**Project**

***Let’s Get Drunk***

# Overview

Let’s Get Drunk is the game that requires a player to collect objects(liquors) appearing on the screen as many as possible as shown in Figure 1 (left). When the game ends (there is a character collides with waste or a liquor collides with screen border), the game will stop and there is a dialog box showing the score as shown in Figure 1 (right) the screen will change to game over Screen and show the score. Note that a command for controlling the direction which the character will go is “F” for left drunkard (blonde hair) and “j” for right drunkard (black hair). The game can be paused by using “spacebar” key.

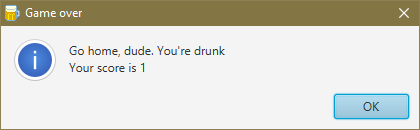
 

Figure 1. GUI of the Let’s Get Drunk game.

The followings are the game components and their conditions:

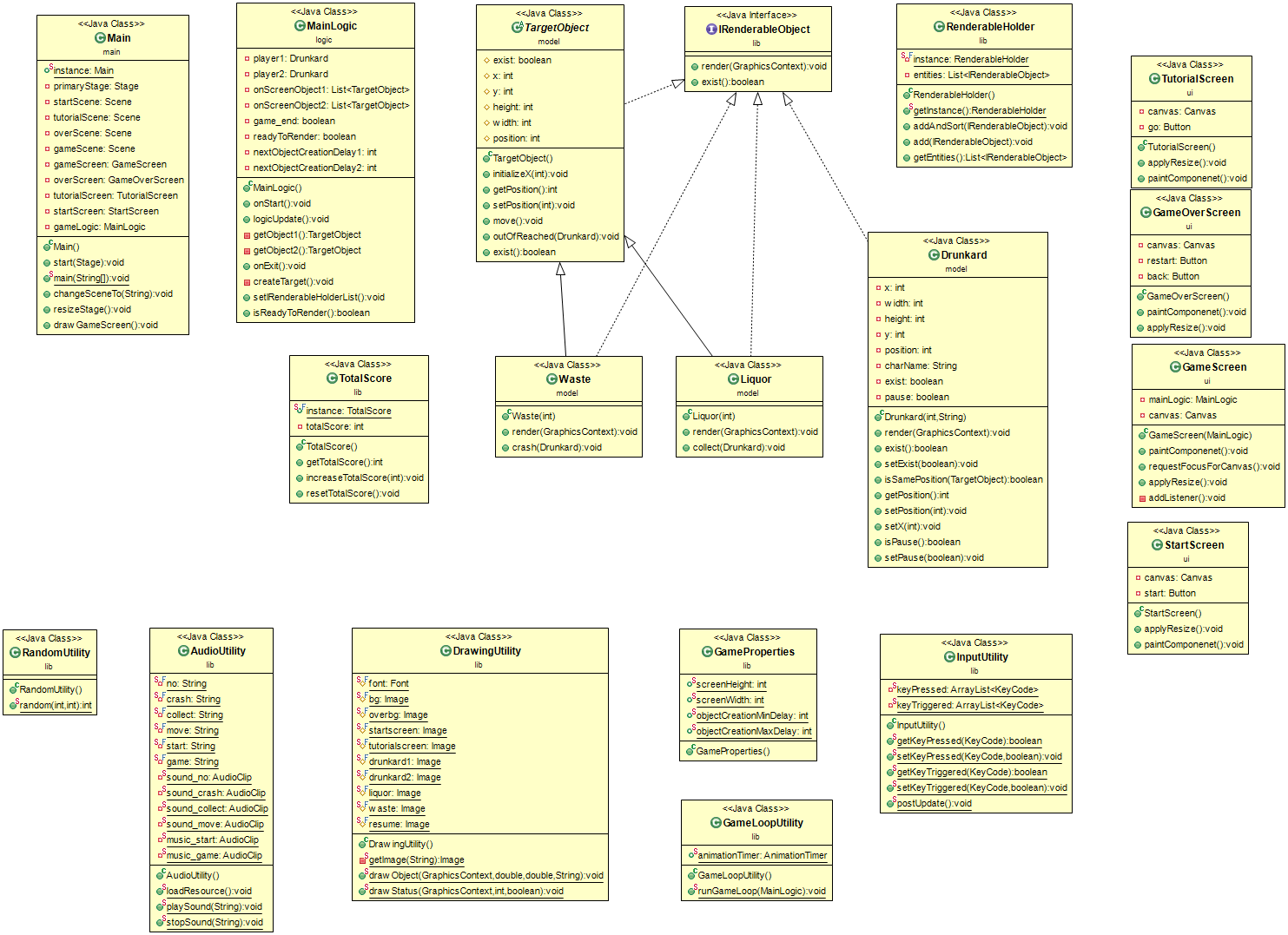
1. Player
   * There are two characters, the player has to control both of them at a time.
   * Player gains score by collecting the liquors until there is a liquor that doesn’t be collected or colliding a waste.
2. Target object
   * Each target object has different characteristics including starting lane defined by (x,y), position, width, and height.
   * A target object will randomly appear on the screen.
   * There are two types of target objects:
     + Obstacle (waste) doesn’t give score to player. When it collide with a player, a game will play sound and end.
     + Collectable object (liquor) gives scores to a player who collects them.

When it collide with a player, a player will collect it and add a point to the total score.

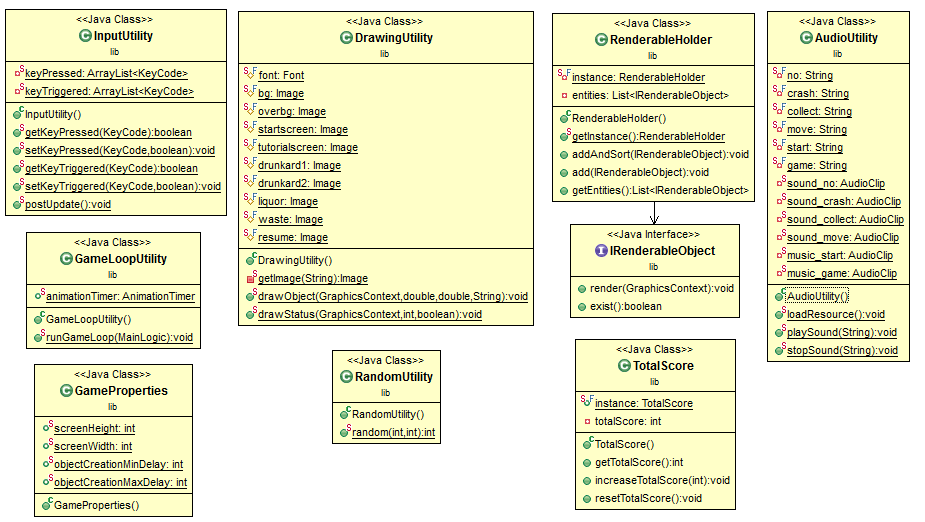
# Implementation Detailed

# In this part, an implementation of the game is followed the class diagram in Figure 2. The program have 5 packages; lib, logic, main, model and ui. The details for each package are shown in Figure 2.1, 2.2, 2.3, 2.4 and 2.5 below.

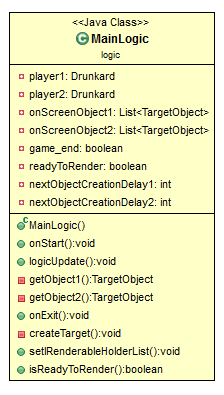
## 2.1 UML Diagram

****

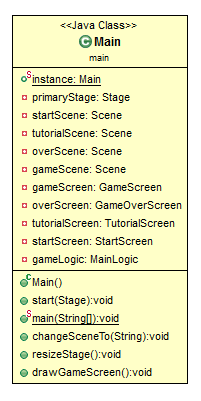
**Figure 2.1 Structure of classes in all Packages**



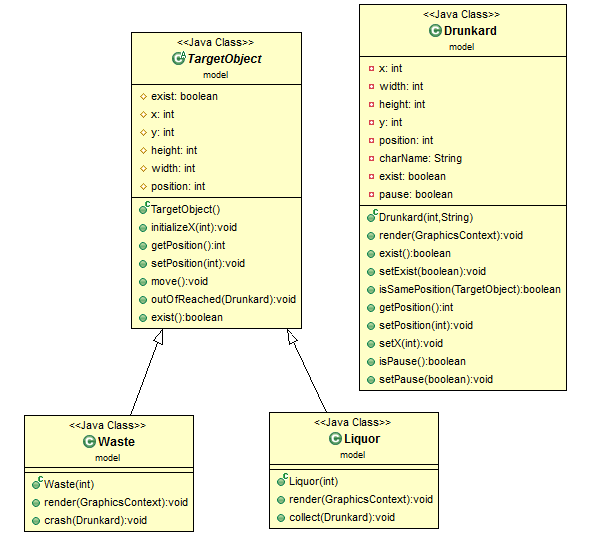
**Figure 2.2 Structure of classes in Package “lib”**



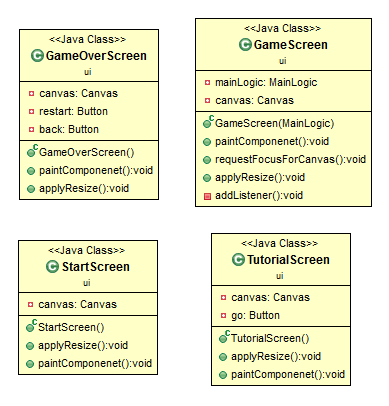
**Figure 2.3 Structure of classes in Package “logic”**



**Figure 2.4 Structure of classes in Package “main”**



**Figure 2.5 Structure of classes in Package “model”**



**Figure 2.6 Structure of classes in Package “ui”**



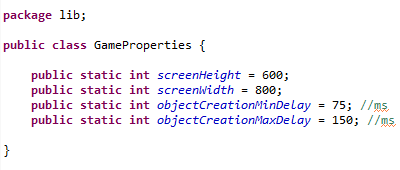
**Figure 3 Starting Screen / Class “StartScreen”**

## 2.2 Class “lib.GameProperties”

This class stores values data of the game screen and contains fields as follows:

### 2.2.1 Field

* + - **static int screenWidth, screenHeight**; width and height of every screen
    - **static int objectCreationMinDelay, objectCreationMaxDelay**; a delay interval for creating each object in the game



## 2.3 Class “lib.TotalScore”

This class is represents player’s total score.

### 2.3.1 Field

* + - **static final TotalScore instance;** a singleton instance
    - **int totalScore;** a summation of player’s score from each character

### 2.3.2 Constructor

* + - **TotalScore();** it set totalScore to 0.

### 2.3.3 Method

* + - **int getTotalScore();** it returns the player’s total score.
    - **void increaseTotalScore(int amount);** it increases totalScore by the given parameter.
    - **void resetTotalScore();** it resets totalScore to 0.

## 2.4 Interface “lib.IRenderableObject”

## Any class that can be drawn will implement this interface

### 2.4.1 Methods

* + - **void render(GraphicsContext gc);**
      * implement drawing

### boolean exist();

* + - * able to hide/show, return whether to draw this object or not

## 2.5 Class “lib.RenderableHolder”

### A share collection of renderable objects.

### 2.5.1 Fields

* + - ***RenderableHolder* instance**; a singleton object of RenderableHolder.
    - ***List<IRenderableObject>* entities**; list of IRenderable objects.

### 2.5.2 Constructor

* + - **RenderableHolder()**; Initializes entities.

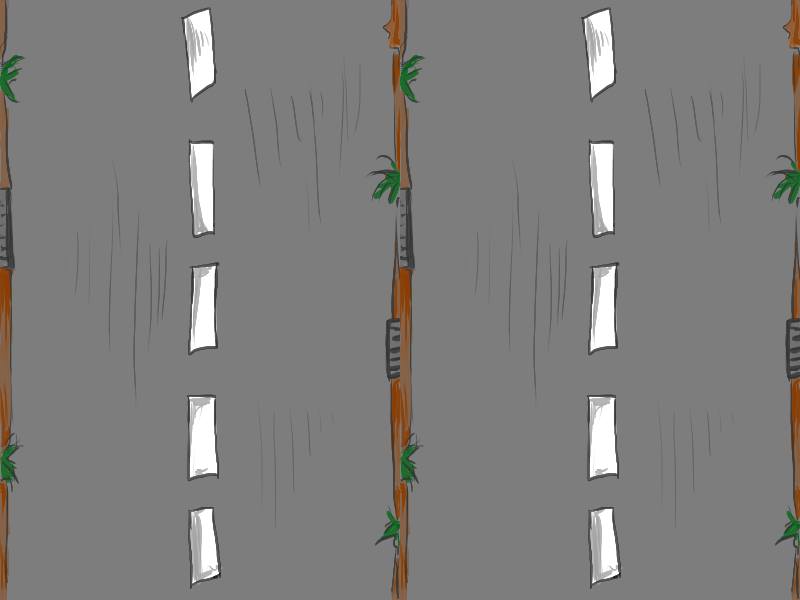
### 2.5.3 Methods

* + - ***RenderableHolder* getInstance()**; returns instance.
    - ***void* add(IRenderableObject entity)**; adds an IRenderable object to instance.
    - ***List<IRenderableObject>* getEntities()**; returns entities.

## 2.6 Class “lib.DrawingUtility”

This class is responsible for drawing backgrounds and objects that appear on the screen, such as characters, liquors, wastes, buttons and status (score and pause).



### 2.6.1 Field

* + - ***Font* font**; Font for score and pause using Font The Drunked Man, size 25
    - ***Image* bg**; Background image for game screen
    - ***Image* overbg**; Background image for game over screen
    - ***Image* startscreen**; Background image for start screen
    - ***Image* tutorialscreen**; Background image for tutorial screen
    - ***Image* drunkard1**; Image for drunkard1
    - ***Image* drunkard2**; Image for drunkard2
    - ***Image* liquor**; Image for liquor
    - ***Image* waste**; Image for waste

### 2.6.2 Methods

* + - **static *Image* getImage(String directory);**
      * Loads image from a specify directory with *ClassLoader* and returns the image in the format of Image. Return null if the image can’t be loaded.

### static *void* drawObject(GraphicsContext gc, int x, int y, String name);

* + - * It is used for drawing objects.

### Static void drawStatus (GraphicsContext gc, int score, boolean pause);



* + - * This method is used for drawing a status on the game screen.
      * Use white color string with *font* to show the status as below:
        + The score (on the top right) always reflect current game score.
      * If the game is paused, draw “SPACEBAR TO RESUME” in the center of game’s screen.

## 2.7 Class “lib.InputUtility”

This class is used for managing the keys.

### 2.7.1 Fields

* + - ***ArrayList<KeyCode>* keyPressed**; list of a key which is still pressed.
    - ***ArrayList<KeyCode>* keyTriggered**; list of a key which just triggered.

### 2.7.2 Methods

* + - ***boolean* getKeyPressed(KeyCode keycode);** getter for keyPressed
    - ***void* setKeyPressed(KeyCode keycode, Boolean pressed);** setter for keyPressed
    - ***boolean* getKeyTriggered(KeyCode keycode);** getter for keyTriggered
    - ***void* setKeyTriggered(KeyCode keycode, Boolean pressed);** setter for keyTriggered
    - ***void* postUpdate();**
      * This method is used for managing input in the triggering way.

## 2.8 Class “lib.AudioUtility”

This class is used for managing the sound.

### 2.8.1 Fields

* + - ***AudioClip* sound\_crash, sound\_collect, sound\_no, sound\_move, music\_start and music\_game**
      * The sound play while colliding wastes (res/se/crash.wav), collecting liquors (res/se/collect.wav), not collecting liquors (res/se/no.wav), and moving drunkards (res/se/move.wav), at start screen and tutorial screen (res/se/startsong.wav) and at game screen (res/se/gamesong.wav).

### 2.8.2 Methods

* + - * ***void* playSound(String identifier);**
        + Play the crash sound when the identifier value is “crash”
        + Play the collect sound when the identifier value is “collect”
        + Play the no sound when the identifier value is “no”
        + Play the move sound when the identifier value is “move”
        + Play the start music when the identifier value is “start”
        + Play the game music when the identifier value is “game”
      * ***void* stopSound(String identifier);**
        + Stop the start music when the identifier value is “start”
        + Stop the game music when the identifier value is “game”

## 2.9 Class “lib.GameLoopUtility”

This class is used for updating the game.

### 2.9.1 Fields

* + - ***AnimationTimer* animationTimer**;

### 2.9.2 Methods

* + - ***void* runGameLoop(MainLogic gameLogic);**
      * This method is used for updating the game by calling logicUpdate() from MainLogic.

## 2.10 Class “lib.RandomUtility”

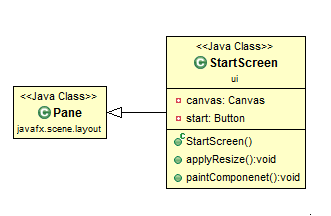
## This class is used for random the number.

### 2.10.1 Methods

* + - **static int random(int start, int end);**
      * randoms the number between start and end.

## 2.11 Class “ui.StartScreen” extends Pane

It is used for creating the starting GUI (Figure 3) and its diagram is shown in Figure 4.



**Figure 4 The structure of the “StartScreen” class**

### 2.11.1 Field

* + - * **Canvas canvas**; JavaFX component is an image on which we can draw text and images.
      * **Button start**; a button for starting a tutorial.

### 2.11.2 Constructor

* + - * It creates GUI (Figure 3).
      * The layout containing three main parts: logo (above), button (center), and text (below).
        + **Part1 - logo (above):** it is defined as image showing the game’s title
        + **Part2 - button (center):** it contains the start button.



When pressing the start button, it will change to tutorial screen.

* + - * + **Part3 - text (below):** names of developer.

### 2.11.3 Methods

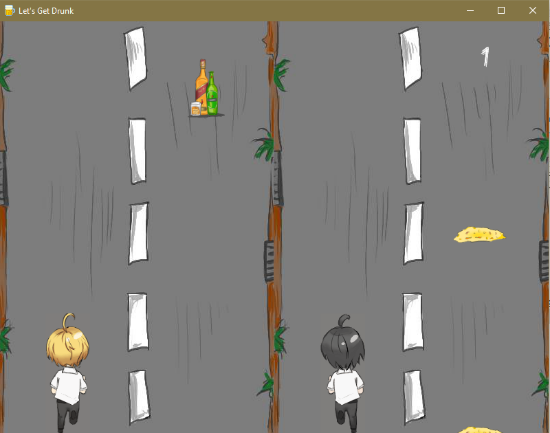
* + - **void paintComponenet();**
      * draw background
      * draw each IRenderable from theRenderableHolder instance

### void applyResize();

* + - * resize canvas with width and height from GameProperties

## 2.12 Class “ui.GameScreen” extends StackPane

This class contains canvas for drawing and main logic for updating the logic.



**Figure 5 game screen used for playing game**

### 2.12.1 Fields

* + - **MainLogic mainLogic;** logic that use to update all object in game
    - **Canvas canvas;** JavaFX component is an image on which we can draw text and images

### 2.12.2 Constructor

* + - **GameScreen(MainLogic mainLogic);** creates new GameScreen and initialize its main logic by given parameter
    - This class is extended from Pane Class. We create canvas used for drawing things in the game and store the canvas to the pane
    - Specifys value of variable fields, add canvas to the pane and call method addListener()
    - Initializes the values for canvas using width and height from GameProperties

### 2.12.3 Methods

* + - **void paintComponenet();**
      * draw background
      * draw each IRenderable from theRenderableHolder instance

### void requestFocusForCanvas();

* + - * used to make the program focus on canvas

### void applyResize();

* + - * resize canvas with width and height from GameProperties

### void addListener(); it is responsible for capturing action events. The event add to the Pane. This game records all of the inputs and, then, they will be further used by the “lib.InputUtillity” class, via the following events:

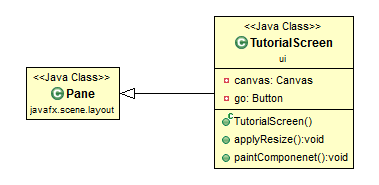
* + - * set KeyEvent to pane and canvas
      * KeyEvent
        + Used for capturing two keyboard events: (1) keyPrepressed (called when key pressed) and (2) keyReleased (called when key released)

## 2.13 Class “ui.TutorialScreen” extends Pane

It is used for creating the tutorial GUI (Figure 6) and its diagram is shown in Figure 7.



**Figure 6 Tutorial Screen / Class “TutorialScreen”**



**Figure 7 The structure of the “TutorialScreen” class**

### 2.13.1 Field

* + - * **Canvas canvas**; JavaFX component is an image on which we can draw text and images.
      * **Button go**; a button for going to a new game.

### 2.13.2 Constructor

* + - * It creates GUI (Figure 6).
      * The layout containing two main parts: how to play (above) and button (below).
        + **Part1 – how to play (above):** it is defined as image showing how to play the game
        + **Part2 - button (below):** it contains the go button.



When pressing the go button, it will change to game screen and play sound “game” (gamesong.wav).

### 2.13.3 Methods

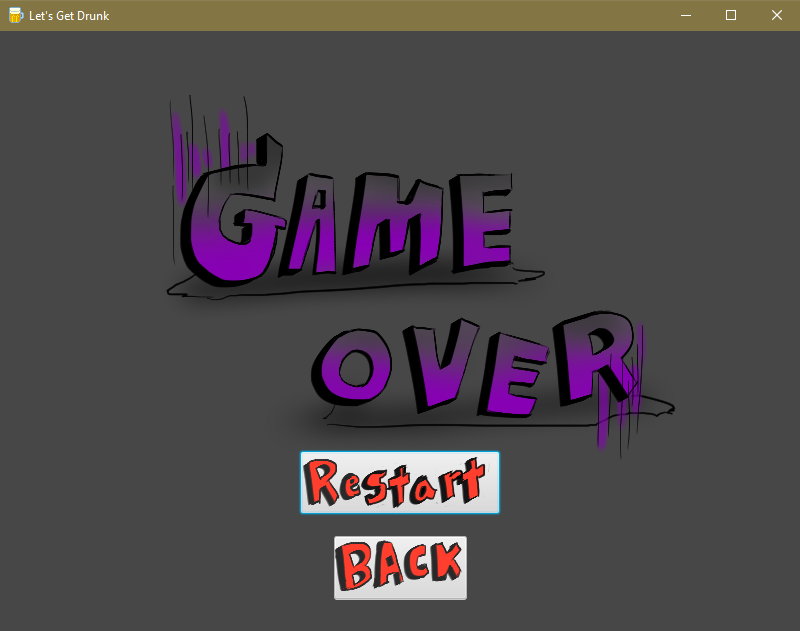
* + - **void paintComponenet();**
      * draw background
      * draw each IRenderable from theRenderableHolder instance

### void applyResize();

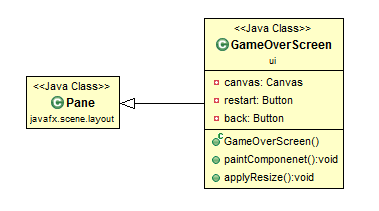
* + - * resize canvas with width and height from GameProperties

## 2.14 Class “ui.GameOverScreen” extends Pane

It is used for creating the game over GUI (Figure 8) and its diagram is shown in Figure 9.



**Figure 8 Game Over Screen / Class “GameOVerScreen”**



**Figure 9 The structure of the “GameOverScreen” class**

### 2.14.1 Field

* + - * **Canvas canvas**; JavaFX component is an image on which we can draw text and images.
      * **Button restart**; a button for starting a new game.
      * **Button back**; a button for going to a start screen.

### 2.14.2 Constructor

* + - * It creates GUI (Figure 8).
      * The layout containing two main parts: logo (above) and button (below).
        + **Part1 - logo (above):** it is defined as image showing that game over
        + **Part2 - button (below):** it contains the restart button and back button.

When pressing the restart button, it will change to game screen and play sound “game” (gamesong.wav).

When pressing the back button, it will change to start screen and play sound “start” (startsong.wav).

### 2.14.3 Methods

* + - **void paintComponenet();**
      * draw background
      * draw each IRenderable from theRenderableHolder instance

### void requestFocusForCanvas();

* + - * used to make the program focus on canvas

### void applyResize();

* + - * resize canvas with width and height from GameProperties

## 2.15 Class “main.Main” extends Application

The main class for handling game’s GUI

### 2.15.1 Fields

* **static Main instance;** used to calling Main.
* **Stage primaryStage;** a primary stage.
* **Scene startScene;** scene for start screen
* **Scene tutorialScene;** scene for tutorial screen
* **Scene gameScene;** scene for game screen
* **Scene overScene;** scene for game over screen
* **MainLogic gameLogic;** a logic of this game
* **GameOverScreen overScreen;** drawing module for game over screen as shown in Figure 8
* **GameScreen gameScreen;** drawing module for game screen as shown in Figure 1(left)
* **TutorialScreen tutorialScreen;** drawing module for tutorial screen as shown in Figure 6
* **StartScreen startScreen;** drawing module for start screen as shown in Figure 3

### 2.15.2 Methods

* **static void main(String[] args)**; start the game by launching application
* **void start(Stage primaryStage)**; override from Application. Set up stage scene and logic.
* **synchronized void changeSceneTo(String scene)**; used to change scene to startScene, tutorialScene, gameScene or overScene and play sound according to scene.
* **void resizeStage()**; resize the start Screen, tutorial Screen, game Screen and over Screen using method applyResize() and set primary stage to have the same size of scene using method sizeToScene()
* **void drawGameScreen();** paint all component in gameScreen

## 2.16 Class “logic.MainLogic”

This class controls the game.

### 2.16.1 Fields

* **Drunkard player1;** the drunkard.
* **Drunkard player2;** the drunkard.
* **List<TargetObject> onScreenObject1;** the list of target objects in lane 0 and 1 (half left of the screen).
* **List<TargetObject> onScreenObject2;** the list of target objects in lane 2 and 3 (half right of the screen).
* **boolean game\_end;** the game status (ended or not).
* **boolean readyToRender;** the rendering status (ready or not).
* **int nextObjectCreationDelay1;**  time interval before creating next object in lane 0 and 1 (half left of the screen).
* **int nextObjectCreationDelay2;** time interval before creating next object in lane 2 and 3 (half right of the screen).

### 2.16.2 Methods

* + - **synchronized void onStart()**; initializes the drunkard by given parameter, game\_end and exist, then random nextObjectCreationDelay1 and nextObjectCreationDelay2 by objectCreationMinDelay and objectCreationMaxDelay from GameProperties
      * Default player1 = Drunkard(0)
      * Default player2 = Drunkard(2)
      * Default game\_end = false
      * Default exist = true
    - **void logicUpdate()**; update the progress of the game. Ex:
      * moves players.
      * checks players collecting a liquor.
      * randomly puts a new object on the screen.
* **synchronized void onExit()**; clears onScreenObject1 and onScreenObject2, then sets readyToRender to false.
* **void createTarget()**; used nextObjectCreationDelay1, nextObjectCreationDelay2 to create target object in random position and type.
* **synchronized void setIRenderableHolderList();** calls entities to remove an object that didn’t exist and adds players and an object to the RenderableHolder instance.
* **boolean isReadyToRender();**
* **TargetObject getObject1();** returns target object that its position is same as character1 (blonde hair).
* **TargetObject getObject2();**returns target object that its position is same as character2 (black hair).

## 2.17 Abstract Class “model.TargetObject” implements IRenderableObject

This class represents a base class for liquor and waste.

### 2.17.1 Field

* + - **int x,y;** position of this object on the screen.
    - **int height;** height of this object’s image.
    - **int width;** width of this object’s image.
    - **int position;** position (lane in road background) where this object is.
      * 0 = first lane from left
      * 1 = second lane from left
      * 2 = second lane from right
      * 3 = first lane from right
    - **boolean exist;** represents the exist status for this object.

### 2.17.2 Constructor

* + - **TargerObject();** create new target object.
      * Default y = 0
      * Default exist = true

### 2.17.3 Method

* + - **int getPosition();** getter for position.
    - **void setPosition(int position);** setter for position.
    - **void initializeX(int width);** setter for x.
    - **void move();** moves the object.
    - **void outOfReached(Drunkard player);** check whether the object has collide with border of the screen or not.
      * If yes, play sound “no” (no.wav) then change the object and the player status to don’t exist.
    - **boolean exist();** return exist.

## 2.18 Class “model.Liquor” extends TargetObject

This class represents a collectible liquor.

### 2.18.1 Constructor

* + - **Liquor (int position);** create new liquor and initialize its position by given parameter.
      * Default width = 57
      * Default height = 90
      * Initialize x by width

### 2.18.2 Method

* + - **void render(GraphicsContext gc);** draw liquor image from RenderableHoler(liquor.png) at x,y position.
    - **void collect(Drunkard player);** checks whether the player has collide with a liquor or not.
      * If yes, play sound “collect” (collect.wav) then increases total score by one and the liquor doesn’t exist.

## 2.19 Class “model.Waste” extends TargetObject

This class represents an obstacle waste.

### 2.19.1 Constructor

* + - **Liquor (int position);** create new liquor and initialize its position by given parameter.
      * Default width = 78
      * Default height = 25
      * Initialize x by width

### 2.19.2 Method

* + - **void render(GraphicsContext gc);** draw waste image from RenderableHoler(waste.png) at x,y position.
    - **void crash(Drunkard player);** checks whether the player has collide with a waste or not.
      * If yes, play sound “crash” (crash.wav) then change the waste and the player status to don’t exist.

## 2.20 Class “model.Drunkard” implements IRenderableObject

This class represents characters.

### 2.20.1 Field

* + - **int x,y;** position of this character on the screen.
    - **int height;** height of this character’s image.
    - **int width;** width of this character’s image.
    - **int position;** position (lane in road background) where this character is.
      * 0 = first lane from left
      * 1 = second lane from left
      * 2 = second lane from right
      * 3 = first lane from right
    - **String charName;** represents the name for this character.
    - **boolean exist;** represents the exist status for this character.
    - **boolean pause;** represents the pause status for this character.

### 2.20.2 Constructor

* + - **Drunkard(int position, String charName);** create new player and initialize its position and image by given parameters.
      * Default y = screenHeight from GameProperties – its height
      * Default exist = true
      * Default pause = false

### 2.20.3 Method

* + - **int getPosition();** getter for position.
    - **void setPosition(int position);** setter for position.
    - **void setX(int width);** setter for x.
    - **void render(GraphicsContext gc);** draws drunkard image from RenderableHolder (drunkard1.gif or drunkard2.gif) at x,y position.
    - **boolean isSamePosition(TargetObject other);** checks whether the character has same position as target object or not.
      * If yes, return true. Else, return false.
    - **boolean exist();** return exist.
    - **void setExist(boolean exist);** setter for exist.
    - **boolean isPause();** return pause.
    - **void setPause(boolean pause);** setter for pause.