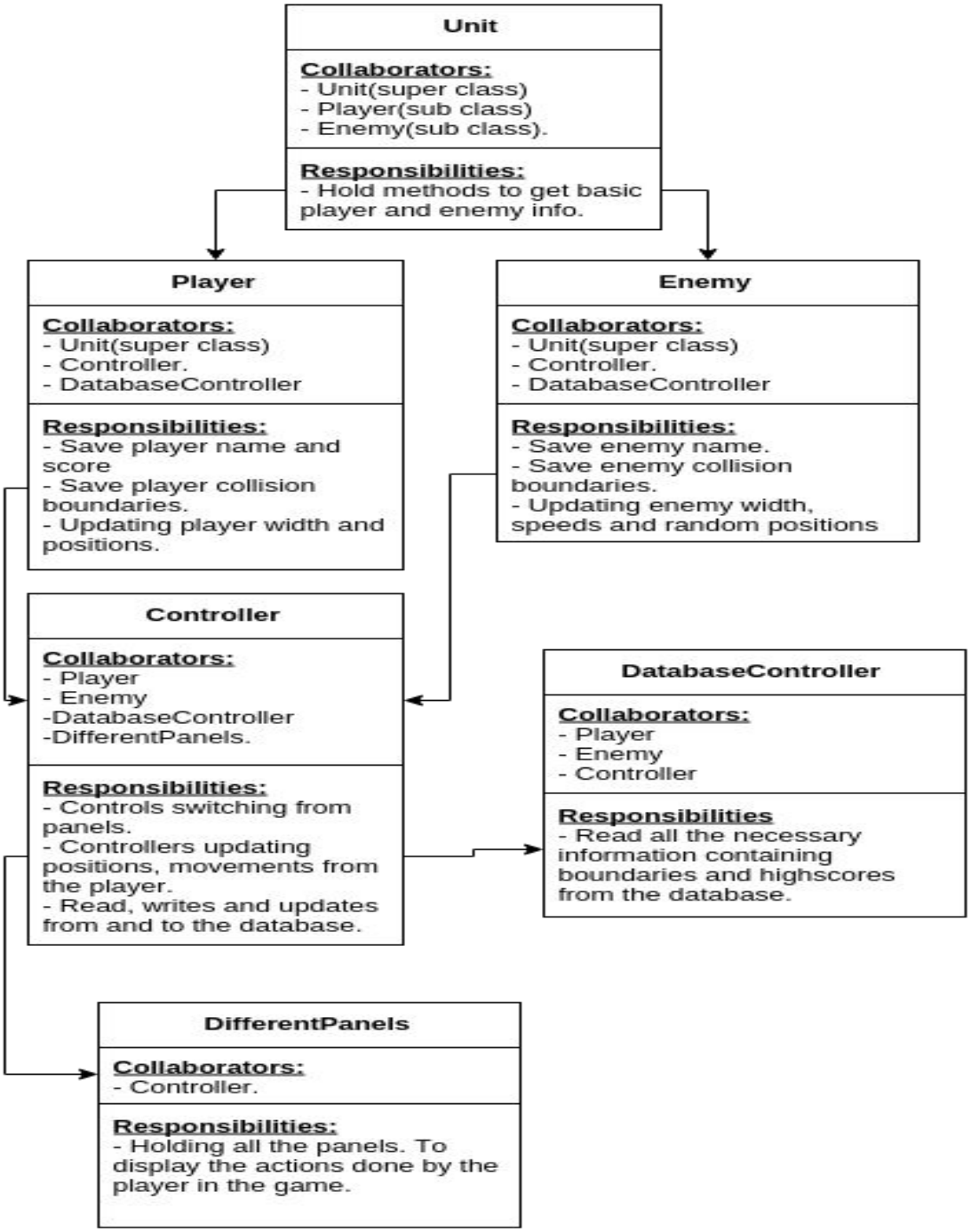
Group 36

Assignment 1

Exercise 1.1

Following Responsibility Driven Design we get the following classes:



- In our implementation we do not have a DifferentPanels class. We have multiple panels with various names. This because it was more neat to write different panels with their buttons and their particular responsibilities. The gamePanel in particular has to update every single movement of the player.

Exercises 1.2 + 1.3

The main difference between the main classes and less important classes is that the less important classes aren't necessary required to play the game, you can't play the game without them.

Main classes

These classes are required to play the game.

Default package:

Application.java – This class is called when the program starts and is creating a new Controller.

Controller package:

Collision.java – This class calculates the collisions and contains two methods: 'collide' and 'collDist'. The first method calculates if the player collides with the enemy. The second method calculates the distance between the enemy and the player.

Controller.java – This class basically connects the Model package and View package. In other words, this class connects the Graphical User Interface with the back-end algorithms.

Keylistener.java – This class handles the keyboard inputs. It has two methods: 'movePlayerKeyListener' and 'calcDir'. The first method only checks if there is a certain key being pressed and sets the Boolean variable of that key to true. The second method processes the Booleans and sets the String variable of the direction to the right direction.

- assigned

Model package:

BoundsPro.java – This class calculates, translates and scales the boundaries of the units, so we can use it in the Collision class.

Enemy.java - This class is a subclass of the Unit class and creates a random enemy, chosen from the list of sprites.

JSonRW.java – This class reads and writes JSon files. These files are database.json, for the highscores and Boundaries.json, for the information of the boundaries of the several different enemies.

Player.java – This class is a subclass of the Unit class and creates the player from the player sprite.

Unit.java – This class is the superclass of the Enemy and Player class and can't be instantiated. Next to that, this class contains the most properties the Player and Enemy have in common, such as the boundaries, direction, height, width, etc.

View package:

Frame.java – This class is the actual game window and triggers if you run the application.

GamePanel.java - This class is the GamePanel and triggers when you play the actual game.

StartPanel.java - This class displays the StartPanel when you start the application.

Less important classes
These classes aren't necessary required to play the game.

Controller package:
None
Model package:
None
View package:
CommonPanel.java – This class contains the methods for displaying the panels, which only contain text. These panels are being triggered by the about button and the instructions button.
HighScorePanel.java - This class shows the High Scores when you press the button in the start panel, which says high scores.
Sound.java - This class creates a separate thread, which displays the sound of the game, when you start the application.

Exercise 1.4 + 1.5

****

We placed the class diagram and the sequence diagram in the last two pages.

****

Exercise 2.1+2.2

Parametrized classes
We've used the following parameterized classes:
List<Entry<String, Integer>> highscore
List<Image> sprites
List<Enemy> enemies

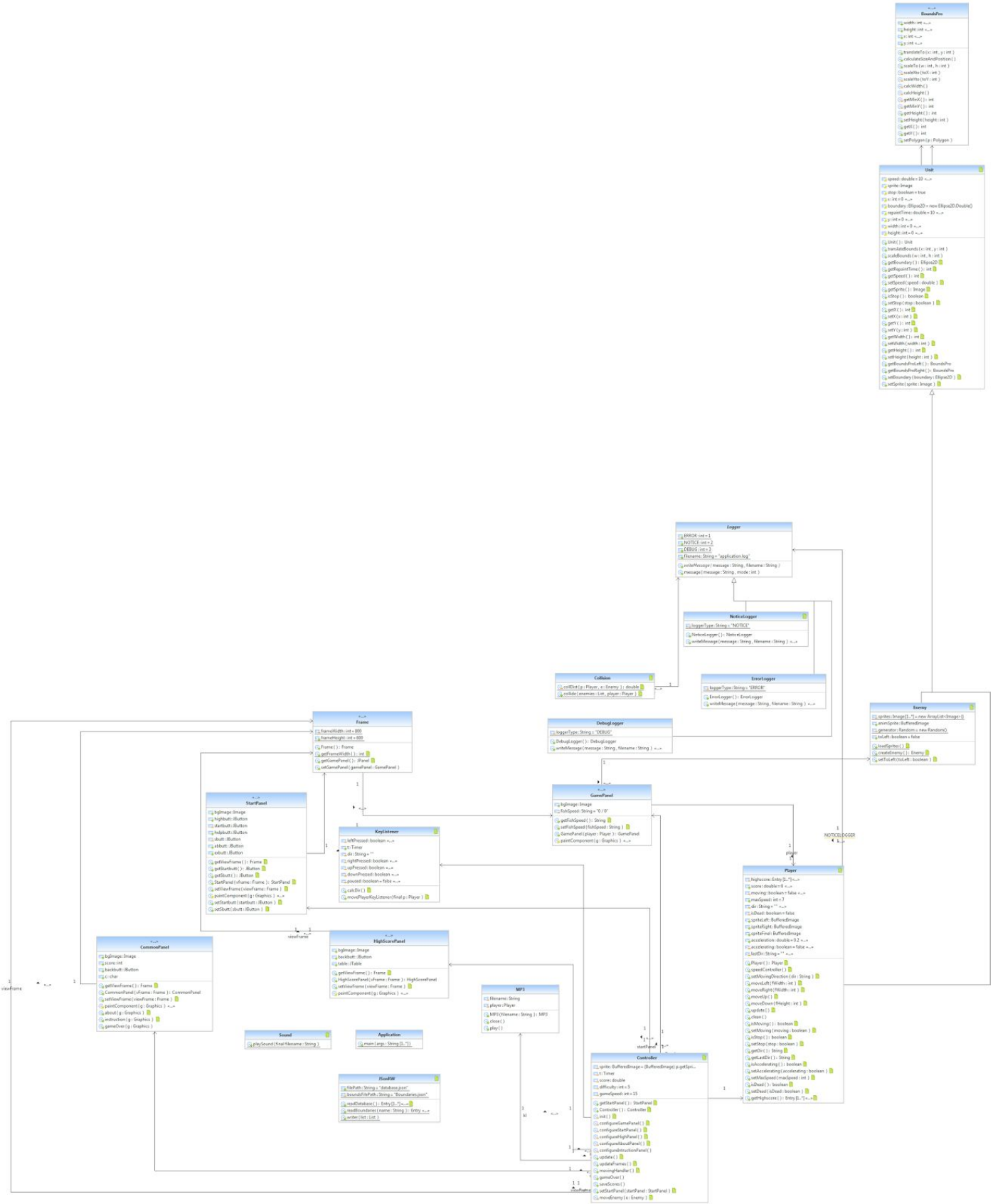
We have used these parametrized classes, because it was easier to sort them and change them without doing many actions.
Next to that parametrized classes offer compile time verification. This is not present in non-parametrized version of the class.
As last parameterized classes don't throw ClassCastException as Type verification was already done at compile time.

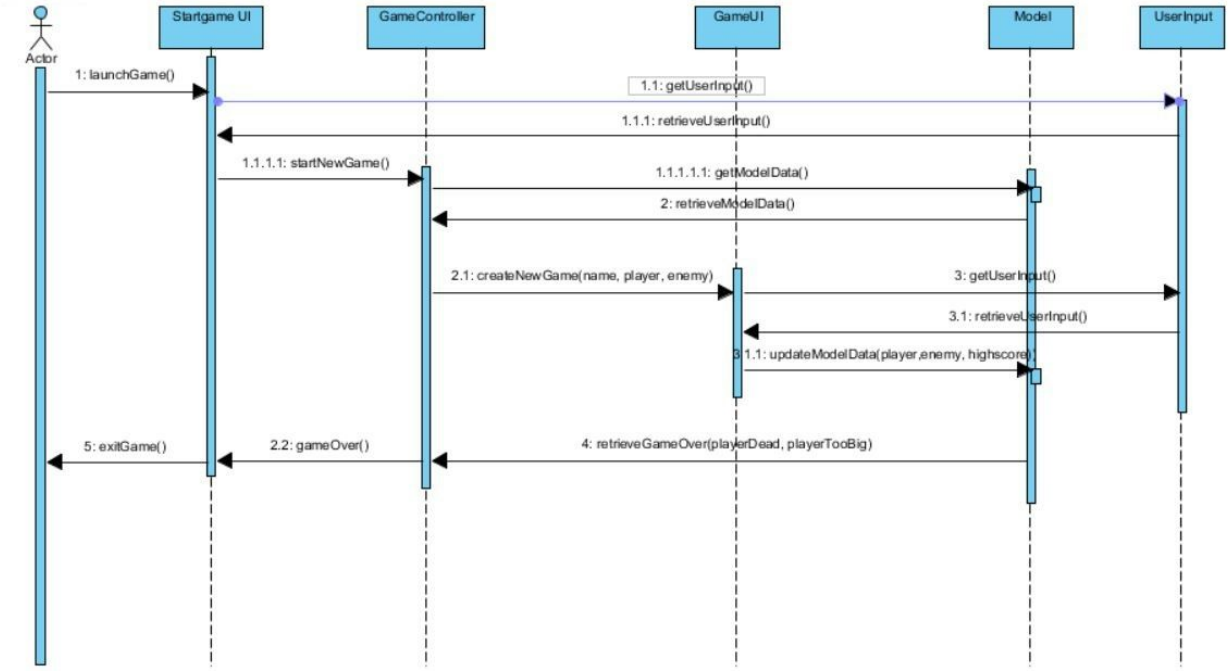
Exercise 3
Simple logging tool

- Error logging: Log the error if an error occurs.
- Notification logging:
    - Notify if the player size increases.
    - Notify if the player score increases.
    - Notify if the player eats a smaller fishy.
    - Notify if the player dies.
    - Notify if the player has a new highscore.
    - System access and application component access.
- Debug logging:
    - Log all the functions activated in the notifications.
    - Log if the player enters to the moveLeft function.
    - Log if the player enters to the moveRight function.
    - Log if the player enters to the moveUp function.
    - Log if the player enters to the moveDown function.
    - Log all the collisions.

Classes

- Logger : Super class
- ErrorLogging: prints error message if error occurs.
- NotificationLogging: print notification message if one of the aforementioned actions occurs.
- DebugLogger: print debug notification messages if one of the aforementioned actions occurs.


BoundsPro
Unit
Logger
NoticeLogger
ErrorLogger
DebugLogger
Collision
Enemy
Frame
StartPanel
GamePanel
KeyListener
CommonPanel
HighScorePanel
Player
Controller
MP3
Sound
Application
JSonRW


Actor
Startgame UI
GameController
GameUI
Model
UserInput
1: launchGame()
1.1: getUserInput()
1.1.1: retrieveUserInput()
1.1.1.1: startNewGame()
1.1.1.1.1: getModelData()
2: retrieveModelData()
2.1: createNewGame(name, player, enemy)
3: getUserInput()
3.1: retrieveUserInput()
3.1.1: updateModelData(player,enemy, highscore)
5: exitGame()
2.2: gameOver()
4: retrieveGameOver(playerDead, playerTooBig)