#### Camera class



The camera class is basically the frame of the game play that keeps moving according to the movement of the player, and updates the background on the screen.

*Constructor*

**public Camera** **( float xPos, float yPos):** This constructor initializes the horizontal and vertical positions of the frame.

*Attributes*

**private float xPos:** This attribute holds the horizontal position of the player on the screen and keeps updating as the player moves.

**private float yPos:** This attribute holds the vertical position of the player on the screen and keeps updating as the player moves.

*Methods*

**public void updateFrame( GameObject player ):** This method is updating the frame by getting the player’s positions and adding that to half the window’s width, so that the player always remains at the centre of the screen.

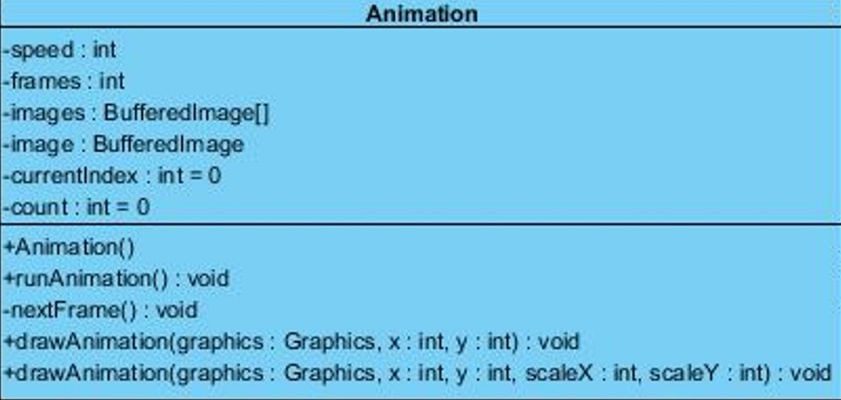
**public float getXPos():** This method returns the float variable xPos.

**public void setXPos(float xPos):** This method sets the xPos variable to the parameter xPos.

**public float getYPos():** This method returns the float variable yPos.

**public void setYPos(float yPos):** This method sets the yPos variable to the parameter yPos.

#### Animation class



The Animation class is responsible for the initialization and creation of animation and the smooth flow of images. It is the base class for producing the movement of player, weapons and enemies.

*Constructor*

**public Animation( int speed, BufferedImage... args ):** This constructor initializes the Animation object by drawing all the objects for the first time and it saves each image to the array to be displayed for the movement of objects.

*Attributes*

**private BufferedImage[] images:** This attribute holds the image icons for the animation of each object in an image array to be repeated in a loop continuously.

**private BufferedImage image: :** This attribute holds the image icon.

**private int currentIndex :** This attribute holds the current index and initialize to zero.

**private int count :** this is a counter to load as many frames as their are images.

*Methods*

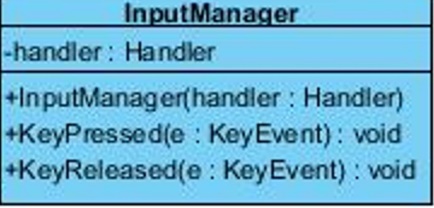
**public void runAnimation():** This method keeps on incrementing the currentIndex until it becomes greater than the speed of this animation, after which it sets the currentIndex to zero and calls the nextFrame method, which displays the next frame. So basically this method is responsible for displaying the image icons according to the given speed.

**public void drawAnimation( Graphics graphics, int x, int y ):** This method basically draws the graphics onto the screen at x and y position by calling the drawImage method.

**public void drawAnimation( Graphics graphics, int x, int y, int scaleX, int scaleY ):** this is the overloaded version of the above method, the only difference is that it also takes the width and height of the graphics.

**private void nextFrame():** This method loads each image from the image array and keeps there to display, meaning it loads the next frame of the animation, and keeps on incrementing the count until it becomes greater than the number of frames of this animation, after which it sets the count to zero, meaning the animation starts all over again.

#### InputManager class



This class is for directing the player according to the keyboard inputs.

*Constructor*

**public InputManager( Handler handler ):** This constructor will initialize the handler variable.

*Attributes*

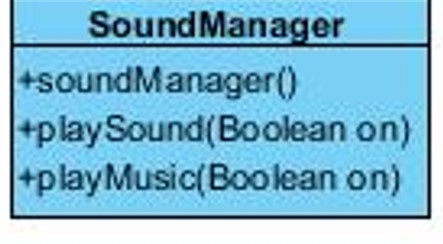
**Handler handler:**  this attribute is responsible for storing the handler instance of the game.

*Methods*

**public void keyPressed(KeyEvent event):** This is a method for managing the pressed input and creating an event accordingly to that. This method takes event variable as a parameter and gets the key code of event. If the key is equal to the space bar or arrow keys the method creates an event accordingly. For each time a key is pressed, the player object in handler’s objectLinkedList, is directed to do something, for example: UP or space = jump, DOWN = move down fastly if the player is in the air, LEFT = move east, RIGHT = move west, ESCAPE = quit game.

**public void keyReleased(KeyEvent event):** This is a method for managing the released input and creating an event according to that. This method takes event variable as a parameter and gets the key code of event. If the key is equal to the space bar or arrow keys the method creates an event accordingly. For example, if the key A is released then a pen is thrown.

#### SoundManager class



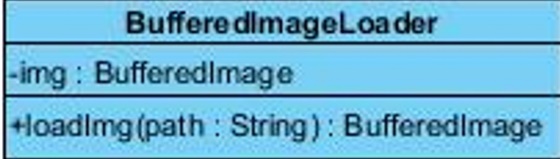
SoundManager class is for managing the sounds and music.

*Constructor* **public SoundManager():** This constructor will set the music and sounds to on.

*Methods*

**public playMusic(boolean on):** This method is for turning the music on. **public playSound(boolean on):** This method is for turning the sound on.

#### BufferedImageLoader Class



*Attributes*

**private BufferedImage img: This** attribute holds the image icons for the objects.

*Methods*

**public BufferedImage loadImg (String path):** This method takes string path as a parameter loads the image in the path with the help of calling getResource method and returns that image.

#### SpriteSheet Class



*Constructor*

**public SpriteSheet( BufferedImage image ):** This constructor initializes the image variable.

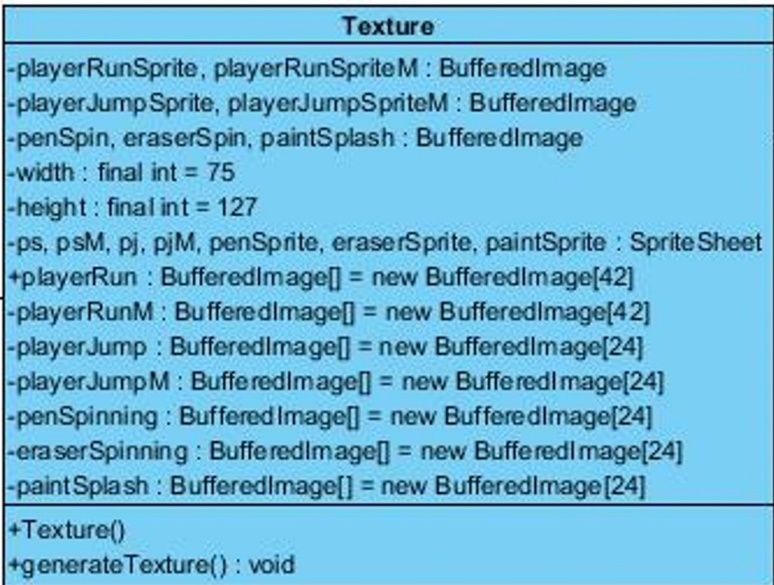
*Attributes*

**private BufferedImage img:** Thisattribute holds the image of the sprite sheet that contains all the icons for each object’s animation

*Methods*

**public BufferedImage divideImage( int row, int col, int width, int height ):** this method divides the sprite sheet into parts and gets each single image for the animation of an object. It uses the getSubImage method to get the image at (x,y) coordinate with the given width and height and returns that image.

#### Texture Class



*Constructor*

**public Texture():** this constructor initializes the running, jumping and weapon throwing animations of the player.

*Attributes*

**private BufferedImage playerRunSprite, playerRunSpriteM, playerJumpSprite:** these attributes save the sprite sheets for the running and jumping animation.

**public BufferedImage[] playerRun :** this attribute holds each image for the running animation towards right.

**public BufferedImage[] playerRunM :** this attribute holds each image for the running animation towards left.

**public BufferedImage[] playerJump:** this attribute holds each image for the running animation towards right.

**public BufferedImage[] playerJumpM :** this attribute holds each image for the running animation towards left.

**public BufferedImage[] penSpinning :** this atttribute holds each image for the animation of pen spinning.

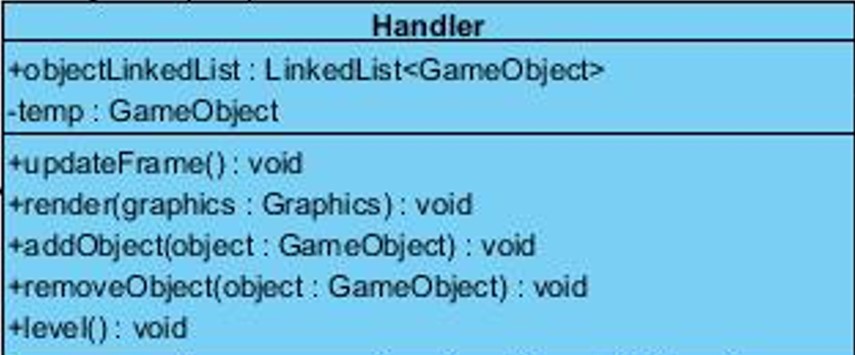
**private final int width, height:** these hold the height and width of the player images that we use.

**private SpriteSheet ps, psM, pj, pjM, penSprite**: these hold the spritesheets for running, jumping and weapon throwing animations.

*Methods*

**public void generateTextures():** this method calls the divideImage method of the SpriteSheet class and gets each single image from the spriteSheets and stores them into the respective arrays for animations.

#### Handler Class



*Attributes*

public LinkedList<GameObject> objectLinkedList: holds all the GameObjects that have been created during the since the start of the game.

private GameObject temp: a temporary GameObject variable to store any gameObject.

*Methods*

public void updateFrame(): this method calls the collsionDetector for each object in the objectLinkedList, which is responsible for calling the runAnimation method of that object. So basically whenever this method is called the frames of the object are updated, by calling nextFrame method in runAnimation.

public void render(Graphics graphics): this method calls the render method for each object in the objectLinkedList, which is responsible for calling the drawImage method of that object, so that the graphics are drawn and updated.

public void addObject (GameObject object): this method adds a new object to the objectLinkedList of this class by calling the add method of the linkedList class.

public void removeObject (GameObject object): this method removes an object from the objectLinkedList of this class by calling the remove method of the linkedList class.

public void level(): this method is just a test method for testing a sample level on the sceen by adding new block objects to the objectLinkedList.