 분야에서 2차원 좌표계를 사용할 때 y축은 **위에서 아래**로 향합니다.  
  따라서 (0, 0)은 화면의 왼쪽 위 끝을 나타내며 y속도가 양수면 요소는 아래로 이동합니다.  
  (2차원 배열을 생각하면 편합니다)
* 일반적으로 2차원 좌표계에서 특별한 언급을 하지 않는 경우 어떤 요소의 '원점'은 해당 요소의 왼쪽 위 끝 좌표를 의미합니다. myObject 또한, 눈으로 보기에는 화면 중심점과 요소 중심점 사이의 거리를 재는 듯 보이지만 실제로는 화면 중심에서 (-50, -50)만큼 이동한 점을 원점으로 요소의 왼쪽 위 끝 점과의 거리를 재어 물리값 계산에 활용하고 있습니다.
  + 단, 만약 width 또는 height가 음수인 경우에는 원점은 해당 요소의 오른쪽 또는 아래 끝 좌표가 됩니다.
* myObject는 이미 '화면 상의 위치'를 나타내는 x, y 필드를 가지고 있지만 여기서는 pos\_x, pos\_y를 추가로 선언하여 사용했습니다. x, y는 int 형식이므로 이를 그냥 사용하면 물리값의 소수 부분이 사라지게 되어 물리계가 몹시 어색하거나 몹시 불안해지게 됩니다. 여러분의 코드를 작성할 때도 속도 / 가속도와 같은 물리값은 가급적 float 형식 필드로 다루도록 하세요.
* Update()와 Draw()는 정해진 시간 간격에 따라 반복적으로 호출되며 이러한 호출 간격은 보통 일정하게 유지됩니다. settings를 건드리지 않았다면 이 두 메서드는 1초에 약 60번 호출되며 따라서 1/60초마다 가속도를 속도에 더하고 속도를 변위에 더하는 과정을 수행하게 됩니다.
* TextBox는 여러분이 문자열을 게임 화면에 적고 싶을 때 사용할 수 있는 클래스입니다. String.format()을 사용하여 원하는 문자열을 만든 다음 TextBox의 text 필드에 설정해 두면 다음 번 Draw(g)에서 해당 문자열을 화면에 적어 줍니다. TextBox는 text 필드 이외에도 글자색, 배경색, 줄 간격, 글자체 등을 설정할 수 있는 필드들을 노출하고 있습니다.
* TextBox class와 DrawableObject class는 모두 VisualObject class의 자식 클래스입니다. VisualObject class에는 x, y, width, height와 같이 그리기 작업에 필요한 핵심 필드가 포함되어 있으며 Draw(g) 추상 메서드 또한 VisualObject class에 선언되어 있습니다. 따라서 두 클래스는 사용 방법이 비슷하며 각각의 특징에 맞게 myObject.Draw(g);는 여러분의 그림을 해당 위치에 그리고 tb\_physics.Draw(g)는 여러분이 지정한 문자열을 해당 위치에 적어 줍니다.
* ClearScreen()은 현재 게임 화면에 그려져 있는 내용을 모두 지우고 settings에서 설정해 둔 배경색으로 초기화하기 위해 사용됩니다. 대부분의 경우 이 메서드는 BeginDraw()를 호출한 직후 단 한 번 호출됩니다.
* '주목할 점'이 많긴 하지만 이러한 내용을 모두 외울 필요는 전혀 없습니다. 여러분의 프로그램을 작성할 때 화면 구성 과정에서 어려운 점이 발생한 경우 예제 코드를 참조하거나 조교의 도움을 요청하세요.

## 추가로 해 볼 만한 것들

* Draw()에 있는 ClearScreen()을 지워 보세요! 좀 더 아름다운 광경이 펼쳐집니다!
* 장력의 경우 거리에 비례하는, 중점을 향한 힘으로 작용하므로 물리값이 좀 과하거나 어긋나도 대기권을 뚫고 나간다거나 하는 일은 잘 발생하지 않습니다. 여러 상수들, 변수들의 값을 바꾸어 보면서 이러한 차이가 요소에 어떤 영향을 주는지 확인해 보세요.
  + x, y 값을 바꿀 때와 마찬가지로 width, height 값 또한 아무 때나 바꿀 수 있습니다.  
    width 또는 height 값이 음수인 경우 그림은 좌/우 또는 상/하로 반전되어 그려집니다.
  + 물론, 그리기 작업에 사용할 image 필드를 바꿀 수도 있습니다.  
    예를 들면, 속도가 일정 값보다 큰 경우 일반 공 대신 불타는 공을 그리게 할 수도 있습니다.
* pos\_x, pos\_y가 게임 화면의 가장자리( 0, 0, settings.canvas\_width, settings.canvas\_height )에 도달했거나 아예 뚫고 나갔는지 검사하여 '충돌'을 감지하고 이를 처리할 수도 있습니다.
  + 요소가 화면 상단에 부딪힌 경우 y속도의 부호를 바꾸면 올라가던 요소가 이제 내려가기 시작합니다.
  + '충돌'을 좀 더 사실적으로 묘사하기 위해서는 조금 더 오랜 작업이 필요합니다.  
    그러니 그런 작업은 야심한 밤에 모니터만 앞에 두고 묵묵히 시도하는 편이 좋습니다.
* tb\_physics에 '.' 을 찍으면 꽤 많은 필드들이 나타납니다. 이들의 값을 수정하여 문자열을 적는 방법을 변경할 수 있습니다.
  + 하지만 보통 TextBox를 사용하는 것보다 미리 그린 그림으로 게임 메시지를 보여주는 편이 더 좋습니다. 문자열을 화면에 '그리기' 위해서는 전체 화면 그림보다 몇 십 배는 더 무거운 '글자체'를 불러 와야 하기 때문입니다. 따라서 TextBox는 '어떤 필드의 값 확인'과 같은 테스트 용도로만 쓰는 편이 좋겠습니다.
* settings.gameloop\_interval\_ms 필드의 값을 바꿔서 Update()/Draw()를 호출하는 '간격'을 조정할 수 있습니다. 여기서는 'Δt'를 별도로 다루지 않았기 때문에 호출 간격이 짧을수록 물리값들이 더 빠르게 변화하게 됩니다.
  + Update() / Draw()를 한 차례 호출하는 것을 가리켜 'frame'이라 부르며  
    60 FPS란 1초에 60번 Update()가 호출된다는 것을 의미합니다.
* 아마 대부분의 경우에는 필요 없겠지만, Update()가 false를 반환하면 해당 프레임의 Draw()는 생략됩니다.  
  여러분이 만드는 프로그램이 '너무 느린 경우' 이 기능을 고려해 보면 좋겠지만  
  그렇게까지 무거운(무거워야 할) 프로그램을 한 달 안에 만드는 것은 어차피 불가능하니  
  성능 문제로 크게 걱정할 일은 아마 발생하지 않을 것입니다.

# 기본 3 - 3차원 공간 다루기

이 단원에서는 LOOT를 사용하여 가상 3차원 공간에 요소를 배치한 다음 다루는 방법을 설명합니다. '기본 2' 단원의 내용을 수행한 다음 아래의 순서에 따라 프로그램을 변경해 보세요.

* MyObject class 정의 부분을 다음과 같이 대폭 수정하세요:

class MyObject extends DrawableObject**3D** // 여기서도 역시 Ctrl + Space{  
 double vel\_x, vel\_y, vel\_z; //원래 있던 pos\_x, pos\_y는 지웁니다.  
 double acc\_x, acc\_y, acc\_z;  
  
 public MyObject()  
 {  
 super(0, 0, 200, 50, 50, images.GetImage("img"));  
 vel\_x = 6.4;  
 vel\_y = 6.4;  
 }  
}

수정 후 아무 오류도 발생하지 않으니 이를 통해 DrawableObject3D에 pos\_x, pos\_y가 이미 정의되어 있다는 것을 추측할 수 있습니다(물론 pos\_z도 함께 정의되어 있습니다). 발생하는 경고는 코드 파일 맨 위의 import loot.graphics.DrawableObject; 라인을 통째로 지워 주면 사라집니다.

* MyObject class에 있는 origin\_x, origin\_y 선언 부분을 지우고 아래의 코드를 추가하세요:

class Origin extends DrawableObject3D  
{  
 public Origin()  
 {  
 super(0, 0, 0, 50, 50, images.GetImage("origin"));  
 }  
}

이 클래스는 장력이 작용되는 중심점을 표현하기 위해 사용됩니다.

* 이제 발생한 몇몇 오류를 먼저 처리해 봅시다.
  + 새 클래스의 생성자 안에서 오류를 일으키는 두 줄(origin\_x, y 초기화)을 그냥 지워버리세요.
  + MyObject myObject; 아래에 Origin origin; 을 추가한 다음  
    Update() 안의 origin\_x, y 사용 부분을 각각 origin.pos\_x, origin.pos\_y로 변경하세요.
* Images 폴더 안에 새로 준비한 그림 파일을 드래그 & 드롭으로 추가하세요. 이 때 뜨는 창에서 Copy files를 선택하면 해당 파일을 Images 폴더에 복사해 줍니다.
* 가상 3차원 공간을 사용하기 위해 Viewport를 사용해 봅시다.  
  방금 Origin origin; 을 추가한 부분 바로 아래에 Viewport viewport; 를 추가하세요.
* Initialize()의 내용을 아래의 코드로 바꾸세요:

images.LoadImage("Images/원래\_있던\_그림\_파일\_이름", "img");  
images.LoadImage("Images/새\_그림\_파일\_이름", "origin");  
  
myObject = new MyObject();  
origin = new Origin();  
tb\_physics = new TextBox(10, 10, 200, 70);  
viewport = new Viewport(0, 0, settings.canvas\_width, settings.canvas\_height);  
  
viewport.children.add(myObject);  
viewport.children.add(origin);  
viewport.children.add(tb\_physics);  
  
//'카메라' 설정 부분  
viewport.pointOfView\_z = 500;  
viewport.view\_baseDistance = 500;  
viewport.view\_minDistance = 0.1;  
viewport.view\_maxDistance = 1000;  
viewport.view\_width = settings.canvas\_width;  
viewport.view\_height = settings.canvas\_height;  
  
return true;

* Update()의 내용을 아래의 코드로 바꾸세요:

myObject.acc\_x = ( origin.pos\_x - myObject.pos\_x ) \* coef\_tension;  
myObject.acc\_y = ( origin.pos\_y - myObject.pos\_y ) \* coef\_tension;  
myObject.acc\_z = ( origin.pos\_z - myObject.pos\_z ) \* coef\_tension;  
  
myObject.vel\_x += myObject.acc\_x;  
myObject.vel\_y += myObject.acc\_y;  
myObject.vel\_z += myObject.acc\_z;  
  
myObject.pos\_x += myObject.vel\_x;  
myObject.pos\_y += myObject.vel\_y;  
myObject.pos\_z += myObject.vel\_z;  
  
tb\_physics.text = String.format(  
 "pos: (%+.2f, %+.2f, %+.2f)\n" +  
 "vel: (%+.2f, %+.2f, %+.2f)\n" +  
 "acc: (%+.2f, %+.2f, %+.2f)",  
 myObject.pos\_x, myObject.pos\_y, myObject.pos\_z,   
 myObject.vel\_x, myObject.vel\_y, myObject.vel\_z,  
 myObject.acc\_x, myObject.acc\_y, myObject.acc\_z);  
  
return true;

* 마지막으로 Draw()의 내용을 아래의 코드로 바꾸세요:

BeginDraw();  
ClearScreen();  
viewport.Draw(g);  
EndDraw();

* 이제 Ctrl + F11을 눌러 프로그램을 실행해 보세요.  
  화면 중앙에 새 그림이 보이고 원래 그림이 이를 중심으로 천천히 공전 운동을 하면 성공입니다!
  + 만약 그림이 안 보이는 경우 images.LoadImage() 호출 부분을 다시 한 번 확인해 보세요.

## 주목할 점

* 이전에 살펴 본 2차원 좌표계와 다르게, 보통 '그리기를 위한' 3차원 공간에서는 y축이 **아래에서 위**로 향하고 오른손 법칙에 따라 z축은 화면 멀리서 눈 앞으로 다가오는 방향을 향합니다(수학적 배경이 깔린 곳에서는 보통 수직으로 올라가는 축이 z축이 되고 x, y축이 오른손 법칙에 따라 바닥에 깔립니다).
* 일반적으로 3차원 좌표계에서 어떤 요소의 '원점'은 해당 요소의 중심점을 의미합니다. 따라서 LOOT에서는 3차원 공간에 배치할 요소를 초기화할 때 width, height 대신 이들 값의 절반이 되는 radius\_x, radius\_y를 사용합니다.
* DrawableObject3D는 VisualObject3D의 자식 클래스고 VisualObject3D는 VisualObject의 자식 클래스입니다. VisualObject3D는 VisualObject에서 상속받은 필드 이외에 3차원 공간에서 해당 요소가 갖는 위치와 크기를 기록할 필드들을 추가로 가집니다.
* Viewport.Draw(g)는 현재 Viewport에 포함된 요소들의 위치 및 크기를 토대로 자신이 점유한 화면 영역에 각 요소를 어떤 순서로, 어디에, 얼마나 크게 그릴 것인지 계산하여 전체 그리기 작업을 수행합니다.
  + 공전 운동을 하는 그림은 z좌표가 음수인 경우 원점보다 먼저 그려집니다(더 뒤에 있는 듯 보임).
* Viewport.children은 VisualObject를 담는 목록이며 따라서 MyObject와 Origin과 TextBox를 모두 다 담을 수 있습니다. TextBox는 VisualObject3D에서 상속받지 않기 때문에 3차원 공간에 배치될 수 없으며 이러한 2차원 요소는 항상 그리기 작업의 가장 마지막에 그려집니다(항상 위에 있는 것으로 보입니다). 이를 응용하여 게임 화면 전체를 점유하는 Viewport를 만든 다음 '액자' 역할을 수행할 2차원 그림을 추가하여 게임 화면을 시각적으로 구성할 수도 있습니다.
  + Viewport 자체도 VisualObject3D의 자손이 되는 클래스입니다. 따라서 마음만 먹으면 Viewport 안에 다른 Viewport를 포함시킬 수도 있습니다. 이 때 내부 Viewport가 점유할 영역의 위치와 크기는 외부 Viewport에 의해 결정됩니다.
  + 이 단원에서 viewport는 항상 전체 화면을 점유하도록 구성되어 있으며 따라서 viewport는 2차원 화면에 배치되어 있습니다. 이처럼 3차원 요소를 2차원 요소처럼 활용하는 것은 언제나 가능하며 이는 해당 요소를 만들 때 생성자를 int가 잔뜩 달린 2차원 버전을 썼는지 float이 잔뜩 달린 3차원 버전을 썼는지에 따라 자동으로 지정되게 됩니다.
* viewport.pointOfView\_x, y, z는 광활한 3차원 공간 안에서 '카메라'가 어디에 위치해 있는지를 나타냅니다.
* viewport.view\_... 필드들은 카메라의 시야 범위와 배율을 설정하기 위해 사용됩니다. 이를 완벽히 이해하는 것은 2학년 수준에서는 어려운 편이므로 Update()에서 이들의 값을 조금씩 바꾸도록 구성하여 각 필드의 변화가 화면에 주는 영향을 확인해 보며 유추하는 편이 낫겠습니다.

# 기본 4 - 입력 적용하기

이 단원에서는 LOOT를 사용하여 키보드 / 마우스 입력을 가져와 적용하는 방법을 설명합니다. '기본 3' 단원의 내용을 수행한 다음 아래의 순서에 따라 프로그램을 변경해 보세요.

* 먼저, 코드를 간단하게 작성하기 위해 몇몇 필드의 값을 조건에 맞추어 두어야 합니다. 아래에 나열된 각 필드의 값이 조건에 맞는지 확인해 보세요(이전 단원에서 해당 값을 바꾸지 않았다면 그대로 진행하면 됩니다):
  + viewport.pointOfView\_z의 값은 viewport.view\_baseDistance의 값과 같아야 합니다.
  + viewport.view\_width, view\_height의 값은 settings.canvas\_width, canvas\_height의 값과 같아야 합니다.
  + Origin의 pos\_z는 항상 0이어야 합니다. (생성자의 세 번째 argument가 pos\_z를 설정합니다)
* Initialize()에 다음 코드를 추가하세요:

inputs.BindKey(KeyEvent.VK\_E, 0);  
inputs.BindKey(KeyEvent.VK\_SPACE, 1);  
inputs.BindMouseButton(MouseEvent.BUTTON1, 2);

이 코드는 키보드의 두 키와 마우스의 주 버튼(오른손잡이 모드일 때 왼 쪽 버튼)을 가상 버튼에 bind합니다.

* Update()의 가장 윗 부분에 다음 코드를 추가하세요:

inputs.AcceptInputs(); //이제까지 들어온 입력을 가상 버튼에 반영  
  
//현재 마우스를 누르고 있다면 원점의 위치를 현재 마우스 포인터의 위치로 설정  
if ( inputs.buttons[2].isPressed == true )  
{  
 Point pos\_new\_origin = viewport.GetRelativePositionFromCenter(inputs.pos\_mouseCursor);  
 origin.pos\_x = pos\_new\_origin.x;  
 origin.pos\_y = -pos\_new\_origin.y;  
}  
  
//현재 E 키를 누르고 있다면 원점의 위치를 화면 중앙으로 초기화  
if ( inputs.buttons[0].isPressed == true )  
{  
 origin.pos\_x = 0;  
 origin.pos\_y = 0;  
}  
  
//이번 프레임에 space bar를 누르기 시작했다면  
//공전 운동을 하는 요소의 물리값을 현재 원점을 기준으로 하는 초기 상태로 재설정  
if ( inputs.buttons[1].IsPressedNow() == true )  
{  
 myObject.pos\_x = origin.pos\_x;  
 myObject.pos\_y = origin.pos\_y;  
 myObject.pos\_z = 200 + origin.pos\_z;  
   
 myObject.vel\_x = 6.4;  
 myObject.vel\_y = 6.4;  
 myObject.vel\_z = 0;  
}

* 이제 Ctrl + F11을 눌러 프로그램을 실행해 보세요.  
  마우스 주 버튼을 누르는 동안 원점이 마우스 포인터를 따라 다니고  
  E 키를 누르는 동안 원점의 위치가 화면 중앙으로 고정되고  
  Space bar를 누른 순간 공전 운동을 하는 요소의 물리값이 초기화되면 성공입니다!

## 주목할 점

* Bind 가능한 가상 버튼의 수는 settings.numberOfButtons에 의해 결정됩니다.
  + 가상 버튼은 배열 형태로 관리되므로 만약 버튼을 3개 사용하는 경우 이들의 번호는 각각 0, 1, 2여야 합니다.
* 가상 버튼을 사용하는 경우 Update()의 가장 윗 부분에서 inputs.AcceptInputs()를 꼭 호출해 주어야 합니다.
* 각 가상 버튼의 현재 상태는 isPressed, isChanged, IsPressedNow(), IsReleasedNow()로 확인할 수 있습니다.
  + isPressed는 해당 버튼이 현재 눌려 있는지를 나타내며  
    isChanged는 해당 버튼의 누름 상태가 이번 프레임에 변경되었는지 여부를 나타냅니다.  
    IsPressedNow()와 IsReleasedNow()는 이 두 필드의 값을 토대로 이번 프레임에 해당 버튼을 눌렀는지 / 뗐는지 여부를 반환해 줍니다.
* 이번 프레임의 마우스 포인터 위치는 inputs.pos\_mouseCursor에 기록되어 있습니다.
* 현재 코드대로라면, 마우스와 E 키를 모두 누르고 있으면 'E 키 입력'으로 간주됩니다. 만약 그 반대로 동작하는 프로그램을 작성하고 싶다면 두 if문의 위치를 바꾸거나 아래 if문에 else를 붙여 주면 됩니다.
* VisualObject, VisualObject3D, Viewport에는 좌표를 변환하거나 다루기 위한 메서드들이 다수 정의되어 있습니다. VisualObject.GetRelativePositionFromCenter()는 현재 요소의 2차원 화면에서의 중점에 대한 해당 좌표의 상대 위치를 계산해 줍니다.
* origin.pos\_y는 3차원 좌표값이므로 마우스 커서의 2차원 상대 위치인 pos\_new\_origin.y를 옮길 때 부호를 바꾸어 주고 있습니다. 헷갈릴 수 있으니 2차원에선 y축이 위에서 아래로, 3차원에선 아래에서 위로 향한다는 것을 꼭 기억해 두세요. 안 그러면 멀쩡한 캐릭터가 갑자기 하늘로 치솟거나 총알을 바닥에 대고 쏘게 될 수 있습니다.