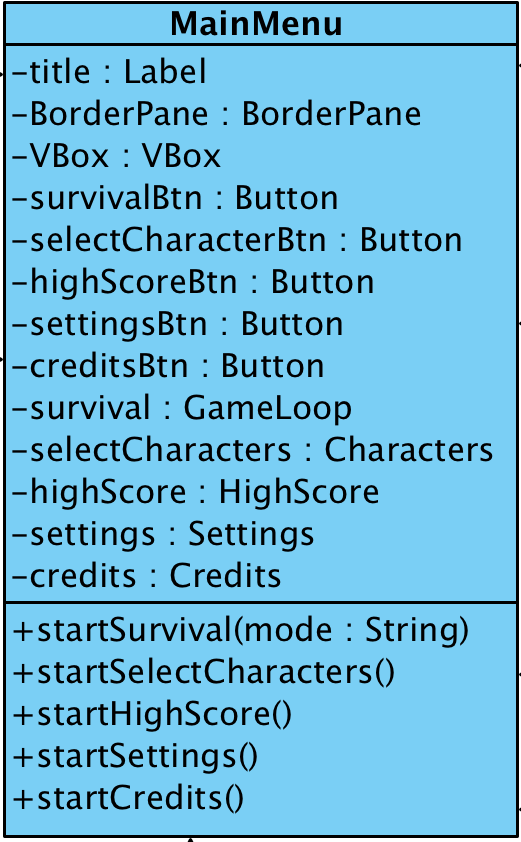
****

**Figure X. Low Level design of User Interface**

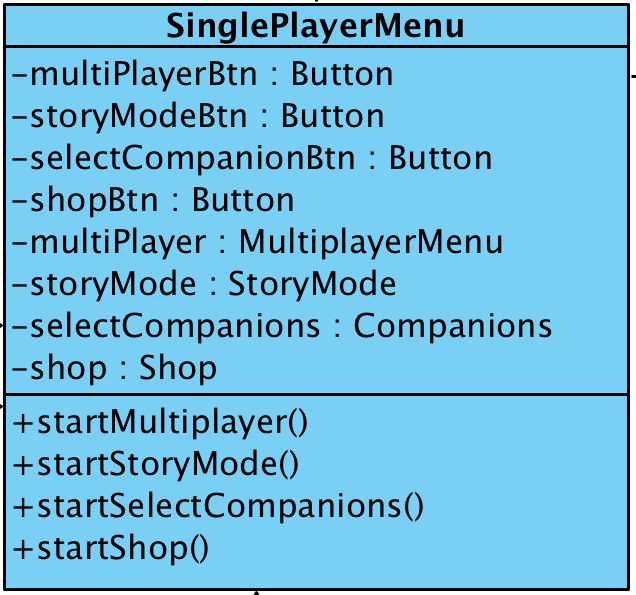
**4.4.1 MainMenu**

**Attributes**

* private Button survivalBtn - initializes the Survival Mode Game Loop
* private Button selectCharacterBtn - accesses the Select Character Screen
* private Button highScoreBtn - accesses the HighScore Screen
* private Button settingsBtn - accesses the Settings Screen
* private Button creditsBtn - accesses the Credits Screens

**Methods:**

* public void startSurvival(String mode) - method that initializes survival mode. String mode that is passed as a parameter decides whether the survival for one player (singleplayer) or two players (multiplayer) will be initialized.
* public void startSelectCharacters() - initializes the Characters Screen - to choose specific character.
* public void startHighScore() - initializes the HighScore Screen - to check the high scores
* public void startSettings() - initializes the Settings Screen - to make custom changes to the game - difficulty, audio, music.
* public void startCredits() - initializes the Credits Screen - to check credits.

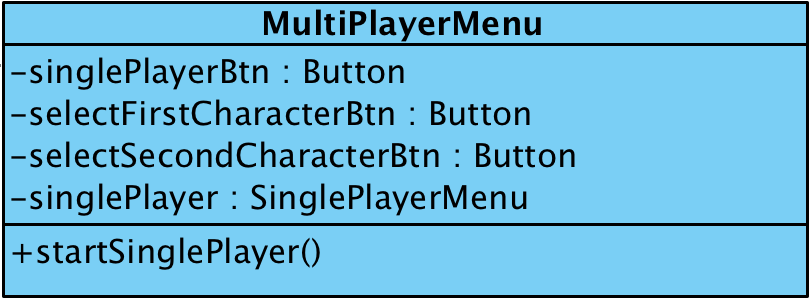
**4.4.2 SinglePlayerMenu**

**Attributes**

* private Button multiPlayerBtn - changes the Game Mode to the Multi-player
* private Button storyModeBtn - initializes the Story Mode Game Mode
* private Button selectCompanionBtn - accesses the Select Companion Screen
* private Button shopBtn - accesses the Shop Screen

**Methods:**

* public void startMultiplayer() - initializes the MultiPlayer Game Mode - to play with other human beings
* public void startStoryMode() - initializes the Story Game Mode - to play and learn the main story, plot of the game.
* public void startSelectCompanions() - initializes the Companions Screen - to choose specific companion.
* public void startShop() - initializes the Shop Screen - to buy/equip needed items.

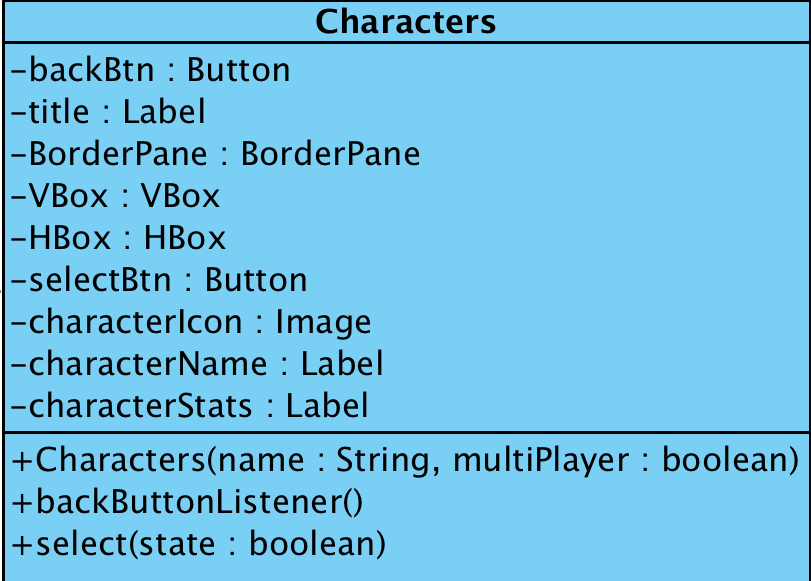
**4.4.3 MultiPlayerMenu**

**Attributes**

* private Button singlePlayerBtn - changes the Game Mode to the Single-player
* private Button firstCharacterBtn - accesses the Characters Screen
* private Button secondCharacterBtn - accesses the Characters Screen

**Methods:**

* public void startSinglePlayer() - initializes the SinglePlayer Game Mode - to play without other human beings and enjoy alone time in the game.

**4.4.4 Characters**

**Attributes**

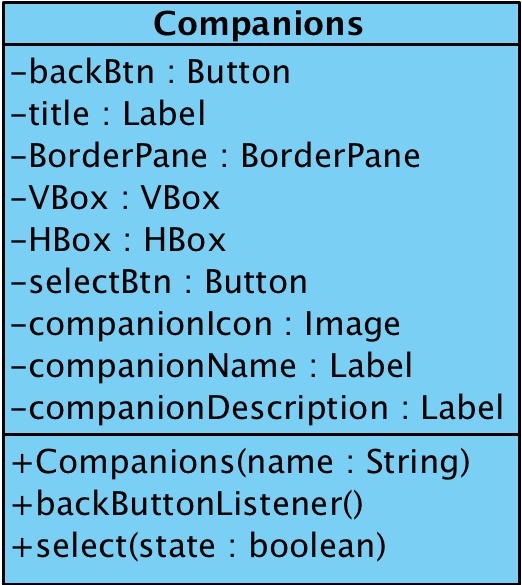
* private Button backBtn - back button, returns the player to the main menu
* private Button selectBtn - buttons that selects/unselects any specified character
* private Image characterIcon - the icon of the character (applies to all)
* private Label characterName - the title that shows the name of the character (applies to all)
* private Label characterStats - depicts the description of the character (applies to all)

**Methods:**

* public void backButtonListener() - returns to the MainMenu
* public void select(boolean state) - selects or unselects any available character. If the boolean value of state is equal to true, then the character is equipped, otherwise it is not.

**Constructor:**

* Character(String name, boolean MultiPlayer) - passes the name of the character as an indicator to the game loop (which character is selected). Boolean multiplayer; - if it’s value is equal to true then the selected character is assigned to second player instance, as the system knows its the multiplayer mode. Otherwise it is assigned to the main - first - player.

**4.4.5 Companions**

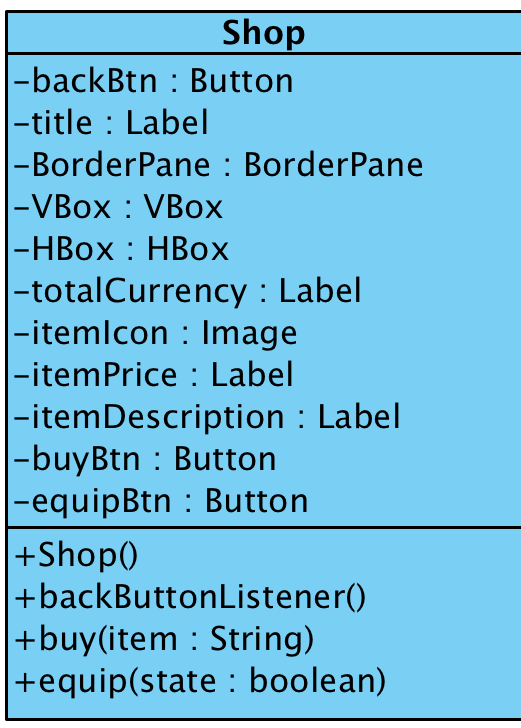
**Attributes**

* private Button backBtn - back button, returns the player to the main menu
* private Button selectBtn - buttons that selects/unselects any specified companion
* private Image companionIcon - icon of the companion (applies to all)
* private Label companionDescription - description text of the companion (applies to all)
* private Label companionName - the title that shows the name of the companion (applies to all)

**Methods:**

* public void backButtonListener() - returns to the MainMenu
* public void select(boolean state) - selects or unselects any available companion. If the boolean value of state is equal to true, then the companion is equipped, otherwise it is not.

**Constructor:**

* Companion(String name) - passes the name of the companion as an indicator to the game loop (which companion is selected).

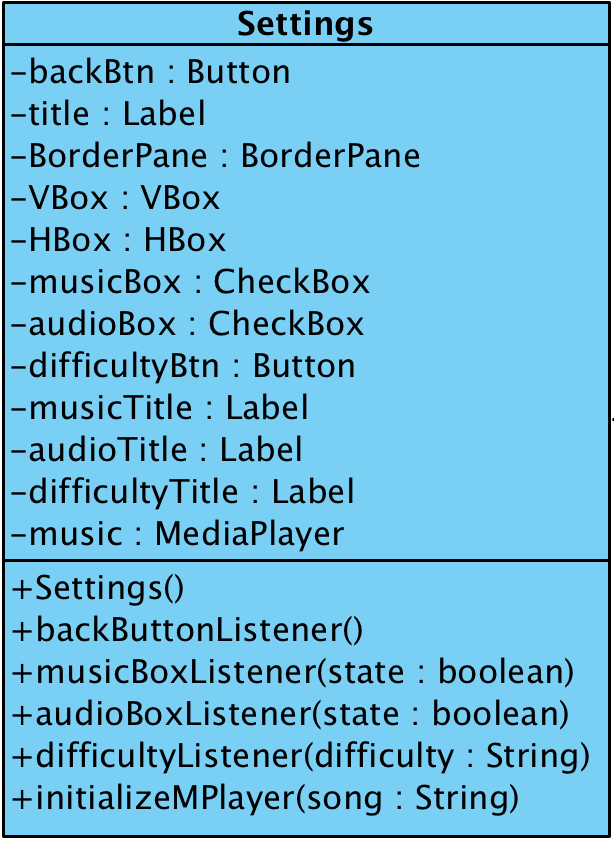
**4.4.6 Shop**

**Attributes**

* private Button backBtn - back button, returns the player to the main menu
* private Button buyBtn - buttons that buys any specified item
* private Button equipBtn - buttons that equips any specified item
* private Label totalCurrency - represents total currency of the user currently
* private Image itemIcon - icon of the item (applies to all)
* private Label itemDescription - represents description for the item (applies to all)
* private Label itemPrice - represents price for the item (applies to all)

**Methods:**

* public void backButtonListener() - returns to the MainMenu
* public void buy(String item) - buys any available item according to the parameter string - item name.
* public void equip(boolean state) - equips any bought item. If the boolean value of state is equal to true, then the item is equipped, otherwise it is not.

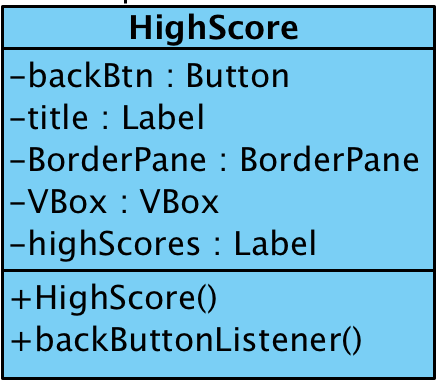
**4.4.7 Settings**

**Attributes**

* private Button backBtn - back button, returns the player to the main menu
* private CheckBox musicBox - checked music is on, unchecked music is muted
* private CheckBox audioBox - checked audio is on, unchecked audio is muted
* private Button difficultyBtn - 3 buttons together as a group: easy, normal and hard difficulty selector
* private Label musicTitle - title for the music check box
* private Label audioTitle - title for the audio check box
* private Label difficultyTitle - title for the difficulty selector button
* private MediaPlayer music - instantiates the Java FX’s media player to produce music

**Methods:**

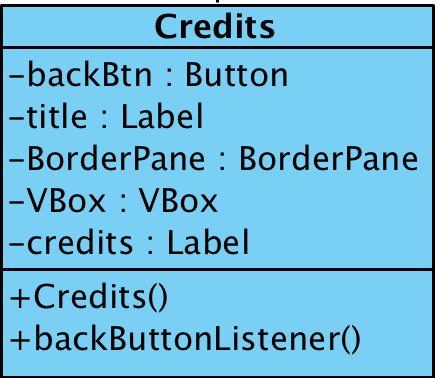
* public void backButtonListener() - returns to the MainMenu
* public void musicBoxListener(boolean state) - if the boolean value of the state is true then the music is turned on and playing, otherwise its off
* public void audioBoxListener(boolean state) - if the boolean value of the state is true then the audio is turned on in the game loop, otherwise its off
* public void difficultyListener(String difficulty) - the difficulty string that is passed as a parameter initialize the difficulty of the game in the game loop accordingly
* public void initializeMPLayer(String song) - initializes the media player - passing the song as a string to the input of url reader of media player.

**4.4.8 HighScore**

**Attributes**

* private Button backButton - back button, returns the player to the main menu
* private Label highScores - represents the high scores

**Methods:**

* public void backButtonListener() - returns to the MainMenu

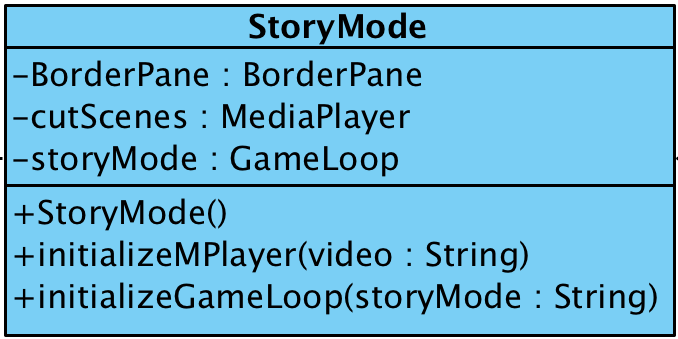
**4.4.9 Credits Screen**

**Attributes**

* private Button backButton - back button, returns the player to the main menu
* private Label credits - represents the credits rolling from bottom to top

**Methods:**

* public void backButtonListener() - returns to the MainMenu



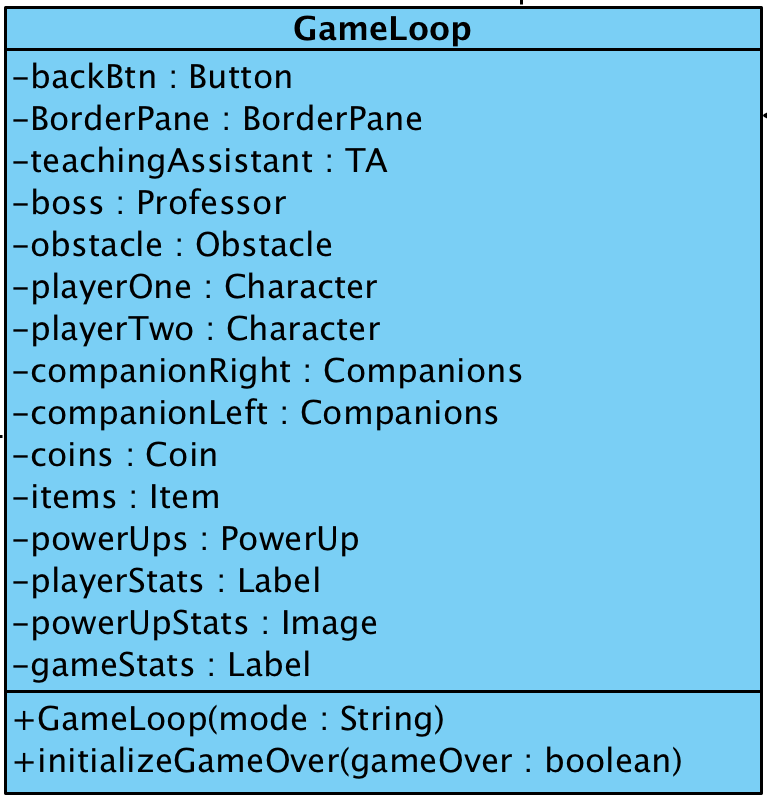
**4.4.10 StoryMode**

**Attributes**

* private MediaPlayer cutScenes - instantiates the Java FX’s media player to produce video as a cutscenes between semesters (waves)

**Methods:**

* public void initializeMPLayer(String video) - initializes the media player - passing the video as a string to the input of url reader of media player
* public void initializeGameLoop(String storyMode) - initializes the game loop - story mode - passing the story mode as a string will result in a game loop behaving according to the rules of story mode - finite waves; not survival mode.

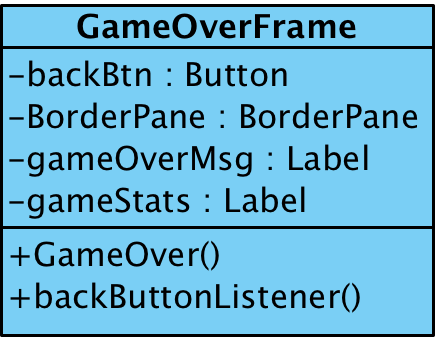
**4.4.11 GameLoop**

**Methods:**

* public void initializeGameOver(boolean gameOver) - if the boolean value of gameOver is true then the game is finished and the game over frame pops up. Otherwise the game continues.

**Constructor:**

* GameLoop(String mode) - initializes the mode of the game loop - story mode/survival mode for one player/ survival mode for two players

**4.4.12 GameOver Frame**

**Attributes**

* private Label gameOverMsg - the message that is showed when the game is over
* private Label gameStats - the stats of the player during the game - score, coins and semesters (waves) passed
* private Button backButton - back button, returns the player to the main menu

**Methods:**

* public void backButtonListener() - returns to the MainMenu